SLA-Based and Consumer-Centric Dynamic Provisioning for Cloud Databases

Sherif Sakr [*†], Anna Liu [*†]

[*]*National ICT Australia (NICTA)*
[†]*School of Computer Science and Engineering*
University of New South Wales, Australia
{firstname.lastname}@Nicta.com.au

***Abstract*—One of the main advantages of the cloud computing paradigm is that it simplifies the time-consuming processes of hardware provisioning, hardware purchasing and software deployment. Currently, we are witnessing a proliferation in the number of cloud-hosted applications with a tremendous increase in the scale of the data generated as well as being consumed by such applications. Cloud-hosted database systems powering these applications form a critical component in the software stack of these applications. Service Level Agreements (SLA) represent the contract which captures the agreed upon guarantees between a service provider and its customers. The specifications of existing service level agreement (SLA) for cloud services are not designed for flexibly handling even relatively straightforward performance and technical requirements of consumer applications. The concerns of consumers for cloud services regarding the SLA management of their hosted applications within the cloud environments will gain increasing importance as cloud computing becomes more pervasive. This paper introduces the notion, challenges and the importance of SLA-based provisioning and cost management for cloud-hosted databases from the *consumer* perspective. We present an end-to-end framework that acts as a middleware which resides between the consumer applications and the cloud-hosted databases. The aim of the framework is to facilitate adaptive and dynamic provisioning of the database tier of the software applications based on application-defined policies for satisfying their own SLA performance requirements, avoiding the cost of any SLA violation and controlling the monetary cost of the allocated computing resources. The experimental results demonstrate that SLA-based provisioning is more adequate for providing consumer applications the required flexibility in achieving their goals.**

***Keywords*-Cloud Database, Service Level Agreements, Elasticity**

I. Introduction

Cloud computing technology represents a new paradigm for the provisioning of computing infrastructure. This paradigm shifts the location of this infrastructure to the network to reduce the costs associated with the management of hardware and software resources. It represents the long-held dream of envisioning computing as a utility [1] where the economy of scale principles help to effectively drive down the cost of computing infrastructure. Cloud computing simplifies the time-consuming processes of hardware provisioning, hardware purchasing and software deployment. Therefore, it promises a number of advantages for the deployment of data-intensive applications such as elasticity of resources, pay-per-use cost model, low time to market and the perception of (virtually) unlimited resources and infinite scalability. Hence, it becomes possible, at least theoretically, to achieve unlimited throughput by continuously adding computing resources (e.g. database servers) if the workload increases.

In practice, the advantages of the cloud computing paradigm opens up new avenues for deploying novel applications which were not economically feasible in a traditional enterprise infrastructure setting. Therefore, the cloud has become an increasingly popular platform for hosting software applications in a variety of domains such as e-retail, finance, news and social networking. Thus, we are witnessing a proliferation in the number of applications with a tremendous increase in the scale of the data generated as well as being consumed by such applications. Cloud-hosted database systems powering these applications form a critical component in the software stack of these applications. In general, successful cloud data management systems should satisfy, as much as possible, the following goals [2]:

- *Availability*: They must be always accessible even on the occasions where there is a network failure or a whole datacenter has gone offline.
- *Scalability*: They must be able to support very large databases with very high request rates at very low latency. In particular, the system must be able to automatically replicate and redistribute data to take advantage of the new hardware. They must be also able to automatically move load between servers (replicas).
- *Elasticity*: They must cope with changing application needs in both directions (scaling up/out or scaling down/in).
- *Performance*: On public cloud computing platforms, pricing is structured in a way such that one pays only for what one uses, so the vendor price increases linearly with the requisite storage, network bandwidth, and compute power. Hence, the system performance has a direct effect on its costs. Thus, efficient system performance is a crucial requirement to save money.

Cloud computing is by its nature a fast changing environment which is designed to provide services to unpredictably diverse sets of clients and heterogenous workloads. Several studies have also reported that the variation of the performance of cloud computing resources is high. These characteristics raise serious concerns from cloud consumers' perspective regarding the manner in which the SLA of their application can be managed. According to a Gartner market report released in November 2010, SaaS is forecast to have a 15.8% growth rate through 2014 which makes SaaS and cloud very interesting to services industry, but the viability of the business models depends on the practicality and the success of the terms and conditions (SLAs) being offered by the service provider(s) in addition to their satisfaction to the service consumers. Therefore, successful SLA management is a critical factor to be considered by both providers and consumers alike. Existing service level agreements (SLAs) of cloud providers are not designed for supporting the requirements and restrictions under which SLA of consumers' applications need to be handled. Particularly, most providers guarantee only the availability (but not the performance) of their services [3]. Therefore, consumer concerns on SLA handling for their cloud-hosted databases along with the limitations of existing SLA frameworks to express and enforce SLA requirements in an automated manner creates the need for SLA-based management techniques for cloud-hosted databases.

