RepliSmart: A Smart Replication framework for optimal query throughput in read-heavy environments

R K N Sai Krishna[†]
Teradata India
Hyderabad, India
rkn.sai@gmail.com

Chandrasekhar Tekur
Teradata India
Hyderabad, India
t.chandrasekhar@yahoo.com

Arnab Phani
Teradata India
Hyderabad, India
phaniarnab@gmail.com

ABSTRACT

Replication of data in the context of databases is a way to improve the performance of queries (throughput). An ecosystem where data is replicated can also result in increased parallelism. With replicated data, there would be better fault tolerance. In some cases, replicating a set of data only in few nodes for higher efficiency (in terms of space), could be a choice. A particular set of data could be replicated in many nodes while others in only few, based on the access ratio of the data. Today, the decision of what data to be replicated on which all nodes, is taken based on few presumptions at the time of replication. Once the data is replicated, it remains in those nodes. Over a period of time, the requirements/queries accessing a set of data might change, and it may happen that the data that is less replicated might be the most desired, and vice versa.

Another aspect to be considered is the storage format of the replicas. From the data storage perspective, columnar database could be a great choice for some applications, whereas row based option could be a better bid for another set of applications. Storing all the replicas in either of the storage formats would be inefficient. In this paper, we propose a framework, RepliSmart, in which there is a smart controller that redirects the incoming queries appropriately among the nodes connected, to balance the workload. The framework employs learning based on-demand replication, where in the number of replicas corresponding to a data unit (at a table or database level) vary as the data access patterns vary over a period. Additionally, the smart controller would dynamically define the storage format of a replica such that few of the replicas could be in columnar whereas the remaining in row based storage. The smart controller would redirect any of the user's requests to appropriate nodes based on the decision whether a query could be better executed on columnar data or row based. The proposed framework results in higher query throughput, and better space utilization for read-heavy query workloads.

[†]Primary author

CCS CONCEPTS

• Information systems---Database management system engines;

KEYWORDS

Columnar storage, dynamic replication, query throughput, query routing, databases, load balancing, data synchronization

ACM Reference format:

R K N Sai Krishna, Chandrasekhar Tekur and Arnab Phani, RepliSmart: A Smart Replication framework for Optimal query throughput in read-heavy environments, In *Proceedings of ACM India Joint International Conference on Data Sciences and Management of Data, Kolkata, India, January 3-5, 2019 (CoDS-COMAD '19), 7 pages.*
https://doi.org/10.1145/3297001.3297022

1 Introduction

High availability is one of the key factors that affect query throughput. It refers to a system where any given dataset is available for processing all the time. Redundant/replicated copies could be maintained across different nodes to eliminate single point of failure. To take advantage of the replicated copies, load balancing must be done so that the clients can be served at any of the given nodes that hold the desired replica. If the query workload mostly consists of read-only queries, then maintenance of the replicas could be relatively easy. If the query workload includes queries that modify the replicas, load balancing could get complicated, as consistency of replicas must be enforced. Bettina et al. [10] discuss the problems associated with replication and further propose a replication protocol suite addressing them, maintaining data consistency. Partial replication based on the access ratio of the data is considered in many cases for higher efficiency. Since not all data is equally accessed or of same priority, data that is more desired could be replicated in more nodes. In [11], Mesaac et al. propose a scalable storage system to effectively manage databases partially replicated.
On-Demand replication could be an area of predictive analytics dedicated to understanding user's demand for accessing a set of data. That understanding is harnessed and used to forecast user's demand. Knowledge of how demand will fluctuate empowers a framework to keep the right data on hand. If demand is underestimated, query execution can take long due to the lack of data at nodes that are ready to accept new requests. Understanding

demand and the ability to accurately predict it is imperative for efficient query execution. To be able to meet users' needs, appropriate replication is vital. Although no replication model is flawless, unnecessary costs stemming from too much or too little replication can often be avoided using data mining methods. Using these techniques, a business is better prepared to meet the actual demands of its users. In [20], Lee et al. show that for real workloads, Dynamic dAta Clustering (DAC) hot/cold separation policy performs best compared to Multiple Bloom Filter (MBF), and 2-level LRU (LRU) with reasonable memory overhead and computation. Park et al. [21], discuss Window-based Direct Address Counting (WDAC) algorithm for identifying hot data. Chen et al. [22], propose a Hot Data Catcher (HDCat), leveraging temporal locality for effective hot data identification.

