Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

GRAPH BASED WORKLOAD DRIVEN PARTITIONING SYSTEM FOR NOSQL DATABASE

[1]SHIVANJALI KANASE, [2]SWATI AHIRRAO
[1]Symbiosis International University, Department of Computer Science, Pune
[2]Symbiosis International University, Department of Computer Science, Pune
E-mail: [1]shivanjali.kanase@sitpune.edu.in , [2]swatia@sitpune.edu.in

ABSTRACT

Cloud computing is the rising technology, which deploys the scalable web applications and provides the storage solutions to the individuals and enterprises. The massively scalable NoSQL data store exhibits the highly desirable scalability and availability by using various partitioning methods, to handle the exponentially growing demands of the users. The data stores are partitioned horizontally and distributed across the geographically dispersed data centers to balance the load. However, if the partitions are not formed properly, it may result in expensive distributed transactions. There are number of existing partitioning techniques available like Hash and Range partitioning, but they are not effectively applied to the workload consisting of fairly small transactions, that touch fewer records and which also do not considers the n-to-n relationship between them. In this paper, "Graph-based Workload Driven Partitioning System for NoSQL Database using OLTP workload" is proposed to improve the scalability minimizing the distributed transaction. This system works in two phases: 1) generates the graph from transaction workload such as the database tuples are represented as nodes and number of tuples accessed in same transaction are connected by edges. Then graph partitioning technique is applied to find min-cut balanced partition. 2) Uses the decision tree classifier, which gives a range predicate partitioning identical to the partitions, formed by the graph partitioning algorithm. The TPC-C schema is implemented by utilizing Apache CouchDB NoSQL data store, in order to assess the proposed system. The experimental end result indicates that the graph-based schemes significantly improves the response time and latency.

Keywords: *Graph Partitioning Methods, OLTP Workload, Scalability, Distributed Transactions, NoSQL Database, Decision Tree Classifier, Lookup Table.*

1. INTRODUCTION

The emergence of Big Data, Internet of Things and Hybrid Cloud applications enable the business to perform faster, smarter and more efficiently than ever. The companies need vigorous foundation for data handling and running machine critical applications in order to harness the competitive edge. This tremendous growth of unstructured data and new applications has caused to an evolution in database management companies. Nowadays these companies are shifting to the NoSQL database from traditional relational database, as they lack the flexibility and ability to scale, to handle globally massive amounts of unstructured data. Nowadays people are equalizing NoSQL with the scalability.

NoSQL database can be scale out efficiently by utilizing various partitioning methods. With the objective of improving scalability, partitioning provides availability, easy maintenance and improvised query performance to the database users. In order to increase the scalability of applications, many cloud providers, partition data and distribute these partitions across geographically dispersed data centers, to balance the load. However, if the partitions are not correctly formed, it may result in expensive distributed transactions.

The existing partitioning schemes like Round-robin, Range and Hash partitioning are inefficiently applicable to the workloads having of small transactions, which touch a couple of records [1]. Moreover, these approaches also ignore the relations among the database tuples and end up with the cluster of unrelated tuples on the same partition which eventually results in the costly distributed transactions. In OLTP workloads to ascertain transactional properties, distributed transactions should use a distributed consensus protocol, which might result in immoderate and undesirable

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

latencies, decreased throughput and network messages [2]. Hence, it is necessary to find partitioning technique, which will consider the relativity among the tuples while partitioning and minimize the adverse effects of multisite transactions.

The proposed system frames the NoSQL database workload in the form of graph wherein, the nodes represent the database tuples and each edge connects a couple of tuples accessed inside the same transaction. Graph partitioning technique is applied to find min-cut k balanced partitions which minimizes the multisite transactions. Unless the workload characteristics change drastically, and tuples from a single partition are associated with one another, the events and unfavorable consequences of Distributed Transactions are reduced rapidly. Lastly, the decision tree classifier techniques are applied to extract the set of rules to explore the predicates, which explain the graph partitioning strategy. This step improves the runtime of the query processing.

The strengths of proposed system are: i) Not dependent on any schema layout. ii) Can be implemented in the social networking data stores as it considers the n-to-n relationship between data tuples. iii) Fine-grained approach for NoSQL database partitioning. This system is basically designed for OLTP workload using TPC-C schema. In addition to this new partitioning approach, the proposed system offers some extra contributions:

- A design is presented based on decision tree classifier which gives range predicate partitioning similar to graph partitioning algorithm and using this lookup table is a built to improve query performance.
- A mapping of TPC-C schema to Apache CouchDB NoSQL data store is presented.
- An integration of Apache CouchDB is done with Neo4j NoSQL graph database for representing OLTP workload in the form of a graph.
- METIS graph partitioning algorithm is implemented on the graph stored in Neo4j graph database.
- The system takes a reasonable time that is just a few minutes to partition the million tuple dataset. Some heuristics like Transaction level sampling, Tuple level sampling and Relevance filtering are proposed, to diminish the size of the graph to additional improve partitioning time of largely increasing graph.
- The practical implementation of graph-based workload driven partitioning system on Apache CouchDB is presented and its performance is evaluated using TPC-C schema.

