Service Level Agreement(SLA) based SaaS Cloud Management System

Linlin Wu[1], Saurabh Kumar Garg[2], and Rajkumar Buyya[1]
[1]Cloud Computing and Distributed Systems (CLOUDS) Lab
Department of Computing and Information Systems
The University of Melbourne, Australia
Email: rbuyya@unimelb.edu.au
[2]School of Engineering and ICT, University of Tasmania
Hobart, Tasmania, Australia
Email: skgarg@utas.edu.au

***Abstract*—Cloud computing has emerged as a new computing paradigm which has revolutionized the IT industry. It has particularly transformed the licensing of software products which are now being offered as a Service on pay-as-you-go basis. This has tremendously increased the complexity for software providers as they now have to not only manage their resources on which software are hosted but also they need to provide expected Quality of Service for customers. The Quality of Service (QoS) required by customers is guaranteed using a legal document SLA (Service Level Agreement). Current, resource management systems do not cater to the needs of a Software as a Service (SaaS) provider who requires to provide flexible and low cost services while not affecting their profit and market share. Most of them focus either at infrastructure level or at platform level. This work fills this gap by proposing a novel SLA based resource management system designed after analysing requirements of SaaS in Clouds. The proposed system is implemented using latest technologies and can scale in and out depending on updates in the user demand. We present the architectural design and evaluate the implementation with a real case study in a real Cloud environment.**

I. Introduction

Cloud computing is a solution for addressing challenges such as licensing, distribution, configuration, and operation of enterprise applications associated with the traditional IT infrastructure, software sales and deployment models[1]. Migrating from a traditional model to the Cloud model reduces the maintenance complexity and cost for enterprise customers, and provides on-going revenue for Software as a Service (SaaS) providers. One can find that more and more complex applications are being delivered in Software as a Service (SaaS) model on Cloud[2]. With this development, these applications hosted under SaaS model have complex provisioning, configuration, and deployment requirements.

In real business environment, resource requirements keeps changing based on various factors such as customers demand and new services. SaaS providers also need to allocate infrastructure resources on the fly according to consumers dynamic Quality of Service (QoS) requirements written in the form of Service Level Agreement (SLA)[3]. Even though efforts have been made by several projects, still most do not offer capabilities such as support for adapting dynamic customer demands using Cloud resources. Given Cloud resources are elastic in nature, it seems trivial to adapt number of resources based on user requirements, however it becomes challenging when SaaS providers have to maximise the customer satisfaction level simultaneously with their profit. Therefore, there are efforts needed that focus on design, development, and implementation of software systems based on novel SLA-based resource allocation models exclusively designed for SaaS providers.

In the case of SaaS provider, to enable crucial business operations of customer companies, there are several critical QoS parameters that are needed to be consider in a service request provisioning, such as response time. Service provisioning becomes challenging when QoS requirements cannot be static and need to be dynamically updated over time due to continuing changes in business operations and operational environments of the customer[4]. To have competitive advantage, SaaS providers need to pay greater importance on customers since they pay for accessing services in data canters. In other words, any resource management system that focuses on SaaS Clouds should also be customer driven[5]. Therefore, to fill this gap, this paper presents the design of a novel SLA-based resource allocation system adapting dynamic customer demands using cloud infrastructure resources. A prototype of the customer requirements driven SLA-based resource management system is implemented taking care of the changes in customer requirements and resource side heterogeneity using SharePoint platform and .Net technologies.

This paper presents first various SLA based customer requirements of SLA driven resource management system. Then, the proposed architecture of a SaaS management system with the details of design in subsequent section. In the following sections, the realization of SLA-based resource management system (SLARMS) is presented with evaluation of a prototype system in an operational data centre.

II. Requirements of SaaS

For SaaS Cloud providers to host software as a commercial offering and to enable seamless execution of crucial business operations of companies, there are many critical QoS parameters that SaaS management system needs to take into account. In particular, as Cloud computing environment is considered to be dynamic and elastic, customers QoS requirements cannot be static and need to be dynamically updated over time with business requirements. For example, just imagine if ebay

service would have been hosted on Clouds, then with the number of their end users, resource needs of ebay will also change. Another good examples are consulting or service based companies such as Accenture, which may need SaaS resources depend on their current client. The approach for realization of customer based or SLA based SaaS management system, the following requirements should be included:

1) Support for customer-driven service management based on customer QoS requirements;
2) Incorporation of autonomic resource management models that effectively self-manage changes in service requirements to satisfy both new service demands and existing service obligations;
3) Leverage of Virtual Machine (VM) technology to dynamically deploy, configure and assign resources according to service requirements; and
4) Implementation of developed resource management strategies into a real computing server in an operational data centre.