In addition, the data storage format plays a crucial role in query performance, which ultimately affects the query throughput largely. Row based storage is usually a better choice for transaction processing, whereas columnar storage is best suited for analytical queries. Row based storage has the ability to modify a dataset very quickly. Columnar storage results in best performance for processing large volumes of data for a subset of attributes. In [12], Said et al. proposes an approach, which categorizes a given workload as either OLTP or DSS. The approach builds a classification model based on workload characteristics, and uses the same to identify change in workload type.

Abadi et al. [1], discusses the columnar database technology and solutions to various questions like row based system achieving column-store performance, can large-scale data intensive applications be better supported with column-stores etc. In [16], Bhagat et al. discuss pros and cons of Columnar and Row Oriented Databases. The studies of Kanade et al., Abadi et al., Wu Qiyue, Matei, David in [19],[3],[4],[2],[5] respectively focus on the advantages of columnar databases in the domain of data warehousing and business intelligence, and discuss various architectural features for adaptive indexing, vectorization, late materialization, compression, and join processing. For different kinds of DML queries, Kamal et al. [17] analyze the best suited storage (columnar or row based storage) format, evaluating the performance in terms of I/O, CPU, elapsed time and operator costs. Mike et al. [18], proposes a design of relational database that is read-optimized in contrast to the current write-optimized databases.

The biggest advantage one gets by storing data in a columnar database is that some of the queries become fast. For example, user wants to know the average salary of all the employees. Instead of looking up the salary for each record row by row in a row-oriented database, one can simply jump to the area where the "salary" data is stored and read just the data in need. While querying, columnar storage lets one skip over all the non-relevant data very quickly. Aggregation queries on columnar databases can be executed very efficiently compared to row oriented databases. In addition, compression algorithms work better for columnar databases.

In our work, we propose a framework RepliSmart, which has a smart controller that receives the incoming queries from various clients across different sessions, and routes the queries to appropriate nodes connected to the controller, for execution. Data is partially replicated across nodes, and the controller adjusts the number of replicas as per the access ratio, and defines the storage format of each of the replica.

The paper is organized as follows. In section 2, we describe the system model that we have adopted. It also introduces the needed notation and terminology. Section 3 summarizes the proposed scheme, especially the algorithm for adjusting the number of replicas of a data set as per the demand and the storage format of the replicas. In section 4, we illustrate our scheme using an example. Section 5 shows experimental results. Finally, section 6 summarizes the contributions of this paper and describes plans for further extensions.

TABLE I
NOTATION

Notation	**Description**
$D(t_1, t_2,....,t_m)$	database comprised of relational tables t_1, t_2, ..., t_m
$T(a_1,a_2,....,a_n)$	relation schema or set of attributes
n_i	node or database instance
C	smart controller
Q	query
N_i	set of nodes holding replica of a table t_i
N'_i	set of nodes with no replica of a table t_i
S_c	relational table stored in columnar format
S_r	relational table stored in row format
T_r	set of tables accessed in queries that perform better on row based data
T_c	set of tables accessed in queries that perform better on columnar data
r_i	replica corresponding to a table t_i
M	replica map
$Q(Q_1, Q_2,...,Q_x)$	set of queries
WR_i	wait ratio of table t_i

2 System Model

Consider a 'SALES' table comprised of the columns SalesId, SellerId, BuyerId, Timestamp, ItemId, Quantity, ItemPrice, Commission, TotalPrice, and four nodes n_1,n_2,n_3,n_4 configured in the RepliSmart framework. Initially, let the table be replicated on two nodes n_1, and n_3 both in row based storage format, and let the following be a subset (~70%) of queries, being issued on SALES table.

```
Q1.  SELECT SUM(Quantity)
     FROM SALES;
Q2.  SELECT Commission
     FROM SALES
     WHERE SellerId = ‘AP186009’;
Q3.  SELECT COUNT(*)
     FROM SALES
     WHERE ItemPrice > 30000
     AND ItemPrice < 80000;
Q4.  SELECT *
      FROM SALES
      WHERE SalesId = ‘9999’;
Q5.  SELECT MIN(TotalPrice)
     FROM SALES
     WHERE BuyerId = ‘CT186012’;
Q6.  SELECT TotalPrice – Commission AS CostPrice
     FROM SALES;
Q7.  SELECT ItemPrice, Quantity
     FROM SALES
     WHERE SellerId = ‘RK186064’;
```

Following are the observations from the set of queries.

