EPHEMERAL PARTIALLY REPLICATED DATABASES

Ritesh Agrawal[1] and Peter I. Frazier[1,2]

[1]Uber Technologies, Inc., 555 Market St., San Francisco, CA, USA
[2]School of ORIE, Cornell University, 232 Rhodes Hall, Ithaca, NY, USA

ABSTRACT

In today's analytics-driven world, fully replicated isolated databases provide much-needed database availability and compute scalability but at the cost of storage scalability, an issue that is addressed by partially replicated isolated databases. However, a partially replicated database that is optimal at the time of design is soon made inefficient by changing business needs, products & services it offers, datasets and query workloads. To this address this issue, we introduce the notion of migration cost as an additional factor that influences the design of a partially replicated databases. In this paper, we formalize the notion of migration cost and present a new cost-based objective function to partition and allocate data elements across available databases. Further, we discuss its implementation in the context of Uber and demonstrate its effectiveness based on a 10-week simulation study.

KEYWORDS

Databases, Partially Replicated Databases, Mixed Integer Linear Optimization

1. INTRODUCTION

In today's data-driven economy, the design of a holistic database solution is governed by three factors: database availability[2, 7], compute scalability [5, 10]and storage scalability. Database availability is concerned with the up-time of the database and its ability to execute requests. Compute scalability is concerned with the query throughput of the database. Lastly, storage scalability is concerned with the amount of data that can be managed by the database. Different approaches exist to maximize the three different factors. One of the approaches to maximize database availability and compute scalability is to have a multiple fully replicated isolated databases. Fully replicated isolated databases consists of multiple isolated databases with the same data copied across all the databases . Further, the connection between the client applications and the databases is often abstracted using a thin service layer. Such a database design helps achieve a high degree of database availability by routing queries to operational databases and compute scalability by distributing the query load across all the operational databases[13].

Nevertheless, the compute scalability achieved using the fully replicated databases doesn't grow linearly with the number of database replications. Each new replication of the data requires additional copying of data from one database to another, and thereby significant amount of resources are wasted on reading and writing data[11]. Furthermore, while the fully replicated database system helps achieve a high degree of database availability and compute scalability, it fails to provide storage scalability. At Uber, achieving a high degree of storage scalability along

David C. Wyld et al. (Eds) : AI, CSTY, NMOCT - 2018
pp. 01–13, 2018. © CS & IT-CSCP 2018 DOI : 10.5121/csit.2018.81401

with database availability and compute scalability is important for multiple reasons. First, we rely on Vertica, a fast analytic engine, to munge over petabytes of data and provide critical business insights as quickly as possible; the mean average time to complete a query is about 17 seconds. Further, we have multiple fully replicated Vertica database cluster to maximize database availability and to handle millions of queries on a daily basis. However, Vertica's licensing fee is tied to the amount of data stored. As a result, each additional replication significantly increases the licensing cost. In particular, since the number of replications is governed by the compute scalability requirement rather than the database availability, there is a significant amount of storage, and licensing fee is wasted due to additional replications of all the data elements in the case of a fully replicated isolated databases. The other reason to maximize storage utilization is that Vertica operates at its peak performance on a limited amount of data. Full replication thereby limits our ability to maximize the amount of data that can be quickly processed across all the databases. Apart from storage scalability, a fully replicated isolated database system suffer from other challenges as well such as data consistency management[9], non-linear computational growth as each additional replication requires additional computational resource for writing and managing data[12].

A natural solution to achieve database availability, compute scalability and storage scalability is a partially replicated databases system. As shown in Figure 1, a partially replicated databases optimize for storage by copying different overlapping subsets of datasets to different databases. There are many different proposed approaches for determining an optimal partially replicated database configuration, but one challenge that remains unattended is the ephemeral nature of partially replicated isolated databases. Different business units grow at different rates and thereby associated data elements. As a result, partial database configurations eventually become imbalanced in terms of data distributions across databases. Furthermore, as business evolve so the analytical requirements. As a result, the query patterns might change over time, causing imbalances in the distribution of the query load. As a result, a partial database configuration eventually becomes sub-optimal and thereby requires rebalancing of partial databases by moving data elements from databases to another. This migration of data elements is termed here as "migration cost."