III. Architecture of A SaaS Cloud

In order to fulfil the aforementioned requirements, a SaaS model for serving customers in Cloud is shown in Figure 1. A customer sends a request for software services offered by a SaaS provider, who uses three layers, namely application layer, platform layer and infrastructure layer, to satisfy the customers request. The application layer manages all application services that are offered to customers by the SaaS provider. In the platform layer, the request monitor is used to monitor requests including new and upgrade requests. Whenever a customer changes QoS requirements, the mapper and decision maker are invoked. The mapper is responsible for translating the customers' QoS requirements to infrastructure level parameters and the decision maker is used to make decision that whether a the request can be accepted or not and where to schedule the acceptable request. In addition, the resource allocator is responsible for initiating or allocating Virtual Machines (VMs) to serve the request. Moreover, the SLA manager is used to track SLA violations according to actual resource information. Based on SLA terms, the market manager updates the final cost and profit accordingly. The infrastructure layer includes data centres where VMs are hosted.

IV. Design of SLA based SaaS Management System

In this section, we provide finer details related to fundamental classes of the SLA-based resource allocation system, which are also the building blocks of the system. The overall Class, Sequence, and States design diagrams are shown below.

A. Class Diagram

The main components of class diagram (shown in Figure 2) is described below:

- (*QoS) Request Monitor:* When a customer submits a new request or changes an existing request for the service, this class monitors changes and then invokes Mapper and DecisionMaker classes to reschedule the request.
- *Mapper:* This class maps customer QoS requirements to a suitable type of resource by method getVMTypebyServiceType(servType).
- *SLA Service Setting:* This class provides functions to access and operate the SaaS providers predefined service characteristics. For example, getServiceResponseTime(servType) is used to retrieve the predefined service response time.
- *Decision Maker:* This class invokes the admission control and scheduling classes to make decision on whether to admit the customer request and how to assign resource to the customer.
- *Admission Control:* This class is used to interpret and analyse customers QoS requirements and receive the pre-scheduling result from scheduler, and then it uses admission control criteria to decide whether to accept or reject the request.
- *Scheduler:* This class is responsible for pre-scheduling the request with scheduling strategies and returning where the request can be scheduled.
- *SLA Manager:* SLA Manager is the class that keeps track of SLAs fulfilment between customers and service providers. It also detects the penalty delay and updates the market manager.
- *Market Manager:* It is responsible for calculating and updating the cost and profit according to the actual resource usage. When there is a SLA violation, penalty cost is calculated and final profit is adjusted by the market manager.
- *Data Centre:* Characteristics and related functions of data centres are represented in this class.
- *VM:* This class represents actual VMs and includes their related data, such as VM initiation time.
- *VM Setting:* This class includes characteristics of VMs, which are average values based on history records.
- *Resource Coordinator:* This class assigns existing resource or initiating new resources for customer requests according to the decision. It includes VM Initiator, VM Assigner, VM Monitor, and VM Cleaner.
  - *VM Initiator:* It takes the responsibility of creating, deploying and configuring VMs using VM templates in an appropriate data centre.
  - *VM Assigner:* It is responsible for configuring software on the appropriate VM.

B. Sequence Diagram

1) Internal interaction among system entities: Figure 3 shows how different entities interact at system level. When the system receives a request from a customer, the QoS request monitor invokes the class mappers function called getVMTypebyServType(servType), which returns a suitable VM type. Following this, the QoS request monitor invokes the function MakeDecision() in class DecisionMaker to get

