Chapter 3

Workload-Driven Data Partitioning in Cloud Data Stores

3.1 Introduction

In this chapter, system architecture, design of workload-driven partitioning is introduced. Data partitioning strategy is presented. The mathematical formulation of workload-driven partitioning is described. The workload-driven partitioning algorithm which reorganizes partitions according to data access patterns is discussed. It also covers the definition of data access patterns, static, dynamic, workload-driven partitioning. The comparison of static, dynamic, workload-driven partitioning is explained in detail.

3.2 Workload-Driven Partitioning

This thesis describes techniques to build a scalable data store for web applications. Web applications require the data store to be scalable for update-intensive workload [26] to maintain performance under varying workload. Due to the emerging popularity of the internet, many applications are deployed on the internet and have faced the challenge of serving thousands of customers. Therefore scalability of e-commerce web applications has become an important issue. These modern web applications generate huge amount of data. The database management system plays an important role in managing large amount of data. In order to maintain consistent and reasonable performance, the DBMS must scale out to low cost commodity hardware. Traditional, relational databases could not be scaled out to low cost commodity servers [22]. This
gives birth to the No SQL data stores [1, 4, 14, 25, 49]. In the real world, data is managed in two families of database. On the one hand, relational databases, Online Transaction Processing (OLTP) which provides strong consistency, but cannot scale out. On the other hand, NoSQL data stores [1, 4, 14, 25, 49] used for Analytics are scalable but support weak consistency. The chapter, describes how scalable web applications can be developed using NoSQL data stores [1, 4, 14, 25, 49] and how they can be used to support transactional properties. This requires reorganization of application data, based on data access patterns [53] of web applications and distributing data across multiple data partitions. This work presents a technique for restructuring of application data, based on data access patterns of web applications and show that restructuring data of an application, scales the system linearly. The technique which is commonly used to improve scalability is data partitioning. Different methods which are used for partitioning data: horizontal partitioning, vertical partitioning, and hybrid partitioning [41]. Horizontal partitioning divides the table based on particular column and groups related fragments into different subtables. In vertical partitioning the table is split based on column and each partition consists of a subset of columns. In hybrid partitioning it splits the entire table into fragments which contains a subset of rows and columns. When the table is partitioned horizontally, it is split into different fragments and allocated to available nodes. So the partitions are dependent on total number of rows in the table. Scalability is achieved by distributing data among different nodes. The following aspects should be taken into account before designing the horizontal partitioning scheme: selection of partitioning key, data partitioning technique, and routing of queries to the concerned node. To split the table horizontally, first the partitioning key is selected. It is usually unique id. Once the partitioning key is selected, the fragments are split using the data partitioning algorithm. Horizontal data partitioning algorithm will map the data, based on partitioning keys to available nodes. Different techniques used for horizontal data partitioning are range, hash and workload-aware partitioning [19]. To achieve, the optimized query performance transaction should visit a smaller number of nodes. The data partitioning algorithm [19, 22] should group the related rows together and kept on one partition for efficient execution of transactions. There is need to take care of load balancing to gain throughput from horizontal data partitioning algorithm [19, 22]. The number of transactions should also be distributed uniformly between all partitions on nodes. An unbalanced workload on the partition creates hot spots and affects the overall throughput of the system. Therefore load balancing is achieved by uniformly distributing the data items and workload across the partitions.
The range partitioning algorithm selects the partitioning key and splits the table into ranges. These ranges are mapped to partitions across the nodes. A hashing algorithm is popularly used to evenly distribute the data among the partitions. It uses a hash function for uniform distribution of data. Workload-aware algorithm partitions the database with graph partitioning. The concept is to group the related rows together so that the transaction should access a minimum number of partitions. The limitations with these classical techniques is that they result in distributed transactions when accessing the data partitions across multiple servers. This results in limited scalability. In comparison, of classical range and hash partitioning, workload-aware partitioning algorithms are efficient. However, the existing workload-aware algorithms, do not address the problem of changing workloads in dynamic environments.

In real world scenario, when the customer places any order for an item, an order is fulfilled by a warehouse. If the warehouses on one partition are running out of stock, it is fulfilled by warehouse on another partition. In this way, there is always a pattern that which warehouse is more probable to supply to particular warehouse. This pattern is referred as Data Access Pattern. However, the existing static partitioning techniques do not model real world e-commerce application scenario. Thus, there is a need to develop a workload-driven partitioning scheme based on the access patterns of data to improve scalability. An efficient partitioning scheme should access a minimum number of partitions when the transaction (query) is executed. In this work a new metric of data access patterns is presented, which are constantly monitored and the partitions are formed accordingly. This chapter discusses techniques to improve the scalability of web applications. In this chapter, workload-driven partitioning is proposed to restructure the application data, based on data access patterns of web applications. The design and prototype implementation of workload-driven partitioning in cloud data stores such as Amazon SimpleDB and Hadoop HBase [1] is presented.

The main contributions of this chapter are structured as follows:

- The architecture of the system is presented 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. It 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 one partition is explained.

- A Mathematical Formulation of the workload-driven partitioning scheme is also presented. Data partitioning strategy, which describes how the partitions are formed to foster the scalability is also explained.

- The workload-driven partitioning algorithm, which restructures application data (warehouses) based on data access patterns is developed. Demonstration of detailed experiments that show the effectiveness of workload-driven partitioning scheme in forming partitions, that balance the workload among the partitions is described.

- The scalability of workload-driven partitioning over cloud data stores such as Amazon SimpleDB and Hadoop HBase is proved. This partitioning scheme is being evaluated using TPC-C industry standard benchmark.

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

- The statistical analysis of workload-driven partitioning is presented using Multiple Linear Regression (MLR) and forecasting the scalability of the workload-driven partitioning scheme using a universal scalability law.

- The experiments with workload-driven partitioning scheme and schema level partitioning is presented and the proposed partitioning scheme is compared with the schema level partitioning to show its effectiveness. The performance of the workload-driven partitioning in public as well as a private cloud is analyzed.

- Demonstration of workload-driven partitioning algorithm, which forms optimized partitions and load balancing capabilities for transactional workloads is shown here. Experimental and analytical results are observed and it shows that workload-driven partitioning outperforms schema level partitioning.

