Software Engineering for Multi-tenancy Computing Challenges and Implications

Jia Ru, John Grundy
School of Software and Electrical Engineering
Swinburne University of Technology
Melbourne, Australia
{rjia, jgrundy}@swin.edu.au

Jacky Keung
Department of Computer Science
City University of Hong Kong
Hong Kong SAR
Jacky.Keung@cityu.edu.hk

ABSTRACT

Multi-tenancy is a cloud computing phenomenon. Multiple instances of an application occupy and share resources from a large pool, allowing different users to have their own version of the same application running and coexisting on the same hardware but in isolated virtual spaces. In this position paper we survey the current landscape of multi-tenancy, laying out the challenges and complexity of software engineering where multi-tenancy issues are involved. Multi-tenancy allows cloud service providers to better utilise computing resources, supporting the development of more flexible services to customers based on economy of scale, reducing overheads and infrastructural costs. Nevertheless, there are major challenges in migration from single tenant applications to multi-tenancy. These have not been fully explored in research or practice to date. In particular, the reengineering effort of multi-tenancy in Software-as-a-Service cloud applications requires many complex and important aspects that should be taken into consideration, such as security, scalability, scheduling, data isolation, etc. Our study emphasizes scheduling policies and cloud provisioning and deployment with regards to multi-tenancy issues. We employ CloudSim and MapReduce in our experiments to simulate and analyse multi-tenancy models, scenarios, performance, scalability, scheduling and reliability on cloud platforms.

Categories and Subject Descriptors

I.6 [**Computing Methodologies**]: Simulation and modeling; C.2.4 [**Distributed Systems**]: Cloud Computing; D.2.8 [**Software Engineering**]: Metrics

General Terms

Performance, Measurement, Software Development, Cloud Computing

Keywords

Multi-tenancy, Scheduling policies, Cloud computing, Resource allocation, Software development, SaaS

InnoSWDev'14 , November 16, 2014, Hong Kong, China

1. INTRODUCTION

Cloud computing is a new software system technology that enables dynamic and elastic resource allocation on consolidated physical computing resources using a combination of techniques from parallel computing, distributed computing, and platform virtualization technologies [30] [31]. Software engineering for cloud-based systems is a very new domain of research and practice requiring careful consideration of its characteristics with respect to traditional software development paradigms [30].

Cloud computing enables resource sharing but different kinds of resources reflect different levels of dynamic behaviour and a diversity of user demands. This makes resource management very complex. To make full use of the cloud's scalability and elasticity for cloud service providers, multi-tenancy is a key cloud characteristic that enables sharing the same service instance, computational resources, storage, etc among different tenants [6] [3]. In the multi-tenancy model, data and resources are deployed in the same cloud, controlled and distinguished via labeling for the unique identification of resources owned by individual users [34].

Use of multi-tenancy for cloud applications can help service providers lower costs through economical scalability, improve resource utilization through sharing hardware resources, improve ease of maintenance, as well as reduce service customization time [2]. In a multi-tenancy environment, new clients can access the same software, and thus scaling has fewer infrastructure implication for vendors [22]. Meanwhile, the Software as a Service (SaaS) model allows all the tenants (clients) to share compute and data infrastructure, reducing operational costs. This obviates the need to add applications and more hardware to data centers [22]. In addition, application deployment is much easier for the service provider, since only one service instance of the application needs to be deployed [6].

Due to the advantages of multi-tenancy, migrating single tenant applications to multi-tenancy has recently become a major focus for enterprises [7]. However, effectively realizing multi-tenancy requires the SaaS applications to capture, process and store data of different tenants in the same application instance, which involves the changing of SaaS technology and refining as well as developing the multi-tenancy technologies with regard to different factors [1].

The aim of our work reported here is to present the current state-of-art in engineering and developing multi-tenancy cloud applications, and to identify some significant factors and key

aspects that influence and restrict the development of such multi-tenancy technologies and solutions. The rest of paper is organized as follows: Section 2 presents the background of multi-tenancy on clouds and some of the major challenges faced. Section 3 surveys some key related work. Section 4 describes our set of experimental platforms and tools we are using to better understand multi-tenancy implications on the software engineering of cloud applications. Section 5 presents our future work in this area and Section 6 summarises our conclusions to date.

2. LITERATURE REVIEW

2.1 Cloud computing architecture

Cloud computing is defined as *"a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"*, by **The National Institute of Standards and Technology (NIST)**, which provides 3 service delivery models shown in Figure 1 [27].

Software as a Service (SaaS): provides a fully functional software system ready-to-use for its end-users, offering e.g. a variety of business process management, analysis and customer relationship management (CRM) tools. Most Software cloud services hosted on the cloud architecture are web-based applications that can be accessed by different client devices via a thin client interface with limited configurations. Salesforce.com is such a well-known SaaS exemplar.

Platform as a Service (PaaS): provides the development platforms hosted on top of computing infrastructure, where the applications and services can be set up, deployed and controlled. These provide much less hosting environment configuration and maintenance, compared to having to install these platforms or tools on local enterprise machines. Google App Engine is a well-known PaaS exemplar.

Infrastructure as a Service (IaaS): offers cloud application developers on-demand virtual machines to deploy their software, including data and compute platforms. Amazon Elastic Compute Cloud is the most well-known IaaS exemplar.

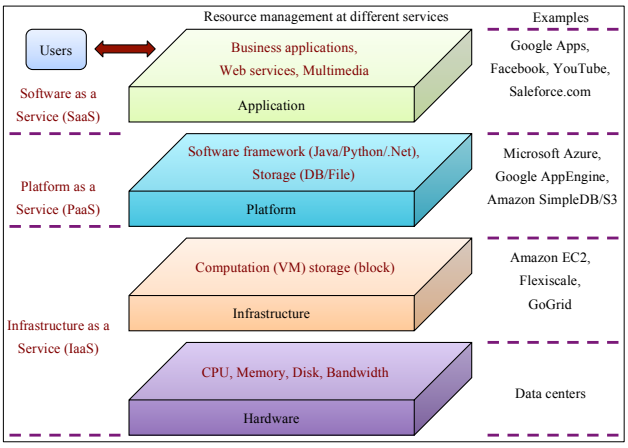

Figure 1: 3 main services delivery models [36]

2.2 Multi-tenancy implications

Multi-tenancy is a specific characteristic of cloud applications that can change the underlying economics of applications through sharing infrastructure, platform and services. It enables each cloud application "tenant", each with their own customers, processes and data, to obtain a single application instance [35]. A good definition of multi-tenancy [4] is *"Multi-tenancy refers to the architectural principle, where a single instance of the software runs on a software-as-a-service (SaaS) vendor's servers, serving multiple client organizations (tenants). Multi-tenancy is contrasted to a multi-instance architecture where separate software instances (or hardware and software systems) are set up for different client organizations. With a multi-tenant architecture, a software application is designed to virtually partition its data and configuration so that each client organization works with a customized virtual application instance. [32]"*.