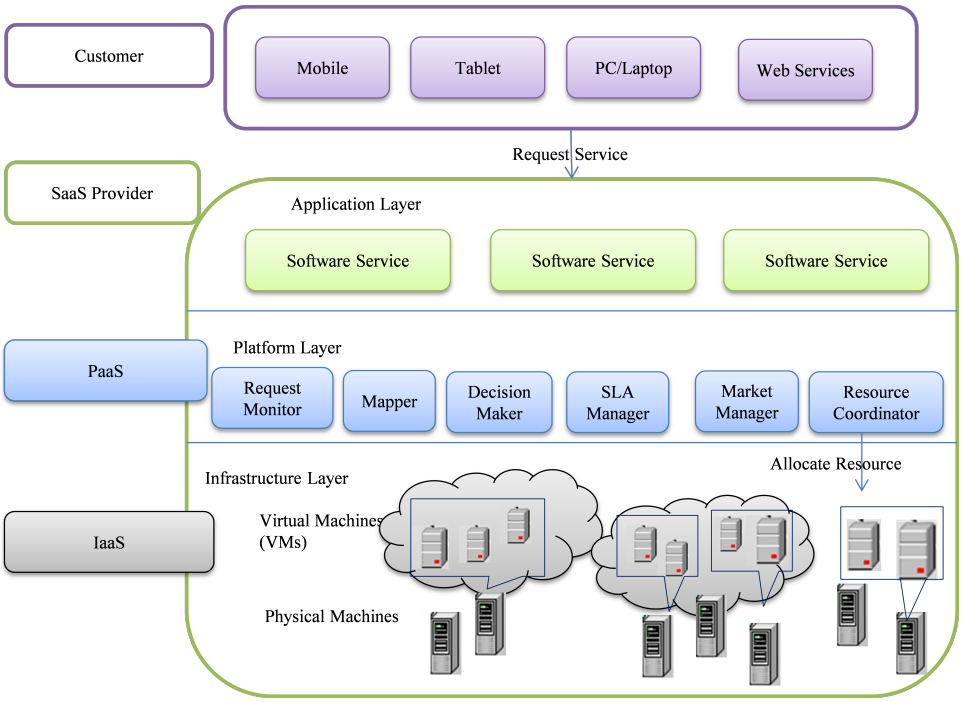


Fig. 1: SLA-based resource management system high level architecture

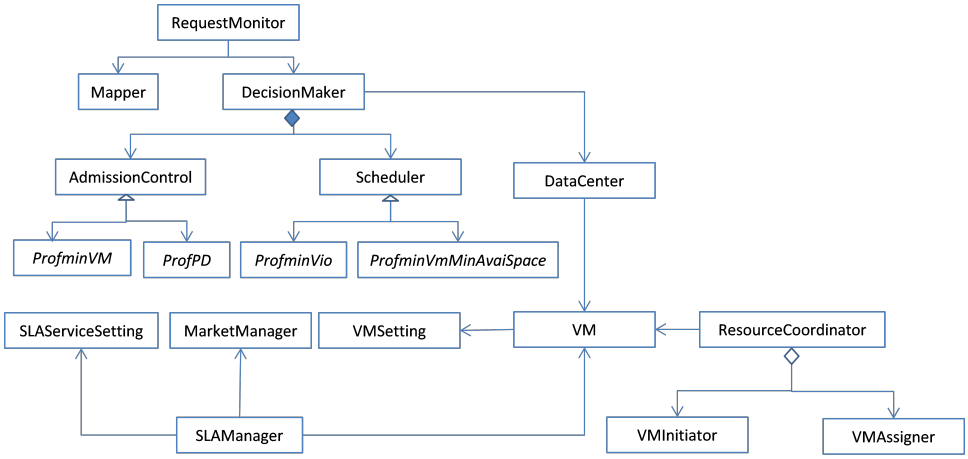


Fig. 2: Class Diagram

decision whether this request can be accepted. Next, the DecisionMaker class invokes the function AdmissionControlProcess() in class AdmissionControl, which includes two stages: the first stage AdmissionControlAnalysis() calls the schedulers SchedulingAnalysis() function, which checks current resource availability and capability using scheduling strategies and returns where the request can be scheduled. The second stage,AdmissionControlDecisionMaking(), checks if the request can be accepted regarding to the admission control criteria and returns the result to Decision Maker. Finally, the request monitor receives the decision.

2) Internal interaction at resource level: Figure 4 shows how resource level entities interact with each other. The resource coordinator detects the decision made by the decision maker. If the decision result is accept and scheduling result is initiateNewVM, then the request state goes into provisioning and resource coordinator calls the initiateVM() function in VMInitiator class to create and deploy a suitable VM image. If the scheduling result is Wait or Insert, then the resource coordinator calls the assignRequest() function in class VMAssigner to assign the request to an appropriate existing VM by configuring the software service. The status of the request becomes inserting or waiting. Following that, the monitorVMIni() function in class VMMonitor detects the actual VM initiation time and then updates the VMinitiation time by calling theupdateVM() function in the class VM. When all requests are finished on a VM, the VMCleaner invokes function PowerOff() to power off the VM.

