Ahirrao and Ingle *Journal of Cloud Computing: Advances, Systems and Applications* (2015) 4:21
DOI 10.1186/s13677-015-0047-3

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Scalable transactions in cloud data stores

Swati Ahirrao[1*] and Rajesh Ingle[2]

Abstract

Cloud Computing is a successful paradigm for deploying scalable and highly available web applications at low cost. In real life scenarios, the applications are expected to be scalable and consistent. Data partitioning is a commonly used technique for improving scalability. Traditional horizontal partitioning techniques are not capable of tracking the data access patterns of web applications. The development of novel, scalable workload-driven data partitioning is a requirement for improving scalability. This paper proposes a novel workload-aware approach, with scalable workload-driven data partitioning based on data access patterns of web applications for transaction processing. It is specially designed to scale out using NoSQL data stores. In contrast to the existing static approaches, this approach offers high throughput, lower response time, and a less number of distributed transactions. Further, implementation and validation of scalable workload-driven partitioning scheme is carried out through experimentation over cloud data stores such as Hadoop HBase and Amazon SimpleDB. An experimental results of the concerned partitioning scheme is conducted using the industry standard TPC-C benchmark. Analytical and experimental results are observed and it shows that scalable workload-driven data partitioning outperforms the schema level and graph partitioning in terms of throughput, response time and distributed transactions.

Keywords: Data partitioning, Cloud computing, Workload-driven, Scalability, Partitioning scheme

Introduction

Building scalable and consistent data management have been the vision of database researchers for the last few years. With 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. This gives birth to the No SQL data stores [1–5]. The key-value stores [6] includes properties such as scalability, availability, and elasticity. Scalability is achieved using data partitioning [6]. Data partitioning is a commonly used technique for performing scale out operation. In an e-commerce application, when the customer places any order, the order is fulfilled by a *warehouse*. If the *warehouses* on one partition is 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 [7]. Static partitioning systems [8–11] are the systems in which 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 called as static. In scalable workload-driven partitioning scheme, the transaction logs, are analyzed and the data access patterns are monitored, that is the movement of data periodically. Based on this movement of data, partitions are formed and do not remain same forever. Classical partitioning techniques such as hash, range partitioning are simple to use, but they result in distributed transactions when accessing data from different servers. However, the existing partitioning algorithms [10, 11] (static) do not work well when the data access pattern changes and also do not model real world e-commerce application scenario. Thus, there

*Correspondence: swatia@sitpune.edu.in
[1] Symbiosis International University, Pune, India
Full list of author information is available at the end of the article

is a need to develop a partitioning scheme based on the access patterns of data to improve scalability. Scalable workload-driven partitioning is specially designed for OLTP web applications. 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. The main contributions of this paper are structured as follows:

- 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 single 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 the application data (warehouses) based on Data Access Pattern 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 practical implementation of the workload-driven partitioning scheme over a cloud data stores such as Hadoop HBase and Amazon SimpleDB is presented. The TPC-C benchmark is also used for performance evaluation.

The remainder of the paper is organized as follows. Section 'Related work', presents the existing work. It analyses the powerful models of Online Transaction Processing (OLTP). It also explains the different partitioning techniques such as schema level, and graph partitioning. Section 'Design of scalable workload-driven partitioning', presents the design of the scalable workload-driven partitioning and partitioning strategy. Section 'Mathematical formulation of scalable workload-driven partitioning scheme', explains how scalable workload-driven partitioning is formulated mathematically. Section 'Scalable workload-driven partitioning algorithm', presents the algorithm which generates partitions with optimized load and association. Section 'Comparison of static, dynamic and scalable workload-driven partitioning' discusses comparison of static, dynamic and scalable workload-driven partitioning. Section 'System implementation', shows an experimental evaluation. Finally, Section 'Conclusion' concludes the paper.

Related work

Researchers have proposed a variety of systems and partitioning techniques to provide scalable transaction support for web applications. Some of them have been listed here. Grolinger K., et al. presented [6] different partitioning techniques used by NoSQL data stores and NewSQL data stores to achieve scalability. Sandholm T. et al. presented [12] notes on Cloud Computing principles which describes horizontal scalability for achieving scalable transactions in cloud data stores. Sudipto Das et al., proposed ElasTras [13], which uses schema level partitioning 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 [9] where transactions are forced to execute on one partition. In the Cloud SQL Server, the partition is called a table group, which is normally keyless or keyed. If it is keyed, then all the tables in the table group must have a common key (primary key). The row group is a set of all tuples that have a common partitioning key. Curino et al., suggested the Relational Cloud [14] in, which scalability is achieved with the workload-aware approach termed as graph partitioning [10]. In graph partitioning, the data items, which are accessed by the transactions are kept on a single partition. J. Baker et al., presented Megastore [8] 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 algorithm and it is designed with the common objective, where the related rows are kept on a single partition. But there are some applications such as online games, where groups are formed dynamically with time and therefore, Sudipto Das et al., proposed G-Store [15], where the keys from a group on a different node are put together and form a new group on a single partition. There is another approach, which works without the partitioning technique. Aguilera et al., presented Sinfonia [16], 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. Wei et al., introduced Cloud TPS [17], 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 an advance. Scalability is achieved by distributing the data among the Local Transaction Managers. D. Lomet et al., proposed design Deuteronomy [18] in which 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. Therefore, it is not suitable for large cloud deployments. Ahirrao and Ingle presented [19] scalable transactions in cloud data stores which explains the concept of partitioning based on data access patterns of web applications.

Data partitioning methods

Schema level partitioning

The Schema Level partitioning scheme [11] is a static partitioning scheme designed to improve the scalability of ElasTras [13]. It is derived from the TPC-C [20] schema, so it is called as Schema Level partitioning. The TPC-C schema has a hierarchical tree structure. In 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.

Graph partitioning

Graph partitioning [10] 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.

