Chapter 2

Related Work

In this chapter, research work in the area of scalable and elastic data stores is analyzed. Researchers have proposed a variety of systems and partitioning techniques to provide scalable transaction support for web applications. To offer reasonable and consistent performance for update-intensive workload, data store should be scalable. To achieve scalability for update-intensive queries, it is essential to partition the data. Data partitioning is performed by splitting the data into smaller multiple fragments and spanning them across different machines. This chapter surveys the research work in the area of scalable and elastic data management. This chapter is organized as follows: Section 2.1 presents a taxonomy of data management in the cloud and surveys existing work on scalable transactions in cloud data stores. Section 2.2, briefly addresses related work on elastic scale out in cloud data stores. Section 2.3, discusses related work on the partitioning evaluator. Section 2.4 concludes the chapter.

2.1 Scalable Transactions in the Cloud

2.1.1 Taxonomy of Data Management in Cloud

In this section, a taxonomy of data management in the cloud is proposed. In the real world, data is managed in two families of databases according to the requirements of applications: i) Online Transaction Processing ii) Analytics. In this work, the focus is on transaction processing.

In Online Transaction Processing, scalability is achieved with two different approaches. i) Data partitioning ii) Without explicit partitioning.

There are three ways to partition the data i) Static partitioning ii) Dynamic partitioning iii) Partitioning based on data access patterns. In this work, the focus is on
designing partitioning based on data access patterns of web applications. Taxonomy of data management is shown in figure 2-1.

![Taxonomy of Data Management in Cloud](image)

**Figure 2-1: Taxonomy of Data Management in Cloud.**

- **Partitioning based approach:** In this partitioning based approach, scalability is achieved using different data partitioning techniques.
- **Without partitioning:** In this approach, scalability is achieved without using any data partitioning techniques.
- **Static partitioning:** Static partitioning systems [11, 13, 19, 22] are the systems where related data items are collocated at one partition. Once the partitions are formed, those partitions do not change. Therefore, these partitioning systems are termed as static.
- **Dynamic partitioning:** Dynamic partitioning systems [24] are the systems where partitions are formed dynamically and change very frequently.
- **Partitioning based on data access patterns:**
  This is not static or dynamic partitioning scheme. It lays between static and dynamic partitioning scheme. In this partitioning, the transaction logs are monitored and the data access patterns are analyzed.

In this section, a survey of scalable transactions in the cloud is presented. Scalable transactions, broadly classified in two different approaches. Approaches that are using data partitioning where the data items are collocated on a single partition and approaches that do not use data partitioning.
Sudipto Das et al., proposed ElasTras [23], which provides support for scalable transactions in the cloud. It uses schema level partitioning [22] to improve scalability. It also uses a range partitioning. Scalability is accomplished by restricting the execution of a transaction to a single partition.

P. A. Bernstein et al. suggested the Cloud SQL Server [13] is a relational database where scalability is achieved by scaling out to low cost servers. It also uses schema level static partitioning where transactions are enforced to execute on one partition. In Cloud SQL Server, partition is normally a table group, keyless or keyed. For keyed table group, all the tables in the table group have common partitioning key. The row group is a collection of related rows that has common partitioning key.

Curino et al. suggested the Relational Cloud [21] in, which scalability is achieved with the workload-aware approach called as graph partitioning. In graph partitioning, the data items, which are accessed by the transactions are kept on a single partition.

J. Baker et al., presented Megastore [11] in which data is partitioned into a collection of entity groups. An entity group is a collection of related data items, and is put on a single node so that the data items required for execution are accessed from a single node. It is developed to offer transactional consistency for web applications. Megastore provides synchronous replication, but comes at the cost of increased transaction latencies.

As discussed above, four systems use the static partitioning and it is designed with the common objective where the related rows are kept on a single partition. There are some applications such as online games where groups are formed dynamically with time and therefore, Sudipto Das et al., proposed G-Store [24], where multiple keys from group on different node are collected together and formed a new group on a single node. Keys are located on different nodes. Keys in a newly formed group can be part of multiple groups. A single node gains ownership of all keys in a newly formed group for faster multi key access. A newly formed group has a leader key, which is key selected from the members of the group and the remaining members in a group are referred as a follower keys. In G-Store, ownership of the keys in a key group is collocated at a particular node, their collocation is by choice so that scalability is fostered. Node owning the leader key gains exclusive read and writes access to all the followers. So, though the followers are situated on different nodes, distributed synchronization is not required. It uses a key grouping protocol to transfer key ownership safely from the followers to the leader during group formation and from the leader to the followers during group deletion. But the creation of
groups is overhead. Wei et. al, developed the system Cloud TPS [56], which splits the Transaction Manager into any number of Local Transaction Managers (LTMs). Cloud TPS has certain assumptions that the transactions are short, access a small amount of data, and are well identified in advance. Scalability is achieved by distributing the data among the Local Transaction Managers.

