An Approach to Database Replication in the Cloud

Flávio R. C. Sousa 1, Supervised by: Javam C. Machado 1

1 Mestrado e Doutorado em Ciência da Computação (MDCC)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

{sousa,javam}@ufc.br

Level: Ph.D.
Admission: 2008
PhD Qualifying Exam: May 2010
PhD Proposal: July 2011
Conclusion expected to: July 2012
Steps Completed: literature review, problem definition and solution
Future Steps: Complete implementation and evaluation

Abstract. Cloud computing is an extremely successful paradigm of service oriented computing and has revolutionized the way computing infrastructure is abstracted and used. The cloud computing model requires significant changes in data management systems, since cloud systems demand scalability, availability, performance and quality of service. Database replication techniques have been used to address these challenges. However, only few current solutions to database replication in the cloud address elasticity and quality of service, primarily to support a large number of applications, each with a small data footprint. This paper presents RepliC, an approach to database replication in the cloud with quality of service, elasticity and efficient use of resources.

keywords: Cloud Computing, Data Management and Replication.

*RepliC project http://replic.sf.net
1. Introduction and Motivation

Cloud computing is an extremely successful paradigm of service oriented computing and has revolutionized the way computing infrastructure is abstracted and used. Scalability, elasticity, pay-per-use pricing, and economies of scale are the major reasons for the successful and widespread adoption of cloud infrastructures. Since a majority of cloud applications are data driven, database management systems (DBMSs) powering these applications are critical components in the cloud software stack [Elmore et al. 2011]. Data management in the cloud can be divided into two classes of systems: (i) for supporting update heavy applications, and (ii) for ad-hoc analytics and decision support. The class of systems that are designed to support heavy update web applications deployed in the cloud can be divided into two sub-classes: one where the goal of the system is to support a single large application with large amounts of data (scalable single tenant DBMS); and another where the goal of the system is to support a large number of applications each with a small data footprint (large multitenant DBMS with tens of MBs to a few GB) [Agrawal et al. 2010].

Large distributed Key-Value stores - such as Bigtable, Dynamo, PNUTS and Cassandra - are designed to scale to large numbers of concurrent requests using commodity infrastructure of thousands of servers while being elastic and fault-tolerant [Elmore et al. 2011]. Although extremely successful, the Key-Value stores’ simplified data model, lack of transactional support, and lack of attribute based accesses can result in considerable overhead in re-architecting legacy applications which are predominantly based on RDBMS technology. Additionally, an application with smaller storage requirements (tens of MBs to a few GB) would not take advantage of Key-Value stores [Yang et al. 2009]. Recent studies show that RDBMS such as SQL Azure have had good results in different cloud application scenarios [Kossmann et al. 2010].

Database systems serving cloud platforms must serve large numbers of applications or tenants. Multitenancy in databases has been prevalent for hosting multiple tenants within a single DBMS while enabling effective resource sharing. Sharing of resources at different levels of abstraction and distinct isolation levels results in various multitenancy models; the shared machine, shared process, and shared table models are well known. However, dealing with unpredicted load patterns and elasticity is critical to ensure that the tenants’ service level agreements (SLA) are met [Elmore et al. 2011]. Quality of service is defined between the service provider and customer and expressed through a SLA [Xiong et al. 2011], which specifies a level of performance and availability that must be met and penalties in case of failure. Many companies rely on the SLA, for example, to display a web page within a certain time interval. These companies expect that cloud providers ensure quality of service using SLAs based on performance characteristics. However, in general, providers usually base their SLAs only on the availability of services, while cloud services require high performance [Schad et al. 2010].

Database replication techniques have been used to improve availability, performance and scalability in different environments [¨Ozsu and Valduriez 2011]. Solutions for database replication have focused on these aspects of the system with a static number of replicas. Aspects related to dynamic provisioning of capacity, such as adding replicas on-the-fly, have received little attention. This issue is important in cloud environments, where changes of workload require quick initializing of new replicas of databases.
Current solutions to database replication in the cloud try to adapt existing concepts [Yang et al. 2009] [Voicu et al. 2010] [Savinov and Daudjee 2010] [Amazon 2011] [Azure 2011] [Curino et al. 2011] [Xiong et al. 2011]. However, few solutions deal with the database replication in the cloud considering the quality of service, primarily to support a large number of applications, each with a small data footprint.