Researchers have proposed various partitioning techniques such as range, hash, list, schema level, and graph partitioning to improve scalability. But there is no partitioning technique present, which forms the partitions based on data access patterns of web applications. Partitioning plays a very important role to optimize the performance and scalability of Online Transaction Processing (OLTP). ElasTras [13] 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 [8–11] 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.

Design of scalable workload-driven partitioning

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 1 illustrates the data access patterns of web applications.

Definition 2. *Static Partitioning:*
Static partitioning systems [8–11] 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 called as static.

Definition 3. *Scalable Workload-Driven Partitioning:*
In this partitioning, we analyze the transaction logs, and monitor the data access pattern, that is the movement of data periodically. The partitions are formed, based on this movement of data.

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

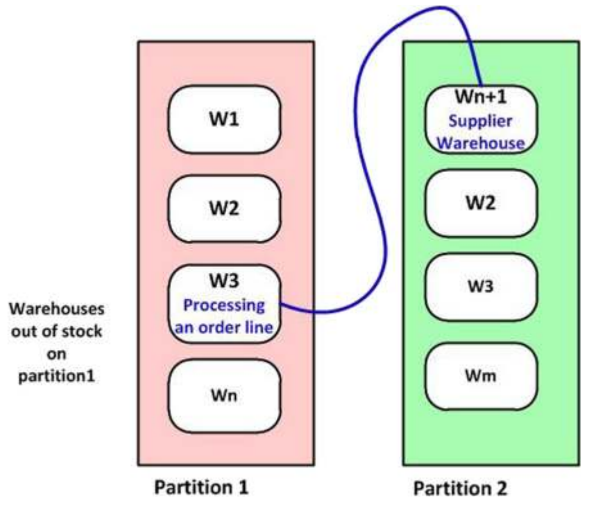


Fig. 1 Data access patterns of web applications

using mutation. The partition with optimized load and association are selected and give a higher throughput.

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 population. 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.
- Sort this combination, based on their fitness. Lower the rank, higher the chance of being selected.
- These selected combinations again regenerate all possible combinations of solution. We call it as a new population.
- This continues until the optimized solution has been found.

In this section, the design of the scalable workload-driven partitioning system, is discussed. The proposed partitioning scheme improves the scalability and reduce the distributed transactions than the existing partitioning algorithm. Scalable workload-driven partitioning allows to design scalable and real-life web applications. The TPC-C schema [20] 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 stocks to fulfill the order. Though, 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 2 shows that the partitions are formed based on data access patterns of web applications. Figure 3 illustrates the scalable workload-driven partitioning.

Mathematical formulation of scalable workload-driven partitioning scheme

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

- First, the scalable workload-driven partitioning scheme aims to minimize the distributed transactions than the existing static partitioning scheme.
- The second objective of the scalable 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 the scalable workload-driven partitioning scheme. Table 1 shows notations used in this paper. In this section, the problem of workload-driven partitioning is defined. Scalable workload-driven partitioning is done over a set of *warehouses* (*wid* as the partitioning key), and for a given workload.

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

Scalable workload-driven partitioning is defined as follows:

Definition 4. *Scalable 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 count of the element in set F are equal to the count of elements in set P.*

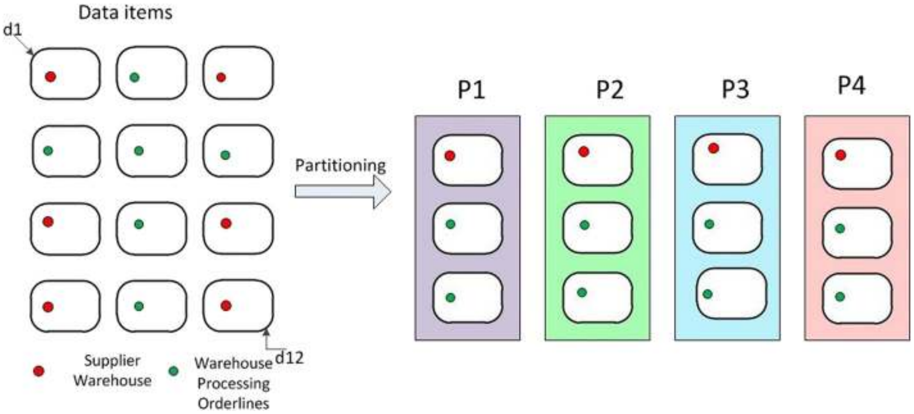


Fig. 2 Scalable Workload-Driven Partitioning

$$F = \{f_1, f_2, \cdots, f_p\} \quad (1)$$

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 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:*

$$Efficiency(W) = 1 - dt/T \quad (2)$$

dt is distributed transactions and T is total number of transactions.

Definition 6. *In scalable 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.

$$Lp_k = \sum_{i=1}^{n} Lw_i \quad (3)$$

Definition 7. *The average load of a partition is denoted as LD_{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.*

$$LD_{mean} = \frac{\sum_{k=1}^{s} Lp_k}{s} \quad (4)$$

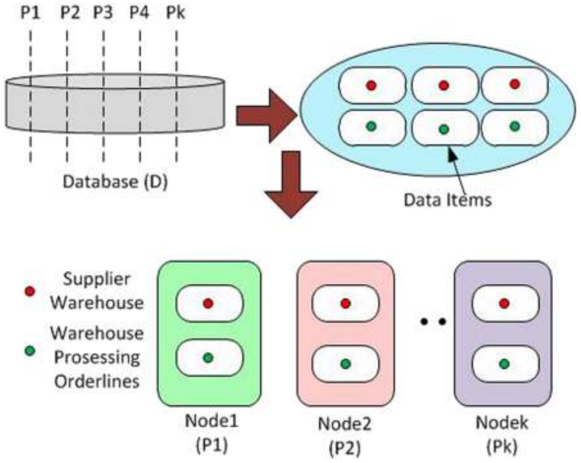


Fig. 3 Design of scalable workload-driven partitioning based on data access patterns