Further this paper is structured as follows: In Section 2, papers related to scalability and graph database partitioning is discussed. Section 3 gives brief overview of proposed system. Design of graph based workload driven partitioning system is presented in Section 4. An Implementation details in Section 5 explains implementation and the performance evaluation of the partitioning system. In Section 6 and Section 7, experimental setup and results are described respectively. Finally, Section 8 concludes the paper.

2. RELATED WORK

For more than few years, Scalable and distributed database administration have been the area of interest of the database research group. Accordingly, a plenty of systems along with techniques for scalable database transaction have been documented in the literature. This chapter discusses related work done on different partitioning algorithms and graph partitioning techniques.

2.1Workload-driven Partitioning Algorithms

Data-driven partitioning approach called 'Schism' is first introduced by Curino et al. [1], for OLTP databases. Schism models the workload as a graph and enforces k-way min-cut graph partitioning algorithm to lessen the effect of distributed transactions and increases the throughput. However, the graph generated by 'Schism' is extremely massive and usually does not deal with progressive workload deviation and repartitioning.

Sudipto Das et. al presents " ElasTras [3], which is scalable and elastic to handle the varying amount of loads with the additional functionality of fault-tolerance and self-management. ElasTras achieves the scalability by partitioning the large database, using schema level partitioning which restricts the transactional updates to a specific database partition. The associated rows are grouped together in a singular partition using schema patterns. Despite the fact that the system possesses the various functionalities, however, it lacks in

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

replication mechanism to benefit the high availability.

J. Baker, et. al has developed a Megastore [4] which is a scalable repository for interactive online systems. Megastore integrates the scalability of a NoSQL with regard to RDBMS and offers the strong consistency and high availability. Megastore gives full ACID properties inside the partitions. This system partitions the data into a stock of entity group that is the set of correlated data items which independently replicated over the set of servers and each update is replicated synchronously across the identical partitions with acceptable latency. The transaction which requires the strong consistent view of the database to fulfill its execution restricts its execution to a single entity group.

Ahirrao S. [5] invented a scalable workload-driven data partitioning scheme specifically designed for the NoSQL data store based on the analysis of data access patterns. The access patterns are examined by monitoring the web access logs. Based on these access patterns, partitions are formed which can be changed adaptively, to balance the load among all partitions.

2.2 Graph Partitioning Methods

Graph partitioning problem has been the area of interest of many researchers in last few decades, which was then reviewed and applied in many disciplines. The major objective function of graph partitioning is to find balanced quality partitions which reduce the edge cut. There have been many solutions available to handle the challenge of discovering the balanced graph partitions which itself is the NP-complete problem.

JA-BE-JA [6] is distributed graph partition algorithm which uses sampling and swapping techniques to see the balanced partitions in graph. Every node is processed individually and only the immediate adjacent of the vertex and a small group of arbitrary vertices are needed to be known locally. This algorithm does not require the entire graph at the once, and processes only with the partial information, hence, it is more efficient for computing very big graphs. Originally, every vertex is put into the random partitions and over the time, they adaptively swap with the other vertices to collocate the vertex with their highly related neighbors in one partition to improve the scalability. The swapping cost is also increased when the number of partitions increases. Multilevel algorithms for partitioning graphs were first described by the Karypis and Kumar [7]. Typically such multilevel schemes combine the vertices and edges using matching techniques, to decrease the size of the graph. This procedure recursively iterates till the enough small size graph is formed. Then this reduced graph is initially partitioned into the k partitions which iteratively refines to find the k partitions of the initial graph. It uses graph coloring techniques to parallelize the procedures of the multilevel partitioning algorithm, to provide the high degree of concurrency.

A large community of the researchers examined their partitioning designs on traditional databases, yet there is a requirement for to build up an effective partitioning framework for NoSQL database which enables it to scale greatly. Previously the graph partitioning techniques were used in combination with the decision tree classifier to improve the performance of SQL database. Here, this exploration work concentrates on the same magnificent integration of graph partitioning techniques and machine learning technique like decision tree classifier, along with some improvisation on NoSQL database using TPC-C workload.

3. PROPOSED SYSTEM OVERVIEW

Definition 1. *Workload Driven Partitioning System:*

Workload driven partitioning systems are the system in which the database is partition based on the relativity of tuples and their access patterns. The related tuples mean the numbers of tuples which are accessed together in most of the transaction are placed together in one partition.