2.2.1 Multi-tenancy-aware Cloud Applications

Multi-tenancy-aware applications can be defined as follow, they are suggested by Bezemer et al. [7] [5]:

*"A **multi-tenant application** lets customers (tenants) share the same hardware resources, by offering them one shared application and database instance, while allowing them to configure the application to fit their needs as if it runs on a dedicated environment."*

A multi-tenant application looks and behaves as tenants and similar to that of a single-tenant application, but in reality multiple tenants share the same cloud application services, platform and infrastructure.

To achieve multi-tenancy awareness, sophisticated multi-tenancy supports are required to be incorporated into the entire development and maintenance lifecycle of such applications [28]. The supports can be conveniently classified into two phases, they are the development phase and the deployment phase [28]. In the development phase, the multi-tenancy support is realized though high-degree configuration for different tenants. In the deployment phase, the service instances will be introduced and deployed in the target cloud infrastructure according to different multi-tenancy requirements of the applications [28].

To fully utilize the economy of scale offered by multi-tenancy cloud computing, services are usually hosted using multi-tenancy models (patterns) [24]. In service based applications, service multi-tenancy patterns can be classified into three main categories characterising the degree of customisability; the level of data isolation; the ease deployment and update; the scalability of the whole system; and so on [28]:

Single instance service: A service is the same for all the tenants, and if updating the application, the service will be only updated once for all the tenants. A single instance service can be adopted to deploy once and have the same behavior for all the tenants. It does not need special consideration regarding multi-tenancy. All the data used in the service is shared by all of the tenants, since the single instance service does not differ between the tenants. [28]

Single configurable instance service: A service has some specific tenant behavior. This means that a single service has different behavior for different tenants according to configuration parameters that describe processes and data used by an SaaS application to support specific tenant behavior [18]. It is difficult to update parts related to a specific tenant configuration, since all the tenants use the same in-

stance of a service and it needs to redeploy the configured service for each tenant. [28]

Multiple instances service: The service varies from one tenant to another, and an application needs specific tenant behavior for a service. Multiple instances services can be used to realize specific tenant behavior and each tenant or each group of tenants adopt their own service to achieve their own specific behavior. Each tenant is served by its own instance and it may result in load unbalance on a service. [28]

Compared with the multi-instance model, multi-tenancy customizes a single instance based on multiple requirements of different tenants. In comparison to the much more common multi-instance model, each tenant has their own virtualized instance of the application [7] [6] [16].

2.3 Achievement of multi-tenancy

In a multi-tenancy cloud, multiple vendors can use the same infrastructure to access and utilize an application. Figure 2 presents the overview of such multi-tenancy cloud architectures. Multi-tenancy at the datacenter layer can be a service provider renting datacenter space, and supplying servers, routers, etc, that supports multiple customer software requests [29]. Multi-tenancy at the infrastructure layer can be achieved through software stacks, where one stack belongs to a specific customer. Compared to datacenter-layer multi-tenancy, this infrastructure layer multi-tenancy saves costs because these stacks are deployed in accordance with actual customer accounts [29]. Implementation of multi-tenancy at the cloud application service layer requires architectural implementations at both the software layer and the infrastructure layer. Modifications are required to existing multi-instance software architectures, including a variety of multi-tenant patterns being used throughout the application layer [29].

There are 3 common methods used to achieve multi-tenancy in cloud applications [29]:

Database: this uses database configuration with data isolation provided at the application layer. Design of different aspects of the application is used to automatically modify their different behaviors for different tenants at runtime.

Virtualization: this uses software to create application hosting environments which can provide logical boundaries between each tenant, especially for IaaS. In addition, it can run multiple copies of server operating systems within one physical machine and share physical hardware.

Physical Separation: this depends on deploying each tenant with their own specific hardware resources. For instance, assigning separate physical servers to different tenants, or giving a significant section of a datacenter to a large client [35] [12].

2.4 Challenges of multi-tenancy

Multi-tenancy can help service providers to operate, customize, maintain, and upgrade a single instance [2]. Multi-tenancy has benefits but a number of major challenges are introduced. For example, all the tenants share the same resources so a failure caused by one tenant may adversely influence the others. Multi-tenancy data is stored in the same server, and it will introduce new requirements for data management, scalability, security, privacy, integrity and performance measures than in single-tenant based software [7].

2.4.1 Data management

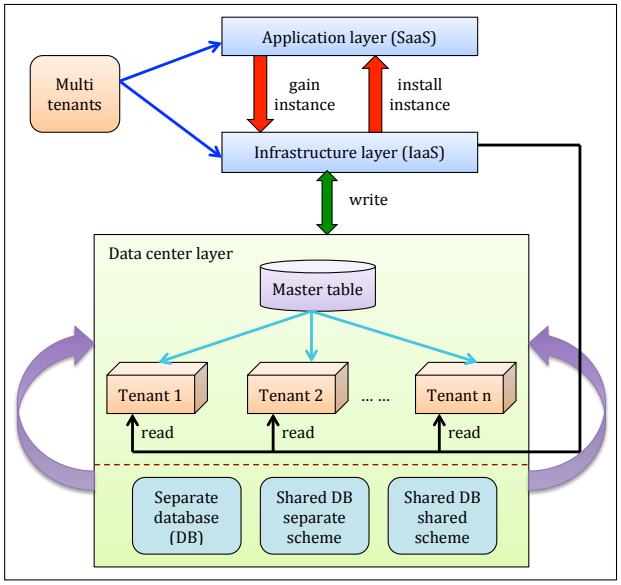

Figure 2: Multi-tenancy architecture overview [35]

Service providers need to design and implement a specific class, and create an object of the class, which serves the requirements of multiple users efficiently. Design of such SaaS applications can address some multi-tenancy issues, such as data security, data separation, data security isolation and customized applications to minimize the hard binding of runtime computing resources [12] [35]. Data architecture is an area where the optimal degree of isolation for a SaaS application can change based on technical and business considerations [16]. There are 3 approaches to managing multi-tenant data's isolation and sharing, lying at different locations [16]:

Separate Databases: store different tenants' data in separate databases. Computing resources and applications are shared among all the tenants on a server, however each tenant has its own data which keeps logically isolated from data belonging to all other tenants. Metadata is associated with each database with the correct tenant, and database security prevents any tenant from accidentally or maliciously accessing other tenants data [16]. This maximises flexibility and reduces data integrity, security and scaling issues, but complicates implementation and deployment.