Table 1 Notations

Symbol	Description
F	It is a set of fragments.
f	It is a fragment.
Lw_i	It is the number of transactions executed on the warehouse 'i'.
Lp_k	It is the number of transactions executed on the partition 'k'.
LD_{mean}	It is the average of transactions executed on the all the partitions.
$\Delta(LD)$	It is the standard deviation of load.
$Association\,(f)$	Number of local transactions executed on the fragment.
dt	Total number of distributed transactions.
r	Total number of records.
T	Total number of transactions.
s	Total number of partitions.
n	Total number of warehouses.

$$\Delta\,(LD) = \sqrt{\sum_{k=1}^{s}\left(\frac{(Lp_k - LD_{mean})^2}{s}\right)} \tag{5}$$

Definition 8. *The association of fragment f, denoted Association(f) is defined as the number of local transactions (queries) executed on a particular combination of warehouses in fragment.*

$$Association(f) = r \tag{6}$$

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

Scalable workload-driven partitioning algorithm

Scalable 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.

Algorithm 1: Scalable 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
    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
            $LD_{mean} = \frac{\sum_{k=1}^{s} Lp_k}{s}$;
            $\Delta\,(LD) = \sqrt{\sum_{k=1}^{s}(\frac{(Lp_k - LD_{mean})^2}{s}})$;
        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;
until end of partitions;
10. Select the top combination as the best combination
with effective load balancing and association.
```

Comparison of static, dynamic and scalable workload-driven partitioning

1. Static Partitioning
   Static partitioning systems [8–11] 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 [15] 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. Scalable Workload-Driven Partitioning (Partitioning based on Data Access Patterns) Scalable 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.

System implementation

This section discusses implementation details of scalable workload-driven partitioning on two inherently scalable database layers. NoSQL data stores [1–5] support different data models. An adaption of scalable workload-driven partitioning to these NoSQL data stores is actually a challenge and require minor changes for implementation. Scalable workload-driven partitioning is being implemented using two prominent and widely used NoSQL data stores: Amazon SimpleDB [2] and Hadoop HBase [1].

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:

$$T = \mathcal{O}(n.s) + \mathcal{O}(r.T) + \mathcal{O}(slog(s)) \qquad (7)$$

Since n, s < r, T

$$T = \mathcal{O}(r.T) \qquad (8)$$

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 Eq. 9 and dependent on ‘r’ and ‘T’.

$$T = \mathcal{O}(r.T) \qquad (9)$$

Performance evaluation in Amazon SimpleDB

An extensive experimental evaluation is performed in a cluster of 5 machines in Amazon Web Services Elastic Compute Cloud [21] (EC2) infrastructure. The scalability of scalable workload-driven partitioning is validated by showing the performance evaluation of a prototype implementation on Amazon SimpleDB [2] running in the Amazon Cloud.

Experimental setup

The experiments were performed on a cluster of 5 machines in Amazon EC2 [21]. 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 2 shows the experimental setup.

