2014-01-29

Burstiness and Uncertainty Aware Service Level Planning for Enterprise Clouds

Youssef, Anas

UNIVERSITY OF CALGARY

Burstiness and Uncertainty Aware Service Level Planning for Enterprise Clouds

by

Anas Abdelaziz Youssef

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

January, 2014

Abstract

As enterprises begin to increasingly adopt the cloud paradigm, Cloud Service Providers (SPs) need tools to help them plan their infrastructure capacity and decide on Service Level Agreements (SLAs) with customers prior to deploying their customers' applications. Service Level Objectives (SLOs) are specified for customers' applications as part of customers' SLAs with cloud SPs. A cloud SP need Service Level Planning (SLP) tools that consider the workloads of the applications deployed on the cloud to determine the adequate capacity required to satisfy the applications' SLOs. Existing SLP approaches have not considered important challenges such as workload burstiness, workload uncertainty, and scalability to large number of applications.

This thesis presents an SLP framework that addresses the above challenges simultaneously. The framework implements a novel Resource Allocation Planning (RAP) method to identify a time varying allocation of resources to applications to satisfy their bursts. RAP is a heuristic optimization technique that in conjunction with a *trace-driven* performance prediction technique estimates the near minimal degree of service level violations that the cloud SP can incur with a given cloud resource capacity. RAP works in consort with a Monte Carlo simulation technique, which allows cloud SPs to systematically consider the impact of workload uncertainty in SLP. Finally, a new *burstiness-aware* workload clustering algorithm is proposed to increase the scalability of the SLP framework while preserving workload burstiness.

Detailed simulation results are presented to characterize the behaviour of the proposed SLP framework. The results show that the proposed RAP variants can identify optimal or near optimal resource allocation plans without exhaustively generating all possible plans. Secondly, the results show that RAP can permit cloud SPs to more accurately determine the capacity required for delivering specified SLOs compared to other competing techniques

especially for bursty workloads. Thirdly, the results demonstrate that the proposed Monte Carlo simulation technique enables cloud SPs to accurately estimate the impact of workload uncertainty in their SLP exercises without exhaustively traversing all combinations of application workload scenarios. Finally, the results show that the proposed workload clustering algorithm reduces the number of computations needed to support SLP exercises without significantly impacting accuracy.

Acknowledgements

The research work presented in this thesis would not have been possible without the support of many people. The author wishes to express her gratitude to her supervisor, Dr. Diwakar Krishnamurthy, who was constantly helpful and offered invaluable assistance, support and guidance. Dr. Krishnamurthy helped me a lot during the course of this work by continuous guidance, feedback and advice which really helped me to be an independent researcher and a critical thinker.

Deepest gratitude are also due to the members of the supervisory committee, Dr. Robert Simmonds and Dr. Guenther Ruhe, for their valuable comments and guidance to make this work successful. The author would also like to thank the members of the examination committee, Dr. Behrouz Far, Dr. Zongpeng Li and Dr. Raffaela Mirandola, for doing a strong effort in reviewing this work and pointing out valuable recommendations which greatly contributes to the successful delivery of this work.

The author would like to convey thanks to the Computer Performance Lab at University of Calgary, the Natural Sciences and Engineering Research Council of Canada (NSERC) and Hewlett-Packard for providing financial support and laboratory facilities. This support strongly helped the author to pursue her studies and acquire her degree.

Special thanks are also due to all the author's graduate colleagues, especially the group members in the Computer Performance Lab for sharing the literature, technical material and invaluable assistance.

Finally, the author wishes to express her love and gratitude to her beloved families, especially my mother, my sister and my wife; for their understanding, patience and endless love, through the duration of her studies.

To the memory of my father, Abdelaziz Youssef.

To my dear mother, Hekmat Seliem.

To my lovely wife, Soha Makady.

To my beloved son, Malik Youssef.

Table of Contents

List of Tables

List of Figures and Illustrations

List of Symbols, Abbreviations and Nomenclature

Symbol	Definition
Amazon EC2	Amazon Elastic Compute Cloud
Amazon S3	Amazon Simple Storage Service
Amazon RDS	Amazon Relational Database Service
API	Application Programming Interface
AWS	Amazon Web Services
FCFS	First Come First Serve
GPU	Graphics Processing Unit
HaaS	Hardware-as-a-Service
IDC	Index of Dispersion for Counts
IDI	Index of Dispersion for Intervals
IaaS	Infrastructure-as-a-Service
LQM	Layered Queuing Model
MAP	Markovian Arrival Process
MVA	Mean Value Analysis
PaaS	Platform-as-a-Service
PDF	Probability Distribution Function
PS	Processor Sharing
QNM	Queuing Network Model
QoS	Quality of Service
RAP	Resource Allocation Planning
SaaS	Software-as-a-Service
SCV	Squared Coefficient of Variation
SLA	Service Level Agreement

SLO	Service Level Objective
SLP	Service Level Planning
SP	Service Provider
VM	Virtual Machine
WAM	Weighted Average Method

Chapter 1

INTRODUCTION

1.1 Background and Motivation

Enterprises and organizations used to provide IT services to their customers by purchasing their dedicated hardware systems including among others servers, storage, switches, routers; and building their own infrastructure of networks, platforms, and software applications. This implies the need for administration and management of all of these resources by the enterprise itself. A cost problem arises here especially for small and medium business organizations where the resources are not used effectively to outweigh the cost of purchase and management. The workloads of these organizations change from time to time over the week and during the time of the day. For example, the resources can be heavily loaded during working hours while lightly loaded in evenings and weekends. So the cost of purchasing and managing dedicated resources while not utilizing them efficiently affects an enterprise's operating costs and revenues.

Consequently many enterprises today are consuming their IT services including hardware, platforms and software applications by using the notions of "pay-as-you-go" or "pay-for-use-only". Using these paradigms enterprises have to pay for using resources only when they need these resources. Enterprises do not have to purchase dedicated resources to host their applications or provide their services. Instead they can rent the required resources on demand from a Service Provider (SP) and release them when no longer required. In that way the SP takes the responsibility of administering and management of a system of interconnected resources to be shared among many enterprises. Consequently, costs can be distributed among hosted enterprise systems while efficiently utilizing resources. Such a system of interconnected resources implemented and managed by SPs to support the hosting

of different IT systems or services is referred to as a *cloud* [37].

Today there are many commercial cloud SPs such as Amazon Elastic Compute Cloud (Amazon EC2) [3], Google App Engine [12] and Microsoft Azure [16]. Cloud SPs host different types of systems or services on their managed resources. Depending on the type of service provided a cloud system can be classified into various categories such as Hardware-as-a-Service (HaaS), Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) [85].

HaaS refers to the service of renting physical hardware, e.g., physical servers, to cloud customers. However, due to the wide implementation of virtualization technologies [97] in cloud architectures, HaaS is not currently offered by many cloud SPs.

IaaS systems allow customer applications to execute within Virtual Machines (VMs). Virtualization technologies allow multiple VMs to execute on a single physical machine. This feature can be exploited to consolidate multiple VMs on a single physical machine to effectively utilize the machine's resources and save costs [40, 96]. Examples of IaaS cloud SPs include Amazon EC2 [3], RackSpace [20] and GoGrid [11].

A PaaS system provides software developers with a complete software development platform. This platform includes tools and environments to support software developers during the life-cycle of software development including analysis, design, development, testing and deployment of software applications. Examples of PaaS cloud SPs include Google App Engine [12] and Microsoft Azure [16].

Finally, SaaS systems allow cloud customers to share the use of general purpose software packages deployed on the cloud instead of running these packages locally. Examples of these packages include online office, web hosting, database and email applications. An example of a SaaS cloud SP is Google Docs [13].

Many of today's enterprise applications are interactive in nature, e.g., web-based applications that employ a multi-tier architecture composed of web, application and database tiers.

Typically, the owners of these applications need to satisfy good Quality of Service (QoS) to their end users. For example, the owners of these applications may specify an upper limit on the response time for a given transaction. As a result, the owners of enterprise applications often specify the required QoS in the form of Service Level Agreements (SLAs). Specifically, SLAs stipulate a number of Service Level Objectives (SLOs) to be satisfied by a cloud SP when delivering a service to an application owner and the penalties imposed on the cloud SP if the SLOs are violated [31].

Existing cloud offerings do not provide adequate support to explicitly guarantee SLOs for enterprise applications [31]. Typically, cloud services are provided to cloud customers with only basic SLO guarantees. For example Amazon EC2 [5] provides a guarantee on the upper limit of the monthly downtime of their VM instances. RackSpace [21] applies a similar approach to guarantee the availability time of their hardware, network and infrastructure. However, most cloud systems currently do not provide performance guarantees for their customers' applications [31]. For example, a typical cloud system does not provide any guarantee on how fast a transaction pertaining to an enterprise application will complete.

Such a performance-agnostic approach poses risks to cloud SPs. If insufficient resources are provisioned to customers' applications, the performance of these applications may degrade significantly. In particular, violation of performance-related SLOs such as transaction response times and throughputs can cause cloud SPs to incur significant SLO violation penalties. SLO violations affect a customer's satisfaction with the service and as result, the customer may migrate to other cloud offerings. On the contrary, if a cloud SP overprovisions resources to preempt performance problems with customers' applications, this might result in underutilization of the cloud SP's resources. As a result, this may increase the cloud SP's resource and energy costs. These factors can drive up the cost of the cloud offering thereby putting the cloud SP at a strategic disadvantage with other cloud SPs. Consequently, cloud SPs need systematic tools to provide them insights on resource management strategies for

optimally utilizing their resources while simultaneously satisfying customers' SLAs.

Addressing performance related issues in cloud-based services requires both long term and short term resource management approaches [58]. Both approaches are discussed in the ensuing paragraphs.

Long term tools help cloud SPs to plan their infrastructure capacity and decide on SLAs with a customer *prior* to deploying the customer's applications. For example, consider a customer that stipulates a certain mean response time threshold for her application's transactions. A cloud SP should consider the application's workload as well as the workload of other existing applications on the cloud to determine whether there is adequate capacity to satisfy this requirement. If the requirement cannot be satisfied, the cloud SP may suggest a less stringent performance requirement for the transactions of the customer's application. Alternatively, the cloud SP may determine the additional capacity needed to satisfy the request and use that information to present a revised cost estimate for delivering the service to the customer. Long term tools are referred to in the thesis as *Service Level Planning (SLP)* tools. The reader can refer to [24, 63, 83, 87] for examples of these tools.

SLP tools are typically used to plan for cloud resource management over coarse-grained time periods, e.g., hours or days. Typically, these tools need historical workload traces [58] that describe the workload experienced by the applications deployed on the cloud over the time period under consideration. Given such traces, SLP tools attempt to determine a good baseline strategy for how resources need to be allocated to applications and how such allocation should change over the duration of interest such that applications' SLOs are met while resource costs are minimized. SLP tools are typically used to plan for cloud resources prior to deploying customers' applications on the cloud, i.e., offline. This planning can also be implemented after application deployment on a weekly, bi-weekly or monthly basis to account for any deviations in customers' workloads.

SLP tools can provide an optimal resource allocation strategy for a set of applications

deployed together on the cloud such that all applications' SLOs are satisfied while minimizing resource costs. Solving this "global" SLO and resource allocation optimization problem is challenging for a large number of applications and for applications characterized by workloads which continuously change over time. Specifically, solving this optimization problem involves a number of issues.

Firstly, SLP exercises should consider the trade-off between two conflicting types of cost factors namely, cost of resources allocated and penalties due to applications' SLO violations. Minimizing resource costs can lead to degrading performance of the applications deployed on the cloud. As a result, applications are more likely to violate their SLOs resulting in SLO violation penalties. On the other hand, satisfying all applications SLOs might end up using a lot of resources which may drive up costs or which may not be feasible given the resource constraints faced by the cloud SP.

Secondly, SLP tools should handle heterogeneity among applications with respect to SLAs and SLO violation penalties. Each application typically has an SLO violation metric defined in terms of the application's performance requirements, e.g., an upper threshold for mean transaction response times [51]. An application SLO violation penalty is typically defined as a cost function which stipulates the amount of monetary units that a cloud SP must pay to a customer for a given degree of SLO violation [51]. A set of applications can have different SLO penalty cost functions. For example, one application can have an SLO violation which incurs a penalty of 100 monetary units to the cloud SP while another application can have an SLO violation which incurs a penalty of 10 monetary units. Solving the global SLO and resource allocation optimization problem for a set of applications while prioritizing applications with heterogeneous SLO violation penalties increases the complexity of SLP exercises.

To consider the aforementioned trade-offs, SLP tools typically optimize for a global SLO violation metric that combines SLO metrics of individual applications together. SLP exer-

cises optimize this global SLO violation metric while reducing resource costs. For example, an average or a weighted average SLO violation metric over all applications can be minimized. Another example is to minimize the number of violating applications if all of them have same SLO violation penalty cost. Determining which global SLO violation metric to use can affect the complexity of the SLP exercises. In summary, SLP tools should support a systematic and automated way to address such issues.

To solve the global SLO and resource allocation optimization problem for a set of applications, SLP tools usually rely on two underlying techniques namely, a search technique and a performance model. The search technique explores alternative application resource allocation solutions and selects the optimal solution that satisfies the optimization objective. As described previously, this optimization objective depends on the applications' SLO violations penalties and resource costs. Ideally, the search technique should be able to trade-off optimality in favour of less computational complexity when a large number of applications are involved. Specifically, with the large number of applications typically deployed on the cloud, e.g., hundreds or thousands, it might be prohibitive to traverse all possible solutions to obtain an exact optimal solution. In such a case obtaining a near optimal solution while exploring a subset of the possible solutions might be more practical and can reduce the computational complexity of the problem significantly.

SLP tools also rely on analytic performance models [41, 79, 88] to search for optimal resource allocation strategies for applications. Analytic models with fast solution techniques [79] allow a large number of resource allocation strategies to be quickly explored during SLP exercises. By leveraging information on an application's workload and the resources allocated to the application, the model predicts the application's performance. This in turn can be used to compute the application's SLO violation penalty during the search process.

Short term tools adjust cloud resources dynamically, i.e., online, to handle any sudden workload fluctuations during system operation. These tools are based on measurements

obtained from the system over fine-grained time scales in the order of seconds or minutes. Short term tools are typically categorized into approaches that rely on performance models and those that rely on control theory [91]. The reader can refer to [25,39,92,93] for examples of these tools.

In contrast to SLP tools, short term tools typically provide a solution to a "local" SLO and resource optimization problem that considers each application deployed on the cloud in isolation from other applications. Short term tools have to generate resource allocation strategies at real time while applications are running. Therefore, it may not be feasible to do global optimization over short time scales in clouds with large number of applications.

While short term tools are important, they have to be complemented by SLP tools that provide pre-deployment insights to cloud SPs [62]. Good pre-deployment resource allocation strategies obtained from SLP tools can increase the effectiveness of short term tools. Specifically, good pre-deployment strategies can help to minimize the number of resource migrations among applications after the deployment of these applications on the cloud. For example, consider a set of applications that are deployed on a cloud. If, prior to deployment, each application is allocated the required number and type of resources to match its workload over a given time period, then short term tools may trigger only minor movement of resources between these applications over the given time period. However, if the pre-deployment resource allocation strategy is not accurate enough to reflect the workloads of these applications, short term tools may need to do large scale of deallocation of resources from some applications and allocate these resources to other applications to match application workloads. This involves a lot of resource migrations between applications while the system is running. These on the fly resource migrations incur migration overheads which might affect the performance of the running applications [58].

The work in this thesis focuses on SLP tools. Realizing SLP tools for enterprise application clouds requires addressing three main challenges namely, workload burstiness, workload

uncertainty and scalability to a large number of applications. Details on each of these challenges will be described in the remaining part of this section.

The first challenge which faces the development of SLP tools for clouds is workload burstiness. Several studies have characterized the behaviour of enterprise application workloads [77, 94]. Many of these studies have indicated the presence of workload burstiness in real workloads [41–43, 81]. *Workload burstiness* refers to serial correlations in workload patterns such as correlation between successive arrivals of requests 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. This phenomenon can adversely affect the performance of applications running in the cloud and complicates resource management. Resource contention can increase during workload peaks thereby degrading performance significantly. Also, resources allocated to applications may experience underutilization during workload troughs leading to cost increase.

Since clouds allow resources to be allocated and deallocated on the fly, such elasticity can be exploited to address workload burstiness. However, leveraging this benefit of the cloud requires SLP tools that are able to accurately predict how many resources each application should get and how the number of allocated resources should change over time to match workload patterns. To achieve this objective, application workload characteristics such as burstiness should be exploited by SLP tools.

Resource allocation policies evaluated as part of SLP exercises that do not leverage complex workload behaviour such as burstiness can provide inaccurate estimates of the capacity and cost involved in delivering a target SLO. Figure 1.1 shows a simple example of two applications characterized by non-bursty and bursty workloads. Each workload is represented by the rate at which the requests arrive over time to the application system. The figure shows that resource allocation based on the average request arrival rate can provide adequate estimates of the number of cloud resources needed to satisfy the non-bursty workload. However,

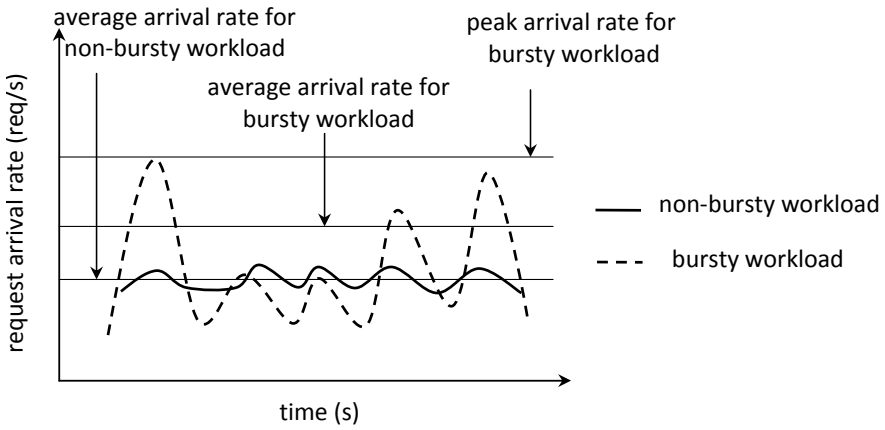


Figure 1.1: Workload burstiness

for the bursty workload cloud resource estimation analyses cannot be done solely based on average or peak workload requirements. Using average request arrival rate may lead to underprovisioning of resources and hence violation of SLOs. Using peak request arrival rate may lead to overprovisioning and underutilization of resources. Therefore, SLP tools should adapt the resource allocation strategies to the burstiness pattern of application workloads to obtain accurate resource allocation polices. Doing this automatically for clouds hosting a large number of applications is a big challenge. Furthermore, as mentioned previously in this section, enterprise applications typically employ the multi-tier architecture. Each tier needs to be allocated an appropriate number of resources of an appropriate flavour. For example, as typically done in practice, a web tier is allocated multiple instances where each instance has a low number of cores, e.g., 1,2 or 4 [4], while a database tier is allocated a single more powerful instance with more cores, e.g., 8,16 or 32. Supporting different types of resources at each tier imposes further complications to the design of SLP tools.

Furthermore, as described previously in this section, SLP tools often rely on performance models to predict service levels for a customer application under a given resource allocation strategy. Various performance models have been employed including traditional product-form Queuing Network Models (QNMs) [79] solved using the Mean Value Analysis (MVA)

technique and enhanced QNMs such as Layered Queuing Models (LQM) [88] which simultaneously consider the effect of contention for software and hardware resources. A limitation of such models is that their predictions can be inaccurate for systems characterized by workload burstiness. This limits the applicability of such models for enterprise systems whose workloads are known to exhibit burstiness [41–43, 81]. Unfortunately, most existing work in SLP uses these models. Therefore future SLP tools need to employ other performance modeling techniques that are capable of predicting application performance accurately under workload burstiness to obtain accurate resource allocation strategies. Recently, performance modeling techniques such as Markovian Arrival Process QNMs (MAP-QNM) [41] and the Weighted Average Method (WAM) [64] have been proposed to address this issue.

The second challenge which faces the development of SLP tools for clouds is the need to assess the risks that arise due to workload uncertainty. The workload experienced by many applications may deviate significantly from the workload projections estimated by application owners. Such applications may encounter many possible workload scenarios with each placing significantly different capacity requirements on the cloud. For example, Armbrust *et. al.* [29] described a scenario in Facebook where a sudden deviation from the normal workload occurred. This sudden peak workload required 3500 servers instead of 50 servers for the normal workload. Cloud SPs need systematic techniques to evaluate the impact of such uncertainties on penalties incurred due to SLO violations and resource allocation costs. Specifically, SLP tools need to allow cloud SPs to quantify the risks which might occur due to SLO violations because of workload uncertainty over time. For example, a cloud SP may need to know what penalties will occur due to SLO violations during workload peaks and how many resources are needed to eliminate these violations.

The final challenge that needs to be addressed by SLP tools is their scalability to large number of customer applications. SLP involves using performance models to search for optimal resource allocation strategies. This can take a lot of time with large number of

customer applications with each application characterized by multiple workload scenarios due to workload uncertainty. Furthermore, advanced performance models [41, 64] which accurately predict performance under workload burstiness take relatively longer execution time than traditional performance modeling techniques such as MVA [79]. This can impact the performance of the SLP toolset. Therefore, SLP tools should scale well to support a large number of applications and a large number of workload scenarios per application.

To solve this scalability problem, data reduction techniques such as clustering [50] can be used. Applications with similar workload characteristics can be grouped together into clusters. SLP exercises can then be done at the level of these clusters. For each of these clusters, a workload trace is needed to represent the workload traces of the applications that belong to this cluster. This introduces two main challenges. The first challenge is to select an appropriate clustering approach and appropriate clustering attributes to ensure that applications with similar burstiness characteristics are grouped together. The second challenge is to generate traces for clusters to accurately preserve the burstiness characteristics observed in the applications that belong to the same cluster. The clustering approach should be able to meet these two challenges otherwise the validity of the resource allocation strategies obtained by SLP tools will be in doubt.

There are no existing tools that address the above challenges simultaneously. Firstly, commercial tools use a resource utilization based approach [15,17,22,23]. These tools control resource utilization thresholds to "indirectly" achieve adequate application request response times. For example, for a multi-tier application, an approach which is usually done in practice is to limit the utilization of the bottleneck tier to a specific threshold , e.g., 30%-40%, so that the application per-request response times do not grow significantly. The *bottleneck* tier is the tier which has the highest impact on the application's mean request response time. An analysis of CPU utilization levels for more than 5,000 servers at Google over a six-month period [30, 96] shows that that the servers operate most of the time at utilization levels of

10% to 50%.

The advantage of such a utilization-based approach is that it does not need a performance model to predict application performance. This is a good advantage especially for bursty workloads because, as described previously, advanced performance models which predict accurate performance metrics under workload burstiness take relatively longer execution time than traditional performance modeling techniques such as MVA. However, as it will be shown later in this thesis, utilization-based approaches may not be adequate for application workloads characterized by burstiness.

Secondly, current SLP tools proposed in the literature [67,69,87] do not explicitly consider application workload burstiness and techniques to leverage information on burstiness to optimize resource allocations. Thirdly, the impact of workload uncertainty on the penalties incurred due to SLO violations and the resource costs needed to eliminate these violations have not been addressed by current SLP tools.

Finally, current SLP tools do not explicitly address the SLP scalability issue when large number of applications are deployed on the cloud especially when these applications are characterized by alternative workload scenarios to represent workload uncertainty as described previously.

1.2 Research Objectives

The thesis aims to achieve a number of objectives in three main areas namely, Resource Allocation Planning (RAP), workload uncertainty characterization and workload clustering. These objectives will be described in detail in the following three sections.

1.2.1 Resource Allocation Planning

Given a set of applications characterized by historical workload traces and SLOs for these applications, this thesis aims to solve the global SLO and resource allocation optimization

problem described in Section 1.1. The solution involves exploring RAP techniques which exploit given application workload characteristics such as burstiness to minimize a global SLO objective defined for a set of applications under given cloud resource constraints. Specifically, the explored RAP techniques should achieve a number of objectives as described in the ensuing paragraphs.

Firstly, the investigated RAP techniques should be computationally efficient while providing near optimal resource allocation solutions. In particular, one of the objectives of the investigated RAP techniques is to determine different heuristics to trade-off the SLP computation time for the optimality of the solution obtained for the resource allocation problem. Furthermore, each heuristic should be analyzed to determine under which conditions and scenarios this heuristic can be used to achieve a trade-off between SLP computation time and resource allocation optimality.

Secondly, the explored RAP techniques should be compared with other existing techniques for resource allocation which ignore workload burstiness during SLP. In particular, this objective aims to answer questions regarding the effect of ignoring workload burstiness in SLP on both resource allocation costs and the costs incurred due to SLO violations.

Thirdly, the explored RAP techniques should be compared with other state-of-the-art techniques for resource allocation. For example, the explored RAP techniques need to be compared with resource allocation techniques which typically predict application performance based on resource utilization thresholds [87]. In particular, this objective aims to understand how SLP using utilization-based approaches will impact the accuracy of resource allocation under workload burstiness.

The fourth objective is to study the fexibility of the explored RAP techniques to handle a variety of alternative application SLOs in the SLP process. In particular, this objective aims to understand which approaches can be used to define application SLOs and what effect each of these approaches imposes on resource allocation costs.

Finally, as described previously, the explored RAP techniques rely on workload traces to allocate a pool of resources to a set of applications. This will increase the computational time of these techniques when large number of applications are deployed on the cloud. As result, the computational time of the RAP techniques may need to be reduced. This objective aims to explore techniques that can be used to reduce this computational time.

1.2.2 Workload Uncertainty Characterization

The second objective of the thesis is characterizing workload uncertainty during SLP. In particular, this objective aims to develop a systematic approach to accommodate workload uncertainty in SLP exercises. This approach needs to evaluate how workload uncertainty will affect the penalties which a cloud SP might incur due to SLO violations. This approach should also be able to determine how many resources are needed to reduce the impact of the SLO violations that might occur due to workload uncertainty.

A key objective in this space is to explore techniques that can address the combinatorial explosion triggered by the need to characterize the impact of workload uncertainty. As mentioned previously in Section 1.1, every application in a cloud may need to be characterized by multiple workload scenarios to assess the impact of uncertainty. When a cloud hosts a large number of applications, this can lead to the need to analyze a prohibitively large number of application workload scenario combinations during SLP. Techniques are needed to achieve specifiable trade-off between the accuracy of predicting the impacts of workload uncertainty and the need to bound SLP analysis time.

1.2.3 Workload Clustering

The final objective of the thesis is to address the scalability of SLP tools to large number of applications as is typically in production clouds. Specifically, this objective aims to allow SLP tools to scale well to a large number of applications. Clustering techniques [50] can be used to identify a small set of application clusters where each cluster comprises applications

with similar workload characteristics. SLP can then be done at the levels of these clusters. If there is significant similarity between application workloads, then the number of clusters will be much less than the number of applications. As a result, SLP exercises will consume less time than if applications are not clustered.

Firstly, the research in this thesis focuses on identifying attributes and techniques that are effective for clustering bursty workloads. Specifically, it will consider various workload attributes for conducting the clustering and compare the effectiveness of these attributes using different clustering approaches with multiple levels of clustering.

Secondly, the research will focus on novel trace aggregation algorithms. Since SLP exercises require workload traces, techniques are needed to aggregate workload traces of applications that belong to a cluster into an equivalent trace for this entire cluster. The challenge is to aggregate traces such that the burstiness characteristics present in the individual application traces are preserved in the equivalent aggregate traces. If this challenge is not addressed, then workload clustering will introduce significant errors in the SLP process.

1.3 Research Contributions

The thesis makes a number of contributions in the three main areas described in Section 1.2. Section 1.3.1 describes the proposed RAP method. Section 1.3.2 describes the proposed workload uncertainty characterization technique. Section 1.3.3 describes the proposed workload clustering technique. Section 1.3.4 describes a trace-based SLP framework which integrates the above three techniques together. Finally, a case study is presented in Section 1.3.5 to demonstrate the end-to-end operation of the proposed SLP framework.

1.3.1 Resource Allocation Planning Method

The thesis proposes a novel RAP method to solve the global SLO and resource allocation optimization problem which was described previously in Section 1.1. To facilitate planning,

a cloud SP elicits applications' workload characteristics in the form of a collection of traces defined 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 from an online retail store. 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 CPU 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. Studies have shown that workload behaviour of many enterprise applications have predictable patterns. Gmach *et al.* [56] collected 6 months of resource usage data of 139 workloads in an enterprise data center. They observed that such application workloads typically follow well-defined, periodic hour-of-the-day, day-of-the-week, and day-of-the-month patterns. This provides a strong motivation for a trace-driven RAP method such as the one proposed in this work.

Traces for new applications can be obtained in several ways. For example, customers can provide traces from environments external to the cloud where the application was previously deployed. Alternatively, traces can be derived from application test environments or synthetically generated based on customer estimates of mean or peak workload behaviour.

Figure 1.2 shows an overview of the RAP method. RAP takes as input a collection of workload scenarios for a set of applications and SLOs for these applications. In this work an application SLO is defined based on mean request response time target. However, this work can also accommodate other types of SLOs. RAP also takes as input resource constraints faced by a cloud SP. For example, the cloud SP may only have limited number of resources

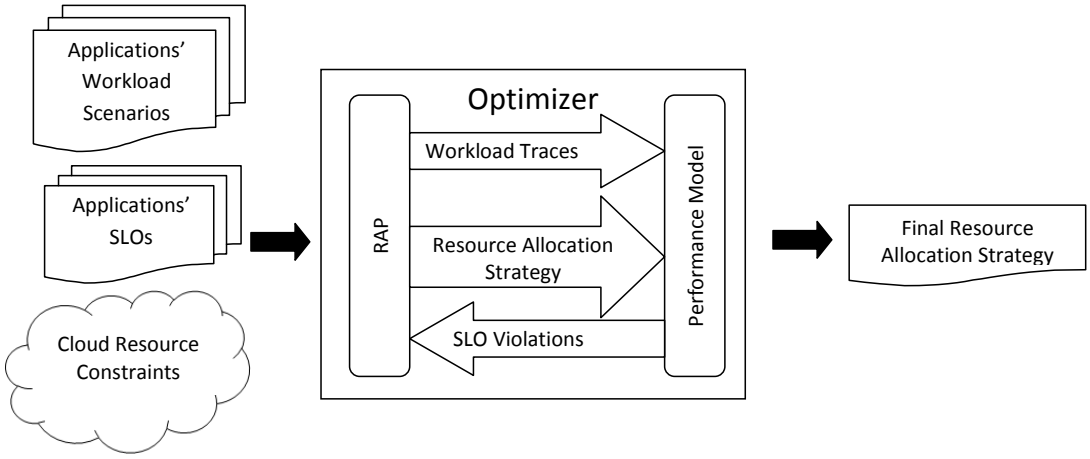


Figure 1.2: RAP overview

for allocation to the applications deployed on the cloud.

Given the aforementioned inputs, RAP searches for a time varying resource allocation strategy that minimizes a global SLO violation metric defined for the input set of applications. To do this search, RAP relies on a performance model to predict SLO violations given application workload traces and a resource allocation strategy. The global SLO violation metric is a combination of the individual SLO violation metrics for all applications. The SLO violation metric considered in this work is the SLO violation percentage. The *SLO violation percentage* for an application is the percentage of deviation of the mean request response time predicted by the performance model for this application from the target mean request response time that defines the application's SLO objective. The global SLO violation metric considered in the thesis is the mean SLO violation percentage for the input set of applications. However, the proposed RAP method can be easily adapted to accommodate other global SLO violation metrics that need to be varied or parameterized by cloud SPs.

Customers specify SLOs for their applications over a planning horizon. The *planning horizon* represents the time period over which a cloud SP is obliged to satisfy applications' SLOs as per its SLA with the customer. The planning horizon can be in the order of minutes, hours, or even days. The proposed RAP method allows cloud SPs 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 time intervals and deallocated 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 minutes within a planning horizon of 24 hours.

Figure 1.3 illustrates the relationship between the planning horizon and the resource allocation interval over time for two applications characterized by burstiness. In general, the planning horizon can span one or more resource allocation intervals. This affects the way an SLO is defined for a customer's application over the planning horizon. Specifically, if the planning horizon spans only one resource allocation interval, then a customer's application's SLO requirement defined over this planning horizon should also be satisfied over this resource allocation interval. Alternatively, if the planning horizon spans more than one resource allocation interval, then there are two approaches to define a customer's application's SLO requirement over this planning horizon. The first approach defines a customer's application's SLO requirement as a single SLO requirement over the entire planning horizon. The second approach defines a customer's application's SLO requirement as a set of multiple SLO requirements where each SLO requirement should be satisfied over each resource allocation interval within the planning horizon.

The main power of the RAP method comes from having a resource allocation interval smaller than the planning horizon. This is a key difference from other techniques proposed in the literature for SLP. The work in this thesis shows that using resource allocation intervals at a finer time scale than the planning horizon can allow cloud SPs to realize resource savings, especially when application workloads are bursty. For example, Figure 1.3 shows two bursty applications with non overlapping peaks, i.e., have peaks in two different resource allocation intervals. If the two applications compete for the same resource to satisfy their peaks, they

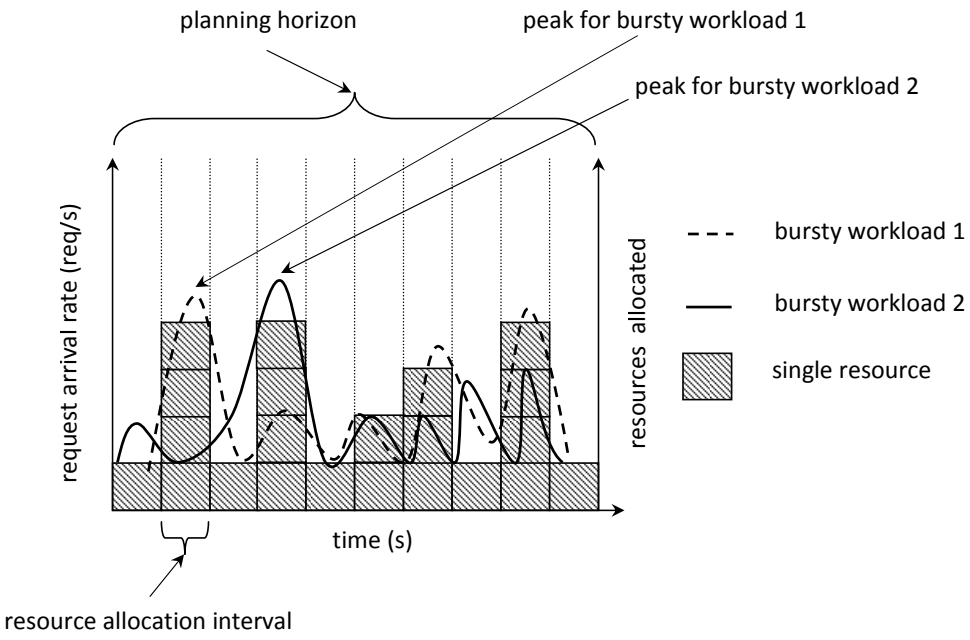


Figure 1.3: Service level planning under workload burstiness

can both be allocated this resource over the planning horizon. This is not possible if the planning horizon is of the same granularity as the resource allocation interval. In essence, having resource allocation intervals at finer time scale than the planning horizon allows a cloud SP to identify time periods with most burstiness where more resources can be allocated while allocating fewer resources to time periods of lower burstiness. This in turn can reduce resource costs.

The RAP method involves searching for an optimal resource allocation plan that minimizes the global SLO violation metric for the input set of applications. A *resource allocation plan* defines 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. Figure 1.3 shows an example of a resource allocation plan for the two bursty applications described previously. As shown in the figure, the resource allocation plan is composed of resources, e.g., web or database servers, allocated over a number of resource allocation intervals.

As described previously in this section, the RAP method relies on a performance model to predict applications' SLO violations. The RAP method is intended to exploit application workload patterns for SLP. Therefore, RAP requires a performance model that takes as input a workload trace for a given application and a possible resource allocation plan and predicts the SLO violation for this application. RAP exploits a trace-driven performance prediction technique called Weighted Average Method (WAM) [64] that allows it to reflect the impact of workload burstiness into SLP exercises. The WAM technique was shown to be more accurate than other traditional performance modeling techniques [79, 88] when it comes to predicting SLO violations under burstiness.

To do the search for an optimal resource allocation plan, the most straight forward way is to enumerate all possible resource allocation plans for a given set of applications under given cloud resource constraints. Then an evaluation of the global SLO violation metric defined for the set of applications is made. In this way, an exact optimal resource allocation plan that minimizes this global SLO violation metric can be obtained. However, in practice this process is prohibitive. This is due to the large number of applications typically considered for SLP, e.g., hundreds or thousands. Also, resources are allocated over a number of resource allocation intervals and as this number increases, the number of possible resource allocation plans increases significantly. Furthermore, an application can have multiple tiers with each tier allocated a certain resource type, e.g., web or database server. Even for a single tier, different flavours of resources can be allocated, e.g., single-core or multi-core web server. Due to the above reasons the enumeration of all possible resource allocation plans is prohibitive in practice .

To solve this problem, three different variants of the RAP method are developed namely, *RAP-Dynamic Programming (RAP-DP)*, *RAP-Heuristic-AllApps (RAP-AllApps)* and *RAP-Heuristic-OneApp (RAP-OneApp)*. The three techniques are described briefly in the ensuing paragraphs. More details will be described in Chapter 3. Table 1.1 shows a summary of the

Table 1.1: Summary of RAP variants

At Each Decision Stage	RAP-DP	RAP-AllApps	RAP-OneApp
applications evaluated	all	all	application with the highest SLO violation metric
resource allocation intervals evaluated	all	interval with the highest mean request response time	interval with the highest mean request response time
number of performance model invocations	product of # of applications and # of resource allocation intervals	# of applications	1

different aspects of the three RAP variants.

RAP-DP uses dynamic programming [49] to provide an optimal solution. As a dynamic programming-based technique, RAP-DP divides the search for an optimal resource allocation plan into decision stages. At each decision stage, a number of possible resource allocation plans are explored given the cloud resources available for allocation to all applications at this decision stage. Each decision stage considers the problem of allocating a *single* resource to the bottleneck tier of exactly one of the applications considered for SLP over one of the resource allocation intervals. This allocation is done to minimize the global SLO violation metric. Thus, the cloud resources available for allocation over this resource allocation interval decreases by one from this decision stage to the next decision stage. At the end of a given decision stage, only one resource allocation plan, i.e., the optimal plan, is selected from the explored resource allocation plans in this decision stage. This optimal resource allocation plan achieves the minimum global SLO violation metric given the cloud resources available for allocation in this decision stage. The optimal resource allocation plan selected in this decision stage is then used as a basis for further exploration in the next decision stage. This process is repeated over a number of decision stages until a final optimal resource allocation plan is obtained to minimize the global SLO violation metric under the resource constraints of the cloud. As shown in Table 1.1, the number of performance model invocations required by RAP-DP at each decision stage is equal to the product of the number of applications and

the number of resource allocation intervals.

Although RAP-DP reduces the number of explored resource allocation plans significantly when compared to the exhaustive approach, it still has to explore a large number of possible resource allocation plans in each decision stage. This involves exploring all possible resource allocation plans for all applications over all resource allocation intervals. For each resource allocation plan explored the performance model employed by RAP has to be invoked to predict the global SLO violation metric for this resource allocation plan. As described previously in Section 1.1 the WAM [64] performance modeling technique employed by RAP which predicts accurate performance metrics under workload burstiness takes relatively longer execution time than traditional performance modeling techniques such as MVA [79]. As a result, RAP-DP may not scale well for large number of applications typically deployed on the cloud and over a large number of resource allocation intervals. To solve this problem, two other variants of RAP are developed, i.e., RAP-AllApps and RAP-OneApp. These two variants are heuristic search techniques that can give nearly optimal solutions but are more computationally efficient than RAP-DP.

Unlike RAP-DP which runs WAM for all applications in all resource allocations intervals in each decision stage, RAP-AllApps runs WAM for all applications only in the resource allocation intervals with the *highest mean request response time*. This resource allocation interval represents the most bursty and congested resource allocation interval for each 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 global SLO violation metric with respect to other intervals. As shown in Table 1.1, at each decision stage, the number of performance model invocations by RAP-Heuristic-AllApps is equal to the number of applications. In contrast to RAP-DP, RAP-AllApps has no dependence on the number of resource allocation intervals.

The third RAP variant developed in the thesis is RAP-OneApp. RAP-OneApp makes the

simplifying assumption that only targeting the application with the highest SLO violation metric likely yields the least global SLO violation metric over all applications. Accordingly, at each decision stage RAP-OneApp ranks all applications in a descending order in terms of their individual SLO violation metrics and selects the topmost application. The allocation of an incremental resource is explored only for this application rather than for all applications as in RAP-DP and RAP-AllApps. Thus, RAP-OneApp invokes the performance model only once at each decision stage as shown in Table 1.1.

The work in this thesis presents detailed simulation results to characterize the behaviour of the explored RAP techniques. Firstly, the results confirmed previous findings in [64] that the WAM performance modeling technique which is used in this work can more accurately predict SLO violations under workload burstiness than other traditional performance modeling techniques such as MVA. Specifically, a comparison is made between WAM and MVA to show the accuracy of WAM over MVA for SLP. WAM shows at most 10% of average prediction error for the performance of applications characterized by different degrees of burstiness. In contrast, for some applications characterized by highly bursty workload scenarios, MVA gives prediction errors as high as 90%.

Secondly, a detailed analysis of the computational complexity of the proposed RAP variants mentioned previously is conducted. The optimality of the three variants is also studied. The results show that each proposed RAP variant can provide a trade-off between SLP computational complexity and optimality of resource allocation under certain workload scenarios as follows:

- The results show that RAP-DP can identify an optimal resource allocation plan that minimizes SLO violations under given cloud resource constraints without exhaustively generating all possible resource allocation plans.

- The results also show that RAP-AllApps which is a more computationally efficient version of RAP-DP is able to identify optimal resource allocation

plans if the bottleneck tier for all applications remains the same in all resource allocation intervals. If this condition is violated, RAP-AllApps can still obtain close to optimal solutions.

- The results show that under both bursty and non-bursty workload scenarios, RAP-OneApp is able to identify near optimal resource allocation plans while reducing SLP computational complexity significantly when compared to the optimal RAP-DP approach and RAP-AllApps. Specifically, in an experiment conducted for a set of applications characterized by different degrees of burstiness in session arrivals, RAP-OneApp reduces the number of resource allocation plans explored at each decision stage by 94% and 75% relative to RAP-DP and RAP-AllApps, respectively. Consequently, RAP-OneApp is better suited for analyses involving a large number of applications without significantly affecting the optimality of the solutions obtained.

Thirdly, the results show that RAP can permit cloud SPs to more accurately determine the resources required for delivering specified SLOs compared to other competing techniques especially for scenarios where applications are subjected to bursty workloads. In particular, RAP is compared with another approach which considers a resource allocation interval that is the same as the planning horizon. This approach is referred to as the *whole* approach. The results show that for bursty workload scenarios, selecting a resource allocation interval that is smaller in size than the planning horizon reduces the resource allocation costs required to deliver a given application SLO. Furthermore, in resource constrained scenarios, the resource allocation plan identified by the whole approach results in much higher SLO violations than RAP for highly bursty workload scenarios. Specifically, using the resource allocation plan suggested by the whole approach achieves a mean SLO violation percentage of 155% for a set of applications characterized by highly bursty scenarios while RAP achieves a mean SLO violation percentage of only 71% for the same set of applications.

Fourthly, the results establish the superiority of RAP to the prevalent practice of considering SLO targets based on resource utilization thresholds over the planning horizon. Specifically, unlike utilization-based approaches the results show that RAP can accurately adapt resource allocation plans to observed burstiness characteristics of application workloads. An experiment is conducted for a set of applications characterized by different degree of burstiness in session arrivals. In each run of this experiment the utilization of the bottleneck tier is limited to a specific target, e.g., 30%, 50% and 60% over the planning horizon. The results show that the number of resources estimated for each utilization target by the utilization-based approach remains the same regardless of the degree of burstiness in session arrivals. On the contrary, the results show that, for each utilization target, the application response time violations increase significantly for highly bursty scenarios. This indicates that merely relying on utilization-based targets may not be an appropriate method for guaranteeing SLOs based on response times for applications characterized by burstiness.

Fifthly, the results show that the proposed RAP method is flexible in accommodating different approaches for defining application SLOs. As described previously in this section, if the planning horizon spans more than one resource allocation interval, then there are two approaches to define an application's SLO over the planning horizon. The first approach defines an application's SLO as a single SLO over the entire planning horizon. The second approach defines an application's SLO as a set of multiple SLOs where each SLO should be satisfied over each resource allocation interval within the planning horizon. A comparison is conducted between these two approaches. The results show that under both bursty and non bursty workload scenarios, cloud customers can obtain cost savings using the first approach when compared with the second approach. The reason for this is that the first approach only considers the *most heavily loaded* resource allocation intervals for additional resource allocation. On the contrary, the second approach allocates additional resources to *all heavily loaded* resource allocation intervals to eliminate all SLO violations which occur in

all resource allocation intervals. These types of analyses can help cloud SPs negotiate with their customers the costs related to different ways of defining application SLOs. Based on this cost estimate cloud customers may revise the granularity of the time period over which they negotiate their SLOs with the cloud SP.

Finally, an approach to improve the performance of RAP-DP and RAPAllApps by exploiting parallelism is shown. Specifically, multi-threaded versions of the two RAP variants are implemented to reduce their computation times for large number of applications. A machine with 12 processor cores is used to compare the computation times of the single-threaded and the multi-threaded versions of the two RAP variants. The results show a performance improvement of the multi-threaded versions over the corresponding single-threaded versions by a factor of 9.