2.1.3 Approaches without Data Partitioning

Aguilera et al., presented Sinfonia [8], in, which the transactions are partitioned into sub transactions called as mini transactions. The mini transactions guarantee transactional semantics on only a small set of operations such as atomic and compare, and swap. D. Lomet et al., proposed design Deuteronomy [33] where scalability is achieved by separating transaction and data management. The key feature of Deuteronomy is that data items can be found anywhere in the Cloud. The single Transaction Component is responsible for handling all the requests so it is not suitable for large cloud deployments. Francisco Cruz et. al presented automated workload-aware table splitting algorithm [18] for NoSQL data stores. Their focus is on finding good splitting point, so that the problem of load balancing is solved. A good splitting point divides the region into two new regions with equal load.

2.1.4 Data Partitioning Methods

2.1.4.1 Schema Level Partitioning

The schema level partitioning [24] scheme is a static partitioning scheme designed to improve the scalability of ElasTras [22]. It is derived from the TPC-C schema [26], so it is called as Schema Level Partitioning. The TPC-C schema has a hierarchical tree structure. The schema level, data partitioning is based on the partitioning key. In the schema level, related rows of tables are collocated on a single partition and the distributed transactions are minimized.

2.1.4.2 Graph Partitioning

Graph partitioning [19] is a workload-based static partitioning algorithm. Transaction logs are analyzed and the workload is monitored to partition the database and therefore, it is called as workload-based partitioning. In graph partitioning, the rows, which are accessed in a transaction are kept on one partition to avoid the distributed transactions.
2.1.4.3 History

An arrangement of work in the area of scalable transactions from 2007 to the latest. Aguilera et al. presented without explicit partitioning based approach in 2007. In 2010, Sudipto Das et. al proposed Elastras [22] (static) and G-Store [24] (dynamic) partitioning scheme to achieve scalability. The concept of entity group for static partitioning in Megastore [11] is introduced in 2011. Curino et. al introduced workload-aware graph partitioning technique [19] in Relational Cloud [21] in 2011. Deuteronomy [33] was proposed in 2011 which also uses without explicit partitioning approach for achieving scalability. Wei et. al introduced Cloud TPS [56] in 2011 which uses consistent hashing algorithm for ACID transactions. The concept of keyed and keyless table group for static partitioning was proposed in Cloud SQL Server [13] in 2011. Francisco Cruz et. al developed workload splitter [18] which finds good splitting point that balances the workload among partitions. In 2013, workload-driven partitioning, that is partitioning based on data access patterns of web applications for achieving scalability is presented. Figure 2-2 shows the milestone in scalable transactions history.

Figure 2-2: Milestones in Scalable Transactions History
Table 2.1: Comparison of Proposed Work with the Existing Work.

<table>
<thead>
<tr>
<th></th>
<th>Partitioning Technique</th>
<th>Partitioning Algorithm Present or Not</th>
<th>Partitioning Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud SQL Server</td>
<td>Schema Level</td>
<td>No</td>
<td>Static</td>
</tr>
<tr>
<td>ElasTraS</td>
<td>Schema Level</td>
<td>No</td>
<td>Static</td>
</tr>
<tr>
<td>Megastore</td>
<td>Schema Level</td>
<td>No</td>
<td>Static</td>
</tr>
<tr>
<td>Workload-Aware Splitting for NoSQL</td>
<td>No</td>
<td>Yes</td>
<td>Static</td>
</tr>
<tr>
<td>Relational Cloud</td>
<td>Graph Based</td>
<td>No</td>
<td>Static</td>
</tr>
<tr>
<td>CloudTPS</td>
<td>Consistent hashing</td>
<td>No</td>
<td>Static</td>
</tr>
<tr>
<td>G-Store</td>
<td>Range</td>
<td>No</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Sinfonia</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Deuteronomy</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td><strong>Proposed Work</strong></td>
<td><strong>Workload-Driven</strong></td>
<td><strong>Yes</strong></td>
<td><strong>Based on data access pattern</strong></td>
</tr>
</tbody>
</table>

Table 2.1 shows a comparison of proposed work with the prior work. As seen from the table, existing work either uses static or dynamic partitioning approach for improving scalability. It also uses approaches without partitioning to achieve scalability. In this work, the focus is on partitioning based on data access patterns of web applications.