The focus of this paper is on introducing the notion of SLA-based management for cloud-hosted database from the *consumer* perspective. We present an end-to-end framework that enables the software applications to declaratively define the SLA of the application in terms of *goals* which are subjected to a number of constraints that are specific on its database transactions. The framework also enables the software provider to declaratively define a set of application-specific rules (action rules) where the admission control of the database tier needs to take corresponding actions in order to meet the expected system performance or to reduce the cost of the allocated cloud resources when they are not efficiently utilized. The framework continuously monitors the database workload, tracks the satisfaction of the application-defined SLA, evaluates the condition of the action rules and takes the necessary actions when required. The framework is database platform-agnostic and relies on virtualization-based database replication mechanism.

The reminder of this paper is structured as follows. Section II discusses the different options of deploying the database tier of software applications on cloud platforms. Section III introduces the challenges of SLA Management for Cloud-Hosted Databases. The details of our framework are described in Section IV. Experimental evaluation of our framework is presented in Section V. We review the related work in Section VI before we conclude the paper in Section VII.

II. Options of Hosting Cloud Databases

In practice, there are different approaches for hosting the database tier of software application in cloud platforms. The *platform-supplied storage layer* approach relies on a new wave of storage platforms named as key-value stores or NoSQL (Not Only SQL) systems. These systems are designed to achieve high throughput and high availability by giving up some functionalities that traditional database systems offer such as joins and ACID transactions [4]. For example, most of NoSQL systems provide simple call level data access interfaces (in contrast to a SQL binding) and rely on weaker consistency management protocols (e.g. eventual consistency [5]). Cloud offerings of this approach include Amazon S3, Amazon SimpleDBand Microsoft Azure Table Storage. In practice, migrating existing software application that uses relational database to NoSQL offerings would require substantial changes in the software code due to the differences in the data model, query interface and transaction management support. In addition, developing applications on top of an eventually consistent NoSQL datastores requires a higher effort compared to traditional databases because they hinder important factors such as data independence, reliable transactions, and other cornerstone characteristics often required by applications that are fundamental to the database industry [6]. In practice, the majority of today's platform-supplied storage systems are suitable for OLAP applications, but not for OLTP [7].

The *relational database as a service* is another approach in which a third party service provider hosts a relational database as a service [8]. Such services alleviate the need for their users to purchase expensive hardware and software, deal with software upgrades and hire professionals for administrative and maintenance tasks. Cloud offerings of this approach include Amazon RDSand Microsoft SQL Azure. For example, Amazon RDS provides access to the capabilities of MySQL or Oracle database while Microsoft SQL Azure has been built on Microsoft SQL Server technologies. As such, users of these services can leverage the capabilities of traditional relational database systems such as creating, accessing and manipulating tables, views, indexes, roles, stored procedures, triggers and functions. It can also execute complex queries and joins across multiple tables. The migration of the database tier of any software application to a relational database service is supposed to require minimal effort if the underlying RDBMS of the existing software application is compatible with the offered service. However, many relational database systems are not, yet, supported by the DaaS paradigm (e.g. DB2, Postgres). In addition, some limitations or restrictions might be introduced by the service provider for different reasons[1]. Moreover, the *consumer* applications do not have enough flexibility in controlling the allocated resources of their applications (e.g. dynamically allocating more resources for dealing with increasing workload or dynamically reducing the allocated resources in order to reduce the operational cost). The whole resource management and allocation process is controlled at the provider side.

Virtualization is a key technology of the cloud computing paradigm. Virtual machine technologies are increasingly being used to improve the manageability of software systems and lower their total cost of ownership. They allow resources to be allocated to different applications on demand and hide the complexity of resource sharing from cloud users by providing a powerful abstraction for application and resource provisioning. In particular, resource virtualization technologies add a flexible and programmable layer of software between

[1]http://msdn.microsoft.com/en-us/library/windowsazure/ee336245.aspx

applications and the resources used by these applications. The *application-managed* approach takes an existing application designed for a conventional data center, and then port it to virtual machines in the public cloud. Such migration process usually requires minimal changes in the architecture or the code of the deployed application. In this approach, database servers, like any other software components, are migrated to run in virtual machines. *The work of this paper belongs to this approach*. In principle, one of the main advantages of the *application-managed* approach is that the application can have the full control in dynamically allocating and configuring the physical resources of the database tier (database servers) as needed [9], [10], [11], [12]. Hence, software applications can fully utilize the elasticity feature of the cloud environment to achieve their defined and customized scalability or cost reduction goals. However, achieving these goals requires the existence of an admission control component which is responsible for monitoring the system state and taking the corresponding actions (e.g. allocating more/less computing resources) according to the defined application requirements and strategies. Several approaches have been proposed for building admission control components based on the *utilization* of the allocated resources [10], [11]. In our approach, we focus on building *SLA-based* admission control as a more *consumer-centric* view for achieving the requirements of their applications.