1.3.2 Workload Uncertainty Module

The thesis proposes a workload uncertainty module to accommodate workload uncertainty in the SLP process. The uncertainty module employs a Monte Carlo simulation technique [82] which allows cloud SPs to consider a set of probable workload scenarios for each application. For example, one workload scenario may represent a typical normal activity while another workload scenario may represent periods of heightened workload activity. Each workload scenario is assigned a probability that embodies a cloud SP's and a customer's estimate of how likely that workload scenario is to occur.

Figure 1.4 shows an overview of the uncertainty module. The uncertainty module implements a Monte Carlo simulation technique which accepts workload traces representing all possible workload scenarios for each application that the cloud SP plans to deploy on the cloud. Each application workload scenario has an associated probability of occurrence. The proposed Monte Carlo simulation technique generates a set of combinations of workload scenarios. Each combination contains exactly one probable workload scenario for each application under consideration. Each combination has an associated probability of occur-

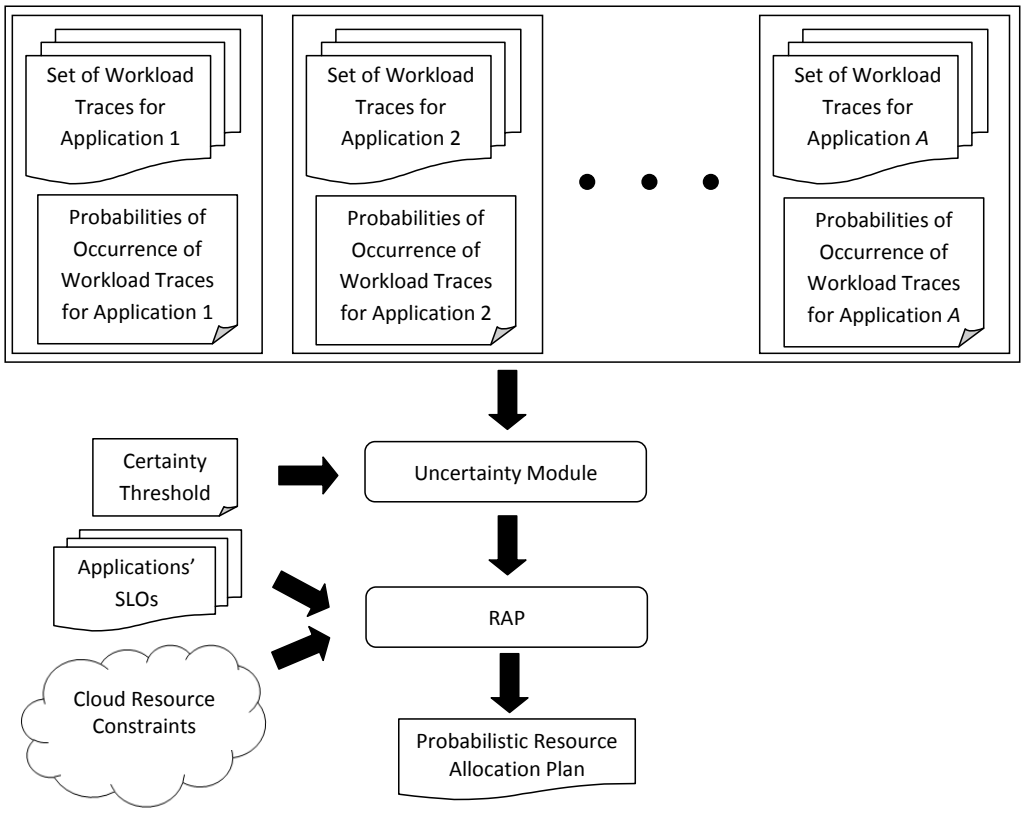


Figure 1.4: Overview of the uncertainty module

rence which is computed based on the probabilities of occurrence of the individual workload scenarios forming this combination. These combinations of workload scenarios together with applications' SLOs and constraints on cloud resources are then input to the RAP module.

The potentially large number of applications involved in SLP, each with multiple workload scenarios, can lead to an explosion in the number of alternative combinations of workload scenarios that need to be considered during the SLP process. Specifically, if the number of applications is denoted by A and the number of alternative workload scenarios for each application a, is denoted by L_a, then the number of combinations of workload scenarios which need to be analyzed will be $(L_a)^A$. For 100 applications and 2 alternative workload scenarios per application the number of combinations of workload scenarios will be 2^{100}. For large numbers of applications and workload scenarios per application, this becomes practically

prohibitive.

To address the issue of combinatorial explosion, a certainty threshold is proposed. The *certainty threshold* is a value that stipulates a lower bound on the summation of the probabilities of occurrence of the workload scenario combinations that need to be analyzed in SLP exercises. Using this value a cloud SP can reduce the number of combinations of workload scenarios that need to be analyzed in the SLP process. The certainty threshold takes a value from 0 to 1 to determine if all or a subset of the workload scenario combinations will be analyzed during SLP. If a subset is selected, the proposed Monte Carlo simulation technique ensures that combinations which are more likely to occur are analyzed. Specifically, the combinations which have a cumulative probability of occurrence that is greater than or equal to the certainty threshold are selected. For example, if the certainty threshold is 0.9, then the subset of combinations selected for SLP analysis has a cumulative probability of occurrence of at least 0.9. A certainty threshold value of 1 means all possible combinations are analyzed in the SLP exercises.

The certainty threshold is used to trade-off the accuracy in predicting the impact of uncertainty for SLP computation time to speed up the SLP analysis. For example, to accurately predict the impact of workload uncertainty on the risks of SLO violations that might occur, the certainty threshold should be set to 1 which means that all possible combinations should be analyzed in the SLP process. However, if this is not feasible, the certainty threshold can be set to a lower value, e.g., 0.95. This reduces the number of combinations that need to be analyzed. However, there is now a 0.05 probability of mispredicting the impact of uncertainty.

As described previously in Section 1.3.1 and illustrated in Figure 1.3, the RAP method estimates a *deterministic* resource allocation plan for each combination of workload scenarios that attempts to minimize the global SLO violation metric for the applications in this combination given the resource constraints of the cloud. As mentioned previously in this

section, each combination of workload scenarios has an associated probability of occurrence. By considering the deterministic resource allocation plans for all combinations of workload scenarios and the probabilities of occurrence for these combinations, a probabilistic resource allocation plan is obtained. A *probabilistic resource allocation plan* defines for a given tier a Probability Density Function (PDF) for the number of resources required by either one or all applications over each resource allocation interval in the planning horizon so that the applications' SLO requirements are satisfied. Figure 1.5 shows an example of a PDF for the number of web server instances required by a set of applications over one of the resource allocation intervals within the planning horizon. The x-axis shows list of values for a parameter denoted by $C_{max,n,t}$. $C_{max,n,t}$ is defined as the maximum number of resources available to all applications at an application tier n in a given resource allocation interval t. In Figure 1.5, the subscript n is replaced with *web* to refer to the web tier. By experimenting with various $C_{max,n,t}$ values, different PDFs, one for each application tier n, can be computed for all applications over each of the resource allocation intervals in the planning horizon. These PDFs are then used to construct the probabilistic resource allocation plan provided by the uncertainty module in conjunction with RAP.

The probabilistic resource allocation plan helps a cloud SP track the probabilities of application resource requirements exceeding available resource capacity over a given planning horizon. These probability estimates can help a cloud SP assess the SLO violation risks triggered by workload uncertainty. Figure 1.6 shows an example of an analysis obtained from a probabilistic resource allocation plan provided by the uncertainty module in conjunction with RAP. Figure 1.6 reports the probability of the number of resource instances for a specific tier, e.g., web server instances, required by all applications to satisfy their SLO requirements at each of the resource allocation intervals in the planning horizon. For example, Figure 1.6 shows that there is a 0.9 probability in resource allocation interval 6 that a cloud SP requires more web server instances than the maximum capacity limit, i.e., $C_{max,web,t}$, to satisfy the

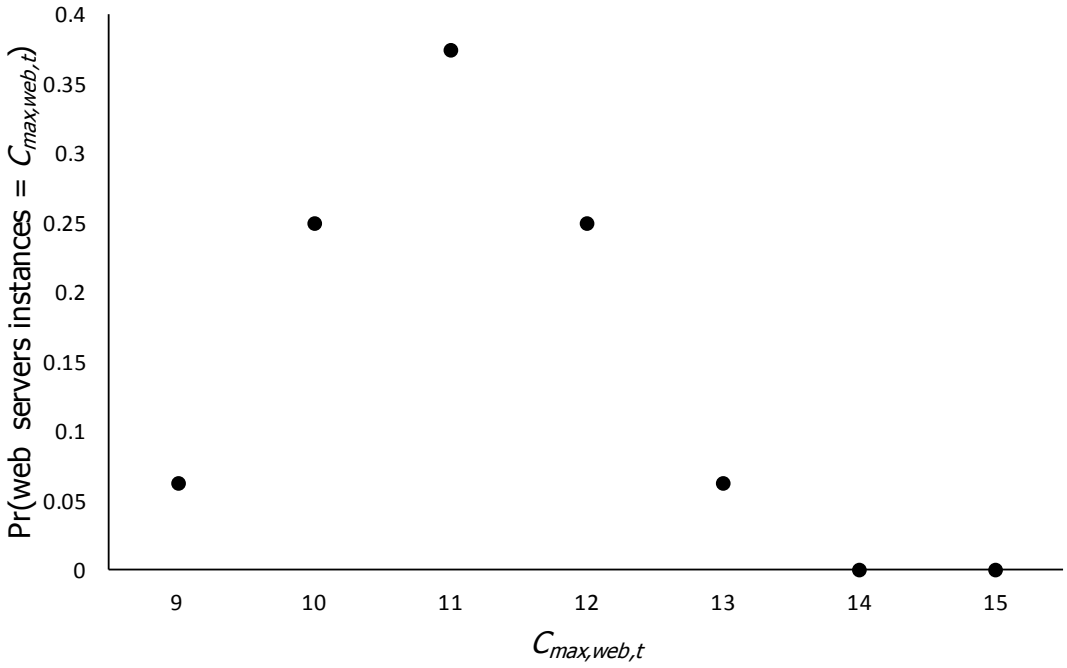


Figure 1.5: Example of a resource PDF generated for one of the resource allocation intervals

SLOs of all applications.

The results obtained in the thesis demonstrate that the proposed Monte Carlo simulation technique enables cloud SPs to accurately estimate the impact of workload uncertainty in their SLP exercises without exhaustively traversing all combinations of application workload scenarios. Specifically, an experiment is conducted where five applications are considered with each having two alternative workload scenarios. One of the alternative workload scenarios is used to represent the normal operation of the application while the other scenario is characterized by a heavier load relative to the normal workload scenario. As described previously in this section, this gives a total possible combinations of workload scenarios of 2^5 = 32. The probability of encountering the heavy load workload scenario for each application is set to 0.1. The experiment considers values of 1.0 and 0.9 for the certainty threshold described previously. By setting the certainty threshold to 1.0, all 32 possible combinations of workload scenarios are covered. A certainty threshold value of 0.9 causes the uncertainty module to generate an average of only 9 combinations which have a cumulative probability

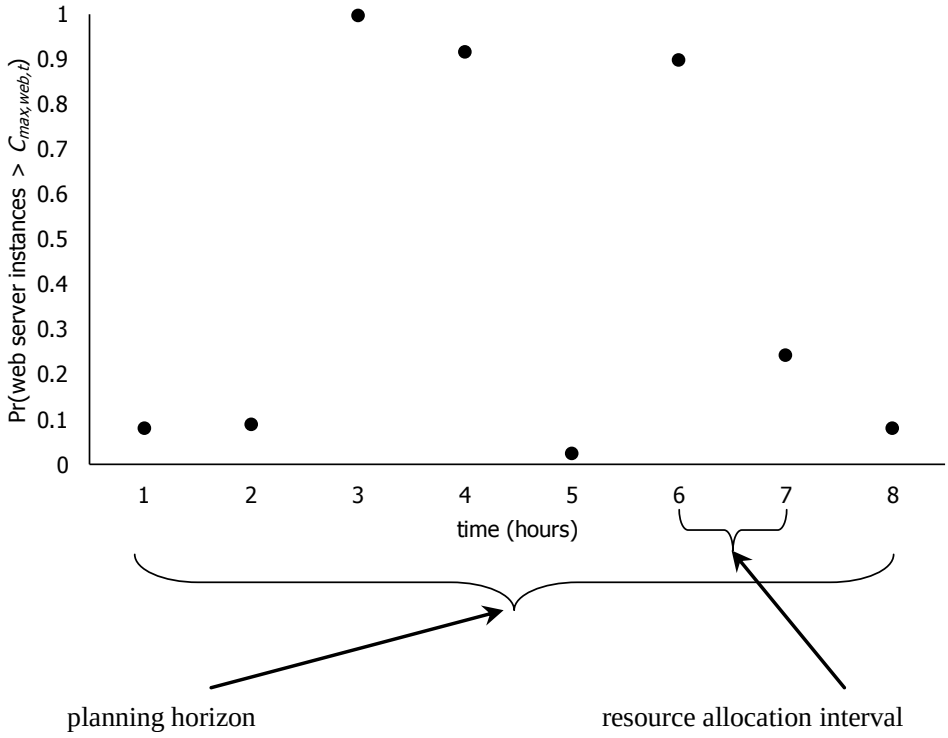


Figure 1.6: Analysis using probabilistic resource allocation plan

of occurrence of at least 0.9. This results in an average reduction of 72% in the number of combinations that need to be analyzed for SLP. In spite of this dramatic reduction, considering only 8 combinations of workload scenarios for SLP generates results that are very close to the results obtained by considering all possible 32 combinations of workload scenarios. This behaviour occurs since not all the possible combinations have a very high likelihood of occurrence.

1.3.3 Workload Clustering Technique

To enhance the scalability of SLP, a new *burstiness-aware* workload clustering technique is developed. The objective of this technique is to group applications with similar workload characteristics together into a small number of clusters and creates equivalent workload scenarios for these clusters. These workload scenarios are referred to as *clustered workload scenarios*. The proposed workload clustering technique performs two main steps namely,

cluster generation and *clustered workload scenarios generation*. The cluster generation step assesses the similarity in workload characteristics between workload scenarios that belong to the same combination generated by the uncertainty module as described previously in Section 1.3.2. This step groups applications with similar workload characteristics into clusters.

The clustered workload scenarios generation step implements an algorithm which takes the clusters generated in the cluster generation step as input and generates clustered workload scenarios to represent each cluster. The main objective of this algorithm is to generate a clustered workload scenario for each cluster to accurately preserve the observed workload burstiness of the applications which belong to this cluster. SLP is then conducted at the granularity of these clusters to facilitate faster solutions. This allows cloud SPs to handle large number of applications typically deployed on the cloud and large number of workload scenarios per application.

The results obtained show that the proposed workload clustering technique is effective in grouping applications with similar burstiness characteristics together. Specifically, the results show the need to consider both coarse-grained, i.e., over entire planning horizon, and fine-grained, i.e., over each resource allocation interval, workload characteristics for the clustering approach to be effective under burstiness. Examples of coarse grained workload characteristics considered in the thesis include mean session arrival rate over the planning horizon and a parameter called the *index of dispersion* defined also over the planning horizon. The index of dispersion is used by Casale *et al.* [41] to characterize burstiness in application workloads. Examples of fine-grained workload characteristics considered in the thesis include the mean session arrival rate over each resource allocation interval within the planning horizon.

Furthermore, the results show that the accuracy of resource allocation is increased by increasing the levels of clustering and the number clustering attributes considered. Specifically, a hierarchical clustering approach that uses three levels of clustering and the clustering

attributes mentioned in the above paragraph provides a maximum resource allocation error of 10%. However, clustering at one level using only one of the attributes mentioned above for clustering gives an error in the range of 25% to 40%.

The results also show the need to implement automatized workload scenario generation algorithms to reconstruct a workload scenario for each cluster to obtain accurate resource allocation plans. Specifically, the algorithm developed to generate the clustered workload scenarios involves accumulating session arrivals at very small time periods, i.e., in the order of minutes, within each resource allocation interval. Each of these time periods is referred to as an *arrival window*. Within each arrival window a histogram of session arrival values is computed to develop an empirical PDF which is then used to generate traces of session arrivals for clusters. More details about this technique will be described in Chapter 6.

Finally, the results obtained show that the proposed workload clustering technique reduces the number of computations needed to support SLP exercises without significantly impacting accuracy. In particular, the resource allocation plan obtained by the proposed workload clustering technique for a case study of 100 applications is compared with the resource allocation plan obtained when no clustering is applied. The results show a maximum resource allocation error of 10% while the reduction obtained in the number of applications considered for SLP is in the range of 88% to 91%. This significant reduction in the number of applications occur because of the high degree of similarity between the workload scenarios of the applications considered in this experiment.

1.3.4 Trace-Based SLP Framework

In this section a trace-based SLP framework is proposed to integrate the above three modules together. Figure 1.7 shows the interactions between the various components in the SLP framework. Firstly, as described in Section 1.3.2 due to the large number of applications involved in SLP, each with multiple workload scenarios, an explosion can occur in the number of alternative workload scenario combinations that need to be considered during the

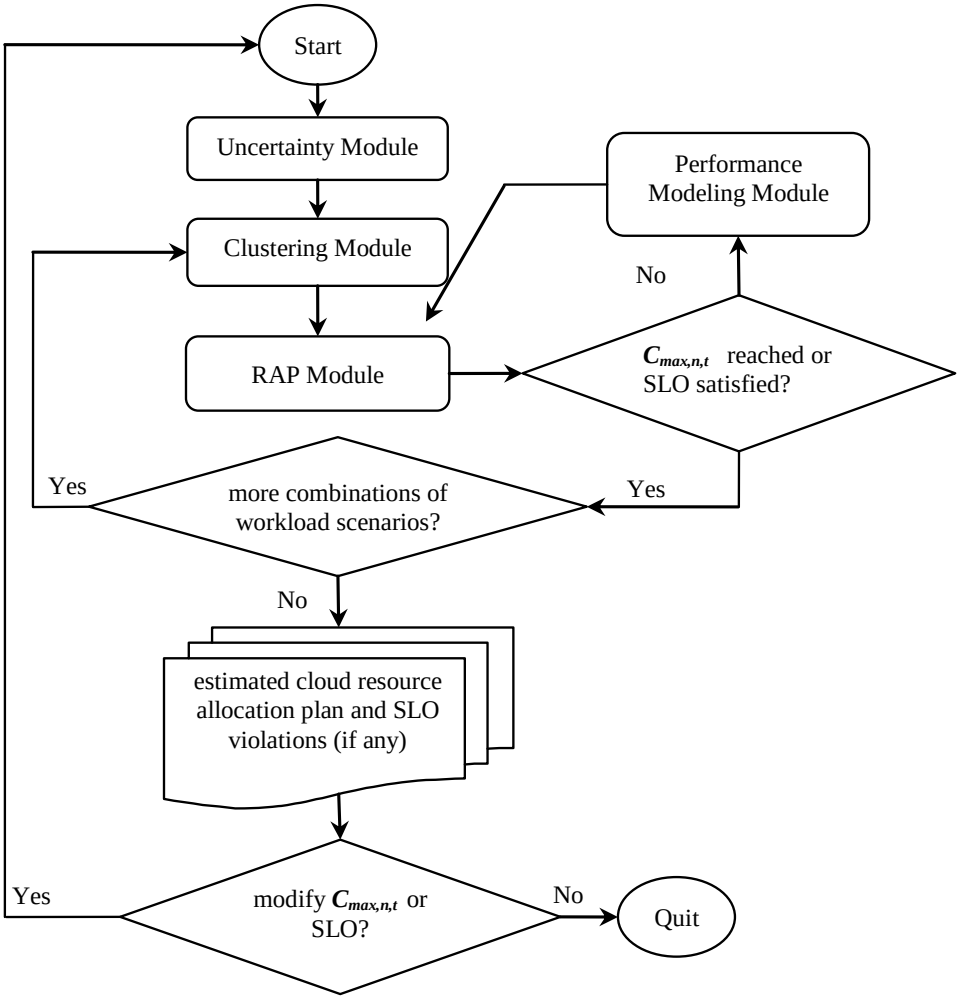


Figure 1.7: SLP framework flowchart

SLP process. The uncertainty module selects a subset of the most likely workload scenario combinations. This is done by leveraging the probabilities of occurrence associated with individual workload scenarios of each application and specifying a value for the certainty threshold parameter as described previously in Section 1.3.2.

Secondly, the clustering module further improves the scalability of the framework. For every workload scenario combination evaluated, the module groups applications with similar workload scenarios together into a small number of clusters and creates equivalent clustered workload scenarios for these clusters. Resource allocations are then explored at the granularity of clusters.

Finally, for each of the clustered workload scenario combinations generated by the clustering module, the RAP module comes up with a deterministic resource allocation plan over a given planning horizon with a specified resource allocation granularity. The RAP module relies on the WAM performance prediction technique which is capable of handling workload scenarios based on traces. The performance modeling module is employed by RAP to predict the performance of the set of applications considered, and hence SLO violations, given a candidate assignment of resources to applications and the applications' traces. This process is repeated for all workload scenario combinations identified by the uncertainty module and clustered using the clustering module. Results are then aggregated taking into account the workload uncertainty information to offer probabilistic estimates of resource requirements and SLO violations.

The information provided by the SLP framework can help a cloud SP understand the physical resources needed to host a given set of applications. Information on the estimated fluctuations in resource usage of applications can also be leveraged by a cloud SP to plan efficient deployments of these applications on the cloud. Furthermore, the framework permits a cloud SP to estimate the cost of hosting each application on the cloud to satisfy its SLO requirements. The framework also provides SLO violation estimates for customers' applications which allow cloud SPs to quantify the penalties incurred due to SLO violations. Moreover, the framework can be used by a cloud SP to experiment with various resource pool capacities to come up with a plan that strikes a balance between resource costs and SLO violation costs.

A cloud SP can deploy applications on the cloud based on the estimates provided by the SLP framework and monitor SLO violations over time. If any deviations occur in customers' workloads after deployment leading to SLO violations, revised estimates of customers' workload scenarios in addition to any changes in resource instance limits, i.e., $C_{max,n,t}$, can be input to the framework again to consider changes in resource allocations. This can also

be used to negotiate new SLAs with customers to specify new SLOs for observed customer workloads. Furthermore, the SLO violations predicted by the framework 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. This iterating process of estimating and deploying resources can be invoked on a regular basis, for example weekly, bi-weekly or monthly, to account for any deviations in customer workloads.

1.3.5 End-To-End Case Study

This section illustrates the end-to-end operation of the proposed SLP framework with a case study. For details on the simulation setup the reader can refer to Section 4.2. In this case study, 100 applications are considered. While 95 applications have only one workload scenario, the other 5 applications have uncertainty in their workloads. Specifically, they include an additional heavy workload scenario with a probability of occurrence of 0.1. To fully cover all possible workload scenario combinations where each combination contains 100 workload scenarios pertaining to 100 applications, 2^5=32 workload scenario combinations have to be analyzed. As described previously in Section 1.3.2 the uncertainty module requires a cloud SP to specify a value for the certainty threshold that represents a trade-off between coverage of all possible workload scenario combinations that can arise due to workload uncertainties and SLP computation time. In this experiment a certainty threshold value of 0.9 is used. This generates a subset of the 32 possible workload scenario combinations with a cumulative probability of occurrence of *at least* 0.9.

Table 1.2 shows a detailed list of workload combinations identified in the end-to-end case study. As shown in the table, the uncertainty module identified 8 combinations of workload scenarios involving these 100 applications that spanned a cumulative probability of occurrence of 0.96. This reduces the SLP computation time by about 75% with respect to the exhaustive evaluation of all possible combinations. Each of these combinations is

Table 1.2: Workload scenario combinations identified in the end-to-end case study

Combinations identified	Probability of occurrence	Number of clusters
Combination 1	0.59	12
Combination 2	0.07	9
Combination 3	0.07	12
Combination 4	0.01	9
Combination 5	0.07	11
Combination 6	0.07	9
Combination 7	0.01	10
Combination 8	0.07	9
Total	0.96	

considered in order by the SLP framework. For each combination, workloads with similar workload characteristics are then clustered together using the clustering module. Table 1.2 provides information on the number of clusters identified for each of these combinations. For example, the number of workload scenarios which are considered for SLP in combination number 2 is reduced from 100 to 9 due to clustering. This reduces the SLP computation time by more than 90%. This reduction does not affect the accuracy of resource allocation significantly as will be shown later in Section 6.4. Synthetic workload scenarios are generated for clusters identified for each combination and these workload scenarios are then input to the RAP module.

The RAP module was configured to estimate the number of resources at two application tiers, i.e., web and database, over all resource allocation intervals needed to satisfy application SLOs. These results are obtained for all 8 combinations and used in conjunction with the probability of occurrence of each combination as shown in Table 1.2 to obtain a probabilistic resource allocation plan as described previously in Section 1.3.2. Figure 1.8 shows the probability of the cloud requiring more than certain number of web server instances, $C_{max,web,t}$, to satisfy all applications' SLOs over a planning horizon of 8 hours with 1-hour resource allocation intervals. This figure explores the effect of imposing different web server instance constraints, $C_{max,web,t}$, on the risk of satisfying the SLOs of all applications. For example, in resource allocation interval 3 there is over 0.9 likelihood that the number of web

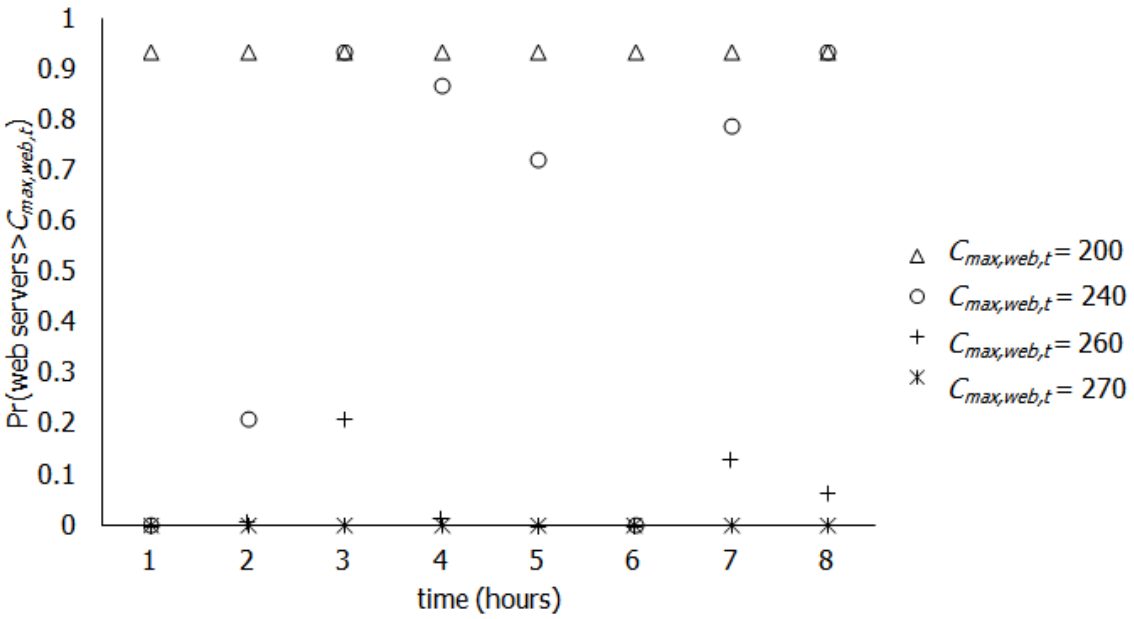


Figure 1.8: Sensitivity analysis of the end-to-end case study of the SLP framework

server instances required to avoid SLO violations is more than 240. This probability is reduced to 0.2 and reaches zero by relaxing the limits on the number of web server instances to 260 and 270, respectively. In summary, the figure suggests that provisioning 270 web server instances in each resource allocation interval would allow a cloud SP, with a probability of 0.96, to avoid all risks of SLO violations for this scenario.

1.4 Publications

The work in this thesis is published as follows:

1. Anas Youssef and Diwakar Krishnamurthy. A Trace-Based Service Level Planning Framework for Enterprise Application Clouds. In *Proceedings of the International Conference on Network and Service Management (CNSM)*, pages 422-426, 2011

2. Anas Youssef and Diwakar Krishnamurthy. Cloud Service Level Planning

Under Burstiness. In *Proceedings of the International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 107-114, 2013.

3. Anas Youssef and Diwakar Krishnamurthy. A Burstiness and Uncertainty-Aware Service Level Planning Framework for Clouds. *IEEE Transactions on Software Engineering,* 2013. Under second round of review

1.5 Thesis Organization

The thesis is organized as follows. Chapter 2 discusses background information and previous work in areas related to the work in this thesis. Chapter 3 provides a detailed description of the proposed RAP method. The simulation setup implemented to evaluate the RAP method and the results obtained are presented in Chapter 4. Chapter 5 provides a detailed description of the characterization and incorporation of workload uncertainty in SLP and presents associated evaluation results. Chapter 6 describes in detail the proposed burstiness-aware clustering technique and presents associated evaluation results. Chapter 7 provides a detailed description of the SLP framework and presents statistics on the computational complexity of the framework. Chapter 8 summarizes the thesis conclusions, discusses the limitations of the SLP framework and outlines future research directions. Finally, a set of Appendixes are listed. Specifically, Appendix A lists all notations used throughout the thesis while Appendix B describes the dynamic programming formulation of RAP-DP and Appendix C provides a proof of the optimality of RAP-DP.

Chapter 2

Background And Related Work

This chapter describes background information and previous work in areas related to the work developed in this thesis. Section 2.1 introduces background information on cloud computing. Section 2.2 discusses the prevalence of workload burstiness in enterprise applications and the metrics used to characterize burstiness in application workloads. Section 2.3 describes two different performance modeling approaches used for resource allocation in this work namely, the traditional MVA-based approach and the trace-based WAM approach. Section 2.4 discusses related work in cloud resource management including long term and short term resource management approaches. Section 2.5 describes the k-means clustering technique which is used in the thesis. Finally, Section 2.6 provides an overview of the dynamic programming optimization approach which is employed in this work.

2.1 Cloud Computing

This section describes background information on cloud computing. Firstly, an introduction to cloud computing and a description of its benefits to enterprises are presented. Secondly, a description of a typical cloud architecture is provided. Thirdly, two cloud categories are described namely, public and private clouds. Finally, a description of Amazon Web Services [8] is provided including computing [3], storage [7] and database [6] services. Furthermore, the Auto Scaling feature [9] of Amazon EC2, which is related to the work developed in the thesis, is discussed.

Many enterprises today are using the cloud computing model to host their IT services. As described previously in Section 1.1, a cloud is a pool of interconnected resources implemented and managed by SPs to support the hosting of different IT systems or services [37, 95]. The

cloud computing model provides a "pay-per-use-only" paradigm for delivering IT services to customers in the same manner as traditional utilities such as water, gas, electricity and telephony [37]. Using this model enterprises have to pay for using resources only when required without the need to purchase dedicated resources to host their applications or provide their services.

A typical cloud computing architecture includes the following components [37]: Physical resources, VMs, SLA resource manager, and user interface to manage the cloud. Todays cloud SPs implement different variations of this architecture to support a number of features such as virtualization, heterogeneity of resources, automatic scaling and adaptation, resource management and optimization, fault tolerance, QoS management and "pay-per-use" payment model [95].

Clouds can belong to two main categories namely, public and private clouds. The *public cloud* provides services to the general public and can be shared among different organizations. The *private cloud* is operated and managed internally by a single organization for the dedicated use of this organization [29]. Large organizations may prefer to set up their own private clouds as this might be more cost effective than being charged per unit time, e.g., per hour, by public cloud SPs [90]. Furthermore, private clouds are usually preferred over public clouds when organizations are more concerned about the security of their businesses. Public clouds are still facing unresolved security challenges which limit their use by some organizations and enterprises [29].

In order for organizations to set up their own private clouds, they need cloud software framework [90]. Several open source cloud platforms exist which can be used to build such private clouds. Examples of these platforms are Eucalyptus [10], OpenNebula [19] and Nimbus [18]. The three cloud platforms provide the IaaS type of service described previously in Section 1.1. They differ in a number of features such as the degree of security supported, the number of machines and users that can be supported in the cloud and other network

Table 2.1: Summary of Amazon EC2 resource instance types

Instance type	**Performance**
General-purpose	Provides a balance of compute, memory, and network resources
Compute-optimized	Optimized for compute-intensive applications
Memory-optimized	Optimized for memory-intensive applications
Storage-optimized	Optimized for applications with specific disk I/O and storage capacity requirements
Micro Instances	Very low-cost instances used to increase the compute capacity in short bursts
GPU Instances	Supports parallel processing using the Graphics Processing Unit(GPU)

configuration issues [90].

Amazon provides a set of cloud services called *Amazon Web Services (AWS)* [8] which include computing, storage and database services. The computing service is referred to as *Amazon Elastic Compute Cloud (Amazon EC2)*. Amazon EC2 provides customers with VMs that operate with various operating systems using resource instances of different computing capacities. Amazon EC2 supports different types or flavours of resource instances to satisfy various performance requirements of customers' applications. The prices for Amazon EC2 resource instances are offered on an hourly basis. Table 2.1 shows a summary of selected types of resources instances supported by Amazon EC2. AWS provide a storage service called *Amazon Simple Storage Service (AmazonS3)*. Amazon S3 allows customers to store, manage and secure their data on the web. The prices for Amazon S3 are offered on a monthly basis for certain amount of data stored, e.g., the number of terabytes stored per month. Finally, the database service supported by AWS is referred to as *Amazon Relational Database Service (AmazonRDS)*. Amazon RDS is a specific web service that provides customers with specific database resource instances to create, manage, and scale a relational database in the cloud. Examples of database engines that can be created include MySQL, Oracle and Microsoft SQL Server. Like Amazon EC2, Amazon RDS offers per-hour prices for the database resource instances supported.

Amazon EC2 offers a dynamic resource management feature called *Auto Scaling* [9]. Auto

Scaling allows customers to dynamically increase or decrease their Amazon EC2 resource instances to match peaks and troughs in workload demand. Therefore, this feature allows customers to minimize resources costs by utilizing resources efficiently while satisfying the demand and performance required by their applications. To use the Auto Scaling feature of Amazon EC2, customers should create Auto Scaling Groups for their resource instances. An *Auto Scaling Group* is a collection of Amazon EC2 instances to which certain scaling conditions are applied. The *scaling conditions* are thresholds defined to pre-selected applications' performance metrics to automatically trigger the addition or removal of Amazon EC2 resource instances while the applications are running. An example of a scaling condition is to add new Amazon EC2 small instances in increments of two to the Auto Scaling Group when the average CPU utilization of the group goes above 70% for more than 5 minutes and remove Amazon EC2 instances in the same increments when the average CPU utilization falls below 10% for the same time period. Applications' performance metrics are regularly monitored using a tool called *Amazon CloudWatch* [1].

In spite of its perceived advantages, auto scaling in cloud environments is not a straightforward task and involves a number of complications [74, 75, 89]. Firstly, auto scaling is typically performed based on low level applications' performance metrics such as CPU utilization [75]. As mentioned previously in Section 1.1 and will be shown later in this thesis, CPU utilization might not accurately reflect high level application performance such as user perceived response time especially with application workloads characterized by burstiness. Consequently, more appropriate performance metrics should be used for auto scaling to be effective in satisfying an application's performance.

Secondly, auto scaling involves acquisition and removal of resource instances while applications are running. These acquisition and removal times are not negligible and may affect the performance of the applications deployed on the cloud [74, 75]. For example, a recent study shows that an average of 10 minutes is required to start a resource instance

on Windows Azure [61]. Furthermore, as mentioned previously in this section, resource instances are typically charged per hour where a fraction of an hour is still charged as a whole hour. This means that an acquisition or removal of resource instances should be avoided in the middle of an hour otherwise unnecessary resource costs are incurred. For example, if a resource instance is started at the start of an hour and is then shut down and restarted in the same hour, the customer will be charged for two instance hours rather than one. A third complication is how long the resource instance should be allocated which depends on the patterns of applications' workloads. Accurate prediction mechanisms of workload patterns are required by auto scaling tools to decide on the time required to allocate resource instances in the cloud [89].

Thirdly, the relation between the processing power of a resource instance and its prices is not linear [75]. For example, the general-purpose instance type shown in Table 2.1 includes a resource instance called *m1.small* while the compute-optimized instance type includes a resource instance called *c1.medium*. The price of c1.medium is twice as much as the price of m1.small while the processing power of c1.medium is five times that of m1.small. Thus, for applications which need a large processing power allocating one c1.medium instance type is cheaper than allocating three m1.small instance types. Consequently, an auto scaling decision of selecting the appropriate number and type of resource instances required to improve performance and save resource costs is not a straightforward task.

Finally, as described previously in Section 1.1, short-term tools used for auto scaling have to make scaling decisions at real time while applications are running. It may not be feasible for these tools to do global optimization over short time scales for large number of applications typically deployed on the cloud.

To alleviate some of the auto scaling problems described above, the SLP framework described previously in Section 1.3.4 can provide customers with pre-deployment baseline resource allocation strategies that define the number and types of resource instances required

over time based on their historical workload patterns. These resource allocation strategies can help customers to reduce the auto scaling decisions made to satisfy the performance of their applications while they are running, therefore reducing the complexity associated with these decisions.

2.2 Workload Burstiness

This section introduces background information on workload burstiness in enterprise applications and describes the parameters used to characterize burstiness in application workloads. Recall from Section 1.1 that workload burstiness is defined as serial correlations or dependencies between successive events which occur in application workload patterns. Examples of these events are arrivals and completions of requests at various system resources. The number of requests arriving to system resources is the reciprocal of the mean inter-arrival time of requests. The *inter-arrival time* of a request is the difference between the arrival instant of the request and the arrival instant of the previous request. The number of requests completed at a system resource is the reciprocal of the mean service time of requests completed at this system resource. The *service time* of a request at a given system resource is defined as the time that elapses from the time instant the request starts receiving service from the resource until the time instant the service is completed.

Correlations can occur between successive requests arriving to system resources if there are correlations between the values of inter-arrival times of these requests. For example, in a correlated workload a high inter-arrival time of a request might imply that the next request is also likely to have a high inter-arrival time. Similarly, correlations can occur between successive request completions at system resources if there are correlations between the values of service times of these requests. Correlations between inter-arrival times or service times of successive requests can be positive or negative. Correlation between two successive requests is said to be positive if the values of their inter-arrival times or service

times change in the same direction, i.e., increase together or decrease together. On the contrary, correlation between two successive requests is said to be negative if the values of their inter-arrival times or service times change in opposite directions.

Workload burstiness may significantly affect the performance of applications deployed in the cloud. This is because an application can experience sustained periods of heavy load followed by sustained periods of relatively low load or even system idle periods. When an application experiences sustained periods of heavy load, this may increase resource contention which can degrade application's performance significantly. Furthermore, workload burstiness may result in underutilized of resources when sustained periods of low load or system idle periods are encountered. Consequently, customers may incur unnecessary resources costs for these low load periods.

Several studies characterized the behaviour of enterprise application workloads [28, 36, 57, 77, 94]. Many of these studies indicated the prevalence of workload burstiness in real workloads. Menasc *et al.* [77] 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. Another study made by Vallamsetty *et al.* [94] confirmed the above findings by characterizing traffic arriving at two other real e-commerce sites. Gmach *et al.* [57] characterized five weeks of workload traces for 139 enterprise applications hosted on a commercial data center. The authors reported strong bursty nature of CPU demands for most of the enterprise applications characterized in this study. Bodik *et al.* [36] analyzed real workload traces from five public Internet sites. These traces included web server logs from World Cup 1998 website [28], UC Berkeley Electrical Engineering and Computer Sciences Department website, Wikipedia.org and Twitter.com. The study showed that these sites experienced periods of arrival burstiness of user requests. Consequently, these studies suggest that workload burstiness should be taken into consideration during capacity planning and resource provisioning [39, 43].

Recall from Section 1.3.1 that an application workload scenario is defined as a collection

of traces. These traces include a session trace which captures information about user sessions related to the application over a period of time and resource traces which capture the application's utilization of low-level resources over the same period. Both session and resource traces can have a bursty nature. The work in this thesis focuses on characterizing burstiness in session traces. Recall from Section 1.3.1 that a session is a group of inter-related requests to perform a certain task. The inter-arrival time of a session is equal to the inter-arrival of the first request in the session. Session traces experience burstiness when there are correlations between the inter-arrival times of successive sessions arriving to system resources. To characterize burstiness in session traces, the work in the thesis exploits previous work proposed in [41–43,60]. Gusella [60] characterized burstiness in the inter-arrival times of network packets while Casale *et al.* [41–43] characterized burstiness in requests service times. This work focuses on characterizing burstiness in session inter-arrival times.

According to the work proposed by Gusella, the burstiness in the arrivals of sessions cannot be accurately captured by simple metrics such as coefficient of variation of session inter-arrival times. The *Coefficient of Variation (CV)* of session inter-arrival times is defined as:

$$CV = \frac{\sigma}{\mu} \tag{2.1}$$

where σ and μ are the standard deviation and mean of session inter-arrival times, respectively. The CV of session inter-arrival times is a statistical measure of the relative variability of session inter-arrival times to the mean. To characterize burstiness in session arrivals, more advanced metrics are defined in the work proposed by Gusella [60]. In this work two dimensionless metrics are defined namely, the *Index of Dispersion for Intervals (IDI)* and the *Index of Dispersion for Counts (IDC)*. These metrics are used to to characterize burstiness in network packets. The IDI is used to characterize burstiness for packet inter-arrival times while the IDC is used to characterize burstiness for the number of packet arrivals in a given time interval. They can be used interchangeably [46,60] to characterize burtsiness in session

inter-arrival times and number of session arrivals, respectively. These two metrics will be described in more detail the ensuing paragraphs.

In addition to the CV of session inter-arrival times, the IDI metric considers the auto-correlation 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 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 the *Squared Coefficient of Variation (SCV)* of session inter-arrival times which is the squared value of CV and a set of ρ_k where $k \geq 1$, the IDI, denoted by I, is defined as:

$$I = SCV(1 + 2\sum_{k=1}^{\infty} \rho_k) \tag{2.2}$$

As shown in Equation (2.2), I depends on SCV 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 value of 1 [41, 60]. This implies that the I value represents the deviation of any observed set of session inter-arrival times from a Poisson process. High value of SCV 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 trace under consideration. Workloads with bursty session arrivals will have very large values for I 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 [39]. In particular, the authors showed that the classic FIFA World Cup trace of web request arrivals has an I value of 8400.

Measuring the value of I based on Equation (2.2) is not practically feasible due to the infinite summation of the ρ_k values. To address this issue Gusella [60] defined the IDC metric. As mentioned previously in this section, IDC characterizes burstiness for number of session arrivals in a given time interval. Given the number of sessions that arrived at a

system in a time window t[1], denoted by Arr_t, the IDC provides an alternative definition for I[2] as follows:

$$I = \lim_{t \to +\infty} \frac{Var(Arr_t)}{E(Arr_t)} \tag{2.3}$$

where *Var(Arr_t)* and *E(Arr_t)* are the variance and mean values of Arr_t, respectively, over a time window t.

Using Equation (2.3) the value of I can be estimated practically. Casale *et al.* proposed an algorithm [41] to estimate an approximate value of I for request completions using Equation (2.3). In this thesis the same algorithm is implemented with minor modifications to estimate the value of I for session arrivals. Figure 2.1 shows a pseudo-code for this algorithm.

The main idea of the algorithm is that an approximate value of I can be computed if the number of session arrivals at a system is observed over a large time window t, e.g., 2 hours. This time window is divided into smaller O equally sized time periods each of which is refereed to as the *sampling resolution*, e.g., 60 seconds. Each sampling resolution is denoted by o. In each $o \in \{1,2...O\}$, the number of session arrivals, denoted by Arr_t^o, is computed. This gives a sequence of Arr_t^o values over t. The variance, *Var(Arr_t)*, and mean, *E(Arr_t)*, of this sequence of values are then evaluated to compute the value of I as defined in Equation (2.3). This process is repeated for larger values of t, e.g., 4, 6, 8 hours, etc... until the value of I achieves a given convergence criteria.

The algorithm shown in Figure 2.1 takes the following inputs:

- *SamplingResolution* which is the size of each o sampling resolution over which the value Arr_t^o is collected.
- *TimeIncrement* which is the initial size of t and is used to increase the size of t in each iteration.

[1]Note that this time window is not the same as the resource allocation interval defined in Section 1.3.1
[2]In the thesis I refers interchangeably to both IDI and IDC

Input:
SamplingResolution (e.g. 60 secs)
TimeIncrement (e.g. 2 hours)
Z (e.g. 100)
Tol (e.g. 0.2)
Output: *I*
t = TimeIncrement
Y(0) = 1
J = 1
While (*ConvergenceMetric* > *Tol*)
O = t / SamplingResolution
Compute a sequence of Arr_t^o values over *t*

if the number of session arrivals per *SamplingResolution* is not enough
restart the algorithm after adjusting the *SamplingResolution*
If any of the computed Arr_t^o values < *Z*
*SamplingResolution = SamplingResolution * J*
t = TimeIncrement
Y(0) = 1
J = J + 1
Break
End If

Given the sequence of Arr_t^o values compute *Y*(*t*)
Y(*t*) = *Var*(*Arr*$_t$) / *E*(*Arr*$_t$)
ConvergenceMetric = | 1 – *Y*(*t*) / *Y*(*t* – *TimeIncrement*) |
t = t + *TimeIncrement*
End While
I = Y(t)
Return *I*

Figure 2.1: Estimation of index of dispersion

- Z which is the lowest number of arrivals that can be observed in each o sampling resolution.

