The Computer Journal Advance Access published September 26, 2013

doi:10.1093/comjnl/bxt107

Performance Modelling and Simulation of Three-Tier Applications in Cloud and Multi-Cloud Environments

NIKOLAY GROZEV* AND RAJKUMAR BUYYA

Department of Computer Science and Information Systems, Cloud Computing and Distributed Systems (CLOUDS) Laboratory, The University of Melbourne, Parkville, Australia
**Corresponding author: ngrozev@student.unimelb.edu.au*

A significant number of Cloud applications follow the 3-tier architectural pattern. Many of them serve customers worldwide and must meet non-functional requirements such as reliability, responsiveness and Quality of Experience (QoE). Thus the flexibility and scalability offered by clouds make them a suitable deployment environment. Latest developments show that using multiple clouds can further increase an application's reliability and user experience to a level that has not been achievable before. However, the research in scheduling and provisioning 3-tier applications in clouds and across clouds is still in its infancy. To foster the research efforts in the area, we propose an analytical performance model of 3-tier applications in Cloud and Multi-Cloud environments. It takes into account the performance of the persistent storage and the heterogeneity of cloud data centres in terms of Virtual Machine (VM) performance. Furthermore, it allows for modelling of heterogeneous workloads directed to different data centres. Based on our model, we have extended the CloudSim simulator, through which we validate the plausibility of our approach. The conducted experiments with the RUBiS workload show that the predicted performance characteristics by the simulation approximate well those of the modelled system.

Keywords: cloud computing; simulation; performance model; multi-cloud

Received 13 April 2013; revised 12 July 2013
Handling editor: Rada Chirkova

1. INTRODUCTION

Cloud computing is an IT model enabling on-demand access to computing resources as a subscription service. Cloud service providers create and maintain large data centres to provide their clients with on-demand computing resources [1, 2]. Clients access and use external resources dynamically in a pay-as-you-go manner. This proves to be very appealing to businesses as it provides greater flexibility and efficiency than maintaining local infrastructure that is underutilized most of the time while at times it may be insufficient.

Many view Cloud computing as an extension of Grid computing, which also envisions on-demand access to a pool of computing resources. One major distinction between the two is the type of applications that are usually hosted. Grids are used for resource-intensive batch processing applications, while Cloud applications are much more general purpose and also include interactive online applications [3]. Practice has shown that Clouds are suitable for interactive 3-tier applications. By 3-tier architecture we denote the architecture of the application, not the data centre. Enterprises hosting 3-tier applications in cloud environments range from e-commerce businesses like *ebay* [4], hosted in a hybrid cloud [5], to government agency web sites, including the US Department of Treasury, hosted in Amazon EC2 [6]. However, the cloud deployment of such applications also raises several challenges.

A cloud service interruption may have a severe impact on clients who are left without access to essential resources [2], as highlighted by several recent Cloud outages [7, 8]. Furthermore, for many applications the geographical location of the serving data centre is essential because of either legislative or network latency considerations. Such reliability, legal and QoE requirements are of special importance for large interactive applications which serve customers worldwide and need to be continuously available. Thus some large-scale applications need to use multiple clouds (i.e. a Multi-Cloud) [1]. This has been

exemplified by several industry reports including IBM's case study about the usage of three private data centres to host the website of the Australian Open tennis championships [9].

There are significant differences between the characteristics of batch processing and interactive 3-tier applications. In essence, while batch processing programs execute without any user interaction, interactive applications have to respond to user interactions constantly and in a timely manner. This imposes substantially different requirements for application scheduling and provisioning. Foster *et al.* highlight that Clouds, unlike Grids, are especially suitable for interactive real-time applications [3]. Hence, the development of provisioning and scheduling techniques for interactive applications is an important research area. Unfortunately, most research works in the area focus on resource intensive Grid-like batch processing applications, leaving interactive ones beyond their scope. For example, Sotiriadis *et al.* present a literature review of meta-scheduling approaches in cloud environments and discuss their applicability in the area of cross-cloud scheduling [10]. They overview 18 meta-scheduling approaches, all of which schedule batch processing applications and none of them schedules interactive ones. In addition, in our previous work we have also outlined that most existing approaches for application brokering across clouds specialize in batch processing applications only [11]. This is not reflective of the more general nature of Clouds, which in contrast to Grids tend to host a wider range of business applications, many of which are interactive and follow the 3-tier reference architecture.

There are many impediments to the research in interactive application provisioning and scheduling in Clouds. Firstly, installing and configuring different middleware and software components (e.g. application and database (DB) servers and load balancers) can be complex and laborious. Another problem is the selection of appropriate workloads and populating the DB with suitable data in order to test different scenarios in practice. Last but not least, the incurred financial costs for repeatedly performing multiple tests on large-scale applications can be significant.

Historically, similar problems have been encountered by researchers in the area of batch processing applications in Grids and Clouds. These have been resolved by introducing formal performance models and simulation environments that allow for the evaluation of solutions without deploying and executing large-scale systems. For example, previous works on distributed systems simulation have significantly fostered the research efforts and have been used in both academia and industry for quick evaluation of new approaches. Unfortunately, existing simulators like GridSim [12], CloudSim [13] and GreenCloud [14] can only be used to simulate batch processing and infrastructure utilization workloads and are not suitable for interactive 3-tier applications.

With this work, we aim to fill this gap and make *contributions* that bring the benefits of modelling and simulation to the area of 3-tier application provisioning and scheduling in Cloud and Multi-Cloud. More specifically we (i) propose a novel analytical model for 3-tier applications; (ii) define algorithms for implementing a simulator based on the model; (iii) describe our enhancement of CloudSim supporting modelling and simulation of such applications and (iv) validate our model through comparison with an actual system. In this work, we walk the entire path from defining an abstract, theoretical performance model to implementing a full scale simulator based on the model, which is used to validate its adequacy. To the best of our knowledge, this is the first work in the area to do so. We identify and address the shortcomings of existing approaches and thus in our model we incorporate the following aspects, which are not accounted for by most related works:

(i) *Performance Variability*—in a cloud environment, VMs' performance can vary depending on their placement among the physical hosts. VMs with equal characteristics may observe significant performance differences [15].
(ii) *Disk I/O performance*—many 3-tier applications perform a huge amount of disk operations, which often cause a performance bottleneck.
(iii) *Usage Patterns*—applications' workload change over time. We describe it analytically and implement utilities in the simulation environment to generate workload based on an analytical description.

The rest of the paper is organized as follows: In Section 2, we describe related works and compare the present one to them. In Section 3, we provide a succinct description of the targeted class of applications, define the scope and assumptions of this work and summarize our model. Section 4 formally defines the performance model. Section 5 describes the design of a simulator following this model. Section 6 describes several use cases of our model. Section 7 evidences the plausibility of our simulator through experiments. Section 8 concludes and defines avenues for future work.

2. RELATED WORK

Urgaonkar *et al.* [16] discuss a model for multi-tier web applications which represents how web requests are processed at the different tiers. This seminal work has been the basis for many other efforts in the area. Their model is represented as a linear network of queues—$Q_1, Q_2, \ldots, Q_m$. Each queue corresponds to an architectural tier. A request comes to a tier's queue Q_i and after being served it is transferred to the queue of the next level Q_{i+1} with probability p or to the queue of the previous tier Q_{i-1} with probability $1 - p$. To model web sessions a new queue Q_0 is introduced which is linked to both Q_1 and Q_m thus forming an infinite queuing system. For each active session a dedicated server is assigned in Q_0. Requests exiting from the last tier proceed to Q_0 and from there they re-enter the system. The time spent at the virtual server in Q_0 models the idle user time.

TABLE 1. Summary of related work.

Work	Analytical approach	Application type	Server concurrency	Level of granularity	Server type
Urgaonkar *et al.* [16]	Closed network of queues	Multi-tier	Single threaded	Request	Hardware
Zhang *et al.* [17]	Closed network of queues	Multi-tier	Single threaded	Transaction	Virtual machine
Bi *et al.* [18]	Hybrid queuing model	Multi-tier	Single threaded	Request	Virtual machine
Zhang and Fan [19]	Queuing model	Single tier	Single threaded	Request	Hardware
Kamra *et al.* [20]	Queuing model, control theory	Single tier	Single threaded	Request	Hardware
Chen *et al.* [21]	Closed multi-station queuing network	Multi-tier	Multi-threaded	Request	Virtual machine
Present work	Performance model based on step functions	Three-tier	Multi-threaded	Session	Virtual machine

In their model, Urgaonkar *et al.* do not consider the performance variability in a virtualized cloud environment. They consider the service time of the servers in the different tiers to be a random variable which is independent of the number of concurrently served sessions. In our model these are reflected. They represent sessions as a set of requests while in our model they are represented as atomic entities continuously creating performance load on the tiers' servers.

A similar analytical model based on a network of queues has been proposed by Zhang *et al.* [17]. Unlike Urgaonkar *et al.*, they use statistical regression to estimate the CPU cost of each server in the system. Also, the unit of work is transactions, not web requests. They do not consider the inherent performance variability in a cloud environment and the load on the servers in terms of used memory and disk I/O.

Several other approaches also use queuing theory for performance modelling. For example, Bi *et al.* [18] propose a hybrid queuing model for provisioning multi-tier applications in cloud data centres. They model the system at the level of incoming requests not sessions. Zhang and Fan [19] have developed a queuing model of a web system with one load balancer and two web servers. Their model does not represent a multi-tier system. Kamra *et al.* [20] model the entire multi-tier application as a single queue. That is, the different layers are not modelled separately.

In standard queuing theory, a server cannot serve more than one request at a time. This is an inherent problem when modelling multi-tier applications, since most application servers are multi-threaded and can serve multiple requests simultaneously. This issue has been identified by Chen *et al.* [21] and thus they model the whole system as a closed multi-station queuing network to capture this aspect. Every server in the application's architecture is represented as N servers in the queuing model, where N is the maximum number of concurrent sessions in a server. Also, in their work they assume a virtualized environment. A major difference between our work and theirs is that alike other queue-based models, they model the system behaviour at a lower level in terms of requests. Another difference is that they do not reflect the performance variability of the virtualized environment.

A major difference between all these works and ours is that our model is much more coarse-grained—we model the workload in terms of sessions and do not represent each and every request. This makes it much easier to define and experiment with different workloads. Also, since we do not represent requests, our model is more general and essentially agnostic of the underlying technologies. While other models are constrained to a specific type of technology, our model can be used for standard web applications, Rich Internet Applications (RIA), thick-client 3-tier applications, etc. Lastly, unlike others, our work takes into account the performance and workload heterogeneity of Cloud and Multi-Cloud environments. Table 1 further summarizes the related work.

In terms of simulation environments GridSim [12] and CloudSim [13] are the most relevant related projects. They are suitable for simulating batch processing workloads, and our present work extends CloudSim to support interactive 3-tier applications as well. Unlike our approach the GreenCloud [14] project focuses on simulating data centre energy consumption, rather than application performance. Similarly, MDCSim [22] focuses on evaluating the overall energy consumption and latency and does not facilitate application performance modelling. The SPECI [23] simulator is designed to evaluate the scalability and resilience of data centre infrastructure, and also cannot be used for application performance evaluation. The iCanCloud simulator [24] has similar capabilities and design philosophy to CloudSim, but is designed to be faster for large workloads and has a graphical user interface (GUI). It does not allow modelling 3-tier applications. In fact we argue that our approach can be implemented as an extension of iCanCloud, as we have done it for CloudSim.

3. OVERVIEW

3.1. Architectural setting

Dividing a system architecture into layers is the fundamental approach for managing complexity. In essence, a system is constructed as a sequence of layers/tiers, each of which is independent of the successive ones and communicates only with the

previous tier(s) through well-defined interfaces. This approach brings numerous benefits including increased understandability and maintainability since the layers can be developed, understood and investigated relatively independently [25].

As to Fowler [25] the main challenges when designing a multi-tier software system is to decide what are the layers and their encapsulated functionalities. A design error at this stage could result in having hard to maintain coarse grained layers which contain too much functionality or too many small layers that communicate excessively. This has precluded the development of layered architectural patterns. Of these, the most prominent is the 3-tier software architectural pattern (not to be confused with 3-tier data centre architecture), which allows for building scalable and adaptable information systems. Although not web specific, it has gained significant momentum with the advent of the web as it allows quickly migrating traditional applications to the web by replacing only the user interface layer [25]. This has resulted in the development of well known and established frameworks like Java EE [26], Ruby On Rails [27] and Django [28], which facilitate the development of 3-tier applications. A standard 3-tier application consists of three logical layers [25, 29, 30]:

(i) *Presentation Layer*—represents the interface, displayed to the user.
(ii) *Business/Domain Layer*—implements the core application logic.
(iii) *Data Layer*—handles access to the persistent storage.

This layering is logical, meaning that the layers could be deployed within a single or two machines. However, in the case of large-scale applications it also becomes physical separation and each logical layer is deployed separately in order to increase performance.

The presentation layer implements the user interface which the end user interacts with. It can be either a separate application (the so-called 'fat client') or web interface accessed through a web browser. Either way it is executed on the end user's machine and thus we do not take it into account in the servers' performance.