This chapter is structured as follows. Section 3.3, presents the architecture of the system. Section 3.4 explains the design of the workload-driven partitioning scheme and the partitioning strategy. Section 3.5, explains about genetic algorithm and how it can be used to find the optimized solution. Section 3.6, explains the mathematical formulation of the workload-driven partitioning scheme. Section 3.7, presents the
algorithm of the workload-driven partitioning scheme. Section 3.8 compares static, dynamic, workload-driven partitioning. Section 3.9, discusses about the performance analysis of the algorithm. Section 3.10 presents the benefits of using Denormalization. Section 3.11, shows an experimental evaluation for Amazon SimpleDB and Hadoop HBase. Section 3.12 describes the statistical model and analysis using the Regression technique for the experimentation carried out in Amazon SimpleDB. Section 3.13 explains about forecasting the scalability of workload-driven partitioning using universal scalability law. Section 3.14 compares the performance of schema level and workload-driven partitioning in public as well as a private cloud. Section 3.15 presents conclusion.

3.3 System Architecture

The architecture of system is described in four layers as shown in figure 3-1.

- Transaction layer
  It is the first layer in the architecture of the system. It is responsible for mapping user operations to application transactions. It invokes application transactions. For example. If the user clicks on the buy button, the new order transaction will be invoked by transaction layer. The application logic resides in transaction

![Figure 3-1: System Architecture](image-url)
layer. It has two different components 1) Transaction Manager 2) Data Manager. Transaction Manager used to execute transactions and Data Manager for storage of data.

- Control Layer
  This layer contains two important elements. 1) Master 2) Metadata Manager
  The Master periodically checks the status of all the domains and keeps uniform load across all the domains. The Metadata manager provides useful information. It consists of a system catalog which maps partition to the domains. The Metadata Manager maintains the information about the mapping of partitions to the domain.

- Routing Layer
  Client library look ups the metadata and it is responsible for routing the query to the appropriate domain. Two look ups are performed when the request is made by the client. A request to the Metadata Manager to retrieve the address of system catalog and then querying the system catalog to find the location of the partition on the domain is performed.

- Database layer
  It consists of cluster of domains where the partitions are stored. It is also responsible for handling the application request for data, processing the queries and providing the result in appropriate format.

3.4 Design of Workload-Driven Partitioning

3.4.1 Formal Definitions

Definition 1 Data Access Pattern:
In an e-commerce application, when the customer places any order, the order is fulfilled by a warehouse. If the warehouses on one partition are running out of stock, it is fulfilled by a warehouse on another partition. So, there is always a pattern, which warehouse is more probable to supply to a particular warehouse. This behavior is tracked and the pattern is identified. This pattern is referred as the Data Access Pattern. Figure 3-2 illustrates the data access patterns of web applications.

Note :- In this work, warehouse is representation of a warehouse in the real world.
Definition 2 Static Partitioning:
Static partitioning systems [11, 13, 21, 22] are the systems where the related data items are put together on one partition. Once the partitions are formed, those partitions do not change. Therefore, these partitioning systems are termed as static.

Definition 3 Workload-Driven Partitioning:
In this partitioning, the transaction logs are analyzed, and the data access patterns are monitored, that is the movement of data periodically. The partitions are formed, based on this movement of data.

Partitioning plays a very important role to optimize the performance and scalability of Online Transaction Processing (OLTP) transactions. ElasTras [22] provides a great way to statically partition the data by providing a very high degree of load balancing and generates less number of distributed transactions. But as in OLTP millions of transactions are expected, so there may be a scope for improvement. The design of the systems [11, 13, 21, 22] described above is based on the assumption that, an application accesses the partition statically. The applications for which there are dynamic changes in the data access pattern, making use of a static partitioning approach would result in distributed transactions. This work presents the technique, which forms the partitions based on data access patterns of web applications.

Note :- In this work, warehouse is representation of a warehouse in the real world.
In this work, the design of the workload-driven partitioning system, is discussed. The proposed partitioning scheme improves the scalability and reduce the distributed transactions than the existing partitioning algorithm. Workload-driven partitioning allows to design scalable and real-life web applications. The TPC-C schema [26] presents an e-commerce application. The TPC-C schema consists of nine tables such as warehouse, district, customer, order, order-line, new-order, item, stock and history. In the TPC-C schema, the warehouse table has wid as a primary key, but act as a foreign key in all the other tables except an item table. The database is partitioned using the partitioning key as wid and all the related rows of wid in the other tables should be kept on one partition. TPC-C assumes that in 10% of the cases, the current warehouse may not have the stock to fulfill the order. In such cases, the TPC-C randomly chooses the supplier warehouse when the order is not satisfied by the current warehouse. In reality, it is hardly random, and usually, the supplier warehouse is the one, which is the most proximate to the warehouse, which is processing that order. In this way there is always a pattern that, which warehouse is more probable to supply to a particular warehouse. In this work, the idea is to track this behavior by analyzing the transaction log processed by the OLTP system and re-organize these static partitions so that the warehouse with more coherency are put in a single partition and reduce the number of distributed transactions required. For example, Let us consider four warehouses as wid – 0 Delhi, wid – 1 Mumbai, wid – 2 Kolkata, wid – 3 Chennai.

Initially, wid – 0 and wid – 1 will be on one partition and wid – 2 and wid – 3 will be on another partition. It is observed that when there is no stock available with wid – 2 the orders are fulfilled by supplier wid – 0, which is more proximate to wid – 2. Wid – 2 is a warehouse, which is processing that order and wid – 0 is a supplier warehouse, which is supplying stock for the order. In this way the transaction log is monitored and the partitions are formed based on data access patterns of web applications. It should be noted that the item table is read only table, and each warehouse tries to maintain stock for the 100,000 items. Figure 3-3 shows that the partitions are formed based on data access patterns of web applications. Figure 3-4 demonstrates an illustration of workload-driven partitioning.