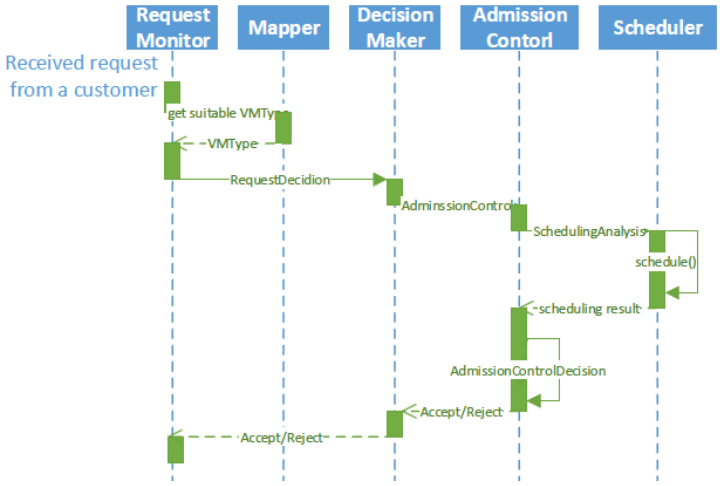


Fig. 3: Interaction between System Entities

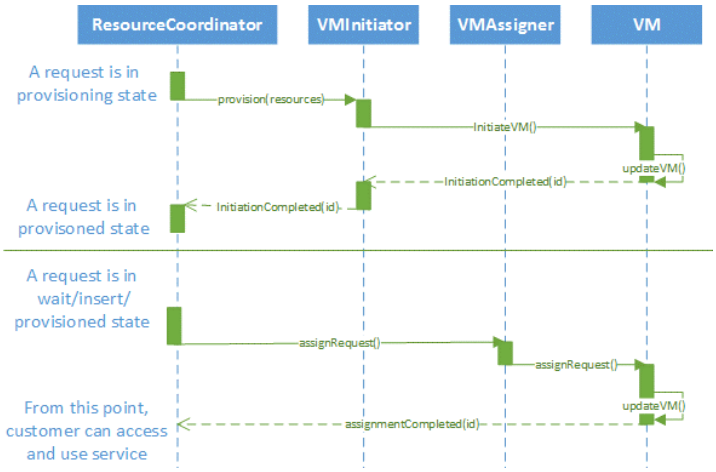


Fig. 4: Interaction between Resource level Entities

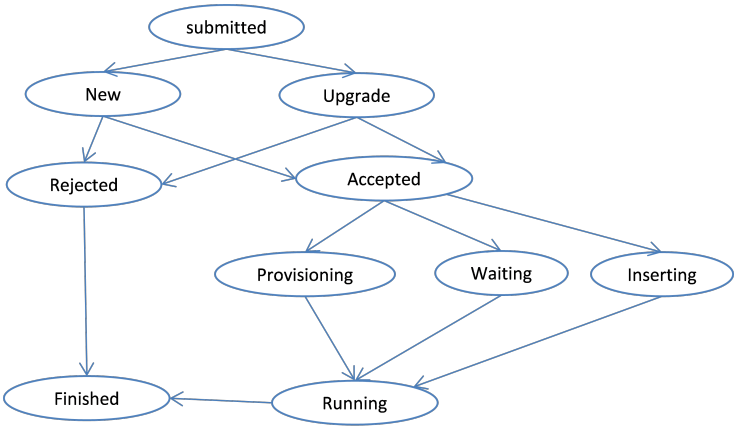


Fig. 5: State Diagram

C. State Diagram

Figure 5 illustrates diverse states that a customer QoS request can experience during its lifetime. When a request is submitted to the system, the new request goes to the new state and the upgrade request goes to the upgrade state. Both new and upgrade requests can go to the rejected state if a SaaS provider cannot gain the expected profit. If service start deadline is achievable with available resources, the request goes to the inserting state. If there is no resource available immediately but some existing requests will finish before the service start deadline, then the request goes to the waiting state. When the Scheduler detects that a new resource needs to be initiated for the request, either because there is no existing resource available before the service start deadline, but new resource can be initiated for the request, then the request goes to the provisioning state. For inserting, waiting and provisioning requests, after the request has been assigned to the VM, the states goes to the running state, which means a customer starts to use the service for enterprise software as a service or a task starts to execute for bag of task service. Also, changes in state may happen every time a request contract expires and then the resource capability is recalculated. For both new and upgrade requests, the finished state is reached in three different situations: (i) contract expires; (ii) system failure; and (iii) the customer cancelled the request

V. System Implementation Technologies

The SLARMS has been implemented by leveraging the following key technologies using $C\#$ on .Net platform: (1) SharePoint 2010, which is a secure, manageable, and web-based platform supporting application development. (2) PowerShell for creating, managing, and configuring VMs hosted on private and public cloud (such as Azure[6]).

A. Implementation Considerations

The design consideration of the SLARMS are the following:

- **Support for dynamic customer requests:** When there is a customer updating the request, the request monitor will be triggered to detect request changes and go through the decision making process.
- **Support for scalable infrastructure resources:** To allow easy utilization of using different types of Cloud infrastructures,SLARMS is designed to use $C\#$ in .Net platform to execute PowerShell command on remote VMs. PowerShell has been chosen because the most popular private VM infrastructure provision technology, VMWare, has a PowerShell based API (PowerCML). In addition, two of the most popular public infrastructure providers - Azure and Amazon, support PowerShell VM provision and configuration.
- **Fault tolerance:** SLARMS can handle failures at two stages: during decision making, and during resource provisioning. Failures during resource provisioning (initiation or allocation) can occur due to various reasons, such as network problems. In this case, the failed resource will be re-provisioned in the next resource allocation cycle.
- **Scalability:** Most of the SLARMS's components work independently and interact through a database, which facilitates the scalable implementation of SLARMS

as each component can be distributed across different servers accessing a shared database.