- *Tol* which is the convergence tolerance of the algorithm. This value is used to halt the algorithm when the value of I converges.

The values assumed for the algorithm inputs are based on the assumptions made by Casale *et al.* in [41].

The algorithm shown in Figure 2.1 is iterative. In each iteration, O time periods are formed by dividing t into O equally sized sampling resolutions. The number of session arrivals, Arr_t^o, is counted over each o time period. Given this sequence of Arr_t^o over t, the ratio *Y(t)* is computed. A convergence criteria, *ConvergenceMetric*, is used to compare the value of *Y(t)* with a previous value *Y(t - TimeIncrement)*. If the value of *Y(t)* converges, the algorithm returns this value as I, otherwise t is incremented and another iteration is started. If the sequence of Arr_t^o values collected in any iteration is less than Z, the algorithm should be restarted after setting the *SamplingResolution* value to a higher value.

2.3 Performance Modeling

As described in Section 1.1, SLP tools usually rely on performance models to predict performance of applications given a resource allocation strategy. This section describes the performance modeling approaches employed in the thesis. Specifically, two performance modeling approaches are described namely, traditional MVA-based modeling and trace-based WAM modeling approaches.

2.3.1 Traditional MVA-Based Modeling Approach

A QNM [65, 79] is a network of inter-connected queues where each queue represents one of the resources of a computer system. Each queue is referred to as a *service center* which serves the *customers* arriving to the system. A service center can be a delay or a queuing

center. If a request is submitted to a delay service center, it is served immediately with no waiting time in a queue, whereas if a request is submitted to a queuing service center, it can spend more time waiting in a queue before being served. This waiting time depends on the number of requests ahead in the queue. Each service center has a scheduling discipline which is the method that determines how customers gain access to the service center. Examples of scheduling disciplines include First Come First Serve (FCFS) and Processor Sharing (PS).

A customer of a QNM can be a request or a session. Recall from Section 1.3.1 that a session is a collection of inter-related requests that fulfill a certain task. Customers arriving to a computer system can impose different demands on system resources. However, it is cumbersome to model the usage profile of system resources for every single customer. Therefore, customers with similar usage profiles are grouped together into classes where each class represents the average usage of system resources among the customers in the class. For example web browsing customers can be modeled by a class which is different from customers who make a payment order. A QNM can serve a single or multiple classes of customers.

A customer class can be open, closed or hybrid. An *open* customer class has no bounds on the number of customers being served by the QNM. A *closed* customer class specifies a limited number of customers that are allowed to use the QNM service centers. A *hybrid* customer class, as the name implies, is a combination of both open and closed customer classes where no bounds are specified on the number of customers arriving to the QNM but each customer specifies a complex workload with a limited number of subcustomers, e.g. requests. An example of a hybrid customer class is the session-based workload which is composed of unlimited number of user sessions arriving at a system with each user session formed of a limited number of user requests. Each customer class has two types of data namely, workload intensity and service demands [79]. The *workload intensity* of a class is the parameters that define the load imposed by its customers on the QNM service centers. The workload intensity of an open class is specified in terms of the number of customers

arriving to the system per unit time while the workload intensity of a closed class is specified in terms of the customer population of the class. The *customer population* of a closed class is the number of customers that are concurrently being served by the QNM service centers. The *service demands* of a class are the parameters that define the average aggregate service times required by the customers of the class at various service centers. A QNM can be open, closed or mixed depending on the type of its customer classes. A QNM is said to be open if all its customers classes are open, closed if all its customer classes are closed, and mixed if some of its customer classes are open and some are closed.

Figure 2.2 shows an example of a QNM of a computer system which has two application tiers namely, web and database tiers. The web tier is modeled by X queuing service centers that operate in parallel where each service center is used to model a server having a single processor core. The database tier is modeled by a single queuing service center that is used to a model a server having Y processor cores. The QNM shown in Figure 2.2 is subjected to an input trace of customer sessions each of which is formed of limited number of requests. Thus, the customer class considered in this QNM is hybrid. Each session starts by submitting a request to the QNM. After the first request in the session is served, another request is submitted to the web tier after spending a portion of time in a delay service center which is used to model the think times between successive requests in the same session. A session leaves the QNM when all its requests complete their service.

The QNM of a computer system can be solved analytically using a technique such as MVA [79]. As described previously in Section 1.1, such analytical performance modeling approaches are usually employed in SLP for multi-tier applications since they allow a large number of resource allocation plans to be analyzed quickly. Specifically, analytical performance models are simpler and faster than simulation models when a large number of resource allocation plans are explored at different application tiers. This section describes briefly the traditional MVA analytic performance modeling technique [79] which solves product-form

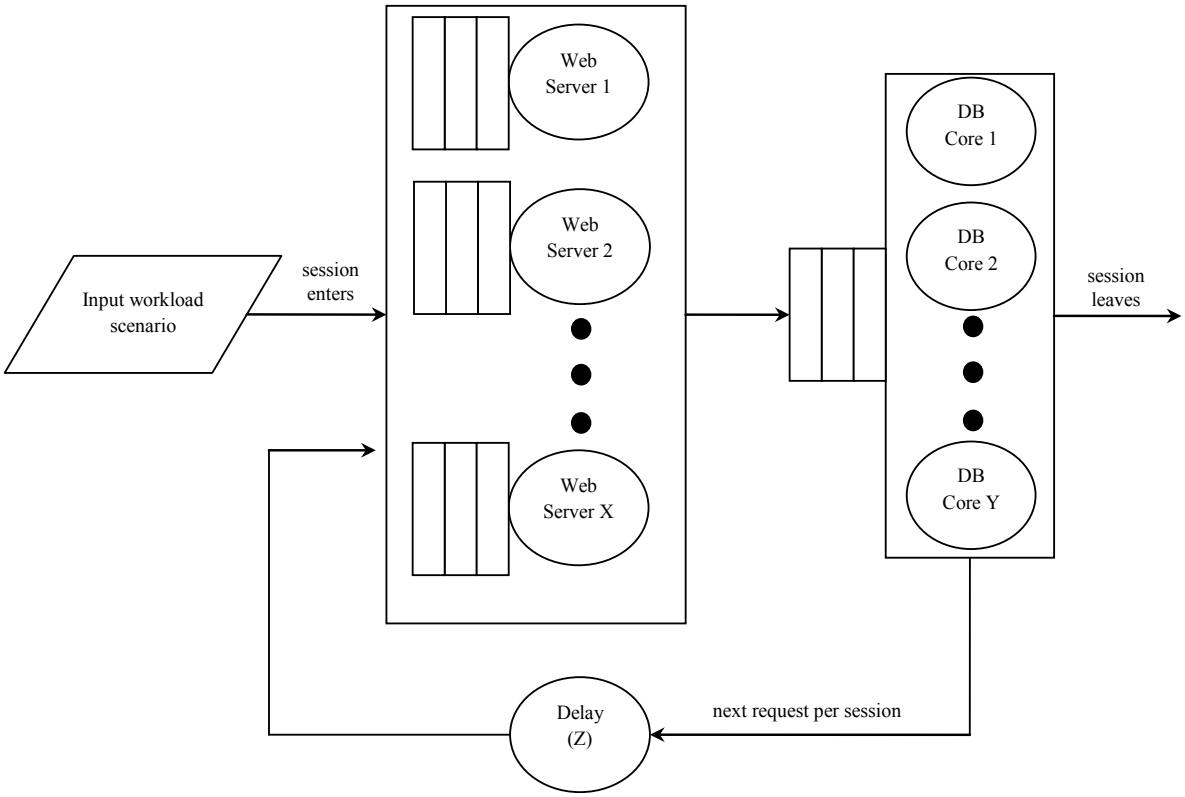


Figure 2.2: Example of a QNM

QNMs [32] that posses a set of assumptions. Product-form assumptions are a set of assumptions that permit efficient analytic solutions of QNMs [65]. MVA is one of the modeling techniques employed by the proposed RAP method described briefly in Section 1.3.1. The thesis refers to this technique as *MVA-QNM*.

MVA-QNM uses a product form QNM to capture an application's use of cloud resources and employs the MVA technique [79] to obtain application performance estimates for a given resource allocation plan. The QNM employed is characterized by the following inputs for a given application:

- number of resources assigned to each application tier, e.g. web and database tiers.
- average request service demands for various application tiers.

- average number of concurrent sessions observed in the queuing network.
- average session think time which is the idle time between successive requests in a session.

MVA-QNM uses the above inputs to derive the average request response time at each application tier when there are S concurrent sessions in the network. The *request response time* at a given tier is the time between the time instant at which the request enters the queue of tier and the time instant at which the request completes its service in the tier. These request response times are then used to compute the overall response time of the system modeled.

Product-form QNMs solved using the traditional MVA technique has been used extensively in SLP. Urgaonkar *et al.* validated such a model for the Rice University Bidding System (RUBIS) multi-tier benchmark application [92]. However, product-form QNMs have several well known limitations [32]. For example, they cannot cannot handle correlations between the inter-arrival times and service times of user requests and sessions. Therefore, MVA-QNM cannot yield accurate performance estimates for systems whose workloads exhibit burstiness in either request arrivals or resource consumption patterns [42, 43, 81]. Furthermore, MVA-QNM abstracts session and resource traces using average quantities such as the average number of concurrent sessions and average request service time at each tier. This may introduce further errors in predictions.

Casale *et al.* proposed a new technique called MAP-QNM to solve closed QNMs which are characterized by burstiness in request service demands [41–43]. The authors show a method that provides upper and lower bounds on several application performance metrics such as request throughput and response time when request service demands are bursty. While MAP-QNM can produce more accurate estimates under service time burstiness than traditional MVA, they are computationally intensive. Furthermore, they currently lack support for several features that are useful for modeling enterprise applications such as modeling contention for software resources, e.g., web server threads, modeling multiprocessors, and

modeling the PS scheduling discipline that is typically used to represent operating system schedulers. This thesis employs a technique developed by Krishnamurthy *et al.* [64] called WAM that can handle burstiness in session arrivals. WAM is described in more detail in Section 2.3.2.

2.3.2 Trace-based WAM Modeling Approach

This section describes the trace-based WAM performance modeling technique [64] employed by the proposed RAP method as described briefly in Section 1.3.1. WAM considers session traces pertaining to applications which is a key difference from MVA-QNM that considers mean quantities pertaining to session traces of applications as described previously in Section 2.3.1. WAM is a hybrid technique that combines a performance modeling technique such as MVA-QNM for a given system under study with a fast trace-based simulation. Specifically, WAM peruses the session trace that is part of an application workload scenario to estimate a probability distribution of the number of concurrent customer sessions at the application system. This distribution is referred to as the *population distribution*. For each population of customer sessions in this population distribution the MVA-QNM is solved to predict performance metrics pertaining to this population as will be shown later. The population distribution reflects any bursty session arrival behavior that might be present in the application's workload scenario. Specifically, more bursty session arrivals cause higher probabilities of observing extremely small and extremely large number of concurrent sessions at the system. In this way, the probability distribution captures the impact of burstiness in session arrivals. For details on the trace-based simulation employed by WAM to estimate the probability distribution, the reader can refer to [64].

For a given application, let the number of concurrent sessions at the application system be s, where $s \in \{1,2...S\}$. The set of S values represent all session population levels observed at the system under study. For each session population, s, an MVA-QNM is solved to predict the mean request response time, R_s, and the mean request throughput, X_s, under a given

resource allocation plan. Each session population, s, has a probability of occurrence, P_s, as given by the population distribution.

The per-population request response times, R_s, are weighted by the per-population request throughputs, X_s, and the per-population probabilities, P_s, to offer an overall request response time prediction, R_{mean}, using Little's Law [79] as shown in the following equation.

$$R_{mean} = \frac{\sum_{s=1}^{S} P_s X_s R_s}{\sum_{s=1}^{S} P_s X_s} \tag{2.4}$$

While the use of a simulation component and the need to predict performance for many different session populations make this technique slower than MVA-QNM, it is shown to produce more accurate results for bursty workloads [64]. Furthermore, due to its combined use of simulation and analytic modeling, WAM is faster than approaches that rely solely on simulation. Relying only on simulations slows down the analysis of large number of resource allocation plans that need to be analyzed for SLP. Given different types of available resources for a given application system, simulations performed to obtain the system's response time values are time consuming as opposed to obtaining them by quickly solving an MVA-QNM for the system. Finally, WAM can be integrated with analytic performance models other than MVA-QNM to support modeling of complex computer systems. For example, WAM can be integrated with LQMs [88] that are capable of modeling contention for software resources. The thesis refers to the WAM technique as *WAM-QNM*

2.4 Cloud Resource Management

This section describes state-of-the-art work in cloud resource management. Specifically, two main approaches to cloud resource management are discussed namely, long term and short term resource management approaches. These approaches will be discussed in the following sections.

2.4.1 Long Term Resource Management

This section describes related work in long term resource management. As described previously in Section 1.1, Long term resource management tools help cloud SPs to plan their infrastructure capacity and decide on SLAs with a customer prior to deploying the customer's applications. As mentioned previously, these tools are referred to in the thesis as SLP tools.

As described previously in Section 1.1 SLP tools are typically used to plan for cloud resource management over coarse-grained time periods using historical traces [58] of applications that describe the workload experienced over the time period under consideration. A number of such SLP tools have been introduced in the literature [24, 38, 53, 56, 63, 67–69, 71, 80, 83, 87].

Rolia *et al.* [87] proposed an SLP tool, *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. 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. However, as will be shown in this thesis, such utilization-based approaches cannot accurately estimate resources required for application workloads characterized by burstiness. The proposed work in this thesis allows both utilization and response time thresholds to be specified directly as part of an application's SLO.

Jung *et al.* [63] proposed an SLP approach which combined an LQM [52] with a heuristic optimization algorithm to generate optimal server configurations. These configurations were then encoded in the form of rules and policies to be used while the system is running. In contrast to the work proposed in this thesis, 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 WAM [64] that is used in this paper.

Mylavarapu *et al.* [83] 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. In contrast to the proposed work, this work only considered SLOs based on CPU utilization targets and did not explicitly handle workload uncertainty.

Meng *et al.* [80] proposed a trace-based approach for SLP in compute clouds through VM multiplexing. 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 the proposed work in the thesis 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 the proposed work considers several service and arrival process parameters that are of importance to SLP. Moreover, the authors did not consider any uncertainties associated with applications workloads.

Instead of using analytic performance modeling in SLP, simulation-driven techniques can be used to help SPs analyze cloud resource provisioning policies under varying workloads. *CloudSim* toolkit was proposed by Calheiros *et al.* [38] to allow SPs to model and evaluate the performance of various cloud components and resource provisioning policies. However it does not allow SPs to incorporate workload scenarios based on traces of request arrivals and service times and hence it cannot handle complex workload behaviors such as burstiness. Furthermore, simulation techniques are more time consuming than analytic modeling techniques.

Malkowski *et al.* [71] proposed an SLP tool, *CloudXplor*, to support resource allocation based on empirical datasets. This empirical dataset is obtained by running a large number

of experiments over a wide range of resource allocation plans and workloads in various environments. Trends present in this dataset are then exploited in resource allocation exercises.

Li *et al.* [67–69] proposed an offline approach for fast optimal deployment of a given set of applications in a resource pool. This work combined an LQM [88] with bin-packing and a linear programming approach based on Network Flow Models (NFMs) [67]. Similar to our proposed work, the authors address scalability by formulating the NFM in a manner that permits a large number of applications to be handled. The authors stated that their approach can provide solutions in 1 or 2 minutes while considering up to 20 applications [69]. Li studied in [66] techniques to apply this approach dynamically while the system is running. Unlike the work proposed in this thesis, this work did not consider workload uncertainty and the impact of bursty workloads.

Chaisiri *et al.* [44] proposed an optimization algorithm that can help a cloud SP to optimally provision cloud resources to the cloud customers. This algorithm provides an optimal long term plan of resources to applications deployed on the cloud taking into account the uncertainties that might occur in the applications' service demands and in the prices of the cloud resources. As opposed to the work proposed in this thesis, this work tries to minimize the total cost of cloud resource provisioning without addressing the problem of optimally allocating resources to applications to satisfy their SLOs. The authors of this work exploited an important feature of current cloud offerings such as Amazon EC2 which is the low price of reserving instances in the cloud relative to purchasing instances when needed. For example, purchasing Amazon EC2 Reserved Instances can provide up to 65% reduction in resource costs over On-Demand Instances [2]. This feature provides a strong motivation to the work proposed in the thesis.

Various commercial offline capacity planning tools have been offered by different vendors. Examples of such tools include *VMware Capacity Planner* [23] , *VMware vCenter CapacityIQ* [22], *NetIQ PlateSpin Recon* [17], and *Lanamark Suite* [15]. To the best of the author's

knowledge based on their data sheets, these tools cannot support SLP based on response time targets. However, they control resource utilization thresholds to indirectly achieve adequate application request response times. Moreover, they do not explicitly consider SLP for complex workloads characterized by burstiness and uncertainty.

2.4.2 Short Term Resource Management

This section describes related work in short term resource management. As described previously in Section 1.1, short term resource management tools adjust cloud resources dynamically to handle any sudden workload fluctuations during system operation. These tools are based on measurements obtained from the system over fine-grained time scales in the order of seconds or minutes. Significant research efforts have focused on techniques for short term resource management. As described previously in Section 1.1 these approaches complement long term techniques such as the SLP framework proposed in this thesis, therefore, a review of selected work in this area is presented. Broadly, work in this area can be categorized into approaches that rely on performance models and those that rely on control theory [91].

Online resource management techniques that rely on performance models employ a queuing network performance model to predict user-level performance metrics such as throughput and response time. This performance model is usually embedded within a runtime controller that controls resource allocations to an application so as to satisfy its performance 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 minutes [91].

Urgaonkar *et al.* [93] proposed a performance-model based approach to dynamically reconfigure an application based on observed and predicted changes to its workload behaviour. In contrast to the proposed SLP framework which attempts to solve a global SLO and resource allocation optimization problem for a set of applications, this work was concerned with local optimization of a single application. Moreover, it did not take into account resource

availability constraints and workload burstiness.

Ardagna *et al.* [25, 26] proposed a runtime resource allocation scheduler that assigns multi-tier applications to physical resources at fine timescales to satisfy cost and performance objectives. The approach uses a multi-class QNM and a heuristic search algorithm that solves a mixed non-linear programming problem. The authors demonstrated that their solution techniques execute faster and provide better resource allocation solutions than traditional optimization techniques. As opposed to the work in this thesis, this work did not consider workload burstiness and uncertainty.

Online resource management techniques that rely on control theory 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.

Gmach *et al.* [54, 55] proposed an approach to integrate two types of controllers: a long-term workload placement controller and a short-term workload migration controller. Another integrated workload management architecture composed of multiple resource controllers was proposed by Zhu *et al.* [99] to consolidate different application workloads having SLOs on a large data center. In contrast to the work proposed in this thesis, both of these approaches determined workload placements based on CPU utilization and did not consider the effect on response time.

Malkowski *et al.* [72] proposed a multi-model controller for dynamically provisioning VMs to multi-tier applications in clouds. A knowledge base was 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 was used to drive future provisioning decisions. The authors pointed out that a large collection of performance data spanning multiple different configurations, workload conditions, and SLA specifications were needed for the knowledge base to be effective.

Caniff *et al.* [39] presented an online resource provisioning algorithm, *Fastrack*, which exploits workload burstiness to guide dynamic resource allocation in multi-tier systems. Fastrack detects the bursty periods in the application's workload trace and accordingly allocates more resources to these periods. This algorithm is quite similar to the work proposed in the thesis in trying to exploit workload burstiness to save the number of resources allocated. However, unlike the proposed work which considers all applications in a cloud simultaneously while making resource allocation decisions, this work only focused on a single application.

2.5 k-means Clustering

This section describes background information on the clustering techniques used in the thesis to group applications with similar workload characteristics together. In general, the objective of clustering [73] is to group a set of data points into subsets or clusters. These clusters should be clearly different from each other such that the degree of similarity between the data points that belong to the same cluster should be as as high as possible while the degree of similarity between the data points that belong to different clusters should be as low as possible. Data points are grouped into clusters using a single or multiple clustering attributes. A *clustering attribute* is a variable that defines a specific characteristic to a data point and is used to assign the data point to a cluster. For example, a data point can be a point in a two-dimensional space characterized by the Cartesian coordinates of the point. Each data point in the thesis is a workload scenario of an application.

There are two main approaches to clustering namely, flat and hierarchical clustering [50,73]. *Flat clustering* creates a flat set of clusters without any explicit structure that relates these clusters to each other. This flat set of clusters is created by applying clustering once to all data points. *Hierarchical clustering* creates a hierarchy of clusters by applying clustering successively in distinct iterations. The hierarchy of clusters represents a structure that is more informative than the unstructured set of clusters created by flat clustering. There

are two approaches to hierarchical clustering namely, bottom-up and top-down approaches. Bottom-up hierarchical clustering starts with individual data points and successively merges these data points into clusters. This approach is referred to as *agglomerative clustering*. Top-down hierarchical clustering starts with all data points grouped in one cluster and splits clusters recursively until individual data points are reached. This approach is referred to as *divisive clustering*. Top-down hierarchical clustering is conceptually more complex than bottom-up clustering since it needs a flat clustering algorithm to be applied at each iteration. However, it has the advantage of being more efficient when a complete hierarchy all the way down to individual data points is not needed. Top-down hierarchical clustering is the clustering approach used in the thesis. At each iteration the k-means clustering algorithm [50, 70] is employed.

The k-means clustering algorithm aims to partition a set of l data points, (x_1, x_2,..., x_l), into k non overlapping clusters, $k<l$. Each cluster, i, contains the set of data points denoted by Φ_i so as to minimize the Within-Cluster Sum of Squares (WCSS) defined as:

$$WCSS = \sum_{i=1}^{k} \sum_{x_j \in \Phi_i} \|x_j - \mu_i\|^2 \tag{2.5}$$

where μ_i is the centroid of cluster i. The *centroid* of a cluster acts as a prototype of the data points in the cluster and is computed based on the values of these data points. For example, a centroid of a cluster can be the mean value of the data points which belong to the cluster. k-means clustering aims to assign each data point, x_j, to the cluster, Φ_i, which has the nearest centroid to x_j among all clusters.

To assign a data point to a cluster the distance between the variables or clustering attributes of the data point and the corresponding variables or clustering attributes of the centroid of each cluster should be measured. To measure this distance, a distance metric is required. Many distance metrics are proposed in the literature [50]. An example of such a distance metric is the Euclidean distance. Given two points p and q where each point is characterized by a vector of r variables, the *Euclidean distance*, $DI_{p,q}$, between the two

points is defined as:

$$DI_{p,q} = \sqrt{\sum_{i=1}^{r}(q_r - p_r)^2} \tag{2.6}$$

where (p_1, p_2, ..., p_r) and (q_1, q_2, ..., q_r) are two r-dimensional vectors which characterize the points p and q, respectively.

The Euclidean distance defined in Equation (2.6) can be squared in order to place greater weight on the data points that are farther apart. This is referred to as the *Squared Euclidean distance*. The work in this thesis uses this metric to measure the distance between data points, i.e., application workload scenarios.

In general, the k-means clustering algorithm [50, 70] operates as follows:

1. Initially, k data points are selected from the l data points. The selected data points are referred to as *seeds*. Each seed forms a cluster of exactly one data point which is the centroid of the cluster.

2. The data points which are not selected in step 1 are then examined in sequence. Each of these data points is assigned to the cluster with the nearest centroid in terms of the distance metric considered such as the Euclidean distance defined in Equation (2.6). When a data point is assigned to a cluster, the centroid of the cluster is recalculated to take into account the added data point.

3. For each data point, the distances between the data point and the centroids of all clusters are computed. If the data point belongs to a cluster which does not have the lowest distance among all clusters, the data point is moved to the cluster with the lowest distance.

4. The centroids of the clusters which experience any change in their data point memberships are recalculated.

5. Steps 3 and 4 are repeated until no more data points are moved between clusters or until a specific maximum number of iterations is reached. If there

are no movements of data points between clusters, then the clusters are said to be stable and the algorithm converges to a final solution.

The k-means clustering algorithm has a number of limitations [84]. Firstly, the final solution obtained by k-means clustering is highly dependent on the way the initial cluster centriods are selected. To overcome this limitation, different methods to initialize the k-means clustering algorithm are proposed in the literature, however, this is out of the scope of the thesis. In this work, the initial cluster centriods are selected at random.

Secondly, the k-means clustering algorithm assumes the *prior* knowledge of the number of clusters, k, before running the algorithm which might not be feasible in practice. Furthermore, selecting the wrong number of clusters may result in too few or two many clusters. Specifically, if the number of clusters is too small, data points with widely different clustering attributes may belong to the same cluster. On the other hand, if the number of clusters is too large, data points that belong to different clusters may have similar clustering attributes. Furthermore, specifying too large number of clusters may result in only a small reduction in the number of data points which might not be beneficial for some applications. In summary, effectiveness of the solution obtained by k-means clustering is highly dependent on the number of clusters specified.

To overcome the limitation of requiring prior knowledge of the number of clusters, the work in the thesis employs an approach described by Menasce *et. al.* [76] to automatically select the number of clusters. This selection optimizes the degree of similarity between the data points within the cluster and the degree of similarity between the clusters. Specifically, two random variables are considered namely, intracluster and intercluster distances. For a cluster i, the *intracluster distance* random variable, denoted by $d(x,\mu_i)$ is defined as follows:

$$d(x, \mu_i) = \textit{the distance between any data point, } x \in \Phi_i \textit{ and } \mu_i \tag{2.7}$$

The *intercluster distance* between clusters i and j, denoted by $\bar{CD}_{i,j}$, is defined as follows:

$$\bar{CD}_{i,j} = \textit{the distance between the centroids of clusters } i \textit{ and } j \textit{ such that } i \neq j \tag{2.8}$$

These two random variables are used in the thesis to automatically decide on the number of clusters, k, used as input to the k-means clustering algorithm as shown in the ensuing paragraph.

The average intracluster distance for a cluster i, denoted by $\bar{d}_i$, is defined as follows:

$$\bar{d}_i = \frac{1}{r_i} \sum_{x \in \Phi_i} d(x, \mu_i) \tag{2.9}$$

where r_i is the number of data points that belong to cluster i. Given $\bar{d}_i \ \forall \ i \in \{1,2,...,k\}$, three statistical measures are computed for the intracluster distance namely, sample mean denoted by d, sample variance denoted by σ^2_{intra} and sample coefficient of variation denoted by CV_{intra}. These statistical measures are computed as follows:

$$d = \frac{1}{k} \sum_{i=1}^{k} \bar{d}_i \tag{2.10}$$

$$\sigma^2_{intra} = \frac{1}{k-1} \sum_{i=1}^{k} (\bar{d}_i - d)^2 \qquad k > 1 \tag{2.11}$$

$$CV_{intra} = \frac{\sigma_{intra}}{d} \tag{2.12}$$

The same three statistical measures are computed for the intercluster distance which are the sample mean denoted by CD, sample variance denoted by σ^2_{inter} and sample coefficient of variation denoted by CV_{inter}. These statistical measures are computed as follows:

$$CD = \frac{1}{\frac{k(k-1)}{2}} \sum_{i=1}^{k} \sum_{j=i+1}^{k} \bar{CD}_{i,j} \qquad k > 1 \tag{2.13}$$

$$\sigma^2_{inter} = \frac{1}{\frac{k(k-1)}{2} - 1} \sum_{i=1}^{k} \sum_{j=i+1}^{k} (\bar{CD}_{i,j} - CD)^2 \qquad k > 2 \tag{2.14}$$

$$CV_{inter} = \frac{\sigma_{inter}}{CD} \tag{2.15}$$

The objective of the approach described in [76] is to select the number of clusters that maximizes the similarity between the data points that belong to each cluster while minimizing the similarity between the data points that belong to different clusters. To achieve this objective the intercluster variance, σ^2_{intra}, should be minimized while maximizing the intracluster variance, σ^2_{intra}. This can be done by minimizing either of the following two ratios:

$$\beta_{var} = (\frac{\sigma_{intra}}{\sigma_{inter}})^2 \quad , \quad \beta_{cv} = \frac{CV_{intra}}{CV_{inter}} \tag{2.16}$$

The values of β_{var} and β_{cv} decrease by increasing the number of clusters. If the number of clusters, k, is equal to the number of the data points, l, considered for clustering, then both β_{var} and β_{cv} are minimized, however, the clustering process will be of no use in this case. Therefore, Menasce *et. al.* described in [76] an empirical method to select an appropriate value of k for clustering. In this method k different values, starting from 1 to l, are tried. Either β_{var} or β_{cv} values are calculated and compared for all k values. The result of this process is selecting the k value above which no significant decrease is achieved in the β values considered. In this way, the selected value of k achieves a relatively small number of clusters while obtaining small values of β_{var} or β_{cv}. It should be noted that the value of k selected in this manner is completely dependent on the data points under consideration.

2.6 Dynamic Programming

This section describes the motivation for selecting the dynamic programming optimization method [49] which is employed by the proposed RAP-DP technique described briefly in Section 1.3.1. The section also provides background information on the dynamic programming method.

As described previously in Section 1.3.1, this work proposes a set of RAP techniques to solve the global SLO and resource allocation optimization problem. This optimization problem involves a non-linear objective function as will be described later in Section 3.1. However, there is no closed form for the objective function which rules out traditional non-linear programming optimization techniques [48]. Existing work that solves similar optimization problems uses heuristic search techniques such as hill climbing [78] to find near optimal solutions. The work proposed by Bennani *et al.* in [35] also uses other heuristic search techniques to solve a similar optimization problem.

Solving the global SLO and resource allocation optimization problem using discrete optimization techniques such as *Branch-and-Bound* is prohibitive with the increase in the number of applications considered for SLP. The Branch-and-Bound technique [48] starts from an initial solution and successively generates branches of possible solutions. Each branch is then evaluated against a specific bound where branches which are expected to generate solutions that will violate this bound are not explored. For example, if the objective is to minimize a cost function, the least cost value obtained from the branches explored is used as a bound to prune other non explored branches. Any non explored branch is not evaluated if the cost obtained from this branch is expected to be higher than the bound. For large scale problems such as the global SLO and resource allocation optimization problem, it is prohibitive to use the Branch-and-Bound technique [48]. This is because a large number of solutions has to be evaluated to obtain the optimal solution which might not be possible for large number of applications deployed on the cloud and for a wide variety of resource types supported to each application tier. Therefore, as will be shown later in the thesis, the dynamic programming optimization method is used to provide a more salable solution for the global SLO and resource allocation optimization problem.

Dynamic programming is an optimization method used to solve complex problems by breaking them down into a set of simpler interrelated subproblems. Each of the subproblems

is solved individually and the solutions of the subproblems are then combined together to reach an overall solution for the complex problem. A problem that can be solved using dynamic programming should exhibit two characteristics namely, overlapping subproblems and optimal substructure. A problem is said to have *overlapping subproblems* if it can be broken down into interrelated subproblems which are reused multiple times. A popular example of this problem is the problem of calculating the terms of the Fibonacci series. The value of any term i, of the Fibonacci series is the sum of the previous two terms of the series, i.e., (i-*1*)th and (i-*2*)th terms. The problem of evaluating the ith term in the series is broken into two subproblems in which the (i-*1*)th and (i-*2*)th terms are evaluated and then the solutions of the two subproblems are combined to obtain the value of the ith term.

The second characteristic of problems that can be solved using dynamic programming is optimal substructure. A problem is said to have *optimal substructure* if an optimal solution for the problem can be constructed from optimal solutions to its subproblems. The optimal substructure characteristic of a problem is described by Richard Bellman who defined the *Principle of Optimality* [33] as follows:

"An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."

Bellman's Principle of Optimality implies that an optimal solution can be obtained for a given complex problem by obtaining the optimal solutions of its simpler subproblems. For example, consider the problem of calculating the shortest path between two points , a and f, as shown in Figure 2.3. To solve this problem using dynamic programming, the shortest path from a to f can be obtained by recursively obtaining the shortest paths from each of points b and c to f. Each direct link between any two points, i and j, in the figure has an associated *cost*, denoted by *C(i,j)*. For example, the cost, *C(a,b)*, of the direct link between points a and b is 2. Any two points, i and j, can be connected with more than one path.

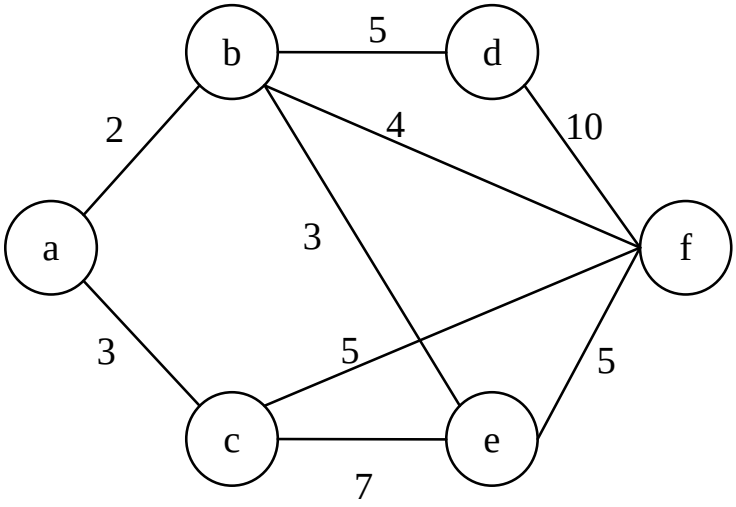


Figure 2.3: Obtaining the shortest path between two points, a and f, using dynamic programming

The set of paths connecting points, i and j, is denoted by $\Psi_{i,j}$. Each of these paths, ψ , has an associated total cost, denoted by TC_{ψ} which is the sum of the costs of the direct links that form this path. This cost is used to determine the shortest path between any two points in Figure 2.3.

Dynamic programming involves a multi-stage process to obtain an optimal solution for a given complex problem. This process starts by breaking the complex problem into simpler subproblems as described previously. In the initial stage an optimal solution is obtained for the simplest subproblem among all subproblems. This solution should be obtained directly from the statement of the problem. In any subsequent stage an optimal solution is obtained for one of the subproblems using the optimal solution of a relatively simpler subproblem obtained in the previous stage. To solve a problem using this process, three terminologies should be defined [49] namely, an optimal value function, a recurrence relation and a boundary condition. The *optimal value function* is a rule which is used to assign a value to the

solution obtained for each subproblem of the given problem. For example, in the shortest path problem shown in Figure 2.3, the optimal value function, *S(a,f)*, can be defined as:

$$S(a, f) = min \quad TC_{\psi} \quad \forall \psi \in \Psi_{a,f} \tag{2.17}$$

According to Bellman's Principle of Optimality the optimal value function of any subproblem should be related to the optimal value function of another subproblem. This relation is defined by a formula or a set of formulae called the *recurrence relation*. For example, the following recurrence relation can defined for the optimal value function, *S(a,f)*:

$$S(a, f) = min \begin{bmatrix} C(a, b) + S(b, f) \\ C(a, c) + S(c, f) \end{bmatrix} \tag{2.18}$$

Finally, as described previously, the solution of the simplest subproblem that is solved in the initial stage should be obtained directly from the statement of the problem without the need for any computations and without dependence on any other stages. This obvious solution is referred to as the *boundary condition* of the problem. For example, the boundary condition for the shortest path problem shown in Figure 2.3 can be defined as:

$$S(f, f) = C(f, f) = 0 \tag{2.19}$$

which states that the minimum total cost among the different paths connecting point f and itself is simply the cost of the direct link between f and itself which is zero.

2.7 Summary

This chapter describes background information and previous work in areas related to the research work developed in this thesis. Firstly, background information on cloud computing is introduced. This information describes typical cloud architecture and cloud services, different categories for cloud systems and examples of commercial and open source clouds. Moreover, a brief description of AWS is provided including computing, storage and database

services. The Auto Scaling feature of Amazon EC2 is also described with a brief discussion of its challenges. Secondly, the prevalence of burstiness in application workloads is discussed and the characterization of workload burstiness using the index of dispersion is described. Also an algorithm for the estimation of the index of dispersion is described.

Thirdly, the performance modeling approaches which are employed in the thesis are introduced. This includes two performance modeling approaches which are the traditional MVA-based modeling and the trace-based WAM modeling approaches. Fourthly, related work in cloud resource management is discussed. This includes long term and short term management approaches. Fifthly, k-means clustering which is employed in the thesis is described. Finally, background information on dynamic programming is introduced. This optimization method is employed by the RAP-DP technique described briefly in Section 1.3.1.

Chapter 3

Resource Allocation Planning (RAP) Method

This chapter provides a detailed description of the RAP method described briefly in Section 1.3.1. Section 3.1 describes the global resource allocation optimization problem which is solved by the RAP method. Section 3.2 describes an overview of the RAP method. Section 3.3 discusses in detail the three RAP variants described briefly in Section 1.3.1. Finally, Section 3.4 describes two SLP approaches which do not take into account workload variability and burstiness. These approaches will be compared with RAP in Section 4.6.

3.1 Global SLO and Resource Allocation Optimization Problem

This section describes in detail the global SLO and resource allocation optimization problem described briefly in Section 1.1. Recall from Section 1.1 that the global SLO and resource allocation optimization problem for a set of applications is a multi-objective optimization problem in which a pool of available resources are allocated to these applications so that each application's SLO violations are minimized while using the least possible number of resources. The 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 cloud SP or one of them can be assigned a higher priority than the other. The work in this thesis focuses on minimizing the SLO violations objective. The study of more complex objective functions that explicitly minimizes resource costs is deferred to future work.

The optimization problem assumes a given set of A applications. Each application a employs a multi-tier architecture which is composed of N_a tiers. For ease of explanation, it is assumed in this work 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 described previously in Section 1.3.2, each application a is characterized by a set of probable workload scenarios, denoted by W_a. Each probable workload scenario is denoted by $W_{a,k}$ where $W_{a,k} \in W_a$ and $k \in \{1,...L_a\}$. L_a is the number of alternative workload scenarios per application to account for uncertainty in customer workloads. In this chapter each application is assumed to have only one workload scenario, i.e., $k = L_a = 1$, however, L_a can take other values as will be shown later in Chapter 5.

As described previously in Section 1.3.1, the optimization problem assumes 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 $C_{max,n,t}$. 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 Section 1.3.1, RAP relies on a performance model to predict applications' SLOs given applications' workloads scenarios and resources allocated to application tiers. This work mainly considers SLOs based on mean response time. However, the proposed RAP method can invoke performance modeling approaches which are capable of predicting other response time statistics such as *95th* percentile of request response times.

The SLO violation percentage V_a for an application a is defined as:

$$V_a = \begin{cases} \frac{R_a - RT_a}{RT_a} * 100\% & \text{if } R_a > RT_a \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

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 T resource allocation intervals constituting the planning horizon. When T equals 1 then the resource allocation interval is the same as the planning horizon. This allows cloud SPs to handle SLO requirements of customers who are more concerned about bursts which may occur at certain resource allocation intervals rather than a single SLO requirement over a wider planning horizon.

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

$$min \sum_{a=1}^{A} Pen_a \tag{3.2}$$

$$Pen_a = f(V_a) \tag{3.3}$$

$$s.t.\ C_{a,n,t} \geq 1\ \forall a \in \{1, ...A\}, n \in \{1, ...N_a\}, t \in \{1, ...T\} \tag{3.4}$$

$$s.t.\ \sum_{a=1}^{A} C_{a,n,t} \leq C_{max,n,t}\ \forall n \in \{1, ...N_a\}, t \in \{1, ...T\} \tag{3.5}$$

$$R_a = f(W_{a,k}\ k \in \{1, ...L_a\}, C_{a,n,t}\ \forall n \in \{1, ...N_a\}, t \in \{1, ...T\}) \tag{3.6}$$

Equation (3.2) specifies the objective to minimize SLO violation penalties over all applications. Equation (3.3) defines an SLO violation penalty cost Pen_a for a given application a as function of the application violation percentage V_a defined in Equation (3.1). 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 (3.4) and (3.5) place upper and lower constraints on the size of the SP's resource pool, respectively. Equation (3.4) shows that each application, a, should be allocated at least one resource of an appropriate flavor in each tier, n, in every interval, t. Equation (3.5) 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 (3.6) relates application mean response time R_a as a function of one of its probable workload scenarios $W_{a,k}$ and the resources allocated to all tiers $n \in \{1,...N_a\}$ of application a for all resource allocation intervals $t \in \{1,...T\}$.

The formulation of the optimization problem provides flexibility with respect to the overall objective of the SLP process. From Equations (3.2) and (3.3) 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:

$$min\ SV = \sum_{a=1}^{A} V_a \tag{3.7}$$

If the cloud SP is interested in minimizing the number of applications violating their SLOs, it can be assumed that all applications' violation penalties are equal. Specifically, each Pen_a value in Equation (3.2) can be defined as a binary variable which takes zero for no violation and one for any positive violation percentage. In this way the objective defined in Equation (3.2) leads to minimizing the total number of applications with positive Pen_a values. Minimizing the SV objective defined in Equation (3.7) is the focus of this work. To solve the global resource allocation optimization problem presented in this section, the RAP method described briefly in Section 1.3.1 is proposed. The next section provides an overview of the RAP method.

3.2 RAP Overview

Recall from Section 1.3.1 that the RAP method allows SPs to a priori garner insights on the relationship between application workloads, resource allocation policy, cloud capacity, and application service levels. To effectively support such an objective, the RAP method should mimic the elastic nature of resource allocation in clouds. For example, cloud SPs should be able to use the RAP method to emulate dynamic allocation and de-allocation of resources to applications based on their workload patterns.

RAP employs an iterative, interval-based resource allocation algorithm in conjunction with a trace-based performance modeling approach such as WAM [64]. The inputs to the algorithm are as follows:

- A set of workload scenarios with *exactly* one workload scenario per application a. This set is denoted by W. The set W contains one probable workload scenario $W_{a,k}$ for each application a considered for resource allocation.
- The SLO for each application a over the planning horizon. This is denoted by SLO_a.
- The maximum number of resources $C_{max,n,t}$ available for allocation to all applications at each tier n in each resource allocation interval t.

The RAP method generates a resource allocation plan for all applications characterized by the workload scenarios in the set W so that all SLOs are satisfied or all available resources are allocated. The allocation plan generated is 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 is returned by the RAP algorithm.

Recall from Section 1.3.1 that one approach to achieve an optimal solution for the global SLO and resource allocation optimization problem described previously in Section 3.1 is to

exhaustively evaluate all resource allocation plans for all tiers of all applications over all resource allocation intervals. This exhaustive evaluation becomes computationally prohibitive with the increase in the number of applications, number of resource allocation intervals, number of tiers and number of resource flavours associated with each tier. To overcome this problem three variants of RAP are introduced in the next section.

3.3 RAP Variants

This section provides a detailed description of the three RAP variants described briefly in Section 1.3.1. Recall from Section 1.3.1 that RAP-DP is a dynamic programming based version [49] of RAP while RAP-AllApps and RAP-OneApp are two heuristic resource allocation approaches.

3.3.1 RAP-Dynamic Programming (RAP-DP)

RAP-DP starts with a basic resource allocation plan as captured previously by Equation (3.4) and then traverses a sequence of decision stages. Each *decision stage* generates an optimal resource allocation plan given that an incremental amount of resources is available for allocation. This involves using the performance modeling module to carry out the following tasks:

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

These tasks are carried out such that the lowest possible SV value as shown in Equation (3.7) is achieved at each decision stage. The next decision stage uses only this optimal plan

as a basis for further incremental resource allocation, ruling out other plans explored in the previous stage. This avoids the exhaustive evaluation of all possible combinations of resource allocation plans as described previously in Section 3.2.

RAP-DP is guaranteed to obtain an optimal solution. Recall from Section 2.6 that an optimal solution is obtained at the end of the final stage if each stage generates an optimal solution while starting from the solution obtained in the previous stage [49]. The reader can refer to Appendix B for a formulation of RAP-DP using dynamic programming concepts [49]. Appendix C also provides an example to show more details on how RAP-DP can achieve an optimal solution. In RAP-DP, each stage yields an optimal resource allocation plan and is based on the optimal plan obtained from the previous stage. It should be noted that the optimal resource allocation plan generated at each decision stage is obtained by allocating the least possible number of resources. Specifically, at each decision stage, *one* resource instance of the appropriate flavor is allocated to the identified tier in the identified resource allocation interval to the identified application. Thus an optimal resource allocation plan is obtained at each decision stage using the least possible number of resources.

There are different approaches to obtain an optimal resource allocation plan at a given decision stage. The naive approach is to evaluate all possible resource allocation plans at this decision stage. As shown in Table 3.1, 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 three variants of RAP are developed. Table 3.1 summarizes the operation and computational complexity of the three RAP variants relative to the naive approach of exhaustively enumerating all possible resource allocation plans at each decision stage.

Table 3.1 illustrates the difference between RAP-DP and the naive enumeration of all possible resource allocation plans at each decision stage. RAP-DP considers only the *bottleneck* tier of each application in all T resource allocation intervals rather than considering

Table 3.1: Computational complexity of RAP variants

At Each Decision Stage	Naive Exhaustive Enumeration At Each Decision Stage	RAP-DP	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 request response time	interval with the highest mean request response time
tiers evaluated	all	bottleneck tier	bottleneck tier	bottleneck tier
number of performance model invocations	$A^*T^*N_a$	A^*T	A	1

all tiers as per the naive approach described previously. As a result, RAP-DP decreases the computational complexity required to generate an optimal resource allocation plan at each stage relative to naively evaluating all possible resource allocation plans. 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 resource allocation interval, one resource instance of the appropriate flavour is allocated to the bottleneck tier of every application. The performance model is then invoked to compute Equation (3.7). 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 decision stage. As shown in Table 3.1, the number of performance model invocations required by RAP-DP is reduced to A^*T.