---

Note :- In this work, warehouse is representation of a warehouse in the real world.
Figure 3-3: Design of Workload-Driven Partitioning

Figure 3-4: Workload-Driven Partitioning
3.4.2 Data Partitioning Strategy

In this partitioning strategy, an exhaustive survey is performed to find the best load distribution. All possible combinations of partitions are found out. The total load and association is calculated for all possible combinations of partition. A Heuristic Search technique was used to find optimized solutions. These combinations are generated using mutation in the genetics algorithm. Mutation is a technique, which have been used in genetic algorithms for introducing diversity. Mutation helps in generating optimized combinations. In mutation, the solution may change entirely from the previous solution. Hence, the genetic algorithm can come to a better solution by using mutation. The partition with optimized load and association are selected and give a higher throughput.

3.5 Genetic Algorithm

A Genetic Algorithm is a technique, used to find approximate solution. They use techniques such as inheritance, mutation, selection, and crossover for finding optimized solution. The following steps explain, how a genetic algorithm can be used to find an optimized solution. The evolution begins with a population of randomly generated possible solutions. Genetic algorithm introduces a sequence of new populations. In each generation, it uses all possible combinations of solution in current generation to create the next population.

- Population is created by finding all possible combinations of solutions.

- These randomly generated possible combinations are evaluated.

- Evaluation function is a criterion for ranking these possible combinations of solutions.

- These possible combinations are selected based on their fitness. Then these combinations are ranked based on their fitness. The fitness of a solution is measured as the result given by that combination. Higher fitness increases the chance of being selected.
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>It is a set of fragments.</td>
</tr>
<tr>
<td>$f$</td>
<td>It is a fragment.</td>
</tr>
<tr>
<td>$L_w_i$</td>
<td>It is the number of transactions executed on the warehouse ‘$i$’.</td>
</tr>
<tr>
<td>$L_p_k$</td>
<td>It is the number of transactions executed on the partition ‘$k$’.</td>
</tr>
<tr>
<td>$LD_{mean}$</td>
<td>It is the average of transactions executed on the all the partitions.</td>
</tr>
<tr>
<td>$\Delta(LD)$</td>
<td>It is the standard deviation of load.</td>
</tr>
<tr>
<td>Association$(f)$</td>
<td>Number of local transactions executed on the fragment.</td>
</tr>
<tr>
<td>$dt$</td>
<td>Total number of distributed transactions.</td>
</tr>
<tr>
<td>$r$</td>
<td>Total number of records.</td>
</tr>
<tr>
<td>$T$</td>
<td>Total number of transactions.</td>
</tr>
<tr>
<td>$s$</td>
<td>Total number of partitions.</td>
</tr>
<tr>
<td>$n$</td>
<td>Total number of warehouses.</td>
</tr>
</tbody>
</table>
3.6 Mathematical Formulation of Workload-Driven Partitioning

In this section, the problem of workload-driven partitioning is modeled using load and association. The goal of the workload-driven partitioning scheme is to find the partitions with optimal association and load. The workload-driven partitioning scheme is designed with different optimization objectives.

- First, the workload-driven partitioning scheme aims to minimize the distributed transactions than the existing static partitioning scheme.
- The second objective of the workload-driven partitioning scheme is to form partitions in such a way that the load is distributed evenly across all the partitions. This is done with an aim to improve the efficiency and throughput of workload-driven partitioning. Table 3.1 shows notations used in this chapter. In this section, the problem of workload-driven partitioning is defined. Workload-driven partitioning scheme is done over a set of *warehouses* (*wid* as the partitioning key), and for a given workload.

Let $D = \{d_1, d_2, \ldots, d_q\}$ be the set of data items. The workload consists of a set of queries $W = \{q_1, q_2, \ldots, q_r\}$, and $P = \{p_1, p_2, \ldots, p_s\}$ be the set of partitions.

Workload-driven partitioning is defined as follows:

**Definition 4** Workload-driven partitioning of a data set $D$ consists of dividing the data of $D$ into a set of fragments which are mutually exclusive sets of fragments where the union of all fragments is equal to $F$ and the count of the element in set $F$ are equal to the count of elements in set $P$.

$$F = \{f_1, f_2, \ldots, f_p\} \quad (3.1)$$

where $f_1$ is $d_1$, $d_2$ and $f_2$ is $d_3$, $d_4$ and so on.

In this paper, we define the efficiency of the partitioning scheme as a transaction (query) should access a minimum number of partitions when it gets executed.

**Definition 5** Given a transaction, the efficiency for a partitioning scheme for a given workload is computed as follows and is denoted as:

$$Efficiency(W) = 1 - \frac{dt}{T} \quad (3.2)$$
\(dt\) is distributed transactions and \(T\) is total number of transactions.

**Definition 6** In workload-driven partitioning, the partitions are formed by calculating the load on warehouses and the association between them. Let us first define the load on the partition. The load of a partition \(p_k\), denoted as \(Lp_k\), is defined as the total of the number of transactions executed on the warehouses in that partition.

Let \(Lp_k\) represent the number of transactions executed on the partition \(k\). \(n\) is number of warehouses in one partition.

\[
Lp_k = \sum_{i=1}^{n} Lw_i \tag{3.3}
\]

**Definition 7** The average load of a partition is denoted as \(LD_{\text{mean}}\) is the total number of transactions executed on all the partitions divided by the total number of partitions. Standard deviation is defined as the deviation of load on the partitions from the average load of the partition. \(s\) is the number of partitions.

\[
LD_{\text{mean}} = \frac{\sum_{k=1}^{s} Lp_k}{s} \tag{3.4}
\]

\[
\Delta (LD) = \sqrt{\frac{\sum_{k=1}^{s} (Lp_k - LD_{\text{mean}})^2}{s}} \tag{3.5}
\]

**Definition 8** The association of fragment \(f\), denoted \(\text{Association}(f)\) is defined as the number of local transactions executed on a fragment.

\[
\text{Association}(f) = r \tag{3.6}
\]

The objective of workload-driven partitioning is to form partitions in such a way that efficiency of the partitioning scheme is maximized.
3.7 Workload-Driven Partitioning Algorithm

Workload-driven partitioning algorithm takes a set of warehouses, set of domains and complete transaction data as an input and gives the optimized partition as a result. The algorithm starts with the static distribution of warehouses. For example, warehouses w0 and w1 are kept on one partition and w2 and w3 are kept on another partition. Then, the partitions are mutated, that is finding out all possible combinations of warehouses to form the partition. It then calculates the load distribution by using standard deviation of all the loads on each of the warehouses for that partition from the average load. The combinations are ranked, based on the load distribution values with the lowest value to the combination with the smallest standard deviation; and higher values to the one with the highest deviation. The association of a combination is calculated by finding out the number of transactions executed, and distributed transactions for the combination. The combinations are ranked, based on association such as a lower rank value to the combination with the highest association and a higher rank to the combinations with a lower association. A summation of both the ranks is computed and the ranks are specified in ascending order considering both load distribution and association.