The high-level architecture of the system is shown in Figure 1. The proposed system is designed to work as a scalable transactional layer on the top of any distributed storage system such as CouchDB or Amazon SimpleDB which implements CRUD (create/ insert, read, update, and delete). Operations. The users interact with the distributed databases as if it is a centralized database system. The system has 2 tier architecture: 1) backend database nodes and 2) coordinator node. The backend database nodes are the NoSQL database servers (CouchDB Database server) which actually handle the data. The coordinator node contains query routers and the lookup table. A client sends requests to a query router, which forwards it to the partitions with the help of lookup table and returns

Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the responses to the clients. The queries from multiple clients are processed in a parallel way by the query router which also dispatches sub-queries in a parallel manner to the backend. More coordinator nodes can be added to distribute the transaction load. The coordinator node is connected to the number of geographically dispersed data nodes where the actual database partitions are placed. Individual data node is synchronously replicated as Master-Master to assure the high availability, thus here in this system the tuple level replication is not explicitly handled. The coordinator node has the partitioning metadata and the network address for each backend data node. The partitioning metadata contains the pattern of division of the individual tuples into the partitions and the patterns of the mapping of these partitions to the backend nodes.

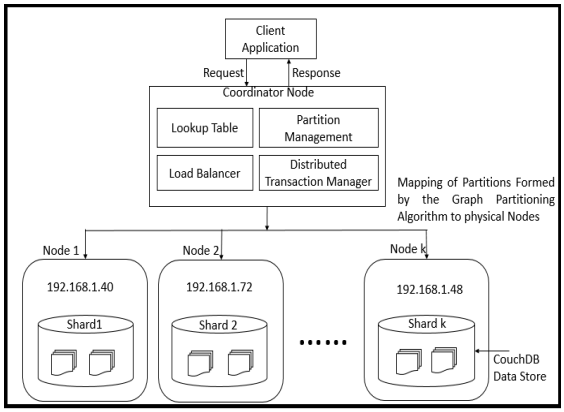


Figure 1. System Architecture

Table 1. Gives a brief overview of our graph-based data-driven partitioning system. The input to the graph-based workload driven partitioning system is workload trace and a number of partitions required, the output is the balanced partitions, which minimize the overall cost of running the workload, with reducing expensive distributed transactions. The basic process is outlined in following steps:

Workload pre-processing: The system takes the input as the workload trace that is the collection of transactions and the tuples they access. Each transaction is processed to extract the set of read and update sub-queries contained in it. For each sub-query, the unique ids of individual tuples are retrieved.

Graph Representation: The tuples accessed in the input transaction are represented as nodes in the graph. The number of tuples accessed in a single transaction is connected by edges. The sampling techniques are used to reduce the size of the graph.

Graph Partitioning: The graph partitioning algorithm is applied on the created graph. The output of the graph algorithm is a high-quality balanced k partition. The individual node from the graph is mapped to the unique partition, such that the maximum of the adjacent edges is collocated in the single partition. Then, these k partitions are assigned to the k physical nodes.

Lookup Table Creation: Lookup table stores the (tuple, partition) pair generated by the graph partitioning. Basically, Lookup table is used for forwarding the query to correct partition, as it specifies which tuple is stored in which partition.

Decision Tree Classifier: Sub-queries from transactions are analyzed to accumulate the collection of frequently accessed attributes in the WHERE clauses. The set of rules are extracted using the decision tree classifier, which gives range predicates partitioning based on the frequent attribute values. This resultant partitioning is identical to the graph partitioning.

The resulting data partitioning strategy can be explicitly added into any NoSQL database, which supports partitioning for scaling database in distributed shared-nothing architecture.

Table 1. System Overview

Step	Algorithm	Output
1) Data Preprocessing	Computes read and write sets for each transaction in workload.	(tuple ID , transaction)
2)Graph Representation	Star-shaped representation	Node-> Tuples(Rows) Edges-> Connects two tuples if they are accessed by same transaction.
4)Graph Partitioning	METIS algorithm	Fined grained mapping between Individual nodes and partition labels.
5)Lookup Tables	Indexes	(value, label) pairs.
6)Decision tree classifier	Weka's correlation based feature selection	Predicate Based range partitioning.

Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

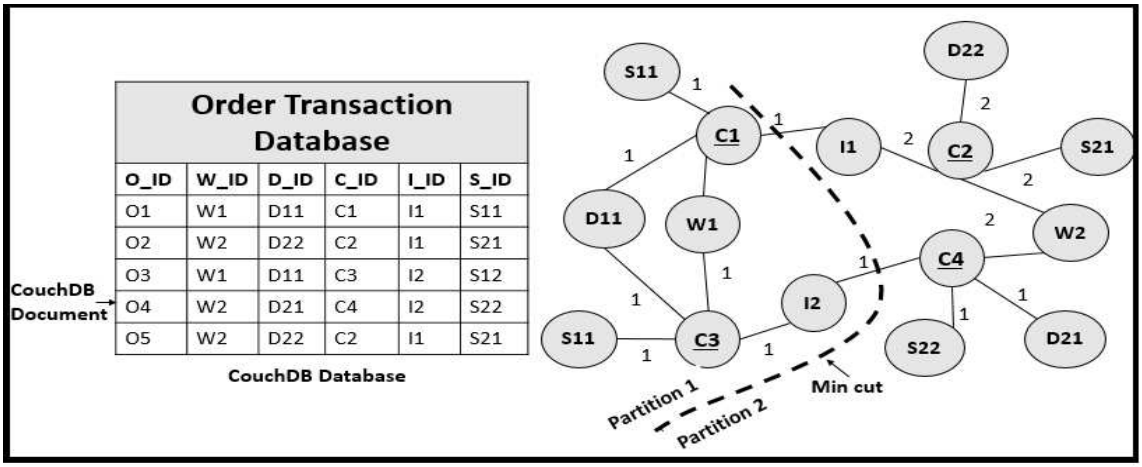


Figure 2. Graph Representation of Transactional Workload Stored in NoSQL Database

4. DESIGN OF GRAPH BASED WORKLOAD DRIVEN PARTITIONING SYSTEM FOR NOSQL DATABASE

There are various NoSQL database management systems available based on the key-value, column family and document-oriented. For designing the proposed approach Document oriented NoSQL database (Apache CouchDB) has been selected as it is more suitable to handle the transactional TPC-C based e-commerce application and provides a semi-structured schema for storing the data.

4.1 Graph Representation

The transaction workload can be represented in the form of a graph. The graph representation approach is presented here with an example. Although given example contains only one transaction database, this approach works on any number of transaction databases with any schema layout. The example has one transaction database called Order as shown in Figure 2. When the new record is added into the document-oriented database, it is stored as the new document with a unique id. This Order transaction database records the details of five order transaction executed on different databases in the CouchDB. Each transaction is inserted into the Order database as the new document with unique O_ID. Each order transaction document has the following attributes: (O_ID, W_ID, D_ID, C_ID, I_ID, S_ID). The attributes of each document is represented as the nodes in the graph, which are connected by the edges. The C_ID node is considered as the base node and all other attribute from transaction of that customer are connected to it with edges. The edge cost indicates the total number of the transaction which co-accesses this pair of tuples. (Example: attributes with value (W2, D22, C2, I1, S12) are accessed in O1 and O2 transaction, the cost of edge between them is 2). When the new transaction occurs, it first checks the presence of the node for the particular attribute value in the graph. If absent then, the new node for that attribute value is added to the graph and respective cost on edges is also updated.

4.1.1. Graph size reduction

The functional graph partitioning system can handle huge databases. As the number of transaction increases, the number of nodes represented in the graph is also increased with increasing the graph representation [1]. The additional partitions are required for effective handling of a large database. It results in the need of finding more cuts into the graph. This increases processing overhead and runtime of the algorithm. Hence, there is a need for reducing the size of the graph. As the reduced graph contains insufficient information to form high quality partitions, the number of heuristics for graph size reduction have been introduced, which considerably increases the runtime of the algorithm, with less effect on the quality of partitions. The following Sampling Techniques for graph size reduction have been implemented in the proposed system:

Transaction-level sampling: It reduces the number of edges by limiting the number of transactions represented in the graph. As the TPC-C transaction contains almost 45% of the transaction as New Order Transaction (explained in section 5.2.1), Only the New Order Transaction workload has been represented in the graph.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Tuple-level sampling: It limits the number of tuples shown in the graph.

Relevance filtering: It removes rarely accessed tuples from the graph as they give a little information. For example, O_ID is not represented in the graph as it is unique for each transaction and carries less partitioning information.

These samplings techniques are validated to be powerful in decreasing the graph size for concerned TPC-C workload at the same time keeping high-quality results.

4.2 Graph Partitioning

The system represents the transactional workload in the form of the graph. The graph partitioning algorithm divides the nodes from the graph into k balanced partitions such that the number of edges crossing the partitions is minimized. This graph partitioning problem has many applications in various fields like VLSI design, parallel scientific computing, sparse matrix reordering. However, this problem is identified as NP-complete [9]. In last decades, many multilevel schemes are introduced to solve this problem. For finding solutions to the k way partitioning problem Recursive bisection method has been used. First it combines the random vertex and edges using the matching technique to form the smaller graph. This combined smaller graph is bisected initially, on which refinement algorithm is applied to form the two balanced partitions of the original graph, in such a way that it has min-cut. These steps are repeated until the k partitions of the original graph are formed. This whole process is composed of three phases as shown in Figure 3. Different methods for these phases are described in many papers [8, 9, 10]. We have selected some of this methods with minor changes in it. These phases are illustrated in detail as follows:

Coarsening phase: The fundamental process in this phase is to combine the nodes and edges to form the smaller graph. First, from the graph representation, the adjacency matrix is formed. For each base node in the graph, all adjacent edges on it are sorted into the decreasing order of their cost and added into the queue. At each level, the first node from the queue is retrieved and merged with the base node. Once the node is merged into the base node it is marked as matched and it cannot be added to another base node. This merging of node continues until the small enough graph is formed. The underlying merging technique is called as Heavy Edge Maximal Matching. This coarsening technique has the property that the balanced partitioning of the small graph is identical to the balanced partitioning of the original graph.