Shared Database, Separate Schemas: house multiple tenants in the same database, with each tenant having their specific tables which are grouped into a schema for the tenant [16].

Shared Database, Shared Schema: utilize the same database and the same tables to host multiple tenants' data [16]. This minimises configuration but reduces flexibility and introduces data integrity, security and potentially scaling issues.

2.4.2 Security

Multi-tenancy has different possible deployment models from a separate instance for each tenant to a single instance for all multiple tenants. However, it requires the SaaS application to handle multi-tenancy and tenant data and process isolation themselves or hand the task of processing and

storage over third parties. This introduces new security and privacy issues not presented in single tenant, multi-instance applications. There is a requirement to build adequate security measures into every aspect of the SaaS application, and for every IaaS virtual service. We present these security issues that rely on three underlying patterns: [16] [1].

Filtering: Use an intermediary layer between a tenant and a data source acting as a sieve, which enables the tenant not know about the existence of other tenants and makes it appear to the tenant as if the tenant's data is the only data in the shared database.

Permissions: Use access control lists (ACLs) to determine data rights (who can access data in the application) and data processing models (what data operation is allowed).

Encryption and Obfuscation: Obscure critical data and/or processing of each tenant, in order to keep it inaccessible to unauthorized parties.

2.4.3 QoS - Performance

A multi-tenant SaaS provides multiple end users with the same functionality but with potentially different quality-of-service (QoS) values, such as response time, throughput, reliability, availability, scalability, reusability, efficiency service period, least resource cost, etc [21] [11] [25].

Reusability: It measures whether functionalities provided by services are common to the requirements defined by service consumers [25].

Availability: It measures the ratio of the total time to the time which a SaaS service is capable of being operable [25].

Scalability: It measures the ability to handle or readily enlarge the growing resources [25].

Reliability: It measures the ability of multi-tenants to keep operating with specified level of performance over time or the reliability of the services themselves [25].

To realize negotiation between service providers and tenants, service level agreements (SLAs) are used to configure service resources according to different service requirements [11]. Different tenants may have very different SLA requirements that need to be met by the same shared cloud application.

2.4.4 Maintenance

The multi-tenancy deployment can dramatically reduce the number of application and database instances that need to be configured and updated, especially if there are many tenants for the same application. In addition, end users don't need to pay expensive maintenance fees to keep their software up to date. New features and updates are often included with a SaaS subscription and are rolled out by the vendor [22]. However, introducing multi-tenancy into a software system will increase its complexity, sometimes dramatically, and it will influence the maintenance process. Therefore hardware deployment and scheduling should consider whether the increased benefits outweigh the increased maintenance costs [6].

2.4.5 Scalability

Since all the tenants share data and application services, scalability is distinguishably significant for SaaS [6] [16]. Databases can be scaled up by strengthening a larger server with more powerful processors, more memory, etc and scaled out by partitioning a database onto multiple servers. Developing a scaling strategy should differentiate between scaling applications (increasing the total workload the application can accommodate) and scaling data (increasing capacity for storing and working with data). Replication and partitioning are two core tools to realize scalability. However multi-tenancy scaling has many more constraints than single tenant, multi-instance applications. These include the requirement of placing all data for one tenant on the same server to improve the speed of utilizing database queries. [6] [16].

2.4.6 Scheduling and resource provisioning

Multi-tenancy can scale up and down computing resources, as well as allocate resources according to actual usage, which is most widely used for SaaS applications. The scheduling strategies used in multi-tenancy cloud then become important, which directly influence the runtime performance of software applications. Eventually, scheduling policies should effectively improve the number of completed transactions, increase profit of service party, reduce cost which is undertaken by service party when accepting applications and guarantee QoS (Quality of Service) demand of clients [30].

For example, the priority scheduling of multi-tenancy-aware applications can achieve two goals [11] through processing different resources' requests according to their SLA levels: (1) The tenants with high priority of QoS will be queued prior to the ones of low priority; (2) Under the guarantee of service quality of tenants in high priority, the scheduling policies prevents the influence on low tenants. As another example, scheduling approaches focusing on multi-tenant, instance-intensive workflows should consider another three aspects: (1) the quality of service experience (QoSE) of tenants in different SLA, (2) mean execution time of multiple workflow instances and (3) save the execution cost for service providers [13].

Resource provisioning of new tenants is based on selection from a catalogue released by the cloud vendor to solve peak load and data expectations among multi-tenants [17]. Before resource provisioning, the calculation of resource requirements (such as CPU, storage, etc) for the multi-tenancy in a shared application instance is needed. This must satisfy some constraints (such as response time, availability, business transaction rate, database request rate, etc) and minimise cost, but without violating Service Level Agreement (SLA) requirements. This is a very complex task to perform and to engineer software applications to support [23].

3. RELATED WORK

Even though SaaS attracts much attention in both research community and IT industries, possible solutions to provide multi-tenancy support are still under investigation. For example, the SMURF framework [1] helps service providers reengineer their legacy applications in multi-tenancy environments and automates system updating. The MDSE@R model [2] supports capturing, enforcing, and verifying different tenants' and service providers' security requirements at runtime without modifying the underlying application. MSSOptimiser [21] is a QoS driven approach to support service selection for multi-tenancy SaaS. This helps service developers select appropriate services to compose an optimal SaaS that meets different QoS requirements of multiple stakeholders.

