CloudDB AutoAdmin: Towards a Truly Elastic Cloud-Based Data Store

Sherif Sakr, Liang Zhao, Hiroshi Wada, Anna Liu
NICTA and University of New South Wales, Sydney, Australia
first.last@nicta.com.au

1 Introduction

Database-as-a-service (DaaS) is a new paradigm for data management in which a third party service provider hosts a database as a service [2]. The service provides data management for its customers and thus alleviates the need for the service user to purchase expensive hardware and software, deal with software upgrades and hire professionals for administrative and maintenance tasks. Since using an external database service promises reliable data storage at a low cost, it represents a very attractive solution for companies specially startups. For example, Amazon Relational Database Service (RDS)and Microsoft SQL Azure database system have been recently announced as cloud-based relational database systems. The service level agreements (SLA) of cloud database services are mainly focusing on providing their customers with *high availability* (99.99%) to the hosted databases. However, they are not providing any guarantee or support on the performance and scalability aspects. Therefore, it is on the shoulder of the consumer applications (developers) to take care of additional responsibilities and challenges to achieve the SLA requirements of their applications in an efficient and economical way.

In this paper, we present the design and the architecture of the CloudDB AutoAdmin system which aims to fill the existing gaps between the provided cloud database services and the requirements of the consumer applications. In particular, it focuses on facilitating the job of the cloud database consumers in implementing database applications as *distributed*, *scalable*, and *elastic* services with a minimum effort on the side of the application developer and a limited footprint in the application code.

2 System Goals

Data replication and *data partitioning* are two well-known strategies to achieve the *availability*, *scalability* and *performance improvement* goals in the distributed and large scale data management world [5]. In particular, when the application load increases, there are two main options for achieving scalability at the database tier and make the application able to cope with more client requests: 1) *Scaling up*: aims at allocating a bigger machine to act as a database server. 2) *Scaling out*: aims at *replicating* and *partitioning* data across more machines. 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 storage space or buying another commodity server. Therefore, commercial cloud providers are generally relying on the scaling out model in their services as it fits well with the *pay-as-you-go* pricing philosophy.

In practice, while cloud providers benefit from economies of scale and multiplexing gains afforded by sharing of resources through virtualization mechanisms, maximizing these advantages on the *consumer* side requires a control framework that can orchestrate an automated cloud resources provisioning and configuration process. Therefore, the CloudDB AutoAdmin is designed to act as a *middleware component* that resides between the consumer application and the cloud database service in order to facilitate *adaptive* and *dynamic* configuration of the application data management layer. Thus, it enables *automated* and *transparent* achievement for the application SLA requirements with an efficient resource consumption. Particularly, CloudDB AutoAdmin has the following goals to achieve:

1) *Declarative specification of replication management strategies*: aims to allow developers to specify declarative rules to *adaptively* scale out (adding more replicas) or scale in (removing existing replicas) the database tier in order to meet an application-defined SLA requirements. Therefore, the application developer will be able to achieve *automated elasticity* feature for his database tier which will be also augmented with a load balancing mechanism between the immediately available database replicas at anytime.

2) *Declarative specification of data partitioning and redistribution*: aims to allow developers to define declarative rules to *adaptively* split a specific partition, merge different partitions or moving one partition from a specific location to another in order to meet the requirements of any dynamic workloads or spike situations [3] or to improve the response times for geographically distributed users.

3) *Declarative specification of consistency requirements*: The CAP theorem [1] shows that a distributed data system can only choose two out of three properties: *Consistency*, *Availability* and *Tolerance to Partitions*. It is highly important for cloud-based applications to be always available. Thus, the consistency requirement is typically compromised and various forms of weaker consistency (e.g. *eventual consistency* [6]) are usually applied. In practice, high consistency implies high cost per transaction and reduced availability but avoids penalty costs while low consistency leads to lower costs per operation but might result in higher penalty costs [4]. Therefore, application developers should be able to declaratively specify strict consistency requirements for particular type of transactions while weaker consistency requirements of other types of transactions.

4) *Transparent execution of distributed transactions*. Due to the size limit on a single database (e.g. the maximum size of an SQL Azure database is 50 GB) or geographical distribution of application users, it would be common to run transactions over multiple partitions (databases). Given an existing setting of data partitions and replicas, the CloudDB AutoAdmin need to transparently coordinate the execution of any transaction (in a distributed manner, if required) with oblivious application code and declarative application-defined specifications of the consistency requirement.

3 System Architecture

Figure 1 illustrates the component-architecture of CloudDB AutoAdmin. A brief description of the role of each component is given as follows:

Metadata manager: Stores a metadata information about the different entities such as: the data layout (e.g. number of replicas, location of replicas, partitions), the application-defined SLA requirements, pricing information of required resources. These metadata information is to be utilized by the different components in making their decisions.

Workload monitor: Logs and monitors the executed database operations of the executed workloads.

SLA monitor: Logs and monitors the values of the application-defined SLA metrics (e.g. the processing of any order transaction should not exceed more than 200 ms). The monitoring information must be very fresh and accurate to let the system controllers take the required actions timely and correctly in order avoid any significant disruption.

Performance modeler: Responsible of estimating the performance of executing a specific workload given a specific data management configuration (e.g. data partitioning and replication).

Financial cost modeler: Responsible of estimating the financial cost of a specific workload given the pricing information and the data management configuration.

Replication controller: Regularly track the information provided by the *SLA Monitor* so that it can automatically decide (when required) if new replicas have to be added

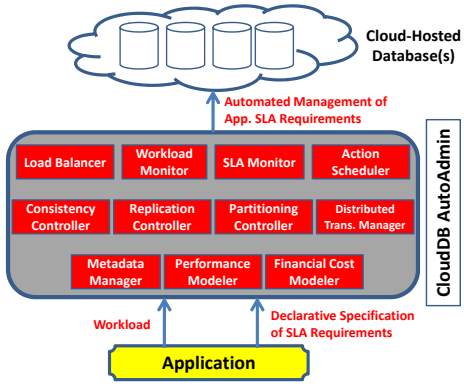

Figure 1. CloudDB AutoAdmin: Component Architecture

to meet degrading SLA satisfaction or existing unnecessary replicas can be removed in order to reduce the cost.

Consistency controller: Transparently executes and switches between different consistency models according to the transaction settings.

Data partitioning controller: Regularly track the information of the *Workload and SLA Monitors* in order to automatically decides if new partitions have to be created to meet the SLA performance requirements of existing *hot spots* or merging existing unnecessary partitions in order to reduce the cost of executing distributed transactions.

Load balancer: Automatically distributes the application workload between the available data replicas and partitions.

Distributed transaction manager: Acts as the coordinator of executing distributed transactions across the available data replicas and partitions.

Action scheduler: Executing all the decision of the different system controllers concurrently might overwhelm the system and reduce performance. Executing the actions sequentially would minimize the performance impact but would be very slow. The action scheduler is responsible of prioritizing and scheduling the concurrent execution of all the required actions in a way that avoids (or minimize) any effect on achieving the application SLA requirements.

References

[1] E. Brewer. Towards robust distributed systems. In *PODC*, 2000.

[2] D. Agrawal et al. Database Management as a Service: Challenges and Opportunities. In *ICDE*, 2009.

[3] P. Bodík et al. Characterizing, modeling, and generating workload spikes for stateful services. In *SoCC*, 2010.

[4] T. Kraska et al. Consistency rationing in the cloud: Pay only when it matters. *PVLDB*, 2(1), 2009.

[5] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Second Edition*. Prentice-Hall, 1999.

[6] W. Vogels. Eventually consistent. *CACM*, 52(1), 2009.