B. Implementation Details

The Implementation design as shown in Figure 6 follows the three layer design pattern containing data layer, business logic layer and presentation layer. The main system entities are implemented using the following technologies:

- Custom web parts and web pages: In the presentation layer, custom web parts and web pages are used to provide an easy to use portal for customers to add or update their requests.
- Workflow: Workflow technology in SharePoint is used to implement QoS Request Monitor. The workflow can be triggered when there is a new request or any field of an existed request is updated. The background technology to support SharePoint workflow is the .Net workflow foundation.
- Event Handler/Event Receiver: SharePoint Event Handler/ Event Receiver technology is used to implement VMMonitor. Whenever there is any change happens on VM, such as actual VM initiation time is updated in the list, then the event handler will detect the change and invokes the SLA manager to calculate the penalty delay.
- Class: Standard $C\#$ classes are used to implement other components, such as main components decision maker, which includes admission control and scheduler.
- Timer job: SharePoint timer job is used to implement VMCleaner. The timer job runs every minute to detect if any VM does not have requests allocated and then the VM will be powered off in one hour.
- PowerShell: is used for most of resource coordinator related operations, such as VM initiation, because PowerCLI (based on PowerShell) is the easiest API to operate VMware Vsphere virtualized Cloud infrastructures (and for the extension of future work it is the common way to access Azure and Amazon EC2). In addition, for guestOS operation, the PowerShell is one of the most powerful technologies to configure the guestOS and install the software.
- Linq and CAML(Collaborative Application Markup Language): To implement the Data Access Layer, both Linq and CAML data access technologies are used because of some issues with Linq. For example, when disposing the data context, there is an error which is a known issue. Therefore, traditional CAML is used for insert operation and keep Linq for the rest data access operations.
- All data tables are presented using SharePoint Column and List technologies, which are more readable and give user friendly ways to structure the information, and all table structures and data are stored in SQL Server 2008.
- Internet Information Services (IIS): IIS for Windows Server is a flexible, secure, and manageable Web server for hosting anything on the Web. From media streaming to web applications, IIS's scalable and open architecture is ready to handle the most demanding tasks.

VI. Case Study: CA (Computer Associates) Directory

This section describes how the SLARMS prototype is implemented using a private enterprise Cloud. This private Cloud is within an enterprise to increase the productivity of their users, hence in other words, it increases the amount of computing resources available within an enterprise to accelerate application performance. The decision making process includes two main components: scheduling and admission control, in this case study, we implemented the algorithms that were introduced in our previous work[4][7].

A. System Setup Details

1) Customer Related Details: A Customer requests CA Directory services. This component is constituted by a simple Web Service client that generates all resource requests to SLARMS with the following QoS:

- Request Type (reqType): It defines the customer request type, which is new or upgrade service. A new request will get one hour free service usage, while an upgrade service is for an existing customer, who wants to upgrade from a lower service edition to an upper service edition (According to the customer usage, there may be a customer loyalty level).
- Product Type (proType): The software products offered to customers. It can be Standard, Silver, and Gold service. The Standard product includes CA Director. The Silver service package contains all functions of Standard plus JExplorer component. The Gold service includes all features of Silver plus dxgrid component.
- Account Number (accNum): It constrains the maximum number of concurrent users from the same organization can use the software service.
- Contract Length (conLen): How long the software service is legally available for a customer to use (minimum is one hour).
- Records storage (recNum): The maximum storage capability for each DSA period and it will impact the data transfer time during the service upgrade (The value of this parameter is predefined in SLA).
- Response Time (respTime): It represents the elapsed time between the end of a demand on a software service and the beginning of a service. Violation occurs when actual elapsed time is longer than the pre-defined response time in the SLA.

2) SaaS Provider Related Details: Application Layer: provides CA Directory services. The CA directory provides a high-performance directory foundation for online applications. It allows customer organizations to meet the needs of new and future dynamic business applications and improve operational