The business/domain layer implements the essential application logic. This includes all application specific business functionalities—e.g. the management of items in a shopping cart in an online store, persistence and access to users' personal data, etc. Traditionally, the domain layer is executed in one or several application servers (AS) which communicate with the presentation and the data layers. In the case of Infrastructure as a Service (IaaS) cloud offerings, AS servers are deployed in separate virtual machines (VMs).

The data layer facilitates access to the persistent storage. It is independent from the other two layers, as the persistent data often outlives the applications that use it and in some cases it can even serve more than one application. The data layer is executed in one or several DB servers.

Traditionally, the data layer has been the most architecturally challenging part of a 3-tier application, since it is hard to scale horizontally as demand rises. For example, this is the case when the data tier is implemented with traditional transactional relational DBs, which are widespread in virtually every application domain. Several approaches like master–slave replication, caching, NoSQL and NewSQL DBs [31] have been used to mitigate this issue and have been widely adopted in cloud environments as well.

This architectural pattern is very generic and there are a lot of possible adjustments that can improve the overall system performance. For example, the number of AS servers can be different and can increase or decrease dynamically in response to the demand. As discussed, the data layer can be composed of both transactional relational DBs, caches and novel approaches like NoSQL and NewSQL DBs. In this work, we introduce a general modelling and simulation approach that encompasses all of them, and can be used to evaluate different design alternatives—e.g. use of caching vs. master–slave replication in the data layer.

In this work we consider two main scenarios: (i) a 3-tier application is deployed within a single data centre and (ii) a 3-tier application is deployed in multiple Clouds (i.e. a Multi-Cloud) with additional redirection of users to appropriate data centres. Clouds A and B on Fig. 1 depict two possible deployments. For each application in a data centre, requests arrive through a single load balancer, which can be implemented in different ways. For example, it can be deployed in a VM or provided by the Cloud as a service. Load balancers distribute the incoming workload to a set of application server VMs deployed in the Cloud. Consequently, clients establish sessions with the assigned application servers. The AS communicate with the servers from the DB layer. Each DB server has one or several hard disks attached, which are used to store persistent data. In the multi-cloud scenario, client requests come to a global entry point, from where they are distributed to data centres, each of which has the aforementioned software stack deployed as depicted in Fig. 1.

3.2. Assumptions and scope

In this work we consider stateful (i.e. maintaining session data) 3-tier applications. Examples of session data are user preferences, shopping carts and history of user actions. Also, we assume that the DB servers are deployed in separate VMs. Alternatively, persistent storage could be provided by a cloud provider as a service. There is a plethora of vendor-specific Database-as-a-Service (DBaaS) solutions and relying on them can easily lead to vendor lock-in. Hence it is an industry best practice to use DB servers deployed in separate VMs [32]. Furthermore, many DBaaS solutions like Amazon RDS [33] actually provide VM instances to customers, and thus they can be represented by our model as well.

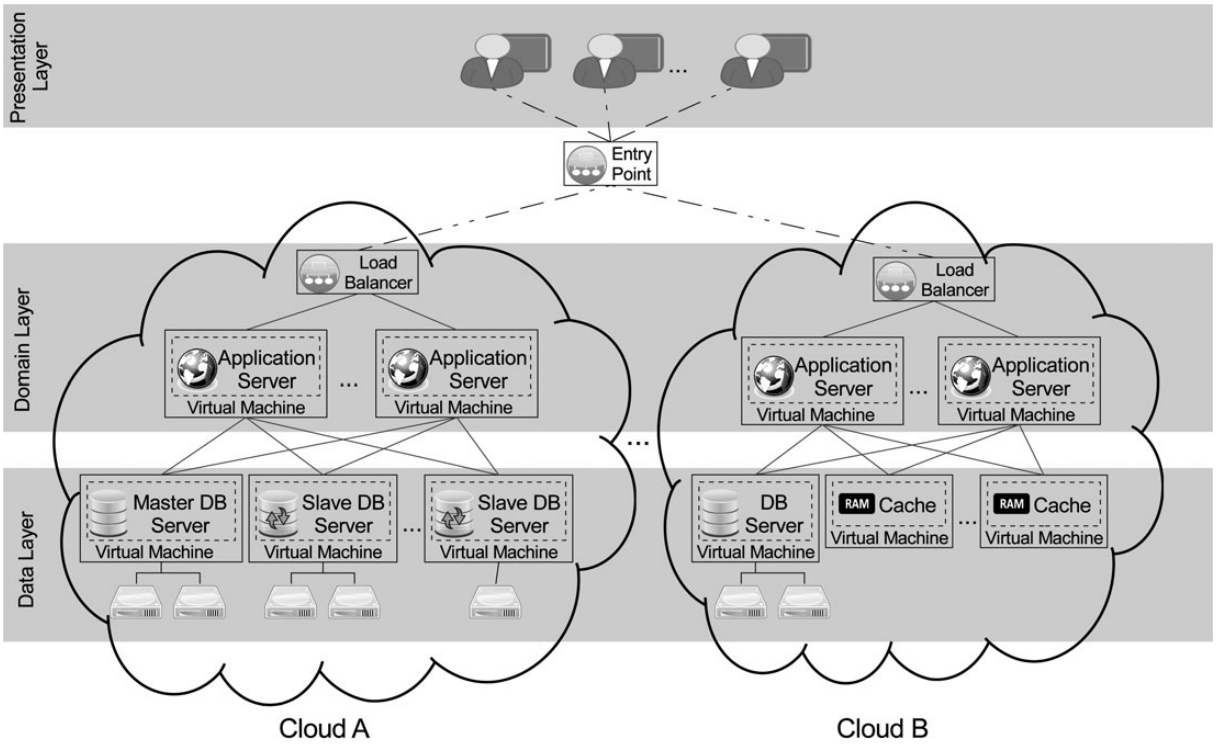

FIGURE 1. 3-tier reference architecture deployed in multiple Clouds.

Very few cloud providers offer their clients non-virtualized physical servers which are optimized for a specific purpose (e.g. running a DB). This can be considered an exception from the general practice of building cloud data centres from commodity hardware and providing it to clients in the from of virtualized resources [1, 2] and hence most cloud simulators do not represent application execution in non-virtualized environments. An example of such cloud service is the *Oracle Database Cloud*, which offers DB hosting on *Oracle Exadata Database Servers* [34, 35]. Such scenario can still be represented by our approach by modelling a host with the desired characteristics and allocating a single VM to it, which utilizes all its resources. Since in the model we do not consider virtualization overhead, this will be equal to running the workload on the host directly.

In this work we consider 3-tier applications deployed in cloud and multi-cloud environments. In the related literature there is a certain terminological ambiguity regarding the terms Inter-Cloud Federation and Multi-Cloud. In essence, an Inter-Cloud Federation is achieved when cloud providers voluntarily interconnect and share their infrastructure [11, 36–38]. This often raises interoperability issues, since resources like VMs need to be transparently migrated from one provider's infrastructure to another. Often this needs to be transparent to the cloud clients. In contrast, the concept of Multi-Cloud does not depend on an underlying collaboration of cloud providers. In a Multi-Cloud environment a cloud client uses and distributes workload among independent clouds [11, 36, 39]. In the current cloud landscape, where there is no inter-operation between major competing cloud vendors like Amazon and Google, the Multi-Cloud approach is much more feasible than the federation. In this configuration, cloud interoperability is not a problem, as cloud clients do not rely on underlying cloud provider collaboration. The main problem is working with the different management APIs of all cloud providers. Industry developments in the area of Multi-Cloud services like RigthScale [40], Enstratius [41], Scalr [42] and Multi-Cloud libraries like JClouds [43], Apache LibCloud [44] and Apache DeltaCloud [45] provide unified APIs for managing diverse cloud infrastructure and hence simplify system development [11]. Therefore in this work we focus our attention on workload distribution across clouds.

We consider that the load balancers implement 'Sticky load balancing' policies. That is, after a session between a client and an application server is established, all subsequent requests from this session are redirected to the same AS server. This is a reasonable assumption, since session sharing among servers often leads to overall performance degradation.

Clouds have made it possible for many clients to execute complex and time consuming analysis over huge amounts of

data (i.e. 'Big Data') [46]. Often there is a standard 3-tier web application serving as a front-end for accessing the data intensive analytical back end. Thus we can largely classify two types of data processing—synchronous (a.k.a online) and asynchronous (a.k.a offline). The synchronous data processing is concerned with performing data retrieval or manipulation in real time—e.g. extracting a list of items from online shop and passing them to the AS server for display to the user. In this case the user interactions are blocked until the operations are complete. Asynchronous data processing is concerned with running long batch processing jobs in the background. In this case user interactions are not blocked while the jobs are running. In this work we address synchronous data processing as it is in the standard 3-tier reference model.

We have outlined modelling data centre heterogeneity as one of our goals. More specifically in this work we focus one the following aspects of heterogeneity:

(i) Workload heterogeneity—for a Multi-Cloud application, sessions arrive with different frequencies based on the cloud geographical location, time zone, etc.
(ii) Cloud offerings heterogeneity—different cloud providers offer different VMs in terms of their CPU, memory and disk capacity.
(iii) VM consolidation policies—different cloud providers have different policies for placing VMs on physical hosts, which can affect CPU and disk performance.
(iv) Differences in the hardware capacity of the used physical hosts/nodes.

Finally, when multiple data centres are used we assume that the workload is distributed among data centres based on its geographical origin and do not model the work of the entry point. That is, clients directly submit their sessions to the nearby data centre.

3.3. Essence of the model

We propose a session-based analytical approach for performance modelling. The workload is represented in terms of user sessions established between users and application servers. Each session incurs performance load on the application and the DB servers. The workload of a given server at any point in time is defined as the sum of the workloads of the served sessions. Sessions make use of CPU time and RAM on the application servers and CPU, RAM and disk I/O on the DB servers. We assume that the disk operations on the application server are negligible—e.g. logging, loading configuration files, etc.

A key problem is how to represent the workloads (in terms of CPU, RAM and I/O) incurred by a session. Logically, they should be represented as continuous functions of time so that workload changes over time can be modelled. Since we are aiming to implement a discrete event simulator, we aim to represent these continuous functions in a discrete manner. One approach is to use stepwise functions. That way, by using a finite number of values one can define a continuous function of time.

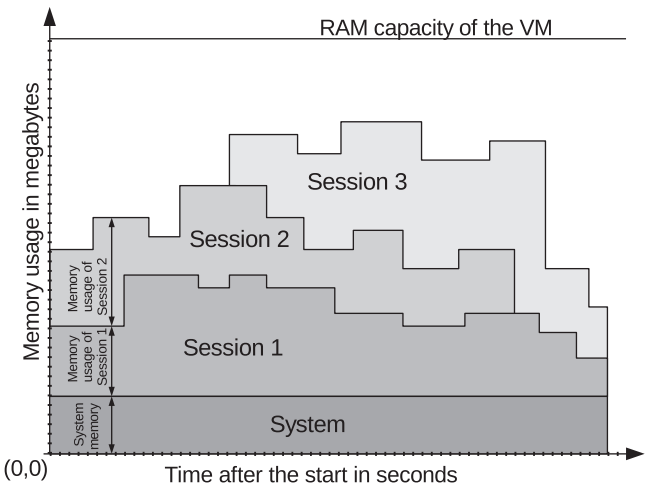

FIGURE 2. RAM load represented as stacked sessions' load.

Figure 2 depicts how a server's RAM usage can be composed of the RAM usage of three sessions and the underlying software stack—i.e. operating system and middleware. The system's RAM footprint can be defined as the used RAM, when no sessions are being served. The RAM usage of the sessions is represented by stepwise functions and the system's RAM usage is considered to be constant. Similar diagrams could be drawn for the CPU utilizations as well. The same approach can be used to represent the I/O utilization of a given hard disk on a DB server. That is, the utilization of a hard disk is defined by the I/O operations performed on it by the sessions using this disk on any of the DB servers. A minor difference is that the CPU and I/O capacity are not constant and change over time, because as discussed VMs can observer performance variability.

The frequency of the establishment of sessions is another aspect to model. Typically such phenomena are modelled with Poisson distribution representing that sessions arrive with some predefined frequency λ [47, 48]. Furthermore, Paxson and Floyd [49] have found that events like user arrivals/logins are well modelled with a Poisson process for fixed time intervals. However, in the case of a large-scale application it is not typical to observe constant arrival rates at all times. Moreover, in a Multi-Cloud environment the workload of the different data centres may be substantially different. For example, a data centre in the US will have higher arrival rates during the working hours in the US time zones and lower otherwise. Thus we model session establishment rates for a given data centre as a Poisson distribution of a frequency function of time $\lambda(t)$, which is also a stepwise function. If $\lambda(t)$ is a constant function, the arrival rates remain with pure Poisson distribution, and thus our approach is more general than the usual one. Our approach allows to model

demand fluctuations and spikes over time and to evaluate how the modelled system behaves in such cases.

If we have several different types of sessions in terms of the incurred performance, we can model the arrival rate of each of these as a separate Poisson distribution over a different frequency function.