An SLA-based scheduling algorithm [11] guarantees service quality of tenants and improves the system performance

using request load as the measure of resource utilization. A scheduling algorithm for multi-tenancy workflow instances proposed in [13] improves the quality of service (QoS) for tenants and saves the execution cost of workflows. A Tenancy Requirements Model (TRM) presented in [17] based on mapping of functional and non-functional tenancy requirements with appropriate resources, their parameters, and health monitoring policy allows dynamic re-provisioning for existing tenants based on either changing tenancy requirements or health grading predictions. A multi-tenancy placement tool for application deployment using a minimum number of servers is proposed in [23]. A framework with a set of multi-tenancy common services proposed in [18]helps people design and implement a high quality native multi-tenant application. However, to date none of these provide a wide ranging and effective multi-tenancy solution for complex cloud applications.

4. EXPERIMENTAL TOOLS

Multi-tenancy is a relatively new concept in cloud computing. We review three significant tools that are being used to model, understand and deploy such approaches. Cloudsim can simulate multi-tenancy environments to enable fast experimental simulation and verification of proposed ideas before deploying the approaches on real cloud platform. Hadoop provides a platform to realise MapReduce-style implementations. StressCloud is a tool to model cloud application data, compute and communication characteristics and cloud load models, and generate real virtual machine provisioning and deployment testbeds.

4.1 Cloudsim

CloudSim is an extensible simulation platform which enables seamless modeling, simulation, and experimentation of emerging Cloud computing infrastructures and management services [8]. It is a toolkit for simulation of different cloud scenarios, such as multi-tenancy, heterogeneous resources and communication environments. We are usung it to build a range of multi-tenant and multi-instance cloud scenarios [26]. The CloudSim architecture contains 4 layers: SimJava, GridSim, CloudSim and User code [8]. Figure 3 illustrates the layered implementation of the CloudSim software framework and architectural components.

At the lowest layer is the SimJava discrete event simulation engine that implements the core functionalities required for higher-level simulation frameworks. The next layer is GridSim, a toolkit that supports high level software components for modeling multiple Grid infrastructures [8] [9]. The Simulation layer provides support for modeling and simulation of virtualized Cloud-based data center environments including dedicated management interfaces for VMs, memory, storage, and bandwidth [8] [9]. The top-most User Code layer exposes configuration related functionalities for hosts, applications, VMs, their application types, scheduling policies, etc [8] [9].

4.2 Hadoop MapReduce

Hadoop is an open source implementation of Google MapReduce, which allows for multiple-tenants to securely share a large cluster such that their applications are allocated resources in a timely manner [33]. Hadoop architecture mainly contains 4 components: Client, JobTracker, TaskTracker and Hadoop Distributed File System (HDFS). MapReduce

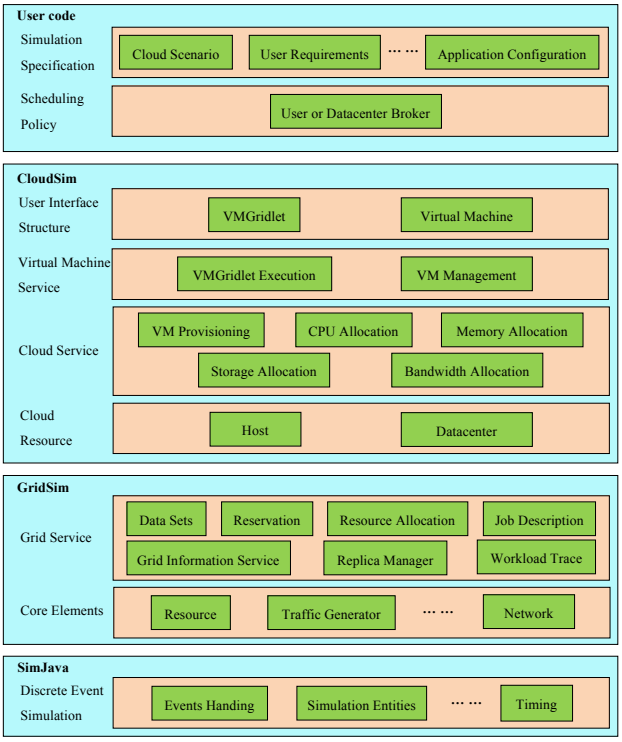

Figure 3: Layered CloudSim architecture [8] [9]

system partitions input data and schedules the execution of programs in clusters of commodity machines. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key. The functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance [15] [14]. To achieve multi-tenancy, the Fair Scheduler or Capacity Scheduler can provide SLAs and guaranteed resource availability to tenants appropriately [33].

Capacity Scheduler: It is designed to run Hadoop MapReduce as a shared, multi-tenant cluster in an operator-friendly manner while maximizing the throughput and the utilization of the cluster while running Map-Reduce applications. It allows sharing a large cluster while giving each queue a minimum capacity guarantee. When a task is submitted, it is put into a queue. Each queue gets some TaskTracker resources based on configuration to process Map and Reduce operations. Available resources can be dynamically allocated to the queues with heavy workloads. In a queue, tasks can be operated according to their different priorities. High-level priority task will be executed first, but Capacity Scheduler does not support preempting priority. Comprehensive set of limits are provided to prevent a single job, user and queue from monpolizing resources of the queue or the cluster as a whole to ensure that the system, particularly the JobTracker, isn't overwhelmed by too many tasks or jobs. Sharing clusters across organizations necessitates strong support for multi-tenancy since each organization must be guaranteed capacity and safe-guards to ensure the shared cluster is impervious to single rouge job or

user. Nevertheless, this scheduling policy cannot automatically set up configuration of queues and also cannot choose queues by itself [19].

Fair Scheduler: It supports classification of tasks and allocates different types of resources to the various types of tasks in order to improve performance quality. The fair scheduler organizes jobs into pools, divides resources fairly between these pools and also allows assigning guaranteed minimum shares to pools. By default, there is a separate pool for each user, so that each user gets an equal share of the cluster. It allows assigning guaranteed minimum shares to pools, which is useful for ensuring that certain users, groups or production applications always get sufficient resources. When a pool contains jobs, it gets at least its minimum share, but when the pool does not need its full guaranteed share, the excess is split between other pools. It is also possible to set a job's pool based on a configurable attribute, such as user name and unix group. Within each pool, jobs can be scheduled using either fair sharing or FIFO scheduling [33]. However, this scheduling policy does not consider actual workload of the task nodes and it results in workload unbalance. At runtime, the actual workload of a task node is determined by resource consumption of the running tasks rather than number of those [20].