To solve this problem, this paper presents RepliC, an approach to database replication in the cloud with quality of service, elasticity and efficient resource usage. RepliC uses information from databases and provider’s infrastructure to providing resources as well as implements different levels of consistency. In RepliC, elasticity adjusts the system’s capacity at runtime by adding and removing replicas without service interruption in order to handle the workload variation.

1.1. Contributions

The major contributions of this paper are as follows:

- An approach to database replication in the cloud.
- A strategy for database quality of service in the cloud.
- An implementation of the proposed approach and its architecture.
- A method for evaluating database in the cloud.

1.2. Problem Statement

This work is related to the problem of database replication in the cloud. This problem domain is restricted from a set of characteristics: (i) the Infrastructure as a Service (IaaS) provider offers an infrastructure with virtual machines. Each virtual machine can be used to run many DBMS. The data is persisted in a cloud storage system and, (ii) the customer defines restrictions for the database (e.g. CPU, memory and storage) and an SLA (e.g. response time, throughput, availability). Each DBMS has a complete copy of the database (small data footprint) and uses the relational data model.

Hence, the problem we consider is how to minimize the number of replicas of each database in the cloud and still ensure quality of service.

This paper is organized as follows. Section 2 explains RepliC and the current status of this work. Section 3 describes the proposed evaluation. Section 4 surveys related work and Section 5 concludes the paper. More informations about cloud data management can be obtained from [Sousa et al. 2010] [Sousa et al. 2011a].

2. Our Proposed Approach

RepliC is an approach to database replication in the cloud with quality of service, elasticity and efficient use of resources. The elasticity adjusts the system’s capacity by adding and removing replicas according to current workload. Monitoring techniques are used to check the status of the system and make modifications in the replication strategy to ensure quality of service. RepliC implements multi-tenant database instance to improve the use of resources. RepliC uses the relational data model but implements different levels of replica consistency, since cloud applications may have different consistency requirements.
2.1. Architecture

The architecture of the RepliC is divided into three parts: RepliCController, RepliCCoordinator and RepliCAgent. The RepliCController is the service responsible for receiving requests and directs them to the RepliCCoordinator. The RepliCCoordinator consists of a set of services that address the management of replicas. The RepliCAgent is a component added to each VM and is also responsible for interacting with the VM and the DBMS. Specifically, this agent monitors and interacts with several DBMSs, as well as checks the state of monitored resources. An overview of the architecture of the RepliC is shown in Figure 1.

![RepliC architecture diagram]

Figura 1. RepliC architecture.

Monitoring Service is responsible for managing the information about the state of the VMs and the DBMS collected by the agent. SLA Service manages agreements between customers and service provider. A catalog stores the data collected and the constraints of the resources needed for execution. Balancing Service distributes the requests to the replicas and Provisioning Service defines the resources required to ensure quality of service. Finally, Scheduling Service directs requests to the replicas and keeps a log of the last transactions submitted to the system. These services can be replicated to improve availability.

2.2. Quality of Service for DBMS

RepliC makes use of a profit-oriented strategy to address the quality of service for the database in the cloud [Sousa et al. 2011b]. Profit-oriented approach presents a reliable operation of the systems, since the provider is motivated to provide a high-quality service. Whenever the provider is not able to meet highest performance standards, it is encouraged to continue providing the service until getting the profit. This strategy defines an SLA with different metrics such as response time, throughput, availability and consistency and penalties for SLA failures. For each DBMS, CPU resources, memory and databases sizes, as well as SLA metrics are monitored. In order to check the quality of service, RepliC implements monitoring techniques to make decisions (i.e. add or remove replicas). These techniques capture the workload and avoid unnecessary decisions, improving the overall system performance and optimizing the resource utilization.
2.3. Elastic Database Replication

RepliC implements the elasticity by adding and removing replicas according to workload. To add a new replica, RepliC tries to add replica in a running virtual machine that has available resources. For this, it uses a first-fit algorithm to select the virtual machine. If not possible, a new virtual machine is started and the new replica is added in this machine. With the reduction in workload, RepliC removes replicas of the database, since the quality of service is maintained. The new replicas are added and updated through the data migration.