- Except Q_4, rest of the queries could be categorized as analytical queries that perform better on a column oriented table.
- ~70% of the queries are issued on SALES table making the table a part of hot data.

Since the table is replicated in two nodes, the queries Q_1 and Q_2 can be executed in parallel. Even if the other two nodes n_2 and n_4 are free, the remaining queries have to be queued. Thus, replicating the SALES table, which is currently a part of hot data, makes query processing efficient, increasing the query throughput. Additionally, since majority of the queries issued on the SALES tables are analytical queries, storing the table in columnar format reduces the I/O, CPU costs which as well results in further increase of query throughput.
In the context of queuing up the queries, we define a metric Wait Ratio (WR) as

$$\mathrm{WRi} = \frac{\text{Number of times a query (accessing ti)queued}}{\text{Total number of queries (accessing ti)}}$$

For example, the wait ratio for SALES table is 4/7.

3 Proposed Scheme

Replication in RepliSmart is a process that enables data to be copied between nodes connected to the smart controller. Primarily, data replication is provided to allow read access to data on various nodes simultaneously for scalability, though it could be used as a backup secondarily.
Any of the queries from the clients in our framework would be received at the smart controller which maintains the information about all the replicas, and which has the knowledge about the workload of all the nodes connected. Based on the information, it decides which query should be redirected to which connected nodes. Essentially, following are the components in the framework with the corresponding responsibilities listed.

3.1 Smart Controller

The smart controller in our framework is comprised of the following components:

3.1.1 Load Balancer

Load balancer routes the incoming queries among the nodes connected based on the information corresponding to replicas for higher throughput.
For efficient query execution, the load balancer maintains ‘Replica Map’ shown in Table II, for routing the incoming requests to the most appropriate node holding the data replica being accessed in the request.

Table II
Replica Map

Table Name	Node Number	Storage format
t_1	1	S_r
t_1	2	S_c
t_1	3	S_r
.		
.		
.		
t_x	1	S_c
t_x	3	S_r

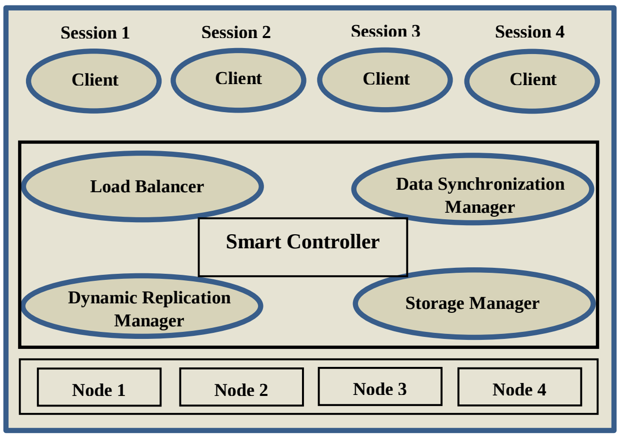
The lookup table holds information pertaining to all the relational tables. For each table, the lookup table lists all the nodes holding a replica and the corresponding storage format. Apart from the above lookup table, the load balancer maintains the workload of all the nodes connected and the status of each of them, whether a node is busy executing a query or is ready to accept a new query. Using some of the known techniques [12] (or heuristics like the number of columns being accessed in the query), it quickly determines whether the incoming query is better executed on a columnar or a row based storage. Accordingly, it redirects the incoming queries to the appropriate node. In addition, the load balancer makes sure that, in the selected node, there are no write requests pending on the table being accessed. Even if an incoming request could be better handled with a columnar table, if all the nodes holding the columnar table replicas are busy and a node n_i, holding a replica stored in row based fashion is idle, the load balancer would redirect the query to n_i.
The load balancer would handle only the incoming read requests. Upon execution, the corresponding node sends the read output directly to the client, and indicates the load balancer that it is ready to accept the next request.