4.3 StressCloud

StressCloud [10] is a new performance profiling and energy consumption analysis tool for cloud applications. It supports users to generate cloud application services comprised of data, computing and communication tasks to realize load test and automatically deploy these load test services. It can model realistic cloud application workloads and stress test applications generated from models or real, deployed applications. As StressCloud can deploy and run cloud services in a real cloud deployment environment, its use results in more precise and realistic estimation of cloud application performance and behavior. StressCloud currently supports VMware as the cloud platform hypervisor. Cloud services are composed of composite tasks which relate to various types of cloud resources, such as data storage (Hard Disk), CPU, RAM, and network devices. In a multi-tenant environment, since each tenant has their own requirements and specific behaviors, the tasks of different tenants are very complex and composite, which results in the unbalanced and diversity workload. However, StressCloud can generate these kinds of composite tasks as well as form a workload model based on transition probabilities and properties between different types of tasks. A set of services are developed to model the target applications. Multi-tenancy configuration and customization can be realized through using services in SOA-based applications [28]. StressCloud allows these services to take the user requests in order to execute tasks defined in the workload model and send the corresponding responses. For each task, a stochastic form chart is built to describe the detailed user requests and required response from the cloud services [10]. The tasks can be classified as computation-intensive tasks, data-intensive tasks, and communication-intensive tasks and StressCloud can model all of these and mix different task types according to different services: [10] (1) Computation-intensive tasks: It mainly cost CPU and RAM resources, so it can be divided into CPU-intensive tasks and memory-intensive tasks. (2) Data-intensive tasks: It is local and global I/O bound in common, requires to process large volumes of data, and spend most time on moving and manipulating data in databases and files. (3) Communication-intensive tasks: In a cloud application, it usually generates a large number of network transactions between cloud user devices and cloud systems.

In a cloud computing system, scheduling is the act of dispatching these tasks to a pool of resources providing high processing capability to achieve some optimized objectives, such as minimize cost, reduce turnaround time, etc, or to achieve required QoS levels as specified in a SLA. Therefore, StessCloud is a significantly useful tool to model different task scheduling schemes and run these in a real cloud deployment environment. This can help us to test different scheduling policy effects and efficiency for multiple tenants.

5. DISCUSSION

In the multi-tenancy cloud environment, adopting an appropriate scheduling policy and optimizing scheduling mechanisms to improve the performance and utilization of resources as well as guarantee QoS of services according to different requirements of tenants are of a significant research area [30]. Each tenant has its own QoS requirements for processing tasks, such as processing time, processing cost and processing priority. However, current scheduling supports for multi-tenancy are very limited or overly simplistic (one-size-fits-all). Therefore, we are interested in investigating a dynamic scheduling model to improve the efficiency of the entire system for multi-tenant cloud applications, allowing tailoring of schedulers to different tenant SLAs.

Figure 5 presents an overview of this. Here, each scheduling parameter is regarded as one component/module. There are several aspects to consider:

(1) how to realize initialization: According to our hypothesis, we will build a scheduling model and configure different scheduling schemes. How to define/describe these schemes is a key first goal.

(2) how to find the relationship between parameters: There are many parameters in scheduling mechanisms and how to find the relationship between all these parameters (such as resource utilization-based, priority-based, performance-based, cost-based, etc.) as well as balance them is necessary, since all these parameters are not independent. For example, the overall cost will effect the profit directly. Taking into account of the priority, the task with high-priority is processed first, but this may reduce the entire performance. Considering the QoS parameter, choosing a suitable scheduling policy maximises system performance but cost may be increased. Changing one parameter will influence other parameters. Consequently, it is very complex to find a threshold or key value to balance these parameters.

(3) how to improve efficiency: Based on the different scheduling schemes, how to choose a suitable parameter as main focus to improve the efficiency is a challenge. We need to define the thresholds to determine which schemes corresponding to which parameters. Furthermore, in a fixed scheduling scheme, we need to find out which parameter should be considered first, which parameter is most important and what is the weight of this parameter. In the initial phase, to reduce the complexity, one scheduling scenario will only consider one or two parameters. In the final phase, many more parameters will be considered in one scheme, and the weight of each parameter will be calculated.

(4) how to build in scheduling approaches for each

component: In scheduling mechanisms, there exists numerous of algorithms even if only considering one parameter. Therefore, how to choose suitable algorithms building in each component and how many algorithms being selected in one component is another key. Additionally, a more complicated multi-tenancy scheduling model may not only focus on tasks, but also may consider hardware resource provisioning (from the virtual machine layer), data placement, and business process models.

In Figure 5, tasks generated by StressCloud will be randomly labeled by user 1, user 2 ... user n, respectively. Firstly, each user will only add at most one parameter to consider, such as deadline (related to completion time), response time (which means the task with the less response time constraint has the higher priority), cost and availability. The first 2 parameters obey a normal distribution. Availability can be set manually using timelines and periods of unavailability. Some users need 24/7 availability and some VMs will be put into an availability pool. When considering cost, a user may choose not consider completion time. When all the machines are not busy their request will be processed. In addition, the tasks with hard deadline constraints should be ordered in a queue. Some VMs should be put into a deadline pool and these tasks are deployed to this pool. From the resource provisioning perspective, CPU and memory usage should be monitored. When some VM's usage is lower than a pre-defined threshold, the VM will be shut down untill all the current tasks are finished. If the usage of each VM is higher than the threshold, a new VM will be spooled up and added. The VMs availability in the pool will never be stopped and under the worst condition, when there are almost no new tasks coming and no unfinished tasks, at least 2 VMs are kept live.