Although, the solution obtained by RAP-DP is guaranteed to be optimal, the performance model has to be invoked for all applications in all intervals to decide which application and which resource allocation interval to select for additional resource allocation at each decision stage. This increases the computation time at each decision stage because the performance model is the most time consuming part of the RAP algorithm. Alternatively, two more computationally efficient variants of RAP are proposed. These two variants trade-off optimality for computation time as will be described in the next two sections.

3.3.2 RAP-Heuristic-AllApps (RAP-AllApps)

Unlike RAP-DP which runs the performance model for all applications in all resource allocation intervals at each decision stage, RAP-AllApps runs the performance model for all applications only in the resource allocation intervals with the *highest mean response time*. This resource allocation 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 in this resource allocation interval is likely to result in the most reduction in the application's V_a with respect to other resource allocation intervals. As shown in Table 3.1, at each decision stage RAP-AllApps invokes the performance model A times. In contrast to RAP-DP, the algorithm has no dependence on the number of resource allocation intervals T.

The optimality of RAP-AllApps is guaranteed if the bottleneck tier is the same in all resource allocation intervals for a given application. If all resource allocation intervals for a certain application have the same bottleneck tier, then allocating more resources to this tier in the highest mean response time interval results in the most reduction in the application's V_a. However, if the bottleneck tier for an application is not the same in all resource allocation intervals, an optimal solution is not guaranteed. To obtain an optimal solution in this case, all resource allocation intervals have to be evaluated since the addition of different resource flavours can have different impact on the value of V_a. Results which confirm this finding will be shown later in Section 4.4.

3.3.3 RAP-Heuristic-OneApp (RAP-OneApp)

RAP-OneApp makes the simplifying assumption that targeting the application with the highest V_a likely yields the least SV value. Accordingly, at each decision stage it ranks all applications in a descending order in terms of their V_a and selects the topmost application. The allocation of one additional resource instance is explored only for this application rather than for all applications as in RAP-DP and RAP-AllApps. Specifically, similar to RAP-

AllApps, the additional resource instance is allocated for the bottleneck tier of the selected application in the resource allocation interval with the highest mean response time. As a result, RAP-OneApp invokes the performance model only once at each decision stage as shown in Table 3.1. 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. Results which evaluate the optimality and computational complexity of the three RAP variants will be shown later in Section 4.4.

In the remaining chapters of the thesis RAP-OneApp and RAP are used interchangeably to refer to the same RAP variant. If the other two variants of RAP, i.e., RAP-DP and RAP-AllApps, are involved, then all variants are explicitly called by their names. Figure 3.1 shows the pseudo-code of the RAP-OneApp algorithm or simply the RAP algorithm. As described previously in Section 3.2 that 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 $C_{max,n,t}$ available for allocation to all applications at each tier n in each resource allocation interval t.

The pseudo-code of the RAP algorithm shown in Figure 3.1 is described in more detail in the remaining part of this section. Initially each probable workload scenario $W_{a,k}$, where $k = 1$, representing an application a is allocated *one* resource instance at each tier n and for each resource allocation interval t. The WAM performance modeling technique is then invoked by RAP through the function *WAM-QNM()* as shown in Figure 3.1. This function takes as inputs the workload scenario $W_{a,k}$ 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

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
    $C_{a,n,t} = 1 \ \forall \ a \in \{1,2,\ldots, A\}$, $n \in \{1,2,\ldots, N_a\}$, $t \in \{1,2,\ldots, T\}$
    # invoke the WAM performance model for each application workload scenario $W_{a,k} \in W$, $a \in \{1,2,\ldots,A\}$
    $[R_{a,t}, R_a] = WAM\text{-}QNM\ (W_{a,k,}\ C_{a,n,t} \ \forall \ n \in \{1,2,\ldots, N_a\}) \ \forall \ t \in \{1,2,\ldots, T\}$
    $V_a = CalculateViolationPercentage(R_a, SLO_a)$
End For
#calculate total number of remaining resources at tier n over all intervals
this is used to stop the algorithm once all remaining resources are allocated
For $n = 1$ ***to*** N_a
    $RemainResourcesTier(n) = C_{max,n,t} * T - (\sum C_{a,n,t} \ \forall \ a \in \{1,2,\ldots, A\}, t \in \{1,2,\ldots, T\})$
End For
While $(RemainResourcesTier(n) > 0 \ \forall \ n \in \{1,2,\ldots, N_a\})$
        $[a_{max}, MaxViolationPercentage_{,}] = GetAppwithMaxViolationPerentage(V_a \forall a \in \{1,2,\ldots, A\})$
        # if all clustered workload scenarios satisfy their violation percentages
        If $MaxViolationPercentage_{,} == 0$
         Return $C_{a,n,t} \ \forall \ a \in \{1,2,\ldots, A\}$, $n \in \{1,2,\ldots, N_a\}$, $t \in \{1,2,\ldots, T\}$, $V_a \ \forall \ a \in \{1,2,\ldots, A\}$
         Where $V_a = 0 \forall \ a \in \{1,2,\ldots, A\}$
        End If
        $t_{max} = GetFreeIntervalwithMaxRmean(a_{max}, (R_{a,t} \ for \ a = a_{max}, \ \forall \ t \in \{1,2,\ldots, T\}))$
     $n_{bottleneck} = GetBottleneckTier(a_{max}, t_{max})$
    # allocate one additional resource instance to tier n of application a_{max} in interval t_{max}
    $\Delta C_{a,n,t} = 1 \ for \ a = a_{max}, \ n \ n_{bottleneck}, \ t = t_{max}$
      $C_{a,n,t} = C_{a,n,t} + \Delta C_{a,n,t} \ \ for \ a = a_{max}, \ n = n_{bottleneck}, t = t_{max}$
    $RemainResourcesTier(n_{bottleneck}) = RemainResourcesTier(n_{bottleneck}) - \Delta C_{a,n,t}$
    #re-invoke The WAM performance model for the application a_{max} only
    $[R_{a,t}, R_a] = WAM\text{-}QNM\ (W_{a,k,}\ C_{a,n,t} \ \forall \ n \in \{1,2,\ldots, N_e\}) \ \ for \ e = a_{max}, \forall \ t \in \{1,2,\ldots, T\},$
    $V_a = CalculateViolationPercentage(R_a, SLO_a) \ for \ a = a_{max}$
End While
Return $C_{a,n,t} \ \forall \ a \in \{1,2,\ldots, A\}$, $n \in \{1,2,\ldots, N_a\}$, $t \in \{1,2,\ldots, T\}$, $V_a \ \forall \ a \in \{1,2,\ldots, A\}$

Figure 3.1: RAP algorithm

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 is allocated to application a_{max} in tier $n_{bottleneck}$ at resource allocation interval t_{max}. The algorithm terminates when either all the available resources are allocated in all resource

allocation intervals or the maximum V_a is equal to zero which means that all applications have achieved their SLOs.

As described previously in Section 1.3.1, RAP can rely on any performance model that can predict an application's SLO under a given resource allocation plan. Instead of WAM-QNM, RAP can invoke MVA-QNM. RAP invokes MVA-QNM by calling the function *MVA-QNM()*. This function takes as inputs the following parameters:

- The number of resources assigned to each tier n of application a in each time interval t.

- A vector of mean number of concurrent sessions, i.e., session populations, observed for application a per resource allocation interval t as shown in:

$$\begin{bmatrix} MS_{a,1} & MS_{a,2} & ... & MS_{a,t} & ... & MS_{a,T} \end{bmatrix} \tag{3.8}$$

  This vector can be approximated from the application's workload scenario using the method described by Krishnamurthy *et al.* [64].

- A vector of mean session think time values for application a per resource allocation interval as shown in:

$$\begin{bmatrix} Z_{a,1} & Z_{a,2} & ... & Z_{a,t} & ... & Z_{a,T} \end{bmatrix} \tag{3.9}$$

- The following matrix of mean service demands of each application a for various N_a tiers over each of the T resource allocation intervals

$$\begin{bmatrix} D_{a,1,1} & D_{a,1,2} & ... & D_{a,1,t} & ... & D_{a,1,T} \\ . & & & & & \\ . & & & & & \\ . & & & & & \\ D_{a,N_a,1} & D_{a,N_a,2} & ... & D_{a,N_a,t} & ... & D_{a,N_a,T} \end{bmatrix} \tag{3.10}$$

MVA-QNM is used to solve a single hybrid class QNM characterized by these inputs. The function *MVA-QNM()* returns the same outputs returned by the function *WAM-QNM()* as described previously.

3.4 Burstiness-Agnostic SLP Approaches

This section describes two SLP approaches in which workload variability and burstiness are not taken into account. Experiments that show the operation of these two approaches and results that compare them with RAP will be shown later in Section 4.6.

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. In other words, the whole approach considers a resource allocation interval of the same size as the planning horizon. 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. Simulation results that show the operation of the whole approach will be shown later in Section 4.6.

The second burstiness-agnostic 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 allocated to an application tier at each decision stage. In RAP the candidate resource allocation interval is the resource allocation interval with the highest mean response time, i.e., the most heavily loaded resource allocation interval, to account for both workload variability and 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 resource instances are allocated for the resource allocation interval which comes in sequence starting from the first resource allocation interval in the planning horizon. After each decision stage the applications are re-ranked in terms of their V_a values to select the next candidate application. An additional resource instance is allocated to an application tier in the resource allocation interval which comes next in the chronological sequence for this application provided that the selected resource allocation interval has free resources. This process is repeated until all application SLOs are satisfied or all resources in the cloud are exhausted for all resource allocation intervals. Section 4.6 shows results which describe the operation of the basic interval approach.

3.5 Summary

This chapter provides a detailed description of the RAP method. Firstly, it describes the global resource allocation optimization problem which is solved by the RAP method. Secondly, a detailed description of the RAP method and its variants are presented. Finally, two burstiness-agnostic SLP approaches are described. These approaches will be compared with RAP as will be shown later in Section 4.6. The next chapter discusses RAP evaluation results.

Chapter 4

RAP Evaluation

This Chapter discusses results of evaluating the RAP method described previously in Chapter 3. Section 4.1 describes an approach used in the thesis to characterize the workload scenarios which are considered in the experiments. Section 4.2 provides a description of the simulation setup used to obtain the results. Section 4.3 presents results of comparing MVA-QNM and WAM-QNM performance modeling techniques for SLP. Section 4.4 shows results which characterize the optimality of the three RAP variants described previously in Section 3.3. Section 4.5 shows a sensitivity analysis of the three RAP variants to the degree of similarity in resource demands between application tiers and the degree of homogeneity in resource scaling among application tiers. Section 4.6 shows results of comparing RAP with the two burstiness-agnostic approaches for SLP described previously in Section 3.4. Section 4.7 presents results of comparing RAP with SLP approaches that rely on utilization of system resources as described previously in Section 1.1. The flexibility of RAP in handling different SLOs is shown in Section 4.8. Finally, Section 4.9 presents results that show the improvement achieved in the performance of RAP-DP and RAP-AllApps by exploiting parallelism within these algorithms.

4.1 Using Traces to Characterize Application Workloads

This section introduces a formal approach for succinctly characterizing an application's workload scenario. As described previously in Section 1.3.1, each application a is characterized by a workload scenario consisting of a collection of traces. An arrival process model is used to characterize the session trace associated with a workload scenario. Similarly, a service process model is used to describe the resource usage trace of a workload scenario. As men-

tioned previously in Section 3.1, for each application a there are many probable workload scenarios. Each probable workload scenario $W_{a,k}$ has a certain probability of occurrence, denoted by $P_{a,k}$, where $k \in \{1,2,...L_a\}$, with the sum of all these probabilities from k=1 to L_a is equal to 1. Similar to the previous chapter, L_a is set to 1 in this chapter.

The arrival process model of the probable workload scenario $W_{a,k}$ of application a over the planning horizon is denoted by $AM_{a,k}$. $AM_{a,k}$ is characterized by the set of parameters $\{S_{a,k}, F_{a,k}, Z_{a,k}, \lambda_{a,k}, SCV_{a,k}, I_{a,k}\}$. This characterization is based on the approach described by Casale *et al.* [41] that accounts for both variability and burstiness in the arrival of user sessions to an application system. The parameter $S_{a,k}$ is the number of user sessions in the workload scenario. $F_{a,k}$ and $Z_{a,k}$ specify the distributions of the number of requests per session and the think time between session requests, respectively. $\lambda_{a,k}$ denotes the mean session arrival rate. The variability of the session inter-arrival time distribution is captured by $SCV_{a,k}$ which is the SCV of inter-arrival time between chronologically successive sessions in the trace. Finally, the arrival burstiness is summarized by the parameter $I_{a,k}$ which is the index of dispersion of session inter-arrival times as described previously in Section 2.2.

The application's service process model for the probable workload scenario $W_{a,k}$ is denoted by $SM_{a,k}$. $SM_{a,k}$ captures the service demands placed on application system resources, e.g., CPUs and disks. As mentioned previous in Section 3.1, each application a has a number of application tiers N_a. The application's service process model $SM_{a,k}$ is characterized by the set $D_{a,k,n}$ representing the mean service demands at various tiers, i.e., $n \in \{1,2,...N_a\}$.

4.2 Simulation Setup

To evaluate RAP a discrete event simulation model for each application is used. In this simulation model each application is represented by a queuing network consisting of two resources representing the typical web and database tiers of an enterprise application. Figure 4.1 shows the same queuing network shown previously in Figure 2.2 and described in Section

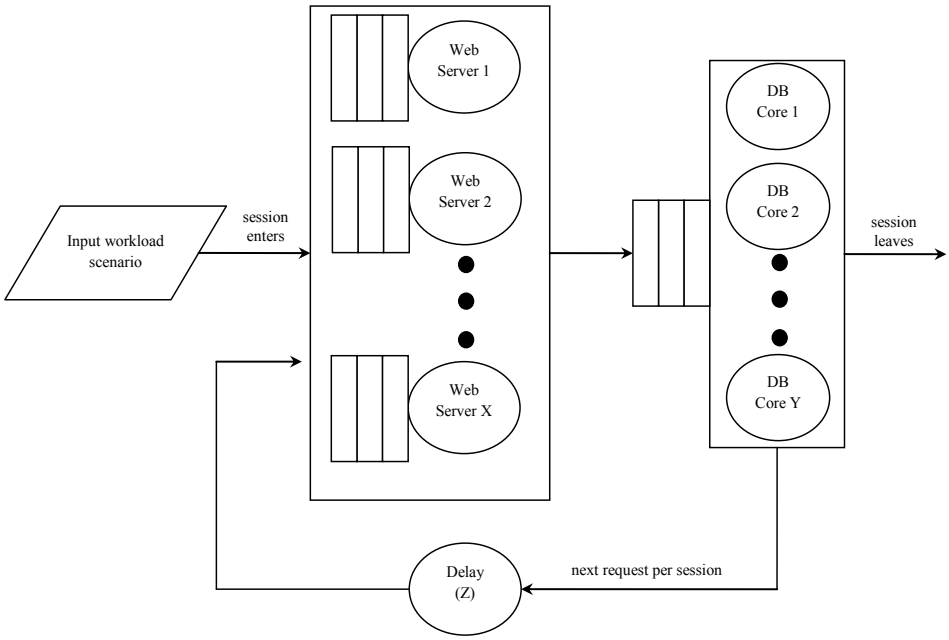


Figure 4.1: Queuing network representing a two tier application

2.3.1. This figure is reproduced in this section for completeness. As shown in Figure 4.1, similar to best practices used in enterprise applications, the web and database tiers employ different types 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 during the SLP process to achieve application SLO objectives. The single core instance is similar to the standard small instance type used in Amazon EC2 while the multi-core instance is similar to Amazon EC2 instance types characterized by a higher number of cores such as the standard large and extra large instances [3]. The simulation model takes into account the processing power of each instance without considering memory, storage or I/O issues. These factors will be considered in future work.

As described previously in Section 2.3.1, the application represented by the queuing network of Figure 4.1 is subjected to an input workload scenario consisting of a trace of customer sessions. The experiments use a variety of synthetic traces with characteristics as

Table 4.1: Values of the parameters used in the experiments

Parameter	Value
Planning horizon	$\sim$ 4 and 8 hours
Resource allocation interval	1 hour
$S_{a,k}$	{75,000, 85,000, 100,000}
$F_{a,k}$	empirical distribution with mean 9.4 request/session [27]
$Z_{a,k}$	empirical distribution with mean 40 s [27]
$\lambda_{a,k}$	3.33, 2.86, 2.5 session/s
$SCV_{a,k}$	1, 3, 4, 5
$I_{a,k}$	{1, 100, 500, 1000, 10,000}
$D_{a,k,1}$	exponential distribution with mean 20 ms
$D_{a,k,2}$	exponential distribution with mean 10 ms

shown in Table 4.1 to assess the behaviour of RAP under varying levels of arrival variability and burstiness. The parameters shown in Table 4.1 were described previously in Section 4.1. The service demands at the web and database tiers for all experiments are fixed as per Table 4.1. The session length and think time distributions are chosen to match empirical distributions observed at a real web-based application system [27]. Experiments presented in the thesis employ a planning horizon of 4 and 8 hours and a resource allocation granularity of 1 hour. All applications in the experiments have an application SLO defined as a target mean response time that is twice the demand at the bottleneck resource. Practitioners often use such an SLO as an indirect measure to indicate the queuing that can be tolerated at the bottleneck resource.

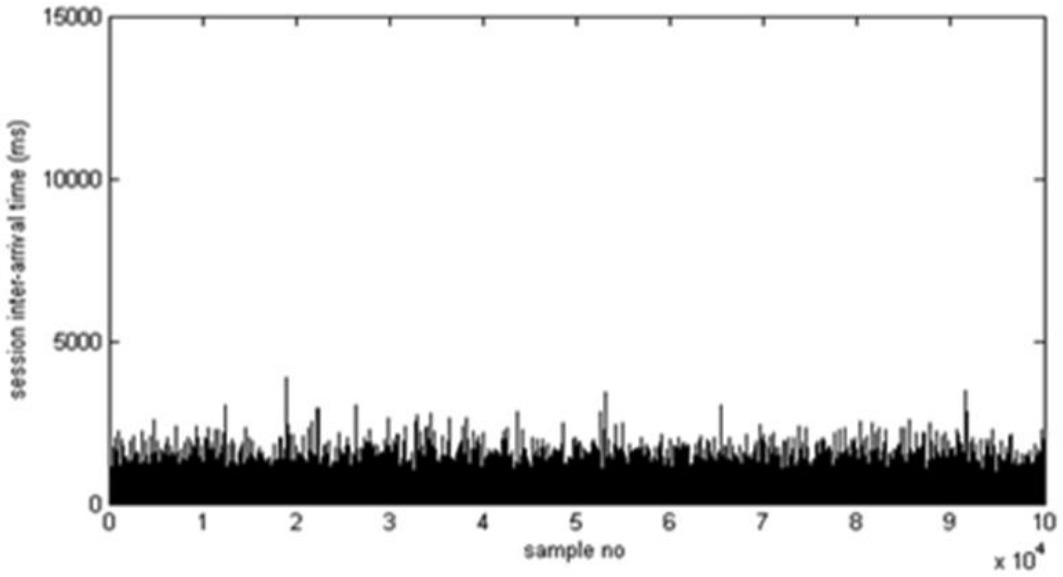
As mentioned previously in Section 2.3, both MVA-QNM and WAM-QNM techniques are used to model the simulated cloud environment. Standard single server residence time expressions [65] are used to model the single-core web tier. Multi-server residence time expressions developed by Rolia [86] are used for modeling the multi-core database tier. It should be noted that analytically modeling this system is a challenge when workload traces exhibit characteristics such as burstiness. The impact of model inaccuracies can have on the SLP process is characterized in Section 4.3.

4.3 Comparing MVA-QNM and WAM-QNM for SLP

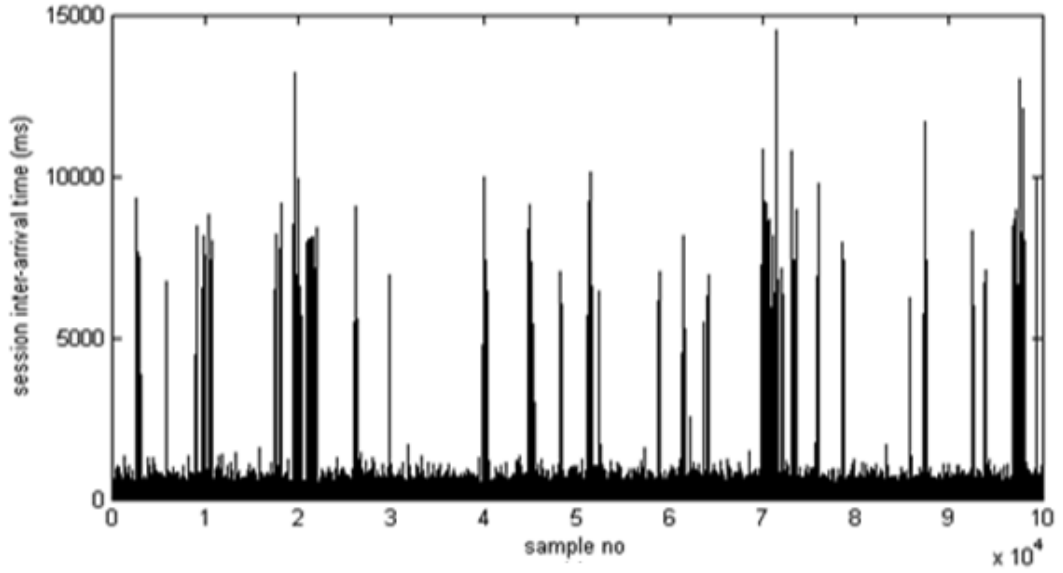
In this section MVA-QNM and WAM-QNM are used to predict the mean response times for a number of applications whose workloads exhibit the same mean session arrival rate but progressively higher degrees of session arrival variability and burstiness. The degree of variability and burstiness is controlled by using different values for $SCV_{a,k}$ and $I_{a,k}$, respectively, as described previously in Section 4.1. Figure 4.2 shows an example of two typical session arrival traces. The *exponential* trace shown in Figure 4.2a is non-bursty with session inter-arrival times following an exponential distribution. In contrast, a *variable* and *bursty* trace shown in Figure 4.2b is obtained by specifying $SCV_{a,k}$=5 and $I_{a,k}$= 1000. Both of these traces have the same mean session arrival rate i.e., $\lambda_{a,k}$ = 3.33 sess/s, over the planning horizon and are part of the workloads used in evaluating RAP. From the figure, one can observe that the highly variable and bursty workload has more pronounced crests and troughs when compared to the exponential workload.

The mean response time predictions of WAM-QNM and MVA-QNM are compared for various degrees of variability, i.e., exponential and $SCV_{a,k}$ = 1, 3, 4 and 5, and various degrees of burstiness, i.e., $I_{a,k}$= 100, 1000 and 10,000. Actual mean response times are obtained by using the discrete event simulation model outlined previously in Section 4.2. These mean response times are compared with their corresponding predicted mean response times from both modeling approaches.

Figure 4.3 shows the average prediction error over an eight-hour planning horizon for both approaches for different session arrival processes. Table 4.2 lists the $SCV_{a,k}$ and $I_{a,k}$ values used to obtain each session arrival process in Figure 4.3. From the figure, for the non-bursty exponential workload both techniques yield very accurate predictions. However, the MVA-QNM technique yields very poor predictions with increasing variability and burstiness for most of the session arrival processes shown. For example, for a session trace with medium variability and high burstiness, i.e., $SCV_{a,k}$ = 3 and $I_{a,k}$ = 1000, MVA-QNM yields an



(a) exponential



(b) variable and bursty ($SCV_{a,k} = 5$, $I_{a,k} = 1000$)

Figure 4.2: Two sets of arrival instances with same mean session inter-arrival time = 300 ms

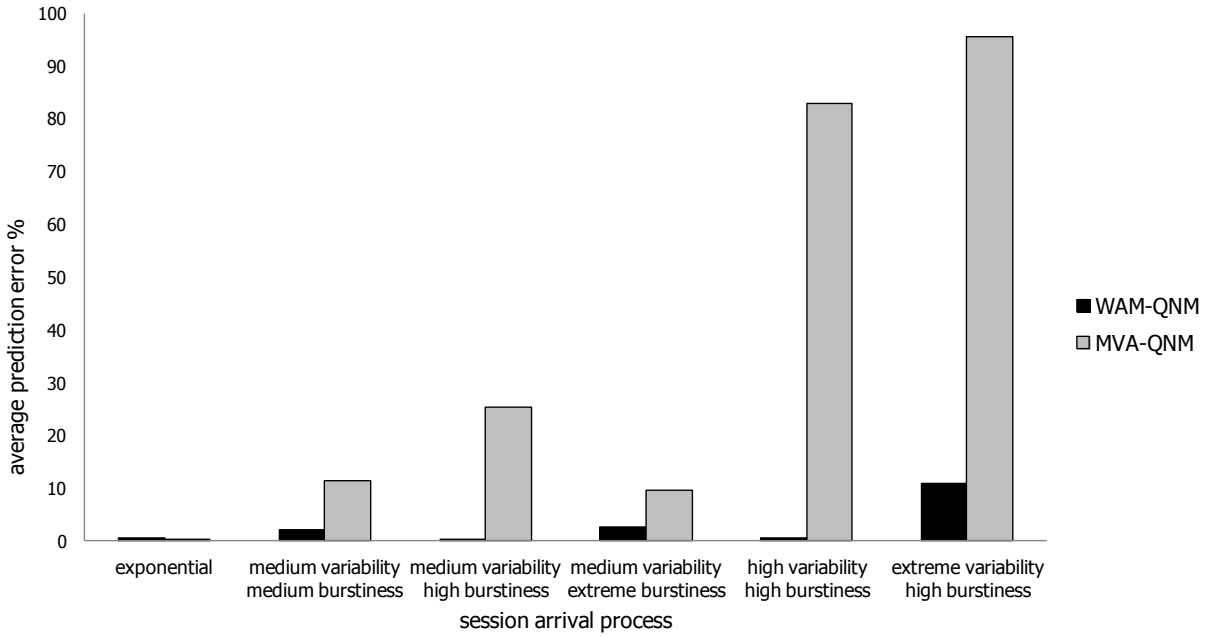


Figure 4.3: Mean response time prediction error of WAM-QNM and MVA-QNM

accuracy of 25% while WAM-QNM's predictions closely match the actual mean response time obtained through simulation. However, it can be noticed from the figure that for the session trace with medium variability and extreme burstiness, i.e., $SCV_{a,k} = 3$ and $I_{a,k} =$ 10,000, MVA-QNM yields an average prediction error of only 10%. This is because for this session trace the extremely bursty session arrivals occur only in some resource allocation intervals while other resource allocation intervals experience much lower level of burstiness. Therefore, MVA-QNM was able to obtain low average prediction error over the planing horizon. This can also be seen in the results shown later in Figure 4.4d. In summary, the results shown in 4.3 broaden the findings reported by Krishnamurthy *et al.* on a TPC-W web application system [64].

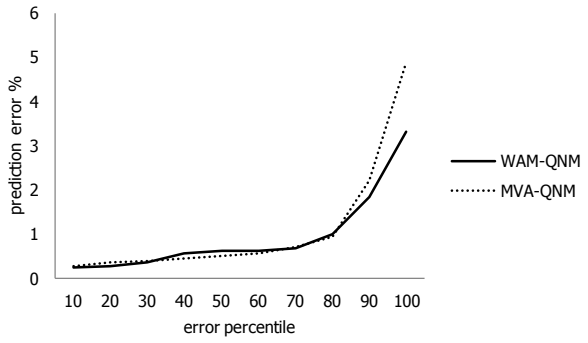
Figure 4.4 shows error percentiles for mean response times predicted by both WAM-QNM and MVA-QNM over the same eight-hour planning horizon shown in Figure 4.3. As shown in Figure 4.4, the prediction accuracies of both WAM-QNM and MVA-QNM are affected by increasing the degree of variability and burstiness in session arrivals, however,

Table 4.2: Parameters of the session arrival processes shown in Figure 4.3

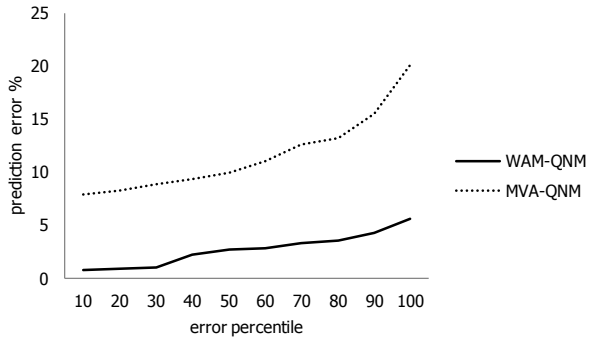
Session Arrival Process	$S_{a,k}$	$I_{a,k}$
Exponential	1	1
Medium variability, medium burstiness	3	100
Medium variability, high burstiness	3	1000
Medium variability, extreme burstiness	3	10,000
High variability, high burstiness	4	1000
Extreme variability, high burstiness	5	1000

the effect is more significant on MVA-QNM than on WAM-QNM. For example, for the non-bursty exponential session arrival process shown in Figure 4.4a the maximum prediction error over all resource allocation intervals for MVA-QNM is only 5% while the maximum error percentile for the session arrival process shown in Figure 4.4f, which is characterized by extreme variability and high burstiness, is close to 100%. On the other hand, the error percentiles of WAM-QNM are much lower than their corresponding error percentiles of MVA-QNM for all session arrival processes. Specifically, the maximum per-interval prediction error of WAM-QNM for the session arrival processes shown in Figures 4.4a to 4.4d does not exceed 20%. However, WAM-QNM gives poor prediction errors for the session arrival processes characterized by high variability as shown in Figures 4.4e and 4.4f. The reason for this is that WAM-QNM depends on the predictions obtained by MVA-QNM as described previously in Section 2.3. Therefore, the accuracy of MVA-QNM affects that of WAM-QNM. The prediction accuracy of WAM-QNM can be improved by relying on other advanced performance modeling techniques that are capable of predicting application performance more accurately under high session arrival variability. Addressing this issue is left for future work.

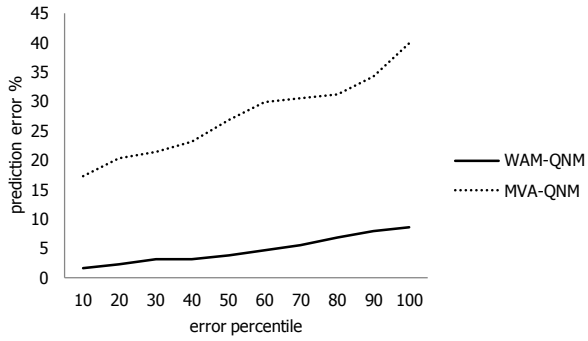
In summary, the results shown in Figures 4.3 and 4.4 suggest that using WAM-QNM in conjunction with RAP is likely to give more accurate resource allocation estimates than MVA-QNM when the application workload scenarios are bursty in nature. As a result, WAM-QNM is used in the experiments which involve bursty workloads in the rest of the



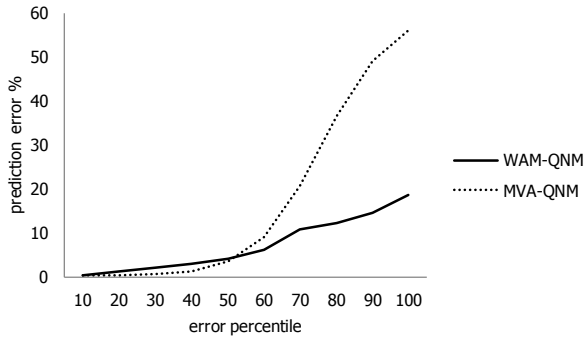
(a) exponential



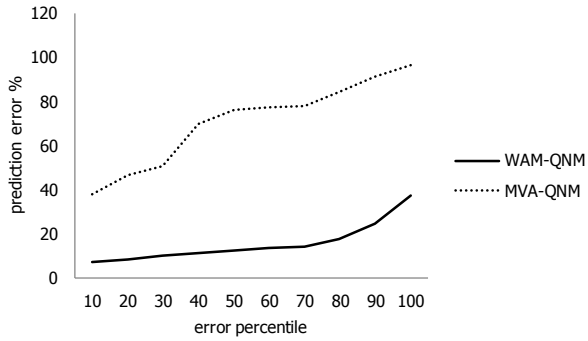
(b) medium varaibility and medium burstiness



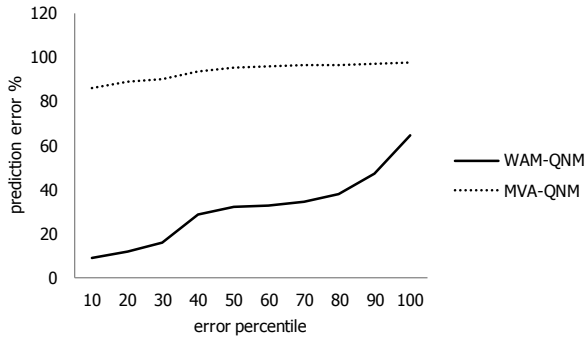
(c) medium varaibility and high burstiness



(d) medium varaibility and extreme burstiness



(e) high varaibility and high burstiness



(f) extreme varaibility and high burstiness

Figure 4.4: Response time prediction error percentiles of WAM-QNM and MVA-QNM

thesis. In this work it is assumed that WAM-QNM gives correct predictions for all session arrival processes used in the experiments. Although this assumption might not be valid for the extremely bursty session arrival processes as shown in the previous paragraph, the development of more accurate performance models to deal with such workloads is beyond the scope of this work.

4.4 Characterizing Optimality of RAP Variants

This section focuses on characterizing the optimality of the three RAP variants. As described previously in Section 3.3 there are three variants of RAP, i.e., RAP-DP and RAP-AllApps and RAP-OneApp. These variants are compared against the exhaustive enumeration of all possible solutions.

Two controlled experiments are preformed to show the degree of optimality achieved by the three variants and their limitations. In the first experiment four different applications subjected to exponential session arrivals are analyzed over a planning horizon of 4 hours with a resource allocation interval of 1 hour. 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 in the cloud is set to 7. These settings allow the exhaustive enumeration of all possible resource allocation plans for these applications. Since MVA-QNM is effective for exponential workloads, mean response times were predicted for each application for each of these resource allocation plans using this modeling technique. Based on these predictions, the SV value defined previously in Equation (3.2) divided by the number of applications is calculated for each resource allocation plan. This value represents the mean SLO violation percentage for each resource allocation plan. Finally, each of the three RAP variants is used in conjunction with MVA-QNM to obtain a resource allocation plan for this experiment.

The second experiment explores more bursty workloads. The four applications are characterized by more session arrival variability, i.e.,$SCV_{a,k}$= 3, and progressively higher degrees of session arrival burstiness, i.e., $I_{a,k}$ = 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. WAM-QNM is used as the performance modeling technique instead of MVA-QNM because it is more accurate for bursty workloads as shown previously in Section 4.3.

Table 4.3 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 a small-scale SLP exercises.

Figure 4.5 shows the optimality of the solutions obtained by the three RAP variants. It should be noted that the optimality of the RAP variants can be affected by the accuracy of the performance model used in the experiments. Therefore, Figure 4.5 characterizes the optimality of the RAP variants given the assumption stated previously in Section 4.3 that WAM-QNM gives correct performance predictions for the applications used in these experiments.

Figure 4.5 compares the mean SLO 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 example, in Figure 4.5a an initial resource allocation plan is generated at decision stage 0 by allocating one web resource instance to each of the four applications over each of the 4

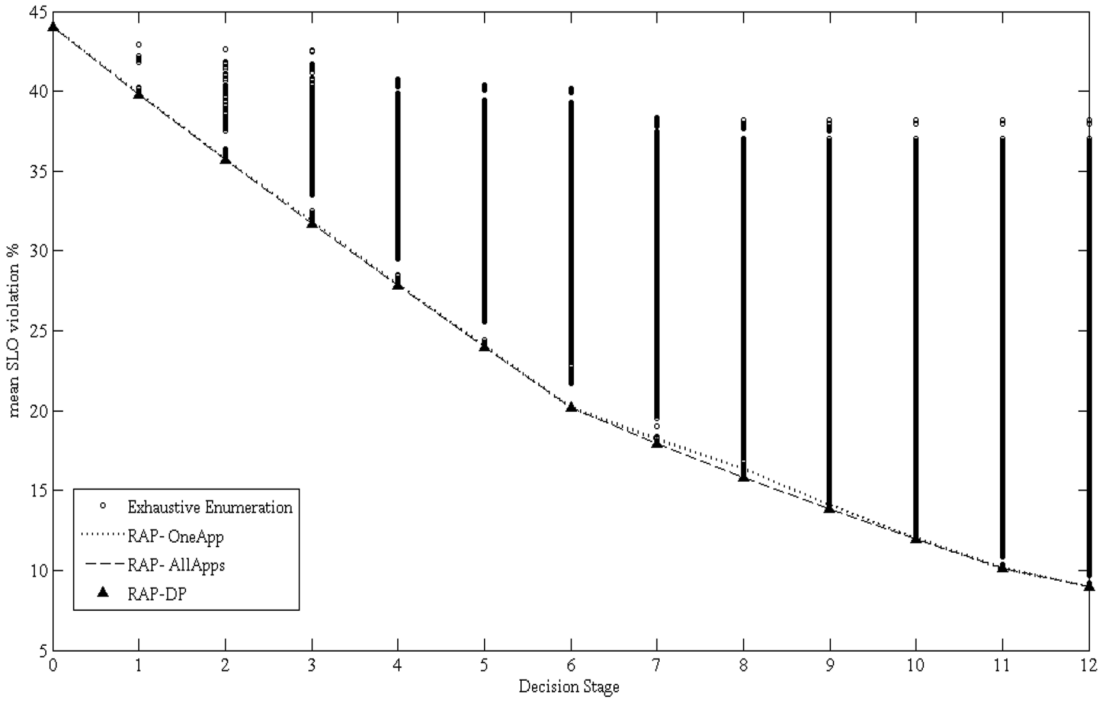
Table 4.3: Statistics of the results obtained in Figure 4.5

	Exhaustive enumeration of all resource allocation plans	**RAP-DP**	**RAP-AllApps**	**RAP-OneApp**
performance model invocations per decision stage	N/A	16	4	1
number of solutions explored in Figure 4.5a	1.5 million	192	48	12
number of solutions explored in Figure 4.5b	50,000	128	32	8

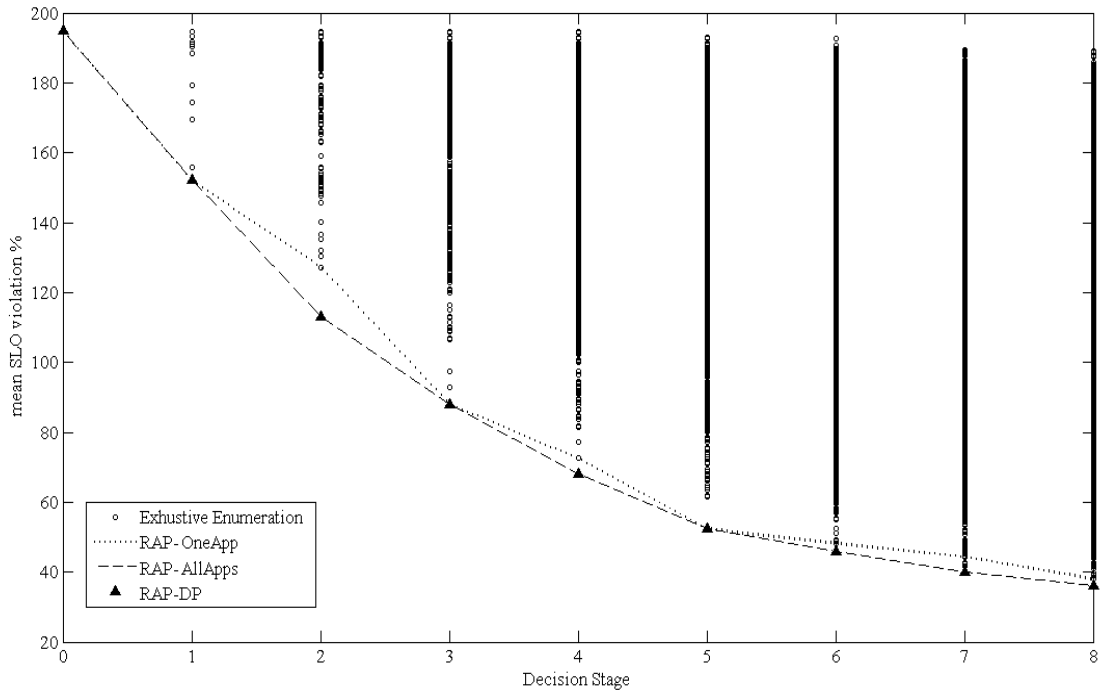
resource allocation intervals leading to an *aggregate* of 16 web resource instances allocated in this resource allocation plan. For each subsequent decision stage i the black dots represent the mean SLO violation percentages obtained for the various possibilities of allocating i additional web server instances to the initial resource allocation plan generated at decision stage 0.

The total number of decision stages shown in Figure 4.5 can be calculated by subtracting the aggregate number of web resource instances allocated to all applications over all resource allocation intervals at decision stage 0 from the maximum number of web resource instances available over all resource allocation intervals. For example in Figure 4.5a a total of 16 web server instances are initially allocated over all resource allocation intervals and a maximum of 28 web server instances are available for allocation over all resource allocation intervals. This results in a total of 12 decision stages. It should be noted that the legends used to represent the results shown in Figure 4.5 for RAP-OneAPP and RAP-AllApps are selected to be straight lines not discrete points although there is no meaning of intermediate results that can exist between decision stages. The reason for this selection is to clearly illustrate the differences in accuracies between the RAP variants.

Figure 4.5 provides some insights on the behaviour of the RAP variants. Figure 4.5a shows the results obtained for application workloads with exponential session arrivals while Figure 4.5b shows workloads with variable and bursty session arrivals. It is observed in Figure



(a) workloads with exponential session arrivals



(b) workloads with variable and bursty session arrivals

Figure 4.5: Optimality of RAP variants

4.5a with exponential workloads that both RAP-DP 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 SV value has also the highest V_a value. The difference in behaviour between RAP-DP and RAP-AllApps on one side and RAP-OneApp on the other side can be observed more clearly in Figure 4.5b with bursty workloads. However, the RAP-OneApp's solution is still very close to the solutions obtained by the other two variants. Appendix C describes a more detailed analysis of the results shown in Figure 4.5b to show how RAP-DP achieves an optimal solution in this experiment.

Figure 4.5 characterizes the optimality of the resource allocation plans obtained at each decision stage by the three RAP variants with respect to the SLO violation percentage only. However, the figure does not show the optimality of the solutions obtained by the RAP variants with respect to the aggregate number of web server instances allocated in each plan. Analysis of the results obtained from this experiment show that the optimal plans obtained by RAP-DP at the different decision stages have the same aggregate number of web server instances allocated when compared to the optimal plan obtained using exhaustive enumeration of all possible plans at the corresponding decision stages. This is because, as described previously in Section 3.3.1, RAP-DP allocates exactly one extra resource instance to the bottleneck tier of one of the applications in one resource allocation interval. Therefore, RAP-DP allocates the least possible number of resources at each decision stage to achieve the most reduction in the SLO violation percentage.

It can be noticed in Figure 4.5 that there is no difference between the solutions obtained by RAP-DP 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-DP and RAP-AllApps, another experiment is conducted with the same settings

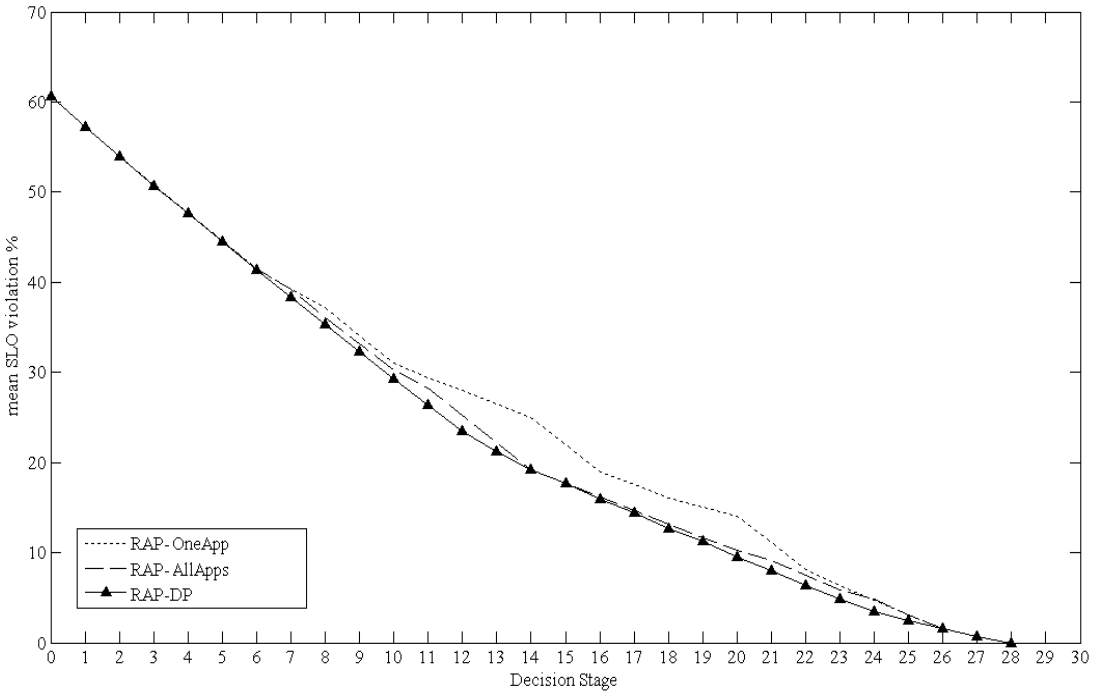


Figure 4.6: Solutions obtained by RAP variants with workloads having different bottleneck tiers over the planning horizon

used in the first experiment whose results are shown in Figure 4.5a. 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 demand is increased from 10 to 18 ms to be close to the mean web demand of 20 ms.