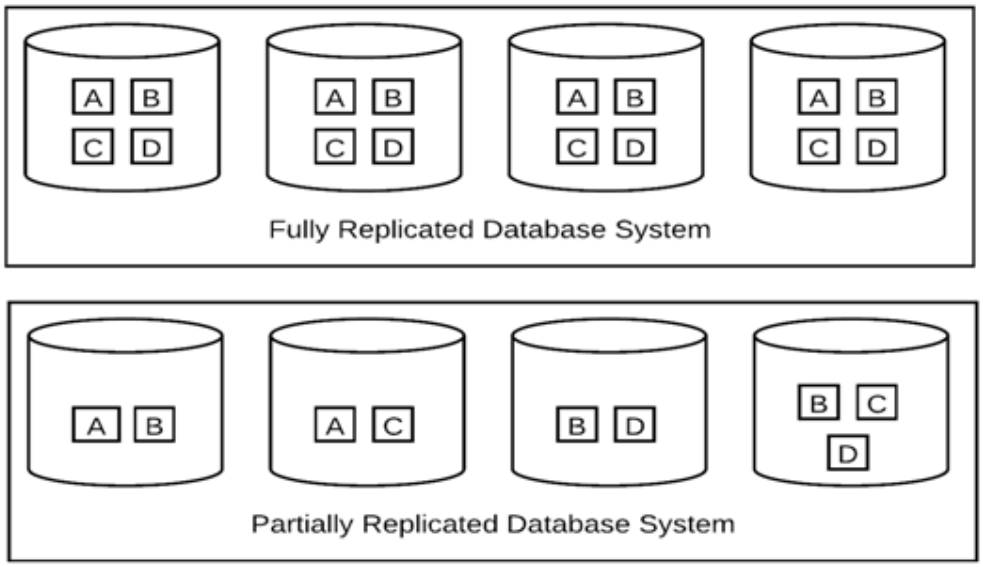

Figure 1. Sample Configuration of a fully replicated isolated databases and a partially replicated isolated databases. All data elements are replicated across all the databases in the case of a fully replicated

databases. In the case of a partially replicated isolated databases, different datasets contains different overlapping sets of data elements.

In this paper, we present a migration cost based formalization of partially replicated isolated databases and discuss its implementation in the context of Uber. In Section 2, we discuss prior approaches for designing partially replicated databases. In section 3, we discuss our formalization of the problem that explicitly incorporates migration cost for determining the optimal placement of data elements across available databases. In Section 4, we present the implementation details of our ephemeral partially replicated databases at Uber and the savings achieved as compared to the fully replicated databases. Lastly, in Section 5, we present our conclusion. Note that the scope of this paper is limited to isolated databases i.e. to complete a given query, all the required data elements should be part of a single database.

2. RELATED WORK

A partially replicated database system is described here as a system of multiple isolated databases with overlapping sets of data elements. Since these databases are isolated, all the data elements necessary to satisfy a query are required to be available on a single database. Designing a partially replicated database system is usually a two-step process involving partitioning and allocation[11].

Partitioning is concerned with generating partially overlapping sets of data elements. Previous efforts on partitioning mainly focused on "data marts", partitioning data elements to address specific problems and is often organized around various teams or products [3]. A more data-driven approach to partition is presented in [1]. In general, the data-driven approach uses historical query patterns to identify strongly correlated sets of data elements. Depending on how data elements are defined, these data-driven approaches can generate horizontal (dissecting data by rows) [4]or vertical (dissecting data by columns/attributes) partitioning of the data[1]. In contrast to partitioning, the allocation problem is concerned with assigning partitions of data elements to available databases and often studied as an optimization problem.

In [11], partitioning and allocation are considered as a single problem and proposed a data-driven approach to generate a partially replicated database configuration. The objective of their data-driven solution is to balance storage requirement and query load across various isolated databases. Queries are grouped into various classes based on the different sets of data elements required to satisfy a query and are further assigned compute cost. Then the objective for generating partially replicated database configuration is to assign these different classes of queries to different databases to balance data size and query load across all the databases.

One important aspect missing in previous approaches is the lack of consideration to the ephemeral nature of partially replicated database systems. Different product or services offered by a business grow at a different rate and thereby associated data size. Further, to maintain the competitive edge, business constantly add new services and products as well deprecate some. As a result, data elements and associated business queries continuously change. As a result, partially replicated databases often needs to be re-configured to adapt to the changing business and uneven workload. The cost of transitioning from one state of a partially replicated database to another is termed here as the migration cost. A high migration cost negatively hurts database availability and compute scalability as a significant amount of computing resources are spent on migrating data elements from one database to another. In this paper, we extend the idea presented in the above approaches and especially in [11] to explicitly account for the migration cost.

3. MIGRATION AWARE PARTIALLY REPLICATED DATABASES

Four factors influence the design of partially replicated isolated databases: database availability, compute scalability, storage scalability and migration cost. We approach the design of the partially replicated isolated databases as a single optimization problem that explicitly models the influence of all the four factors. Similar to[6, 11], our approach combines partitioning and

allocation steps in a single optimization problem. However, we first introduce key elements of our formalization in Section 3.1. Thereby, in Section 3.2, we introduce cost functions associated with each of the four factors and combine them into a single objective function.

3.1. Notations

Below are the various notations used while constructing our objective function.