Initial partitioning Phase: In the second phase, the smaller graph is partitioned into two parts such that each of the partitions has equal node weight. The equal number nodes to each partition is assigned for balancing the partition. On this initial partition, the refinement algorithm is applied to find the final fine-grained partitions.

Uncoarsening Phase: Many refinement algorithms are variants of the popular Kernighan–Lin (KL) partitioning algorithm [11]. The initial partitioning is assumed to be the best possible partition for the higher level graph. The KL algorithm iteratively swaps the subsets of a node from one partition with the same number of nodes from another partition in such a way that the edges crossing the partition have the minimum cost. This algorithm gives the best possible local optimum solutions.

These phases are recursively applied till the k balanced partitions of the original graph is generated. The output of the whole partition system is the fine-grained mapping between the nodes and partition labels. As highly correlated nodes are placed into one partition, this helps in minimizing the distributed transaction and increases the system performance.

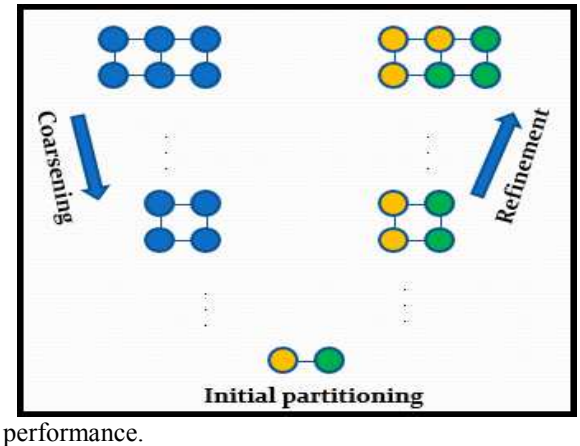


Figure 3. Multilevel Recursive Bisection

4.2 Lookup Table Creation

A lookup table as shown in Table 2 is used for storing the location of each node. This lookup table is present at the coordinator node and helps

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the router to forward the incoming query to the appropriate physical node [12]. The router parses the incoming query and retrieves the predicate over the attribute values from the WHERE clause and compares it with the lookup table entries, to find the location of the attribute and execute the query on the correct backend. This lookup table Structure effectively increases the throughput of the system. The lookup table needs to be stored in the RAM memory of the router [1]. It stores the mapping of unique key to the partition label. Each row in the lookup table requires 10 bytes of the memory out of which 8 bytes are for the unique key and 2 bytes for the partition label. With this memory requirement, the RAM of the single machine is sufficient to handle the OLTP applications [12]. If lookup table does not fit into the RAM of the single machine, it can be distributed to several machines. When a new tuple is to be inserted into the database, it is first inserted into the random partition and eventually it migrates to its correct partition. This technique is efficient when the fine-grained partition is wanted but it is not suitable for the insertion heavy databases as it increases the network overhead.

One way to reduce the memory requirements of the lookup table is to map the Fine-grained mapping of lookup table with memory efficient range predicate partitioning. For this purpose, Analysis tool is used which gives the range predicate partitioning identical to the graph partition.

Table 2. Lookup Table generated form the graph in Figure 2.

Lookup Table		Lookup Table	
ID	Partition Label	ID	Partition Label
W1	1	C1	1
W2	2	C2	2
D11	1	C3	1
D21	2	C4	2
D22	2	S11	1
I1	2	S12	1
I2	1	S21	2
		S22	2

4.3 Explanation Phase Using Decision Tree Classifier

The output of the partitioning phase is the fine-grained mapping between attribute values and the partition table. The compact model for the generation of the lookup table is required which stores the minimum number of mapping such that the location of the target attribute can be predicted [13]. For that purpose, Decision Tree Classifier is used which generates the range predicate partitioning based on rules generated by the decision tree. Decision tree classifier takes the set of (attribute value, partition label) pairs formed by the graph partitioning as input and generates the decision tree of the predicates over the attribute values going down to leaf nodes with particular partition labels. The location of the unlabeled attribute value can find by traversing the tree downwards and applying predicates at each level till the leaf node with partition label is reached.

