Chapter 1

Introduction

Cloud Computing is a successful paradigm for deploying scalable and highly available web applications. Cloud Computing has become a popular service oriented computing paradigm, because of its features such as scalability, elasticity, pay per usage and so on. With the emerging popularity of the internet, many applications are deployed on the web. The trend is that desktop applications are replaced by web applications. In real life scenarios, web applications are expected to be scalable and consistent. The recent trend is most of these modern applications are shifting to the cloud.

1.1 Motivation and Challenges

Data is at the heart of modern e-commerce web applications. With the increasing popularity of internet many applications are deployed on the web and have faced the challenge of serving thousands of customers. Therefore, scalability of e-commerce web applications has become an important issue. Today modern e-commerce web applications are generating huge amounts of data. The database system plays an important role in managing large amount of data.

These modern web applications use two families of databases for storage:

- Online Transaction Processing (Relational databases) used for online transaction processing
- Online Analytical Processing (NoSQL databases) used for analysis and decision making

In this work, the focus is on deploying Online Transaction Processing Systems (OLTP) in the cloud.
Different applications are relying on OLTP systems. These applications include, banking applications, gaming applications, e-commerce web applications. In this work, the focus is specifically on e-commerce web applications. The relational databases are widely used for OLTP in an enterprise. They provide strong consistency. But they are not scalable in nature. They cannot be scaled out. Relational database management systems are not suitable for deployment in the cloud [22]. Therefore deploying a relational database management system in the cloud is a challenge [37]. Figure 1-1 shows the different layers of a web application.

The client uses the web application with internet through load balancers. Load balancers accept the requests from the multiple clients and direct them to the ap-
propriate web server. Storage of data and query is handled by the database layer. Frequently used requests are cached on multiple servers, that is caching tier. When the number of users is increased, load balancing tier is scaled by adding more servers. Web tier is scaled by adding more servers. Caching tier is also scaled. But the database tier does not scale [22].

Two different approaches are used for scalability

- Scale up [22] uses large, powerful sever mostly used in relational databases. This allows relational databases to offer rich functionality. But this approach is not feasible because of the cost of hardware.

- Scale out [22] is increasing the capacity of the system by adding low cost commodity servers. Thus scale out minimizes the total system cost. This feature is supported by key-value data store for large scale operations. For example, Google’s Bigtable [14], Yahoo!’s PNUTS [49], Amazon’s Dynamo [25] and so on. Data is distributed and span across low cost commodity servers. These data stores [1, 4, 49, 14, 25] are specially designed for scaling out to commodity servers and include all the properties such as scalability, elasticity and availability.

The aim of this work is to accommodate the requirements, including scalability, consistency of two families of databases for storage.

Many platforms, languages and tools are accessible for developing high level web applications to fulfill the business needs. Building scalable web application is a difficult task [55]. The development time required by web application can be minimized using various high level tools that help the programmer to improve the efficiency. These tools include database systems. While developing the web application, if the data store is providing high database functionality such as strong consistency, then the developer can concentrate only on business logic rather than application level consistency. Therefore scalable, elastic data store providing strong consistency can relieve the Application Developer to focus on business logic.

Another challenge for these web applications is to maintain performance, that is throughput and response time. The response time is the time between the submission of request and response and also includes the time to execute application code and retrieval of data from the data store. As the workload is changing seasonably so the data store should also be elastic such that the database servers can grow or shrink efficiently. On demand resource provisioning is effective than static resource
provisioning. Therefore, the cloud data store should be elastic to facilitate on demand resource provisioning.

The focus of this work is to build scalable and elastic data store with transaction processing capabilities. In this work, two different approaches are proposed to achieve this goal:

- First, how NoSQL data stores can be extended for transaction processing and provide better scalability and elasticity with high-level database functionalities. This requires restructuring of application data, based on data access patterns of web applications.

- Second, using workload-aware approach while performing scale-in and scale-out operation.

The major contribution of this work is the design of workload-driven partitioning for building scalable and elastic systems. It follows the data model of these NoSQL data stores [1, 4, 14, 25, 49] which find out the optimized partitions and assign these partitions to available nodes. Figure 1-2 show the dimensions of data management in the cloud and the focus of this work. Data management is classified in transaction processing and analytics. The approach can be partitioned based or without partitioning. Partitioning scheme can be static, dynamic, and partitioning based on data access patterns of web applications. In this work, the focus is on partitioning based on data access patterns of web applications in cloud data stores in transaction processing to achieve scalability by preserving consistency and reliability.

In summary, the aim of this work is to build scalable and elastic data store. The following objectives are identified to achieve this goal.

- Define the problem of workload-driven partitioning based on data access patterns of web applications.

- To develop algorithm of workload-driven partitioning.

- To introduce workload-aware elasticity framework.

- To present workload-aware elasticity algorithm.