If a query needs access to multiple tables which are not stored in the same node, a selected node (usually the one with maximum number of referenced tables stored locally) fetches the required tables from the other nodes using remote table operator [24]. In majority of the cases, RepliSmart makes sure that the tables accessed together are replicated in same node eventually. Data synchronization manager handles all the write requests.

3.1.2 Data synchronization manager

Synchronization manager makes sure that any of the incoming write requests executed on any node, must be performed on the remaining nodes holding the corresponding data replica using existing synchronization mechanisms. Upon execution of the write request, synchronization manager would respond back to the client.
Apart from data synchronization, it maintains a log of all the write queries executed. The log is helpful in bringing a node up to date as part of failure recovery.

RepliSmart Architecture



3.1.3 Dynamic replication manager

Replication manager is responsible for adjusting the number of replicas based on data temperature and the access patterns in the queries.
Using one of the existing techniques like [6], [7], [8], [20], [21], [22], we can determine the temperature of the data and categorize a table as hot or cold. In addition, for analyzing the query access patterns, RepliSmart builds a graph G, where the relational tables accessed in the query log are represented as nodes and the tables accessed in the same query being represented as edges between the tables accessed. In addition, each node holds information about its neighbors and access frequency for each neighbor, representing the number of times a neighbor is accessed in the same query. For example, if there is a query Q_1, which needs access to say tables t_1, t_2, t_5, then the tables t_1, t_2, t_5 would be represented in the form of vertices in G, three edges would be built between the pairs $\{t_1,t_2\}$, $\{t_2,t_5\}$, $\{t_1,t_5\}$. The vertex level information for t_1 holds neighbors as t_2, t_5 with the access frequency set to 1. If another query Q_2 accessing tables t_1, t_5, t_7 is issued subsequently, a new vertex t_7 would be added to the graph, and two new edges would be built between the pairs $\{t_5,t_7\}$, $\{t_1,t_7\}$, and the access frequency of the neighbor t_5 would be incremented by 1 in t_1's vertex information. Neighbors and access frequency information would be maintained at each vertex in the graph.
Apart from temperature and access patterns, replication manager would calculate the wait ratio for each table.
Using Algorithm 1, replication manager periodically deduces whether a new replica of a table is needed based on temperature and weight ratio. If a table is part of hot data and the WR is more than a configurable threshold μ_i, replication manager decides that a new replica is needed. Additionally, it also takes care of reducing the number of replicas corresponding to cold data using graph G.

```
--------------------------------------------------------------------
Algorithm 1: DYNAMIC REPLICATION
     input: Temp, Associated Tables, Replica Map,
Query Log, Wait Ratio(WR), threshold($\mu_i$)
     output: S = <$r_i$, L>
1.   T:= Set of tables accessed in the query log
2.   for each $t_i \in T$ do
3.   |    $Temp_i$ := Temperature of $t_i$
4.   |    $R_i$ := existing number of replicas of $t_i$
5.   |    $N_i$ := Set of nodes holding the replica of $t_i$
6.   |    $N'_i$ := Set of nodes with no replica of $t_i$
7.   |    $wr_i$ := wait ratio of $t_i$
8.   |    L := List of node candidates for replication
9.   |
10.  |    if $Temp_i$ is Hot and $wr_i > \mu_i$ then
11.  |    |    replica needed := true
12.  |    end
13.  |    if new replica is needed then
14.  |    |    $A_i$ := List of tables associated(tables joined
15.  |    |          with $t_i$ frequently)
16.  |    |    for each node $n \in N'_i$ do
17.  |    |    node weightage, nw := $\sum$ freq (number of
18.  |    |                          tables $\in A_i$,
19.  |    |                          replicated in n)
20.  |    |    add the key value pair (n, nw) to L
21.  |    |    done
22.  |    sort the list L by node weightage
23.  |    append <$t_i$, L> to S
24.  |    end
25.  done
26.  return S
--------------------------------------------------------------------
```