4. ANALYTICAL MODEL

4.1. Session model

We take a session-based approach and instead of jobs or requests, the unit of workload in our model is a session, which represents a series of consecutive user actions incurring system load. Each session is assigned to an application server VM and can use the available DB servers in the data centre for the application.

Unlike jobs, a session generates variable workload over time. For example, for the first few minutes after its establishment a session may utilize heavily the processor of the application server, for the next five minutes it may incur very low resource consumption, and over the next five minutes it may incur significant disk I/O on a DB server. In contrast, a job is usually modelled in terms of the required resources (CPU, RAM, etc.) and there are no requirements as to how these resources are provided over time. More formally, a session is defined with the following variables and functions:

Ideal session duration
τ_i denotes the ideal duration of a session s_i measured in seconds. It is the session duration given that all resources for its execution are available. If there are more sessions than a tier can handle, there will be some resource pre-emption and eventually the session will be served in more than τ_i seconds or will fail. By Δ_i we denote the time by which s_i has exceeded τ_i and we consider it as a measure of application responsiveness.

Data item
In order to model the locality of disk operations, we need to represent the data resident on the disks. Thus, we introduce the notion of *data item*, which represents an entity stored on a disk. Examples of data items are files and portions of DB records, which are often accessed and stored together (e.g. a DB shard). In the model, we consider that a finite number of data items $d_1 \ldots d_n$ is specified.

Step size
We define a common step size δ for all stepwise functions in our model. Generally, we would aim for small value of δ, since that would make the simulation more refined.

CPU load of the application server
By $\nu_{as}(t)$ we denote the CPU load on the application server caused by the session. It is measured in millions instructions per second (MIPS) and represents the average number of CPU instructions required by the session over time. At a given point in time, the required number of operations can be defined as $n(t)/2\epsilon$ where $n(t)$ is the number of instructions required by a session in the time interval $[t-\epsilon, t+\epsilon]$ after its start given a small predefined $\epsilon > 0$. The ϵ value is an input parameter to the model (e.g. $\epsilon = 0.001$). Based on that, we can formally define the values of the stepwise function ν_{as} for the jth step (i.e. $t \in [(j-1)\delta, j\delta)$) as $\nu_{as}(t) = (1/\delta)\int_{(j-1)\delta}^{j\delta}(n(x)/2\epsilon)\,\mathrm{d}x$.

Memory load of the application server
The stepwise function $\phi_{as}(t)$ denotes the RAM usage in the application server by the session over time and is defined analogously to $\nu_{as}(t)$. $\phi_{as}(t)$ defines how many megabytes of memory the session uses t seconds after its start.

CPU load of the database servers
By $\nu_{db}(t, d_k)$, we denote the number of CPU operations needed for the processing of data item d_k by some of the DB servers. It is measured in MIPS and is defined analogously to $\nu_{as}(t)$.

Memory load of the database servers
$\phi_{db}(t, d_k)$ defines the RAM usage needed for the processing of data item d_k by a DB server and is formally defined analogously to $\phi_{as}(t)$.

Disk I/O load
By $\sigma_{db}(t, d_k)$ we denote the number of required disk I/O operations on data item d_k over time. It is measured in Millions of Input/Output Operations Per Second (MIOPS). The number of I/O operations with d_k at time t can be defined as $n(t, d_k)/2\epsilon$, where $n(t, d_k)$ is the number of instructions with data item d_k required by a session in the time interval $[t-\epsilon, t+\epsilon]$ and given the predefined $\epsilon > 0$. Analogously to ν_{as}, we can define the 'averaged' step values of the σ_{db} functions for the jth step (i.e. $t \in [(j-1)\delta, j\delta)$) as $\sigma_{db}(t, d_k) = (1/\delta)\int_{(j-1)\delta}^{j\delta}(n(x, d_k)/2\epsilon)\,\mathrm{d}x$.

Network delay and 'think times'
Within our architectural settings, we can largely classify two types of network: (i) the internal data centre network, used to connect application tiers within a cloud and (ii) the network between the end users and the serving data centre.

Nowadays the internal data centre network usually has sophisticated topology and offers high speed and low latency [50]. Moreover, previous research in multi-tier applications shows that inter-tier network performance typically does not cause significant performance penalty compared to CPU, RAM and I/O utilization [51, 52]. In fact Lloyd *et al.* have empirically shown that for a multi-tier application in a cloud environment, the outgoing and incoming VM network transfer between tiers explain, respectively, only 0.75% and 0.74% of the overall performance variability. In contrast, they have found that CPU load and disk reads and writes explain, respectively, over 71%,

37% and 14% of the observed performance [53]. Hence we will consider the effect of the internal network to be negligible and will not present it in our model.

In contrast, the delays caused by the wide area network connecting end users with the serving data centre can be significant. Similarly, users' 'think times' are typically present and need to be modelled. From the perspective of the servers both the external network delays and the 'think times' result in idle periods, during which the servers are not utilized by the session. More formally, if a session is idle (either because of a network delay or user's inactivity) during the time period $[t_1, t_2]$, then the number of required CPU instructions by the AS server will be $n(t) = 0$ for $t \in [t_1, t_2]$. Since $\nu_{as}(t)$ is dependent on $n(t)$ its value for this time interval will be affected accordingly. The values of ν_{db} and σ_{db}, representing the CPU and disk utilization of a DB server, for an idle time period can be defined analogously. The values of ϕ_{as} and ϕ_{db} for an idle period can be defined as the amount of session data kept in the memory by the respective AS and DB servers during these periods.

4.2. Modelling resource contention

Resource contention appears when some of a server's resources (e.g. CPU) are insufficient to serve all its sessions timely. Although clouds offer seemingly endless resource pools, it is still possible for an application to experience resource shortage. Firstly, some applications allocate resources statically and given a substantial workload their capacity can be exceeded. Secondly, public online applications can experience a sudden and unexpected demand peak. In some cases resources cannot be provisioned quickly enough and thus the already provisioned servers experience contention. Thirdly, in many 3-tier deployment models, the data tier cannot scale horizontally. For example, when a single relational DB server is used without any replication or sharding. In such cases the DB servers can experience contention. To explore what is the system performance in such cases, we need to be able to model and simulate contentions. Next, we describe how different resource contentions are handled in our model.

CPU resource contention
When the CPU resources of a server are insufficient, standard process/thread scheduling policies are used to assign processing time to each session. For the time periods that the session is preempted in a tier, it is also considered preempted on the other application tier. In other words resource preemption in one tier is reflected by an overall slowdown of the entire session, which is representative of synchronous data processing.

I/O resource contention
When the I/O throughput of the DB server is insufficient, standard I/O scheduling policies are used to assign I/O processing to each session. Alike with CPU, an I/O preemption in one tier leads to a preemtion in the other tier as well.

RAM resource contention
RAM resource contention occurs when at a given point in time the sum of the sessions' and the system's memory usages exceed the amount of memory the server has. When RAM resource contention appears the server stops, resulting in '*out of memory*' error. Consequently, all sessions served on this server fail and it stops accepting new ones. The work of the other servers however continues. For simplicity in our model we do not consider the usage of swapped disk space. The modelled behaviour is representative of an operating system going out of memory and killing the process of the server.

4.3. Session arrival model

As discussed, we model the arrival of a session of a given type in a data centre as a Poisson distribution of a frequency function—$Po(\lambda(t))$, where $\lambda(t)$ is represented as a stepwise function of time. In a Multi-Cloud scenario, the arrival rates in each Cloud should be defined. Hence, frequency functions need to be specified per data centre. This allows for modelling different workloads coming at different times within each data centre. Thus, we can represent workload influencing factors like time zone differences. Formally, for each session type st_i and data centre dc_j, the number of arrivals can be modelled as $Po(\lambda_{ij}(t))$, where $\lambda_{ij}(t)$ is a user specified function modelling the arrival rate.

4.4. Performance variability across clouds

The VM performance in a Multi-Cloud environment can vary significantly. The two main factors for this are:

(i) *VM setup*—a typical VM configuration allows a VM to use additional (but still limited) resources in terms of CPU and I/O if the host machine provides for that. Thus in the model it is important to specify what is the set-up of the VMs with respect to sharing hardware resources.
(ii) *VM consolidation*—if the VM setup allows for opportunistic usage of host resources, then it is important to note what the VM consolidation policy of each cloud provider is.

In order to adequately model the variability of the VM performance in a Multi-Cloud environment, we need to note both these characteristics for every data centre. In the implementation section we discuss further how these policies can be specified.

5. SIMULATOR IMPLEMENTATION

The previously described analytical model allows for coarse grained performance analysis of 3-tier applications. In

this section, we describe our implementation in terms of algorithms and data structures which extend the CloudSim [13] environment to support our model. CloudSim is a mature simulation environment and by extending it our approach can make use of all of its existing features like VM placement policies and energy consumption evaluation.

5.1. Representation of disk I/O operations

To represent an application's performance in terms of access to persistence storage, we extended CloudSim to support disk operations. This was done at three levels:

(i) *Host*—we extended the base CloudSim *Host* (physical machine) entity by adding hard disks to it. Each disk is represented as an extension of the *Processing Element* (Pe) component. A Pe in CloudSim models a core of a processor.
(ii) *VM*—in a similar fashion, we also extended the VMs to support disk operations. We also extended the schedulers that distribute VM resources among the executing jobs to distribute the available I/O operations.
(iii) *Cloudlet (Job)*—each job (*cloudlet* in terms of CloudSim) has been extended to define a number of required I/O operations. Also each cloudlet is associated with the data item it uses. Lastly, each cloudlet defines a boolean value, showing whether it modifies the data item or not.

Users are allowed to specify the policies for sharing I/O operations among VMs in a Host and cloudlets in a VM the same way they do it for CPU.

5.2. Provisioning of I/O operations

Disk I/O operations need to be distributed at two levels. Firstly, a *Host* should distribute the available disk operations among the *VM*s using the disks. Secondly, *VM*s should distribute their portion of the I/O operations among the jobs/cloudlets deployed in them. There is clear analogy between distributing I/O and CPU operations among *VM*s and cloudlets. Furthermore, as we represent hard disks as extended *Processing Element*s (Pe) we could directly reuse the already existing CloudSim policies for CPU scheduling.

However, unlike CPU, I/O operations are localized. I/O operations on a data item stored on one drive can not be executed on another, while CPU operations can be executed on every available CPU. Since CloudSim does not support such locality in the assignment of CPU operations, directly reusing these policies will lead to erroneous simulation behaviour. To overcome this problem we assign a separate scheduler to each of the hard disks of a *Host*. Thus disk operations can be provisioned per disk. This allows for VMs to have access only to a portion of the available disks and to achieve locality. We have also implemented locality in an extension to the job/cloudlet scheduling policies, so that jobs utilize only the disks, where the corresponding data items reside.

5.3. Representing sessions and contention

In the analytical model, we represented a session as several stepwise functions defining its resource demands on the servers. Within each step, a session has constant resource requirements in terms of CPU, RAM and I/O operations. To bring this concept to the simulation environment, we represent the behaviour of each session within a step as one cloudlet executed by the AS server and several cloudlets on the DB servers. Each of the DB cloudlets is associated with the data item it uses. The breakdown of a session's behaviour for a given step into DB cloudlets is based on the data items it accesses during this period. For example if a session accesses three data items during a given step, then there will be three DB cloudlets for this step.

In other words for each session and for each step in the simulation we have one cloudlet assigned to the AS server and several cloudlets assigned to the DB servers. Thus in terms of implementation the step values of ν_{as}, ϕ_{as}, ν_{db}, ϕ_{db}, σ_{db} define the corresponding cloudlets' resource requirements in terms of CPU, RAM and I/O.

The assignment of cloudlets to DB servers is based on the data items they use—cloudlets are assigned to servers which have access to their data. This assignment is done by a new simulation entity *DBCloudletScheduler*. The default implementation of this entity assigns each DB cloudlet to the first server which has access to its data item. By specifying a custom *DBCloudletScheduler* one can simulate different policies in the data layer. For example, cloudlets performing read operations could be redirected to slave DB servers, or can be cancelled if the data they retrieve is in a cache.

Formally, for a given session the required amount of CPU operations in the AS cloudlet corresponding to the step interval $[t_i, t_i + \delta]$ is $\int_{t_i}^{t_i+\delta} \nu_{as}(t)\,\mathrm{d}t$, which simply equals $\delta\nu_{as}(t_i)$ as $\nu_{as}(t)$ is a stepwise function has a constant value within this interval. The other resource requirements (CPU and RAM of a server and I/O on a disk) of the cloudlets are defined analogously.

Each session is represented with two equal-sized queues—one for the AS server and one for the DB servers. The elements of both queues correspond to the steps in the model. The elements of the AS queue are single cloudlets, which are executed sequentially. The elements of the DB queue are sets of cloudlets, whose members execute simultaneously. Cloudlets from subsequent elements of the DB queue execute sequentially. The cloudlets from the corresponding elements of both queues are associated with the same *start time*, and no cloudlet is submitted to the servers before its *start time* has passed.