Data migration involves applying snapshot and logs of missing updates to the new replica to bring it up-to-date. The first step in the data migration is to create a checkpoint in the transactional log and to remove a replica temporarily out of the system to take a snapshot (i.e. database dump) of the database content. As soon as the snapshot has been taken, this replica is resynchronized by replaying the transactions written in the transactional log since the checkpoint and this replica rejoins in the system. This snapshot is persisted in the cloud storage service. In second step, the scheduling service stores commit transaction in the log since the checkpoint. In next step, the snapshot is added in the new replica. Finally, the log persisted in the scheduling service is sent to this new replica. So, the new replica becomes up-to-date. To remove a replica, RepliC selects the replica with the lowest workload and stops sending requests to it. The replica is removed and, if the SLA remains satisfied, other replicas can be removed.

2.4. Consistency and Fault Tolerance

Due to application requirements, RepliC implements strong and weak consistency as different levels of consistency. In this work, our replication strategy [Sousa et al. 2007] was extended to address the following aspects of cloud computing: support for virtualized environments and the relational data model as well as for different levels of replica consistency. This replication strategy employs a group abstraction which prevents a single point of failure. RepliC divides replicas into two groups: a read group, which executes read-only transactions, and an update group, which executes update transactions. With strong consistency, update transactions are performed synchronously in the group update. Thus, they are sent asynchronously to the read group. In weak consistency, all operations are performed asynchronously. Update transactions are performed in update group, and then sent to the read group.

The replica of the update group that received the transaction is called primary replica, and is responsible for verifying conflicts with the other transactions that might be executing locally, sending a multicast with a total ordering property to the other replicas in this group. These replicas are called secondaries replicas to the primary replica that sent the multicast, and perform a certification test, which verifies if a local transaction in the primary is in conflict with any other transaction executing in the secondaries. Modifications in the update group are serialized and sent continuously by the primary to the read group through a multicast with a FIFO ordering property. These modifications are pushed into local queues in each replica in the read group and executed in the same sequence as in the update group. The read group executes two types of transaction: propagation and refresh. Propagation transactions are executed while the replica is idle, when no read or refresh transactions are being executed, with the purpose of carrying out the updates.
Refresh transaction are applied to execute the transactions in the local queue of a read replica.

There are different types of faults (e.g. hardware, networking, storage, operating system and the DBMS instance). IaaS providers primarily apply redundant hardware to address the majority of these failures. RepliC uses an agent to check the DBMS state and its hosting virtual machine in order to detect failures. Data is stored in a cloud storage service (e.g. distributed file system). When a transaction is committed and persisted in the cloud storage service, in case of failure, data can be retrieved through this service. If the primary replica fails, the transaction can be retrieved through the scheduling service, which stores the transactions to be executed. Replicas of the read and update group are allocated logically in a ring network topology. Thus, failures can be detected by the adjacent replicas. In case of failure in a secondary or read replica, transactions are not sent to these replicas, and they are removed from the system. In the future, if this replica has become operational, a process checks the state of the replica to add it. Otherwise, a new secondary or read replica is created.

2.5. Present Stage of Work

The current state is as follows. We have implemented the agent and monitoring, SLA and load balancing services. Now, we are developing the scheduling and provisioning services and making changes in our previous work [Sousa et al. 2007] to address issues of support for virtualized environments, relational data model and data consistency. We are also extending the benchmark proposed by [Kossmann et al. 2010]. In parallel, we are conducting initial tests on different infrastructures.