Table 2 Experimental setting for Amazon SimpleDB

No. of machines	Environment	Description
5	CPU	M3 general purpose extra large, (4 core * 3.25 unit)
	Memory	15GB
	Storage	(2 × 40 GB)SSD
All	OS	Windows 8
	.NET Framework	4.0
	NO SQL Database	Amazon SimpleDB

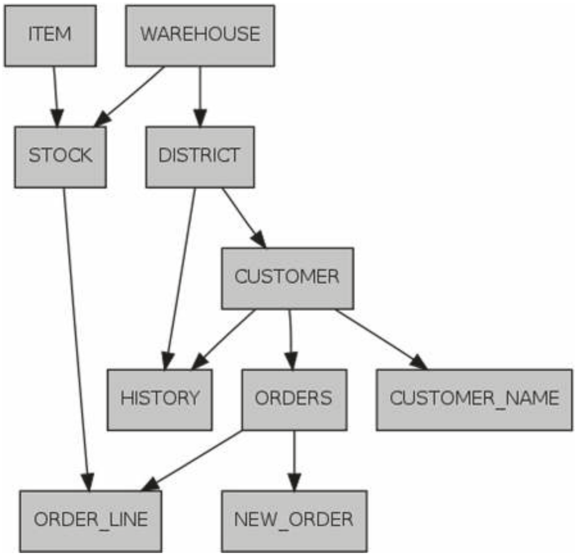


Fig. 4 TPC-C schema

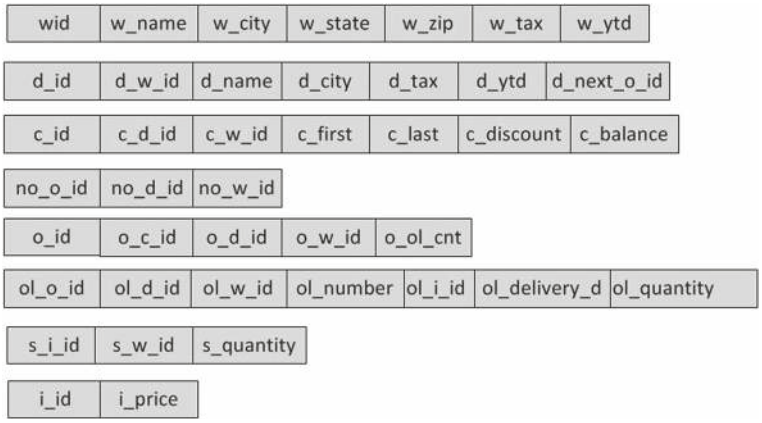
Partitioning algorithm

The quality of the scalable workload-driven partitioning algorithm is assessed using TPC-C workload [20]. The quality of the scalable workload-driven partitioning algorithm is compared with the schema level partitioning [11] and graph partitioning [10] in terms of the distributed transactions. The scalability of the concerned partitioning scheme is evaluated by the throughput and response time of the transaction.

TPC-C benchmark

Workload-driven partitioning scheme was applied to the standard web application such as TPC-C [20] 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 Fig. 4.

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



wid	w_name	w_city	w_state	w_zip	w_tax	w_ytd
d_id	d_w_id	d_name	d_city	d_tax	d_ytd	d_next_o_id
c_id	c_d_id	c_w_id	c_first	c_last	c_discount	c_balance
no_o_id	no_d_id	no_w_id				
o_id	o_c_id	o_d_id	o_w_id	o_ol_cnt		
ol_o_id	ol_d_id	ol_w_id	ol_number	ol_i_id	ol_delivery_d	ol_quantity
s_i_id	s_w_id	s_quantity				
i_id	i_price					

Fig. 5 Mapping of TPC-C schema to Amazon SimpleDB

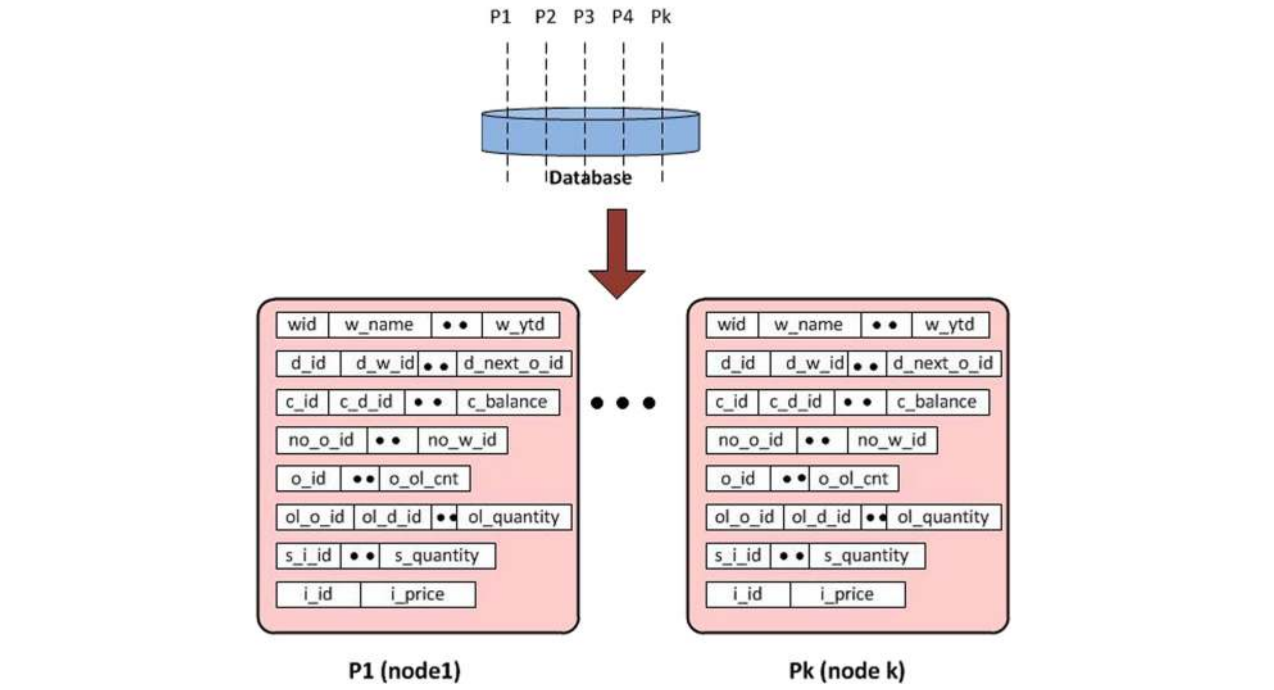


Fig. 6 Workload-driven partitioning in Amazon SimpleDB

- NEW ORDER transaction, which accepts and creates a new order for 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.

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

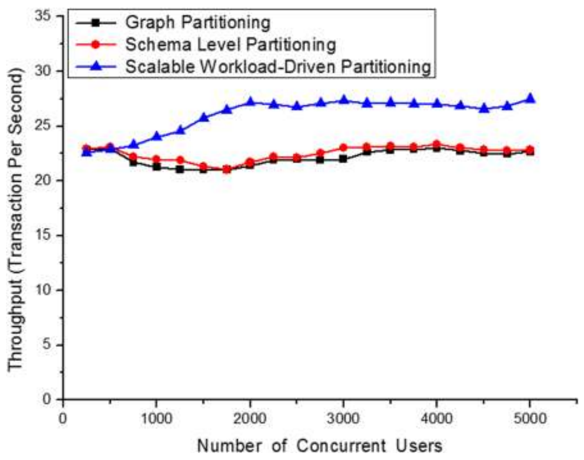


Fig. 7 Throughput for varying number of concurrent users

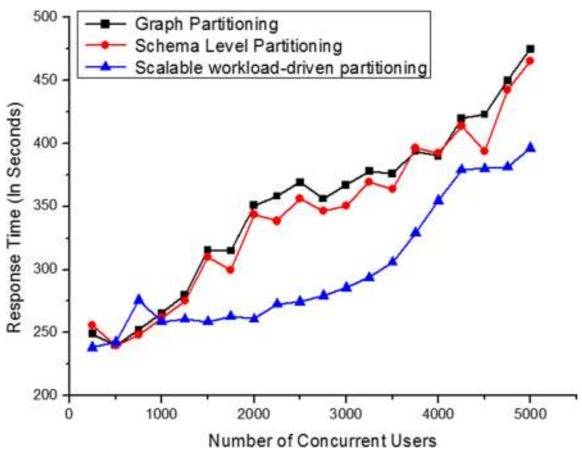


Fig. 8 Response time for varying number of concurrent users

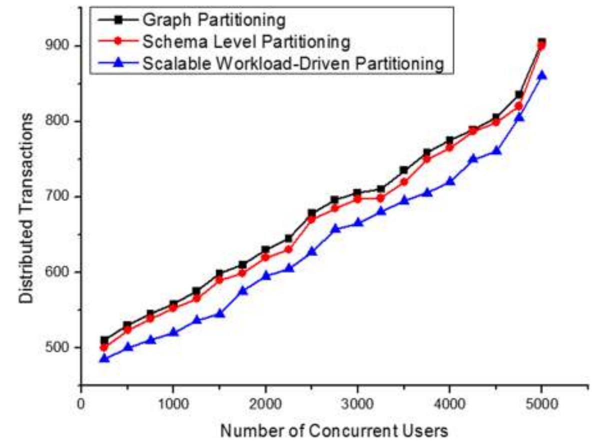


Fig. 9 Distributed transactions

to adapt the relational data model of TPC-C to Amazon SimpleDB data model [2]. 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 5 shows the conversion of TPC-C schema to the Amazon SimpleDB domain. Figure 6 demonstrates horizontal workload-driven partitioning in Amazon SimpleDB.

Scalability evaluation

In this section, the performance of the workload-driven partitioning is extensively evaluated and compared it with a schema level and Graph partitioning. In schema level partitioning [22], partitions are formed by collocating the related data items. In graph partitioning [10], the database is partitioned with frequently used attributes, that is common partitioning key (*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 7 shows the throughput of scalable workload-driven partitioning, schema level and graph partitioning. Along x-axis, there are varying number of concurrent users and along the y-axis, there is a throughput (transactions per second).

Figure 8 shows the response time with varying number of users of scalable workload-driven, schema level and graph 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 Fig. 8, scalable workload-driven partitioning has lesser response time than schema level and graph partitioning.

