Consumer-Centric SLA Manager for Cloud-Hosted Databases

Liang Zhao Sherif Sakr Anna Liu
National ICT Australia (NICTA)
University of New South Wales
Sydney, Australia
{firstname.lastname}@nicta.com.au

ABSTRACT

We present an end-to-end framework for consumer-centric SLA management of virtualized database servers. The framework 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. In this framework, the SLA of the consumer applications are declaratively defined in terms of *goals* which are subjected to a number of constraints that are specific to the application requirements. The framework continuously monitors the application-defined SLA and automatically triggers the execution of necessary corrective actions (scaling out/in the database tier) when required. The framework is database platform-agnostic, uses virtualization-based database replication mechanisms and requires zero source code changes of the cloud-hosted application.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Cloud Database, Service Level Agreement (SLA).

1. INTRODUCTION

Virtualization is a key technology of the cloud computing paradigm which is 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. One approach for deploying data-intensive applications in cloud platforms is the *virtualized database server* approach [2, 12] where an existing database tier of a software application that has been designed to be used in a conventional data center can be directly ported to virtual machines in the public cloud. In this approach, database servers, like any other software components, are migrated to run in virtual machines. One of the main advantages of this approach is that the application can have full control in dynamically allocating and configuring the physical resources of the database tier (database servers) as required. 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, in order to achieve this goal, this approach requires the existence of an *admission control* component which is responsible for monitoring the system state and dynamically allocating the computing resources for the database tier.

Service Level Agreements (SLAs) capture the agreed upon guarantees between a service provider and its customer. It indicates the level of service agreed upon as well as the associated cost if the service provider fails to deliver the level of service. In general, cloud computing technology faces challenges on providing certainty to the commodities exchanged among buyers and sellers. Several studies have reported that the variation of the performance of cloud computing resources is high [5]. Currently cloud infrastructure providers (e.g., virtual machines) - and similarly providers of cloud database services such as Amazon SimpleDB, Amazon RDS and SQL Azure - do not provide adequate SLAs for their service offerings. Particularly, most providers guarantee only the availability (but not the performance) of their services [3]. Often, cloud-hosted applications need to meet SLAs that involve maximum response times of their transactions. Further, they may need to specify SLAs for data freshness guarantees of workloads that involve intensive write operations. 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 tenant applications that can be simultaneously hosted and running in a cloud environment. This puts a burden on the cloud consumers as it is their responsibility to assure their *application* service levels to their end users [8].

We present an end-to-end framework for a *consumer-centric* SLA management for cloud-hosted databases in a virtualized environment. The framework enables the consumer applications to declaratively define their SLA in terms of *goals* which are subjected to a number of constraints that are specific to their application requirements. The framework also enables the consumer applications to declaratively define a set of application-specific rules (action rules) where the admission controller of the database tier needs to automatically trigger corrective actions in order to meet the expected system goals when necessary. Therefore, 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, uses virtualization-based database

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.
ACM 978-1-4503-2263-8/13/10.
http://dx.doi.org/10.1145/2505515.2508196.

replication mechanism and requires zero changes in the line codes of the cloud-hosted software applications.

2. SLA MANAGEMENT IN CLOUD

Although software applications which are deployed on a public cloud infrastructure can benefit highly from on-demand access to various computing resources at low prices, its service levels are highly likely to become more uncertain. In practice, traditional cloud monitoring technologies focus on low-level computing resources (e.g. 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 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. Therefore, it becomes a significant issue for the cloud consumer to monitor and adjust the deployment of their systems if they intend to offer viable service level agreements (SLAs) of a higher level of abstraction (e.g. business transaction or data freshness) to their end users. In principle, meeting SLAs which are agreed with end-users by consumer applications of cloud resources using the traditional techniques for resource provisioning is a very challenging task due to many reasons such as:

- *Highly dynamic workload*: An application service can be used by large numbers of end-users and highly variable load spikes in demand can occur depending on the day and the time of year, and the popularity of the application.
- *Performance variability of cloud resources*: Several studies have reported the high variation on the performance of cloud computing resources [5].
- *Uncertain behavior*: 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 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 infrastructure service provider. In order to tackle this challenge, *our framework is designed to support cloud-hosted applications to achieve their defined SLA with a main focus on the database tier.*

3. SYSTEM DESCRIPTION

3.1 Framework Architecture

Figure 1 shows an overview of our framework architecture which consists of three main modules: the *monitor module*, the *control module* and the *action module* [12]. In this architecture, the monitor module is responsible for continuously tracking the application-defined SLAs and feeding the control module with the collected information. The control module is responsible for continuously checking the monitored SLA values against their associated application-defined SLAs and triggers the action module to scale out/in the database according to the application-defined action rules. The design principles of our framework architecture are to be *application-independent* and to require *no code modification* on the consumer software applications that the framework will support. To achieve these goals, we rely on a database proxying mechanism which provides the ability to forward database requests to the underlying databases 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.