This is depicted in Fig. 3. For each session s_i on the diagram the jth element of the AS queue is s_{ij}^{as} and the jth element of the DB queue is the set $\{s_{ij1}^{db} \cdots s_{ijl_{ij}}^{db}\}$, where l_{ij} is the set size.

Algorithm 1 Submit Session.

1: **procedure** SUBMITSESSION(s_i, t_{curr}, σ) ▷ s_i - session, t_{curr} - current rime, σ - step size
2: submit s_{i1}^{as} to the assigned AS server
3: submit $\{s_{i11}^{db}, \ldots s_{i1l_{i1}}^{db}\}$ to the *DBCloudletScheduler*
4: setStartTime(s_i, 2, $t_{curr} + \sigma$) ▷ Sets the start times of the cloudlets at position 2 in the queues
5: **end procedure**

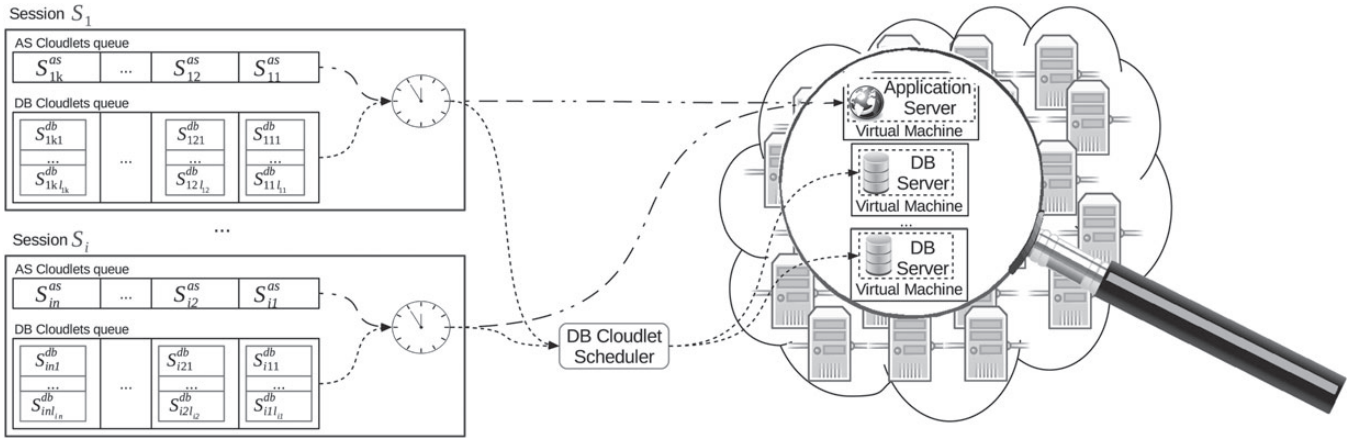

FIGURE 3. Session representation.

Cloudlets associated with the jth step are submitted only if all their *start times* have passed and all cloudlets from the previous steps have finished execution.

After a session s_i is submitted, the cloudlets form the heads of both queues are set the current simulation time t_{cur} as *start times*. The head of the AS queue s_{i1}^{as} is directly submitted to the assigned AS server. Cloudlets $s_{i11}^{db} \cdots s_{i1l_{i1}}^{db}$ from the head of the DB queue are sent to the *DBCloudletScheduler*, which submits them to the DB servers in accordance with the DB layer architecture—e.g. Master–Slave, caching, etc. The cloudlets from the next elements of the queues $s_{i2}^{as}, s_{i21}^{db} \ldots s_{i2l_{i2}}^{db}$ are set starting time $t_{curr} + \sigma$. This procedure is defined in Algorithm 1.

Upon the completion of every submitted cloudlet and the elapse of the *start time* of every cloudlet, which has not been yet submitted, the heads of the two queues are inspected. If all cloudlets from the two heads have finished and the *start times* of the cloudlets form the next queue elements have passed, then the following actions are taken:

(1) the heads of the queues are removed;
(2) the cloudlets from the new heads are submitted;
(3) the cloudlets from the next elements (after the heads) of the queues are set to have starting times σ seconds after the present moment.

Algorithm 2 defines formally the previously described procedure for updating a session. The two key invariants of this algorithm are:

(i) No cloudlet is submitted before its start time and the start time of its counterparts from both queues have come. Formally, no cloudlet $c \in \{s_{ij}^{as}, s_{ij1}^{db} \ldots s_{ijl_{ij}}^{db}\}$ is submitted before the *start times* for all $s_{ij}^{as}, s_{ij1}^{db} \ldots s_{ijl_{ij}}^{db}$ have elapsed. This ensures that even if one of the servers is quick to finish its cloudlets, this will not lead to premature exhaustion of the session's cloudlets and the session will still take at least its predefined ideal time.
(ii) No cloudlet is submitted before all its predecessors and all predecessors of its counterparts from both queues have finished. Thus if there is contention/bottleneck in any of the tiers this will result in an overall delay of the whole session.

As a consequence a bottleneck in a tier results in an overall delay of the served sessions.

CloudSim is a discrete event simulator and one can associate custom logic with every event. This allowed us to check throughout the simulation if the memory usage within a server exceeds its capacity. In accordance with the described model, in such a case the VM simulation entity is stopped and all of its running sessions are marked as failed.

5.4. Performance variability

CloudSim supports a special CPU provisioning policy called *VmSchedulerTimeSharedOverSubscription*. It allows setting the maximum CPU capacity of a VM. If there are sufficient resources this capacity is provided. Otherwise a smaller share is

Algorithm 2 Update Session Processing - executed when a cloudlet finishes, or the start time of a cloudlet elapses.

```
1: procedure UPDATESESSIONPROCESSING(s_i, t_curr, σ)        ▷ s_i - session, t_curr - current rime, σ - step size
2:     j ← index of the heads of the queues
3:     if t_curr ≥ getStartTime(s_i, j + 1) then
4:         submitNextStep ← TRUE
5:         for all c ∈ {s_ij^as, s_ij1^db ... s_ijl_ij^db} do
6:             if c is not finished then
7:                 submitNextStep ← FALSE
8:                 break
9:             end if
10:        end for

11:        if submitNextStep then
12:            pop s_i's queues
13:            j ← j + 1
14:            submit s_ij^as to the assigned AS server
15:            submit {s_ij1^db, ... s_ijl_ij^db} to the DBCloudletScheduler

16:            setStartTime(s_i, j + 1, t_curr + σ)
17:        end if
18:    end if
19: end procedure
```

allocated. This allows us to implement the previously discussed VM performance variability with respect to their mapping to hosts. Since we represent hard disks as extended processors, we reuse this policy to allocate I/O instructions among VMs.

When instantiating a physical machine in the simulation environment, one needs to specify two policies—one for sharing CPU time among VMs and one for sharing I/O operations. By choosing the appropriate policy one can either take into account the performance variability or assume that VMs use constant resources.

By using *VmSchedulerTimeSharedOverSubscription* and implementing different VM consolidation logic in each data centre, the performance heterogeneity of a Multi-Cloud environment can be represented.

5.5. Load balancing

To distribute the incoming sessions among AS servers, we introduce a new entity called *LoadBalancer*. It is responsible for assigning the incoming sessions to the AS servers within a data centre. Currently, the load balancer is a separate entity and is not deployed within a VM. We implemented a *default load balancer* that distributes the incoming sessions to the AS server with the least percentage of CPU utilization. However, one can implement different load balancing policies and run simulations with them.

5.6. Workload generation

In our implementation, workload is generated by a new entity called *WorkloadGenerator*. It generates workload for a single data centre and application. Thus each generator is associated with a single load balancer. At the start of every simulation step, each generator is queried and returns a list of sessions that will be submitted to the load balancer during the step.

In order to define the behaviour of a generator, two additional parameters are needed—a *FrequencyFunction* and a *SessionGenerator*. The *FrequencyFunction* represents the $\lambda(t)$ function from the formal model. In our implementation it has an additional property—the *unit*, which represents the time period the frequency is defined for. For example, if the *unit* is 60 s and the frequency is $\lambda(t) = 3$ for every t, then sessions appear with frequency 3 per minute during the entire simulation. One can implement arbitrary frequency functions that meet the requirements of their simulation.

The default implementation is *PeriodicStochasticFrequencyFunction*, which defines a periodic frequency function. For example, it can be used to define the frequencies on a daily or weekly basis. Using it, one can further define frequency values as samples of a normal distribution. For example, one can specify that on a daily basis in the period 13–15 h the session occurrence frequency per hour has a mean of 20 and a standard deviation of 2. At simulation time, the workload generator generates values that meet these criteria. One can also specify the so-called *null point* of such a function. It represents the moment at which a period starts. *Null points* represent an easy way to define workloads originating in different time zones. That is, if we have a given workload generator, we can 'move' it through time zones by just setting the appropriate *null value*.

The *SessionGenerator* is responsible for creating sessions. The default implementation is *ConstSessionGenerator* which

generates equal sessions every time it is called. Alike frequency functions one can implement and plug their own session generation policies.

The work of a workload generator is depicted in Fig. 4. At the beginning of every step of the simulation, the *WorkloadGenerator* is requested to generate the sessions for the step. Based on the step length, the frequency and the function's unit the generator computes how many sessions need to be submitted for the step and asks the *SessionGenerator* to create them. The resulting sessions' submission times are uniformly distributed throughout the step's duration.

6. USE CASES

This section describes several typical use cases of the proposed modelling and simulation environment. We envision that the proposed environment can be used by both researchers in the field of distributed systems and system architects for performance evaluation and comparison of different provisioning and scheduling approaches. The common use cases are depicted in Fig. 5.

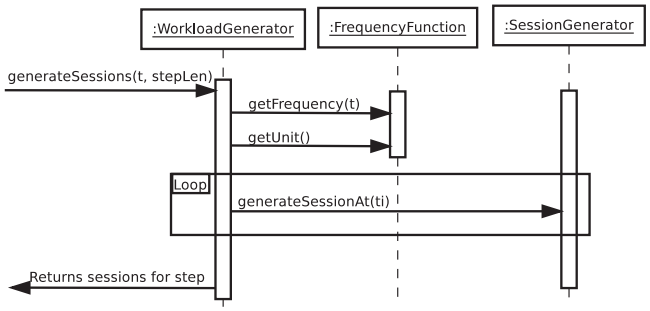

FIGURE 4. Workload generation.

During the early stages of designing a system, an architect often needs to predict how the designed system would behave under different workloads and if it is going to meet the stated non-functional requirements at all times. Key non-functional requirements usually are the response time to users and the cost of the utilized Cloud resources. Having multiple configurable Commercial Off-The-Shelf (COTS) and Open Source software components and middleware at their disposal, they need to compare the available approaches, to weight the architectural trade-offs and make an architecture wise decision. This is usually achieved by developing small-scale proof of concept (POC) prototypes that demonstrate the overall plausibility of an approach. The cost and efforts for developing multiple POCs to explore various architectural alternatives have been a major inhibitor for system architects. Not to mention that small scale POCs can not be used to test for workload peaks and scaling under extreme workload.

Hence, a simulation environment can be used during these early stages of system development. An architect can perform simulations by 'plugging' different policies for load balancing and data replication and caching by implementing the *LoadBalancer* and *DBCloudletScheduler* APIs. This would allow for evaluation and comparison of different approaches.

Similarly, the proposed modelling and simulation environment could also be of use to researchers, developing new provisioning and scheduling approaches for 3-tier applications in Clouds and Multi-Clouds. New load balancing, data replication and caching strategies could be developed and evaluated through the *LoadBalancer* and *DBCloudletScheduler* APIs. New entry point policies can also be evaluated.

A key point in all these use cases is the definition of the workload directed to every data centre. Firstly, one needs to define the resource consumption of the sessions that make up the workload. One approach is to define one or several types of sessions, whose performance characteristics over

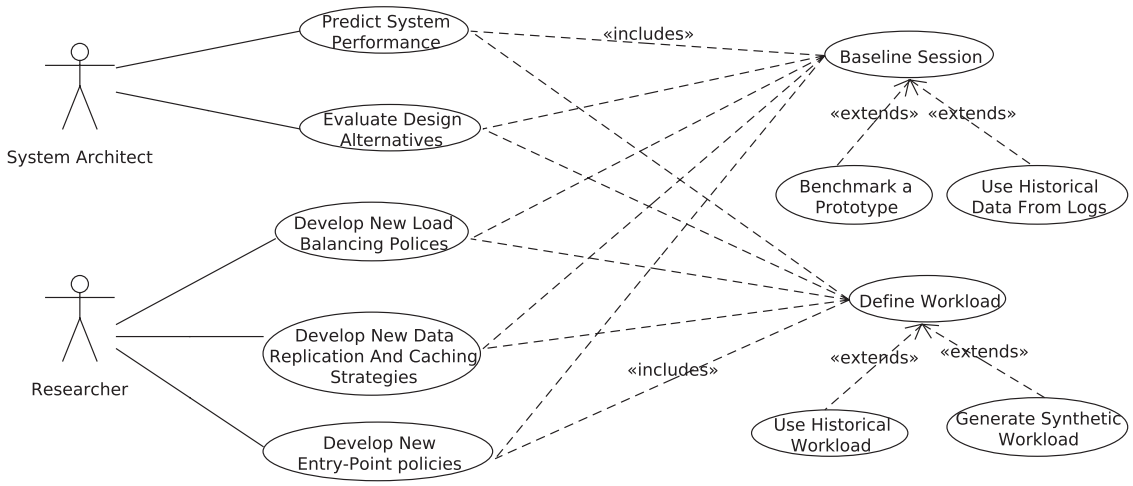

FIGURE 5. Main use case scenarios.

time (expressed in the step-wise functions) are averaged from monitoring data from actual executions. For this baseline benchmarks of application executions are needed. They can be extracted from historical data from log files of a prototype of the system or a similar system. Alternatively, they can be extracted by performing experiments with a prototype of the system or one with similar characteristics.