1. Database (N): Let N represent the number of available databases. These databases are isolated i.e. that operate independently of each other. Further, these N databases can be either homogenous (same hardware configuration) or heterogeneous (different hardware configuration).

2. Disk Capacity (D): Let real vector D represent the disk space capacity for each of the N databases.

3. Compute Capacity (P): Let real vector P represent the compute capacity for each of the N databases. As compared to measuring disk capacity,estimating compute capacity is often a challenge and depends on multiple factors including hardware and software configuration. For the purpose of this paper, we measure compute capacity in terms of maximum query load that a database can handle. In the case of heterogeneous database system, an optimal query load for a database can be estimated using the equation shown below:

$$P_I = \frac{Concurrency_i \; X \; [2]\text{Error! Bookmark not defined.}\; 86400}{AvgQueryRuntime_i} \qquad (1)$$

   Above, $Concurrency_i$ represents observed historical query concurrency and $AvgQueryRuntime_i$ represents the average query runtime on database i to run a query on database i. Thus, P_i represents the average number of queries that a database can handle on a daily basis. In the case of homogenous databases, one can estimate P_i by distributing existing query volume equally across all the databases i.e. $P_i = TotalQueriesPerDay/N$.

4. Data Elements (T): A data element is defined abstractly over here and can indicate a table, rows of a table or columns of a table. Depending on the definition of data elements, the proposed algorithm will either generate partitions similar to data mart, horizontal or vertical partitioning, respectively. As a result, often the definition of data element is constraint by engineering feasibility. For instance, if data elements are described as set of rows then migration of a partially replicated databases from one state to another would involve migrating rows of a table. For the explanation purpose, we here defined data elements as individual tables and represent the set of tables as T. Thus T_i represents a specific table i.

5. Data Element Size (O): Measured in the same unit as D, O_i captures the disk space occupied by the data element i.

6. Query Classes (Q): A query involves operating on a set of data elements. Thus, one can group queries based on the set of data elements, i.e. queries using the same set of data elements are grouped together. For instance, if a query q requires three data elements, say

T_1, T_3 *and* T_5, we can represent q by the set of $\{T_1, T_3, T_5\}$and further group all the queries this particular set of data elements into one class, referred in [11] as query class. Let Q represent the set of query classes. Note that query classes can have different number of data elements and can be overlapping.

7. Data Element and Query Class Mapping (C): Let C be a binary matrix that describes mapping between query classes and data elements. Thus, if $C_{\{q,t\}} = 1 \; \forall q \in Q, t \in T$ indicates that query class q requires data element t.

8. Query Class Weight (W): Expressed in the same unit as that of P, W_i represents compute capacity needed for the query class Q_i. Here, P_i is measured in terms of daily query load and, thereby,W_i represents average daily number of queries associated with query class Q_i.

9. Replication factor (r): Canonically the term replication factor is used to indicate the minimum number of replicas of data elements. However, we consider it an aspect of a query class and represents the minimum number of isolated databases that should be able to handle a given query class. We find this interpretation more suitable as it ensures not only that data elements are replicated at-least r times but that each query class is covered by at-least by$r$ databases.

Apart from the above variables, we have two decision variables X and Y. X, a binary matrix, represents assignment of data elements to databases;$X_{t,n} = 1$ indicates that the data element t is assigned to a database n, and $X_{t,n} = 0$ otherwise. Y, also a binary matrix, represents assignment of query classes to databases. Similar to X, $Y_{q,n} = 1$ indicates query class q assigned to database n, and $Y_{q,n} = 0$ otherwise. Additionally, we define X' as the existing state of assignment of T databases to N databases. This will be used to determine the migration cost by comparing X to X'.

3.2. Cost Function

As discussed above, four factors influence the design of partially replicated isolated databases: database availability, compute scalability, storage scalability and migration cost. Database availability is a function of the replication factor. As the replication factor increases, database availability increases and vice-versa. In our approach, we assume that database availability is a given parameter and hence one doesn't need to explicitly model for database availability. For the remaining three factors, we propose a simple mixed integer quadratic constraint formalization. Minimizing this formalization provides an optimal partitioning and allocation strategy to uniformly distribute compute and storage utilization across all the available databases. As explained below, the proposed formalization is a linear combination of three individual cost functions associated with the compute scalability, storage scalability and migration cost.

1. Storage Utilization: One of the primary motivations of a partially replicated isolated databases is to minimizeoverall storage consumption across N databases. In practice, however, another important constraint is that disk utilization across N databases to be uniformly distributed. Disk utilization of a database is described as the percentage of the total disk space that is consumed. Given the data element assignment matrix X, the percentage disk utilized$s_i$on a database i can be computed as:

$$s_i = \frac{\sum_{t=1}^{|T|} X_{t,i} O_t}{D_i} \; \forall \; t \in T \tag{2}$$

Since X is a binary matrix, the numerator in the above equation represents the sum of disk space required by the data elements assigned to database i. Normalized by total disk space capacity of database i, s_i represents the percentage of disk space utilized by the all the data elements assigned to database i.