Using the list L, the replication manager would decide the node with highest node weightage as the one for replication based on free space available. If a node cannot accommodate a replica because of space constraints, the node with next higher node weightage is considered.
If a new replica r_i of a table t_i needs to be created, replication manager calculates the node weightage for each node in N'_i, as

sum of access frequencies of its neighbors (in G) residing in the node. Replication manager creates r_i in the node with maximum weightage. For example, if a query Q, accesses tables t_1,t_2,t_5,t_7, there would be four vertices in G and seven edges between the pairs $\{t_1,t_2\}$, $\{t_1,t_5\}$, $\{t_1,t_7\}$, $\{t_2,t_5\}$, $\{t_2,t_7\}$, $\{t_5,t_7\}$. Say the vertex information of t_1 is ($\{t_2{:}7\}$, $\{t_5{:}2\}$, $\{t_7{:}1\}$) as {neighbor: access frequency}. Consider that there are three nodes n_1, n_2, n_3 with n_1 holding a replica r_1 (corresponding to t_1) and n_2, n_3 holding replicas $\{r_2\}$, $\{r_5,r_7\}$ respectively. If a new replica of t_1 needs to be created, replication manager calculates the node weightage of node n_2 as 7 because it holds replica r_2 corresponding to t_2 alone and the access frequency of t_2 in the vertex information of t_1 is 7. Similarly, node weightage of node n_3 is calculated as 3 (2+1). So, node n_2 is chosen for creating a new replica. If n_2 were discarded because of space constraints, n_3 would be considered.

When there is a space constraint at any node, referring to the last accessed timestamp of each table t_i for which replicas exist, the replication manager removes one or more replicas considering the list of associated tables (A_i).

3.1.4 Storage Manager:

Storage manager decides the storage format of a new replica being created. For each relational table t_i in T_c, storage manager maintains the access frequency f_i^c. Similarly, f_i^r is maintained for each table t_i in T_r. If $f_i^c > f_i^r$ the new replica is created in columnar storage format. Otherwise, the replica is created in row based format.

Algorithm 2: DEFINE STORAGE FORMAT

```
   input: S     := <r_i, n_j> ,
          T_c{t_j}:= Set of tables accessed in queries that perform better on columnar data
          T_r{t_j} := Set of tables accessed in queries that perform better on row based data
1. $f^i_c$ := access frequency of table $t_i \in T_c$
2. $f^i_r$ := access frequency of table $t_i \in T_r$
3. if $f^i_c > f^i_r$ then
4.     create replica $r_i$ with format $S_c$ in $n_j$
5. else
6.     create replica $r_i$ with format $S_r$ in $n_j$
7. End
```

The framework ensures data consistency across the replicas for write operations using state of the art techniques proposed in [13], [14], [15]. In addition, the framework plans to use the techniques proposed in [23] to allow on-line updates to columnar databases, leaving intact their high read-only performance.

3.2 Node

The nodes connected to the smart controller would perform the following tasks.

- Execute the query and send the result back.
- Send a heartbeat message to smart controller to convey that it is up.
- Notify smart controller about the current load status

4 Illustration

In this section, we illustrate the proposed scheme using an example scenario mentioned in the system model section. Upon receiving the queries Q_1 and Q_2 that access the SALES table, load balancer looks into the replica map, which says that the table is replicated in the nodes n_1 and n_3. Say Q_1 is sent to n_1 and Q_2 is sent to n3 for execution, after observing the workload of n_1 and n_3. At this juncture, if Q_3, Q_4, Q_5, Q_6, Q_7 are the subsequent incoming requests, which as well access the SALES table, since both the nodes holding the replica are busy, all the three queries are queued up. Meanwhile, the Dynamic replication manager observes that the wait ratio of SALES table exceeding threshold μ_i (user defined), and it detects that the table is part of hot data (as the table is accessed ~70% of the times). As a result, the replication manager decides to create a new replica. Since only a single table is being accessed, any of the remaining two nodes is selected for creating a new replica (can be based on the resources available) of SALES table. Say n_2 is the node selected for replication. Further, storage manager is notified to create a replica with appropriate storage format. Storage manager observes that majority of the queries perform better on columnar storage (six queries perform better on columnar data and one query performs better on row based data). The access frequency of SALES table in T_c is 6 and the access frequency in T_r is 1. Thus, the storage manager creates the SALES table replica with columnar format in the node n_2.