III. Challenges of SLA Management for Cloud-Hosted Databases

An SLA is a contract between a service provider and its customers. *Service Level Agreements* (SLAs) capture the agreed upon guarantees between a service provider and its customer. They define the characteristics of the provided service including service level objectives (SLOs), like maximum response times, minimum throughput rates and data freshness, and define penalties if these objectives are not met by the service provider. In general, SLA management is a common general problem for the different types of software systems which are hosted in cloud environments for different reasons such as: the unpredictable and bursty workloads from various users in addition to the performance variability in the underlying cloud resources. In particular, there are three typical parties in the cloud. To keep terminology straight through the rest of the paper, these parties are defined as follows:

1) *Cloud Service Providers (CSP)*: They offer client-provisioned and metered computing resources (e.g. CPU, storage, memory, network) that can be rented for flexible time durations. In particular, they include: Infrastructure-as-a-service providers (IaaS) and Platform-as-a-service providers (PaaS). Examples are: Amazon, Microsoft, Google and Rackspace.
2) *Cloud Consumers*: They represent the cloud-hosted software applications that utilize the services of CSP and are financially responsible for their resource consumptions.
3) *End Users*: They represent the legitimate users for the services (applications) that are offered by cloud consumers.

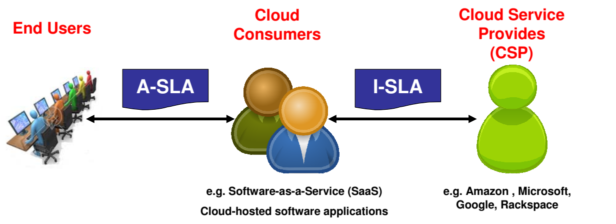

End Users
Cloud Consumers
Cloud Service Provides (CSP)
A-SLA
I-SLA
e.g. Software-as-a-Service (SaaS)
Cloud-hosted software applications
e.g. Amazon , Microsoft, Google, Rackspace

Fig. 1. SLA Parties in Cloud Environments

While cloud service providers charge cloud consumers for renting computing resources to deploy their applications, cloud consumers may charge their end user users for processing their workloads (e.g. Software-as-a-Service) or may process the user requests for free (cloud-hosted business application). In both cases, the cloud consumers need to guarantee their users' SLA. Penalties are applied in the case of SaaS and reputation loss is incurred in the case of cloud-hosted business applications. For example, Amazon found every 100ms of latency cost them 1% in sales and Google found an extra 500ms in search page generation time dropped traffic by 20%[2]. In addition, large enterprise web applications (e.g., eBay and Facebook) need to provide high assurance in terms of SLA metrics such as response times and service availability to their users. Without such assurances, service providers of these applications stand to lose their user base, and hence their revenues.

In practice, resource management and SLA guarantee falls into two layers: the cloud service providers and the cloud consumers. In particular, the cloud service provider is responsible for the efficient utilization of the physical resources and guarantee their availability for their customers (cloud consumers). The cloud consumers are responsible for the efficient utilization of their allocated resources in order to satisfy the SLA of their customers (end users) and achieve their business goals. Therefore, we distinguish between two types of service level agreements (SLAs):

1) *Cloud Infrastructure SLA (I-SLA)*: These SLA are offered by cloud providers to cloud consumers to assure the quality levels of their cloud computing resources (e.g., server performance, network speed, resources availability, storage capacity).
2) *Cloud-hosted Application SLA (A-SLA)*: These guarantees relate to the levels of quality for the software applications which are deployed on a cloud infrastructure. In particular, cloud consumers often offer such guarantees to their application's end users in order to assure the quality of services that are offered such as the application's response time and availability.

Figure 1 illustrates the relationship between *I-SLA* and *A-SLA* in the software stack of cloud-hosted applications. In practice, traditional cloud monitoring technologies (e.g. Amazon CloudWatch[3]) focus on low-level computing resources

[2]http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html
[3]http://aws.amazon.com/cloudwatch/

(e.g. CPU speed, CPU utilization, disk speed). In principle, translating the SLAs of applications' transactions to the thresholds of utilization for low-level computing resources is a very challenging task and is usually done in an ad-hoc manner due to the complexity and dynamism inherent in the interaction between the different tiers and components of the system in addition to the possible fluctuations on the workload characteristics which could be of several orders of magnitude on the same business day. Therefore, it becomes a significant issue for the cloud consumers to monitor and adjust the deployment of their systems if they intend to offer viable service level agreements (SLAs) to their customers (end users). Failing to achieve this goals will jeopardize the sustainable growth of cloud computing in the future and may result in valuable applications move away from the cloud.