- To develop partitioning evaluator.
1.2 Solution Outline

The thesis makes several contributions for building scalable and elastic cloud data store useful for OLTP applications:

- A taxonomy of data management that highlights different approaches and various techniques are presented. This thesis explores notable prior work in the area of scalable transactions in cloud data stores. Comparison of proposed work with the existing work is also presented.

- The architecture of the system is proposed to process scalable transactions on the partitions, which are distributed among a cluster of low cost commodity servers.

- The design of the workload-driven partitioning, which forms the partitions based on data access patterns of web application is introduced. The proposed partitioning scheme uniformly balances the load among all partitions, which in turn increases the throughput of the overall system. Demonstration of how this workload-driven partitioning can be used to limit the transaction to single partition is explained.
• A Mathematical Formulation of workload-driven partitioning is presented. Data partitioning strategy is introduced which finds out all possible combinations of partitions using mutation and selects the partitions with optimized load and association.

• Workload-driven partitioning algorithm for achieving scalability is proposed. The proposed algorithm reorganizes the application data based on data access patterns of web applications. Performance evaluation is conducted through experimentation over data stores such as Amazon SimpleDB and Hadoop HBase. Experimental and analytical results are observed and it clearly shows that the proposed scheme outperforms the well known schema level partitioning in terms of throughput, response time, efficiency.

• Demonstration of detailed experiments that show the effectiveness of workload-driven partitioning scheme in forming partitions that balance the workload among the partitions is explained.

• A metric for finding efficiency of workload-driven partitioning algorithm is introduced. Comparison of static, dynamic and workload-driven partitioning is explained in detail.

• Statistical model and analysis of workload-driven partitioning in Amazon SimpleDB is presented.

• The workload-aware elasticity algorithm is demonstrated by adding or removing the domains of Amazon SimpleDB based on average load. A framework for workload-aware elasticity is proposed. The framework, is introduced to build elastic and scalable data store in the cloud. Experimental results clearly show that throughput increases linearly while performing scale-in and scale-out operation.

• A partitioning evaluator is proposed which finds relationship between throughput, number of users and number of fragments. It shows the impact of these factors on throughput. An analysis of workload-driven partitioning, schema level and graph partitioning is performed. From the analysis, it is proved that these two factors affect the throughput. It also evaluates throughput with the input workload.
1.3 Dissertation Overview

To build scalable and elastic data store with transaction capabilities, following contributions is presented. The first contribution, workload-driven partitioning provides high scalability. The second contribution, workload-aware elasticity used for achieving elastic scalability and the third contribution, partitioning evaluator which finds the relationship between throughput, number of users and number of fragments. In this section, a brief overview of these contributions is presented as follows:

1.3.1 Workload-Driven Data Partitioning in Cloud Data Stores

Database systems are difficult to scale out. Maintaining consistency is challenging task while scaling out data across different servers. This problem is solved using a technique called database partitioning. Database partitioning is a commonly used technique for scaling out. In the first contribution of this thesis, workload-driven data partitioning for building a scalable data management for web applications is introduced. The mathematical formulation of workload-driven data partitioning is described. Workload-driven data partitioning is the first technique which is based on data access patterns of web applications. Data partitioning strategy, which uses mutation for generating various combinations of partitions is proposed. Workload-driven data partitioning algorithm, which will restructure the application data based on data access patterns of web applications is developed. The proposed technique can be applied to any cloud data store with some modifications. An experimental evaluation using two popular data stores, that is Amazon SimpleDB and Hadoop HBase is presented. Analytical and experimental results are observed and it shows that workload-driven data partitioning outperforms the schema level partitioning in terms of throughput, response time, distributed transactions. The detailed implementation of workload-driven data partitioning is provided in chapter 3.

1.3.2 Workload-Aware Elasticity in Cloud Data Store

Due to varying and seasonable workload on web applications, an important goal of database system is to provide elastic scale-out. Therefore, it is important for a database system to add and remove database servers based on the concurrent number of users and load. The workload-aware elasticity framework for building elastic and scalable cloud data store is presented. The workload-aware elasticity algorithm is developed. It not only adds and removes domains of Amazon SimpleDB, but also
redistributes the partitions based on data access patterns of web applications. An implementation of elastic scale-out is performed using Amazon SimpleDB. The notion is to start with a minimum number of domains in the cluster, and elastic scale out by dynamically adding or removing the domains and redistributing partitions using workload-driven partitioning. An experimental results are observed and it clearly shows that workload-awareness with scale-in and scale-out performs better. The detailed implementation is provided in chapter 4.

1.3.3 Partitioning Evaluator

With the increasing popularity of e-commerce applications, many update-intensive commercial workloads are emerging in recent time. The commercial workloads which are used by these e-commerce applications generate high volumes of update intensive transactions than read transactions. Scale out operation is performed to handle this increasing high volume update-intensive transactions. *Data partitioning* is a commonly used technique for performing scale out operation. Scalability of data store is relying on the underlying data partitioning scheme used by the data store. Therefore, there is need to evaluate the partitioning scheme. Development of partitioning evaluator is a requirement to evaluate a partitioning algorithm. Partitioning evaluator evaluates the throughput of schema level, graph, workload-driven partitioning algorithm with given input workload, as number of fragments and number of concurrent users. Partitioning evaluator finds relationship between throughput, number of users and number of fragments. It is observed that it shows the impact of these two factors on throughput. Analysis of workload-driven, graph and schema level partitioning is performed and these two parameters influence the throughput in all three partitioning algorithms is proved. The detailed implementation is provided in chapter 5.