Figure 9 shows distributed transactions for scalable workload-driven, schema level and graph partitioning. From Fig. 9, it is observed that in most of the cases scalable workload-driven partitioning has lesser number of distributed transactions than schema level and graph partitioning. After analyzing the performance of scalable workload-driven partitioning in Amazon SimpleDB, it is comprehended that scalable workload-driven partitioning has got higher throughput and low response time in Amazon SimpleDB.

Scalable workload-driven partitioning technique works better than schema level and graph 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

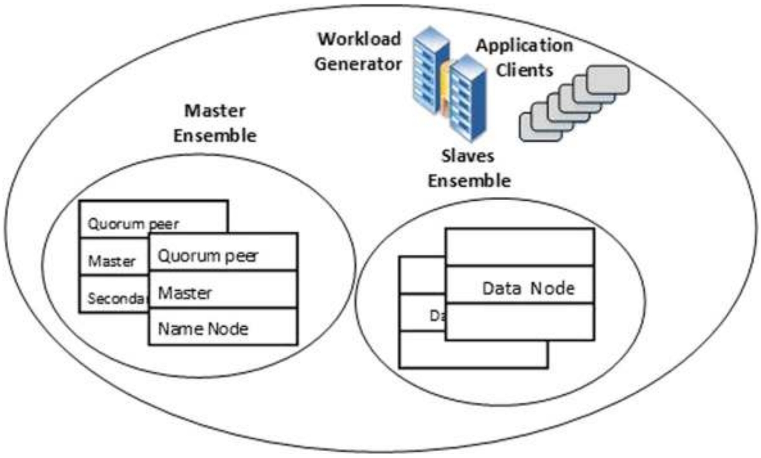


Fig. 10 Experimental setup

Table 3 Experimental setting for the Hadoop HBase cluster

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

cloud data store is needed. Therefore, scalable workload-driven partitioning is presented using NoSQL database such as Hadoop HBase.

Experimental evaluation in Hadoop HBase

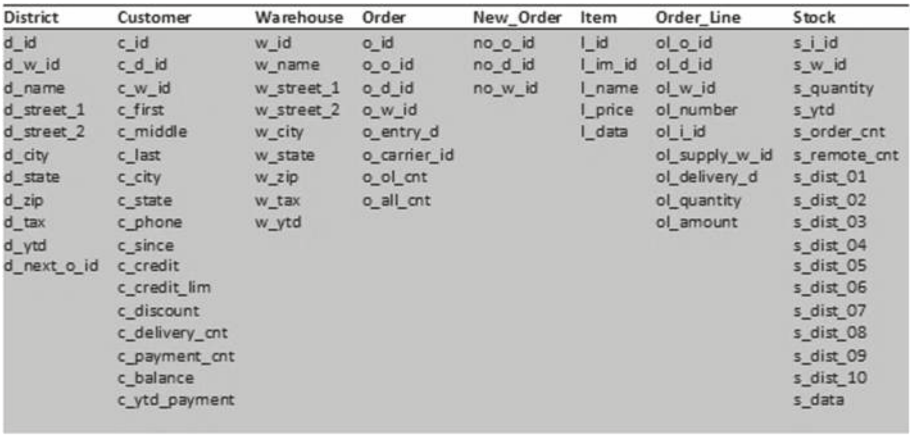
The scalability of the scalable workload-driven partitioning algorithm is shown by presenting the performance evaluation of a prototype implementation on scalable database layer as Hadoop HBase [1] running in the existing local cluster. The effect of the Heuristics in the partitioning algorithm is observed,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

Experimental setup

In this section, an experimental validation of the scalable workload-driven partitioning algorithm is presented. The schematic diagram of the experiment's setup is shown in Fig. 10. The performance of the concerned partitioning scheme is experimentally evaluated on contemporary cloud data store such as Hadoop HBase. Table 3 shows the experimental setting for Hadoop HBase cluster. In the existing experimental setup one node acts as a master (Name Node) for HDFS and HBase. 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 3.1 GHz, 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.

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 a 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 11 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.



District	Customer	Warehouse	Order	New_Order	Item	Order_Line	Stock
d_id	c_id	w_id	o_id	no_o_id	I_id	ol_o_id	s_i_id
d_w_id	c_d_id	w_name	o_o_id	no_d_id	I_im_id	ol_d_id	s_w_id
d_name	c_w_id	w_street_1	o_d_id	no_w_id	I_name	ol_w_id	s_quantity
d_street_1	c_first	w_street_2	o_w_id		I_price	ol_number	s_ytd
d_street_2	c_middle	w_city	o_entry_d		I_data	ol_i_id	s_order_cnt
d_city	c_last	w_state	o_carrier_id			ol_supply_w_id	s_remote_cnt
d_state	c_city	w_zip	o_ol_cnt			ol_delivery_d	s_dist_01
d_zip	c_state	w_tax	o_all_cnt			ol_quantity	s_dist_02
d_tax	c_phone	w_ytd				ol_amount	s_dist_03
d_ytd	c_since						s_dist_04
d_next_o_id	c_credit						s_dist_05
	c_credit_lim						s_dist_06
	c_discount						s_dist_07
	c_delivery_cnt						s_dist_08
	c_payment_cnt						s_dist_09
	c_balance						s_dist_10
	c_ytd_payment						s_data

Fig. 11 Mapping of TPC-C schema to Hadoop HBase

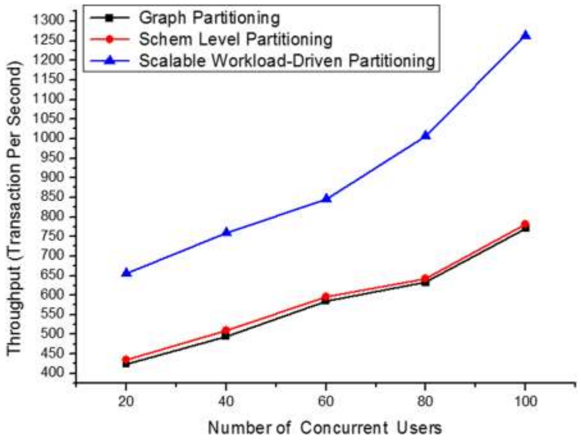


Fig. 12 Throughput with 15 warehouses

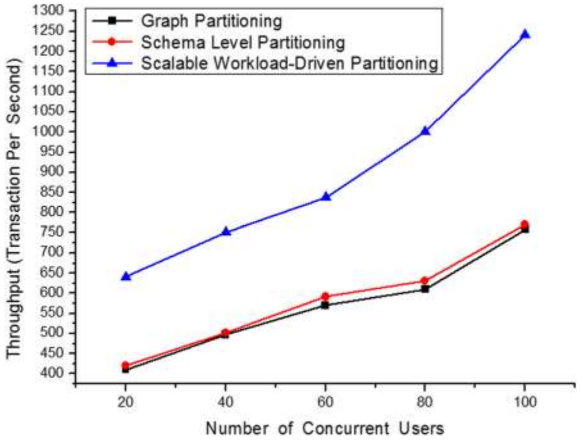


Fig. 14 Throughput with 60 warehouses