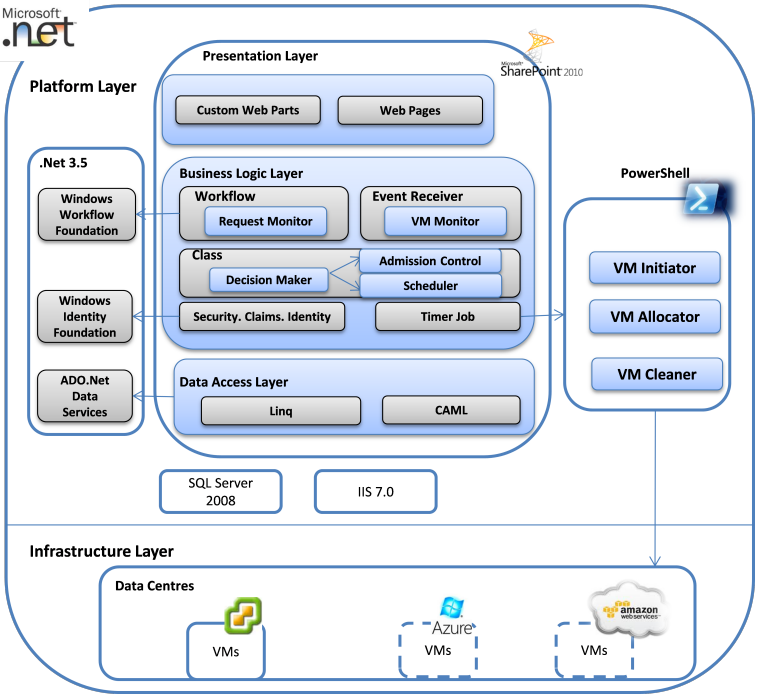


Fig. 6: Implementation Technologies

VM Type	Service	Account #	Storage	VM Price	CPU	Memory	Storage
Small	Standard	[0, m]	[0, n]	$0.085	1 core	1G	50G
Medium	Silver	[m+1,2m]	[n+1,2n]	$0.34	2 cores	2G	50G
Large	Gold	[2m+1,5m]	[2n+1,5n]	$0.68	4 cores	4G	50G

Fig. 7: Mapper Details

efficiency by consolidating islands of data into a single information backbone.

Platform Layer: SharePoint 2010 platform and PowerShell are two main technologies used in this layer. SharePoint is used to implement most platform layer components except the resource allocator, which is implemented in PowerShell. The SharePoint platform and PowerShell scripts are integrated with CSharp language on .Net platform. Details are described in the next section.

Infrastructure Layer: In CA Lab, the internal operable Cloud infrastructure is built using VMware VSphere, which is the industry leading virtualization platform. This layer can be extended into public clouds.

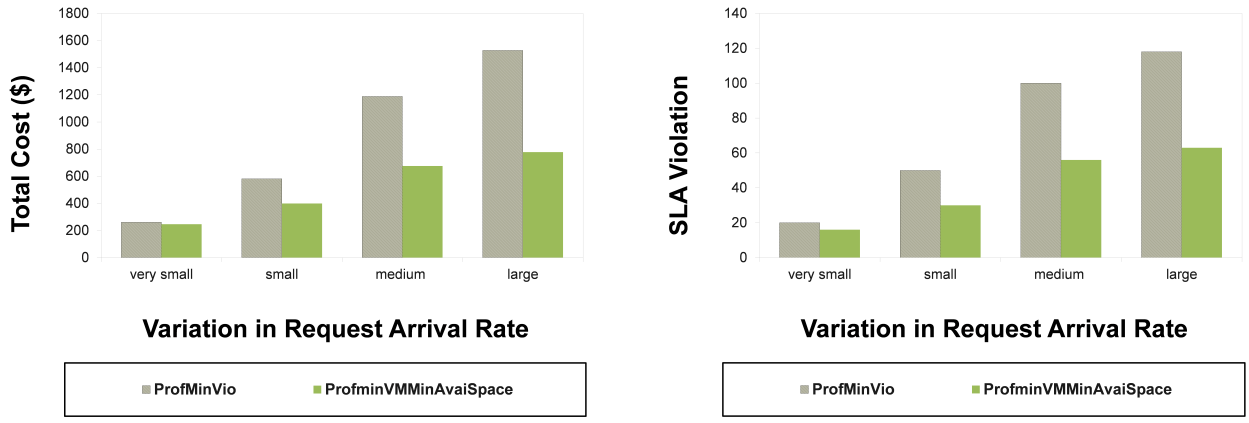
The platform layer of a SaaS provider uses VM images to create instances according to the mapping (Table in Figure 7) and decision. (Table in Figure 7, m is 5, n is 10). Therefore, it is important to identify the following properties for resource allocation mechanisms to ensure that the SLA is adequately drafted:

- VM types (l): How many types of VM can be used and what they are. For example, there are three types of VM, which are large, medium, and small. The capacity of one large VM equals to that of two medium VMs or four small VMs.
- VM Service Initiation Time (iniTimeSev): How long it takes to initiate a VM, which is deployed with the service appliance.
- VM Price (PriVM): How much it costs to a SaaS provider for using a VM to serve the customer request per time unit. It includes the physical equipment, power, network, and administration price.