Figure 4.6 shows slight differences in the solutions obtained by RAP-AllApps from the optimal solutions obtained by RAP-DP. 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 optimlaity 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. This is opposed to RAP-DP which evaluates the performance of all applications in all resource allocation intervals at each decision stage as described previously in Section 3.3.2. Section 4.5 shows more detailed analysis of the workload characteristics which cause the solutions obtained by RAP-AllApps to differ from the corresponding solutions obtained by RAP-DP.

Figure 4.6 shows that the three RAP variants are 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.

Table 4.3 shows that RAP-OneApp reduces the number of solutions explored, i.e., performance model invocations, relative to RAP-DP and RAP-AllApps at each decision stage significantly. For example, in the experiment shown in Figure 4.5a RAP-OneApp reduces the number of solutions explored per decision stage by 94% and 75% relative to RAP-DP and RAP-AllApps, respectively. Consequently, RAP-OneApp is better suited for analyses involving a large number of applications without significantly affecting the optimality of the solutions obtained.

4.5 Sensitivity Analysis of RAP Variants

This section shows results which analyze the sensitivity of the three RAP variants to two factors namely, the degree of similarity in resource demands between application tiers and the degree of homogeneity in resource scaling among application tiers. Both factors will be described in more detail in the ensuing paragraphs.

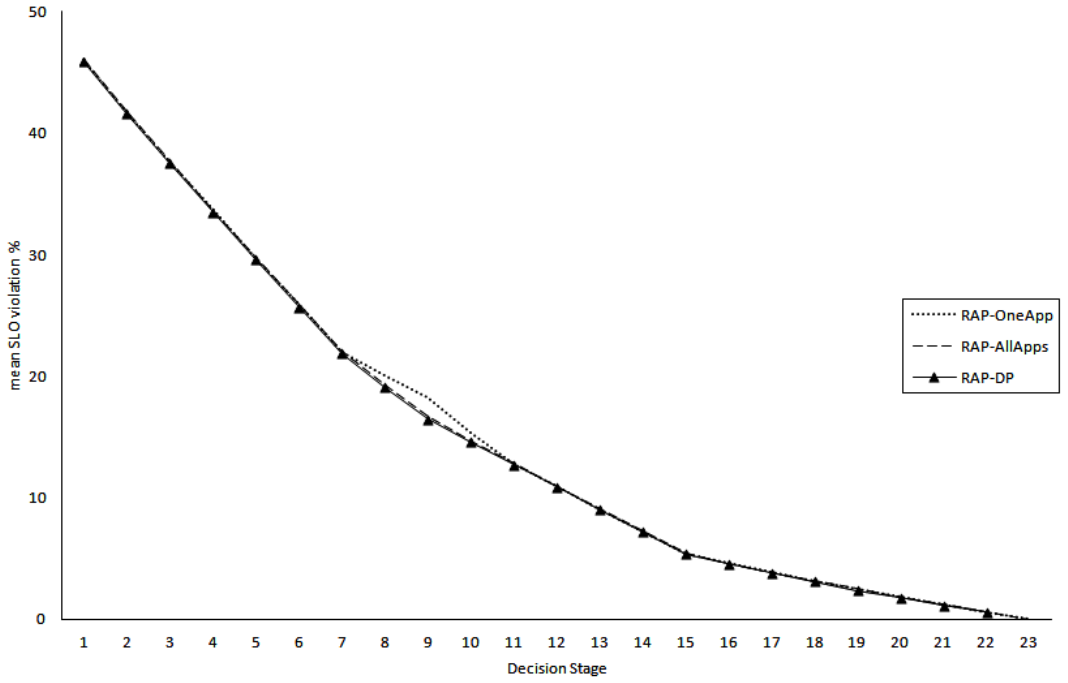
The sensitivity of the RAP variants to the degree of similarity in resource demands between application tiers is explored first. An experiment is conducted where four different applications subjected to exponential session arrivals are analyzed over a planning horizon

of 4 hours with a resource allocation interval of 1 hour. The maximum number of web server instances and database instance cores per resource allocation interval are set to very high values which are sufficient to allocate enough resources to all applications so that they can satisfy their SLO requirements. In this experiment the mean demand of the web tier is kept at 20 ms while the mean demand of the database tier is varied from 10 to 18 ms. In this way, the degree of demand similarity between the two tiers is varied. Finally, each of the three RAP variants is used in conjunction with WAM-QNM to obtain a resource allocation plan for this experiment.

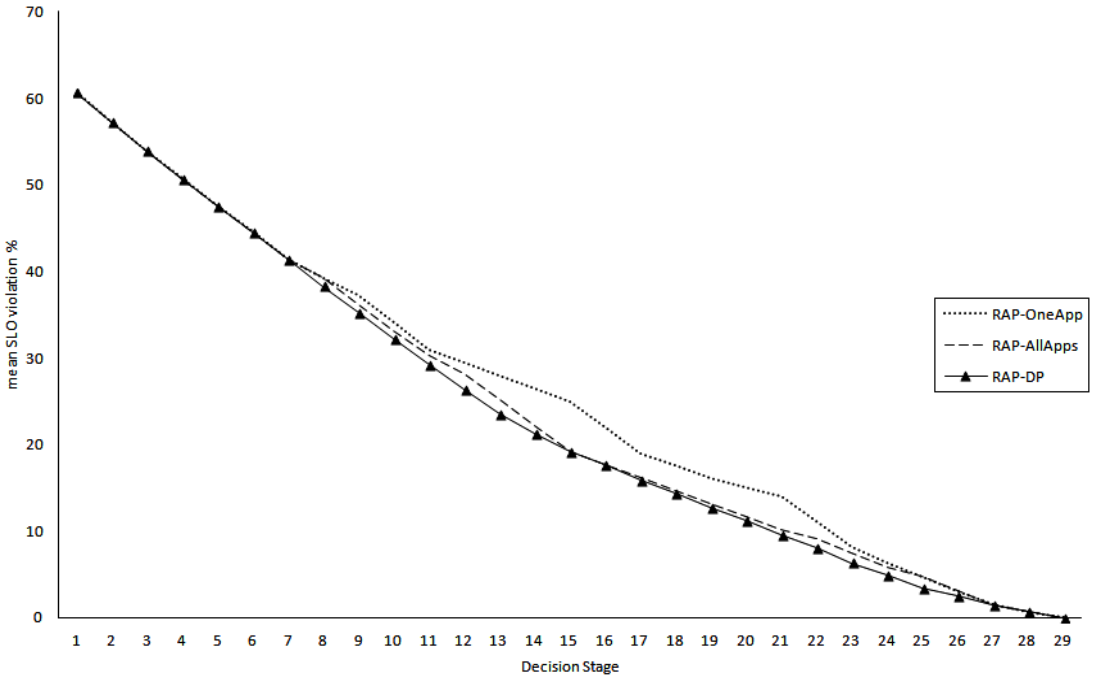
Figure 4.7 shows the sensitivity of the three RAP variants to the similarity in demands between application tiers when subjected to workloads with exponential session arrivals. Figure 4.7a compares the mean SLO violation percentages of the solutions obtained by each of the three RAP variants when the mean demands of the two tiers are significantly different while Figure 4.7b compares the mean SLO violation percentages of the solutions obtained by the three RAP variants when the mean demands of the two tiers are more similar. It should be noted that Figure 4.7b reproduces the same results shown previously in Figure 4.6 for comparison with the results shown in Figure 4.7a and with the results shown later in Figures 4.8 and 4.9.

Figure 4.7a shows that the solutions obtained by the optimal RAP-DP and RAP-AllApps are the same in all decision stages. This is because the web mean demand is twice as much as the database mean demand which makes the web tier always the bottleneck tier for all workloads considered in the experiment in all resource allocation intervals at all decision stages. The solutions obtained by RAP-OneApp are also the same as those obtained by RAP-DP except for the decision stages 8, 9 and 10.

Figure 4.7b shows more differences in the solutions obtained by RAP-AllApps from those obtained by RAP-DP at some decision stages when compared with the results shown in Figure 4.7a. The reason for this is that the similarity of the demands between the web and



(a) mean demands of web and database tiers are significantly different



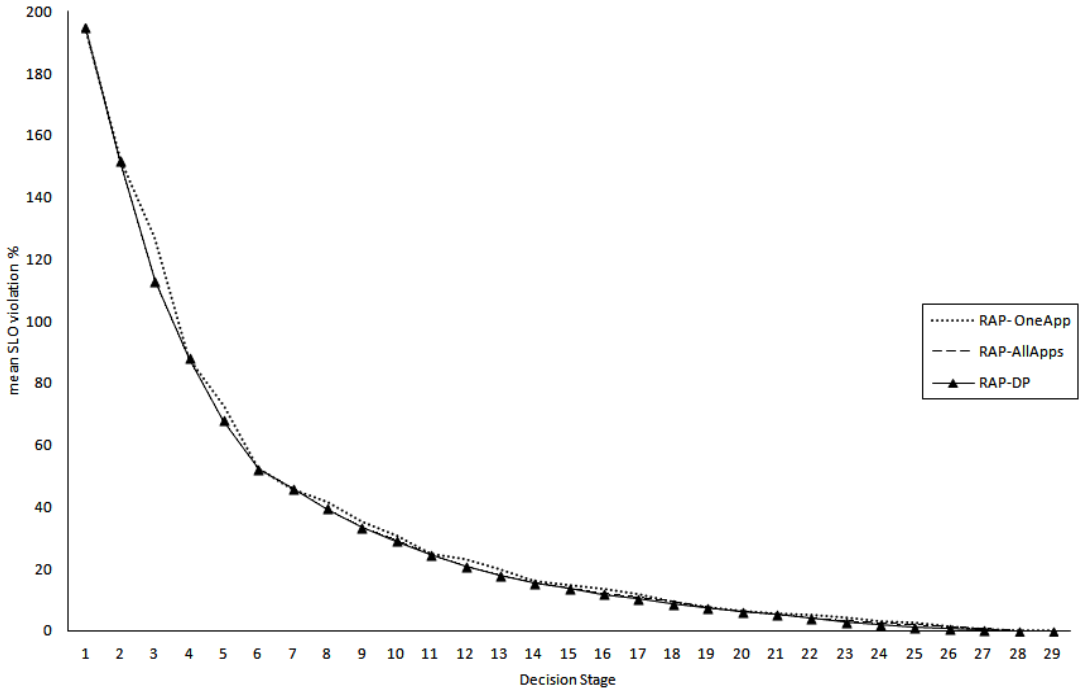
(b) mean demands of web and database tiers are smiliar

Figure 4.7: Sensitivity of RAP variants to similarity resource demands between application tiers when subjected to workloads with exponential session arrivals

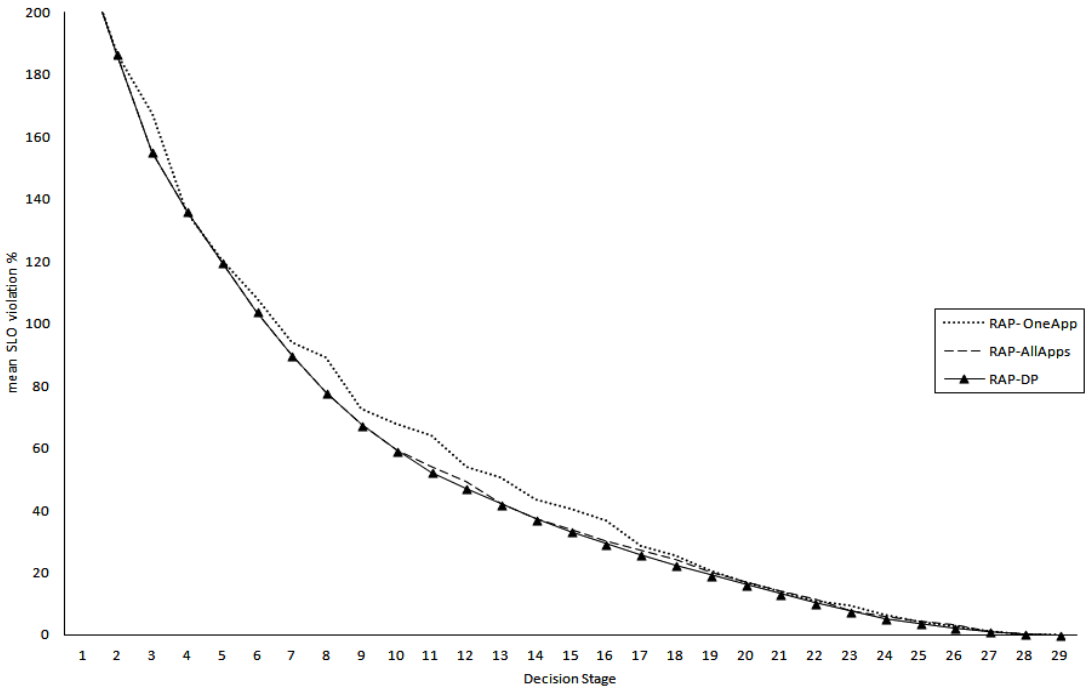
database tiers makes the workloads change their bottleneck tiers in some resource allocation intervals at some decision stages. This affects the optimlaity of the solutions obtained by RAP-AllApps as described previously in Section 3.3.2. In comparison with the results shown in Figure 4.7a, Figure 4.8b shows more differences in the solutions obtained by RAP-OneApp from those obtained by RAP-DP. As described previously in Section 3.3.3, RAP-OneApp makes the simplifying assumption that targeting the application with the highest SLO violation percentage likely yields the highest reduction in mean SLO violation percentage over all applications. Changing the bottleneck tier in some resource allocation intervals at some decision stages causes RAP-OneApp to not obtain the optimal solution at these decision stages.

Figure 4.8 shows the same results shown in Figure 4.7 for four workloads characterized by more session arrival variability, i.e.,$SCV_{a,k}$= 3, and progressively higher degrees of session arrival burstiness, i.e., $I_{a,k}$ = 100, 1000 and 10,000. In general, the same trends for the three RAP variants shown in Figure 4.7 for exponential session arrivals are confirmed in Figure 4.8 for bursty session arrivals. However, for the bursty workloads shown in Figure 4.8a the solutions obtained by RAP-OneApp are deviated slightly more from those obtained by RAP-DP when compared with corresponding results for exponential workloads shown in Figure 4.7a.

The sensitivity of the RAP variants to the degree of homogeneity in resource scaling among application tiers is now explored. In the experiments described previously in this section the web tier is modeled by a multi-server service center while the database tier is modeled by a multi-core service center. 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. To study the effect of heterogeneous resource scaling among application tiers on the optimlaity of the RAP variants a set of experiments are conducted.



(a) mean demands of web and database tiers are significantly different



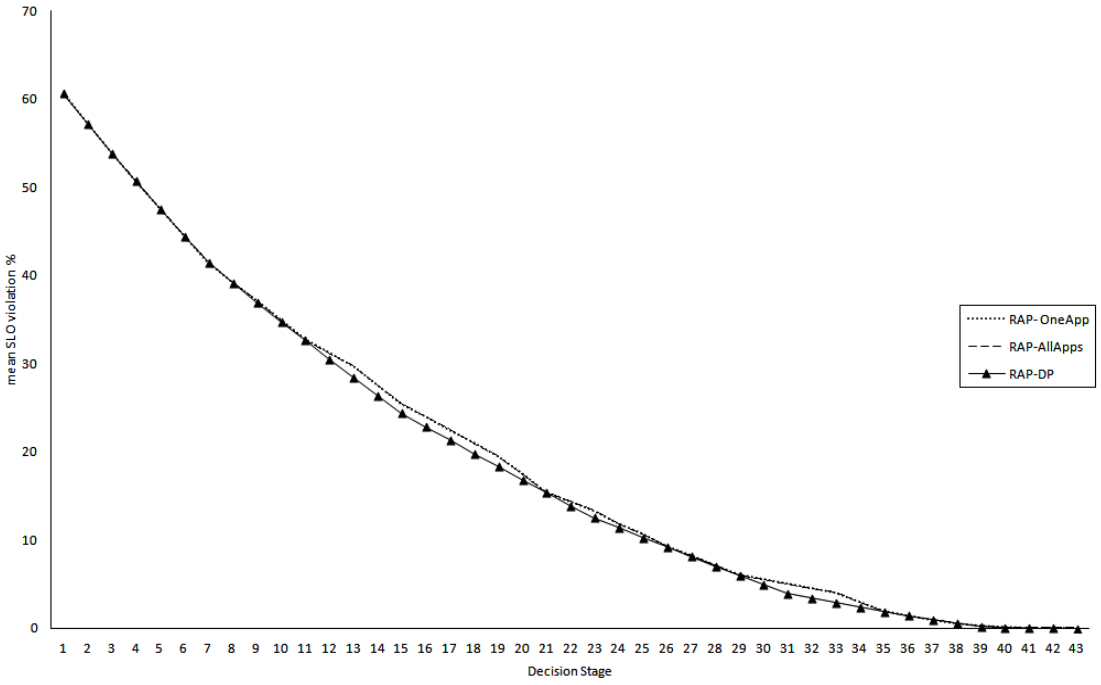
(b) mean demands of web and database tiers are smiliar

Figure 4.8: Sensitivity of RAP variants to similarity in resource demands between application tiers when subjected to workloads with bursty session arrivals

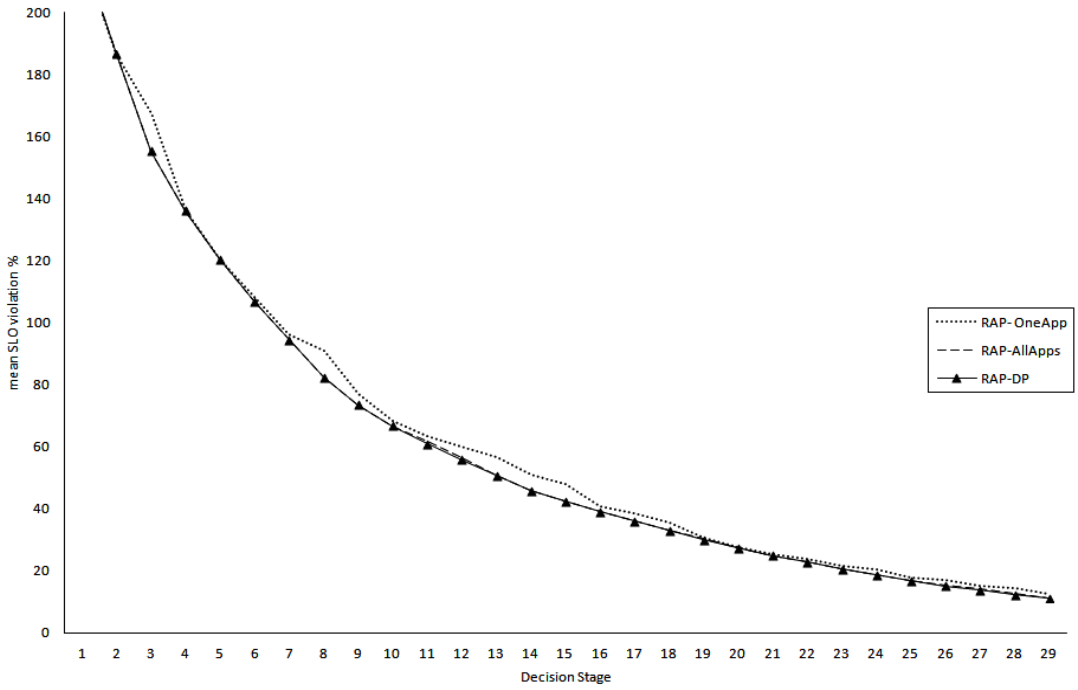
The experiments described previously in this section with similar mean web and database demands are repeated. However, the database tier is modeled by a multi-server service center instead of a multi-core service center. In this way, the web and database tier are allocated same types of resource instances which yields *homogeneous* scaling of resources in both tiers. The results obtained from the three RAP variants using homogeneous resource scaling are compared with the previous results obtained using heterogeneous resource scaling. It should be noted that in the following experiments only similar mean web and database demands are explored. This allows resource scaling to be explored in both tiers rather than the web tier only which is the case when the mean web demand is much higher than the mean database demand.

Figure 4.9 shows the mean SLO violation percentages of the solutions obtained by the three RAP variants for both exponential and bursty workloads when resource scaling is homogeneous in the two tiers. The results obtained in Figure 4.9a for exponential workloads using homogeneous resource scaling are compared with corresponding results shown previously in Figure 4.7b using heterogeneous resource scaling. It is noticed from this comparison that allocating same types of resource instances to application tiers, i.e., homogeneous resource scaling, makes the solutions obtained by the RAP variants at various decision stages more close than those obtained when the application tiers are allocated different resource types, i.e., heterogeneous resource scaling. The same findings can be observed for bursty workloads when comparing the results shown in Figure 4.8b with corresponding results shown in Figure 4.9b. The is because using heterogeneous resource scaling among application tiers can trigger more changes in the bottleneck tier over the planning horizon than with using homogeneous resource scaling.

In summary, the more degree of similarity in resource 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



(a) workloads with exponential session arrivals



(b) workloads with bursty session arrivals

Figure 4.9: Sensitivity of RAP variants to homogeneity in resource scaling among application tiers

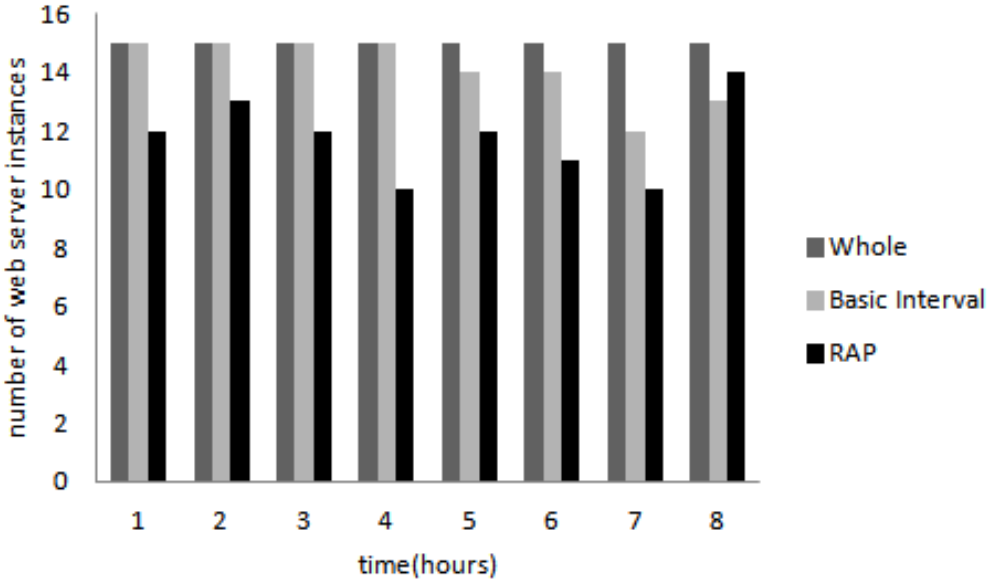
obtained by RAP-DP. However, both heuristic RAP variants, i.e., RAP-AllApps and RAP-OneApp, can obtain close to optimal solutions in most of the experiments conducted for both exponential and bursty workloads.

4.6 Comparing RAP with Burstiness-Agnostic SLP Approaches

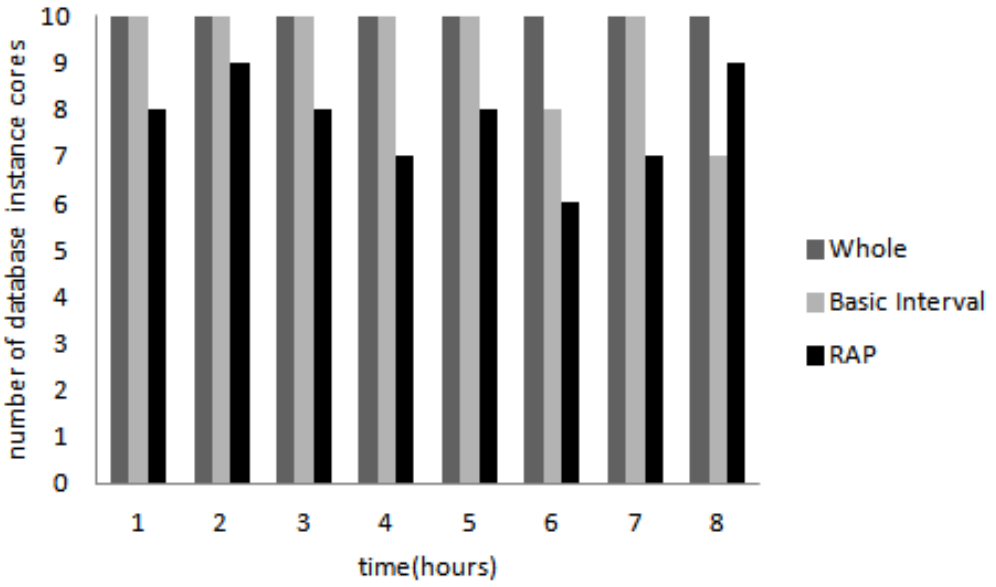
In this section and in the following chapters of the thesis RAP refers to the RAP-OneApp variant. The other two variants of RAP, i.e., RAP-DP and RAP-AllApps, are explicitly called by their names. In this section RAP is compared to the whole and basic interval approaches described previously in Section 3.4. As mentioned previously these two approaches are simpler than RAP. However, they do not take into account either workload variability or burstiness. 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.

Figure 4.10 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 eight 1-hour intervals. Specifically, the five workload scenarios are characterized by medium variability and extremely high burstiness in session arrivals, i.e., $SCV_{a,k}$= 3 and $I_{a,k}$= 10,000. The setup of this experiment is performed to allocate as many resources needed to satisfy the SLOs of all five applications.

Figure 4.10 illustrates the operation of each resource allocation approach. In the whole approach all resource allocation intervals are allocated the same number of web server instances and database instance cores. In the basic interval approach resource allocations follow almost a chronological order. Specifically, for resource allocation intervals 1 to 6, resource allocations in any given resource allocation interval either remain the same or drop when compared to the resource allocations in the previous resource allocation interval. In resource allocation interval 7 this observation is violated since the bottleneck tier switches from the



(a) web server instances



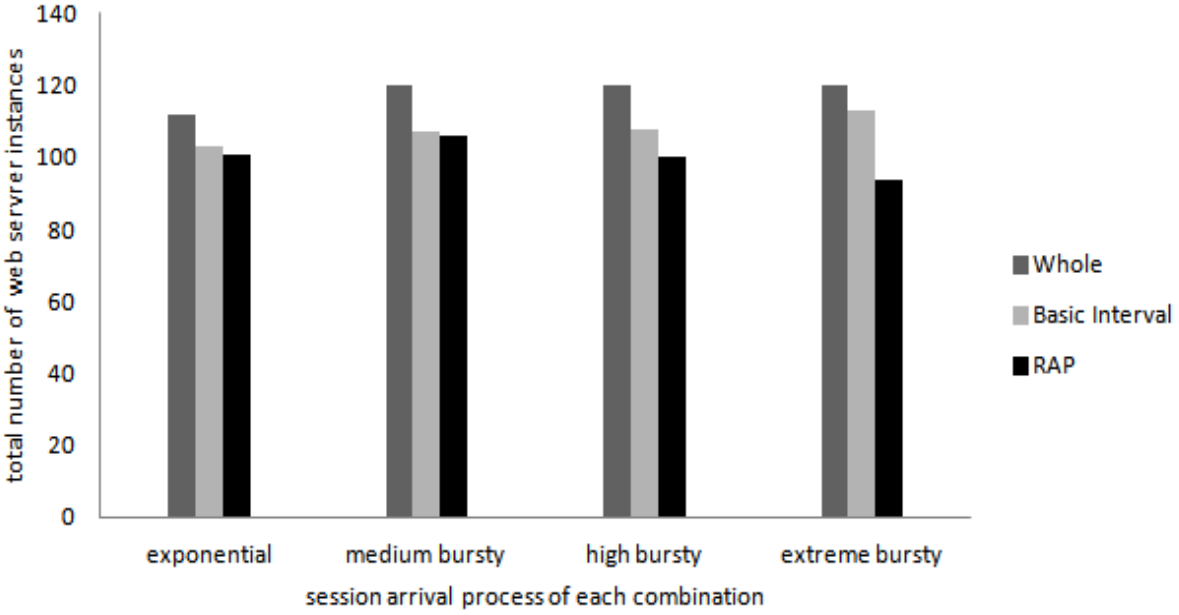
(b) database instance cores

Figure 4.10: Three different resource allocation approaches over an eight-hour planning horizon

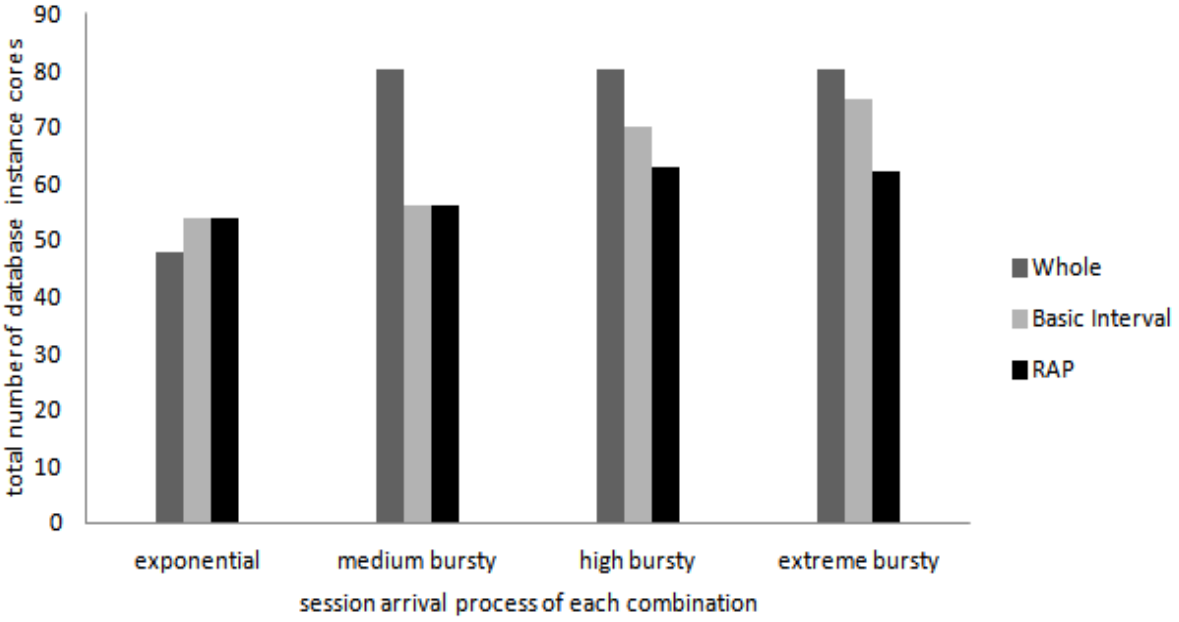
web tier to the database tier in one of the workload scenarios. RAP selects the resource allocation interval with the highest mean response time. Consequently, its resource allocations show a pattern that reflects the burstiness characteristics of the application workload scenarios. For example, in Figure 4.10a the five applications considered in this experiment are allocated the highest number of web server instances in resource allocation interval 8. The happens because this resource allocation interval is the most bursty one among all other resource allocation intervals. On the contrary, the resource allocation interval 4 is allocated the least number of web server instances because it is the least bursty one. From Figure 4.10, RAP estimates a requirement of at most 14 web server instances and 9 database instance cores per resource allocation interval to satisfy application SLOs. The other two approaches estimate higher numbers since they do not take into account either variability or burstiness.

The sensitivity of the three policies to burstiness is now explored. Figure 4.11 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 identical session arrival process characteristics. Table 4.4 lists the values of the parameters which characterize the session arrival process of each combination of workload scenarios shown in Figure 4.11. Figure 4.11 confirms that RAP estimates the least number of resources in most cases especially those with high degree of burstiness.

Figure 4.12 shows the effect of each of the three resource allocation approaches on the mean violation percentage for four different combinations of workload scenarios. Each combination consists of ten application workload scenarios. The four combinations have progressively higher degrees of burstiness. Refer to Table 4.4 for a list of parameters that characterize the session arrival process of each combination. The number of web server instances and



(a) web server instances



(b) database instance cores

Figure 4.11: Total number of resource instances allocated by three different approaches

Table 4.4: Parameters of the session arrival processes of the workload scenario combinations used in the experiments

Session Arrival Process of Each Combination	$S_{a,k}$	$I_{a,k}$
Exponential	1	1
Medium bursty	3	100
High bursty	3	1000
Extreme bursty	3	10,000

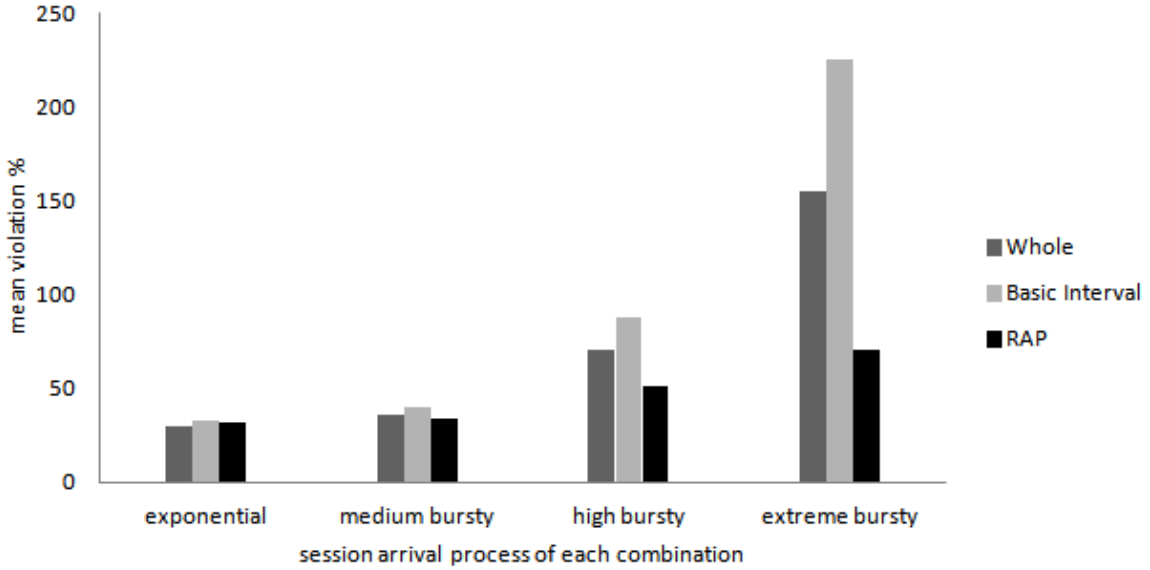


Figure 4.12: Mean SLO violation percentages of four combinations of workload scenarios

the number of database instance cores available per interval are set to 16 each. This setting is selected such that the SLOs of the applications represented by the workload scenarios in each combination are violated. Therefore, the violation percentages of the different combinations using the three resource allocation approaches can be compared. Finally, the results are shown again for a planning horizon of eight hours and a resource allocation interval of 1 hour.

As shown in Figure 4.12, 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 provide highly pessimistic SLO violation percentage estimates for the highly bursty combinations of workload scenarios. In particular, both of these approaches estimate mean SLO violation percentages of 155% and 226%, respectively, for the extremely bursty combination of workload scenarios while RAP estimates 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.

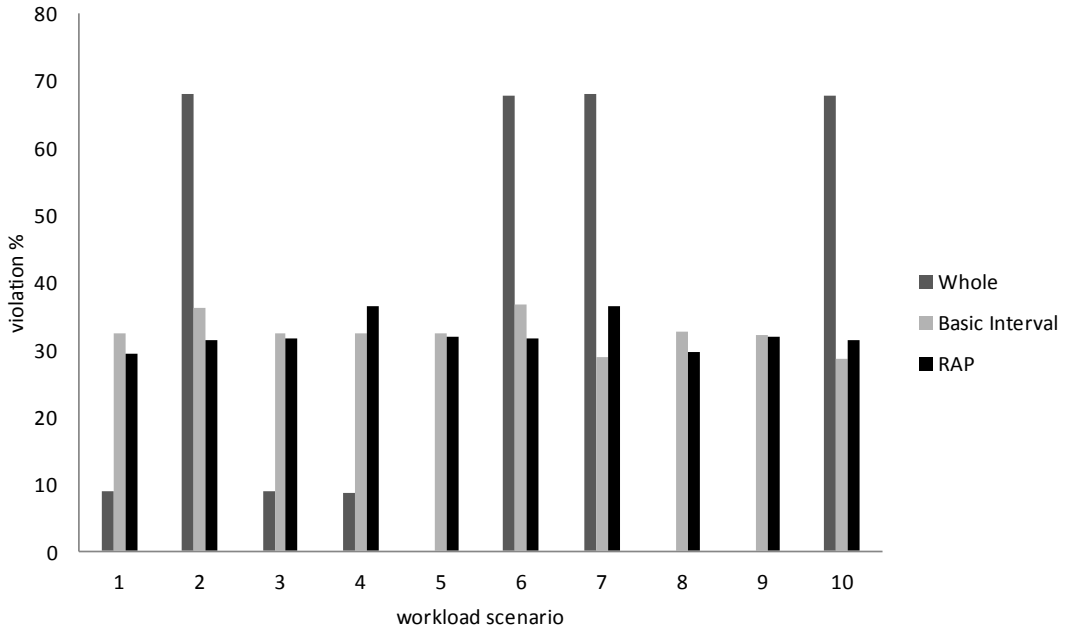
Figure 4.13 provides a more detailed view of the four combinations of workload scenarios evaluated in 4.12. This detailed view shows the individual SLO violation percentages of the ten workload scenarios constituting each combination. It can be noticed that for all combinations, the whole approach achieves very low violation percentages for some workload scenarios while very high violation percentages for other workload scenarios. For example, in Figure 4.13a which shows the SLO violation percentages of the exponential combination, the SLO violation percentage estimated by the whole approach is close to 70% for workload scenarios 2, 6, 7 and 10 while the SLO violation percentage for workload scenarios 1,3,4 and 5 are less than 10%. This is not the case with both basic interval and RAP approaches. These two approaches achieve an SLO violation balance across the ten applications. However RAP has the added advantage of obtaining the least per-application violation percentages except for the workload scenarios 4, 7 and 10 in the exponential distribution shown in Figure 4.13a. Furthermore, as shown in Figures 4.13b to 4.13d, as the degree of burstiness increases RAP achieves less SLO violation percentages than those achieved by the basic interval approach. For example, for the medium bursty combination shown in Figure 4.13b, the maximum difference in SLO violation percentage achieved by the basic interval approach and RAP among all workload scenarios is 13% for workload scenario 10. However, for the extremely bursty combination shown in Figure 4.13d, the maximum difference is 189% for workload

scenario 5.

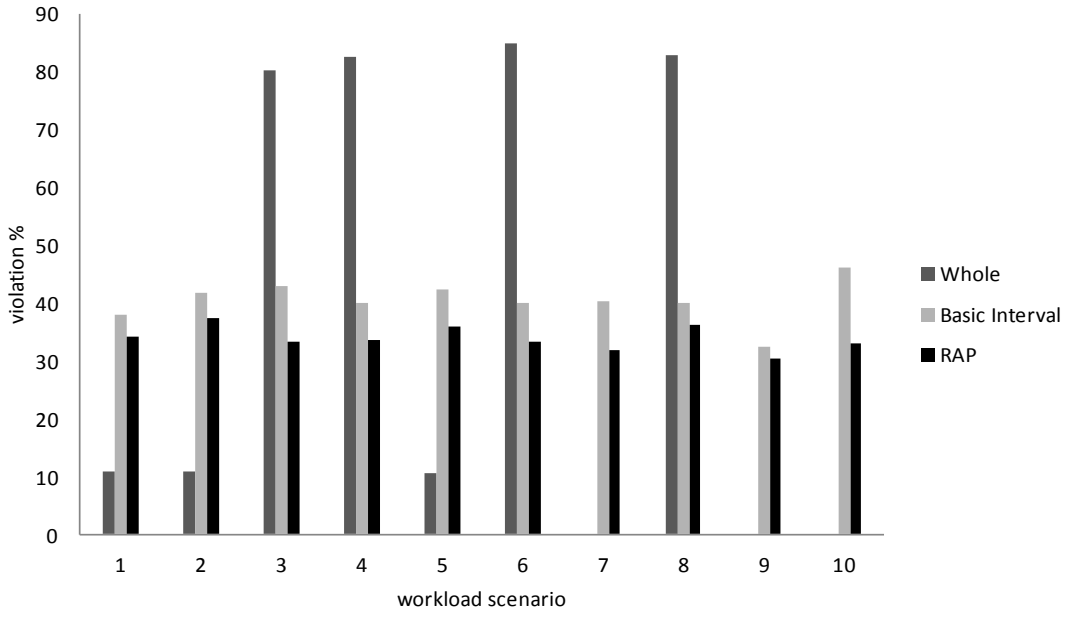
Figure 4.13 also illustrates the number of violating applications that result from each of the three approaches.In general, the figure shows that neither basic interval nor RAP approaches are able to reduce the number of violating applications. However, the whole approach reduces the number of violating applications in the exponential and medium bursty combinations shown in Figures 4.13a and 4.13b, receptively. For example, the whole approach reduces the number of violating applications in the exponential combination to 7 instead of 10 where application 5, 8 and 9 have zero violations. However, this happens at the expense of other violating applications such as applications 2, 6, 7 and 10, which have much higher violation percentages than those achieved by basic interval and RAP approaches. Furthermore, the whole approach is not able to reduce the number of violating applications in the high and extremely bursty combinations shown in Figures 4.13c and 4.13d, receptively. These results are consistent with the formulation of the optimization problem described previously in Section 3.1 which minimizes the sum of SLO violations for a given set of applications rather than minimizing the number of SLO violating applications.

Recall from Section 3.1 that the objective of the global SLO and resource allocation optimization problem can be modified to minimize the number of violating applications instead of minimizing the SV value as captured by Equation (3.7). To minimize the number of violating applications at each decision stage, the RAP-DP approach described in Section 3.3.1 can be modified to select the application which achieves a V_a value of zero instead of the application which results in the most reduction in V_a.