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*, 30 *warehouses*, 60 *warehouses* and 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. This experiment was conducted to check the sensitivity of scalable workload-driven partitioning algorithm with the increasing size of the database. The database size was set to 15 *warehouses*, 30 *warehouses*, 60 *warehouses*. Figure 12 shows the throughput of the scalable 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). Figures 13 and 14 demonstrate the throughput of the scalable workload-driven partitioning scheme with database size set to 30 and 60 *warehouses*. Throughput of scalable workload-driven partitioning scheme scales linearly with database size set to 15, 30, 60 *warehouses* is observed in Figs. 12, 13 and 14. There is no change in the performance of the system even if the database size is seen growing with 15, 30, 60 *warehouses*. Figures 15, 16 and 17 shows the response time of system with database size 15, 30, 60 *warehouses*. Along the x-axis, there are number of concurrent users, and along the y-axis, there is response time. From Figs. 15, 16 and 17 it is observed that the response time is almost same with database size 15, 30, 60 *warehouses*. It is observed from the Fig. 18, that scalable workload-driven partitioning has least number of distributed transactions as compared to the schema level and graph partitioning. In schema level and graph partitioning, once the partitions are formed, they 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

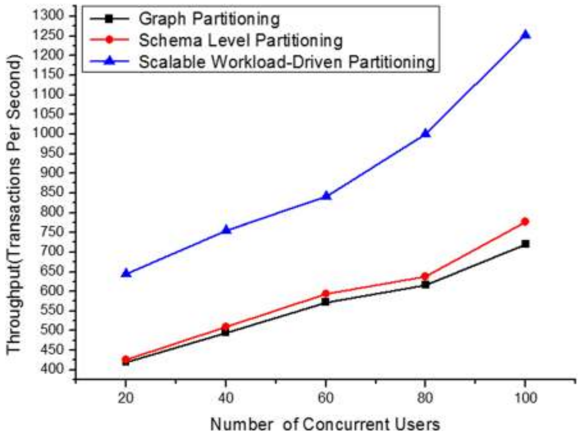


Fig. 13 Throughput with 30 warehouses

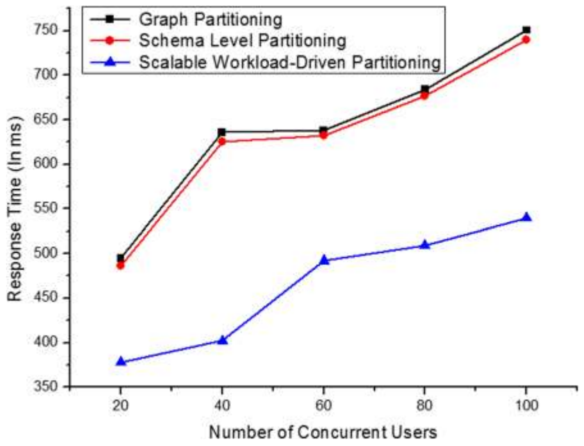


Fig. 15 Response time with 15 warehouses

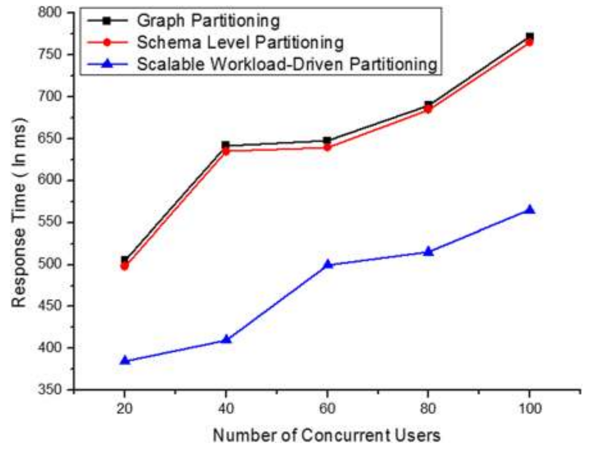


Fig. 16 Response time with 30 warehouses

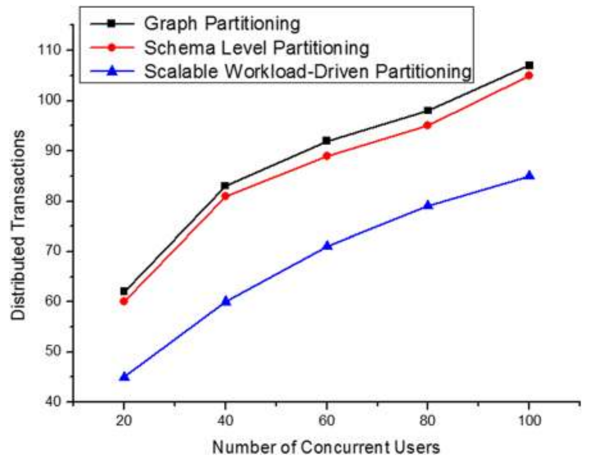


Fig. 18 Distributed transactions

occur. On the other hand, in scalable 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. Scalable workload-driven partitioning performs better than schema level and graph 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.

Forecasting the scalability with the universal scalability law

In this section, the scalability of scalable workload-driven partitioning is predicted using the universal scalability law [23] 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 Fig. 12 it can be observed that the system reaches to a throughput of 655 queries per second at a concurrency of 20. Figure 19 shows the predicted and experimental throughput of workload-driven partitioning.

Conclusion

In this paper, scalable workload-driven partitioning is presented to fulfill the requirements of modern cloud based applications. The mathematical formulation of scalable workload-driven partitioning is described. The solutions through experimentation over contemporary data store such as Hadoop HBase and Amazon SimpleDB were validated. The TPC-C benchmark is used for the evaluation of the concerned partitioning scheme. Furthermore, it is demonstrated that a prototype implementation deployed on a cluster of commodity servers can efficiently serve thousands of users while maintaining throughput. By demonstrating the concerned scheme using the TPC-C

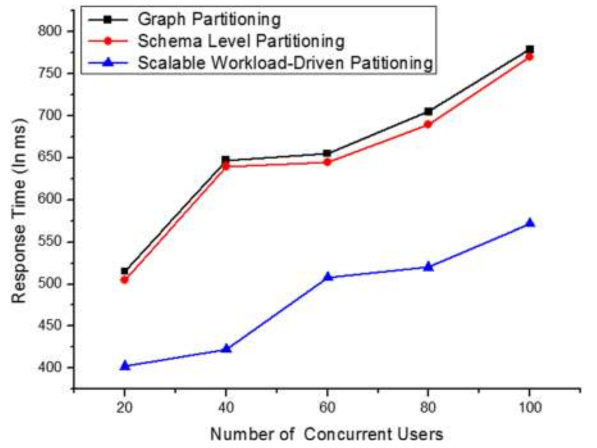


Fig. 17 Response time with 60 warehouses

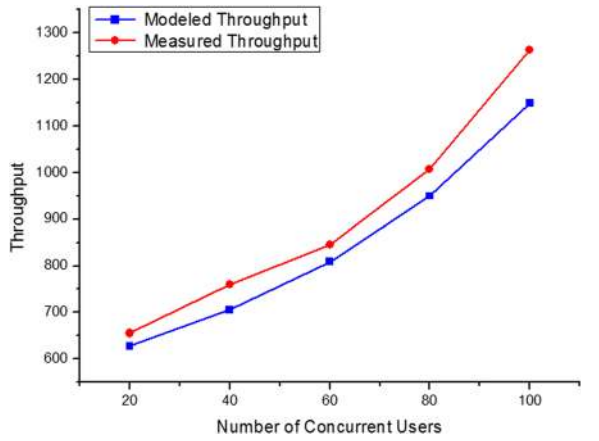


Fig. 19 Forecasting scalability of workload-driven partitioning using USL