The set of the rules generated from the decision tree captures the core of the graph partitioning in the compact form. For example from Figure 2. rules identified by decision tree are:

(W_ID =W1) → Partition=1

(W_ID = W2) → Partition=2

As the every generated rule is not useful, the rules which are generated based on the frequently accessed attribute from the WHERE clause are considered. (e.g., in this application more than half of the queries use W_ID attribute from their WHERE clause to route the transaction to the appropriate partition and minimize the distributed transaction.) The W_ID act as the reference key for (Customer, District, Order, New Order, Order Line, Stock, History) tables of TPC-C (Explained in the section 5.2.1), hence this table can be classified according to the value of W_ID using the decision tree classifier.

5. IMPLEMENTATION DETAILS

In this Section, System implementation details is presented followed by the experimental evaluation of graph partition algorithm in Apache CouchDB. The implementation of graph based workload driven partitioning system in NoSQL database is actually a challenge and requires some modifications. Here additionally, some of the design decisions made during the system implementation are discussed.

Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

5.1 System Implementation

5.1.1 Integration of Apache CouchDB with Neo4j NoSQL graph database

Neo4j is the open source graph database completely developed in java [17]. This schema-free NoSQL graph database has been utilized for the graph representation of the transactional workload stored in CouchDB database. This two NoSQL databases have been integrated to enable automatic synchronization of the data in between them. When new order transaction is executed on the CouchDB, new node is being created automatically in Neo4j graph database. The graph partitioner takes the advantages of the relationships provided by the Neo4j to collocate the related tuples.

5.1.1 Implementation of Range Predicate Partitioning based on Decision Tree Classifier

The popular suite of machine learning that is Weka 3 has been used to get range predicate based explanation of the graph partitioning.

Training set creation: The system extracts the sub-queries from each transaction and attribute value accessed in transaction workload. Each accessed attribute value is labeled with the partition name generated by the graph partitioning. This training set is given as the input to the decision tree classifier to identify the candidate attributes.

Selection of Attribute: System parses the sub-queries and counts the frequency of each attribute in its WHERE clause. The attribute with less frequency is discarded as it carries less information for routing query to the right partition. For example, in TPC-C for a stock table the two frequently accessed attributes (S_W_ID, S_I_ID) is got. These candidate attributes are inserted into Weka's correlation based feature selection to calculate the set of attributes which are correlated with the partition label of the candidate attributes. The candidate attribute which has a high number of correlated attribute with same partition label are selected as classification attribute and discards the remaining. The selected attribute is used for building the decision tree in which it acts as the root node and classifies the target attributes to get their location.

Build the Decision Tree Classifier: For constructing decision tree classifier J48 has been used which is java implementation of C4.5 Classifier algorithm [18]. The classification attributes are the partition labels which has to be learned from the candidate attribute. The rules generated from this decision tree gives the range predicate partitioning identical to the graph-based partitioning. The rules with less support are discarded to avoid the risk of over-fitting as they are not that much useful. The classifier rules indicates that all tuples from the TPC-C Item table have to be replicated on all partitions as it don't have reference to other TPC-C tables and can be placed independently on each partition. The overall result of the range predicate partitioning is to classify the database according to W_ID as it is the frequently accessed attribute in all transaction with highly correlated with other attributes and the item table is replicated on each partition.

5.2 Performance Evaluation

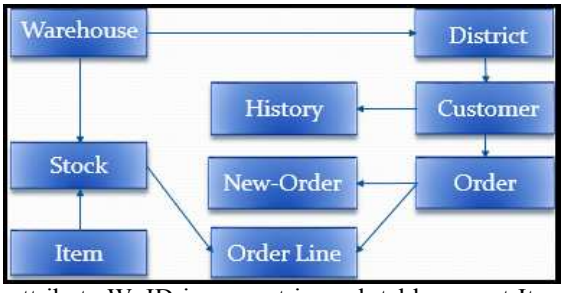
The performance of the system has been evaluated by using TPC-C schema.

5.2.1 TPC-C benchmark

TPC-C benchmark is an authorized yardstick for analyzing OLTP web applications [15]. It stimulates OLTP Workload for the E-commerce web application. There are total five transactions, such as New Order, Payment, Order Status, Stock Level, and Delivery. Every transaction contains read and update sub-queries. These 5 transaction are executed on the 9 tables of the TPC-C which are: Warehouse, District, Customer, Order, New Order, Order Line, History, Stock, and Item as shown in a Figure 4. In a real life scenario, typically 45% transactions are New Order, 43% transactions are Payment and 4% transactions are Delivery, Order Status and Stock [5].

5.2.2 Mapping of TPC-C to Apache CouchDB

The TPCC schema are primarily designed for the Relational database with the nine tables. These 9 tables of TPC-C have been mapped to the 9 documents of CouchDB. CouchDB is the document-oriented NoSQL database which completely embraces the Web [16]. It stores the data hierarchically in semi-structured format. Each document, similar to the row from the relational database is named uniquely with the ID. The



attribute W_ID is present in each table except Item table.