In summary, the RAP method 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 balanced SLO violations across applications thereby minimizing penalties due to SLO violations. Furthermore, selecting a resource allocation interval that is smaller in size than the planning horizon reduces the number of

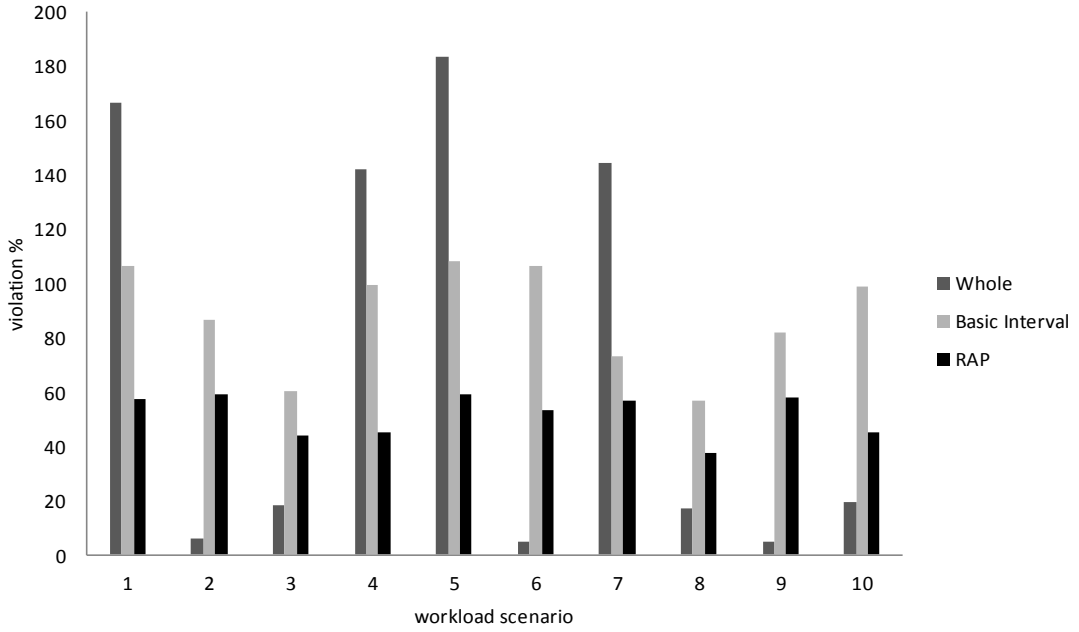


(a) violation percentages of the exponential combination

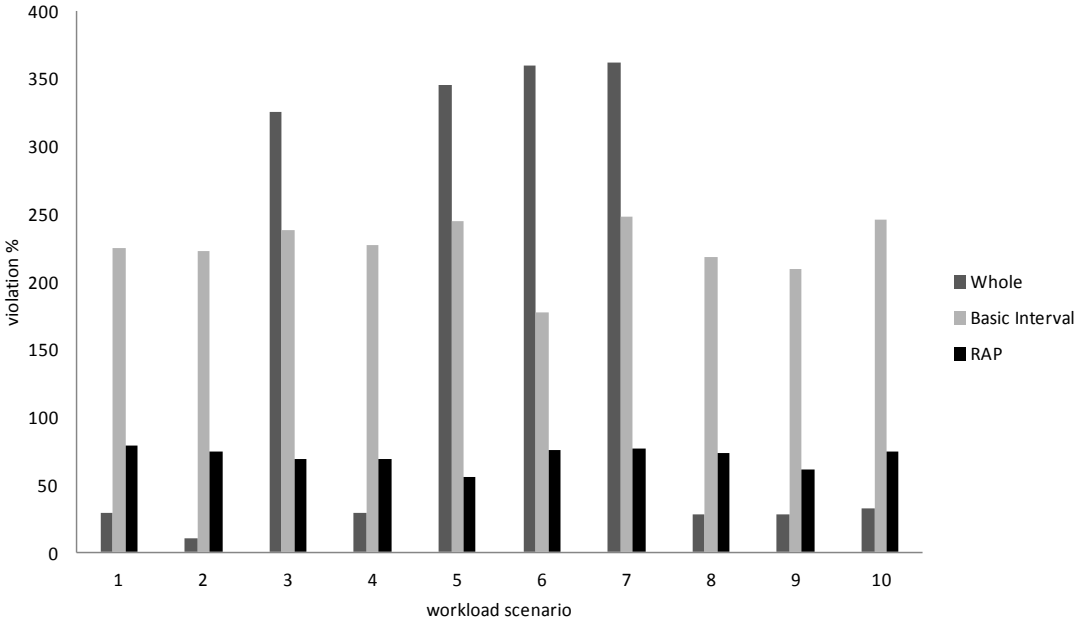


(b) violation percentages of the medium bursty combination

Figure 4.13: Detailed SLO violation percentages of four combinations of workload scenarios with each combination consisting of ten workload scenarios



(c) violation percentages of the high bursty combination



(d) violation percentages of the extreme bursty combination

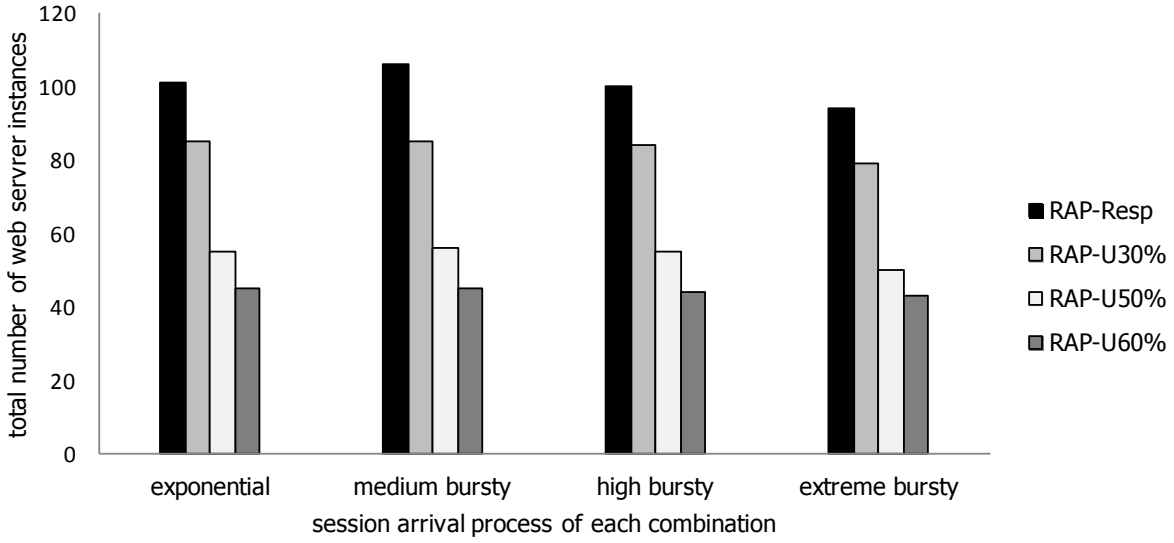
Figure 4.13: Detailed SLO violation percentages of four combinations of workload scenarios with each combination consisting of ten workload scenarios (Continued)

resources required to satisfy the SLOs of applications characterized by bursty workload scenarios.

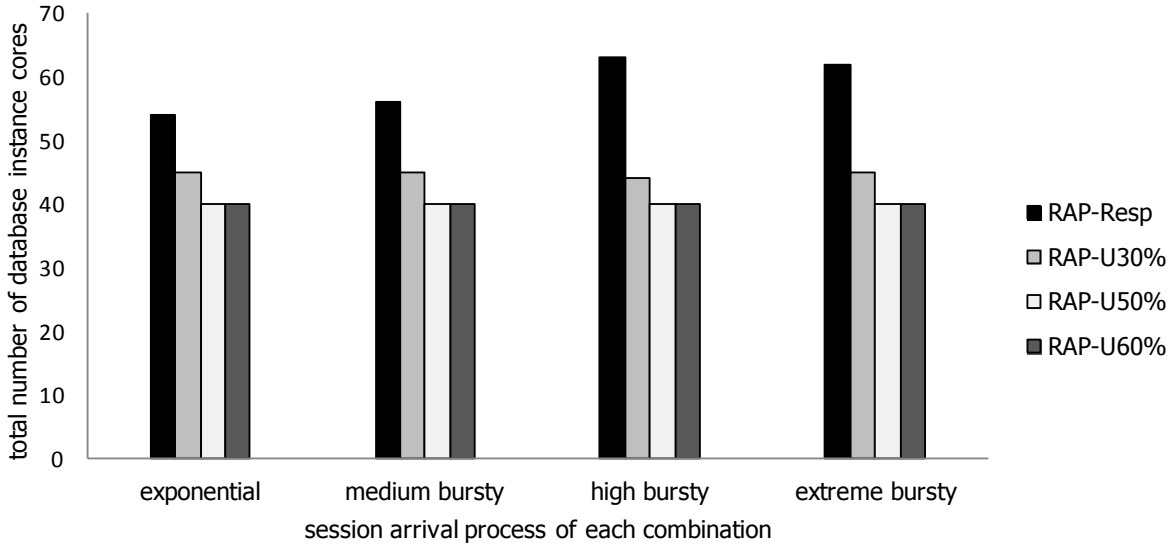
4.7 Comparing RAP with Utilization Based SLP

This section investigates a technique that only considers resource utilization thresholds, e.g., a CPU utilization target, instead of SLOs based on response times. As mentioned previously in Section 1.1, these techniques are commonly used in practice since they do not require detailed performance models such as those employed in this work. An example of a utilization-based technique is the *Quartermaster* capacity manager service proposed by Rolia *et al.* [87]. Quartermaster allows resource utilization thresholds to be set as an indirect mechanism for achieving adequate application response times. The RAP method accommodates both utilization and response time based SLO thresholds. This section uses this feature to compare these two different methods to SLP.

Figure 4.14 shows the total number of web server instances and database instance cores allocated to four different combinations of workload scenarios where each combination has five applications. Similar to the previous sections, the four combinations have progressively higher degrees of burstiness. Refer to Table 4.4 for a list of parameters that characterize the session arrival process of each combination. RAP, i.e., RAP-OneApp, is used as the resource allocation approach in all cases. However, two different versions are explored. The first version is the same as the approach taken in the previous sections where SLOs are specified based on mean response times. This version is referred to as *RAP-Resp*. The other version accepts an SLO threshold based on a mean resource utilization target of the bottleneck resource over the planning horizon. This is referred to as *RAP-Ux%* where x refers to the value of the threshold. Specifically, we experiment with x values of 30%, 50% and 60%. The setup of this experiment is performed to allocate as many resources needed to satisfy the SLOs of all five applications in each combination.



(a) web server instances



(b) database instance cores

Figure 4.14: Comparing utilization based SLP with response time based SLP

Figure 4.14 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 4.15 shows the effect of using utilization-based targets on application response times. For each of the *RAP-Ux%* estimated allocation plans, the mean response time SLO violations are calculated as defined previously in Section 3.1. From Figure 4.15, 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. However, such policy requires trying a wide range of utilization targets to reach a certain target which achieves the required SLO. Such randomly selected target may allocate excessive resources over the planning horizon thereby increasing costs. For example, Figure 4.15 shows that a 30% utilization target cannot satisfy the SLOs of all applications which requires trying a lower utilization target such as 20%. This lower utilization target may satisfy the SLOs of all applications at the expense of increased resource costs..

In summary, these results show the challenges in meeting response time targets for applications with bursty workloads by controlling their resource utilizations. The results suggest that SLP methods should incorporate analytic models that can help SPs directly specify response time based SLOs.

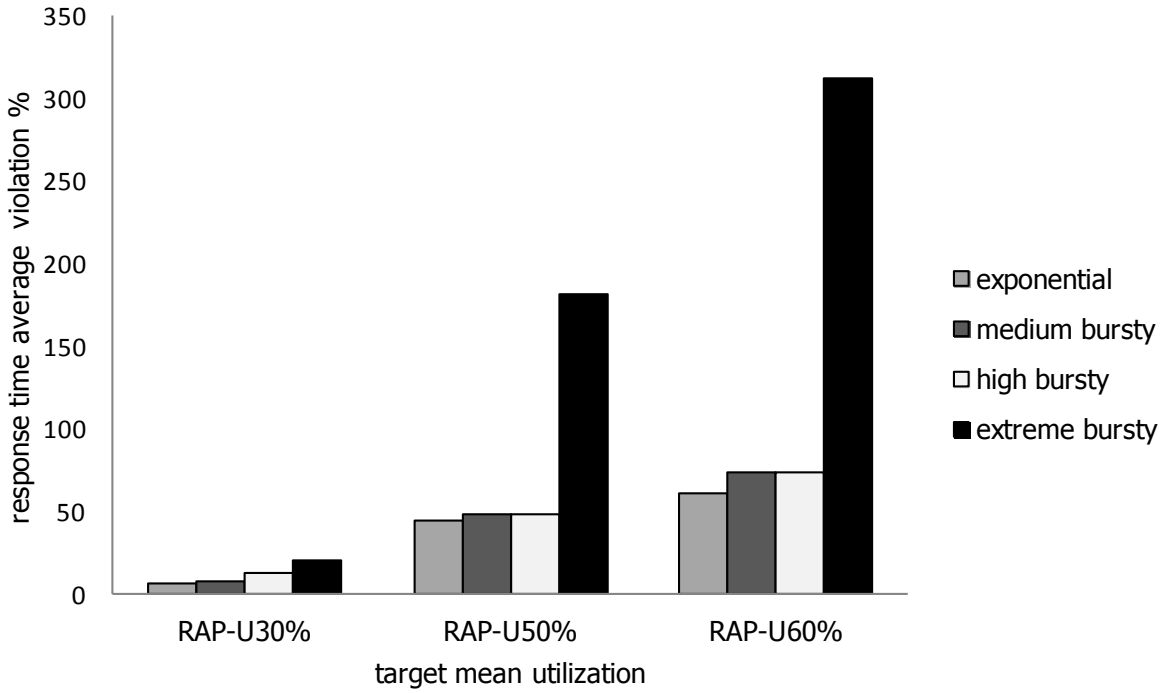


Figure 4.15: Effect of utilization based SLP on response time violation

4.8 Flexibility of RAP in Handling Alternative Service Level Objectives

In all preceding sections it is assumed that an application's SLO is defined over a planning horizon spanning multiple resource allocation intervals. In this section this assumption is relaxed by showing the ability of RAP to accommodate different techniques for defining application SLOs. Specifically, the previous approach of defining application SLOs over the entire planning horizon is compared with another approach of defining application SLOs over each resource allocation interval.

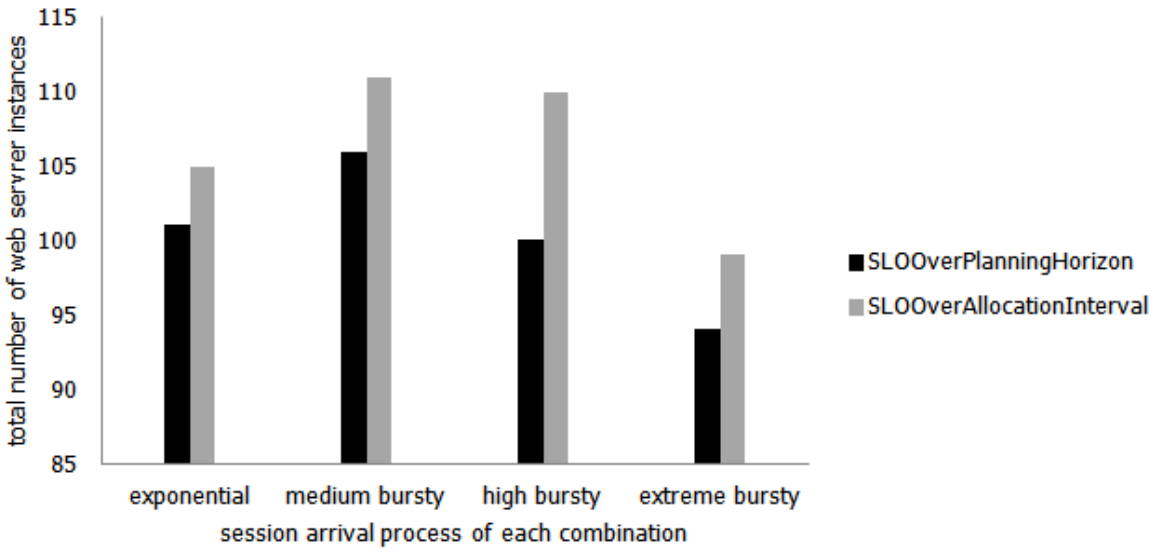
An experiment is conducted for four different combinations of workload scenarios with progressively higher degrees of session arrival burstiness over a planning horizon of eight 1-hour resource allocation intervals. Each combination has five workload scenarios. Refer to Table 4.4 for a list of parameters that characterize the session arrival process of each combination. For each combination of workload scenarios, resource estimate obtained by

RAP to satisfy each application's SLO defined over the entire planning horizon is compared with the corresponding resource estimate obtained when the application's SLO is defined over each resource allocation interval.

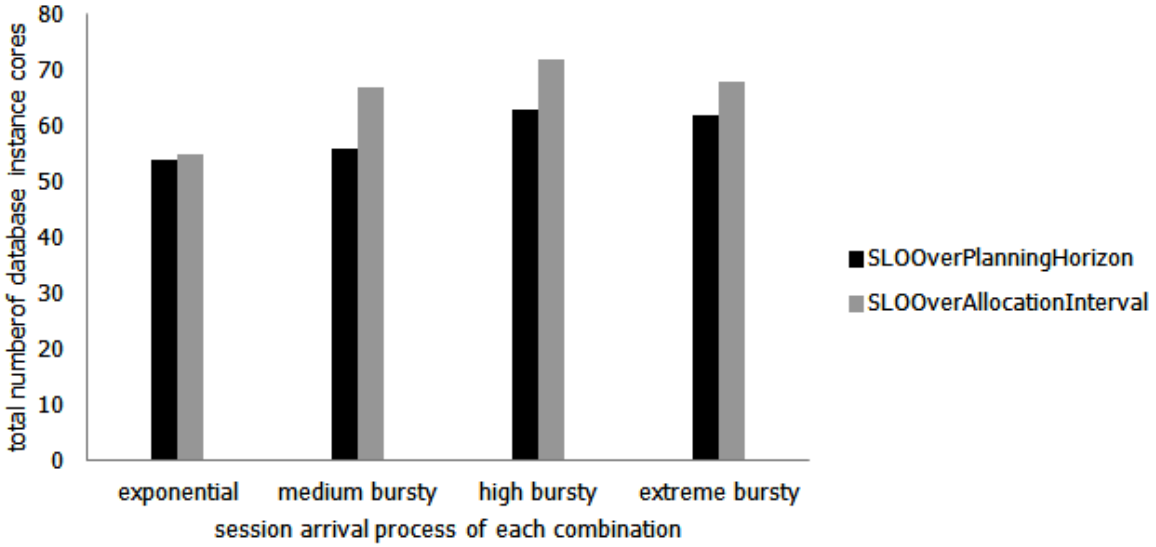
Figure 4.16 shows the total number of web server instances and database instance cores allocated over eight hours for the two approaches denoted by *SLOOverPlanningHorizon* and *SLOOverAllocationInterval*. The figure shows that the *SLOOverAllocationInterval* approach needs more resources than *SLOOverPlanningHorizon* approach in all combinations of workload scenarios. The reason for this is that the *SLOOverPlanningHorizon* approach only considers the *most heavily loaded* resource allocation intervals for additional resource instances to handle any SLO violations which occur over the whole planning horizon. On the contrary, the *SLPOverAllocationInterval* approach allocates additional resource instances to *all heavily loaded* resource allocation intervals to handle the SLO violations which can occur in these resource allocation intervals. In summary, the results show that under both bursty and non bursty workload scenarios, customers can obtain cost savings if their SLOs are defined over timescales that are longer than the timescales over which dynamic resource allocations are explored. This type of an analysis can help cloud SPs negotiate with their customers the cost of resource allocation estimated for different ways of defining SLOs. Based on this cost estimate customers may revise their way of defining the SLOs of their applications to lower their costs.

4.9 Improving Performance of RAP-DP and RAP-AllApps by Exploiting Parallelism

This section discusses the improvement that can be achieved in the performance of RAP-DP and RAP-AllApps by exploiting parallelism. Recall from Section 3.3.1 that RAP-DP is guaranteed to obtain an optimal solution for the global SLO and resource allocation optimization problem described previously in Section 3.1. However, this requires the invocation



(a) web server instances



(b) database instance cores

Figure 4.16: Flexibility of RAP to different application SLO definitions

of WAM-QNM for all applications in all resource allocation intervals that have free resource instances to decide on the candidate application and the candidate resource allocation interval for additional resource instance. This set of invocations should be done at each decision stage. As described previously, the number of WAM-QNM invocations made by RAP-DP at each decision stage is equal to the product of the number of applications and the number of resource allocation intervals. This increases the computation time because the invocation of WAM-QNM is the most time consuming part of RAP-DP.

Parallelism can be exploited to improve the performance of RAP DP since at each decision stage the invocation of WAM-QNM for a given application in a given resource allocation interval is completely independent of the WAM-QNM invocation for the the same application in a different resource allocation interval. It is also independent from the WAM-QNM invocation for another application in either the same or in a different resource allocation interval. Therefore, this set of invocations can be done in parallel at each decision stage by assigning each invocation to a separate thread of execution.

The same parallelization approach can also be applied to RAP-AllApps. As described previously in Section 3.3.2, at each decision stage RAP-AllApps has to invoke WAM-QNM for all applications like RAP-DP. However, RAP-AllApps makes only one WAM-QNM invocation for each application by allocating one additional resource instance in the resource allocation interval which has the highest mean response time and which has free resource instances available for allocation. Therefore, at each decision stage WAM-QNM is invoked by RAP-AllApps a number of times equal to the number of applications. This also increases the computation time at each decision stage if the number of applications increases significantly. This set of invocations can also be done in parallel at each decision stage by exploiting multi-threading.

As described previously in Section 3.3.3, RAP-OneApp is the most computationally efficient RAP variant because it makes only one WAM-QNM invocation at each decision stage.

Therefore, RAP-OneApp will not benefit from parallelism at each decision stage. As a result, only the previously implemented single-threaded version of RAP-OneApp is used in this section.

Based on the above observations multi-threaded versions of both RAP-DP and RAP-AllApps are implemented. These versions are implemented using the well-known Open Multi-Processing (OpenMP) Application Programming Interfaces (APIs) for shared memory multiprogramming [47]. OpenMP defines a set of compiler directives that creates and assigns selected sections from the code of a software program to separate threads of execution. As as result, these threads can run in parallel by assigning each thread to an idle processor core in the machine running the code. To compare the performance of the single-threaded with that of the multi-threaded versions of RAP-DP and RAP-AllApps variants a machine with 12 processor cores and 64 GB RAM is used. Each core is a 2.4 GHz Intel Xeon processor.

An experiment is conducted to compare the performance of the previously implemented single-threaded versions of the different RAP variants with the corresponding multi-threaded versions. In this experiment forty different applications are analyzed for SLP with twenty applications subjected to exponential session arrivals and twenty applications subjected to bursty session arrivals. These applications are analyzed over a planning horizon of 4 hours with a resource allocation interval of 1 hour. The mean demands of the web and database tier are selected to be similar, i.e., 20 ms and 18 ms, receptively. These settings are done to allow the bottleneck tier to change over the planning horizon for some or all of the applications when moving from one decision stage to another. Therefore, both the web server instances and database instance cores available for allocation per resource allocation interval can be used.

The maximum number of web server instances and database instance cores per resource allocation interval are set to 41 each. The choice of these settings were driven by the need to bound the computation time of the single-threaded versions of RAP-DP to a few hours.

Specifically, as described previously in Section 4.4, all RAP variants start with an initial resource allocation plan at decision stage 0 which involves allocating one web server instance and one database instance core to each application in each resource allocation interval. This makes an initial aggregate allocation of 40 web server instances and 40 database instance cores to the forty applications over all resource allocation intervals. Therefore, only one web server instance and one database instance core will be available for allocation for all applications in each resource allocation interval in the subsequent decision stages. In this way each RAP variant will run for a maximum of 8 decision stages which puts an upper bound on the computation time of each RAP variant. Finally, each single-threaded and multi-threaded version of the three RAP variants is used in conjunction with WAM-QNM to obtain a resource allocation plan for this experiment.

Table 4.5 lists the number of WAM-QNM invocations and the corresponding computation times for each of the single-threaded and multi-threaded versions of the different RAP variants. As shown in the table, both RAP-AllApps and RAP-DP gained performance improvement by exploiting parallelism. Specifically, when comparing the computation times of the single-threaded and the multi-threaded versions of both variants, it can be noticed that their computation times are improved by a factor of 9. Although this factor is expected to be close to 12 which is the number of processor cores of the machine used in the experiment, this is not the case. The reason for this is the tight resource constraints applied in the experiment. For example, at the initial decision stages RAP-DP can make 160 WAM-QNM invocation, i.e., 40 applications times 4 resource allocation intervals, per decision stage. This can be done because initially each resource allocation interval has free resource instances. However, as the resource allocation process continues in subsequent decision stages the number of resource allocation intervals that have free resource instances decreases. Therefore, WAM-QNM is not invoked for all applications in these reserved resource allocation intervals. Furthermore, WAM-QNM is invoked by an application in a given resource allocation interval

Table 4.5: Execution times of single-threaded and multi-threaded versions of the three RAP variants on a 12 core machine

RAP Variant	Number of WAM-QNM Invocations	Computation Time in Minutes
RAP-OneApp	48	18
Single-threaded RAP-AllApps	210	75
Multi-threaded RAP-AllApps	210	9
Single-threaded RAP-DP	456	163
Multi-threaded RAP-DP	456	19

if and only if this resource allocation interval has free resource instances for the bottleneck tier of this application. These reasons limit the number of WAM-QNM invocations that can be assigned to separate threads of execution at the final decision stages.

It can also be noticed in Table 4.5 that the computation time of RAP-OneApp is still slightly lower than that of the multi-threaded version of RAP-DP. However, the computation time of the multi-threaded version of RAP-AllApps is half that of RAP-OneApp. This might suggest that the multi-threaded version of RAP-AllApps can be used instead of RAP-OneApp because it provides more close to optimal solutions and at the same time it is more computationally efficient. However, the performance gain depends on the number of applications and the number of processor cores. For a given number of cores, the gain of the multi-threaded version of RAP-AllApps over RAP-OneApp will diminish with an increase in the number of applications. Since the number of WAM-QNM invocations is even more in RAP-DP than in RAP-AllApps, the gains from using the multi-threaded version of RAP-DP for a given number of cores will reduce even more rapidly with an increase in the number of applications to be analyzed and with an increase in the number of intervals over which resource allocations are explored.

In summary, the multi-threaded versions of RAP-DP and RAP-AllApps can provide significant performance gains if the number of cores is comparable to the number of WAM-QNM invocations in each decision stage. This suggests that cloud SPs can use large, inexpensive, "pay as you go" compute clusters such as Amazon Cluster Compute Instances [14] to speed

up SLP analysis using these techniques. In scenarios where deployment of such clusters is not feasible, RAP-OneApp remains the most computationally efficient variant for conducting SLP analysis. Chapter 6 introduces a clustering technique to reduce SLP analysis time using all three RAP variants.

4.10 Summary

This chapter describes a detailed simulation evaluation of RAP. Firstly, it describes the approach used in the thesis to characterize application workload scenarios and the simulation setup employed in evaluating RAP. Secondly, it shows results that compare the accuracy of WAM-QNM and MVA-QNM techniques which are employed by RAP. These results confirms the findings reported by Krishnamurthy *et al.* which show the accuracy of WAM-QNM over MVA-QNM for bursty workload scenarios.

Thirdly, results which characterize the optimality of the three RAP variants are shown. The results show that the proposed RAP variants can identify optimal or near optimal resource allocation plans without exhaustively generating all possible plans. Specifically, RAP-DP can identify a plan that minimizes SLO violations for a given cloud resource capacity. It is also shown that RAP-AllApps is able to identify optimal plans of resource allocations if the bottleneck resource remains the same in all resource allocation intervals. Furthermore, the results show that under both bursty and non-bursty workload scenarios, the heuristic baseline RAP approach, i.e., RAP-OneApp, is able to identify near optimal plans of resource allocations while reducing computational complexity significantly when compared to the optimal RAP-DP approach.

A study is also conducted which explores the sensitivity of the RAP variants to the degree of similarity in resource demands between application tiers and the degree of homogeneity in resource scaling among application tiers. This study shows that more deviation occurs in the solutions obtained by RAP-AllApps and RAP-OneApp from those obtained by RAP-DP

when the demands of application tiers become more similar and when the application tiers are allocated resources of different types.

Fourthly, the chapter shows results that RAP is able to more accurately determine the resources required for delivering specified SLOs compared to other competing techniques especially for applications characterized by bursty workload scenarios. Furthermore, the results establish the superiority of RAP to the prevalent practice of considering SLO targets based on resource utilization thresholds over the planning horizon.

Fifthly, the results show the flexibility of RAP to accommodate different techniques for defining application SLOs. Specifically, the results show that under both bursty and non bursty workload scenarios, customers can obtain cost savings if their SLOs are defined over timescales that are longer than the timescales over which dynamic resource allocations are explored.

Finally, an approach to improve the performance of RAP-DP and RAPAllApps by exploiting parallelism is shown. Specifically, multi-threaded versions of the two RAP variants are implemented to reduce their computation times for large number of applications. A machine with 12 processor cores is used to compare the computation times of the single-threaded and the multi-threaded versions of the two RAP variants. The results show a performance improvement of the multi-threaded versions over the corresponding single-threaded versions by a factor of 9. These results suggest that a cloud SP can find optimal or close to optimal solutions by leveraging inexpensive, on-demand, cluster compute instances to execute the parallel versions of RAP.

Chapter 5

Workload Uncerainty Module

This chapter provides a detailed description of the workload uncertainty module described briefly in Section 1.3.2. Section 5.1 describes the proposed Monte Carlo simulation technique [82] employed by the uncertainty module. Section 5.2 presents results to illustrate the utility and sensitivity of the Monte Carlo simulation technique to various parameters.

5.1 Monte Carlo Simulation Technique

As mentioned previously in Section 1.3.2 and Section 4.1, each application is characterized by a set of alternative workload scenarios. As opposed to the previous chapter, the value of k for each workload scenario $W_{a,k}$ can be more than one. Each workload scenario has a certain probability of occurrence $P_{a,k}$. Characterizing each application by a set of alternative workload scenarios can lead to an explosion in the number of alternative combinations of workload scenarios that need to be considered during the SLP process. The objective of the uncertainty module is to select a small subset of the most likely workload scenario combinations out of all the possible combinations of workload scenarios.

The uncertainty module employs a Monte Carlo simulation algorithm. As mentioned previously in Section 1.3.2, the algorithm requires a cloud SP to specify a certainty threshold h that specifies a trade-off between the accuracy in predicting the impact of uncertainty for SLP and computation time. The basic idea behind this approach is that many of the workload scenario combinations may not be worth exploring since they may have a very small probability of occurrence. The Monte Carlo simulation algorithm generates a subset of workload scenario combinations such that these combinations should have a cumulative probability of occurrence greater than or equal to the certainty threshold h.

Figure 5.1 shows the pseudo-code of the Monte Carlo simulation algorithm. As shown in the figure, the algorithm takes the following inputs:

- A set of alternative workload scenarios $W_{a,k}$ for each application $a \in \{1,2,...A\}$. This set is denoted by W_a.
- A set of probabilities of occurrence $P_{a,k}$ of these workload scenarios for each application a. This set is denoted by P_a. It should be noted that the probabilities in this set should add up to 1.0 for $a \in \{1,2,...A\}$.
- The certainty threshold $0 \leq h \leq 1$.
- A value for a variable called *RunLength* which represents the maximum number of iterations of the algorithm.

In each iteration, the algorithm selects a workload scenario $W_{a,k}$ for each application a from its set W_a. This selection is driven by the PDF of workload scenarios for this application as specified in P_a. This is repeated for all A applications. The selected workload scenarios for all A applications form one workload scenario combination denoted by $m \in M$. It is assumed that the probability of occurrence $P_{a,k}$ of a workload scenario of any given application in any workload scenario combination m is independent of the probability of occurrences of the other workload scenarios in the same combination. The reason for this assumption is that application workloads belong to different customers. Therefore, as shown in Figure 5.1, the probability of occurrence $Prob_m$ of a combination m can simply be obtained as the product of the probability of occurrences of the individual application workload scenarios in the combination. The iterations are continued to select more scenario combinations till the sum of the $Prob_m$ values of the selected scenario combinations is greater than or equal to h or the number of iterations equals *RunLength*. If the simulation terminates before producing the desired h, a cloud SP can rerun the simulation by reducing h or increasing *RunLength*. As a

```
Input: W_a, P_a, ∀ a ∈ {1, 2,…, A}, h, RunLength
Output: M, Prob_M, ToTProb
ToTProb = 0
M = { }
Run = 1
# run Monte Carlo simulation until achieving a list of workload scenario combinations M
# with a total probability of occurrence more than or equal to h or RunLength reached
While (ToTProb < h && Run ≤ RunLength)
   For Run = 1 to RunLength
      m = { }
      # cumulative probability distribution of set P_a
      # is used to select a workload scenario for each application a
      For a = 1 to A
         r= GenerateRandomNumber() where r ∈ [0,1]
         ProbSum = 0
         For k = 1 to L_a
            ProbSum += P_a,k
            If r ≤ ProbSum
               Add W_a,k to m
               Break
            End If
         End For
      End For
      # if the selected workload scenario combination m is not generated before, add it to set M
      If m is not found in M
         Add m to M
         Prob_m = P_1,f1*P_2,f2*…*P_a,fa*…*P_A,fA | P_a,fa ∈ P_a
         Add Prob_m to Prob_M
         ToTProb += Prob_m
      End If
   End For
   Run ++
End While
Return M, Prob_M, ToTProb
```

Figure 5.1: Monte Carlo simulation algorithm

Table 5.1: Alternative workload scenarios characterizing five applications considered in Figures 5.2,5.3 and 5.4

Application	**Workload Scenarios**	$\lambda_{a,k}$	$SCV_{a,k}$	$I_{a,k}$
1	heavy load	3.33	3	1000
	normal load	2.5	1 (Exp)	1(Exp)
2	heavy load	3.33	3	10,000
	normal load	2.5	1 (Exp)	1(Exp)
3	heavy load	3.33	3	10,000
	normal load	3.33	1 (Exp)	1(Exp)
4	heavy load	3.33	4	1000
	normal load	2.5	1 (Exp)	1(Exp)
5	heavy load	3.33	5	1000
	normal load	2.5	1 (Exp)	1(Exp)

result of this process, M captures a set of workload scenario combinations whose probability of occurrences cumulatively sum up to a value greater than or equal to h.

5.2 Evaluation Results

This section discusses evaluation results of the uncertainty module. The section illustrates the utility of the uncertainty module and its sensitivity various parameters. As mentioned previously in Section 5.1, the uncertainty module considers a number of probable workload scenarios, each with a certain likelihood of occurrence, for each application. Based on these workload scenarios, a number of highly likely combinations of workload scenarios are generated such that their combined probability of occurrence is greater than or equal to the certainty threshold h. As described previously in Section 1.3.2, RAP in conjunction with WAM-QNM are then invoked on each combination to obtain estimates of the number of resource instances needed over all resource allocation intervals over a planning horizon.

To illustrate the utility of the uncertainty module, a set of experiments are conducted where five applications are considered with each having two alternative workload scenarios. Table 5.1 lists the selected workload parameters for the two alternative workload scenarios. These parameters were described previously in Section 4.1. For each application, one of the

alternative workload scenarios is used to represent the normal operation of the application. It is assumed that this scenario has exponential, i.e., "Exp", session inter-arrival times. The other workload scenario is characterized by a higher session arrival rate, arrival variability, and arrival burstiness relative to the normal workload scenario. While the normal workload scenario could represent typical behaviour over a planning horizon, the other workload scenario could represent the activity in the planning horizon during periods of heightened customer use of the application, e.g., customers using an e-commerce site during the holiday season. The probability of encountering the heavy load workload scenario is denoted by p. Different values of p are explored in the experiments. Furthermore, the experiments consider values of 1.0 and 0.9 for the certainty threshold h. By setting h to 1.0, all possible combinations of workload scenarios are covered. A h value of 0.9 causes the uncertainty module to generate a subset of combinations with that cumulative likelihood of occurrence. It is assumed that this system has a capacity constraint which limits the maximum number of resource instances available at each application tier n over any given resource allocation interval. To show the utility of the uncertainty module under different resource constraints, $C_{max,n,t}$ values of 11, 12, and 13 are explored in the experiments.

Figure 5.2 illustrates the types of insights a cloud SP can obtain from the uncertainty module. The figure shows the probability of the cloud requiring more than $C_{max,n,t} = 12$ web server instances for each resource allocation interval for three different values of p. The results shown in this figure are obtained by setting h to 1.0. As shown in the figure, increasing the value of p, i.e., the probability of encountering the heavier load workload scenario for each application, increases the likelihood of requiring more resources in most of the resource allocation intervals. Interestingly, there are some exceptions to this behaviour in resource allocation intervals 6 and 7. In these two resource allocation intervals an increase in the value of p does not translate to an increase in the number of resources required. Deeper analysis revealed that these two resource allocation intervals are not likely to experience a

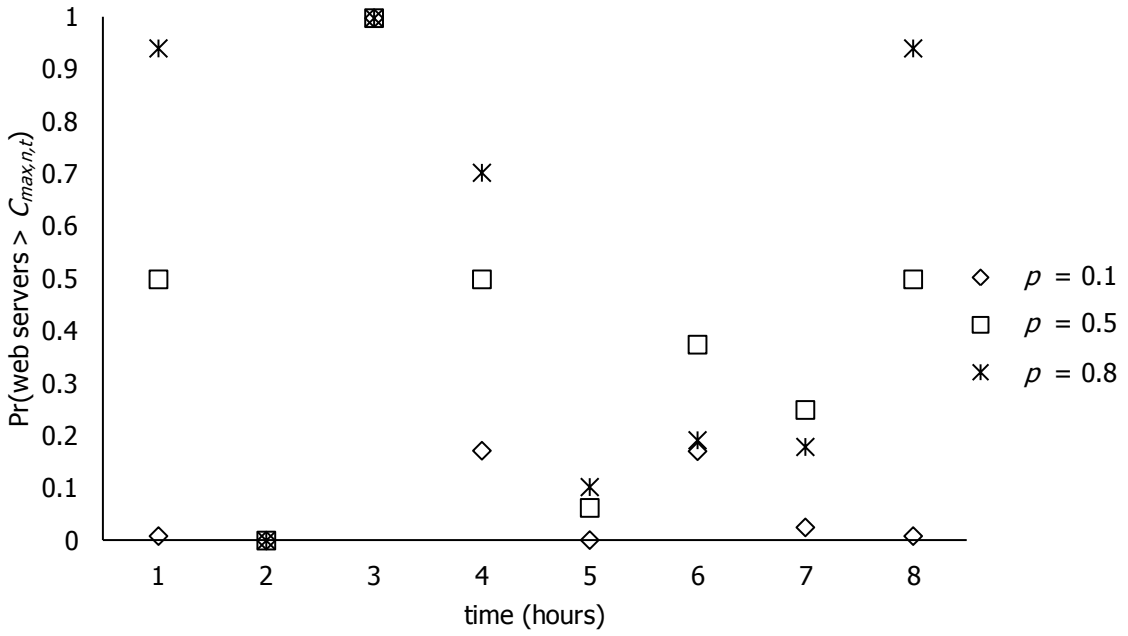


Figure 5.2: Probability of allocating per interval web server instances greater than $C_{max,n,t}$ with different p values, $h = 1.0$, $C_{max,n,t} = 12$

heavier load with the increase in the value of p even though the load increases over the entire planning horizon. Conducting such detailed analysis for all resource allocation intervals with different values of p can help cloud SPs to derive insights on such seemingly counter-intuitive situations.

It can also be noticed in Figure 5.2 that the cloud runs the highest risk of being under-provisioned and hence causing SLO violations in resource allocation interval 3. In this resource allocation interval the probability of exceeding 12 web server instances is 1 for all p values considered. Similar risks can also be visualized for resource allocation intervals 1, 4 and 8 for certain values of p. As a result, the cloud SP should focus carefully on these critical resource allocation intervals to ensure adequate resource instances are available.

Figure 5.3 shows the impact of relaxing the capacity constraint. The results shown in this figure are obtained by setting p to 0.1 and h to 1.0. The figure shows that increasing the web server instance capacity reduces the risk of SLO violations due to under-provisioning. A pool

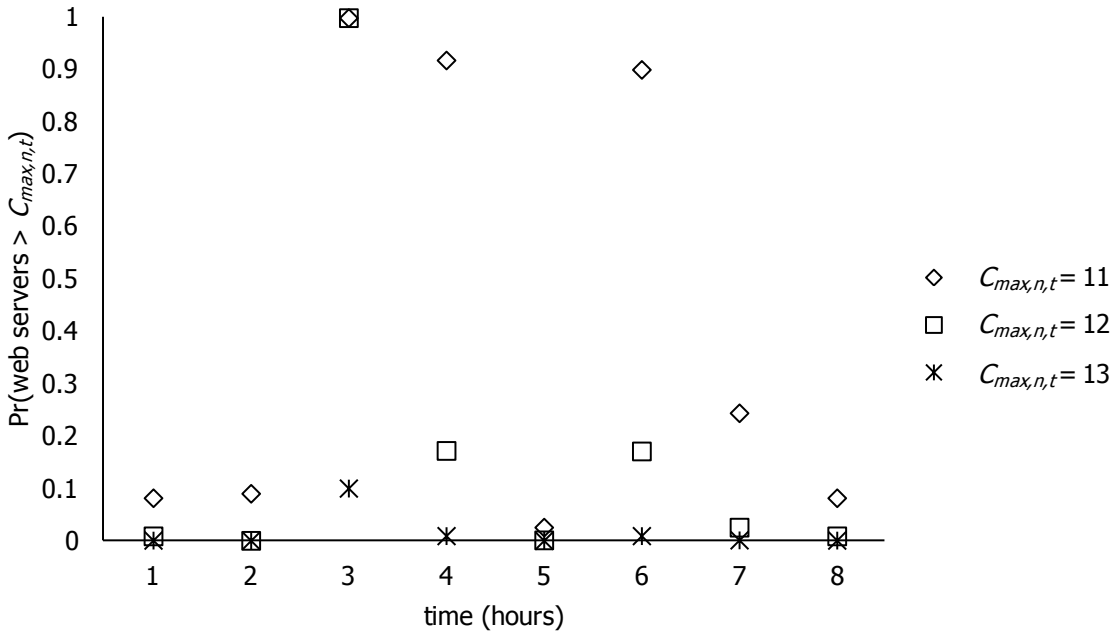


Figure 5.3: Sensitivity analysis of $C_{max,n,t}$ web server instances, $p = 0.1$ and $h = 1.0$

of 13 web server instances seems to be adequate to reduce the risk of under-provisioning to below 0.1 over all intervals. These types of sensitivity analyses can help cloud SPs quantify the risks of workload uncertainty.

Figure 5.4 shows the sensitivity of the Monte Carlo simulation technique when h changes from 1.0 to 0.9 for a p value of 0.1. To ensure statistical confidence in the results shown in the figure, 32 statistically identical Monte Carlo simulations are conducted for h=0.9 using different random seeds. In each simulation, a number of combinations of workload scenarios are generated such that their cumulative probability of occurrence is at least equal to h. For each resource allocation interval in the planning horizon, the average probability of exceeding $C_{max,n,t} = 11$ web server instances is calculated over the 32 simulations. These average probability values are compared with their corresponding probability values obtained when h=1.0. Finally, for each probability value obtained in each resource resource allocation interval, a 90% confidence interval is calculated. However, the values of the upper and lower limits of each confidence interval are not shown in the figure due to their high proximity to

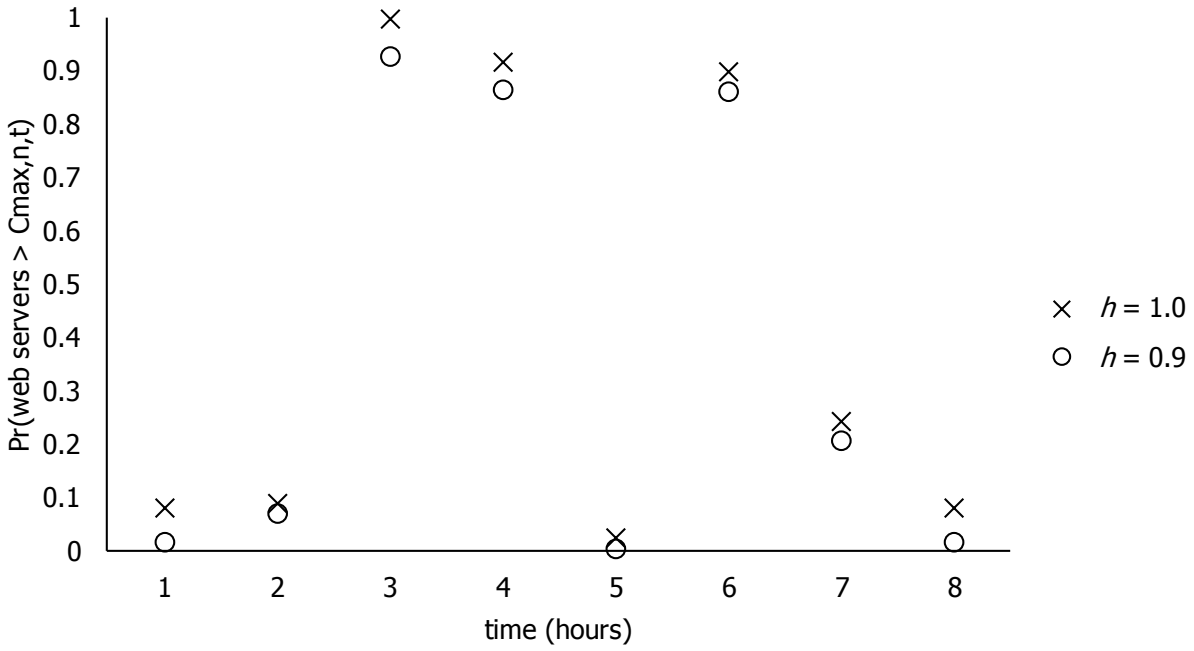


Figure 5.4: Sensitivity of the uncertainty module to h, $p = 0.1$, $C_{max,n,t} = 11$

each other.

Considering the five applications listed in Table 5.1 with two workload scenarios per application, exhaustive elaboration involves invoking the framework for a total of $2^5 = 32$ different combinations of workload scenarios. This was the approach used in the analyses embodied by Figure 5.2 and Figure 5.3. In contrast, over the 32 simulations conducted for h= 0.9, the uncertainty module identified an average of only 9 combinations that need to be evaluated representing an average reduction of 72% in the number of combinations. These 9 combinations result in an average cumulative probability of occurrence of 0.94. Figure 5.4 shows that the results obtained from the uncertainty module in conjunction with RAP and WAM-QNM for h=1.0 and h=0.9 are very close to each other thereby motivating the need for the uncertainty module. It should be noted that this significant reduction occurs since not all the combinations in the exhaustive list of 32 have a very high likelihood of occurrence. However, such dramatic benefits might not occur if there is extremely high levels of uncertainty, e.g., when all alternative workload scenarios for an application are

equally likely to occur. Another set of 32 statistically identical simulations is conducted in which p is increased to 0.5 while keeping h=0.9. The number of combinations obtained form this set of simulations to represent the above five applications is on average 29 with an average cumulative probability of occurrence of 0.91; leading to only 9% average reduction in the number of combinations. A detailed analysis of the relation between the values of h and p will be discussed in the next experiment.

A sensitivity analysis is conducted to study the effect of selecting different values for p and h on the percentage of reduction in the number of combinations of workload scenarios analyzed for SLP. An experiment is conducted where ten applications are considered for SLP. Each application has two alternative workload scenarios where one workload scenario represents normal load while the other one represents higher load as described previously. This results in a $2^{10} = 1024$ different combinations of workload scenarios. Figure 5.5 shows the percentage of reduction achieved in the number of combinations of workload scenarios for different values of p and h. As described previously, if the value of h is equal to 1.0, then all combinations of workload scenarios are analyzed, therefor, no reduction occurs in the number of combinations analyzed for SLP. This is shown in Figure 5.5 by the solid line directly on the x-axis which represents $h = 1.0$. As the h decreases, the percentage of reduction in the combinations analyzed decreases. This is expected because the h value dictates a lower bound on the cumulative probability of occurrence of the combinations analyzed. If this lower bound is relaxed, less number of combinations are required to be analyzed. For example, if the value of $h = 0.9$, the combinations required to be analyzed should have a minimum cumulative probability of occurrence of 0.9. The number of combinations analyzed in this case will be less than the number of combinations analyzed when $h = 0.7$.