benchmark, it has been observed that scalable workload-driven partitioning reduces the number of distributed transactions better than the existing partitioning schemes such as graph and schema level partitioning, and gives higher throughput, efficiency and lower response time.

Abbreviations

OLTP: online transaction processing; NoSQL: not only SQL; EC2: elastic compute cloud; HDFS: hadoop distributed file system; LAN: local area network; USL: universal scalability law; TPC-C: transaction processing council.

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

SA has been involved in the design scalable workload-driven partitioning and mathematical formulation of it. SA carried out implementation of scalable workload-driven partitioning scheme. RI have been involved in drafting the manuscript and revising it critically. Both authors read and approved the final manuscript.

Authors' information

Swati Ahirrao received her M.E (Computer Engineering) from Pune University and pursuing Ph.D. from Symbiosis International University. Her reasearch interests include cloud computing, database systems, distributed systems. She works as Asst. Professor in Symbiosis Institute of Technology under Symbiosis International University and is a research student at the same university. Rajesh Ingle is a Dean and Professor (CSE), at Pune Institute of Computer Technology, and Head, EDC and Business Incubation Centre. He has received Ph.D. Computer Science and Engineering from Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Powai, Mumbai. He has received the B.E. Computer Engineering from Pune Institute of Computer Technology, University of Pune, and M.E. Computer Engineering from Government College of Engineering, University of Pune. He has also received M.S. Software Systems from BITS, Pilani, India. His research interests include distributed system security, grid middleware, cloud computing, multimedia networks and spontaneously networked environments. He has more than 20 research publications in conferences and Journals. He has authored four books. He is a senior member of the IEEE, IEEE Communications Society, and IEEE Computer Society. He is working as Chairman, IEEE R10 SAC.

Acknowledgements

We sincerely thank Dr. Arundhati Warke and Prof Vijaykumar Jatti for their valuable suggestions.

Author details

[1] Symbiosis International University, Pune, India. [2] Pune Institute of Computer Technology, Pune, India.

Received: 8 July 2015 Accepted: 29 October 2015
Published online: 15 November 2015

References

1. HBase:Bigtable-like structured storage for Hadoop HDFS. (2009) http://hbase.apache.org, [Accessed 23-Sep-2013]
2. Amazon.com AmazonSimpleDB (2012). http://aws.amazon.com/simpledb [Accessed 23-Aug-2013]
3. Chang F, Dean J, Ghemawat S, Hsieh WC, Wallach DA, Burrows M, Gruber RE (2008) Bigtable: A distributed storage system for structured data. In: Proceedings of the 7th Symposium on Operating Systems Design and Implementation. USENIX Association, Berkeley, CA. pp 205–218
4. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Vogels W (2007) Dynamo: amazon's highly available key-value store. In: Proceedings of the 21st ACM Symposium on Operating System Principles. ACM, New York. pp 205–220