Once this algorithm is run, fragment set $F$ with the optimized load balancing and optimized association is generated. Then the final combination is used to populate data for workload-driven partitioning.

This algorithm, reads the transaction log and builds all the different combinations of the possible partitions and calculates the total load and total association of that partition (total number of local transactions; if the database is partitioned in this way.) Then, the ranks are assigned to each of the combinations according to their load in increasing order as well as an association in decreasing order. Then, ranks are summed up and this sum is used to generate the final ranks of each of the partitions. Then top five combinations are selected based on the final rank and repeat steps 2 to 9 for 5 times to generate more combinations. After generation of these combinations, again select top five combinations and repeat steps 2 to 9 of them for 5 times. The reason for doing so is for generating more number of partitions and also to check that the same combination of partition is generated. The reason for performing this step 5 times is that it has been observed that no new combinations are generated after repeating for 5 times. The partition with the smallest rank (faring best in load as well as an association) will be used to repartition the data.
Workload-Driven Partitioning Algorithm

**Input:** 1. Number of Partitions, 2. Set of Warehouse, 3. Transaction Data

**Output:** Partition with the optimized load balancing and optimized association.

1. Start with static distribution.

```
repeat
    2. Mutate_partition(partition, warehouse);
    3. foreach partition do
        foreach warehouse do
            calculate $Lw_i$;
            $Lp_k = \sum_{i=1}^{n} Lw_i$;
        end
        $LD_{mean} = \frac{\sum_{k=1}^{s} Lp_k}{s}$;
        $\Delta (LD) = \sqrt{\sum_{k=1}^{s} \left(\frac{(Lp_k-LD_{mean})^2}{s}\right)}$
    end
    4. Sort_partition_load_ascending_order($\Delta (LD)$, s);
    5. foreach transaction do
        requester_warehouse;
        supplier_warehouse;
        array[requester_warehouse],[supplier_warehouse]=cnt++;
    end
    6. Sort_partition_association_descending_order();
    7. Read_partition_load_rank();
    Read_association_rank();
    repeat
        | Rank Value = $\sum (partition\_load\_rank, association\_rule\_rank)$;
    until end;
    8. Sort_rank_value_ascending_order();
    9. Select top 5 combinations
until end of partitions;
```

10. Select the top combination as the best combination with effective load balancing and association.

**Algorithm 1:** Workload-Driven Partitioning Algorithm
3.8 Comparison of Static, Dynamic and Workload-Driven Partitioning

1. 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. The advantage of using static partitioning is the partitions are fixed, so that the data need not be migrated very frequently. The disadvantage of static partitioning is more number of distributed transactions occur. In an e-commerce application when an order is placed by customers and if the current warehouse does not have stock to fulfill the orders, it goes to warehouse on another partition. Therefore, distributed transactions occur.

2. Dynamic Partitioning

Dynamic partitioning systems [24] are the systems where partitions are formed dynamically and change very frequently. The advantage of using dynamic partitioning is an occurrence of less number of distributed transactions. The cost of migrating data is an overhead. The restructuring of application data in a partition introduces additional cost due to data migration.

3. Workload-Driven Partitioning (Partitioning based on data access patterns)

Workload-driven partitioning is not static or dynamic partitioning scheme. It lays between static and dynamic partitioning scheme. In this partitioning, the transaction logs and the data access patterns are analyzed (that is, which warehouse is more probable to supply to particular warehouse). This analysis is performed periodically and the partitions are formed based on data access patterns. Once the partitions are formed, they may change in future, based on data access patterns. Therefore, this scheme cannot be classified as static or dynamic partitioning. The advantage of using this partitioning scheme is partitions are formed after performing an analysis. Therefore the least number of distributed transactions occur. This analysis is performed periodically and therefore the reorganization of application data is not frequent. Thus the cost is also minimized.
3.9 Performance Analysis of Algorithm

The performance of the above stated algorithm depends majorly on ‘r’ and ‘T’. Step 3 stated in the above algorithm has $\mathcal{O}(n.s)$. First, for loop run ‘s’ times and the inner for loop executes ‘n’ times. Step 4 sort partition on the basis of load distributions using an efficient sorting algorithm like merge sort. It will take $\mathcal{O}(slog(s))$ time. Step 5 executes for $\mathcal{O}(r.T)$. Again, ranking using merge sort in step 6 will take time $\mathcal{O}(slog(s))$. Thus, total time complexity can be stated as below:

$$T = \mathcal{O}(n.s) + \mathcal{O}(r.T) + \mathcal{O}(slog(s))$$

(3.7)

Since $n, s < r$, $T$

$$T = \mathcal{O}(r.T)$$

(3.8)

Analysis of the above algorithm for best, average and worst case is also performed. As per the analysis the complexity in all the three cases is given in equation 3.9 and dependent on ‘r’ and ‘T’.

$$T = \mathcal{O}(r.T)$$

(3.9)

3.10 Data Denormalization

In the classical approach, web applications which are using a relational database as a backend, query to multiple tables for the execution of transactions. The process of changing data structure for improving performance is called as Data Denormalization. It is used to introduce redundancy by adding attributes to existing tables. With data denormalization expensive queries can be replaced by simple queries. It also helps to distribute the data among different servers and thus the scalability is improved. Data denormalization improves performance and minimizes the cost.

3.11 System Implementation

This section discusses implementation details of the workload-driven partitioning on two scalable database layers. NoSQL data stores [1, 4, 14, 25, 49] support different data models. Adaption of workload-driven partitioning to these NoSQL data stores is actually a challenge and require minor changes for implementation. Workload-driven
partitioning is being implemented using two prominent and widely used NoSQL data stores: Amazon SimpleDB [4] and Hadoop HBase [1].