In general, cloud computing technology faces key challenges on providing certainty to the commodities that are exchanged among buyers and sellers. In particular, one complexity that arises with the virtualization technology is that it becomes harder to provide performance guarantees and to reason about a particular application's performance because the performance of an application hosted on a virtual machine becomes a function of applications running in other virtual machines hosted on the same physical machine. In addition, it may be challenging to harness the full performance of the underlying hardware, given the additional layers of indirection in virtualized resource management [13]. Several studies have reported that the variation of the performance of cloud computing resources is high [14], [15]. As a result, currently, cloud service providers (IaaS or PaaS) do not provide adequate SLAs for their service offerings. Particularly, most providers guarantee only the availability (but not the performance and technical requirements) of their services [1], [16]. In practice, it is a very challenging goal to delegate the management of the SLA requirements of the consumer applications to the side of the cloud service provider due to the wide heterogeneity in the workload characteristics, details of SLA requirements and cost management objectives of the very large number of consumer applications that can be simultaneously running in a cloud environment. The same challenging issue exists for the other two options of hosting cloud databases as well (the *platform-supplied storage layer* and the *relational database as a service*). Therefore, the burden to manage and assure their application SLA (in terms of performance) to their end users rests on the shoulders of cloud consumers themselves [12]. In addition, the cloud consumers would still need to improve their utilization of the renting cloud resource and thus reduce their operating cost.

In practice, enterprise applications and services are typically comprised of a large number of components that are deployed in different tiers (e.g. web server tier, application tier and database tier) and interact with one another in a complex manner. In principle, the performance of each tier potentially affects the overall behavior of the system. Hence, any high level goal (e.g., performance, availability, security, etc.) specified for the service potentially relates to all components in the different tiers. *The focus of our presented framework is on the database tier of the software stack.*

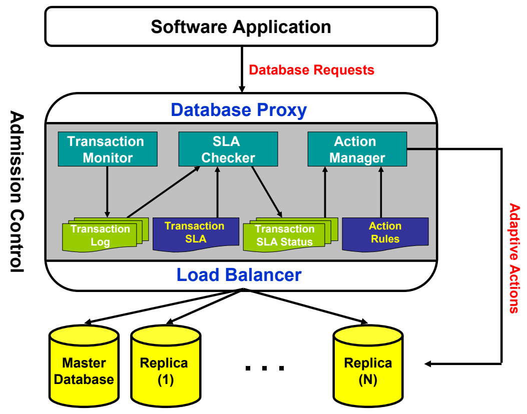


Fig. 2. Framework Architecture.

IV. SLA-Based Management for the Database Tier

A. Overview of Framework Architecture

Cloud computing is by its nature a fast changing environment which is designed to provide services to unpredictably diverse sets of clients and heterogenous workloads [17]. Dynamic provisioning techniques for computing resources are commonly used as an effective means for handling workload fluctuations. Our framework is designed to facilitate adaptive and dynamic provisioning of the database tier for cloud-hosted software applications. In particular, it provides a *declarative* and automated management for the application-defined SLA in terms of their performance requirements of the database transactions while minimizing the cost of used cloud resources.

Database proxying is the ability to forward database requests to a database systems using an intermediate piece of software, the proxy, and to return the results from those request transparently to the client program without the need of having any database drivers installed. In particular, a database proxy software is a simple program that sits between the client application and the database server that can monitor, analyze or transform their communications. Such flexibility allows for a wide variety of uses such as: load balancing, query analysis and query filtering. On the runtime of our framework, we use the database proxying mechanism to perform a lightweight logging and monitoring process for the database transactions of the application workloads. In addition, we utilize the proxy layer for automatically distributing the application workload between the immediately available database replicas at anytime. Figure 2 shows an overview of our framework architecture with the following main components:

- **Transaction/Workload Monitor**: Continuously logs and monitors the executed database operations of the application workloads. During the monitoring process, it correlates the received database operations based on their sender, detect the transaction patterns (which is defined during the SLA declaration process) and measures the total execution times of the workload transactions (as the sum of its raw database operations).

- **SLA Checker**: Responsible for checking the results of the monitoring module and comparing them against the application-defined SLA of the different transaction types in order to report the percentage of SLA violations for each of them (if any). The SLA checker relies on a dynamic scheduler that uses the transaction rate as a main metric. Thus, it ensures that when the load is high, the SLA checks run more frequently since its most likely for the violations to happen during these periods of time.
- **Action Manager**: Continuously evaluates the conditions of the application-defined action rules (e.g. *scaling out/in* the database tier) and executes the necessary action when these conditions are satisfied.

B. Dynamic Provisioning for the Database Tier

In general, dynamic provisioning at the database-tier involves increasing or decreasing the number of database servers allocated to an application in response to workload changes. *Data replication* [18] is a well-known strategy to achieve the availability, scalability and performance improvement goals in the data management world. In particular, when the application load increases and the database tier becomes the *bottleneck* in the stack of the software application, there are two main options for achieving scalability at the database tier to enable the application to cope with more client requests:

1) *Scaling up* aims at allocating a bigger machine with more horsepower (e.g. more processors, memory, bandwidth) to act as a database server.
2) *Scaling out* aims at replicating the data across more machines.