In summary, all these aspects are highly dependent. To reduce complexity, we will build our model considering one aspect at a time. For example, the main objective of the scheduling model is to minimize the completion time shown in Figure 4. There are 4 modules in the model: cost-focus (minimize the cost), priority-focus (minimize waiting time), QoS-focus (satisfy all the QoS attributes), and minimized completion time-focus. All the tenants submit their tasks to the scheduling broker (the core of scheduling model to manage/control other modules, and deploy tasks to the virtual machines), which needs to meet not only the fundamental QoS requirements but also their own specific requirements (cost-focus, priority-focus, minimized completion time-focus, etc). Therefore, we need to calculate a final weight value for each task based on these requirements, in order to rank all the submitted tasks according to their weight values to determine optimal task execution sequence. Therefore, for the first layer, we need to choose a suitable algorithm to calculate the multi-dimensionality QoS value. For the second layer, we justify the first task execution order through launching different adaptive algorithms to different specific requirements, combining these to a matrix and multiplying to the first execution sequence. Finally, according to the final execution order, we choose priority scheduling to allocate the tasks to the virtual machines. Importantly, the adaptability is considerate factor, since different types tasks will be loaded. In the initial phase, we will not consider the difference of task types.

In the initial phase, through investigating different scheduling polices, a task scheduling algorithm has been proposed

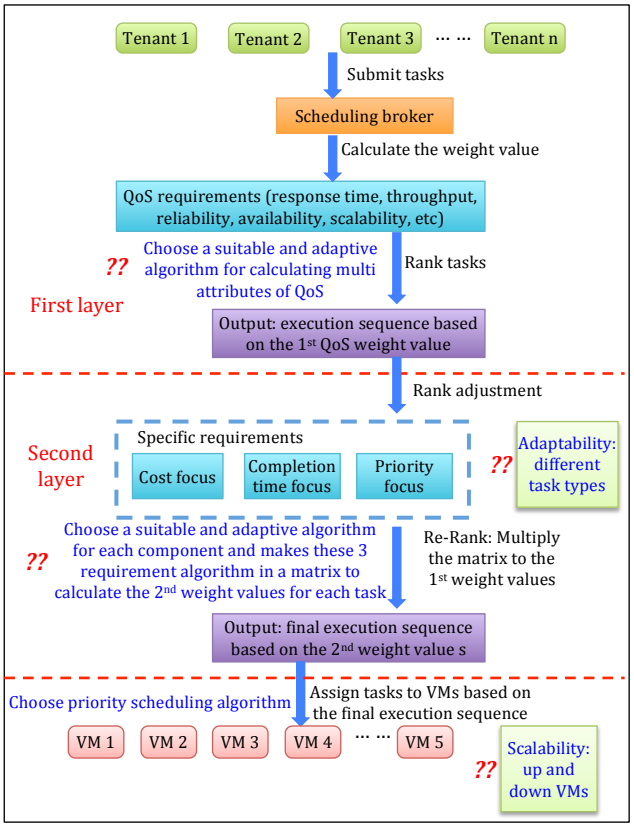

Figure 4: Implementation of scheduling model

that integrates with task grouping, prioritization (bandwidth-aware) and SJF (Shortest-Job-First) [30]. The proposed algorithm has been simulated on the Cloudsim platform that only considers the tasks themselves without taking into account of multi-users, or the multi-instancy issue. It aims to effectively minimize task's waiting time and processing time as well as to reduce tasks' processing cost through effective utilization of cloud resources available.

Compared with traditional grouping-based scheduling algorithms, our proposed algorithm and model is suitable for both very lightweight jobs and the tasks with random and unpredictable processing requirements. It also considers the communication and transmission rate of resources to reduce the transmission latency. Meanwhile, it could achieve optimum resource utilization and minimum overhead, as well as reduced influence of bandwidth bottleneck in communications. Figure 6 shows the main parts of our proposed algorithm. This involves sorting jobs (step 1); identifying and organising resources (step 2); analysing jobs and resources (steps 3, 4); and grouping tasks and resource allocation (step 5); and scheduling activities (step 6).

Figure 7 shows the total processing time of different number of tasks from some of our preliminary analysis of possible algorithms. Figure 8 presents the average waiting time of different number of tasks. In comparison with existing task grouping algorithms, our preliminary scheduling results show that the proposed algorithm waiting time significantly decreased over 30% and processing time decreased by 7.25%. This algorithm reduces makespan greatly on the simulation platform, so from a cloud provider respective, ordered tasks