3. Experimental Evaluation

The evaluation of cloud database services presents significant differences when compared to the current systems that make use of static provisioning [Sousa et al. 2010]. Systems with static provisioning presuppose the existence of fixed configurations of resources and aim to maximize the performance or availability. In the cloud environment, the goal is to minimize or adjust the amount of resources needed to ensure quality of service. The benchmarks in the cloud must address issues of cost, response time, throughput, scalability, consistency and flexibility. There are some benchmarks for evaluating DBMSs in the cloud [Kossmann et al. 2010]. However, the development of standard cloud database services benchmarks is a challenge, because cloud systems have different characteristics (e.g. data model, levels of consistency, query language). For the evaluation of the RepliC, the benchmark proposed by [Kossmann et al. 2010] was extended to add characteristics of elasticity and quality of service in order to enable the execution of the experiments. We are conducting a detailed evaluation to verify the quality of service (e.g. response time, throughput, availability and consistency), elasticity and stability. In this work, the environment used for evaluation is a cluster with OpenNebula\(^1\) and Amazon AWS\(^2\). Each machine runs OS Ubuntu and DBMS MySQL.

---

1http://www.opennebula.com
2http://aws.amazon.com/
4. Related Work

Azure SQL implements the primary copy protocol with strong consistency [Azure 2011]. Each database hosted in the SQL Azure has three replicas: one primary replica and two secondary replicas. All reads and writes go through the primary replica, and any changes are replicated to the secondary replicas asynchronously. SQL Azure also implements high availability, fault tolerance and multi-tenant. However, it does not address performance issues in quality of service and implements traditional replication protocols. [Yang et al. 2009] presents the design and implementation of a data-management platform that can scale to a large number of small applications. This work implements various techniques for database deployment, replication, ACID, migration and SLA management that ensure high throughput and high availability. However, this platform has no elasticity and makes use of two-phase commit protocol, which interferes with the performance. In [Voicu et al. 2010] is proposed Re:FRESHiT, a novel protocol capable of handling the propagation of updates to read-only nodes and the freshness-aware routing of requests in data clouds. Re:FRESHiT’s unique features include the possibility to access up-to-date data as well as data with any freshness level. This protocol combines eager and lazy replication protocols that take into account different levels of freshness, but it does not implement multi-tenant and elasticity.

In [Cecchet et al. 2011] is proposed Dolly, a database provisioning system based on a virtual machine cloning technique to spawn database replicas in the cloud. In Dolly, each database replica runs in a separate virtual machine and Dolly clones the entire virtual machine. This work presents cost models to adapt the provisioning policy to the cloud infrastructure specifics and application requirements. The main objective of this work is provisioning database in virtualized environments. So, Dolly does not address multi-tenant, quality of service and techniques for data replication. [Savinov and Daudjee 2010] presents a study about data replication in a virtualized environment, focusing on provisioning when the master database server is heavily loaded or when it fails. This work implements primary copy protocol but does not address other aspects of database replication in the cloud. Amazon Relational Database Service (RDS) [Amazon 2011] implements the primary copy protocol and works similar to traditional databases.

In [Xiong et al. 2011] is proposed SmartSLA, a cost-aware resource management system. SmartSLA uses machine learning techniques to learn a system performance model through a data-driven approach. The model explicitly captures relationships between the systems resources and database performance. SmartSLA implements the primary copy protocol asynchronously and addresses only the question of the number of replicas needed to maintain the SLA. In addition, SmartSLA needs many input data, since it uses machine learning techniques. [Curino et al. 2011] introduces Relational Cloud, a scalable relational database-as-a-service for cloud computing environment. Relational Cloud focuses on efficient multi-tenancy, elastic scalability and database privacy. This work uses a workload-aware partitioning strategy. Relational Cloud does not detail the strategy of replication and does not address the quality of service.

5. Conclusion and Future Work

In this work, we have presented RepliC, an approach to database replication in the cloud. RepliC intends to improve the quality of service and efficient use of resources through the
elasticity. The elasticity adjusts the system’s capacity by adding and removing replicas according to current workload. Monitoring techniques are used to check the status of the system and make modifications in the replication strategy to ensure quality of service. Due to the characteristics of cloud computing, mechanisms for consistency and fault tolerance are also proposed. As future work, we intend to conduct a study with techniques of autonomic computing to improve the resource management and load balancing. Another important aspect is the development of efficient solutions to perform provisioning.

Acknowledgments. This work is partly supported by Amazon AWS Research Grant.

Referências