Once the characteristics of the session type(s) have been defined, the frequencies of the session establishments need to be defined. One approach is to use historical workload—e.g. the times of session establishment can be taken from logs. However, it is often the case that no historical information is available—e.g. if a new system is being architected. Moreover, one of the main benefits of performance modelling and simulation is that one can test with a range of workloads including both high and low ones and with static or dynamic arrival frequencies over time. The aforementioned workload generation functionality of the simulation environment can be used to generate synthetic workload. As discussed, it allows for the generation of arbitrary time-dependent workloads.

7. VALIDATION

In this section, we model and simulate the behaviour of an actual 3-tier system and compare the performances of the simulation and the system itself. The goals are two-fold: (i) to demonstrate how one can model a system's behaviour and (ii) to show that our model can predict a system's performance with reasonable accuracy.

A simulator is a generic tool with a wide applicability scope. Testing and validating our simulator against all possible middleware technologies, operating systems, configurations, virtualization environments and workloads is an extremely laborious and time consuming task. In this section, we demonstrate the plausibility of our approach with two experiments with the established benchmarking environment Rice University Bidding System (RUBiS) [54–57]. It consists of an e-commerce website similar to eBay.com and a client application, which generates requests to the website.

RUBiS follows the standard 3-tier architecture and has an application and a DB server. We use the PHP version of RUBiS with a MySql relational DB. The user interface consists of several web pages. They are not static and have dynamic content [57]. For each incoming HTTP request for a page, the PHP server parses the HTTP parameters, queries or updates the MySql DB and generates the dynamic content of the page, which is then returned to the end user. Hence each page request induces load to both servers. The MySql server ensures that the data are accessed in a transactional and consistent manner, which is important for an e-commerce application. The DB consists of seven main tables, containing information about items, categories, bids, comments, users, regions and direct purchases [54]. Figure 6 shows the relational model in more details. There is an additional table called 'old-items' with the same schema as 'items', which contains historical data about goods, which are no longer for sale. In the RUBiS workload, the DB has been populated with data in a manner similar to *ebay*. More specifically, at every moment there are about 33 000 items classified in 20 categories and 1 million users in 62 regions. There are half a million comments. The total size of the MySql DB and the accompanying indices is approximately 1.4 GB [54].

In RUBiS the workload consists of sessions, each of which comprises a sequence of requests. The succession of requests is based on a transactions table/matrix specifying the probabilities for a user visiting a given page to navigate to every other page or to leave the website. RUBiS also emulates the 'think times' between requests. By default the lengths of the 'think times' follow a negative exponential distribution as advised by the TPC-W specification [58]. The mean of this distribution and the number of sessions are parameters of the workload. At run time, for each session the RUBiS client starts by sending an initial request to the 'Home' page. Then based on the mean 'think time', the last accessed page and the transactions table it randomly generates a 'think time' period, waits for it to expire, and generates the next request. We have modified the client, so that this process continues until a predefined number of requests have been served or the user leaves the system.

RUBiS comes with two types of workload, defined in the provided transactions tables. The first one consists of browsing actions only, that do not result in a modification of the persistent data. Examples of such actions are browsing items and reviewing sellers' information and ranking. The second workload consists of 85% browsing actions and 15% actions resulting in a modification of the DB. It is a much more realistic auction site's workload [55, 57]. This is the default RUBiS workload and it is the one we use in all our experiments.

In RUBiS performance utilization data are gathered from both servers through the SAR (System Activity Report) system monitor, provided by the SYSSTAT package [59]. At execution time, it collects detailed information regarding CPU, RAM, I/O and network utilization. Among these, the measurement of the actual RAM consumption is an elusive goal, because (i) operating systems often keep huge caches of persistent data in memory to avoid some disk operations, (ii) in many middleware technologies memory is not automatically released until garbage collection. Consequently, deciding what part of the allocated memory is actually used by the application is not trivial. To bypass this issue in our experiments we use the SAR metric for amount of *'active memory'*. As to the specification it represents the amount of memory which has been used recently and is not likely to be claimed for other purposes.

In all following experiments we set-up the RUBiS client to execute sessions consisting of maximum 150 requests, and having 7 s average 'think time'. We use the default RUBiS transitions table to emulate users browsing through the website. In all simulations, we consider a step time $\delta = 60$ s.

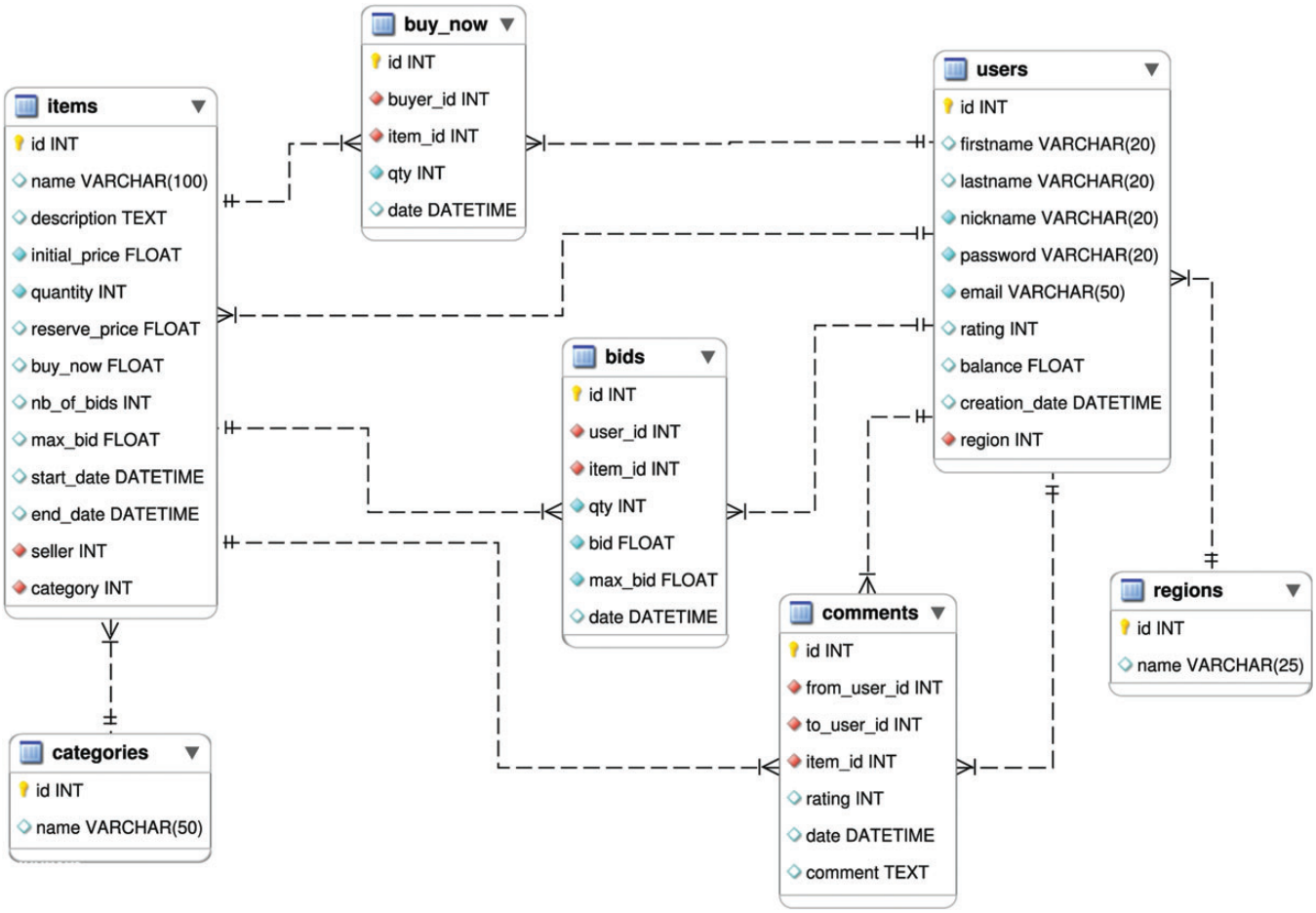

FIGURE 6. Relational model of the RUBiS workload.

7.1. Session performance baseline

As discussed, one can extract a typical session's performance characteristics either from historical data or by benchmarking. Due to the unavailability of suitable performance logs for RUBiS we use benchmarking, which is described in this section.

Firstly, we assume that all sessions have similar performance characteristics. For a RUBiS workload's sessions this is reasonable, since they all follow the same statistical patterns. This is also typical in real life, where one can often find that 'on average' users follow similar usage patterns. In our simulation we will have one session type/prototype defining a session's performance demands over time in terms of the ν_{as}, ϕ_{as}, ν_{db}, ϕ_{db} and σ_{db} functions. The values of these functions are derived from benchmarks of the system's performance. Hence when we emulate n sessions with the aforementioned statistical properties in RUBiS, in the corresponding simulation we will simulate n sessions from this type. Intuitively, as the session type is representative of a typical session the simulation and the actual execution should have similar performance characteristics given a sufficient number of sessions. We test this assumption in the later subsections.

For some applications one can observe that different groups of users (e.g. merchants and buyers in an online store) have diverse usage patterns, resulting in different resource utilization. In such case, different session types should be defined and simulated. However, in the RUBiS environment all users follow the same usage pattern, and thus we consider only one type.

We conduct the benchmarking on a computer with an Intel(R) i7-2600 chipset, which has four hyper-threading 3.4 GHz cores, 8 GB RAM and is running 64bit Ubuntu 12.10 operating system. We have deployed the AS and DB servers in separate VirtualBox VMs, each of which has 512 MB RAM, a dedicated CPU core from the host and is running 32bit Ubuntu 12.4.

The RUBiS benchmark uses a single DB server and does not use any caching, replication or sharding. Thus in the formal model we consider a single data item d_1, which is used by all disk operations. Thus in our model the ν_{db}, ϕ_{db} and σ_{db} functions de facto become functions of time only.

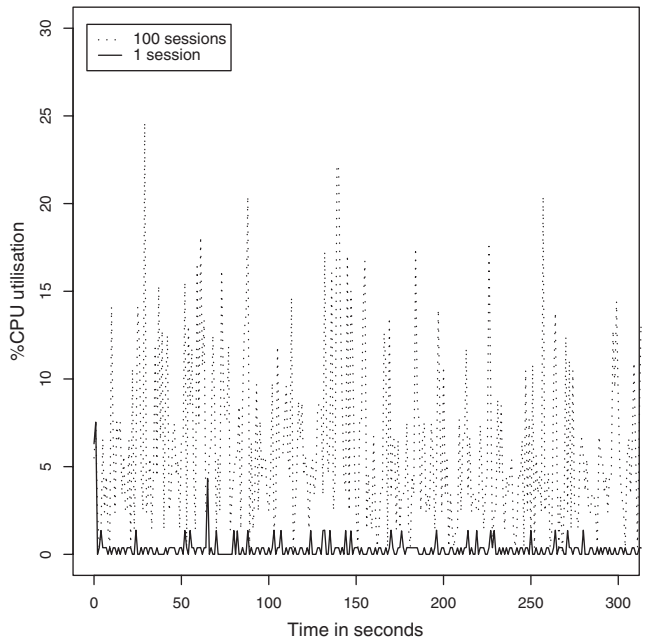

FIGURE 7. CPU utilization of the DB server with 1 and 100 simultaneous sessions for the initial 5 minutes.

We perform two simple RUBiS benchmark tests to define the session type for the simulation. For both tests we get the values of the resource utilization in terms of CPU, RAM and disk I/O from the SAR monitoring service. We take measurements for all servers and for every second from the tests. First, we test the system with only one session. The goal is to measure the utilization caused by the operating system, the middleware and the monitoring service. Secondly, we test with 100 simultaneously started sessions. We attribute the differences in system utilization between the two tests to the increment of 99 sessions in the workload and based on that we estimate the overhead of a single session.

In the rest of the section, we describe how the AS server CPU utilization caused by a session can be represented in our model. The procedure for defining the model representation based on the benchmarks is not CPU specific and only uses the reported by SAR utilizations in percentages. Thus, without loss of generality, the same approach can be applied to model the incurred RAM utilizations of both servers and the CPU and disk utilizations of the DB server.