5 Experimental Results

All experiments were run on a setup with eight nodes where each node is a Teradata machine with 24 parallel Intel x86 processors, and 128GB of memory. For experimental setup, we considered 100 different tables, with each table comprised of 500 columns and 24 million rows with a total size of 12TB. We compared the performance of RepliSmart with two base cases where the number of replicas remains four in both the cases, but the storage format remains row based in one base case and columnar in the other as shown in the table below.

Replica #	**Base case 1**	**Base case 2**	**RepliSmart**
1	Row Based	Columnar	Row Based
2	Row Based	Columnar	Row Based
3	Row Based	Columnar	Columnar
4	Row Based	Columnar	Columnar

Figures 1 and 2 display the resulting graphs on triggering queries as a batch of 500 at seven different time intervals, favoring columnar (~70% queries), and row based formats respectively.

In both the figures, we could notice that as time progressed, the performance of RepliSmart improved over the two base cases considering µ value as 0.5. This is because of the increase in the number of appropriate replicas with appropriate storage format. Initially, since there were no replicas and there is some overhead involved in creating the replicas, RepliSmart performance is a bit lower.

Additionally, we have considered three kinds of batch queries, where each batch is comprised of 500 queries with mixed query loads (Q_c, Queries that perform better on columnar storage, and Q_r, queries favoring row based storage). Specifically,

- Batch 1 consists of 70% Q_c, 30% Q_r
- Batch 2 consists of 30% Q_c, 70% Q_r
- Batch 3 consists of 50% Q_c, 50% Q_r

Figure 1

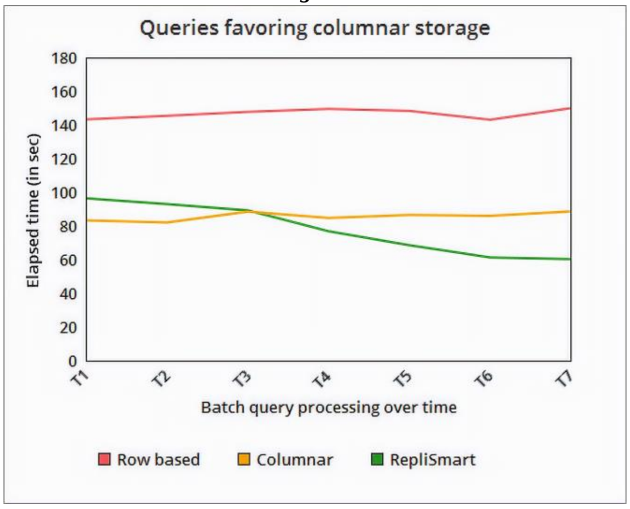
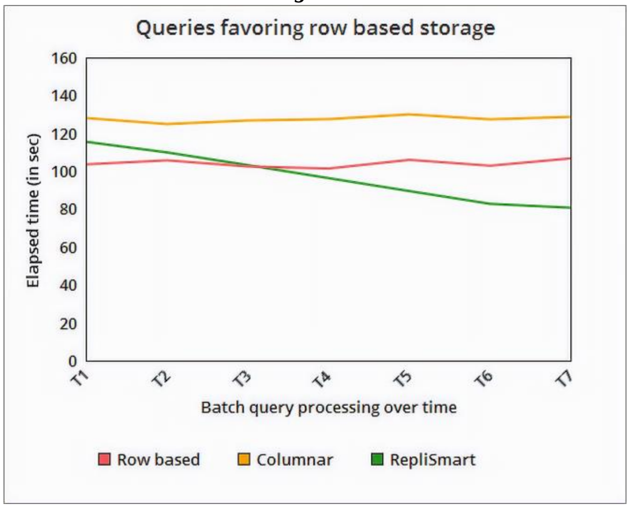
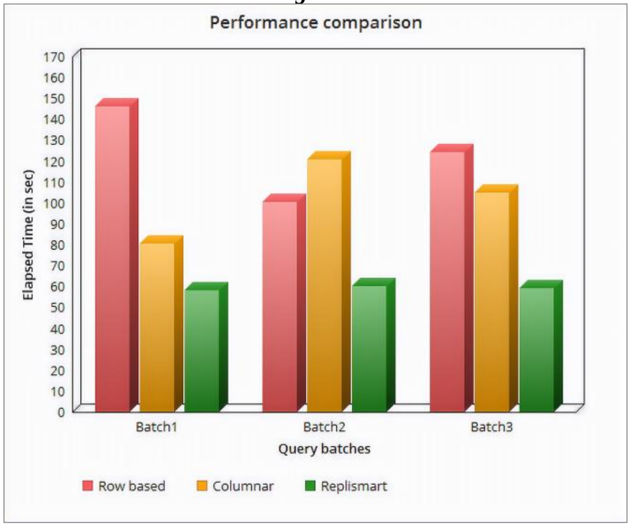