When the application load increases and the database tier becomes the *bottleneck* in the stack of the software application, the scale out approach has shown to be the best practice in the cloud environment as it is extensible, economical and fits well with the *pay-as-you-go* pricing philosophy of cloud computing. Therefore, our framework relies on the scaling out strategy to achieve the elasticity and scalability goals of the database tier of cloud-hosted applications. 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 designed to employ asynchronous master-slave replication strategy among the database replicas and uses the ROWA (read-once-write-all) protocol on the Master copy [7].

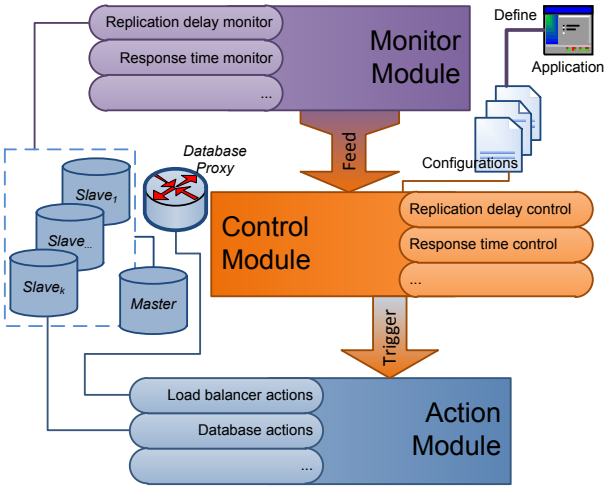

Figure 1: Framework Architecture

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. There is, however, 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 [4].

In general, there exist many forms of SLAs with different metrics. In this demonstration, we focus on the following two main consumer-centric SLA metrics:

- *Transaction response time*: represents the time between a transaction is presented to the database system and the time when the transaction execution is completed [8].
- *Data freshness*: represents the tolerated window of data staleness for each database replica. In other words, it represents the time between a committed update operation on the master database and the time when the operation is propagated and committed to the replica [10, 11].

3.2 SLA of Transaction Response Times

In practice, the SLA requirements can vary between the different types of application transactions. The variation in the SLA of different applications transactions is due to their different natures and consumption behaviour of system resources (e.g. disk I/O, CPU processing). Our framework enables the consumer application to

define an associated maximum response time SLA(S_i) for each transaction type (T_i) so that when an instance of the transaction type is presented to the system at time 0, the system is considered to be achieving its target if it finishes the execution of the transaction at time t where ($t \leq S_i$). Otherwise, if ($t > S_i$) then the instance of the transaction is considered to be violating its defined SLA and hence an instance of SLA violation is recorded. For example, a *user login* application request can define SLA of 100 ms as its maxim response time, a *search* request can define SLA of 600 ms while a request of *submitting an order* would define 1500 ms as its SLA. To achieve this goal, each application transaction is described, using an XML dialect, as a pattern of SQL commands where the transaction response time is computed as the total execution time of these individual operations in the described pattern. The monitoring module uses the database proxy to correlate the received database operations based on their sender in order to detect the defined transaction patterns and feed the controller module with the computed value of the SLA metrics of the transaction instances.

3.3 SLA of Data Freshness

The **CAP** theorem [1] shows that a shared-data system can only choose at most two out of three properties: ***C***onsistency, ***A***vailability, and tolerance to ***P***artitions. Most of the cloud data management systems implement various forms of weaker consistency models (e.g. eventual consistency [9]) so that all replicas do not have to agree on the same value of a data item at every moment of time. In practice, the eventual consistency model has been accepted by many of the new generation of Web 2.0 applications (e.g. social networks) which could be more tolerant with a wider window of *data staleness* (replication delay). However, these applications need to be supported by mechanisms that enable them to manage the extent to which inconsistencies can be tolerated. Our framework enables the consumer applications to define the tolerated SLA replication delay (S_i), in terms of time units, of each database replica (R_i) which is hosted in the data center of the geographic location (L_i). The monitor module tracks the replication delay of each replica by continuously measuring the time difference of each two associated local timestamps committed on the master database and the target replica [11].