Figure 5.5 also shows the effect of changing the value of p, i.e., the probability of encountering the heavy workload scenario for each application, on the percentage of reduction in the combinations analyzed. For a given value of h, the figure shows that the number of

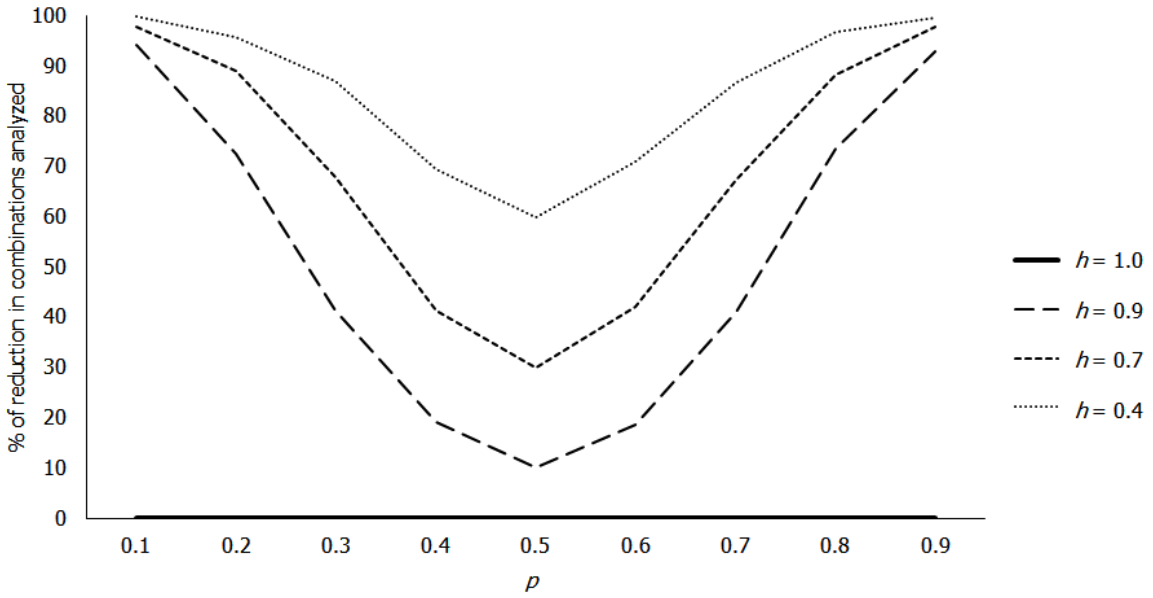


Figure 5.5: Relation between p and h and its effect on the number of workload scenario combinations analyzed for SLP

combinations analyzed increases as the alternative workload scenarios for a given application are equally likely to occur. This happens when the value of p is close to the middle of the x-axis in Figure 5.5, e.g., 0.4, 0.5 and 0.6. As a result, many of the combinations in the exhaustive list of 1024 analyzed in this experiment will have an equal likelihood of occurrence. Therefore, a large number of combinations will be analyzed to achieve a given value of h. However, if the value of p is close to any of the two edges of Figure 5.5, e.g., 0.1, 0.2, 0.8 and 0.9, then one of the two alternative workload scenarios will occur more likely than the other one. As a result, some combinations in the exhaustive list of 1024 will have much higher likelihood of occurrence than other combinations. The combinations with the higher likelihood of occurrence can be sufficient to achieve a given value of h.

The effect of selecting selecting different values for p and h on the number of iterations of the Monte Carol simulation algorithm is now explored for the same experiment shown in Figure 5.5. This number of iterations affects the selection of the *Runlength* parameter of the algorithm which is described previously in Section 5.1. Figure 5.6 shows the number of iterations required by the algorithm to achieve different given values of h for various values

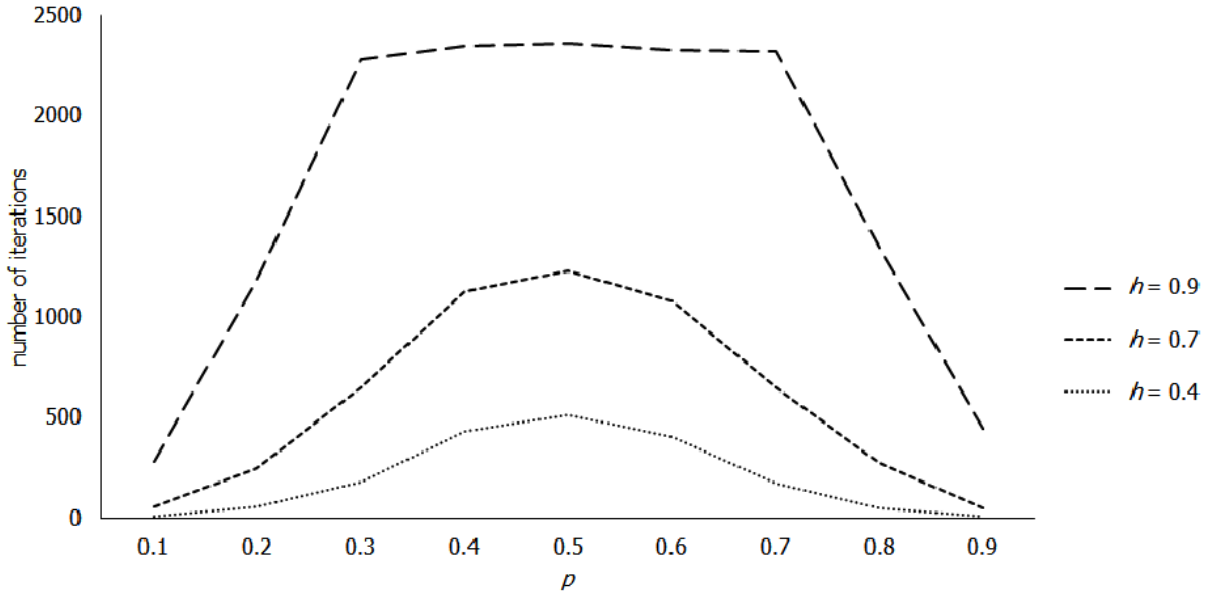


Figure 5.6: Number of iterations of the Monte Carol simulation algorithm for different values of p and h

of p. The figure gives some guidelines on how to set the Runlength value for given values of h and p. As shown in the figure, increasing the value of h requires running the algorithm for greater number of iterations. This is expected because more distinct combinations of workload scenarios should be explored to achieve a higher value of h. It can also be noticed from the figure that, for a given value of h, the number of iterations required increases as the value of p is close to the middle of the x-axis. This is because more combinations are analyzed for these range of values of p as described previously, thereby, more iterations are required. On the other hand, less iterations are required when the value of p is close to any of the two edges of the x-axis in the figure. This is also because less combinations are analyzed for these range of values of p as described previously. In summary, the selected values of h and p can give general guidelines on the computation time of the algorithm.

5.3 Summary

This chapter provides a detailed description of the workload uncertainty module. Firstly, it presents a detailed description of the Monte Carlo simulation technique employed by the uncertainty module. Secondly, results which illustrate the utility and sensitivity of the Monte Carlo simulation technique to various parameters are shown. The results show that the Monte Carlo simulation technique allows cloud SPs to accurately estimate the impact of workload uncertainty in their SLP exercises without exhaustively traversing all combinations of application workload scenarios. Furthermore, the results illustrate the utility of the uncertainty module using different probability values of encountering the heavy load workload scenario. The results also show the impact of relaxing the constraints on the resources available for allocation to applications on the results obtained by the the Monte Carlo simulation technique.

An analysis is also conducted to show the sensitivity of the Monte Carlo simulation technique to the value of the certainty threshold when this value changes from 1.0 to 0.9. This analysis shows that the results obtained with both values are very close. Finally, a sensitivity analysis is conducted to study the effect of selecting different values of h and p on the number of combinations of workload scenarios analyzed for SLP and on the number of iterations of the Monte Carlo algorithm. The results show that more combinations are analyzed and consequently, more iterations need to run for higher values of h. Furthermore, for a given value of h, more combinations are analyzed and more iterations are required for values of p that make the combinations equally likely to occur.

Chapter 6

Burstiness-Aware Workload Clustering Technique

This chapter provides a detailed description of the burstiness-aware workload clustering technique described briefly in Section 1.3.3. Section 6.1 gives an overview of the two steps of the clustering technique namely, cluster generation and clustered workload scenarios generation. Section 6.2 describes the cluster generation step while Section 6.3 describes the clustered workload scenarios generation step. Finally, Section 6.4 shows results which evaluate the clustering technique.

6.1 Overview

As described previously in Section 5.1, The workload uncertainty module generates combinations of workload scenarios where each combination has exactly one probable workload scenario for each application. Recall from Section 3.3 that the the time taken to arrive at a resource allocation plan for a workload scenario combination using each RAP variant depends on the number of times WAM-QNM has to be invoked. Specifically, as shown previously in Table 3.1, the number of invocations of WAM-QNM made by each RAP variant is a function of one or more of the following parameters depending on the RAP variant used:

- the number of applications A hosted on the cloud,
- the number of resource allocation intervals T in the planning horizon
- and the capacity limits $C_{max,n,t}$ of resource instances for any given tier n over any given resource allocation interval t. This parameter determines the maximum number of decision stages as described previously in Section 4.4.

Furthermore, if more than one flavour of resource instance is supported at the bottleneck tier of a given application, then WAM-QNM has to be invoked for each flavour at this tier. This increases the computation time required by each RAP variant.

The processing time can hence be prohibitively high while conducting SLP exercises for large scale clouds, e.g., those that host several hundreds of applications. As mentioned previously in Section 1.3.3, to address this scalability problem a workload clustering technique is developed. The objective of this technique is to group application workload scenarios with similar workload characteristics together. In particular, this technique considers similarity with respect to application resource usage and request arrival patterns. This grouping tends to compact the number of workload scenarios in a single combination from the number of applications, A, to a smaller number of clusters denoted by E.

The clustering technique performs two main steps namely, cluster generation and clustered workload scenarios generation. It is well-known that a large number of workload attributes makes it difficult to detect clusters. Consequently, a hierarchical approach [50, 73] is applied in the cluster generation step. Each level of the hierarchy uses the k-means clustering technique [50, 70] described previously in Section 2.5 to assess similarity based on a single workload characteristic. As described previously, in each level of clustering, the ratio between inter-cluster and intra-cluster distances reported by k-means clustering is used to automatically determine the optimal number of clusters [76]. Clustering at any given level is applied on the clusters obtained from the previous level. In the clustered workload scenarios generation step, workload scenarios, i.e., cluster traces, are generated to represent the clusters generated in the cluster generation step. Both steps are described in more detail in the following sections.

6.2 Generating Clusters

The cluster generation step starts by assessing similarity of the resource traces followed by similarity of the arrival traces. Figure 6.1 shows the inputs, outputs and attributes used to generate a hierarchy of clusters in this step. The reader can refer to Section 4.1 for information on the parameters used in the figure. As shown in the Figure 6.1, the cluster generation step takes as input a combination of workload scenarios $m \in M$ generated from the workload uncertainty module as described previously in Section 5.1. This combination contains one probable workload scenario $W_{a,k}$ for each application a. The workload scenarios in the combination m are first clustered with respect to the per application tier n resource usage, i.e., mean demand $[D_{a,k,1}, D_{a,k,2}, ...D_{a,k,n}]$ where $n \in \{1,2,...N_a\}$. For each of the clusters obtained from above, a clustering is done next in a number of levels with respect to the session trace as follows:

- mean number of requests per session, i.e., mean of the empirical session length distribution $F_{a,k}$,
- mean think time between session requests, i.e., the mean of the empirical think time distribution $Z_{a,k}$,
- mean request arrival rate $\lambda_{a,k}$, i.e., workload intensity, over the planning horizon,
- and arrival burstiness represented by the computed index of dispersion $I_{a,k}$ of session inter-arrival times for application a.

The final level of clustering attempts to group workload scenarios based on fine timescale session arrival patterns. Specifically, a vector of T elements is generated for each workload scenario $W_{a,k}$. Each element $\lambda_{a,k,t}$ in this vector is the mean session arrival rate at a resource allocation interval $t \in \{1,2,...T\}$ for this workload scenario. Workload scenarios are grouped together based on similarities with respect to this vector.

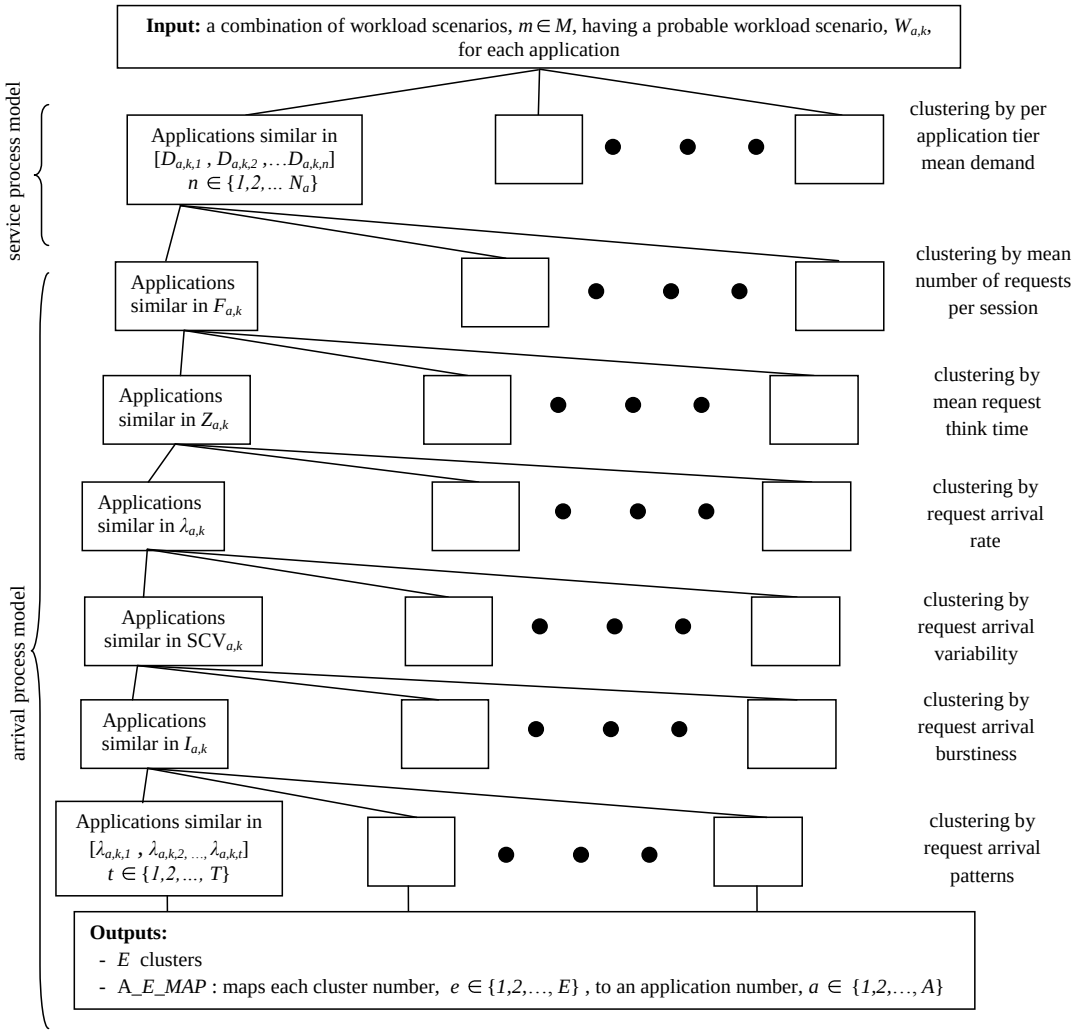


Figure 6.1: Hierarchy of clusters generated by the cluster generation step

The outcomes of this hierarchical clustering approach are the number of clusters E generated and a lookup table denoted by A_E_MAP to map each cluster number $e \in \{1,2,...E\}$ to an application number $a \in \{1,2,...A\}$. It has been found that clustering based on these attributes is effective for grouping similar application workloads together. Furthermore, it has been noticed that the accuracy of SLP exercises improved with such a hierarchical approach when compared to simpler approaches as will be shown later in Section 6.4. It has also been noticed that this hierarchical clustering approach can use any other order of workload attributes provided that each level of clustering generate the number of clusters that minimizes the inter-cluster to intra-cluster distances ratio reported by k-means clustering, as described previously in Section 2.5.

6.3 Generating Workload Scenarios for Clusters

In the clustered workload scenarios generation step, a clustered workload scenario is generated to represent each cluster e. Figure 6.2 shows the pseudo-code of the algorithm to generate a clustered workload scenario. The algorithm takes as input a combination of workload scenarios $m \in M$ under consideration, the number of clusters E, and the lookup table, A_E_MAP, generated from the cluster generation step. The algorithm generates a workload scenario, $W_e \in W_{clustered}$, to represent each cluster $e \in \{1,2,...E\}$. $W_{clustered}$ is the set of all clustered workload scenarios to represent all E clusters. Each W_e is characterized by a service process model denoted by SM_e and an arrival process model denoted by AM_e.

Equivalent resource traces are generated first for a cluster. After experimentation, a simple strategy of selecting resource traces of a random application in a cluster to represent the entire cluster seemed to yield good results. More formally SM_e is selected to be the service process model $SM_{a,fa}$ of any application, a whose workload scenario fa is mapped to cluster e. A similar approach is applied to generate session lengths and think times for the clustered workload scenario.

```
Input: m ∈M, E, A_E_MAP
Output: W_clustered
W_clustered = { }
For e = 1 to E
    For a = 1 to A
        If a maps to e
          # set the service process model SM_e of cluster e to be
          # the service process model of any application in the cluster
          SM_e = SM_a,fa
          # set the session length and think time distributions of cluster e
          # to be the same like those for any application in the cluster
          F_e = F_a,fa
          Z_e = Z_a,fa
          Break;
        End If
    End For
    # AW is the total number of arrival windows
    For aw = 1 to AW
         # generate an empirical PDF of all session arrivals in cluster e
         #  at arrival window aw like the example shown in Fig. 7
        PerIntervalPDF[aw] = GenerateEmpiricalPDF()
        # Use the generated PDF to add arrival samples to the arrival process model AM_e of cluster e
         AM_e = AddArrivalSamples(PerIntervalPDF[aw])
     End For
    # add session length and think time distributions to the arrival process model AM_e of cluster e
    AM_e = { AM_e, F_e, Z_e}
   #generate a workload scenario for cluster e
   W_e= {AM_e, SM_e}
   Add W_e to W_clustered
End For
Return W_clustered
```

Figure 6.2: Clustered workload scenario generation algorithm

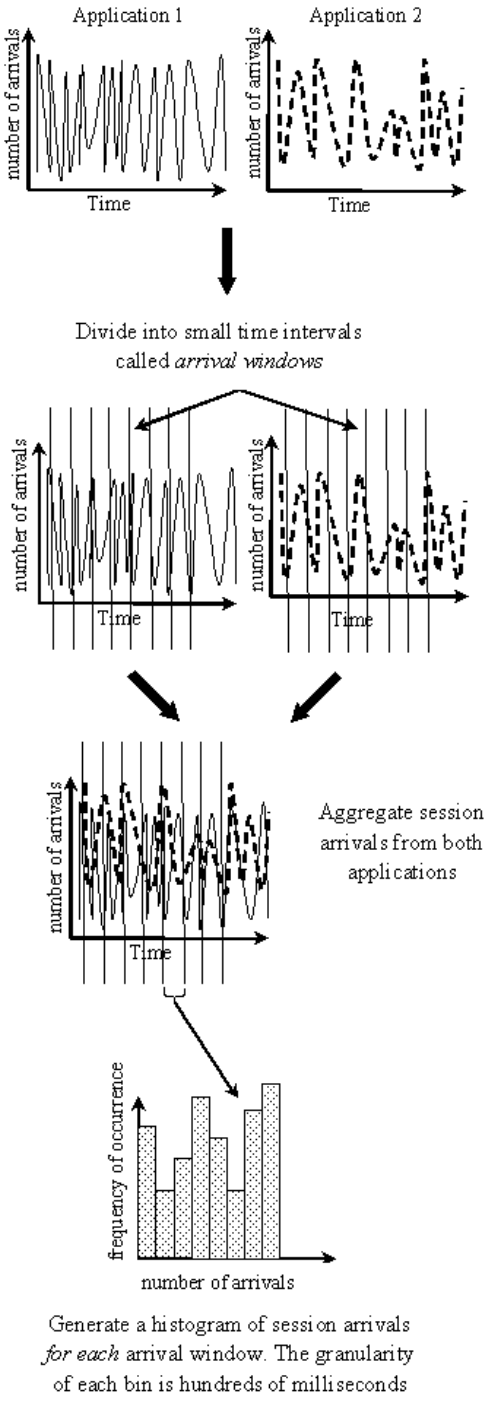


Figure 6.3: Generation of a cluster session arrival trace

However, experiments indicated that this random selection strategy was successful since burstiness in service demands as well as session lengths and think times are not considered in this study. Very poor results are obtained when such a strategy is applied to the session arrivals which are characterized by burstiness. Therefore an algorithm is devised to overcome this problem as illustrated in Figure 6.3.

As described previously, the combination of workload scenarios m under consideration contains the un-clustered workload scenarios of all applications. The group of workload scenarios in combination m that belongs to a certain cluster e are identified first. The session arrival traces of the workload scenarios that map to this cluster are then used to generate AM_e. Every arrival trace that maps to this cluster is divided into very small time intervals called *arrival windows*. To preserve fine-grained burstiness characteristics present in the original traces, each arrival window is selected to be an order of minutes, e.g., 1 min. Every session trace that maps to the cluster is divided into AW arrival windows and a list of session arrivals is maintained for each arrival window $aw \in \{1,2,...AW\}$. This information is used to construct an empirical PDF of session arrivals for that trace for each arrival window. Specifically, a histogram of session arrivals is constructed for each arrival window in which the histogram bins are selected to be in the order of hundreds of milliseconds. Tuning the size of the arrival window and the histogram bin is left for future work. Figure 6.3 shows a simple example of this histogram generation for a cluster of two applications. The empirical PDF of each arrival window is referred to in the pseudo code of Figure 6.2 as *PerIntervalPDF[aw]*. This empirical PDF is then used to generate random samples of session arrivals for each arrival window.

The resource usage and session arrival traces constructed in this manner for the clusters are used by RAP as described previously in Section 3.3.3. Experiments that evaluate the effectiveness of the clustering approach are presented in the next section.

6.4 Evaluation Results

This section discusses evaluation results of the clustering technique. An experiment is conduced where a combination of 100 application workload scenarios is used. For this combination, the resource allocation estimates obtained using RAP *with* and *without* clustering are compared. All 100 applications have similar mean demands at the two application tiers; 20 and 10 ms, respectively. The arrival processes for these applications are generated so as to get coverage of various mean session arrival rates and index of dispersion of session inter-arrival times. Although the workload scenarios are synthetically generated, the clustering technique does not assume any prior knowledge of their characteristics. All parameters characterizing each workload scenario such as arrival rate, SCV and index of dispersion of session inter-arrival times are calculated by analyzing the traces involved. While the calculation of mean arrival rate and SCV of session inter-arrival times is quite straightforward the calculation of the index of dispersion of session inter-arrival times is a challenging task because of the infinite summation that needs to be computed as described previously in Section 2.2. The algorithm shown previously in Figure 2.1 is used to compute the index of dispersion of session inter-arrival times.

Since the mean demands of all applications is assumed to be the same, the first level of clustering shown previously in Figure 6.1 starts directly with similarities in mean session arrival rate (*Level 1*) followed by a second level (*Level 2*) of clustering based on index of dispersion of session inter-arrival time and finally the third level (*Level 3*) of clustering based on a vector of mean session arrival rates for eight 1-hour intervals where the eight hours represents the planning horizon.

Specifically, three different ways of performing the clustering are evaluated. *Level 1* ignores burstiness and clusters only with respect to the mean session arrival rate. The (*Level 1 + Level 2*) approach hierarchically clusters based on *Level 1* (arrival rate) followed by *Level 2* (index of dispersion). Finally, (*Level 1+Level 2+Level 3*) represents hierarchical

clustering as described in the previous paragraph.

The 100 application workload scenarios are clustered using these three approaches. For each method, traces corresponding to the clustered workload scenarios are obtained using the algorithm shown in Figure 6.2. In each level of clustering the clustered workload scenarios generated are input to the RAP algorithm shown previously in Figure 3.1 to obtain an estimate of the number of web server instances and database instance cores allocated per 1-hour interval. Let the numbers of web server instances and database instance cores estimated by a clustering method for interval t be denoted by $WebEst_{t,clustering}$ and $DBEst_{t,clustering}$, respectively. These values are compared with the corresponding estimates obtained from RAP when the original *un-clustered* application traces are input to the method. These corresponding estimates are denoted by $WebEst_{t,no\_clustering}$ and $DBEst_{t,no\_clustering}$, respectively. The absolute differences between the resource estimates obtained with and without clustering normalized by the resource estimate obtained without clustering are used to compute an error measure as shown in the following equations:

$$WebClustErr = \frac{|WebEst_{t,no\_clustering} - WebEst_{t,clustering}|}{WebEst_{t,no\_clustering}} * 100\% \tag{6.1}$$

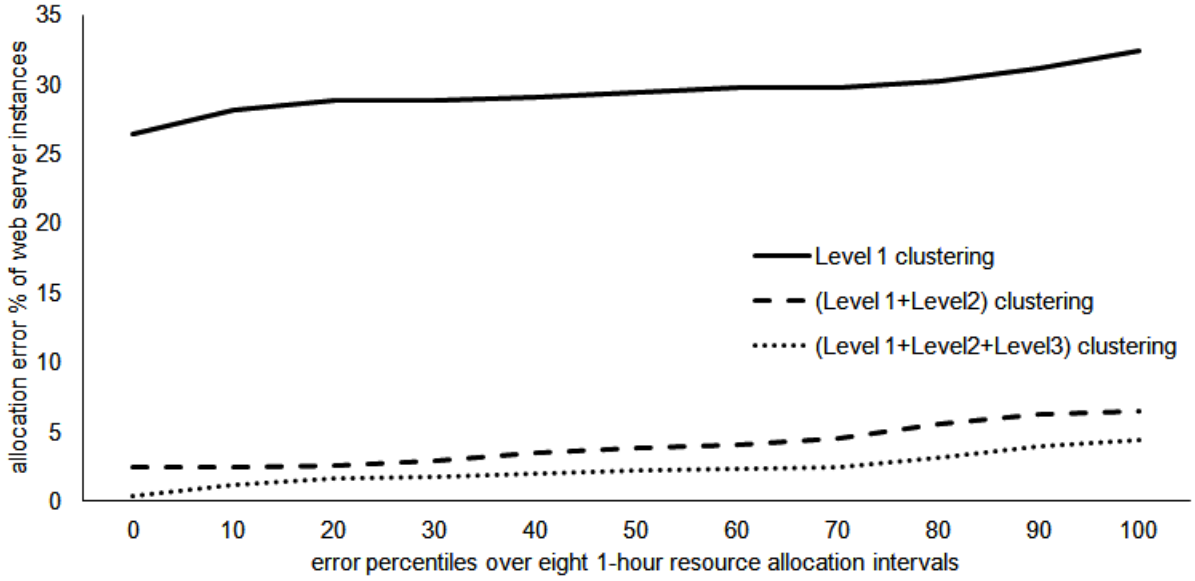
$$DBClustErr = \frac{|DBEst_{t,no\_clustering} - DBEst_{t,clustering}|}{DBEst_{t,no\_clustering}} * 100\% \tag{6.2}$$

Figure 6.4 shows the resource allocation error percentiles of per-interval clustering errors for the three different methods of clustering. The figure shows the resource allocation error percentiles for the web server instances and database instance cores. As shown in Figure 6.4, hierarchical clustering using all three levels yields lower errors. Specifically, *Level 1* approach yields errors in the range of 25% to 35% and 35% to 40% for the web server instances and database instance cores, respectively. (*Level 1+Level 2*) approach reduces errors significantly. (*Level 1+Level 2+ Level 3*) clustering gives slightly better errors than (*Level 1+Level 2*) clustering. In summary, these results demonstrate that the proposed clustering technique that preserves complex burstiness characteristics present in application

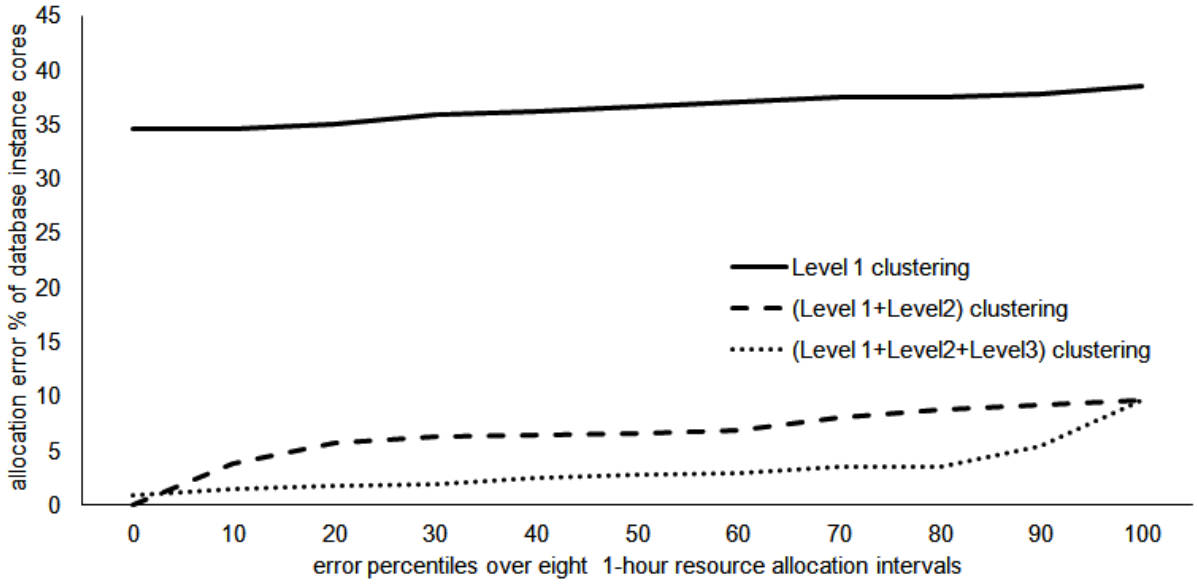
traces is more accurate than burstiness-agnostic clustering approaches.

Figure 6.5 compares the number of clusters generated and the number of WAM-QNM invocations made by RAP in each level of clustering with the the number of clusters and number of WAM-QNM invocations without clustering for the experiment described above. The left y-axis of the figure shows the number of clusters or applications analyzed for SLP while the right y-axis shows the number of WAM-QNM invocations made by RAP. As shown in the figure, if no clustering is applied at all, 100 application workload scenarios are analyzed with 1405 WAM-QNM invocations made by RAP. This is reduced significantly by using clustering. Specifically, the number of clusters analyzed by using *Level 1* clustering is only 4 clusters which consequently reduces the number of WAM-QNM invocations to 44. However, as shown previously in Figure 6.4, *Level 1* clustering gives poor resource allocation accuracy. On the contrary, both (*Level 1+Level 2*) and (*Level 1+Level 2+ Level 3*) clustering approaches give better resource allocation accuracies than *Level 1* clustering but they increase the number of clusters analyzed and the number of WAM-QNM invocations. However, both (*Level 1+Level 2*) and (*Level 1+Level 2+ Level 3*) clustering approaches can still provide much lower number of clusters and number of WAM-QNM invocations when compared with the case where no clustering is used at all.

In summary, Figures 6.4 and 6.5 show a trade-off between the accuracy of resource allocation and the compactness of clusters. Given the workload scenarios considered in this experiment, (*Level 1+Level 2*) clustering provides a good compromise between the number of clusters analyzed and, hence, the number of WAM-QNM invocations required on one side and the accuracy of resource allocation on the other side. Specifically, (*Level 1+Level 2*) clustering gives close to 8% reduction in the number of WAM-QNM invocations relative to the case where no clustering is used at all while providing a maximum resource allocation error of only 10%. It should be noted that this dramatic reduction in the number of clusters and WAM-QNM invocations while still preserving very low resource allocation errors is highly



(a) web server instances



(b) database instance cores

Figure 6.4: Accuracy of resource allocation for three different methods of clustering

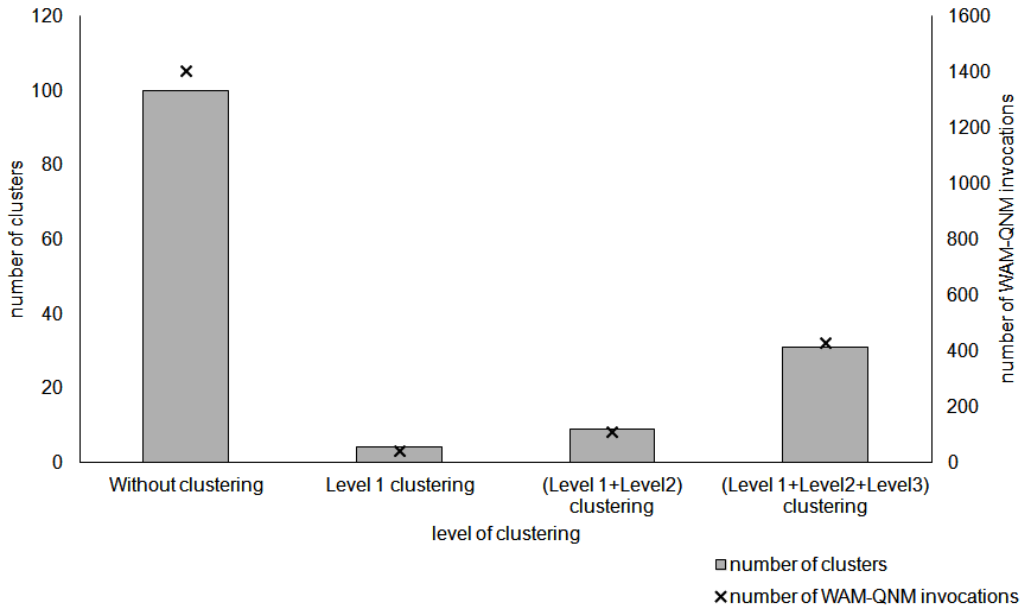


Figure 6.5: Comparing number of clusters and number of WAM-QNM invocations with and without clustering

dependent on the degree of similarity between the characteristics of the workload scenarios under study. This will be analyzed in more detail in the following experiment.

The next experiment demonstrates how workload similarity across applications impacts clustering effectiveness and hence the scalability of the framework. Specifically, three different combinations of workload scenarios are explored. The first combination contains 100 workload scenarios having three different mean session inter-arrival time alternatives and six different index of dispersion of session inter-arrival time alternatives. As a result, this combination has 18 different alternatives with respect to these arrival process attributes. The second combination of workload scenarios consists of the same number of applications, i.e., 100 workload scenarios, but has only 10 different alternatives of session arrival processes. The third combination of workload scenarios contains 5 times the number of applications, i.e., 500 workload scenarios, but has the same session arrival process alternatives as the second combination of workload scenarios.

Figure 6.6 shows the number of clusters generated from the clustering module for the

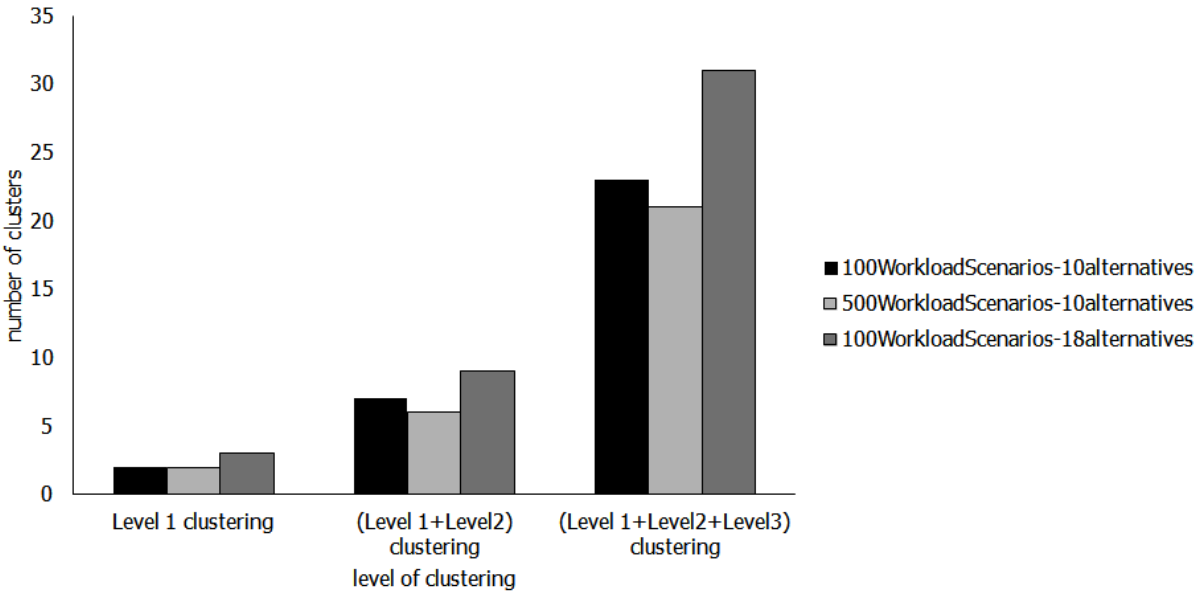


Figure 6.6: Impact of clustering on SLP framework scalability

three methods of clustering for each of the three combinations of workload scenarios described in the previous paragraph. It can be observed from the figure that the number of clusters generated depends only on the diversity of workload scenarios and not on the number of workload scenarios themselves. For example, both the *100WorkloadScenarios-10alternatives* combination and *500WorkloadScenarios-10alternatives* combination reduce to approximately the same number of clusters for any given clustering method. In contrast, the more diverse *100WorkloadScenarios-18alternatives* combination results in more number of clusters than the *500WorkloadScenarios-10alternatives* combination. This suggests that the SLP framework described briefly in Section 1.3.4 can scale well to a large number of applications if there are natural groupings of applications with similar workload patterns. Chen *et al.* [45] characterized CPU and memory utilization traces for 1173 applications covering 8 days of activity at a real enterprise system. The authors found that many of these applications have similar resource utilization patterns. This suggests that the proposed clustering technique can be effective in ensuring the scalability of the SLP framework.

6.5 Summary

This chapter provides a detailed description of the burstiness-aware clustering technique proposed in the thesis. Two main steps of the clustering technique are described namely, cluster generation and clustered workload scenario generation. Results which evaluate the accuracy of the clustering technique are shown. The results show that the accuracy of resource allocation is increased by increasing the levels of clustering and the number clustering attributes considered. Furthermore, the results show the need to implement automatized workload scenario generation algorithms to reconstruct a workload scenario for each cluster to obtain accurate resource allocation plans. Finally, the results show that the proposed workload clustering technique reduces the number of computations needed to support SLP exercises without significantly impacting accuracy. The degree of reduction in the number of computations depends on the degree of similarity between the workload scenarios of the applications considered for SLP.

Chapter 7

Trace-Based SLP Framework

This chapter provides more details on the SLP framework introduced previously in Section 1.3.4. Section 7.1 describes the modifications made for the RAP method to be integrated in the SLP framework. The modifications allow RAP to perform SLP for a given set of application clusters rather than for a given set of applications as described previously in Section 3.3.3. Section 7.2 discusses the computational complexity of the SLP framework by presenting some statistics about the end-to-end case study introduced previously in Section 1.3.5.

7.1 Modifying RAP to Account for Clusters of Applications

This section describes some assumptions and modifications made on RAP to account for clusters of applications in SLP. The RAP algorithm described previously in Section3.3.3 generates a resource allocation plan for a given set of applications characterized by application workload scenarios. To integrate RAP with the clustering technique described previously in Chapter 6, RAP is modified to accept workload scenarios for clusters of applications and thus resources are allocated at the level of clusters instead of applications. The necessary assumptions and modifications to RAP are described in the ensuing paragraphs.

The SLO of a cluster, e, is denoted by SLO_e. For ease of explanation, all applications in the same cluster are assumed to have the same SLO. Therefore SLO_e is assumed to be the same as the SLO_a of any application, a, that belongs to cluster e. This assumption can be relaxed by adding a top most level to the hierarchical clustering algorithm described in Section 6.2. This level will cluster applications by their SLO requirements before clustering based on the lower clustering levels.

It is also assumed that all applications in the same cluster have the same number of application tiers. Therefore the number of tiers for cluster e, denoted by N_e, is set to be the same as the number of tiers, N_a, of any application, a, that belongs to cluster e.

The pseudo-code of the modified RAP algorithm is shown Figure 7.1. The inputs to the algorithm are as follows:

- the set, $W_{clustered}$, of clustered workload scenarios generated by the clustering technique as described previously in Section 6.3,
- the SLO for each application a, SLO_a, over the planning horizon,
- the maximum number of resources, $C_{max,n,t}$, available for allocation to all applications at tier n in each interval t,
- the number of clusters, E, and the lookup table, A_E_MAP by the clustering technique.

Initially each clustered workload scenario representing a cluster, e, is allocated a number of resources equaling the number of applications in the cluster at each tier, n, and for each resource allocation interval, t. WAM-QNM is then invoked to predict the mean response time, R_e, of cluster e over the planning horizon and the mean response time, $R_{e,t}$, of cluster e over each resource allocation interval t. After this step the violation percentage, V_e, is calculated for each cluster, e.

The RAP algorithm then enters a loop. In each iteration of this loop, the cluster e_{max} with the top most SLO violation percentage, i.e., maximum V_e, is selected first. For the selected cluster, e_{max}, the resource allocation interval, t_{max}, with maximum $R_{e,t}$ over all T resource allocation intervals is selected. The bottleneck application tier $n_{bottleneck}$ is then determined for e=e_{max} and t=t_{max}. Finally additional resources are allocated to cluster e_{max} in tier $n_{bottleneck}$ at resource allocation interval t_{max}. To take into account the impact of clustering many applications together, the number of resources allocated for t_{max} is equal to

Input: $W_{clustered}$,SLO_a $\forall$ $a \in \{1,2,\ldots, A\}$, $C_{max,n,t} \forall$ $n \in \{1,2,\ldots, N_a\}$, $t \in \{1,2,\ldots, T\}$, E, A_E_MAP
Output: $C_{e,n,t}$ $\forall$ $e \in \{1,2,\ldots, E\}$, $n \in \{1,2,\ldots, N_e\}$, $t \in \{1,2,\ldots, T\}$, V_e $\forall$ $e \in \{1,2,\ldots, E\}$
For $e = 1$ ***to*** E
since all applications in same cluster have same SLO set cluster SLO_e to SLO_a
$SLO_e = SLO_a$ *for any* $a \rightarrow e$
since all applications in same cluster have same number of application tiers
$N_e = N_a$ *for any* $a \rightarrow e$
in each cluster e allocate initial resources to each tier n in each interval t
the number of resources allocated to each tier n in each interval t
equals the number of applications in the cluster
$C_{e,n,t} = GetNumberOfAppsPerCluster(e, A\_E\_MAP)$ $\forall$ $n \in \{1,2,\ldots, N_e\}$, $t \in \{1,2,\ldots, T\}$
invoke the performance model for each clustered workload scenario $W_e \in W_{clustered}$, $e \in \{1,2,\ldots,E\}$
$[R_{e,t}, R_e] = WAM\text{-}QNM\ (W_e, C_{e,n,t}\ \forall\ n \in \{1,2,\ldots, N_e\})$ $\forall$ $t \in \{1,2,\ldots, T\}$
$V_e = CalculateClusterViolationPercentage(R_e, SLO_e)$
End For
#calculate total number of remaining resources at tier n over all intervals
this is used to stop the algorithm once all remaining resources are allocated
For $n = 1$ ***to*** N_e
$RemainResourcesTier(n) = C_{max,n,t} * T - (\sum C_{e,n,t} \forall\ e \in \{1,2,\ldots, E\}, t \in \{1,2,\ldots, T\})$
End For
While $(RemainResourcesTier(n) > 0\ \forall\ n \in \{1,2,\ldots, N_e\})$
$[e_{max}, MaxViolationPercentage_{,}] = GetClusterwithMaxViolationPerentage(V_e \forall\ e \in \{1,2,\ldots, E\})$
if all clustered workload scenarios satisfy their violation percentages
If $MaxViolationPercentage_{,} == 0$
Return $C_{e,n,t} \forall\ e \in \{1,2,\ldots, E\}, n \in \{1,2,\ldots, N_e\}, t \in \{1,2,\ldots, T\}, V_e\ \forall\ e \in \{1,2,\ldots, E\}$
Where $V_e = 0 \forall\ e \in \{1,2,\ldots, E\}$
End If
$t_{max} = GetFreeIntervalwithMaxRmean(e_{max}, (R_{e,t}\ for\ e = e_{max}\ ,\ \forall\ t \in \{1,2,\ldots, T\}))$
$n_{bottleneck} = GetBottleneckTier(e_{max}, t_{max})$
allocate additional resources to tier n of cluster e_{max} in interval t_{max}
the number of additional resources allocated to each tier n in each interval t
equals to the number of applications in the cluster e_{max}
$\Delta\ C_{e,n,t} = GetNumberOfAppsPerCluster(e, A\_E\_MAP)\ for\ e = e_{max}\ ,\ n = n_{bottleneck},\ t = t_{max}$
$C_{e,n,t} = C_{e,n,t} + \Delta\ C_{e,n,t}$ *for* $e = e_{max}$, $n = n_{bottleneck}$,$t = t_{max}$
$RemainResourcesTier(n_{bottleneck}) = RemainResourcesTier(n_{bottleneck}) - \Delta\ C_{e,n,t}$
#re-invoke performance model for the cluster e_{max} only
$[R_{e,t}, R_e] = WAM\text{-}QNM\ (W_e, C_{e,n,t}\ \forall\ n \in \{1,2,\ldots, N_e\})$ *for* $e = e_{max}$, $\forall\ t \in \{1,2,\ldots, T\}$,
$V_e = CalculateClusterViolationPercentage(R_e, SLO_e)$ *for* $e = e_{max}$
End While
Return $C_{e,n,t} \forall\ e \in \{1,2,\ldots, E\}, n \in \{1,2,\ldots, N_e\}, t \in \{1,2,\ldots, T\}, V_e \forall\ e \in \{1,2,\ldots, E\}$

Figure 7.1: Modified RAP algorithm to account for clusters of applications

Table 7.1: End-to-end case study statistics

	no uncertainty or clustering	**only uncertainty**	**uncertainty and clustering**
workload scenario combinations	32	8	8
average scenarios per combination	100	100	10
number of WAM-QNM invocations	44,960	11,240	1,016

the number of applications in the cluster e_{max}. As described previously in Section 3.3.3, the algorithm terminates when either all the available resources are allocated in all intervals or the maximum V_e is equal to zero.