Based on the two tests, we can define a session's AS CPU utilization $t_i \in N$ seconds after its start as:

$$F^{as}_{\text{cpu}}(t_i) = \frac{F^{as(100)}_{\text{cpu}}(t_i) - \overline{F^{as(1)}_{\text{cpu}}}}{99}, \qquad (1)$$

where $F^{as(1)}_{\text{cpu}}(t_i)$ and $F^{as(100)}_{\text{cpu}}(t_i)$ are the measurements of the CPU utilization of the AS server for the experiments with 1 and 100 sessions, respectively, and $\overline{F^{as(1)}_{\text{cpu}}}$ denotes the mean of all CPU measurements for the AS server from the test with 1 session. The rationale for Eq. (1) is that $\overline{F^{as(1)}_{\text{cpu}}}$ denotes the typical overhead utilization caused by the system components like middleware and OS. Thus when we subtract it from the overall utilization and divide by the increment of sessions we get an estimation of the overhead caused by a single session. We call the values $F^{as}_{\text{cpu}}(t_i)$ *derived observations* of the session's CPU utilization on the AS server. Analogously, we can define the session's *derived observations* on both servers in terms of CPU, I/O and RAM.

Next we need to define the step values of the model step functions based on the derived observations in terms of CPU, RAM and disk I/O. One approach is to use the means of the derived observations falling within a step. However, we observe that the monitoring values returned by SAR for the two experiments include many fluctuations/spikes—see Fig. 7. Thus, to overcome the effects of outliers, we use the medians of the derived observations falling within a step. Recalling that we consider steps of size 60 s, and that we have derived observations for every second, the CPU utilization of the AS server for the jth step (i.e. $t \in [t_{60(j-1)}, t_{60j}]$), can be defined as follows:

$$v_{as}(t) = \text{median}(F^{as}_{\text{cpu}}(t_{60(j-1)}) \cdots F^{as}_{\text{cpu}}(t_{60j-1})). \qquad (2)$$

As discussed, we define the step values of ϕ_{as}, v_{db}, ϕ_{db} and σ_{db} analogously, which gives us a complete session performance model.

Lastly, we need to define the ideal session duration τ of the sessions from the session type. In the experiment with 100 sessions, the utilization of all performance characteristics (CPU, RAM and disk I/O) is much less than 100% at all times. Hence, there has not been any resource contention during this test and thus we can define τ as the mean of the execution times of all sessions from this experiment.

7.2. Experiment 1: static workload

The goal of this experiment is to demonstrate that a simulation can be used to predict the resource utilization of the AS and DB servers and the delays in the sessions' execution. We execute several RUBiS benchmarks, and model and simulate each of them using our approach. We use different workloads in terms of their intensity to test our simulation in the presence of both low and high resource consumption and CPU and disk I/O contention. Then we compare the predicted by the simulations performance with the ones from the actual execution.

For this experiment, we use the same environment as for the baseline measurement we did previously. We execute workloads consisting of 50, 200, 300, 400, 500 and 600 simultaneously starting sessions and we record the delays and the utilization over time. The sessions are based on the default RUBiS transitions table, which defines regular transactional load in an e-commerce website and which we used for the baselining.

From the monitoring data we can conclude that the used workloads are mostly transactional and data intensive, since

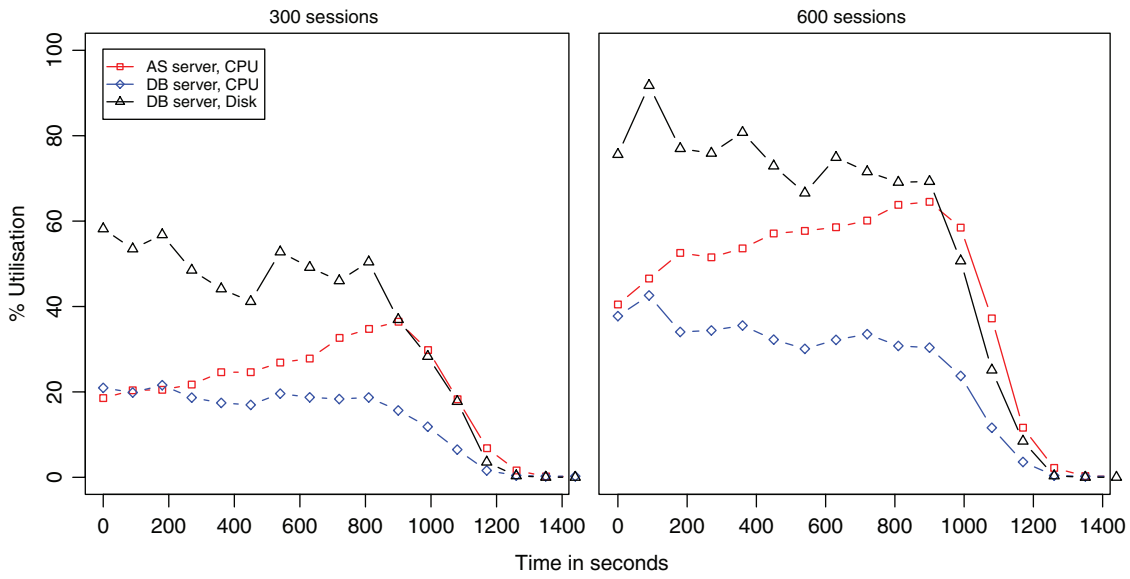

FIGURE 8. CPU and disk utilizations with 300 and 600 simultaneous sessions. The plotted values are averaged for every 90 s.

the DB disk utilization dominates the CPU utilizations of the two servers for the majority of the execution time. This is shown in Fig. 8 which depicts the CPU and disk utilizations of the two servers at execution time given workloads of 300 and 600 sessions. We have plotted the averaged utilizations for every 90 s. For space considerations we have omitted the other four graphs comparing utilizations. An interesting observation is that by the end of the execution, the disk utilization decreases significantly. One reason for this is the suspension of some of the simultaneously started sessions, whose lengths are generated by RUBiS as values of a random variable. In fact, this is why all resource utilizations decrease in the latest stages of the experiment. Another reason is the in-memory caching of disk operations by the operating systems and the DB server, whose effect gradually increases during the experiment. Another interesting observation is that, the CPU utilization of the AS server increases, as the disk utilization decreases. This is because the throughput of the data tier is increased, which allows the AS server to serve more user requests timely.

In our parallel simulation environment we define a single host, with two VMs with parameters modelled after the real ones. Then we execute simulations with the same number of sessions as in the aforementioned RUBiS experiments and we record the resource utilization of the two VMs. The step functions, defining a sessions' behaviour are the ones extracted from the previous section.

First, we compare the accumulated session delays from the RUBiS execution (Δ_i) with the ones from the simulation. Since in the simulation we have identical sessions, which start simultaneously and whose performance utilization does not depend on a random variable, we get a single numeric value as a predictor of the delay. However, in the RUBiS benchmark

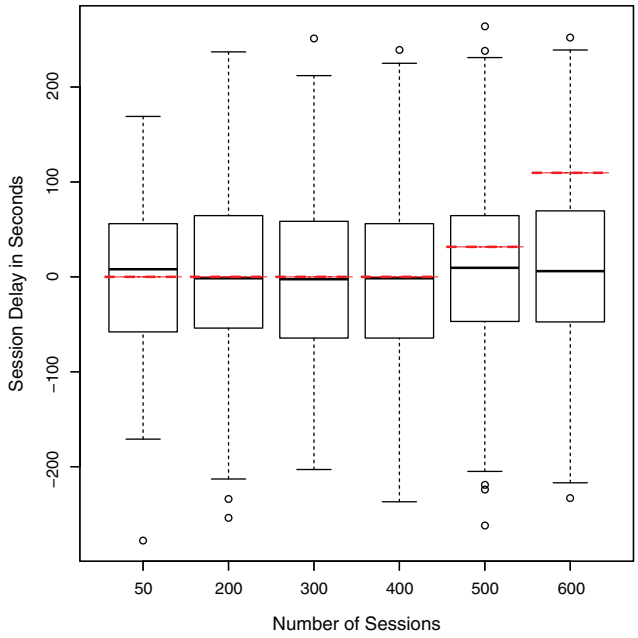

FIGURE 9. Execution delays (boxplots) and predicted delays (crossing horizontal lines) in Experiment 1.

the delays vary, as the 'think times' and the succession of requests vary based on random variables. Figure 9 depicts the estimated by the simulations execution delays and the actual ones and shows that the predicted delay is a good estimator of the actual delays, always falling in the interquartile range, except for the case with 600 sessions, when the 3rd quartile is exceeded with about 40 s. We believe this difference is caused

by performance improvements like in-memory caching, which are not represented in our model, and which can mitigate the consequences of contention in case of high workload. Since the most utilized resource is the disk of the DB server (see Fig. 8) this has impact on the overall delay. It is worthwhile noting that some of the execution delays are actually negative, which means that some sessions finish before the ideal execution time has passed. This is because we defined the ideal execution time τ as the mean of the execution times of a multitude of sessions that do not experience any contention. Thus by pure chance in the actual execution we have sessions, whose own ideal duration will be shorter.

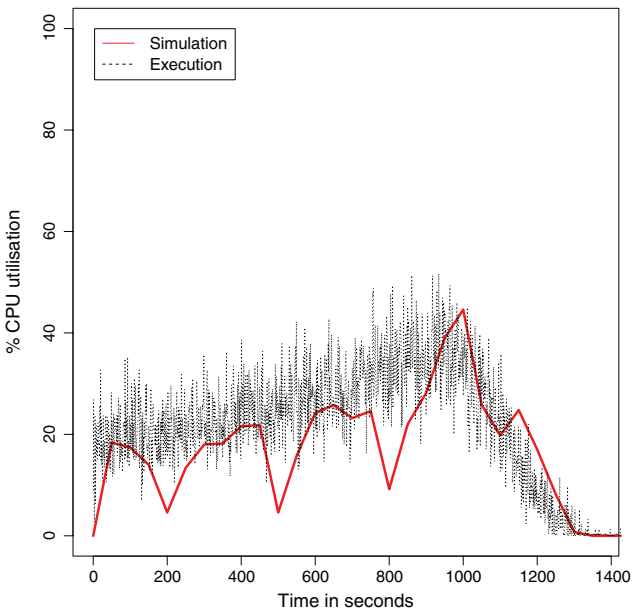

FIGURE 10. Predicted and actual CPU utilization of the AS server with 300 simultaneous sessions in Experiment 1.

Next we compare the utilizations over time in terms of CPU, RAM and I/O of the servers in the execution and simulation experiments. Figure 10 shows how the predicted CPU utilization of the AS server approximates the real one for a workload of 300 sessions. To make the diagram clearer we have plotted the averages of each 50 observations from the simulation, not of each single observation. Figure 11 depicts how the predicted by the simulation disk I/O utilizations approximate the one observed at execution time for workloads of 50, 300 and 600 sessions. Again, we have shown the averages of each 50 observations from the simulation. The figure shows that with the increase of the workload, the actual disk I/O utilization increases as well, and causes contention in the benchmark with 600 sessions. This behaviour is well approximated by the predicted by the simulation disk I/O utilization. For space considerations we do not include the other 26 comparison graphs.

For every workload, server and resource (i.e. CPU, RAM and I/O), we have two matched time sequences of observations for every second of the execution—one from the RUBiS benchmark and one from the simulation. Since in the simulation we can do many monitoring operations without hurting the performance, for better precision we perform 10 observations for each second and then we average them. In other words we have paired samples of the utilization from the executions and the simulations. All utilization measurements are in terms of percentage—e.g. 90% disk utilization. Based on the procedure of the Wilcoxon signed rank test with continuity correction we can compute the 95% confidence intervals (CI) and an estimate (the pseudomedian) for the median of the difference between the populations.

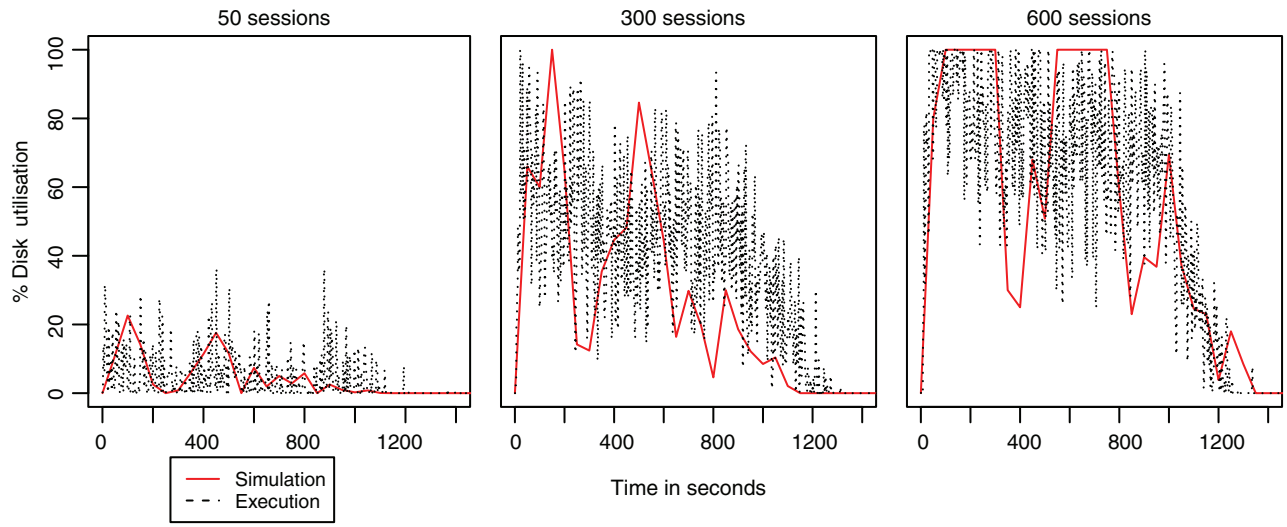

FIGURE 11. Predicted and actual disk I/O utilization of the DB server with 50, 300 and 600 simultaneous sessions in Experiment 1.