3.11.1 Performance Evaluation in Amazon SimpleDB.

An extensive experimental evaluation is performed in a cluster of 5 machines in Amazon Web Services Elastic Compute Cloud [3] (EC2) infrastructure. The scalability of workload-driven partitioning is validated by showing the performance evaluation of a prototype implementation on Amazon SimpleDB [4] running in the Amazon Cloud.

3.11.1.1 Experimental Setup

The experiments were performed in a cluster of 5 machines in Amazon EC2 [3]. All virtual machines used in the cluster were M3 General Purpose Extra Large with 15GB of memory, 20 EC2 Compute Units (4 virtual cores with 3.25 units each), (2*40GB) of local storage. All the 5 machines in the cluster are interconnected by Gigabit LAN. M3 General Purpose Extra Large cost $0.50 per instance-hour. One EC2 Compute Unit provides the CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. The transaction load and the number of users are simulated using multi-threaded requests. Table 3.2 shows the experimental setup.

Table 3.2: Experimental Setup

<table>
<thead>
<tr>
<th>No. of Machines</th>
<th>Environment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>CPU</td>
<td>M3 General Purpose Extra Large,</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>(4 core * 3.25 unit)</td>
</tr>
<tr>
<td></td>
<td>Storage</td>
<td>15 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(2x 40 GB)SSD</td>
</tr>
<tr>
<td>All</td>
<td>OS, .NET Framework</td>
<td>Windows 8</td>
</tr>
<tr>
<td></td>
<td>NO SQL Database</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amazon SimpleDB</td>
</tr>
</tbody>
</table>

3.11.1.2 Partitioning Algorithm

The quality of the workload-driven partitioning algorithm is assessed using TPC-C workload [26]. The quality of the workload-driven partitioning algorithm is compared with the schema level partitioning [22] in terms of the distributed transactions. The scalability of the concerned partitioning scheme is evaluated by the throughput and response time of the transaction.
3.11.1.3 TPC-C benchmark

Workload-driven partitioning scheme was applied to the standard web application such as TPC-C [26] to illustrate its effectiveness. TPC-C is an industry standard benchmark, used for simulating the workload of an e-commerce application. It represents the standard OLTP workload. It contains read as well as update transactions. The benchmark describes a wholesale supplier with a geographically disseminated district and warehouses. The benchmark has five types of transactions and nine tables. Workload-driven partitioning scheme is validated by using the TPC-C industry standard benchmark. The schematic diagram of the TPC-C benchmark, which is selected for performance evaluation is shown in figure 3-5.

The benchmark consists of five different transactions by identifying the business needs of e-commerce applications:

- NEW ORDER transaction, which accepts and creates a new order from the customer. It is a mixture of read as well as write transactions.
- PAYMENT transaction, which updates the balance of the customer by reflecting the payment of the order by the customer. It is also a read and write transaction.
- ORDER STATUS, which keeps track of the status of customers and most recent orders. It is a read only transaction.
- DELIVERY transaction finds a batch of most recent 10 orders, which are not yet delivered to the customer.
- STOCK level transaction, which finds the recently sold items, which have got a stock below threshold. It is a read only transaction.

In a real life scenario, typically 45% transactions are NEW ORDER, 43% transactions are PAYMENT and 4% transactions are ORDER STATUS, DELIVERY, and STOCK.
Figure 3-5: TPC-C Schema
### 3.11.1.4 Conversion of TPC-C Schema to Amazon SimpleDB Domain

TPC-C was originally designed for web application with relational databases as backend. Therefore, it is needed to adapt the relational data model of TPC-C to Amazon SimpleDB data model [4]. In this section, the design of Amazon SimpleDB has been modeled from TPC-C schema. Merging of these nine tables (warehouse, district, customer, neworder, order, orderline, item, stock) into one domain of Amazon SimpleDB is performed to extend this Amazon SimpleDB data model. Figure 3-6 shows the conversion of TPC-C schema to the Amazon SimpleDB domain. Figure 3-7 demonstrates horizontal workload-driven partitioning in Amazon SimpleDB.

<table>
<thead>
<tr>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>wid</td>
</tr>
<tr>
<td>w_name</td>
</tr>
<tr>
<td>w_city</td>
</tr>
<tr>
<td>w_state</td>
</tr>
<tr>
<td>w_zip</td>
</tr>
<tr>
<td>w_tax</td>
</tr>
<tr>
<td>w_ytd</td>
</tr>
<tr>
<td>d_id</td>
</tr>
<tr>
<td>d_w_id</td>
</tr>
<tr>
<td>d_city</td>
</tr>
<tr>
<td>d_ytd</td>
</tr>
<tr>
<td>d_next_o_id</td>
</tr>
<tr>
<td>c_id</td>
</tr>
<tr>
<td>c_d_id</td>
</tr>
<tr>
<td>c_first</td>
</tr>
<tr>
<td>c_discount</td>
</tr>
<tr>
<td>no_o_id</td>
</tr>
<tr>
<td>no_d_id</td>
</tr>
<tr>
<td>no_w_id</td>
</tr>
<tr>
<td>o_id</td>
</tr>
<tr>
<td>o_c_id</td>
</tr>
<tr>
<td>o_w_id</td>
</tr>
<tr>
<td>ol_o_id</td>
</tr>
<tr>
<td>ol_d_id</td>
</tr>
<tr>
<td>ol_number</td>
</tr>
<tr>
<td>ol_delivery_d</td>
</tr>
<tr>
<td>s_i_id</td>
</tr>
<tr>
<td>s_w_id</td>
</tr>
<tr>
<td>s_quantity</td>
</tr>
<tr>
<td>i_id</td>
</tr>
<tr>
<td>i_price</td>
</tr>
</tbody>
</table>

Figure 3-6: Mapping of TPC-C Schema to Amazon SimpleDB Domain
Figure 3-7: Workload-Driven Partitioning
3.11.1.5 Scalability Evaluation

In this section, the performance of the workload-driven partitioning is extensively evaluated and compared it with a schema level partitioning. In schema level partitioning [22], partitions are formed by collocating the related data items. In graph partitioning [19], the database is partitioned with frequently used attributes, that is common partitioning key \((\text{wid})\). All related rows are put together on the same partition. The goal of this experiment is to validate the scalability of system with varying number of concurrent clients. The scalability in terms of throughput and response time and efficiency is evaluated. For conducting the experiments, the database size is set to 15 warehouses. Users are available starting from 250 to 5000 in steps of 250.