It should be noted that exploring resource allocations at the level of clusters instead of the level of applications as described previously in Section 3.3.3 can affect the optimality of the solutions obtained by RAP if the resource allocation error introduced by the clustering technique is high. However, as shown previously in Section 6.4 the proposed clustering technique yields a maximum resource allocation error of only 5%.

7.2 End-to-End Case Study Statistics

This section presents results to show the computational complexity of the SLP framework. Specifically, Table 7.1 shows some statistics about the end-to-end case study introduced previously in Section 1.3.5. Recall from Section 1.3.5 that 100 applications are considered in this case study with 95 applications have only one workload scenario, while the other 5 applications have an additional heavy workload scenario with a probability of occurrence p = 0.1. A certainty threshold, h, value of 0.9 is used.

The combined use of the uncertainty and clustering modules decreases the number of performance model invocations from 44,960 to 1,016 thereby significantly reducing analysis time. The uncertainty module achieves a 75% reduction in the number of workload combinations to be analyzed. The clustering module reduces the number of workload scenarios

per combination by up to 90%. It should be noted that gains from the uncertainty and clustering modules may not be always as dramatic as obtained in this case study. For example, as described previously in Section 5.2 such dramatic reduction will not occur when all alternative workload scenarios for an application are equally likely to occur, i.e, p=0.5. Furthermore, as described previously in Section 6.4 the degree of reduction achieved by the clustering module depends on the level of similarity between application traces.

From the statistics shown in Table 1.2, it is clear that the analysis time depends heavily on the execution time of WAM-QNM. The computation of the analysis time can be obtained given that a single WAM-QNM invocation takes about 3 minutes on a desktop computer with a 3 GHz duel-core Intel Pentium processor and 2 GB RAM. Analysis time can benefit from several optimizations. One can merely invoke MVA-QNM for workload scenarios that are not characterized by burstiness. Each MVA-QNM invocation runs in an order of magnitude of milliseconds which reduces the analysis time significantly. Furthermore, the performance metrics predicted by WAM-QNM can be computed before running the SLP framework and tabulated for various resource allocation plans for given combinations of application workload scenarios. Table lookups can then be used when a certain resource allocation plan is encountered while executing RAP to fetch the corresponding performance metrics predicted by WAM-QNM. Implementing these optimizations is left for future work.

7.3 Summary

This chapter describes the modifications made on the RAP method to be integrated with the uncertainty module and the clustering technique in the SLP framework. RAP is modified to perform SLP for clusters of applications. Results which show the computational complexity of the SLP framework are shown. Ways to optimize the performance of the framework are also proposed.

Chapter 8

Conclusions and Future Work

This chapter summarizes the conclusions and findings obtained from the work in the thesis. It also discusses the limitations of the SLP framework and outlines future work. Section 8.1 summarizes the conclusions. Section 8.2 discuses the limitations of the SLP framework. Finally, Section 8.3 outlines future research directions.

8.1 Summary and Conclusions

Cloud SPs need tools that can help them optimize the allocation of their resources and decide on service level agreements with customers. The thesis presents an SLP framework to optimally allocate cloud resources to customer applications while satisfying their SLOs. The framework addresses a number of challenges. Firstly, SLP tools need to take into account the complex nature of enterprise application workloads such as burstiness inherent in such systems. Specifically, the elastic nature of the cloud can be exploited to address workload burstiness. However, in order to leverage this benefit of the cloud, SLP tools should be able to accurately predict how many resources each application should get and how the number of allocated resources should change over time to match workload patterns.

Secondly, cloud SPs need methods to assess the risks that arise due to workload uncertainty. The workload experienced by many applications may deviate significantly from the workload projections estimated by application owners. Such applications may encounter many possible workload scenarios with each placing significantly different capacity requirements on the cloud. A systematic approach is required to accommodate workload uncertainty in SLP exercises. This approach needs to evaluate how workload uncertainty will affect the penalties which a cloud SP might incur due to SLO violations.

Finally, the need to consider a large number of customer applications typically deployed on the cloud with each characterized by multiple workload scenarios due to workload uncertainty. As described previously, SLP tools rely on performance models to search for optimal resource allocation strategies. This can take a lot of time with large number of applications and large number of workload scenarios per application. Furthermore, advanced performance models such as WAM-QNM which accurately predict performance under workload burstiness take relatively longer execution time than traditional performance modeling techniques such as MVA-QNM. This can impact the performance of the SLP toolset. Therefore, SLP tools should scale well to support a large number of applications and a large number of workload scenarios per application.

The SLP framework addresses the above issues simultaneously. The framework implements a novel RAP method to identify a time varying allocation of resources to applications that ensures adequate resources are available to an application to satisfy its bursts. A Monte Carlo simulation technique is proposed to accommodate workload uncertainty in the SLP process. A new burstiness-aware workload clustering technique is proposed to increase the scalability of the SLP exercises relying on the SLP framework while preserving complex workload characteristics observed in application workload scenarios.

The thesis presents detailed simulation results to characterize the behaviour of individual components of the SLP framework as well as its end-to-end behaviour. Firstly, the results confirmed previous findings in [64] that the WAM-QNM which is employed in this work can more accurately predict SLO violations under workload burstiness than other traditional performance modeling techniques such as MVA-QNM.

Secondly, three variants of the RAP method are proposed. A detailed analysis of the computational complexity and the optimality of the three variants is also studied. The results show that the three RAP variants can provide a cloud SP with optimal and close to optimal resource allocation plans over a given planning horizon without exhaustively generating all

possible resource allocation plans. Specifically, RAP-DP can identify a plan that minimizes SLO violations for a given cloud resource capacity. It is also shown that RAP-AllApps which is a more computationally efficient version of RAP-DP is able to identify optimal solutions when bottlenecks do not switch across tiers over the planning horizon. Furthermore, the results show that under both bursty and non-bursty workload scenarios, RAP-OneApp is able to identify near optimal solutions while reducing computational complexity significantly when compared to the optimal RAP-DP variant. Therefore, RAP-OneApp can be used in SLP exercise involving a large number of applications without significantly affecting the optimality of the solutions obtained.

Thirdly, the results show that the proposed RAP methodology can permit cloud SPs to more accurately determine the capacity required for delivering specified SLOs compared to other competing techniques that ignore burstiness. Moreover, the results show that allocating resources over smaller sized intervals than the planning horizon reduces resource allocation costs for bursty workload scenarios.

Fourthly, the results establish the superiority of RAP to the prevalent practice of considering SLO targets based on resource utilization thresholds over the planning horizon. As opposed to the utilization-based approaches the results show that RAP can accurately adapt resource allocation plans to observed burstiness characteristics of application workloads.

Fifthly, simulation experiments are conducted to show the flexibility of RAP to accommodate different techniques for defining application SLOs. The results show that under both bursty and non bursty workload scenarios, defining SLO targets over a planning horizon formed of smaller sized resource allocation intervals reduces resource allocation costs when compared with defining SLOs over individual resource allocation intervals.

Finally, an approach to improve the performance of RAP-DP and RAPAllApps by exploiting parallelism is shown. Specifically, multi-threaded versions of both RAP variants are implemented to reduce their computation times for large number of applications. A machine

with 12 processor cores is used to compare the computation times of the single-threaded and the multi-threaded versions of the two RAP variants. The results show a performance improvement of the multi-threaded versions over the corresponding single-threaded versions by a factor of 9.

Detailed simulation experiments are conducted to characterize the behaviour, utility and accuracy of the proposed Monte Carlo simulation technique. The results demonstrate that the Monte Carlo simulation technique enables cloud SPs to accurately estimate the impact of workload uncertainty in their SLP exercises without the need to exhaustively analyze all combinations of application workload scenarios.

Detailed simulation experiments are also conducted to evaluate the accuracy of the proposed burstiness-aware workload clustering technique. Firstly, the results show that the workload clustering technique is effective in grouping applications with similar burstiness characteristics together. Specifically, the results show the need to consider workload characteristics over entire planning horizon as well as over each resource allocation interval for the clustering approach to be effective under burstiness. The workload characteristics considered for clustering over the planning horizon include mean session arrival rate and index of dispersion of session inter-arrival time. Mean session arrival patterns over each resource allocation interval are also considered in the clustering technique.

Secondly, the results show that the accuracy of resource allocation is increased by increasing the levels of clustering and the number clustering attributes considered. Thirdly, the results show the ability of the proposed workload scenario generation algorithm which is used to reconstruct a workload scenario for each cluster to obtain accurate resource allocation plans. Finally, the results show that the proposed workload clustering technique reduces the number of computations needed to support SLP exercises without significantly impacting accuracy. Therefore the SLP framework can scale well to large number of applications and large number of workload scenarios per application.

8.2 Limitations

In this section the limitations of the SLP framework are discussed while future work required to overcome these limitations is outlined in Section 8.3. Firstly, the SLP framework gives cloud SPs a general guide on the amount of resources required to satisfy applications' SLOs. However it does not give detailed information on the number and type of resource instances to be assigned at each application tier and the number of physical servers to be purchased.

Secondly, as described previously in Section 5.1, the Monte Carlo simulation algorithm employed by the uncertainty module generates a subset of the workload combinations with a total probability of occurrence above the certainty threshold, h. The generated combinations of workloads will occur most of the time, however, they might not be the most heavily loaded combinations. This may introduce some risk of violating SLOs by cloud SPs when the heavily loaded combinations which are not covered in the analysis are encountered in production.

Thirdly, recall from Section 3.1 that the RAP tries to optimize two conflicting cost objectives which are the cost of resource allocation and the cost incurred due to SLO violation. The work in the thesis explicitly addresses only the SLO objective component. Minimizing resource costs is achieved indirectly by exploring only incremental allocations of a unit resource to each application. Fourthly, this work ignores practical constraints such as minimizing the number of resource instance migrations across applications, and supporting multiple flavours of resources for a given tier.

Fifthly, the simulation model employed in the thesis only takes into account the processing power of each resource instance without considering memory, storage or I/O issues. Furthermore, the simulation setup does not use real traces. It is extremely difficult to obtain real traces since cloud SPs consider them to be confidential. This limitation is overcome by generating synthetic workloads with characteristics observed in a real enterprise system as described previously in Section 4.2. A systematic and comprehensive sensitivity analysis is performed by varying properties such as session arrival variability and burstiness.

Finally, this work ignores burstiness in service demands. This is motivated by the lack of robust performance models that can reflect the impact of service demand burstiness. While new methodologies for such systems are emerging [41], they are still not robust enough to handle many common modeling scenarios. When such models do become more prevalent, they can be integrated in a straightforward manner into the SLP framework.

8.3 Future Work

This section discuses directions for future work. Future work will focus on implementing the optimizations described previously in Section 7.2 to reduce RAP execution time. Specifically, a meta-algorithm can be implemented to guide the selection of the performance model and the RAP variant which are more appropriate for a given trace in order to reduce RAP computation time without affecting the accuracy and optimality of the solution obtained.

Firstly, to select the appropriate performance model for a given workload trace, the meta-algorithm can evaluate the degree of burstiness observed in this trace. The degree of burstiness of the trace can be determined by estimating its index of dispersion as described previously in Section 2.2. MVA-QNM can then be invoked for non-bursty workloads to reduce RAP computation time while WAM-QNM can be invoked for bursty workloads. As shown previously in Section 4.3, WAM-QNM obtains more accurate performance predictions for bursty workloads than MVA-QNM while MVA-QNM can still accurately predict the performance of applications characterized by non-bursty workloads.

Secondly, the meta-algorithm can leverage the results of the sensitivity analysis shown previously in Section 4.5 to select the appropriate RAP variant for a given trace. The selection can be done based on analyzing the service demands of the trace at different application tiers. If the service demands of application tiers are different by a specified factor, then the bottleneck tier is likely to be the same in all resource allocation intervals. Consequently, RAP-AllApps can be invoked on this trace to reduce the computation time without sacri-

ficing the optimality of the solution significantly. On the other hand, if the service demands of application tiers are quite close, then the bottleneck tier most likely tends to change over the planning horizon which makes the optimal RAP-DP better for the trace under study.

Future work will also address the limitations identified previously in Section 8.2. Firstly, the SLP framework should be used in conjunction with benchmark results for various types of resource instances on specific hardware platforms to support to multiple flavours of resources for a given application tier. Benchmarks can be used to evaluate various characteristics of resource instances including processing power, memory, storage and I/O issues. This will give cloud SPs detailed information on the resource instances required by their customers' applications which can be directly interpreted into monetary costs.

Secondly, the certainty threshold h can be exploited to to address the risk of not including heavily loaded combinations in SLP analysis. Recall from Section 5.1 that the higher the value of h the more combinations of workload scenarios are covered in the SLP analysis, thereby decreasing this risk. Furthermore, for a given application, an analysis of all of its alternative workload scenarios can be done before invoking the framework. The objective of this analysis is to determine the most heavily loaded workload scenarios to include them in the SLP analysis irrespective of their probability of occurrence.

Finally, RAP can also be modified to accommodate other SLOs. For example RAP can explicitly minimize the number of applications which violate their SLOs. Other than performance-based QoS, RAP can also address other QoS objectives such as the availability issues of resource instances. Furthermore, future work will integrate both SLO and resource allocation costs objectives explicitly in a single objective. Cloud SPs can exploit this feature to reflect the prioritization of one component over the other.

As described previously in Section 1.3.4, the performance model employed by the SLP framework may predict SLO violations that are different from those observed under deployment. To address these discrepancies the model needs to be calibrated to offer better SLO

predictions. Future work can overcome this limitation by implementing a feedback control loop which continuously modifies the performance model with performance data observed during the operation of workloads on the cloud. Examples of such data for a given application include updated traces of session arrivals to the application system and updated traces of service demands at different application tiers.

Another direction for future work is related to the implementation of the multi-threaded version of RAP-DP described previously in Section 4.9. The multi-threaded versions of RAP-DP and RAP-AllApps can be implemented on clusters of machines using parallel programming models such as *Message-Passing Interface (MPI)* [59] to leverage the very large scale parallelism supported by such clusters.

Future research directions can also investigate ways to extend the SLP framework to support energy-aware SLP strategies [25, 34, 98] in green clouds. Energy-aware SLP tools should implement resource allocation methods that improve the energy efficiency of data centers by reducing their carbon emissions while still not violating applications' SLOs with minimal resource costs. This means that a resource instance should be allocated to an application tier of a given application on the physical server that results in the lowest possible emission of carbon dioxide, thereby reducing environmental footprint. Consequently, the global SLO and resource allocation optimization problem described previously in Section 3.1 should be reformulated to address a third objective which is the carbon emissions of resource instances in addition to SLO violations and resource costs.

Finally, the SLP framework can be extended to combine resource instances from multiple clouds with different energy sources. Specifically, resource instances can be provisioned from cloud systems that rely on renewable energy sources such as wind power and solar energy in addition to clouds systems that rely on traditional energy sources. In this way the SLP framework can satisfy the QoS required by cloud applications while providing energy-aware resource management. Future work will investigate this extension and its associated

challenges and solutions.

Appendix A

List of Notations

Table A.1: List of Notations

Notation	Definition
	Workload Scenario Notations
A	number of applications
L_a	number of probable workload scenarios for application $a \in \{1,2,\ldots A\}$
N_a	number of application tiers for application a
T	number of resource allocation intervals of the planning horizon
$S_{a,k}$	number of representative sessions for application a with workload scenario $k \in \{1,2,\ldots L_a\}$
$F_{a,k}$	distribution of number of requests per session for application a with workload scenario k
$Z_{a,k}$	distribution of think time between session requests in application a with workload scenario k
$\lambda_{a,k}$	mean session arrival rate for application a with workload scenario k
$SCV_{a,k}$	SCV of session inter-arrival time for application a with workload scenario k
$I_{a,k}$	index of dispersion of session inter-arrival time for application a with workload scenario k
$AM_{a,k}$	arrival process model for application a with workload scenario k $\{S_{a,k}, F_{a,k}, Z_{a,k}, \lambda_{a,k}, SCV_{a,k}, I_{a,k}\}$
$D_{a,k,n}$	mean service demand of application a with workload scenario k at tier $n \in \{1,2,\ldots, N_a\}$

$SCV_{a,k,n}$	SCV of service demand of application a with workload scenario k at tier n
$I_{a,k,n}$	index of dispersion of the service demand of application a with workload scenario k at tier n
$SM_{a,k}$	service process model for application a with workload scenario k $\{D_{a,k,1},\ D_{a,k,2},\ldots D_{a,k,n}\ \|n \in \{1,2,\ldots N_a\}\}$
$W_{a,k}$	probable workload scenario k for application a $\{AM_{a,k},\ SM_{a,k}\}$
$P_{a,k}$	probability of occurrence of workload scenario $W_{a,k}$
Uncertainty Module Notations	
W_a	set of possible workload scenarios for application a $\{W_{a,1}, W_{a,2},\ ,\ldots,\ W_{a,k}\ \|k \in \{1,2,\ldots L_a\}\}$
P_a	set of probabilities of workload scenarios for application a $\{P_{a,1},\ P_{a,2},\ldots,\ P_{a,,k})\ \|k \in \{1,2,\ldots L_a\},\ \sum_{k=1}^{L_a} P_{a,k} = 1\}$
h	the certainty threshold of the workload uncertainty module
M	set of workload scenario combinations whose probability of occurrences cumulatively sum up to a value h
$m \in M$	a combination of workload scenarios $\{W_{1,f1}, W_{2,f2},,\ldots, W_{a,fa}\|a \in \{1,2,\ldots,\ A\},$ $f1 \in \{1,2,\ldots L_1\},\ f2 \in \{1,2,\ldots L_2\},\ldots fa \in \{1,2,\ldots L_a\}\}$
$Prob_m$	probability of occurrence of a combination of workload scenarios $m \in M$
$Prob_M$	set of probabilities of occurrence of all combinations of workload scenarios in M $\{Prob_m\ \|\forall m \in M\}$
Clustering Module Notations	
$\lambda_{a,k,t}$	mean session arrival rate for application a with workload scenario k in interval $t \in \{1,2,\ldots T\}$
E	number of workload clusters computed by the clustering module

$W_{clustered}$	set of workload scenarios to represent E clusters $\{(W_1,\ W_2, \ldots W_e \ldots W_E)\ \|e \in \{1,2,\ldots E\}\}$
AM_e	arrival process model of cluster e
SM_e	service process model of cluster e
A_E_MAP	look-up table to map each cluster number e to application number a
RAP and Performance Model Notations	
N_e	number of application tiers for cluster e
$MS_{e,t}$	mean number of concurrent sessions for cluster e per interval t
$C_{max,n,t}$	maximum number of resource instances that can be allocated to all applications at tier n at interval t
$C_{a,n,t}$	number of resource instances allocated to tier n of application a at interval t
$C_{e,n,t}$	number of resource instances allocated to tier n of cluster e at interval t
$\Delta C_{e,n,t}$	increment to the number of resource instances allocated to tier n of cluster e at interval t
$R_{a,t}$	mean request response time of application a over interval t predicted by the performance model
$R_{e,t}$	mean request response time of cluster e over interval t predicted by the performance model
R_a	mean request response time of application a overall T intervals predicted by the performance model
R_e	mean response time of cluster e overall T intervals predicted by the performance model
V_a	SLO violation percentage of application a
V_e	SLO violation percentage of cluster e
SLO_a	SLO of application a

SLO_e	SLO of cluster e

Appendix B

Dynamic Programming Formulation of RAP-DP

This appendix presents a formulation for RAP-DP using the dynamic programming concepts [49] described previously in Section 2.6. Specifically, the appendix defines the three dynamic programming terminologies namely, the optimal value function, the recurrence relation and the boundary condition which are used to solve the global SLO and resource allocation optimization problem described previously in Section 3.1 using RAP-DP.

As described previously in Section 3.3.1 RAP-DP implements a process of multiple decision stages to solve the global SLO and resource allocation optimization problem. The optimization problem is first decomposed into simpler subproblems where an optimal solution for each of these subproblems is obtained at each decision stage. The optimal solutions of the subproblems are then combined together to obtain to the optimal solution of the overall problem. Specifically, an optimal resource allocation plan is obtained at each decision stage by allocating one more resource to the bottleneck tier of one of the applications considered for SLP in one resource allocation interval. In this way, the optimal resource allocation plan obtained at any decision stage i aims to mitigate the effect of the worst bottleneck tier among all applications over all resource allocation intervals on the SV value of the applications. Recall from Section 3.1 that the SV value is the sum of SLO violation percentages for all A applications as defined previously by Equation (3.7). The optimal resource allocation plan obtained at decision stage i is then used as a base for further exploration in subsequent decision stages while other non-optimal resource allocation plans are not explored further. The reason for this is that exploring any non-optimal resource allocation plan in subsequent decision stages cannot result in a better reduction in the SV value of the applications when compared to exploring the optimal resource allocation plan.

Each decision stage i has an associated state $state_i$ which describes all the variables that affect the problem solved at this decision stage. Specifically, for the optimization problem solved by RAP-DP, $state_i$ is defined as the number of resource instances of appropriate types that are available for allocation at decision stage i for each of the application tiers of each application in each resource allocation interval. Given the information provided by $state_i$ at decision stage i, a decision $decision_i$ is selected from a set of feasible decisions $Decisions_i$ to obtain the optimal solution for the problem solved at this decision stage. Specifically, for the optimization problem solved by RAP-DP, at each decision stage i the set $Decisions_i$ specifies the different ways of allocating exactly one extra resource instance of an appropriate type to one tier n of an application a in a resource allocation interval t. The decision $decision_i$ alters the state $state_i$ to another state $state_{i+1}$ which describes the state of the subproblem solved at the next decision stage *i+1*.

Given the state $state_i$ at decision stage i, the optimal value function for the optimization problem solved using RAP-DP is defined as:

$$u_i(state_i) = \max[RSV_i(state_i, Decisions_i) + RSV_{i-1}(state_{i-1}, Decisions_{i-1}) \cdots + RSV_0(state_0, Decisions_0)] \tag{B.1}$$

where $RSV_i(state_i,Decisions_i)$ is the reduction obtained in the SV value of the applications achieved at decision stage i by selecting a decision $decision_i$ from the set $Decisions_i$ given the state $state_i$. Equation (B.1) states that the optimal solution for the problem solved using RAP-DP can be obtained at any decision stage i by adding the maximum value of $RSV_i(state_i,Decisions_i)$ obtained at each prior decision stage starting from decision stage i to 0. The solution obtained for $u_i(state_i)$ at any decision stage i is guaranteed to be optimal because the value $RSV_i(state_i,Decisions_i)$ at any decision stage i is optimal with respect to the state $state_i$ and is non-negative.

The optimal value function defined in Equation (B.1) is defined in terms of the optimal

value function for a previous stage *i-1* by the following recurrence relation:

$$u_i(state_i) = \max_{decision_i \in Decisions_i} [RSV_i(state_i, Decisions_i) + u_{i-1}(state_{i-1})] \tag{B.2}$$

The boundary condition is defined as follows:

$$u_0(state_0) = 0 \tag{B.3}$$

This describes the value of the optimal value function at decision stage 0 which is simply 0 because initially no reduction is made in the SV value achieved at decision stage 0. At stage 0 only one decision is available which is to allocate exactly one resource instance of an appropriate type to each tier of each application in each resource allocation interval. This is the least possible number of resources that can be allocated to each application at this decision stage. It should be noted that the optimal SV value can be obtained at any decision stage i by subtracting the value of the optimal value function at this decision stage from the SV value obtained at decision stage 0.

Appendix C

Detailed Analysis of RAP-DP

This appendix provides a more detailed analysis of the RAP-DP results shown previously in Figure 4.5b. Recall that the figure compares the solutions obtained by RAP-DP with the corresponding solutions obtained by exhaustively enumerating all possible solutions at each decision stage. At each decision stage i, RAP-DP selects the resource allocation plan with the highest RSV_i value as described previously in Appendix B. Optimality is achieved by using an optimal resource allocation plan selected at a decision stage i as a base to obtain the optimal resource allocation plan in the next decision stage *i+1*. Other non-optimal resource allocation plans obtained at decision stage i are not explored further in the decision stage *i+1*. This is because when starting from a *non-optimal* resource allocation at decision stage i to obtain an *optimal* resource allocation plan at decision stage *i+1*, the total RSV value, i.e., the sum of RSV_i and RSV_{i+1}, achieved at these two decision stages will not exceed the total RSV value achieved by the resource allocation plans at decision stages i and *i+1* when both plans are optimal. This will be illustrated in the following paragraph using the results of Figure 4.5b.

Table C.1 show a detailed view of the resource allocation plans obtained at the first three decision stages of Figure 4.5b. Recall that in this experiment four applications are considered for SLP over a planning horizon of four resource allocation intervals denoted by *t1*, *t2*, *t3* and *t4*. In each decision stage a resource allocation plan describes the number of web server instances allocated to each application in each resource allocation interval. Each resource allocation plan shown in Table C.1 has an identifier denoted as Px where x is a number to identify each resource allocation plan. Finally, an RSV value is shown for each resource allocation plan. In each decision stage only the three topmost optimal resource allocation

Table C.1: Detailed view of the results shown previously in Figure 4.5b

Plan #	Application 1				Application 2				Application 3				Application 4				*RSV*
	t1	*t2*	*t3*	*t4*	*t1*	*t2*	*t3*	*t4*	*t1*	*t2*	*t3*	*t4*	*t1*	*t2*	*t3*	*t4*	
Plan obtained initially in decision stage 0																	
P1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
Top 3 optimal plans obtained in decision stage 1 from plan *P1*																	
P2	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	1	42.78%
P3	1	1	1	1	1	1	1	1	1	2	1	1	1	1	1	1	39.06%
P4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	1	25.28%
Top 3 optimal plans obtained in decision stage 2 from plan *P2*																	
P5	1	1	1	1	1	1	1	1	1	2	1	1	1	2	1	1	39.06%
P6	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	24.96%
P7	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	1	20.10%
Top 3 optimal plans obtained in decision stage 2 from plan *P3*																	
P5	1	1	1	1	1	1	1	1	1	2	1	1	1	2	1	1	42.78%
P8	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	1	25.28%
P9	1	1	1	1	1	1	1	1	1	2	1	1	2	1	1	1	20.41%
Top 3 optimal plans obtained in decision stage 2 from plan *P4*																	
P6	1	1	1	1	1	1	1	1	1	1	1	1	1	2	2	1	42.46%
P8	1	1	1	1	1	1	1	1	1	2	1	1	1	1	2	1	39.06%
P10	1	1	1	1	1	1	1	1	1	1	1	1	2	1	2	1	20.35%

plans, i.e., with the highest *RSV* values, are shown. The plans are listed in descending order in terms of their RSV values.

Figure C.1 shows a tree representation of the resource allocation plans listed in Table C.1. This representation depicts the listed plans over the three decision stages. The gray shaded resource allocation plans are those selected by RAP-DP to be the optimal plans in the three decision stages. RAP-DP only evaluated plans *P1* to *P7* in the search. Other plans that are the children of non optimal plans are ignored. In spite of not evaluating all plans, RAP-DP is able to obtain the optimal plan with the highest total *RSV* value in each decision stage. Specifically, plan *P5* has the highest total *RSV* value of 81.84% which is the sum of the individual *RSV* values achieved by plans *P2* and *P5* as listed in Table C.1. As shown in the table, the non-optimal plans *P3* and *P4* obtained in decision stage 1 cannot generate plans in subsequent decision stages with higher total *RSV* values than the optimal plan *P5*. Specifically, plans *P8*, *P9* and *P10* which are not evaluated by RAP-DP achieve

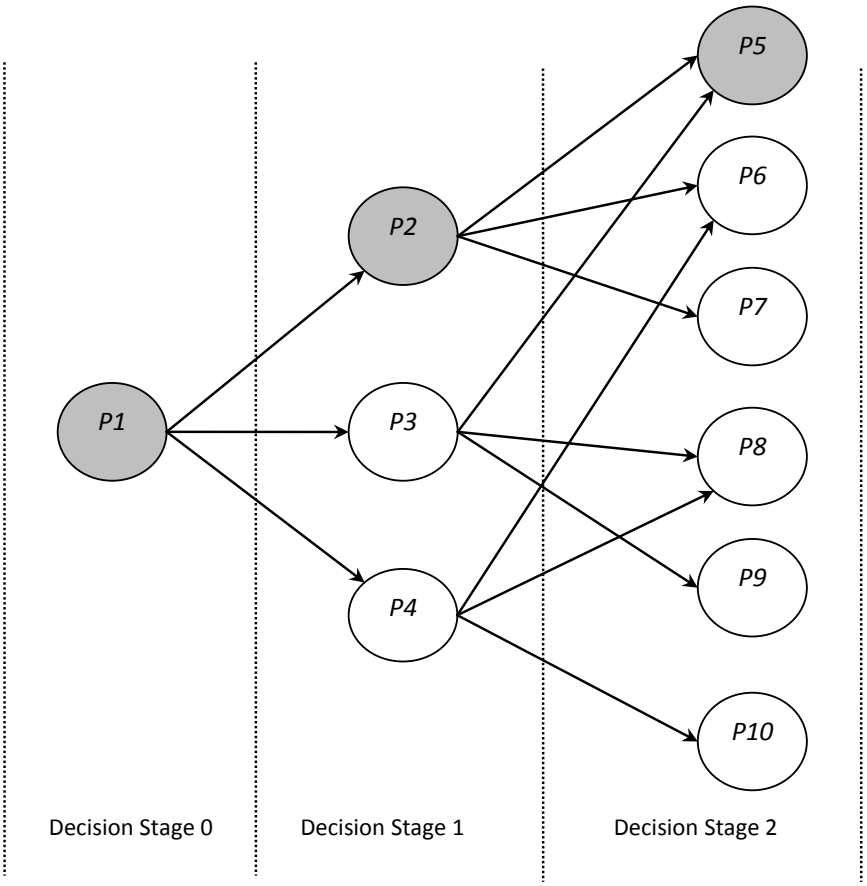


Figure C.1: Tree representation of the resource allocation plans shown in Table C.1

total RSV values of 64.34%, 59.47% and 45.63% which are all less than the total RSV value achieved by the optimal plan *P5*. This is because these three plans are generated based on non-optimal plans obtained in the previous decision stage, i.e, plans *P3* and *P4*.

Bibliography

[1] Amazon Cloud Watch. http://aws.amazon.com/cloudwatch/. Accessed: 2013-09-17.

[2] Amazon EC2 Reserved Instances. http://aws.amazon.com/ec2/purchasing-options/reserved-instances/. Accessed: 2014-01-20.

[3] Amazon Elastic Compute Cloud. http://aws.amazon.com/ec2/. Accessed: 2013-07-19.

[4] Amazon Elastic Compute Cloud instance types. http://aws.amazon.com/ec2/instance-types/. Accessed: 2013-08-15.

[5] Amazon Elastic Compute Cloud Quality of Service. http://aws.amazon.com/ec2-sla/. Accessed: 2013-07-30.

[6] Amazon Relational Database Service. http://aws.amazon.com/rds/. Accessed: 2013-09-16.

[7] Amazon Simple Storage Service. http://aws.amazon.com/s3/. Accessed: 2013-09-16.

[8] Amazon Web Services. http://aws.amazon.com/what-is-cloud-computing/. Accessed: 2013-09-16.

[9] Amazon Web Services Auto Scaling. http://aws.amazon.com/autoscaling/. Accessed: 2013-09-16.

[10] Eucalyptus Open Source AWS-Compatible Private Clouds. http://www.eucalyptus.com/. Accessed: 2013-08-13.

[11] GoGrid Cloud Service Provider. http://www.gogrid.com/. Accessed: 2013-08-13.

[12] Google App Engine. https://cloud.google.com/products/. Accessed: 2013-07-19.

[13] Google Docs. http://docs.google.com. Accessed: 2013-08-13.

[14] High Performance Computing on Amazon Web Services. http://aws.amazon.com/hpc-applications/. Accessed: 2013-11-15.

[15] Lanamark Suite. http://www.lanamark.com/product/overview. Accessed: 2013-04-19.

[16] Microsoft Azure. http://www.windowsazure.com/en-us/. Accessed: 2013-07-19.

[17] NetIQ PlateSpin Recon. https://www.netiq.com/products/recon/. Accessed: 2013-04-19.

[18] Nimbus Cloud Computing for Science. http://www.nimbusproject.org. Accessed: 2013-08-13.

[19] OpenNebula Open Source Data Center Virtualization. http://opennebula.org/. Accessed: 2013-08-13.

[20] RackSpace Cloud Service Provider. http://www.rackspace.com/. Accessed: 2013-08-13.

[21] RackSpace Quality of Service. http://www.rackspace.com/managed_hosting/support/servicelevels/managedsla/. Accessed: 2013-07-30.

[22] VMware Inc., vCenter CapacityIQ. http://www.vmware.com/products/vcenter-capacityiq/overview.html. Accessed: 2013-04-19.

[23] VMware Inc., VMware Capacity Planner. http://www.vmware.com/products/capacity-planner/. Accessed: 2013-04-19.

[24] J. Anselmi, E. Amaldi, and P. Cremonesi. Service Consolidation with End-to-End Response Time Constraints. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 345–352, 2008.

[25] D. Ardagna, B. Panicucci, M. Trubian, and L. Zhang. Energy-Aware Autonomic Resource Allocation in Multitier Virtualized Environments. *IEEE Transactions on Services Computing*, 5(1):2–19, 2012.

[26] D. Ardagna, M. Trubian, and L. Zhang. SLA based Resource Allocation Policies in Autonomic Environments. *Journal of Parallel & Distributed Computing*, 67(3):259–270, Mar. 2007.

[27] M. Ariltt, D. Krishnamurthy, and J. Rolia. Characterizing the Scalability of a Large Web-Based Shopping System. *ACM Trans. Internet Technol.*, 1(1):44–69, Aug. 2001.

[28] M. Arlitt and T. Jin. A Workload Characterization Study of the 1998 World Cup Web Site. *IEEE Network*, 14(3):30–37, 2000.

[29] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A View of Cloud Computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[30] L. Barroso and U. Holzle. The Case for Energy-Proportional Computing. *Computer*, 40(12):33–37, 2007.

[31] S. Baset. Cloud SLAs: Present and Future. *SIGOPS Oper. Syst. Rev.*, 46(2):57–66, July 2012.

[32] F. Baskett, K. Chandy, R. Muntz, and F. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *J. ACM*, 22(2):248–260, Apr. 1975.

[33] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[34] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. *Future Generation Computer Systems*, 28(5):755 – 768, 2012. Special Section: Energy efficiency in large-scale distributed systems.

[35] M. Bennani and D. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 229–240, 2005.

[36] P. Bodik, A. Fox, M. Franklin, M. Jordan, and D. Patterson. Characterizing, Modeling, and Generating Workload Spikes for Stateful Services. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pages 241–252. ACM, 2010.

[37] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616, June 2009.

[38] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, and R. Buyya. Cloudsim: a Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software Practice & Experience*, 41(1):23–50, Jan. 2011.

[39] A. Caniff, L. Lu, N. Mi, L. Cherkasova, and E. Smirni. Fastrack for Taming Burstiness and Saving Power in Multi-Tiered Systems. In *International Teletraffic Congress (ITC)*, pages 1–8, 2010.

[40] M. Cardosa, M. Korupolu, and A. Singh. Shares and Utilities based Power Consolidation in Virtualized Server Environments. In *International Symposium on Integrated Network Management (IM)*, pages 327–334, 2009.

[41] G. Casale, N. Mi, L. Cherkasova, and E. Smirni. Dealing with Burstiness in Multi-tier Applications: Models and Their Parameterization. *IEEE Transactions on Software Engineering*, 38(5):1040–1053, 2012.

[42] G. Casale, N. Mi, and E. Smirni. Bound Analysis of Closed Queueing Networks with

Workload Burstiness. In *Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 13–24, 2008.

[43] G. Casale, N. Mi, and E. Smirni. Model-Driven System Capacity Planning under Workload Burstiness. *IEEE Transactions on Computers*, 59(1):66–80, 2010.

[44] S. Chaisiri, B. Lee, and D. Niyato. Optimization of Resource Provisioning Cost in Cloud Computing. *IEEE Transactions on Services Computing*, 5(2):164–177, 2012.

[45] H. Chen, H. Kang, G. Jiang, K. Yoshihira, and A. Saxena. VCAE: A Virtualization and Consolidation Analysis Engine for Large Scale Data Centers. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 1–10, 2010.

[46] D. Cox and V. Isham. *Point Processes*. New York: Chapman and Hall, 1980.

[47] L. Dagum and R. Menon. OpenMP: An Industry Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998.

[48] U. Diwekar. *Introduction to Applied Optimization*. Springer-Verlag, 2008.

[49] S. Dreyfus. *The Art and Theory of Dynamic Programming*. Mathematics in Science and Engineering. Elsevier, Burlington, MA, 1977.

[50] B. Everitt, S. Landau, M. Leese, and D. Stahl. *Cluster Analysis*. John Wiley & Sons, 2011.

[51] J. Fito, I. Goiri, and J. Guitart. SLA-driven Elastic Cloud Hosting Provider. In *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pages 111–118, 2010.

[52] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside. Performance Analysis of Distributed Server Systems. In *In Proceedings of the International Conference on Software Quality*, pages 15–26, 1996.

[53] D. Gmach, J. Rolia, C. Bash, Y. Chen, T. Christian, A. Shah, R. Sharma, and Z. Wang. Capacity Planning and Power Management to Exploit Sustainable Energy. In *International Conference on Network and Service Management (CNSM)*, pages 96–103, 2010.

[54] D. Gmach, J. Rolia, and L. Cherkasova. Satisfying Service Level Objectices in a Self-Managing Resource Pool. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 243–253, 2009.

[55] D. Gmach, J. Rolia, L. Cherkasova, G. Belrose, T. Turicchi, and A. Kemper. An Integrated Approach to Resource Pool Management: Policies, Efficiency and Quality Metrics. In *IEEE International Conference on Dependable Systems and Networks (DSN)*, pages 326–335, 2008.

[56] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Capacity Management and Demand Prediction for Next Generation Data Centers. In *Proceedings of IEEE International Conference on Web Services (ICWS)*, pages 43–50, 2007.

[57] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload Analysis and Demand Prediction of Enterprise Data Center Applications. In *Proceedings of the International Symposium on Workload Characterization (IISWC)*, pages 171–180. IEEE Computer Society, 2007.

[58] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Resource Pool Management: Reactive versus Proactive or Let's be Friends. *Comput. Netw.*, 53(17):2905–2922, Dec. 2009.

[59] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, 1999.

[60] R. Gusella. Characterizing the Variability of Arrival Processes with Indexes of Dispersion. *IEEE Journal on Selected Areas in Communications*, 9(2):203–211, 1991.

[61] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey. Early Observations on the Performance of Windows Azure. In *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, pages 367–376, New York, NY, USA, 2010. ACM.

[62] C. Hyser, B. McKee, R. Gardner, and B. Watson. Autonomic Virtual Machine Placement in the Data Center. Technical Report HPL-2007-189, HP Labs, February 2007.

[63] G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. Generating Adaptation Policies for Multi-tier Applications in Consolidated Server Environments. In *International Conference on Autonomic Computing (ICAC)*, pages 23–32, 2008.

[64] D. Krishnamurthy, J. Rolia, and M. Xu. WAM -The Weighted Average Method for Predicting the Performance of Systems with Bursts of Customer Sessions. *IEEE Transcations on Software Engineering*, 37(5):718–735, Sept. 2011.

[65] E. Lazowska, J. Zahorjan, G. Graham, and K. Sevcik. *Quantitative System Performance: Computer System Analysis using Queueing Network Models*. Prentice-Hall, Inc., 1984.

[66] J. Li. *Fast Optimization for Scalable Application Deployments in Large Service Centers*. PhD thesis, Carleton University, Ottawa, Canada, 2011.

[67] J. Li, J. Chinneck, M. Woodside, and M. Litoiu. Fast Scalable Optimization to Configure Service Systems having Cost and Quality of Service Constraints. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 159–168, 2009.

[68] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. Performance Model Driven Qos Guarantees and Optimization in Clouds. In *Proceedings of ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD)*, pages 15–22, 2009.

[69] J. Li, M. Woodside, J. Chinneck, and M. Litoiu. CloudOpt: Multi-goal Optimization of Application Deployments across a Cloud. In *International Conference on Network and Service Management (CNSM)*, pages 1–9, 2011.

[70] J. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

[71] S. Malkowski, M. Hedwig, D. Jayasinghe, C. Pu, and D. Neumann. Cloudxplor: a Tool for Configuration Planning in Clouds Based on Empirical Data. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 391–398, 2010.

[72] S. Malkowski, M. Hedwig, J. Li, C. Pu, and D. Neumann. Automated Control for Elastic n-tier Workloads based on Empirical Modeling. In *Proceedings of the international conference on Autonomic computing (ICAC)*, pages 131–140, 2011.

[73] C. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[74] M. Mao and M. Humphrey. Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2011.

[75] M. Mao, J. Li, and M. Humphrey. Cloud Auto-scaling with Deadline and Budget Constraints. In *Proceedings of the International Conference on Grid Computing (GRID)*, pages 41–48, 2010.

[76] D. Menascé, V. Almeida, R. Fonseca, and M. Mendes. A Methodology for Workload Characterization of E-commerce Sites. In *Proceedings of the ACM conference on Electronic commerce (EC)*, pages 119–128, 1999.

[77] D. Menascé, V. Almeida, R. Riedi, F. Ribeiro, R. Fonseca, and W. Meira, Jr. In Search of Invariants for E-business Workloads. In *Proceedings of the ACM conference on Electronic commerce, (EC)*, pages 56–65, 2000.

[78] D. Menascé, D. Barbará, and R. Dodge. Preserving QoS of E-commerce Sites Through Self-Tuning: A Performance Model Approach. In *Proceedings of the ACM Conference on Electronic Commerce*, EC '01, pages 224–234, New York, NY, USA, 2001. ACM.

[79] D. Menascé, L. Dowdy, and V. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, 2004.

[80] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis. Efficient Resource Provisioning in Compute Clouds via VM Multiplexing. In *Proceedings of International Conference on Autonomic Computing (ICAC)*, pages 11–20, 2010.

[81] N. Mi, Q. Zhang, A. Riska, E. Smirni, and E. Riedel. Performance Impacts of Autocorrelated Flows in Multi-tiered Systems. *Perform. Eval.*, 64(9-12):1082–1101, Oct. 2007.

[82] K. Muralidhar. Monte Carlo Simulation. In *Encyclopedia of Information Systems*, pages 193–201. Elsevier, 2003.

[83] S. Mylavarapu, V. Sukthankar, and P. Banerjee. An Optimized Capacity Planning Approach for Virtual Infrastructure Exhibiting Stochastic Workload. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 386–390, 2010.

[84] J. Pea, J. Lozano, and P. Larraaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters*, 20(10):1027 – 1040, 1999.

[85] B. Rimal, E. Choi, and I. Lumb. A Taxonomy and Survey of Cloud Computing Systems. In *Proceedings of the International Joint Conference on INC, IMS and IDC (NCM)*,

pages 44–51, 2009.

[86] J. Rolia. Predicting the Performance of Software Systems. Technical Report CSRI-260, Computer Systems Research Institute, University of Toronto, Toronto, Canada, January 1992.

[87] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak. A Capacity Management Service for Resource Pools. In *Proceedings of the international Workshop on Software and Performance (WOSP)*, pages 229–237, 2005.

[88] J. Rolia and K. Sevcik. The Method of Layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, 1995.

[89] N. Roy, A. Dubey, and A. Gokhale. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*, pages 500–507, 2011.

[90] P. Sempolinski and D. Thain. A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 417–426, 2010.

[91] M. Tanelli, D. Ardagna, M. Lovera, and L. Zhang. Model Identification for Energy-Aware Management of Web Service Systems. In *Proceedings of the International Conference on Service-Oriented Computing (ICSOC)*, pages 599–606, 2008.

[92] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An Analytical Model for Multi-tier Internet Services and its Applications. *ACM SIGMETRICS Perform. Eval. Rev.*, 33(1):291–302, June 2005.

[93] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic Provisioning of Multi-tier Internet Applications. In *International Conference on Autonomic Computing (ICAC)*, pages 217–228, 2005.

[94] U. Vallamsetty, K. Kant, and P. Mohapatra. Characterization of E-commerce Traffic. In *Proceedings of IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, (WECWIS)*, pages 137–144, 2002.

[95] L. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, Dec. 2008.

[96] W. Vogels. Beyond Server Consolidation. *Queue*, 6(1):20–26, Jan. 2008.

[97] Y. Xing and Y. Zhan. Virtualization and Cloud Computing. In *Future Wireless Networks and Information Systems, Lecture Notes in Electrical Engineering*, volume 143, pages 305–312. Springer Berlin Heidelberg, 2012.

[98] A. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers. Efficient Resource Management for Cloud Computing Environments. In *International Conference on Green Computing*, pages 357–364, 2010.

[99] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. 1000 Islands: Integrated Capacity and Workload Management for the Next Generation Data Center. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 172–181, 2008.