TABLE 2. 95% Confidence Intervals (CIs) and estimates of the median of the utilization differences between simulation and execution. Measurements are in percentages.

#	AS server, CPU		AS server, RAM		DB server, CPU		DB server, RAM		DB server, I/O	
Sessions	95% CI	Estimate	95% CI	Estimate	95% CI	Estimate	95% CI	Estimate	95% CI	Estimate
50	(−5.43, −4.93)	−5.22	(−1.61, −1.58)	−1.59	(−2.90, −2.39)	−2.84	(−1.93, −1.91)	−1.92	(−8.48, −4.64)	−6.89
200	(−13.41, −12.51)	−12.97	(−3.27, −3.22)	−3.25	(−7.67, −6.15)	−7.05	(−1.70, −1.69)	−1.7	(−19.81, −15.05)	−17.45
300	(−15.32, −12.82)	−14.05	(−0.67, −0.56)	−0.61	(−2.31, 14.14)	−0.32	(−0.11, −0.09)	−0.11	(−17.16, −11.01)	−13.96
400	(−18.29, −14.04)	−16.52	(−1.05, −0.91)	−0.99	(22.92, 26.66)	24.97	(2.62, 2.70)	2.68	(−13.99, −6.92)	−10.45
500	(−17.16, −5.29)	−9.01	(−2.47, −2.11)	−2.29	(24.21, 27.43)	25.86	(4.36, 4.38)	4.37	(−16.90, −9.65)	−13.25
600	(−11.09, −7.06)	−8.98	(−0.91, −0.25)	−0.63	(27.46, 31.13)	29.34	(5.29, 5.38)	5.33	(−17.32, −9.84)	−13.59

Table 2 lists the 95% CIs and the estimates for all experiments and utilization characteristics. For all experiments and characteristics, except for the CPU utilization of the DB server, the 95% CIs for median of the difference in utilization contain values within 20% range of the ideal case of 0%. The only exception is the CPU of the DB server, the estimate of the deviation for which reaches nearly 30% for the case of 600 sessions. Once again this difference can be explained by performance optimizations like caching, that we do not account for. Such optimizations can mitigate contention and improve performance given a significant workload, which is reflected by the increase of inaccuracies when the number of sessions is increased.

7.3. Experiment 2: dynamic workload

The goal of this experiment is to show that simulation can be used to predict the resource utilizations given a dynamic workload in a heterogeneous Multi-Cloud environment. For this purpose, we set up two installations of RUBiS. The first one is the environment we used in the previous experiment and for the benchmarking. We call this environment Data Centre 1 (DC1). The second one is in the Asia Pacific (Singapore) region of Amazon EC2. It consists of two *m1.small* VM instance—one for the AS and one for the DB server. We call this environment Data Centre 2 (DC2). The two environments have different underlying hardware, virtualization technologies and provide VMs with different capacities. Hence, with this experiment we demonstrate how we can model and simulate application execution in heterogeneous cloud environments.

We modify the RUBiS client, so that it can dynamically generate sessions over a 24 h period, based on predefined frequencies for each hour. For example, we can specify that between the second and third hours of the test the number of arriving sessions is normally distributed with mean μ and standard deviation σ. Then at execution time, at the beginning of the second hour of the test, the RUBiS client generates a random number n, which is an instance of this distribution. Then n sessions are submitted at equal intervals between the second and the third hour.

TABLE 3. Hourly frequencies of the workload for Experiment 2.

Time period (h)	Mean hourly frequency	Standard deviation
0–6	10	1
6–7	30	2
7–10	50	3
10–14	100	4
14–17	50	3
17–18	30	2
18–24	10	1

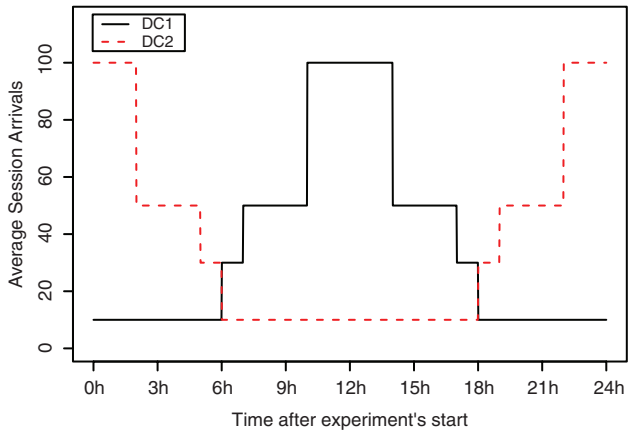

FIGURE 12. Average hourly frequencies of session arrivals for the two data centres in Experiment 2.

Table 3 lists the frequencies we use in this experiment. They are representative of the daily workload of a small website. We use them to generate workload for DC1. To demonstrate the effects of heterogeneous time zone dependent workloads, we assume that DC2 is in a location with 12 h time zone difference and has the same workload pattern. Thus for DC2 we 'shift' the frequencies with 12 hours. Figure 12 depicts the mean session arrivals for the two data centres.

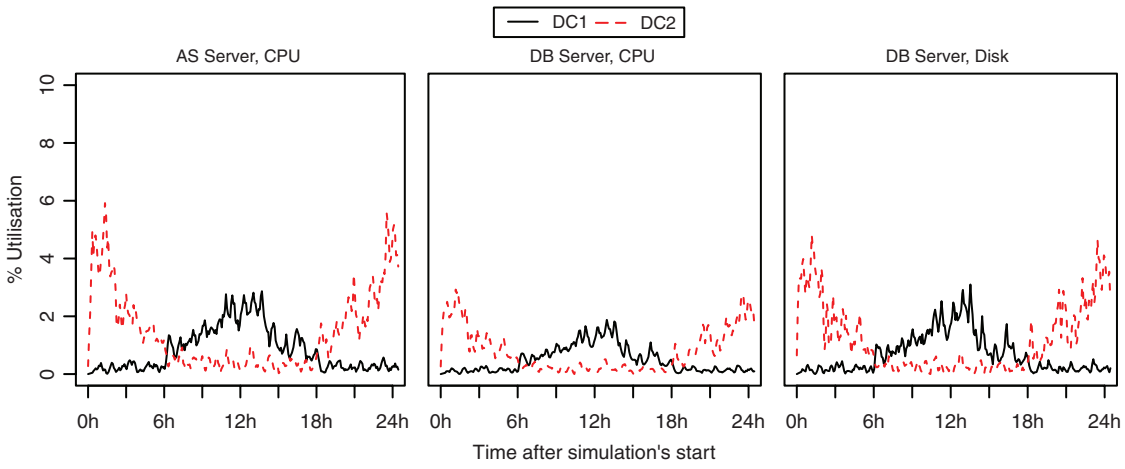

FIGURE 13. CPU and disk utilizations of the AS and DB servers in DC1 and DC2 at simulation time in Experiment 2.

In the simulation environment we use the data centre we used in the previous experiment to represent DC1. In our simulation DC2 has the same characteristics as DC1, with the only difference being the RAM, CPU and I/O capacity of the VMs and the physical machines (hosts).

However, m1.small instances in Amazon EC2 do not have a dedicated processing core, like the ones in our local environment (DC1). Hence, some of the CPU time may be distributed to other VMs located in the same host. As discussed, the dynamism of VM placement can result in significant performance variability. Amazon defines the metric EC2 Compute Unit, to measure the amount of CPU capacity a m1.small VM has. However, often this is just a lower bound, and one can observe better performance if for example the VM does not share the host with others. Typically, one should evaluate their scheduling and provisioning policies against the lower bound capacities of the provider's SLA. However, in this comparative study we aim to statistically compare simulation and execution and thus we need as accurate VM performance data as possible.

To achieve this, firstly we examine the capacity of the host CPU, as defined in the */proc/cpuinfo* Linux kernel file of the two VMs. The SAR monitor defines the *%steal* metric, which describes the percentage of 'stolen' CPU operations by a hypervisor—e.g. for serving other VMs. This metric is always zero for VMs with dedicated cores. At execution time we account for the *%steal* value when we define the CPU utilization. In the simulation environment the CPU capacity of the VMs in DC2 is defined as the mean CPU host capacity, which has not been 'stolen' during execution. Similarly, Amazon does not define (not even with a lower bound) how many disk I/O operations a VM with an *Instance Store* can perform for a second. They do so only for EBS backed instances. Hence, we benchmark our instances to get an estimate for I/O performance. We use the aforementioned VM characteristics to define the DC2 VMs in the simulation environment.

As a result of the aforementioned performance measurements, we have found that there are significant differences in the VM capacities in DC1 and DC2. While the VMs in the local environment DC1 have dedicated CPU cores, the ones in Amazon EC2 do not. Our measurements show that on average the CPU capacities of the AS and DB VMs in DC2 equal approximately 66% and 76% of the capacities of the respective servers in DC1. Similarly, the disk capacity of the DB server in DC2 is approximately 75% of the disk capacity of the DB server in DC1. Furthermore, the VMs in DC2 are standard Amazon EC2 instances with 1.7 GB of RAM, while our local VMs have only 512 MB of RAM. Hence with this experiment we demonstrate application modelling and simulation in heterogeneous environments with respect to both the observed performance and the incoming workload.

In order to simulate the workload we use the previously defined workload functionalities of the simulator. During execution we sample the performance utilizations every minute. In the simulation, we take utilization samples for every second and then define the utilization for every minute as the average of the samples for the included seconds.

Figure 13 displays the average VM utilizations of the servers for every 5 min, as predicted by the simulation. It shows that the predicted resource utilizations of the servers follow the patterns of the workloads shown in Fig. 12. As a result of the heterogeneity of the hardware and VM capacities in the two data centres, the utilizations of the servers in the two data centres differ significantly. For example, the CPU utilization of the AS server in DC2 at the peak times of the workload exceeds almost twice the utilization of the DC1 server at its peak workload time. This is because, the average CPU capacity of the AS server in DC2 is about 66% of the one in DC1. Similarly, the disk utilization during the workload peak of the DB server in DC2 is much higher than the one in DC1, as a result of having only 76% of the disk capacity of DC1. Given intensive workloads,

TABLE 4. 95% Confidence Intervals (CIs) and estimates of the median of the utilization differences between simulation and execution in DC1 and DC2. Measurements are in percentages.

Data centre	AS server, CPU		AS server, RAM		DB server, CPU		DB server, RAM		DB server, I/O	
	95% CI	Estimate	95% CI	Estimate	95% CI	Estimate	95% CI	Estimate	95% CI	Estimate
DC1	(−0.43, −0.37)	−0.4	(−4.06, −3.97)	−4.02	(−0.35, −0.31)	−0.33	(−1.06, −0.89)	−0.97	(−1.16, −0.97)	−1.07
DC2	(0.65, 0.74)	0.7	(−10.46, −10.31)	−10.39	(−2.78, −2.71)	−2.75	(−7.12, −7.07)	−7.1	(0.49, 0.58)	0.53

this performance heterogeneity can easily result in different contentions and consequently application performance in the different data centres.

Table 4 lists the 95% CIs and the estimations of the medians of the differences of the utilizations of the servers in DC1 and DC2. As in the previous experiment, we use the procedure of the Wilcoxon signed rank test with continuity correction to compute the 95% confidence intervals (CI) and the estimates. From the table we can see that the estimated error (the pseudomedian of the differences) is less than 11% for all characteristics. The accumulated delay for every simulated session in this experiment is 0. This is representative of the fact, that at execution time there were no resource contentions with the given workload.

8. CONCLUSION AND FUTURE WORK

We have introduced a new approach for performance modelling of 3-tier applications and their workload in Cloud and Multi-Cloud environments. Unlike others, our model is coarse-grained and session-based and does not represent each and every user interaction. Thus, it is suitable for quick prototyping and testing of scheduling, provisioning and load balancing approaches. We have implemented a CloudSim extension realising this model, which can be used for performance evaluation without the need to deploy large scale prototypes.

By definition a system model is a simplified description of the original system, and as such our model makes some simplifying assumptions. To validate it, we have conducted two experiments demonstrating the plausibility of our approach by comparing the simulation results with the results from an actual system execution. By using both public Cloud and private in-house infrastructures with different virtualization environments, settings and workload dynamism we demonstrated the plausibility of our approach in a wide range of environments.