In practice, the scaling up option has the main drawback that large machines are often very expensive and eventually a physical limit is reached where a more powerful machine cannot be purchased at any cost. Alternatively, it is both extensible and economical, especially in a dynamic workload environment, to scale out by adding another commodity server which fits well with the pay-as-you-go pricing philosophy of cloud computing. In database replication, there are two main replication strategies: *master-slave* and *multi-master*. In master-slave, updates are sent to a single master node and lazily replicated to slave nodes. Data on slave nodes might be stale and it is the responsibility of the application to check for data freshness when accessing a slave node. Multi-master replication enforces a serializable execution order of transactions between all replicas so that each of them applies update transactions in the same order. This way, any replica can serve any read or write request. For the sake of simplicity of achieving the consistency goal among the database replicas and reducing the effect of network communication latency, our framework is currently employing the master-slave replication strategy using the ROWA (read-once-write-all) protocol on the Master copy [19]. However, our framework can be easily extended to support the multi-master replication strategy as well. In general, provisioning of a new database replica involves extracting database content from an existing replica and copying that content to a new replica. In practice, the time of executing these operations mainly depends on the database size. To provision database replicas in a timely fashion, it is necessary to periodically snapshot the database state in order to minimize the database extraction and copying time to that of only the snapshot synchronization time. Clearly, there is a tradeoff between the time to snapshot the database, the size of the transactional log and the amount of update transactions in the workload. In our framework this trade-off can be controlled by application-defined parameters. This tradeoff can be further optimized by applying recently proposed live database migration techniques [20], [21].

C. SLA-Based Provisioning and Cost Management

In practice, there exist many forms of SLA metrics for cloud-hosted database (e.g. response time, throughput, data freshness). In this paper, we focus on the SLA metric of the total execution times of database transactions (the time between a transaction is presented to the database system and the time when the transaction execution is completed) more than other metrics such as database throughput because in practice it has the greatest impact on the user experience and his satisfaction of the underlying services. In other words, individual users are generally more concerned about when their transaction will complete rather than how many transactions the system will be able to execute in a second [22]. To illustrate, assuming a transaction (T) with an associated SLA for its execution time (S) is presented to the system at time 0, if the system is able to finish the execution of the transaction at time ($t \leq S$) then the service provider achieved his target otherwise if ($t \geq S$) then the transaction response cannot be delivered within the defined SLA and hence a penalty p is incurred.

In practice, the SLA requirements can vary between the different types of application transactions (for example, a login application request may have an SLA of 100 ms execution time, a search request may have an SLA of 600 ms while a request of submitting an order information would have 1500 ms). Obviously, the variations in the SLA of different applications transactions is due to their different natures and their differences in the consumption behaviour of system resources (e.g. disk I/O, CPU time). These transaction-specific SLAs are declaratively defined using an XML dialect that can be interpreted by the monitoring and SLA checker modules of our framework. In practice, each business transaction can send one or more operations to the underlying database system. Hence, in our XML dialect, each transaction is described as pattern(s) of SQL commands where the transaction execution time is computed as the total execution time of these individual operations in the described pattern. Therefore, the monitoring module correlates the received database operations based on their sender in order to detect the transaction patterns.

In general, optimizing the application-defined SLA while minimizing the *monetary cost* of used computing resources is a vital goal for the software/application providers. However the demands of the service consumers can vary significantly.

Thus, the service provider has to allocate sufficient resources to support the customer workloads or else he will face SLA violations that could have serious consequences in the form of lost revenue and damaged reputation. In particular, the service provider will have to *scale out* the database tier (adding more replicas/resources) when it deems that existing database replicas are insufficient to accommodate the incoming workload. Meanwhile, the software provider will have to *scale in* the database tier (releasing existing replicas/resources) if he can meet decreasing application workload with a less number of replicas. In fast changing and unpredictable environments, the approaches that rely on *capacity planning* and *workload forecasting* techniques face very challenging conditions. When the capacity of allocated resources is planned for the average workload, there would be less cost incurred due to less hardware used but performance will be a problem when peak load occurs. Bad performance will discourage customers and revenue will be affected. On the other hand if the allocated resource capacity is planned for peak workload, resources will remain idle most of the time. Therefore, SLA-based cost management in cloud computing environments overcomes these challenges and provides an effective solution for controlling the monetary cost of the allocated computing resources according to the specific policies of the hosted application.

Our framework enables the service provider to declaratively define application-specific *action rules* to adaptively scale out or scale in according to the monitored status of the system (e.g. application workload, SLA satsifaction/violation). As an example, an application can define to scale out the underlying database tier if the average percentage of SLA violation for transactions T_1 and T_2 exceeds 10% (of the total number of T_1 and T_2 transactions) for a continuous period of more than 8 minutes. Similarly, the application can define to scale in the database tier if the average percentage of SLA violation for transactions T_1 and T_2 is less than 2% for a continuous period that is more than 8 minutes and the average number of concurrent users per database replica is less than 25. In our framework, these action rules are declaratively defined using another XML dialect that can be interpreted by the *Action Manager* module which continuously evaluates the conditions of the application-defined rules and triggers the execution of their corresponding actions as necessary. The different variables that can be used for defining the action rules are described as follows:

- *TransPrice* (T_i, S_i, V_i): is the amount of monetary/satisfaction units (V_i) that an end user will pay/get if a cloud consumer application brings the transaction (T_i) to completion according to the defined SLA (S_i).
- *TransPenalty* (T_i, S_i, V_i): is the amount of monetary/dissatisfaction units (V_i) that the cloud consumer application (T_i) will pay if it violates the defined SLA (S_i) of the end user transaction (T_i).
- *SLA-Satisfaction* (T_i, P_i, I_i): represents the percentage (P_i) of the instances of the application transaction (T_i) that satisfied their defined SLA during the last period of time units (I_i).
- *SLA-DisSatisfaction* (T_i, P_i, I_i): represents the percentage (P_i) of the instances of the application transaction (T_i) that did not fulfill their defined SLA during the last period of time units (I_i).
- *SLA-ViolationMagnitude* (T_i, N_i, M_i, I_i): represents the number of the instances (N_i) of the application transaction (T_i) that did not fulfill their defined SLA and the average magnitude of SLA violation (M_i) during the last period of time units (I_i).
- *SLA-Threshold* (T_i, P_i, I_i): defines the minimum percentage (P_i) of the instances of the application transaction (T_i) that should fulfill their defined SLA during the last period of time units (I_i).
- *ReplicaCost* (C): is the amount of monetary units (C) which will be paid by the cloud consumer to the cloud service provider (per hour) for renting the required resources for hosting an addition replica of the database tier. This parameter also represents the amount of monetary units which will be saved by the cloud consumer when scaling in the database tier by releasing the allocated resources for hosting a database replica
- *[Min/Max]Replicas*: represents the boundary limits for the minimum/maximum number of replicas that can be allocated for the database tier of the software application.

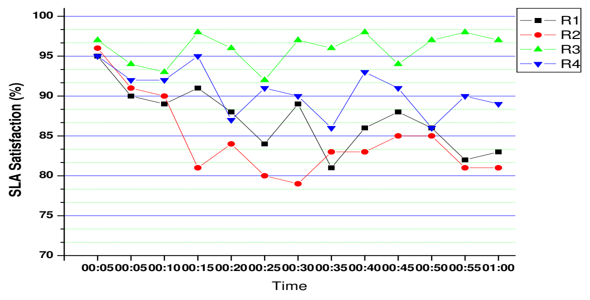
In principle, one of the main advantages of our approach is that it enables the cloud-hosted software applications to control the changing needs of their applications (in terms of SLA guarantees or cost optimization of allocated resources) in a convenient and flexible way. While this paper focuses on the SLA metric of transaction execution time, other *consumer* metrics can be implemented and integrated in the same manner such as *freshness* of replicated data [23], [24], [25].

V. Experimental Evaluation

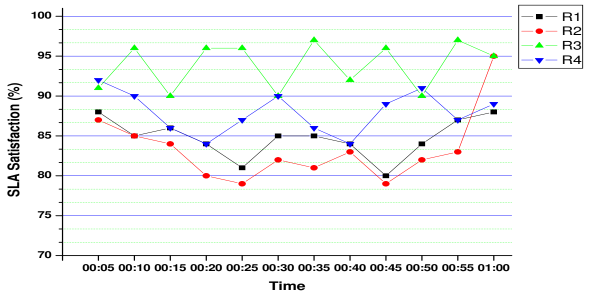
We implemented[4] the components of our framework using a MySQL proxy software[5] and *Lua* scripting language[6]. We conducted our experiments using the berkeley Cloudstone benchmark [26] which follows the canonical three-tier Web application architecture: a web server, an application server and a database server (MySQL database server) where each tier is hosted on a virtual machine. We deploy the application tiers on Amazon EC2 instances as our public cloud where we use small EC2 instances for hosting the virtual machines of the database servers. From the Cloudstone scenario, we borrow the idea of using the following components:

- *Olio*[7]: A social-events application. The application has different types of transactions such as: user login, tag search, creating an event, attending an event, rating an event, adding person and rating an event.
- *Faban*[8]: An open source synthetic workload generator which is used to compose our application workload.

[4]http://cdbslaautoadmin.sourceforge.net/
[5]http://dev.mysql.com/downloads/mysql-proxy/
[6]http://forge.mysql.com/wiki/Lua
[7]http://incubator.apache.org/olio
[8]http://faban.sunsource.net



(a) Workload: 80/20 ((r/w)



(b) Workload: 50/50 ((r/w)

Fig. 3. Comparison of SLA-Based vs Resource-Based Database Provisioning Rules

The goal of our experiments is to evaluate the behavior of our approach for provisioning the database tier of the software application based on monitoring the application-defined SLA in comparison to the traditional mechanism of monitoring the utilization of allocated computing resources. The Cloudstone benchmark mimics a Web 2.0 social events calendar that allows users to perform individual operations (e.g. browsing, searching and creating events), as well as, social operations (e.g. joining and tagging events). We defined two configurations of the read/write ratios of the database workload: 50/50 and 80/20. Each workload starts with 50 users and gradually increases at a fixed step of 20 users each 5 minutes. The experiment starts with a single database copy then the number of the database replicas increases in reaction to the workload increase.