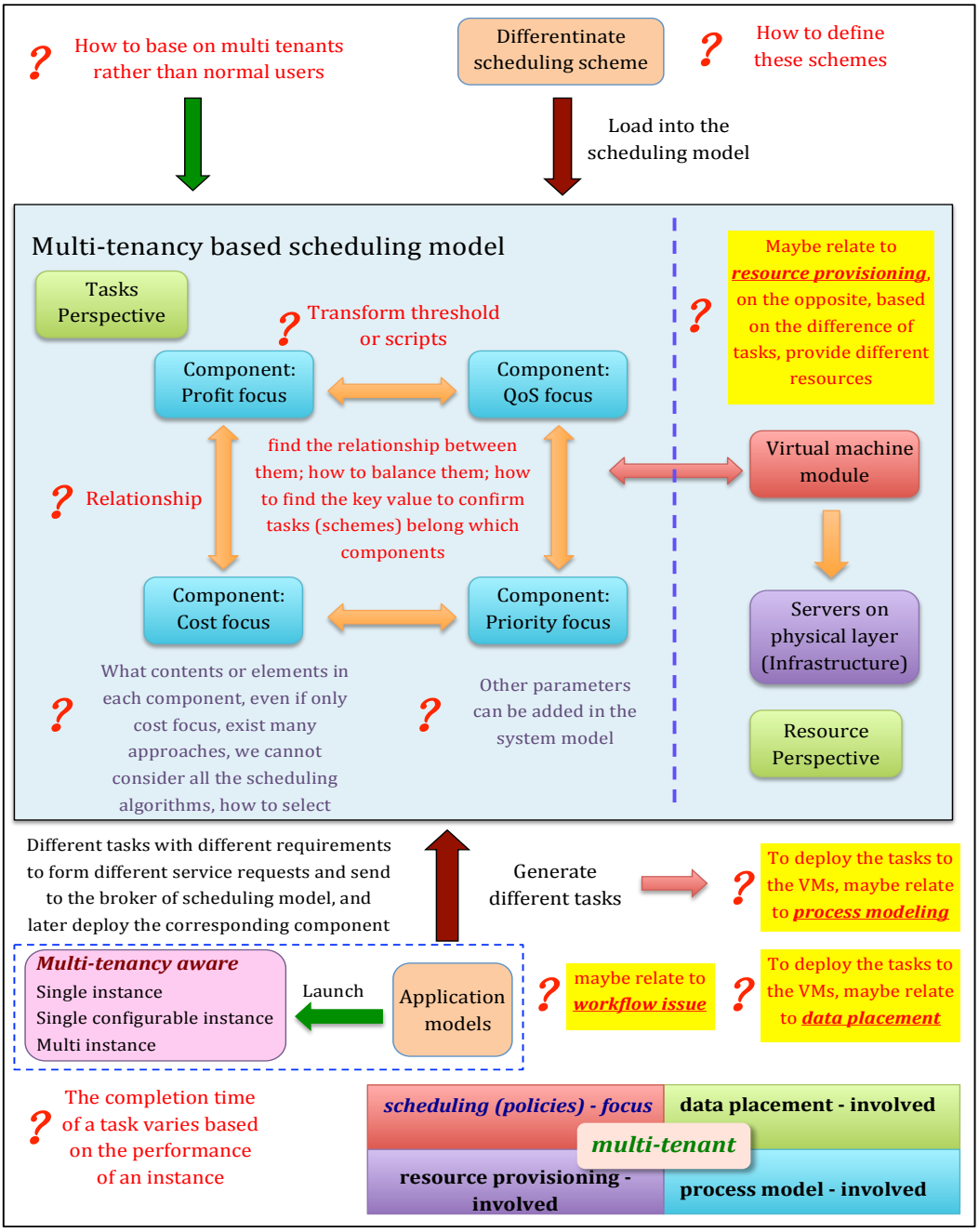

Figure 5: An overview picture of our key future work

of different tenants can be grouped using this algorithm and sent to the virtual machines in our real cloud.

Good modelling, analysis, simulation and testing tool support is needed. This allows engineers to specify multi-tenant cloud application scheduling, provisioning, task, workload, and QoS models. These provide models allowing engineers to reason about appropriate strategies for different tenant requirements. We plan to extend StressCloud to incorporate richer QoS requirement specifications, multi-tenant requirements, schedulers and provisioning specifications. We plan to extend its code generation to support these in target application load and executable model generation.

6. CONCLUSION

Multi-tenancy is a new software architectural pattern with a single instance of applications (or customizing the data and configuration) running on service provider's infrastructure. However, the architecture of multi-tenancy and the customization requirements attract several research challenges, such as the possibility of sharing hardwares resource at a lower cost, utilization of shared application and database to ease the maintenance efforts required, and to provide a high-level configuration to customize a specific workflow instance of the applications to end users, etc. [7] To realize multi-

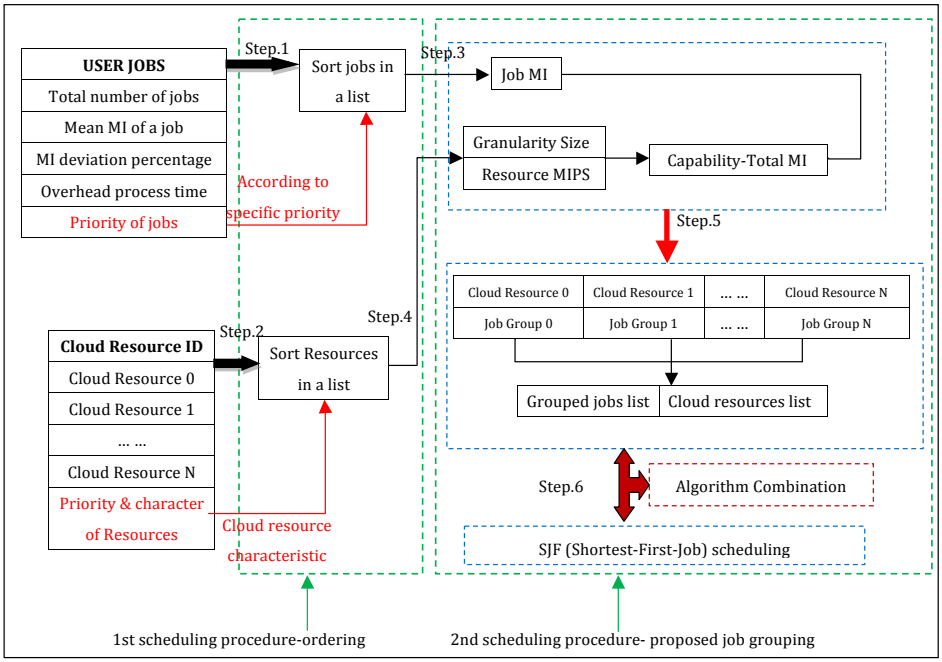

Figure 6: Flow chart of proposed algorithm [30]

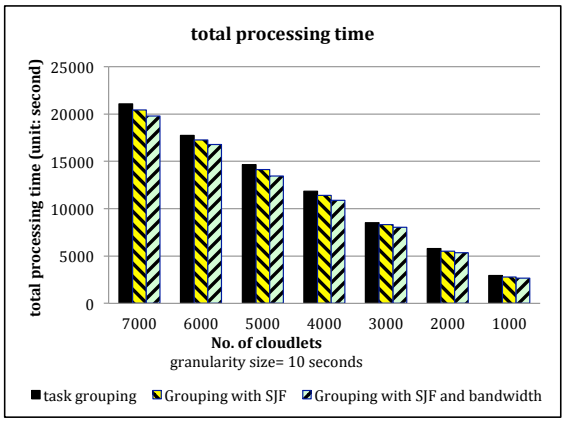

Figure 7: Total processing time with different number of tasks

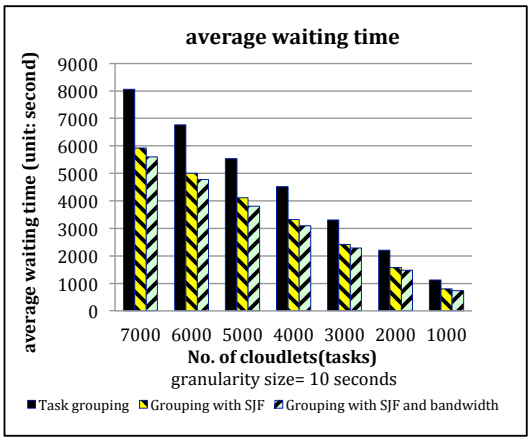

Figure 8: Average waiting time with different number of tasks

tenancy efficiently, some significant parameters must be considered, such as scalability, data management, maintenance, security, and scheduling implications just to name a few. In this study, we have provided a comprehensive overview of key research trends and future directions in an emerging area of software engineering in the cloud environment. We are working on building a generic scheduling model that will deal with multi-tenancy issues, with considerations of QoS constraints and resource provisioning capabilities. The response time, deadline and availability as 3 important elements will be first considered in the current research.