Our future work will focus on: (i) modelling the entry point, which distributes the incoming workload among clouds, (ii) modelling asynchronous data processing, that would allow 'Big Data' jobs to be represented, (iii) incorporation of our model in other simulation environments and (iv) evaluation of the energy consumption impact caused by an application.

In fact CloudSim already supports power consumption modelling [13]. Moreover, Guérout *et al*. have defined DVFS (Dynamic Voltage and Frequency Scaling) aware extension of CloudSim allowing to test the effect of different DVFS approaches on data centre power consumption [60]. All of these functionalities have been implemented in the core CloudSim components (i.e. physical machines/hosts and processing elements/CPU) and thus can be reused by all extensions. Hence, while our approach is focused on application performance modelling and simulation, it can already be used to evaluate the energy consumption impact of application execution in different data centres with respect to their VM consolidation and DVFS management policies. We plan to study these aspects in more details in the future.

ACKNOWLEDGEMENTS

We thank Rodrigo Calheiros, Amir Vahid Dastjerdi and the rest of the CLOUDS lab members for their comments on improving the paper.

FUNDING

This research is partially supported by research grants from the Australian Research Council (ARC) for Discovery Projects (DP1093678 and DP130101378).

REFERENCES

[1] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J. and Brandic, I. (2009) Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, **25**, 599–616.

[2] Armbrust, M. et al. (2010) A view of cloud computing. *Commun. ACM*, **53**, 50–58.

[3] Foster, I., Zhao, Y., Raicu, I. and Lu, S. (2008) Cloud Computing and Grid Computing 360-Degree Compared. *Proc. Grid Computing Environments Workshop (GCE 2008)*, Austin, TX, USA, November 16, pp. 1–10. IEEE Computer Society Press.

[4] Ebay (June 21, 2013) Ebay. http://ebay.com/.

[5] Ebay Tech Blog (June 21, 2013) Cloud Bursting for Fun and Profit. http://ebaytechblog.com/2011/03/28/cloud-bursting-for-fun-and-profit/.

[6] Amazon (February 13, 2013) Public Sector Case Studies. http://aws.amazon.com/publicsector/customer-experiences/.
[7] Amazon (August 6, 2012) Summary of the AWS Service Event in the US East Region. http://aws.amazon.com/message/67457/.
[8] Microsoft (June 14, 2012). Windows Azure Service Disruption Update. http://blogs.msdn.com/b/windowsazure/archive/2012/03/01/windows-azure-service-disruption-update.aspx.
[9] IBM (May 31, 2013) IBM takes Australian Open data onto private cloud. http://www-07.ibm.com/innovation/au/ausopen/.
[10] Sotiriadis, S., Bessis, N. and Antonopoulos, N. (2011) Towards Inter-cloud Schedulers: A Survey of Meta-scheduling Approaches. *Proc. Int. Conf. on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, Barcelona, Catalonia, Spain, October 26–28, pp. 59–66. IEEE Computer Society Press.
[11] Grozev, N. and Buyya, R. (2012) Inter-cloud architectures and application brokering: taxonomy and survey. *Software: Practice and Experience*, doi: 10.1002/spe.2168.
[12] Buyya, R. and Murshed, M. (2002) GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation: Practice and Experience*, **14**, 1175–1220.
[13] Calheiros, R.N., Ranjan, R., Beloglazov, A., Rose, C.A.F.D. and Buyya, R. (2011) CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, **41**, 23–50.
[14] Kliazovich, D., Bouvry, P. and Khan, S.U. (2012) GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. *J. Supercomputing*, **62**, 1263–1283.
[15] Dejun, J., Pierre, G. and Chi, C.-H. (2009) EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications. *Proc. Int. Conf. Service-Oriented Computing (ICSOC 2009)*, Stockholm, Sweden, November 23–27, pp. 197–207. Springer, Berlin, Heidelberg.
[16] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M. and Tantawi, A. (2005) An Analytical Model for Multi-Tier Internet Services and its Applications. *Proc. Int. Conf. Measurement and Modeling of Computer Systems ACM SIGMETRICS 2005*, Banff, Alberta, Canada, June 6–10, pp. 291–302. ACM, New York, NY, USA.
[17] Zhang, Q., Cherkasova, L. and Smirni, E. (2007) A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. *Proc. 4th Int. Conf. Autonomic Computing (ICAC 2007)*, Jacksonville, FL, USA, June 11–15, 27. IEEE Computer Society.
[18] Bi, J., Zhu, Z., Tian, R. and Wang, Q. (2010) Dynamic Provisioning Modeling for Virtualized Multi-tier Applications in Cloud Data Center. *Proc. 3rd Int. Conf. Cloud Computing (CLOUD 2010)*, Miami, FL, USA, July 5–10, pp. 370–377. IEEE Computer Society Press.
[19] Zhang, Z. and Fan, W. (2008) Web server load balancing: a queueing analysis. *Eur. J. Oper. Res.*, **186**, 681–693.
[20] Kamra, A., Misra, V. and Nahum, E. (2004) Yaksha: A Self-Tuning Controller for Managing the Performance of 3-Tiered Web Sites. *Proc. 12th IEEE Int. Worksh. Quality of Service (IWQOS 2004)*, Montreal, Canada, June 7–9, pp. 47–56. IEEE Computer Society Press.
[21] Chen, Y., Iyer, S., Liu, X., Milojicic, D. and Sahai, A. (2007) SLA Decomposition: Translating Service Level Objectives to System Level Thresholds. *Proc. 4th Int. Conf. Autonomic Computing (ICAC 2007)*, Jacksonville, FL, USA, June 11–15, pp. 3–13. IEEE Computer Society.
[22] Lim, S.-H., Sharma, B., Nam, G., Kim, E.K. and Das, C. (2009) MDCSim: A Multi-tier Data Center Simulation, Platform. *Proc. IEEE Int. Conf. Cluster Computing and Workshops (CLUSTER '09)*, New Orleans, LA, USA, August 31–September 4, pp. 1–9. IEEE Computer Society Press.
[23] Sriram, I. (2009) SPECI, a Simulation Tool Exploring Cloud-Scale Data Centres. In Jaatun, M., Zhao, G. and Rong, C. (eds.), *Cloud Computing*, pp. 381–392. Lecture Notes in Computer Science, Vol. 5931. Springer, Berlin Heidelberg.
[24] NúÑez, A., VÁzquez-Poletti, J., Caminero, A., Castañé, G., Carretero, J. and Llorente, I. (2012) iCanCloud: a flexible and scalable cloud infrastructure simulator. *J. Grid Comput.*, **10**, 185–209.
[25] Fowler, M. (2003) *Patterns of Enterprise Application Architecture*. Addison-Wesley, New York, NY, USA.
[26] Java EE (June 21, 2013) Java EE. http://oracle.com/technetwork/java/javaee/overview/index.html.
[27] Ruby on Rails (June 21, 2013) Ruby on Rails. http://rubyonrails.org/.
[28] Django (June 21, 2013) Django. https://djangoproject.com/.
[29] Ramirez, A.O. (2000) Three-Tier architecture. *Linux J.*, **2000**.
[30] Aarsten, A., Brugali, D. and Menga, G. (1996) Patterns for Three-tier Client/server Applications. *Proc. Pattern Languages of Programs (PLoP '96)*, Monticello, IL, USA, September 4–6. Addison-Wesley, Reading, MA, USA.
[31] Cattell, R. (2010) Scalable SQL and NoSQL data stores. *SIGMOD Record*, **39**, 12–27.
[32] Adler, B. (February 13, 2013) Building Scalable Applications in the Cloud. http://rightscale.com/info_center/white-papers/RightScale_White_Paper_Building_Scalable_Applications.pdf.
[33] Amazon (February 13, 2013) Amazon Relational Database Service (Amazon RDS). http://aws.amazon.com/rds/.
[34] Oracle (May 31, 2013) Oracle Database and the Oracle Database Cloud. http://www.oracle.com/technetwork/database/database-cloud/public/oracle-db-and-db-cloud-service-wp-1844127.pdf.
[35] Oracle (May 31, 2013) Oracle Database Cloud Service. http://www.oracle.com/us/solutions/cloud/overview/database-cloud-service-wp-1844123.pdf.
[36] Ferrer A. J. et al. (2012) OPTIMIS: a holistic approach to cloud service provisioning. *Future Gen. Comput. Syst.*, **28**, 66–77.
[37] Arjuna Agility (June 14, 2012) What Is Federation. http://arjuna.com/what-is-federation.
[38] Rochwerger, B. et al. (2009) The RESERVOIR model and architecture for open federated cloud computing. *IBM J. Res. Dev.*, **53**, 1–11.
[39] Petcu, D. (2013) Multi-Cloud: Expectations and Current Approaches. *Proc. Int. Worksh. Multi-cloud Applications and Federated Clouds (MultiCloud '13)*, Prague, Czech Republic, April 22, pp. 1–6. ACM, New York, NY, USA.
[40] RightScale (June 14, 2012) RightScale. http://rightscale.com/.
[41] Enstratius (May 31, 2013) Enstratius. https://enstratius.com/.
[42] Scalr (June 14, 2012) Scalr. http://scalr.net/.
[43] JClouds (June 14, 2012) JClouds. http://jclouds.org/.

[44] Apache Foundation (June 14, 2012) Apache Libcloud. http://libcloud.apache.org/.

[45] Apache Foundation (June 14, 2012) Apache Delta Cloud. http://deltacloud.apache.org/.

[46] Helland, P. (2012) Condos and clouds. *ACM Queue*, **10**, 20–35.

[47] Cao, J., Andersson, M., Nyberg, C. and Kihl, M. (2003) Web Server Performance Modeling Using an M/G/1/K*PS Queue. *Proc. 10th Int. Conf. Telecommunications (ICT'03)*, Tahiti, Papeete, French Polynesia, February 23–March 1, pp. 1501–1506. IEEE Computer Society Press.

[48] Robertson, A., Wittenmark, B. and Kihl, M. (2003) Analysis and Design of Admission Control in Web-server Systems. *Proc. American Control Conference (ACC '03)*, Denver, CO, USA, June 4–6, pp. 254–259. IEEE Computer Society Press.

[49] Paxson, V. and Floyd, S. (1995) Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Trans. Network.*, **3**, 226–244.

[50] Ilyadis, N. (2012) The Evolution of Next-generation Data Center Networks for High Capacity Computing. *Proc. Symp. VLSI Circuits (VLSIC'12)*, June 13–15, Honolulu, HI, USA, pp. 1–5. IEEE Computer Society Press.

[51] Jung, G., Swint, G., Parekh, J., Pu, C. and Sahai, A. (2006) Detecting Bottleneck in n-Tier IT Applications Through Analysis. In State, R., Meer, S., O'Sullivan, D. and Pfeifer, T. (eds.), *Large Scale Management of Distributed Systems*, pp. 149–160. Lecture Notes in Computer Science, Vol. 4269. Springer, Berlin Heidelberg.

[52] Malkowski, S., Jayasinghe, D., Hedwig, M., Park, J., Kanemasa, Y. and Pu, C. (2010) Empirical Analysis of Database Server Scalability using an N-tier Benchmark with Read-intensive Workload. *Proc. ACM Symp. Applied Computing (SAC '10)*, Sierre, March 22–26, pp. 1680–1687. Switzerland SAC'10. ACM, New York, NY, USA.

[53] Lloyd, W., Pallickara, S., David, O., Lyon, J., Arabi, M. and Rojas, K. (2013) Performance implications of multi-tier application deployments on Infrastructure-as-a-Service clouds: Towards performance modeling. *Future Generation Computer Systems*, **29**, 1254–1264.

[54] RUBiS (March 15, 2013) RUBiS: Rice University Bidding System. http://rubis.ow2.org/.

[55] Amza, C., Chanda, A., Cox, A., Elnikety, S., Gil, R., Rajamani, K., Zwaenepoel, W., Cecchet, E. and Marguerite, J. (2002) Specification and Implementation of Dynamic Web Site Benchmarks. *Proc. IEEE Int. Worksh. Workload Characterization (WWC-6)*, Austin, TX, USA, October 27–27, pp. 3–13. IEEE Computer Society Press.

[56] TR02-388 (2002) Bottleneck Characterization of Dynamic Web Site Benchmarks. Technical Report, Department of Computer Science Rice University, Houston, TX, USA.

[57] Cecchet, E., Chanda, A., Elnikety, S., Marguerite, J. and Zwaenepoel, W. (2003) Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. *Proc. ACM/IFIP/USENIX Int. Conf. Middleware (Middleware '03)*, Rio de Janeiro, June 16–20, pp. 242–261. Brazil Middleware '03. Springer, New York, NY, USA.

[58] TPC BenchmarkTM W 1.8 (2002) TPC BENCHMARK W (Web Commerce). Specification, version 1.8. Transaction Processing Performance Council (TPC), San Francisco, CA, USA.

[59] SYSSTAT (March 15, 2013) SYSSTAT. http://sebastien.godard.pagesperso-orange.fr/.

[60] Guérout, T., Monteil, T., Costa, G. D., Calheiros, R. N., Buyya, R. and Alexandru, M. (2013) Energy-aware simulation with DVFS. *Simul. Model. Practice Theory*, doi:10.1016/j.simpat.2013.04.007.