5. Cooper BF, Ramakrishnan R, Srivastava U, Silberstein A, Bohannon P, Jacobsen HA, Yerneni R (2008) PNUTS: Yahoo!'s hosted data serving platform. Proc VLDB Endowment 1(2):1277–1288
6. Grolinger K, Higashino WA, Tiwari A, Capretz MAM (2013) Data management in cloud environments: Nosql and newsql data stores. J Cloud Comput: Adv Syst Appl 2(1):22
7. Vogels W (2007) Data access patterns in the amazon. com technology platform. In: Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment. pp 1–1
8. Baker J, Bond C, Corbett J, Furman JJ, Khorlin A, Larson J, Léon J-M, Li Y, Lloyd A, Yushprakh V (2011) Megastore: Providing scalable, highly available storage for interactive services. In: CIDR, vol 11. pp 223–234
9. Bernstein PA, Cseri I, Dani N, Ellis N, Kalhan A, Kakivaya G, Talius T (2011) Adapting microsoft sql server for cloud computing. In: Proceedings of the 27th International Conference on Data Engineering. pp 1255–1263
10. Curino C, Jones E, Zhang Y, Madden S (2010) Schism: a workload-driven approach to database replication and partitioning. Proc VLDB Endowment 3(1–2):48–57
11. Das S, Agrawal D, El Abbadi A (2013) ElasTraS: An elastic, scalable, and self-managing transactional database for the cloud. ACM Trans Database Syst (TODS) 38(Article 5):1–45
12. Sandholm T, Lee D (2014) Notes on cloud computing principles. J Cloud Comput 3(1):1–10
13. Das S, Agrawal D, El Abbadi A (2009) Elastras: An elastic transactional data store in the cloud. In: Proceedings of the 1st USENIX Workshop on Hot topics on Cloud Computing. USENIX Association, Berkeley, CA. pp 1–5
14. Curino C, Jones EPC, Popa RA, Malviya N, Wu E, Madden S, Zeldovich N (2011) Relational cloud: A database-as-a-service for the cloud. In: Proceedings of the 5th Biennial Conference on Innovative Data Systems Research. pp 235–240
15. Das S, Agrawal D, El Abbadi A (2010) G-store: a scalable data store for transactional multi key access in the cloud. In: Proceedings of the 1st ACM Symposium on Cloud Computing. ACM, New York. pp 163–174
16. Aguilera MK, Merchant A, Shah M, Veitch A, Karamanolis C (2007) Sinfonia: a new paradigm for building scalable distributed systems. In: Proceedings of the 21st ACM Symposium on Operating System Principles. ACM, New York. pp 159–174
17. Zhou W, Guillaume P, Chi-Hung C (2011) Cloud TPS: Scalable transactions for web applications in the cloud. IEEE transactions on service computing 5(4):525–539
18. Levandoski JJ, Lomet DB, Mokbel MF, Zhao K (2011) Deuteronomy: Transaction support for cloud data. In: CIDR Vol. 11. pp 123–133
19. Ahirrao S, Ingle R (2013) Scalable transactions in cloud data stores. In: Advance Computing Conference (IACC): IEEE 3rd International conference. pp 116–119
20. Transaction Processing Council. TPC benchmark C standard specification, revision 5.11. http://www.tpc.org/tpcc/, [Accessed 23-Aug-2013]
21. Amazon.com AmazonEC2 (2010). http://aws.amazon.com/ec2, [Accessed 23-Aug-2013]
22. Das S, Agarwal S, Agrawal D, El Abbadi A (2010) Elastras: An elastic, scalable, and self managing transactional database for the cloud. Technical report, Technical Report 2010-04, CS, UCSB
23. Schwartz B, Fortune E (2010) Forecasting MySQL Scalability with the Universal Scalability Law. https://www.percona.com/files/white-papers/forecasting-mysql-scalability.pdf [Accessed 23-June-2013]