We conducted our experiments with 4 different rules for achieving elasticity and dynamic provisioning for the database tier in the cloud. Two rules are defined based on the average CPU utilization of allocated virtual machines for the database server as follows: Scale out the database tier (add one more replica) when the average CPU utilization of the virtual machines exceeds of 75% for (**R1**) and 85% (**R2**) for a continuous period of 5 minutes. Two other rules are defined based on the percentage of the SLA satisfaction of the workload transactions (the SLA values of the different transactions are defined as specified in the Cloudstone benchmark) as follows: Scale out the database tier when the percentage of SLA satisfaction is less than 97% for (**R3**) and 90% (**R4**) for a continuous period of 5 minutes. Our evaluation metrics are the overall percentage of SLA satisfaction and the number of provisioned database replicas during the experimental time.

Figure 3 illustrates the results of running our experiments over a period of one hour for the 80/20 workload (Figure 3(a)) and the 50/50 workload (Figure 3(b)). In these figures, the *X-axis* represents the elapsed time of the experiment while the *Y-axis* represents the SLA salification of the application workload according to the different elasticity rules. In general, we see that, even for this relatively small deployment, SLA-based can show improved overall SLA satisfaction of different workloads of the application. The results show that the SLA-based rules (**R3** and **R4**) are (by design) more sensitive for achieving the SLA satisfaction and thus they react earlier

Workload / Rule	R1	R2	R3	R4
80/20	4	3	5	5
50/50	5	4	7	6

TABLE I
NUMBER OF PROVISIONED DATABASE REPLICAS

than the resource-based rules. The resource-based rules (**R1** and **R2**) can accept longer time with SLA violations before taking a necessary action (CPU utilization reaches the defined limit). The benefits of SLA-based rules become clear with the workload increase (increasing the number of users during the experiment time). The gap between the SLA-based rules and SLA-based rules is smaller for the workload with the higher write ration (50/50) due to the higher contention of CPU resources for the write operations and thus the conditions of the resource-based rules can be satisfied earlier.

Table I shows the total number of provisioned database replicas using the different elasticity rules for the two different workloads. Clearly, while the SLA-based rules achieves better SLA satisfaction, they may also provision more database replicas. This trade-off shows that there is no clear winner between the two approaches and we can not favor one approach over the other. However, the declarative SLA-based approach empowers the cloud consumer with a more convenient and flexible mechanisms for controlling and achieving their policies in such dynamic environment of the Cloud.

VI. Related Work

Several approaches have been proposed for dynamic provisioning of computing resources based on their effective utilization. These approaches are mainly geared towards the perspective of cloud providers. Wood et. al. [27] have presented an approach for dynamic provisioning of virtual machines. They define a unique metric based on the consumption data of the three physical computing resources: CPU, network and memory to make the provisioning decision. Cunha et. al. [28] designed a queuing model to model the provisioning of virtual servers. They assign each class of jobs in an application onto a virtual machine. In addition, they introduce a pricing model which gives rewards for throughput to be within SLA limits and penalty for throughput going above. This approach would turn to be not cost-effective in a situation where an application has numerous classes. Padala et.al. [29] carry out black box profiling of the applications and build an approximated model which relates performance attributes such as response time to the fraction of processor allocated to the virtual machine running the application.

Dolly [11] is a virtual machine cloning technique to spawn database replicas and provisioning shared-nothing replicated databases in the cloud. It proposes database provisioning cost models to adapt the provisioning policy to the cloud infrastructure specifics and application requirements. Rogers et al. [30] proposed two approaches for managing the resource provisioning challenge for cloud databases. The Black-box provisioning uses end-to-end performance results of sample query executions, whereas white-box provisioning uses a finer grained approach that relies on the DBMS optimizer to predict the physical resource (i.e., I/O, memory, CPU) consumption for each query. Kingfisher [31] is a cost-aware system that provides support for elasticity in the cloud by reducing the time to transition to new configurations and optimizing the selection of a virtual server configuration that minimizes the cost. Floratou et al. [32] have studied the performance and cost in the relational database as a service environments. The results show that given the range of the pricing models and the flexibility of the allocation of resources in cloud-based environments, it is hard for a user to figure out their actual monthly cost upfront. Soror et al. [10] introduced a virtualization design advisor that uses information about the anticipated database workloads to recommend workload-specific virtual machines configurations offline.

VII. Conclusion

In this paper, we introduced the notion and challenges of SLA-based provisioning and cost management for cloud-hosted databases from the consumer perspective. We presented an end-to-end framework that facilitates adaptive and dynamic provisioning of the database tier of the software applications based on application-defined policies for satisfying their own SLA performance requirements, avoiding the cost of any SLA violation and controlling the monetary cost of the allocated computing resources. Our experimental results show that our approach provides the cloud consumers with adequate declarative and more flexible mechanism for controlling the SLA of their applications than relying on monitoring the utilization of the allocated cloud computing resources.