![Figure 3-8: Throughput for Varying Number of Concurrent Users](image)

Figure 3-8 shows the throughput of workload-driven partitioning, schema level partitioning. Along x-axis, there are varying number of concurrent users and along the y-axis, there is a throughput (transactions per second).

Figure 3-9 shows the response time of workload-driven and schema level partitioning. Along the x-axis, the number of concurrent users is plotted and along the y-axis, time is plotted in seconds. As observed from figure 3-9, workload-driven partitioning has lesser response time than schema level graph partitioning.

Figure 3-10 shows distributed transactions that occur in workload-driven and schema level partitioning. From figure 3-10, it is observed that in most of the cases workload-driven partitioning has lesser number of distributed transactions than schema level partitioning.
Figure 3-9: Response Time for Varying Number of Concurrent Users

Figure 3-10: Distributed Transactions
3.11.2 Performance Evaluation in Amazon SimpleDB with Varying Data Size.

In this section, the scalability of workload-driven partitioning is validated with varying data size. The sensitivity of workload-driven partitioning algorithm is verified with varying data size.

3.11.2.1 Experimental Setup

The experiments were conducted in a cluster of 3 machines in Amazon EC2. All virtual machines used in the cluster were M3 General Purpose Extra Large with 15GB of memory, 20 EC2 Compute Units (4 virtual cores with 3.25 units each), (2*40GB) of local storage. All the 5 machines in the cluster are interconnected by Gigabit LAN. M3 General Purpose Extra Large cost $0.50 per instance-hour. One EC2 Compute Unit provides the CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. The transaction load simulated using multi-threaded requests. Table 3.3 shows the experimental setup.

Table 3.3: Experimental Setup

<table>
<thead>
<tr>
<th>No. of Machines</th>
<th>Environment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>CPU</td>
<td>M3 General Purpose Extra Large, (4 core * 3.25 unit)</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td>15GB</td>
</tr>
<tr>
<td></td>
<td>Storage</td>
<td>(2x 40GB)SSD</td>
</tr>
<tr>
<td>All</td>
<td>OS</td>
<td>Windows 8</td>
</tr>
<tr>
<td></td>
<td>.NET Framework</td>
<td>4.0</td>
</tr>
<tr>
<td></td>
<td>NO SQL Database</td>
<td>Amazon SimpleDB</td>
</tr>
</tbody>
</table>

3.11.2.2 Scalability Evaluation

In this section, the performance of the workload-driven partitioning is evaluated with varying data size. Users are varied from 50 to 1450 in steps of 50. The experiments were performed with varying data size, as 15 and 30 warehouses. Figures 3-11 and 3-12 shows the throughput of workload-driven and schema level partitioning with 15 and 30 warehouses respectively. It is observed that workload-driven partitioning is more scalable than schema level partitioning and there is no change in performance with the growing database size. Figures 3-13 and 3-14 plots response time of workload-driven and schema level partitioning with 15 and 30 warehouses respectively. Figures 3-15
and 3-16 plots distributed transactions that occur in workload-driven and schema level partitioning with 15 and 30 *warehouses* respectively. Figures 3-17 and 3-18 shows the efficiency of workload-driven and schema level partitioning with 15 and 30 *warehouses*.

Figure 3-11: Throughput with 15 Warehouses

![Schema Level Partitioning vs Workload-Driven Partitioning](image1)

Figure 3-12: Throughput with 30 Warehouses

After analyzing the performance of the workload-driven partitioning in Amazon SimpleDB, it is comprehended that workload-driven partitioning has got higher
Figure 3-13: Response Time with 15 Warehouses

Figure 3-14: Response Time with 30 Warehouses
Figure 3-15: Distributed Transactions with 15 Warehouses

Figure 3-16: Distributed Transactions with 30 Warehouses
Figure 3-17: Efficiency with 15 Warehouses

Figure 3-18: Efficiency with 30 Warehouses
throughput and low response time in Amazon SimpleDB. Workload-driven partitioning technique works better than schema level partitioning by demonstrating it on Amazon SimpleDB cloud data store. Although the implementation in Amazon SimpleDB cloud data store increases the response time for a concurrent number of users. This restricts the practical utility of this technique in Amazon SimpleDB cloud data store. As a result implementation of this technique in a commercial cloud data store is needed. Therefore, workload-driven partitioning is presented using NoSQL data store such as Hadoop HBase.

3.11.3 Experimental Evaluation in Hadoop HBase

The scalability of the workload-driven partitioning algorithm is shown by presenting the performance evaluation of a prototype implementation on scalable database layer as Hadoop HBase [1] running in the existing local cluster. The effect of the Heuristics in the partitioning algorithm is observed, on partitioning efficiency. Hadoop HBase [1] also provides efficient storage and fast retrieval of data to support high performance web applications.

3.11.3.1 Experimental Setup

In this section, an experimental validation of the workload-driven partitioning algorithm is presented. The schematic diagram of the experiment’s setup is shown in figure 3-19. The performance of the proposed partitioning scheme is experimentally evaluated on contemporary cloud data store such as Hadoop HBase. Table 3.4 shows the experimental setup for Hadoop HBase cluster. In the existing experimental setup one node acts as a master (Name Node) for HDFS and HBase. The Hadoop HBase cluster was composed of 5 region servers, with 5 data nodes and one workload generator. The master node has a configuration of the Intel Core2Duo processor 3.1GHz, with 4GB of memory, and a hard disk of 320GB. All the data nodes used in conducting experiments have a Core2Duo processor at 3.1 GHZ, with 4GB of memory. The TPC-C database with 15 warehouses has been populated. In the experiment, there are 3 warehouses per region server. These machines are connected using Gigabit LAN. Emulated browsers are used for simulating the requests of real users. The client workload is generated by varying the number of emulated browsers.
Table 3.4: Experimental Setup for Hadoop HBase Cluster