7. REFERENCES

[1] M. Almorsy, J. Grundy, and A. S. Ibrahim. Smurf: Supporting multi-tenancy using re-aspects framework. In *Engineering of Complex Computer Systems (ICECCS)*, pages 361–370. IEEE, 2012.

[2] M. Almorsy, J. Grundy, and A. S. Ibrahim. Adaptable, model-driven security engineering for saas cloud-based applications. *Automated Software Engineering*, pages 1–38, 2013.

[3] M. Almorsy, J. Grundy, and I. Müller. An analysis of the cloud computing security problem. In *the proc. of the 2010 Asia Pacific Cloud Workshop, Colocated with APSEC2010, Australia*, 2010.

[4] D. Banks, J. Erickson, and M. Rhodes. Multi-tenancy in cloud-based collaboration services. *Information Systems Journal*, 2009.

[5] C.-P. Bezemer and A. Zaidman. Challenges of reengineering into multi-tenant saas applications.

[6] C.-P. Bezemer and A. Zaidman. Multi-tenant saas applications: maintenance dream or nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 88–92. ACM, 2010.

[7] C.-P. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. t Hart. Enabling multi-tenancy: An industrial experience report. In *Software Maintenance (ICSM)*, pages 1–8. IEEE, 2010.

[8] R. Calheiros, R. Ranjan, C. De Rose, and R. Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*, 2009.

[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[10] F. Chen, J. Grundy, J.-G. Schneider, Y. Yang, and Q. He. Automated analysis of performance and energy consumption for cloud applications. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 39–50. ACM, 2014.

[11] X. Cheng, Y. Shi, and Q. Li. A multi-tenant oriented performance monitoring, detecting and scheduling architecture based on sla. In *Pervasive Computing (JCPC), 2009 Joint Conferences on*, pages 599–604. IEEE, 2009.

[12] F. Chong. Multi-tenancy and virtualization. In *http://blogs.msdn.com/b/fred_chong/archive/2006/10/23/multi-tenancy-and-virtualization.aspx*, 2006.

[13] L. Cui, T. Zhang, G. Xu, and D. Yuan. A scheduling algorithm for multi-tenants instance-intensive workflows. *Applied Mathematics & Information Sciences*, 7, 2013.

[14] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[15] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008.

[16] G. C. Frederick Chong and R. Wolter. Multi-tenant data architecture. In *http://msdn.microsoft.com/en-us/library/aa479086.aspx*.

[17] A. Gohad, K. Ponnalagu, and N. C. Narendra. Model driven provisioning in multi-tenant clouds. In *Global Conference (SRII), Annual*, pages 11–20. IEEE, 2012.

[18] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. A framework for native multi-tenancy application development and management. In *E-Commerce Technology and the 4th International Conference on Enterprise Computing, E-Commerce and E-Services*, pages 551–558. IEEE, 2007.

[19] Hadoop. Capacity scheduler. In *http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html*.

[20] Hadoop. Fair scheduler. In *http://hadoop.apache.org/docs/r2.2.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html*.

[21] Q. He, J. Han, Y. Yang, J. Grundy, and H. Jin. Qos-driven service selection for multi-tenant saas. In *Cloud computing (cloud)*, pages 566–573. IEEE, 2012.

[22] F. H. Judith Hurwitz, Marcia Kaufman and D. Kirsch. Multi-tenancy and its benefits in a saas cloud computing environment. In *http://www.dummies.com/how-to/content/multitenancy-and-its-benefits-in-a-saas-cloud-comp.html*.

[23] T. Kwok and A. Mohindra. Resource calculations with constraints, and placement of tenants and instances for multi-tenant saas applications. In *Service-Oriented Computing, ICSOC*, pages 633–648. Springer, 2008.

[24] T. Kwok, T. Nguyen, and L. Lam. A software as a service with multi-tenancy support for an electronic contract management application. In *Services Computing, 2008. SCC'08. IEEE International Conference on*, volume 2, pages 179–186. IEEE, 2008.

[25] J. Y. Lee, J. W. Lee, S. D. Kim, et al. A quality model for evaluating software-as-a-service in cloud computing. In *Software Engineering Research, Management and Applications, SERA. 7th ACIS International Conference*, pages 261–266. IEEE, 2009.

[26] R. Malhotra and P. Jain. Study and comparison of various cloud simulators available in the cloud computing. *International Journal*, 3(9), 2013.

[27] P. Mell and T. Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

[28] R. Mietzner, T. Unger, R. Titze, and F. Leymann. Combining different multi-tenancy patterns in service-oriented applications. In *Enterprise Distributed Object Computing Conference, 2009. EDOC'09. IEEE International*, pages 131–140. IEEE, 2009.

[29] J. Petersson. Best practices for cloud computing multi-tenancy. In *http://www.ibm.com/developerworks/cloud/library/cl-multitenantcloud/*.

[30] J. Ru and J. Keung. An empirical investigation on the simulation of priority and shortest-job-first scheduling for cloud-based software systems. In *Software Engineering Conference (ASWEC), 2013 22nd Australian*, pages 78–87. IEEE, 2013.

[31] S. Selvarani and G. Sadhasivam. Improved cost-based algorithm for task scheduling in cloud computing. In *Computational Intelligence and Computing Research (ICCIC), IEEE International Conference on*, 2010.

[32] B. Warfield. Multitenancy can have a 16:1 cost advantage over single-tenant. 2007.

[33] T. White. *Hadoop: The definitive guide*. O'Reilly Media, 2012.

[34] K. Wood and M. Anderson. Understanding the complexity surrounding multitenancy in cloud computing. In *e-Business Engineering (ICEBE), 8th Conference on*, pages 119–124. IEEE, 2011.

[35] L.-J. Zhang, J. Fiaidhi, I. Bojanova, and J. Zhang. Enforcing multitenancy for cloud computing environments. *IT professional*, 14(1):0016–18, 2012.

[36] Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.