3.4 Corrective Actions (Action Module)

Clearly, optimizing the application-defined SLA while minimizing the cost of used computing resources is vital goal for the consumer applications. However, the application workload demands can vary significantly. Thus, the application has to allocate sufficient resources to support the user workloads or risk the consequences of SLA violations in the form of lost revenue or damaged reputation. In particular, the application will have to *scale out* the database tier (adding more replicas) when it deems that existing database replicas are insufficient to accommodate the incoming workload. Meanwhile, the application will have to *scale in* the database tier (removing existing replicas) if he can meet decreasing application workload with a less number of replicas. Our framework enables the application to *declaratively* define, using another XML dialect [11], application-specific *action rules* to adaptively scale out or scale in according to the status of the application-defined SLAs. Examples of these *corrective actions* are:

- *Scale out* the database tier if the average percentage of SLA violation for transactions T_1 exceeds 10% (of the total number of T_1 transactions) for a continuous period of 8 minutes.
- *Scale in* the database tier if the percentage of SLA violation for transactions T_1 is less than 2% for a period of 8 minutes and the average number of concurrent users per database replica is less than 25.

4. DEMONSTRATION SETUP/SCENARIOS

Our demonstration will use the scenario of the Berkeley Cloudstone benchmark [6] which mimics a Web 2.0 social events calendar and follows the canonical three-tier Web application architecture: a web server, an application server and a database server (MySQL database server). In principle, the Cloudstone benchmark is more suitable for the purpose of our demonstration rather than other benchmarks such as *TPC-W*or *RUBiS* because Web 2.0 applications are more acceptable to data staleness. For example, it is not a mission-critical goal for a social network application to immediately have a user's new status or comments available to his friends. However, a consistency window of some seconds (or even minutes) would be still acceptable. We deploy the application tiers on Amazon EC2 where each tier is hosted on a virtual machine. On the database tier, we used a MySQL proxy software[1] to implement our monitoring layer. The database replicas of our demonstration will be hosted in different Amazon EC2 regions (e.g. *us-west*, *us-east-1e* and *eu-west*). The demonstration will highlight the aspects of consumer-centric management for the SLA of data-intensive applications. We will present the following two scenarios:

- *Scenario 1: Provisioning the database tier based on application-defined SLA for the response time of its transactions*. In this scenario, the users will be able to edit the SLA (response time) for different transaction types of the application (e.g. user login, creating an event, attending an event). The users will also be able to edit the parameters of the workload generator (e.g. number of users, transaction rate) and define declarative scaling out/in rules for the underlying database tier of the application. During the running phase, the user will be able to monitor the percentage of SLA violations of each transaction type during the timeline. The user will also be able to monitor the triggering of the dynamic provisioning activities of the database tier when the conditions of their defined action rules are satisfied.
- *Scenario 2: Provisioning the database tier based on the SLA of data freshness for each replica*. In this scenario, the user will be able to define the SLA of the tolerated data staleness of each replica. During the running phase, the user will be able to monitor the replication delay of each replica and the trigger of adding new replicas when the threshold of any replica exceed its defined SLA of data staleness.

5. REFERENCES

[1] E. Brewer. Towards robust distributed systems. In *PODC*, 2000.
[2] E. Cecchet, R. Singh, U. Sharma, and P. Shenoy. Dolly: virtualization-driven database provisioning for the cloud. In *VEE*, 2011.
[3] D. Durkee. Why cloud computing will never be free. *CACM*, 2010.
[4] A. Elmore et al. Zephyr: live migration in shared nothing databases for elastic cloud platforms. In *SIGMOD*, 2011.
[5] J. Schad et al. Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance. *PVLDB*, 3(1), 2010.
[6] W. Sobel et al. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *CCA*, 2008.
[7] C. Plattner and G. Alonso. Ganymed: Scalable Replication for Transactional Web Applications. In *Middleware*, 2004.
[8] S. Sakr and A. Liu. SLA-Based and Consumer-Centric Dynamic Provisioning for Cloud Databases. In *IEEE CLOUD*, 2012.
[9] W. Vogels. Eventually consistent. *CACM*, 52(1), 2009.
[10] L. Zhao, S. Sakr, A. Fekete, H. Wada, and A. Liu. Application-Managed Database Replication on Virtualized Cloud Environments. In *Data Management in the Cloud (DMC), ICDE Workshops*, 2012.
[11] L. Zhao, S. Sakr, and A. Liu. Application-Managed Replication Controller for Cloud-Hosted Databases. In *IEEE CLOUD*, 2012.
[12] L. Zhao, S. Sakr, and A. Liu. A Framework for Consumer-Centric SLA Management of Cloud-Hosted Databases. In *IEEE Transactions on Service Computing (TSC)*, 2013.

[1] http://dev.mysql.com/downloads/mysql-proxy/