VII. Performance Evaluation

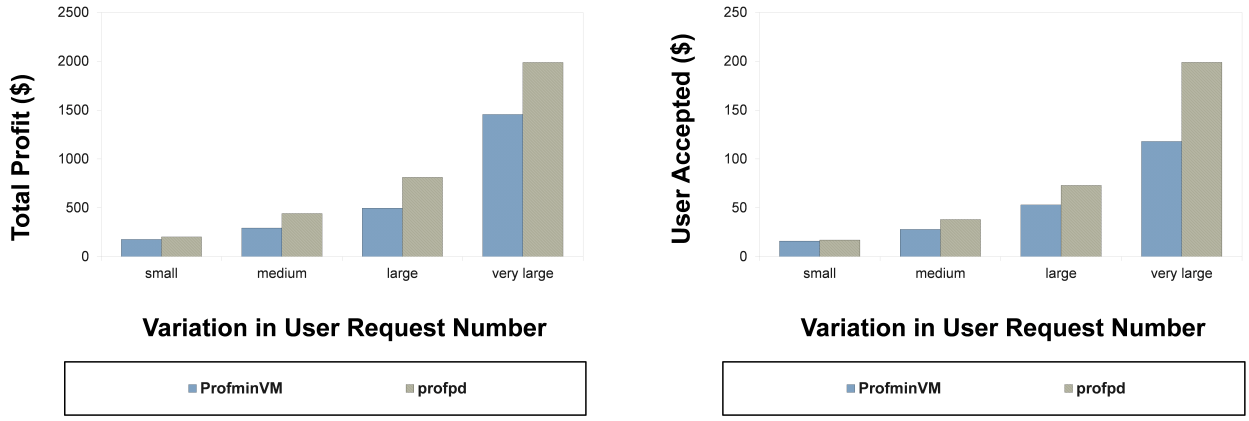
A. Experiment Setup

The evaluation of mechanisms within SLARMS has been carried out entirely in CA Lab VMware Vsphere Cloud infrastructure environment. The experimental setup consists of three types of dynamic resources: small instance (1 GB of memory, 1 CPU core, 50G of local instance storage, Windows OS); medium instance(2 GB of memory, 2 CPU core, 50G of local instance storage, Windows OS); and large instance (4 GB of memory, 4 CPU core, 50G of local instance storage, Windows OS). An enterprise application CA directory is used for experiments. SLA is defined in terms of response



(a) (b)

Fig. 8: Variation in Request Arrival Rate



(a) (b)

Fig. 9: Variation in User Request Number

times. The experiment evaluation is designed based on the CA CloudMinder test strategy and plan. CloudMinder is an online application that uses CA Directory as the directory foundation. In this set of experiments the total profit, number of accepted users and number of SLA violations are evaluated as follows during the variation of request arrival rate from 20 to 200 requests per second. Up to 200 concurrent user requests are considered because 1) The test strategy provided by CA is designed using 200 user requests, which has been analysed through their customer usage data and 2) The capability of the private data centre allocated to this research work is limited, which does not allow a very large number of user requests.

B. Summary of Results

1) Evaluation of SLARMS's Scheduling Algorithms: The evaluation is designed to test the systems decision making using different algorithms implemented in the system, specifically our previously proposed algorithms [4]. As expected, the algorithms perform the similar trend as the simulation results in the prototype implementation environment. In this set of experiment the total cost, SLA violations are evaluated in this section during the variation of request upgrade proportion varies from very low to very high. It can be seen from Figures 8a and 8b, in average the algorithm ProfminVMminAvaiSpace reduces about 50% cost compared with ProfminVio. As Figure 8b shows, during the arrival rate variation, the number of SLA violations caused by ProfminVMminAvaiSpace is less than the ProfminVio because the ProfminVio has more risk to cause VM initiation delay due to network-related issues. Therefore, during the variation of arrival rate, the ProfminVMminAvaiSpace performs better and minimize the SLA violations in the context of resource sharing, where it is impossible to avoid SLA violations.

2) Evaluation of SLARMS's Admission Control Algorithms: The evaluation is designed to test the systems decision making using different algorithms implemented in the system, specifically our previously proposed algorithms [7]. The evaluation results show that the algorithms performs similar trend in the

prototype environment. In this set of experiment total profit and number of accepted users are evaluated during the variation of user request number from 10(small) to 100(very large). Figures 9a and 9b shows that the ProfPD achieves (17%) more profit over ProfminVM by accepting (15%) more user requests, when number of users changes from small to very large. When the number of users is increased from medium to large, the profit difference between ProfPD and ProfminVM became larger. This is because when the number of requests increases, the number of users being accepted increases by utilizing initiated VMs. Therefore, a SaaS provider should use ProfPD to maximize profit.