<table>
<thead>
<tr>
<th>No. of Machines</th>
<th>Environment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (Master)</td>
<td>CPU</td>
<td>Core2Duo processor 3.1GHz 4GB DDR2 320 GB SATA</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard Disk</td>
<td></td>
</tr>
<tr>
<td>5 (Slaves)</td>
<td>CPU</td>
<td>Core2Duo processor 3.1GHz 4GB DDR2 320 GB SATA</td>
</tr>
<tr>
<td></td>
<td>Memory</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hard Disk</td>
<td></td>
</tr>
<tr>
<td>All</td>
<td>OS</td>
<td>Ubuntu 13.04 1.7</td>
</tr>
<tr>
<td></td>
<td>Java</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NO SQL Database</td>
<td>Hadoop HBase 0.92.1</td>
</tr>
</tbody>
</table>

Figure 3-19: Experimental Setup
3.11.3.2 Migration of TPC-C to Cloud

In this section, the mapping of TPC-C schema to the data model of Hadoop HBase is performed. There are total nine tables as district, customer, warehouse, orders, new-order, order-line, stock, item, and history in the TPC-C schema. These nine tables are mapped to a single table in Hadoop HBase. Figure 3-20 shows the mapping of TPC-C schema to Hadoop HBase. The history table has not been considered while creating the HBase table. Each table in TPC-C schema is created as a column family in Hadoop HBase. In the Hadoop HBase, table district, customer, warehouse, order, new-order, item, order-line and stock are all column families, which are a group of related columns. To convert this data model of TPC-C schema to Hadoop HBase, nine tables of TPC-C schema (district, customer, warehouse, order, new-order, item, order-line and stock) are combined into a single table of Hadoop HBase. The reason for creating a separate column family for each table is to minimize the response time for retrieving the results.

<table>
<thead>
<tr>
<th>District</th>
<th>Customer</th>
<th>Warehouse</th>
<th>Order</th>
<th>New_Order</th>
<th>Item</th>
<th>Order_Line</th>
<th>Stock</th>
</tr>
</thead>
<tbody>
<tr>
<td>d_id</td>
<td>c_id</td>
<td>w_id</td>
<td>o_id</td>
<td>no_o_id</td>
<td>i_id</td>
<td>o_l_o_id</td>
<td>s_i_id</td>
</tr>
<tr>
<td>d_w_id</td>
<td>c_d_id</td>
<td>w_name</td>
<td>o_o_id</td>
<td>no_o_id</td>
<td>l_im_id</td>
<td>o_d_id</td>
<td>s_w_id</td>
</tr>
<tr>
<td>d_name</td>
<td>c_w_id</td>
<td>w_street_1</td>
<td>o_d_id</td>
<td>no_w_id</td>
<td>l_name</td>
<td>o_w_id</td>
<td>s_w</td>
</tr>
<tr>
<td>d_street_1</td>
<td>c_first</td>
<td>w_street_2</td>
<td>o_w_id</td>
<td>l_price</td>
<td>l_data</td>
<td>o_number</td>
<td>s_w</td>
</tr>
<tr>
<td>d_street_2</td>
<td>c Midwest</td>
<td>w_city</td>
<td>o_entry_d</td>
<td>l_data</td>
<td></td>
<td>o_price</td>
<td>s_w</td>
</tr>
<tr>
<td>d_city</td>
<td>c_last</td>
<td>w_state</td>
<td>o_carrier_id</td>
<td>l_data</td>
<td></td>
<td>o_time</td>
<td>s_w</td>
</tr>
<tr>
<td>d_state</td>
<td>c_city</td>
<td>w_zip</td>
<td>o_l_lnt</td>
<td>s_deliver</td>
<td></td>
<td>o_delivery</td>
<td>s_w</td>
</tr>
<tr>
<td>d_zip</td>
<td>c_state</td>
<td>w_tax</td>
<td>o_all_lnt</td>
<td>s_deliver</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td>d_tax</td>
<td>c_phone</td>
<td>w_ytd</td>
<td>s_amount</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td>d_ytd</td>
<td>c_since</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td>d_next_o_id</td>
<td>c_credit</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td></td>
<td>c_creditLim</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td></td>
<td>c_discount</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td></td>
<td>c_delivery</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td></td>
<td>c_payment</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td></td>
<td>c_balance</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
<tr>
<td></td>
<td>c_ytd_payment</td>
<td>s_dist</td>
<td>s_dist</td>
<td>s_dist</td>
<td></td>
<td>s_dist</td>
<td>s_w</td>
</tr>
</tbody>
</table>

Figure 3-20: Mapping of TPC-C Schema to Hadoop HBase

3.11.3.3 Scalability Evaluation

In this experiment, the number of concurrent users is varied in a cluster of 6 machines. In the cluster of 6 machines, the database is populated with 15 warehouses. The number of users varied from 20 to 100 in steps of 20. The purpose of this experiment was to validate the scalability of workload-driven partitioning scheme with the increasing number of concurrent users and transactions. The database size is set to 15 warehouses. Figures 3-21, 3-22, 3-23, 3-24, 3-25 shows system throughput of new order, payment, delivery, stock, order status transactions in workload-driven and
schema level partitioning. Figure 3-26 shows the throughput of the workload-driven partitioning scheme with database size set to 15 warehouses. Along the x-axis, there are number of concurrent users, and along the y-axis, there is the throughput (in tps). Throughput of workload-driven partitioning scheme scales linearly with database size set to 15 warehouses is observed in figure 3-26. Figures 3-27 shows the response time of system. Along the x-axis, there are number of concurrent users, and along the y-axis, there is response time. It is observed from the figure 3-28, that workload-driven partitioning has least number of distributed transactions as compared to the schema level partitioning. In schema level partitioning, once the partitions are formed, those partitions do not change. So when the request comes to a particular warehouse of not having stock, it is fulfilled by another warehouse on another partition. Thus, the distributed transactions occur. On the other hand, in workload-driven partitioning, partitions are formed by analyzing the transaction logs. Therefore, a less number of distributed transactions occur, which in turns increases the throughput of the system. Workload-driven partitioning performs better than schema level partitioning and improve throughput by 10%. In OLTP applications, millions of users are active concurrently across the web and placing an order for their item. So this 10% change is also critical for OLTP applications.