Figure 2



As shown in Figure 3, we could observe clearly that the framework takes advantage of both the storage formats, with different kinds of workloads resulting in improved query throughput. In the case of mixed query workloads where the read and write operations are equally executed, we observed that the performance is as worse as the case where there is no dynamic replication and all the replicas are in same storage format.

Figure 3



6 Summary and Future work

With the ever-increasing database applications that generate, collect, or retrieve data from clients, there is an increasing demand for organizing the data as per the client's needs, and in such a way that the data organization must result in high query throughput. As the frequency ratio of the data being accessed in the queries change over a period, there is a need for the data to be reorganized. Thus, there is a need for careful consideration of the data organization, be it the storage format, or the number of replicas to be created, for any given dataset, prior to making the database available for querying. In this paper, we proposed a framework (for read-heavy workloads) consisting of a smart controller and nodes connected to it, which hold data and processes the queries routed to them. The smart controller would adjust the number of replicas across the nodes connected based on the demand ratio. In addition, the smart controller would decide the storage format of each replica. We plan to extend this work to deal with the scenarios where part of the data unit (a table) is stored in columnar and the remaining in row based format. In future, we plan to enhance the smart controller to predict the optimal number of replicas and the storage format of a given data unit using data analytics. Additionally, the framework can be improvised to perform better in balanced workload environments (both reads and writes).

7 Acknowledgements

We acknowledge Teradata India Pvt Ltd for providing the infrastructure to carry out the experiments.

REFERENCES

[1] Daniel J. Abadi, Peter A. Boncz, and Stavros Harizopoulos. 2009. Column-oriented database systems. Proc. VLDB Endow. 2, 2 (August 2009), 1664-1665. DOI: https://doi.org/10.14778/1687553.1687625

[2] Gheorghe MATEI, 2010. "Column-Oriented Databases, an Alternative for Analytical Environment," Database Systems Journal, Academy of Economic Studies - Bucharest, Romania, vol. 1(2), pages 3-16, December.

[3] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, and S. Madden, “The Design and Implementation of Modern Column-Oriented Database Systems,” Foundations and Trends in Databases, vol. 5, no. 3, pp. 197-280, 2013.

[4] Wu Qiyue, "Research on column-store databases optimization techniques," *2015 International Conference on Logistics, Informatics and Service Sciences (LISS)*, Barcelona, 2015, pp. 1-7. doi: 10.1109/LISS.2015.7369708

[5] David Loshin, “Gaining the Performance Edge Using a Column-Oriented Database Management System”, Analytics in the Federal Government, White paper series on how to achieve efficiency, responsiveness and transparency, January 2010.

[6] https://in.teradata.com/Resources/White-Papers/Teradata-Intelligent-Memory.