VIII. Related Work

There are several previous approaches for resource management with respect to SLA. Control-theory approach has been proposed to dynamically adjust resource allocation to maintain the service differentiation [8]. CPU cycles of single servers are main concerns of other approaches, which share resources among multiple customer requests or applications [9][10]. For example, the *Shift* adjusts how much and when CPU resources should be allocated to a VM [11]. In contrast, SLARMS focuses on sharing at the granularity of whole VMs and the management of a whole farm of servers. Most Cloud resource management systems [12][13] only focus on infrastructure based Cloud computing services. Systems like IcorpMaker provides isolation via virtual private networks rather than VM [14]. Oceano attempts to modify the computing environment (e.g. by installing an operating system) to satisfy the allocation[11]. Finally, the Galaxy project [15] focuses on providing tools to build Windows-NT clusters. It does not consider SLA monitoring. SLARMS provides a unique and more comprehensive combination of technologies to address a number of issues ignored by these approaches and focused on SLA-based customer requirement driven resource provisioning.

IX. Conclusions and Future Work

To meet requirements of SLA-based resource management of Cloud services, this paper is focused on the design, development, and implementation of a software system, called SLA-based Resource Management System (SLARMS), based on novel SLA-based resource management algorithms exclusively designed for SaaS. The architecture and implementation of SLARMS is comprehensively described and evaluated. Through the prototype implementation, we also demonstrated the usefulness of the design of proposed system using a real case study. The resource used in this prototype is a private Cloud, hosted by Computer Associates, who is a Cloud software solution provider. The case study used CA Directory as a service because of the availability of the software. However, SaaS providers can offer any software as a service using our algorithms and proposed management systems accordingly. This prototype can be plugged in with different resource management strategies to achieve different objectives. SaaS providers can scale out to use multiple resource providers including 3rd party resource providers with different resource APIs. Two sets of experiments are performed to test previously proposed algorithms implemented as part of the system. In the experiments, the total cost, SLA violations were evaluated. As expected, the algorithms perform the similar trend as the simulation results in the prototype implementation environment.

In future, this work will be extended and evaluated for several other SaaS applications. To make the system more fault tolerant and scalable, a distributed database will be implemented.

References

[1] 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*, vol. 25, no. 6, pp. 599–616, 2009.

[2] S. Dillon and G. Vossen, "SaaS cloud computing in small and medium enterprises: A comparison between Germany and New Zealand," *IJITCC*, vol. 3, no. 2, pp. 87–107, 2015.

[3] H. N. Van, F. D. Tran, and J.-M. Menaud, "SLA-aware virtual resource management for cloud infrastructures," in *Proceedings of Ninth IEEE International Conference on Computer and Information Technology, Xiamen, China*, 2009.

[4] L. Wu, S. K. Garg, S. Versteeg, and R. Buyya, "SLA-based Resource Provisioning for Software as a Service Applications in Cloud Computing Environments," *IEEE Transactions on Services Computing*, vol. 7, no. 3, pp. 465–485, 2014.

[5] L. Wu and R. Buyya, "Service Level Agreement (SLA) in Utility Computing Systems, Performance and Dependability in Service Computing:Concepts, Techniques and Research Directions." IGI Global, 2011.

[6] B. Calder, "Inside windows azure: the challenges and opportunities of a cloud operating system," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1. ACM, 2014, pp. 1–2.

[7] L. Wu, S. K. Garg, and R. Buyya, "SLA-based admission control for a Software-as-a-Service provider in Cloud computing environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1280–1299, 2012.

[8] O. Sukwong, A. Sangpetch, and H. S. Kim, "Sageshift: Managing SLAs for highly consolidated cloud," in *Proceedings of IEEE INFOCOM, Orlanda, Florida, USA*, 2012.

[9] E. M. Maximilien and M. P. Singh, "A framework and ontology for dynamic web services selection," *Internet Computing, IEEE*, vol. 8, no. 5, pp. 84–93, 2004.

[10] W. Vogels and D. Dumitriu, "An overview of the galaxy management framework for scalable enterprise cluster computing," in *Proceedings of IEEE International Conference on Cluster Computing, Saxony, Germany*, 2000.

[11] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. P. Pazel, J. Pershing, and B. Rochwerger, "Oceano-SLA based management of a computing utility," in *Proceedings of 2001 IEEE/IFIP International Symposium on Integrated Network Management*, 2001.

[12] R. Rajavel and T. Mala, "Slaocms: A layered architecture of sla oriented cloud management system for achieving agreement during resource failure," in *Proceedings of the Second International Conference on Soft Computing for Problem Solving (SocProS 2012), Jaipur, Rajisthan*, 2014.

[13] A. Cuomo, G. Di Modica, S. Distefano, A. Pulito, M. Rak, O. Tochio, S. Veque, and U. Villano, "An SLA-based broker for cloud infrastructures," *Journal of grid computing*, vol. 11, no. 1, pp. 1–25, 2013.

[14] J. L. Bruno, E. Gabber, B. Özden, and A. Silberschatz, "The eclipse operating system: Providing quality of service via reservation domains." in *Proceedings of 1998 USENIX Annual Technical Conference, New Orleans, Louisiana*, 1998.

[15] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European conference on Computer systems, Nuremberg, Germany*, 2009.