1.4 NoSQL Storage Systems

To overcome the challenges experienced by relational database management system, NoSQL storage systems [1, 4, 14, 25, 49] have been evolved. They provide scalability, elasticity, availability, but offer eventual consistency. But there is shortcoming associated with these types of systems that the, applications developed on top of these systems are generally complex. The task of maintaining consistency needs to be handled by the Application Developer.
1.4.1 Amazon SimpleDB

Amazon SimpleDB [4] is a document database and provides API for storage and retrieval of data. Amazon SimpleDB is simple to use. Amazon SimpleDB API allows for fast insertion, retrieval and updation of data. The data model of Amazon SimpleDB is very flexible. It is scalable in nature. Domains can be added or removed as the load increases or decreases. It also provides efficient storage and fast retrieval of data for high performance web applications. Amazon SimpleDB service is running in Amazon’s highly available data centers. Therefore, it is reliable. Backup copy of data is stored across multiple servers and data centers. Amazon SimpleDB provides following API’s.

- **CreateDomain**: This command is used to create domain in Amazon SimpleDB.
- **DeleteDomain**: It is used to delete the domain.
- **ListDomains**: It is used to list all domains.
- **PutAttributes**: This command is used to insert, update, and delete operations on data in Amazon SimpleDB domains.
- **BatchPutAttributes**: This command is used to perform multiple put operations at a time.
- **DeleteAttributes**: It is used to remove items, attributes, or attribute values from the domain.
- **BatchDeleteAttributes**: It performs multiple delete operations at a time.
- **GetAttributes**: It retrieves the attribute and value of the specified item id.
- **Select**: It retrieves the values from the domain using a SQL SELECT expression.
- **DomainMetadata**: It displays the information about the domain, such as the creation date, number of items, attributes and size of attributes and its values.

1.4.2 Hadoop HBase

Apache Hadoop HBase [1] is NoSQL database and runs on top of Hadoop Distributed File System (HDFS). It is a column oriented database which provides random access to data and also offers fast record lookup. In Hadoop HBase [1] data is organized into tables, rows and columns.
1.4.2.1 Data Model

The data model of HBase consists of different logical components such as tables, column families, columns, cells, and versions. The data model of Hadoop HBase makes convenient to partition the data and spread it across the different nodes.

- Table:- The HBase table consists of logical rows stored in separate partitions called Regions. Each Region is stored and processed by one Region Server.

- Rows:- Every row is identified using a unique key called as a row key. Row key is used for fast record lookup. The row key is considered as an array of bytes.

- Column Families:- In Hadoop HBase, every record is divided into column families. Each column family contains one or more related columns.

- Columns:- A column family consists of one or more column. It is identified by column qualifier and contains column family name connected with the column name.

- Cell:- A cell contains data called as value. The data type of value is always an array of bytes. It is a unique combination of row key, column family and column.

- Version:- The latest version of data is identified using time stamp. The number of versions of data stored in column family can be configured and by default this value is 3.

1.4.2.2 Hadoop HBase API

- Create Table:- It is used to create a new table. There is need to specify the table name and the name of column family.

- List:- This command is used to list information about the table.

- Put:- It is used to put data into the table.

- Scan:- It is used to scan the table to view data.

- Get:- This command is used to get a data of a particular row.

- Disable:- Before deleting a table, there is need to disable the table.

- Drop:- This command is used to drop a table.
1.5 Organization of the Thesis

This thesis is structured as follows.

- Chapter 2 presents a systematic literature survey for building scalable and elastic database system. Comparison of various techniques of scalable and elastic data management with the proposed work is given in detail. It also presents a taxonomy of data management in the cloud.

- Chapter 3 presents design of workload-driven data partitioning for building scalable web applications. This chapter also describes data partitioning strategy and mathematical formulation of workload-driven partitioning. It also presents workload-driven partitioning algorithm which restructures the application data based on data access patterns of web applications.

- Chapter 4 presents framework for building scalable and elastic data store. This workload-aware elasticity framework adds or remove database domain in Amazon SimpleDB depending on the number of requests. It also presents workload-aware elasticity algorithm which adds or removes domain by analyzing the average load on the domain.

- Chapter 5 presents partitioning evaluator for evaluating the throughput of schema level, graph, workload-driven partitioning algorithm with given input workload, as the number of fragments and number of concurrent users. It also finds the relationship between throughput, number of users and number of fragments.

- Finally, Chapter 6 concludes the thesis and suggests future directions.