References

[1] M. Armbrust, A. Fox, G. Rean, A. Joseph, R. Katz, A. Konwinski, L. Gunho, P. David, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Dept., University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[2] B. Cooper, E. Baldeschwieler, R. Fonseca, J. Kistler, P. Narayan, C. Neerdaels, T. Negrin, R. Ramakrishnan, A. Silberstein, U. Srivastava, and R. Stata, "Building a cloud for yahoo!" *IEEE Data Eng. Bull.*, vol. 32, no. 1, 2009.

[3] B. Suleiman, S. Sakr, R. Jeffrey, and A. Liu, "On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure," *Internet Services and Applications*, 2012.

[4] S. Sakr, A. Liu, D. M. Batista, and M. Alomari, "A survey of large scale data management approaches in cloud environments," *IEEE Communications Surveys and Tutorials*, vol. 13, no. 3, 2011.

[5] W. Vogels, "Eventually consistent," *Queue*, vol. 6, October 2008. [Online]. Available: http://doi.acm.org/10.1145/1466443.1466448

[6] H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu, "Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective," in *CIDR*, 2011.

[7] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," *IEEE Data Eng. Bull.*, vol. 32, no. 1, 2009.

[8] D. Agrawal, A. E. Abbadi, F. Emekçi, and A. Metwally, "Database Management as a Service: Challenges and Opportunities," in *ICDE*, 2009.

[9] A. Aboulnaga, C. Amza, and K. Salem, "Virtualization and databases: state of the art and research challenges," in *EDBT*, 2008.

[10] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath, "Automatic virtual machine configuration for database workloads," in *SIGMOD Conference*, 2008.

[11] E. Cecchet, R. Singh, U. Sharma, and P. J. Shenoy, "Dolly: virtualization-driven database provisioning for the cloud," in *VEE*, 2011.

[12] S. Sakr, L. Zhao, H. Wada, and A. Liu, "CloudDB AutoAdmin: Towards a Truly Elastic Cloud-Based Data Store," in *ICWS*, 2011.

[13] S. Sivathanu, L. Liu, Y. Mei, and X. Pu, "Storage management in virtualized cloud environment," in *IEEE CLOUD*, 2010.

[14] B. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *SoCC*, 2010.

[15] J. Schad, J. Dittrich, and J. Quiané-Ruiz, "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance," *PVLDB*, vol. 3, no. 1, 2010.

[16] D. Durkee, "Why cloud computing will never be free," *Commun. ACM*, vol. 53, no. 5, 2010.

[17] P. Bodík, A. Fox, M. Franklin, M. Jordan, and D. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *SoCC*, 2010.

[18] B. Kemme, R. Jiménez-Peris, and M. Patiño-Martínez, *Database Replication*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

[19] C. Plattner and G. Alonso, "Ganymed: Scalable Replication for Transactional Web Applications," in *Middleware*, 2004.

[20] A. Elmore, S. Das, D. Agrawal, and A. Abbadi, "Zephyr: live migration in shared nothing databases for elastic cloud platforms," in *SIGMOD*, 2011.

[21] Y. Wu and M. Zhao, "Performance modeling of virtual machine live migration," in *IEEE CLOUD*, 2011.

[22] D. Florescu and D. Kossmann, "Rethinking cost and performance of database systems," *SIGMOD Record*, vol. 38, no. 1, 2009.

[23] L. Zhao, S. Sakr, and A. Liu, "Application-managed replication controller for cloud-hosted databases," in *IEEE CLOUD*, 2012.

[24] L. Zhao, S. Sakr, A. Fekete, H. Wada, and A. L. and, "Application-managed database replication on virtualized cloud environments," in *Data Management in the Cloud (DMC), ICDE Workshops*, 2012.

[25] L. C. Voicu, H. Schuldt, Y. Breitbart, and H.-J. Schek, "Flexible data access in a cloud based on freshness requirements," in *IEEE CLOUD*, 2010.

[26] W. Sobel, S. Subramanyam, A. Sucharitakul, J. Nguyen, H. Wong, S. Patil, A. Fox, and D. Patterson, "Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0," in *CCA*, 2008.

[27] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *NSDI*, 2007.

[28] S. Cunha, J. M. Almeida, V. Almeida, and M. Santos, "Self-adaptive capacity management for multi-tier virtualized environments," in *Integrated Network Management*, 2007.

[29] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *EuroSys*, 2007.

[30] J. Rogers, O. Papaemmanouil, and U. Çetintemel, "A generic auto-provisioning framework for cloud databases," in *ICDE Workshops*, 2010.

[31] U. Sharma, P. J. Shenoy, S. Sahu, and A. Shaikh, "A cost-aware elasticity provisioning system for the cloud," in *ICDCS*, 2011.

[32] A. Floratou, J. M. Patel, W. Lang, and A. Halverson, "When free is not really free: What does it cost to run a database workload in the cloud?" in *TPCTC*, 2011.