Journal of Theoretical and Applied Information Technology
15th July 2016. Vol.89. No.1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 4. Components of TPC-C

6. EXPERTIMENTAL SETUP

The proposed system is implemented using java based NetBeans IDE and some tests are carried over it. Where 3 machines are considered for the experiments which have core i3 processor with 2GB of Primary memory for the distributed paradigm. All machines are interconnected through D-Link 8 Port Ethernet. Proposed System uses NoSQL CouchDB for the graph based workload driven partitioning system. The Neo4j graph database is used to stimulate the transactional workload in the form of graph. The developed system is tested in many scenarios to prove its accuracy as mentioned in below tests.

7. RESULTS

In this section, the performance of the graph based workload-driven partitioning is extensively evaluated and compared with a TPCC schema. The goal of this experiment is to validate the scalability of system with varying number of concurrent users. The scalability of the system is measured in terms of response time. Figure shows response time of the Graph Based Workload Driven partitioning system and the TPCC schema. Along x-axis, number of concurrent users and along the y-axis, a response time (time in ms) is plotted. As observed from below Figures, our system has lesser response time than TPC-C schema.

7.1 Response Time Comparison between TPC-C Schema and Graph Based Workload Driven Partition

The response time comparison for all TPC- C transactions can be depicted in the figures given below:

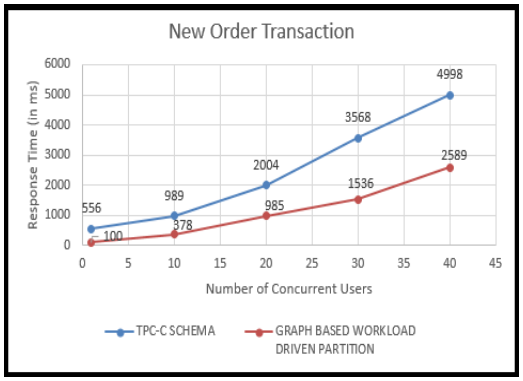


Figure 5: Response Time Comparison for New Order Transaction

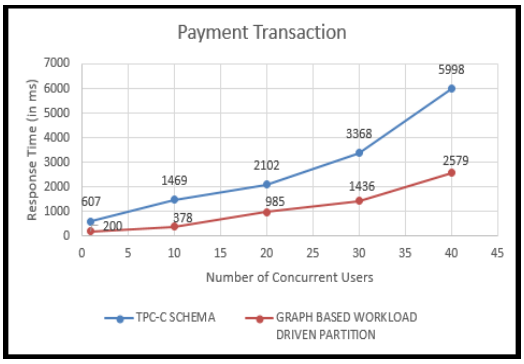


Figure 6: Response Time Comparison for Payment Transaction

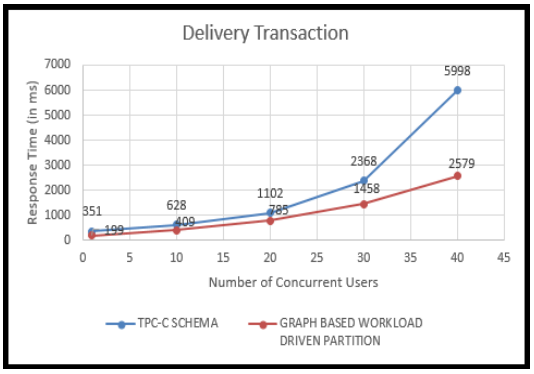


Figure 7: Response Time Comparison for Delivery Transaction

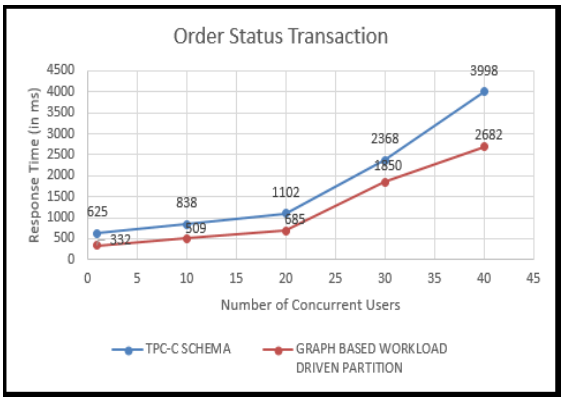


Figure 8: Time Comparison for Stock Level Transaction

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

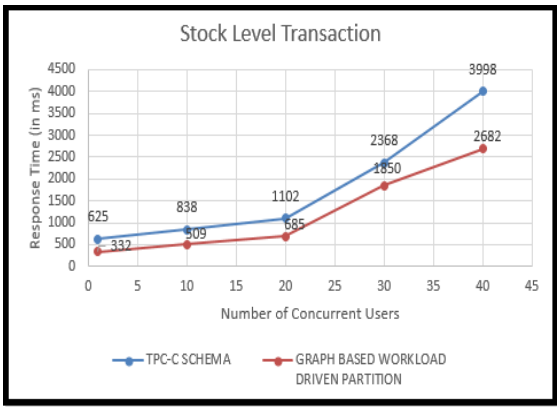


Figure 9: Response Time Comparison for Order Status Transaction

In all the above graphs it indicates that transactions time is diminished as the Database is partitioned and kept in different machines in distributed paradigm. This clearly depicts that the proposed method is efficiently incorporated and the scalability is achieved.