2.2 Workload-Aware Elasticity in Cloud Data Store

Researchers have carried out a survey of partition placement where data is distributed between a fixed number of nodes. The existing work for improving database scalability is surveyed and, it is realized that, the existing techniques are either based on partitioning or replication. In this work, the focus is on using partitioning for developing an elastically scalable system.

Curino et. al presented the Relational Cloud [21] for fostering scalability. It uses the workload-aware partitioning technique. However, their focus is on improving scalability using partitioning, but not on a workload-aware elastic scale out.

Sudipto Das et. al proposed ElasTras [22], which uses schema level partitioning for increasing scalability. In schema level partitioning all the related tuples are collocated on a single partition. However, the focus is on achieving scalability and not workload-aware elasticity.

Francisco Cruz et. al presented MET [17] framework for workload-aware elasticity.
for NoSQL, which will place the partitions to a node as per the workload access patterns. That is, all the reads, and writes and so on. will be placed on different nodes. This paper also presents different partitioning strategies. TPC-C benchmark and YCSB benchmark are used in this work for performance evaluation.

Marco Serafini et. al implemented Accordion [46], for achieving elastic scalability. It adds and removes servers as per the demands of the users, but does not redistribute the partitions based on data access patterns of web applications.

Dimitrios Tsoumakos et. al developed a framework, TIRAMOLA [52], which takes the help of the Markov Decision Process model for decision making. The decision making process includes whether to add a node or remove a node from a cluster. The decision is made by taking into account the parameters such as throughput, response time and the cost of a virtual machine. In TIRAMOLA, the emphasis is on using the Markov Decision Process (MDP) for automatic resizing of the cluster.

Evie Kassela et. al introduced an extended TIRAMOLA [31], which also focuses on automatic resizing of a cluster. But the main emphasis is on the workload-aware approach. It identifies the different workload types and also considers this workload-aware approach for decision making. In this work, Markov Decision Process (MDP model) is fine tuned by incorporating workload-aware approach.

Athanasios Naskos et. al presented the cloud elasticity using the probabilistic model checking [40], approach for resizing a cluster of virtual machines. In this work, probabilistic models are used in the decision making process.

Upendra Sharma et. al presented cost-aware elasticity approach [47]. In this work, they have used a new approach customer centric view where the customer can optimize the capacity by selecting the servers that matches their needs. This approach makes use of both replication and migration for dynamic provisioning the capacity. It also uses an integer linear program formulation for optimizing the cost. They have used TPC-W benchmark for evaluation. Performance evaluation is also conducted on public as well as a private cloud.

2.3 Partitioning Evaluator

Various approaches are proposed by researchers for achieving scalable transactions in the cloud, but there is no approach present for evaluating partitioning algorithm.

Sudipto Das et. al developed ElasTras [22], which uses schema level partitioning for achieving scalability. Schema level partitioning is evaluated using TPC-C benchmark with two different metrics as throughput and response time.
Curino et. al presented the Relational Cloud [21] which uses workload-aware graph partitioning approach for improving scalability. Workload-aware graph partitioning is being evaluated with the metric throughput and response time.

Miguel Liroz-Gistau et. al proposed dynamic workload-based partitioning algorithms for continuously growing databases [34]. They have proposed metric as an efficiency for evaluation of dynamic workload-based partitioning. They have defined efficiency as the ratio between minimum number of relevant fragments of the query and the number of fragments that are actually accessed.

Xiaoyan Wang et. al presented an approach for automatic data distribution [54]. They have compared their approach with hashing and round robin algorithm. They evaluated their automatic data distribution algorithm with the metric efficiency.

Jeyakumar Muthuraj et. al presented partitioning evaluator [39] for the vertical partitioning problem. With the evaluator, one can evaluate any vertical partitioning algorithm. It is also used to compare and evaluate the existing vertical partitioning algorithm. This partition evaluator uses common objective function for evaluating vertical partitioning algorithm. This evaluator is based on the input model as a matrix which contains attributes (columns) and the transactions (rows) which contains the frequency of access to attributes. But this evaluator, works only with the vertical partitioning algorithm.

2.4 Conclusion

In this chapter, various techniques to build scalable and elastic data store are discussed. A notable work in the area of scalable transactions, workload-aware elasticity is presented. A taxonomy of data management in the cloud which highlights different approaches and various techniques is proposed. A comparison of the proposed work with prior work in terms of partitioning technique, partitioning algorithm, partitioning approach is presented. A milestone in scalable transactions from year 2007 to 2015 is presented. In subsequent chapter, different approaches for building scalable and elastic cloud data store is presented.