![Figure 3-21: System Throughput of New Order Transaction](image)

Figure 3-21: System Throughput of New Order Transaction
Figure 3-22: System Throughput of Payment Transaction

Figure 3-23: System Throughput of Delivery Transaction
Figure 3-24: System Throughput of Stock Transaction

Figure 3-25: System Throughput of Order Status Transaction
Figure 3-26: System Throughput for Varying Number of Concurrent Clients in the Hadoop HBase Cluster

Figure 3-27: Response Time for Varying Number of Concurrent Clients in the Hadoop HBase Cluster
3.12 Statistical Model and Data Analysis in Amazon Public Cloud Using Multiple Linear Regression

3.12.1 Workload-Driven Partitioning

In this section statistical analysis of throughput for workload driven partitioning is performed using Multiple Linear Regression.

Multiple explanatory variables are used to predict the output of the response variable. The aim of using a Multiple Linear Regression model is to show the relationship between the throughput, number of users and distributed transactions. The explanatory variables which are identified as number of users, distributed transactions and response variable as throughput. The coefficient of determination $R^2$ is determined through $f$-test. Tables 3-5 and 3-6 shows the coefficient of determination $R^2$ as 0.99 and the coefficient of predictors.

Table 3.5: Model Summary of Workload-Driven Partitioning in Amazon Cloud using Multiple Linear Regression

| Model | R   | R Square | Adjusted R Square | Std. Error of the Estimate | Change Statistics |  |
|-------|-----|----------|-------------------|-----------------------------|-------------------|-
| 1     | 0.999a | .997     | .997              | 76.82789                    | .997              | 3381.353 | 2 | 17 | .000 |
Table 3.6: Statistical Analysis of Workload-Driven Partitioning in Amazon Cloud using Multiple Linear Regression

<table>
<thead>
<tr>
<th>Model</th>
<th>Unstandardized Coefficients</th>
<th>Standardized Coefficients</th>
<th>t</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>5685.639</td>
<td>525.332</td>
<td>1.074</td>
<td>.000</td>
</tr>
<tr>
<td>Users</td>
<td>1.054</td>
<td>.085</td>
<td>12.423</td>
<td>.000</td>
</tr>
<tr>
<td>DistTrans</td>
<td>-1.026</td>
<td>1.164</td>
<td>-.881</td>
<td>.391</td>
</tr>
</tbody>
</table>

From table 3.6, the equation is written in the following form.

$$\text{Throughput} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

(3.10)

By putting values into it.

$$\text{Throughput} = 5685.639 + 1.054 * x_1 - 1.026 * x_2$$

(3.11)

The model of workload-driven partitioning shows that the experimental values found to be in good agreement with the predicted value and has less error.

3.13 Forecasting Scalability of Workload-Driven Partitioning with the USL

In this section, the scalability of workload-driven partitioning is predicted using the Universal Scalability Law [12] and comparing it with the scalability demonstrated by the experiments in Hadoop HBase [1]. The model predicts that the system under test will reach its peak throughput of 591 queries per second at a concurrency of 5. As shown in figure 3-26 it is observed that the system reaches to a throughput of 655 queries per second at a concurrency of 20. Figure 3-29 shows the predicted and experimental throughput of workload-driven partitioning.


In this section, the performance of the workload-driven partitioning is compared in Amazon SimpleDB and Hadoop HBase. The concurrent number of users is varied from 250 to 5000. Linear increase in throughput is observed. For 250 users, through-
put was 5375 and for 5000 users, it was 10143. For processing these 5375 transactions, the response time was 238.34 seconds and for 10143 transactions the response time was 396.41 seconds. It has been observed that a prototype implementation of workload-driven partitioning in Amazon SimpleDB deployed on a cluster of commodity servers in the Amazon public cloud can efficiently serve thousands of users.

The response time and throughput of workload-driven partitioning in Amazon SimpleDB is not encouraging and this inhibits the use of Amazon SimpleDB for real world web applications.

The performance of the workload-driven partitioning in the Hadoop HBase local cluster is also observed. In a Hadoop HBase local cluster, the throughput was 655 transactions per second for 20 users and 1264 transactions for 100 users. The response time is also observed as 378 ms for 20 users and 540 ms for 100 users. From the observations above, it is concluded, that workload-driven partitioning generates higher throughput and lower response time in Hadoop HBase as compared to Amazon SimpleDB.

### 3.15 Conclusion

Most approaches for building scalable web applications uses static data structure. In this chapter, workload-driven partitioning is presented to fulfill the requirements of modern cloud based applications. Workload-driven partitioning algorithm is introduced to restructure application data based on data access patterns of web applica-
tions. This methodology was applied to standard web application such as TPC-C benchmark and portrayed that it allows TPC-C, the most popular and challenging benchmark to scale. Data partitioning strategy for finding out all possible combinations of partitions is introduced. Mutation is a technique, which have been used in genetic algorithms for generating all possible combinations of partitions. The mathematical formulation of workload-driven partitioning is modeled. The solutions is validated through experimentation over contemporary data stores such as Hadoop HBase and Amazon SimpleDB. It is demonstrated that, scalability improvement can be gained using a technique called as Denormalization. Denormalization improves the performance for update intensive queries is concluded. This also suggests that denormalized implementation is more scalable than classical implementation. Denormalization does not imply any inconsistencies. This aspect makes the concerned approach more effective than existing techniques. TPC-C benchmark is used for the evaluation of workload-driven partitioning scheme. By demonstrating the concerned partitioning scheme using the TPC-C benchmark, it is observed that workload-driven partitioning reduces the number of distributed transactions than the existing partitioning schemes as schema level partitioning, and gives higher throughput, efficiency and lower response time. Statistical model using Multiple Linear Regression to predict the throughput of workload-driven partitioning is developed. The comparison of performance of the workload-driven partitioning in public as well as a private cloud is explained. It is demonstrated that a prototype implementation deployed on a cluster of commodity servers can efficiently serve thousands of users while maintaining throughput. In workload-driven partitioning, the partitions are formed based on data access patterns of web applications. But this restructuring of warehouses in a partition introduces additional cost due to data migration. The next chapter shows how this workload-driven partitioning can be extended to achieve workload-aware elasticity in cloud data stores.