8. CONCLUSION

In this paper, a graph based workload driven partitioning system for NoSQL database is presented. The proposed system represents the workload in the form of the graph, to take the advantage of the relationships between them. Using the relationship characteristics between the tuples, the highly connected tuples are combined together in a single partition which reduces the distributed transactions. Some heuristics like sampling techniques are introduced to optimize the graph representation and to improve the performance of the system. The compact model for lookup table based on range predicate partitioning is also generated using decision tree classifier. TPC-C schema used for the evaluation of the system shows that the proposed system improves the response time which is considerably less than the TPC-C schema. The resulting data partitioning strategy can be explicitly added into any distributed shared-nothing NoSQL database to enhance its scalability.

REFERENCES

[1] Curino C, Jones E, Zhang Y, Madden S. Schism: a workload-driven approach to database replication and partitioning. Proceedings of the VLDB Endowment. 2010 Sep 1;3(1-2):48-57.

[2] Quamar A, Kumar KA, Deshpande A. SWORD: scalable workload-aware data placement for transactional workloads. In Proceedings of the 16th International Conference on Extending Database Technology 2013 Mar 18 (pp. 430-441). ACM.

[3] Das S, Agrawal D, El Abbadi A. ElasTraS: An elastic, scalable, and self-managing transactional database for the cloud. ACM Transactions on Database Systems (TODS). 2013 Apr 1;38(1):5.

[4] Baker J, Bond C, Corbett JC, Furman JJ, Khorlin A, Larson J, Leon JM, Li Y, Lloyd A, Yushprakh V. Megastore: Providing Scalable, Highly Available Storage for Interactive Services. InCIDR 2011 Jan 9 (Vol. 11, pp. 223-234).

[5] Ahirrao S, Ingle R. Scalable transactions in cloud data stores. Journal of Cloud Computing. 2015 Dec 1;4(1):1-4.

[6] Rahimian F, Payberah AH, Girdzijauskas S, Jelasity M, Haridi S. Ja-be-ja: A distributed algorithm for balanced graph partitioning.

[7] Karypis G, Kumar V. Parallel multilevel series k-way partitioning scheme for irregular graphs. Siam Review. 1999;41(2):278-300.

[8] Karypis G, Kumar V. multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed computing. 1998 Jan 10;48(1):96-129.

[9] Hendrickson B, Leland R. A multi-level algorithm for partitioning graphs.

[10] Portugal D, Rocha RP. Partitioning Generic Graphs into k Balanced Subgraphs. In Proceedings of the 6th Iberian Congress On Numerical Methods in Engineering (CMNE 2011), Coimbra, Portugal 2011 Jun 14 (pp. 13-16).

[11] Kernighan BW, Lin S. An efficient heuristic procedure for partitioning graphs. Bell system technical journal. 1970 Feb 1;49(2):291-307.

[12] Tatarowicz AL, Curino C, Jones EP, Madden S. Lookup tables: Fine-grained partitioning for distributed databases. In Data Engineering (ICDE), 2012 IEEE 28th International Conference on 2012 Apr 1 (pp. 102-113). IEEE.

[13] Ranka S, Singh V. CLOUDS: A decision tree classifier for large datasets. In Proceedings of the 4th Knowledge Discovery and Data Mining Conference 1998 (pp. 2-8).

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[14] Kamal J, Murshed M, Buyya R. Workload-aware incremental repartitioning of shared-nothing distributed databases for scalable OLTP applications. Future Generation Computer Systems. 2016 Mar 31;56:421-35.
[15] Council TP. TPC benchmark C (standard specification, revision 5.11), 2010. URL: http://www. tpc. org/tpcc.
[16] Anderson JC, Lehnardt J, Slater N. CouchDB: the definitive guide. " O'Reilly Media, Inc."; 2010 Jan 19.
[17] Miller JJ. Graph database applications and concepts with Neo4j. In Proceedings of the Southern Association for Information Systems Conference, Atlanta, GA, USA 2013 Mar 23 (Vol. 2324).
[18] Jain D. A Comparison of Data Mining Tools using the implementation of C4. 5 Algorithm. International Journal of Science and Research Vol3. 2014 Aug(8).