[7] J. J. Levandoski, P. Larson and R. Stoica, "Identifying hot and cold data in main-memory databases," *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, Brisbane, QLD, 2013, pp. 26-37. doi: 10.1109/ICDE.2013.6544811

[8] K. Kim, S. Jung and Y. H. Song, "Compression ratio based hot/cold data identification for flash memory," *2011 IEEE International Conference on Consumer Electronics (ICCE)*, Las Vegas, NV, 2011, pp. 33-34. doi: 10.1109/ICCE.2011.5722616

[9] S. Elnaffar, P. Martin, and R. Horman, “Automatically classifying database workloads”, International Conference on Information and Knowledge Management(CIKM), pp. 622–624, 2002.

[10] Bettina Kemme and Gustavo Alonso. 2000. A new approach to developing and implementing eager database replication protocols. ACM Trans. Database Syst. 25, 3 (September 2000), 333-379. DOI=http://dx.doi.org/10.1145/363951.363955

[11] Makpangou, Mesaac. (2009). P2P based hosting system for scalable replicated databases. 47-54. 10.1145/1698790.1698800.

[12] Said Elnaffar, Pat Martin, Randy Horman, “Automatically Classifying Database Workloads”, International Conference on Information and Knowledge Management(CIKM), November 4-9, 2002

[13] Javier García-García and Carlos Ordonez. 2009. Consistency-aware evaluation of OLAP queries in replicated data warehouses. In Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP (DOLAP '09). ACM, New York, NY, USA, 73-80. DOI: https://doi.org/10.1145/1651291.1651305

[14] Haifeng Yu and Amin Vahdat. 2006. The costs and limits of availability for replicated services. ACM Trans. Comput. Syst. 24, 1 (February 2006), 70-113. DOI=http://dx.doi.org/10.1145/1124153.1124156

[15] Yi Lin, Bettina Kemme, Ricardo Jiménez-Peris, Marta Patiño-Martínez, and José Enrique Armendáriz-Iñigo. 2009. Snapshot isolation and integrity constraints in replicated databases. ACM Trans. Database Syst. 34, 2, Article 11 (July 2009), 49 pages. DOI: https://doi.org/10.1145/1538909.1538913

[16] V. Bhagat and A. Gopal, "Comparative Study of Row and Column Oriented Database," *2012 Fifth International Conference on Emerging Trends in Engineering and Technology*, Himeji, 2012, pp. 196-201. doi: 10.1109/ICETET.2012.56

[17] A. Kamal and S. C. Gupta, "Query based performance analysis of row and column storage data warehouse," *2014 9th International Conference on Industrial and Information Systems (ICIIS)*, Gwalior, 2014, pp. 1-6. doi: 10.1109/ICIINFS.2014.7036537

[18] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. 2005. C-store: a column-oriented DBMS. In Proceedings of the 31st international conference on Very large data bases (VLDB '05). VLDB Endowment 553-564.

[19] A. S. Kanade and A. Gopal, "Choosing right database system: Row or column-store," 2013 International Conference on Information Communication and Embedded Systems (ICICES), Chennai, 2013, pp. 16-20. doi: 10.1109/ICICES.2013.6508217

[20] Jongsung Lee and Jin-Soo Kim. 2013. An empirical study of hot/cold data separation policies in solid state drives (SSDs). In Proceedings of the 6th International Systems and Storage Conference (SYSTOR '13). ACM, New York, NY, USA, , Article 12 , 6 pages. DOI: https://doi.org/10.1145/2485732.2485745

[21] D. Park and D. H. C. Du, "Hot data identification for flash-based storage systems using multiple bloom filters," 2011 IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST), Denver, CO, 2011, pp. 1-11.

[22] Chen J., Deng Y., Huang Z. (2015) HDCat: Effectively Identifying Hot Data in Large-Scale I/O Streams with Enhanced Temporal Locality. In: Wang G., Zomaya A., Martinez G., Li K. (eds) Algorithms and Architectures for Parallel Processing. ICA3PP 2015. Lecture Notes in Computer Science, vol 9529. Springer, Cham

[23] Sándor Héman, Marcin Zukowski, Niels J. Nes, Lefteris Sidirourgos, and Peter Boncz. 2010. Positional update handling in column stores. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 543-554. DOI: https://doi.org/10.1145/1807167.1807227

[24]https://docs.teradata.com/reader/vLlhnTq8biC8lbWbMR3PBA/GNVVgCfo5Bb2qQvRUftASw