2. Compute Utilization: Compute scalability is achieved by evenly distributing the query load across N databases in proportion to individual databases compute capacity P. Thus, compute utilization of database is described as the percentage of compute capacity that will be utilized by queries assigned to the given database. Given the query assignment binary matrix Y, the percentage query load l_i on a database $i$compared to its compute capacity $P_i$can be computed as:

$$l_i = \frac{\sum_{q=1}^{|Q|} Y_{q,i} W_q}{P_i} \; \forall \; q \in Q \tag{3}$$

One caveat with the above the cost function is that it assigns the complete weightW_q of query class q to a single database. However, in practice, queries associated with a particular query class can addressed by r databases. However, we prefer the above formalization as it models the worst case scenario where only one of $r$databases that can handle a given query class is available and thereby has to deal with the complete load of query class q.

3. Migration Cost: Migration cost is associated with the cost of moving from one state of partially replicated databases to another, i.e. migration from state X' to X. Migration involves two types of operations on databases: writing new data elements that are in X but not in X' and deleting data elements that were in X' but not in in X. Since writing is a much more expensive operation as compared to deleting, we define migration cost based on the cost of writing. Further, the cost of writing is directly proportional to the size of the data element. Hence, as shown in eqn. 5, we define the migration cost as the total size of new data element that will be written on a database normalized by the acceptable amount of writing $\Delta_i$on database i, as defined by the database admin. Normalizing migration cost by Δ helps in two ways. First, it helps database admin enable control the influence of the migration cost. Second, it makes the cost function unitless and therefore comparable to other cost functions (eqn. 3 and 4).

$$m_i = \frac{\sum_{t=1}^{|T|} O_t \left(1 - X'_{t,i}\right) X_{t,i}}{\Delta_i} \tag{4}$$

In the above expression, $\left(1 - X'_{t,i}\right) X_{t,i}$ identifies new data elements that were previously not present on database i but are required in the new state X.

Based on the above three individual cost function, we now define our objective function as minimizing linear combination of the maximum of each of the individual cost functions, i.e.:

$$J(X,Y) = \min (S + L + M) \quad (5)$$

where

- $S = \max (s_1, s_2, \dots, s_N)$
- $L = \max (l_1, l_2, \dots, l_N)$
- $M = \max(m_1, m_2, \dots, m_N)$

subject to:

1. $X_{t,n} \in \{0,1\} \forall\, t \in T, n \in [N]$
2. $Y_{q,n} \in \{0,1\} \forall\, q \in Q, n \in [N]$
3. $\sum_{n=1}^{N} Y_{q,n} \geq r$
4. $X_{t,n} \geq Y_{q,n} \forall\, q \in Q\,, t \in T, n \in [N]$

In practice, one prefers a state with minimal variance in terms of percentage disk utilization and compute utilization relative to individual databases' disk and compute capacity. However, minimizing variance doesn't guarantee optimal utilization of resources. For instance, assuming N homogeneous databases, zero variance can be achieved by replicating all the data elements across all the databases. However, such a state will be far from an optimal solution. Hence, along with minimizing variance, one also need to minimize total disk utilization. Instead, as defined by S, minimizing maximum disk utilization across N databases achieves both the objectives. The same argument goes for L and M.

Another important aspect of our objective function shown in eqn. 6 are the four constraints. While constraint 1 and 2 ensure that the data element X and query class Y assignment matrix are binary, constraint 3 ensures database availability by making sure that each query class is assigned to at-least r databases. The fourth constraint ensures that a query assigned to a database i can only be successfully served if all the data elements required the query are available on database n. Also, note that constraint 3 and 4 together ensure that each data element is replicated across r databases. If desired, similar to constraint 3, one can also add a constraint to enforce a different replication factor, say r', for data elements.

We emphasize that the binary matrix$Y$ reproduces the set of databases that could process a query class, rather than the actual assignment of query workload across databases. This assignment is handled dynamically to load balance with knowledge of the current system load, rather than statically.

4. RESULTS

With millions of Uber trips every day, Uber infrastructure handles petabytes of data. As shown inFigure 2, the data is streamed through Apache Kafka and after that stored in the Hadoop Distributed File System (HDFS). Next, Hive is used to clean and create a modeled data. Data analysis on HDFS using Hive or Presto is usually slow, and hence core business data is further copied to a system of fully replicated isolated Vertica databases. Vertica is a proprietary in-

memory database that provides near real-time interactive query experience; 90% of queries complete within 20 seconds as compared to about a minute in the case of Presto.

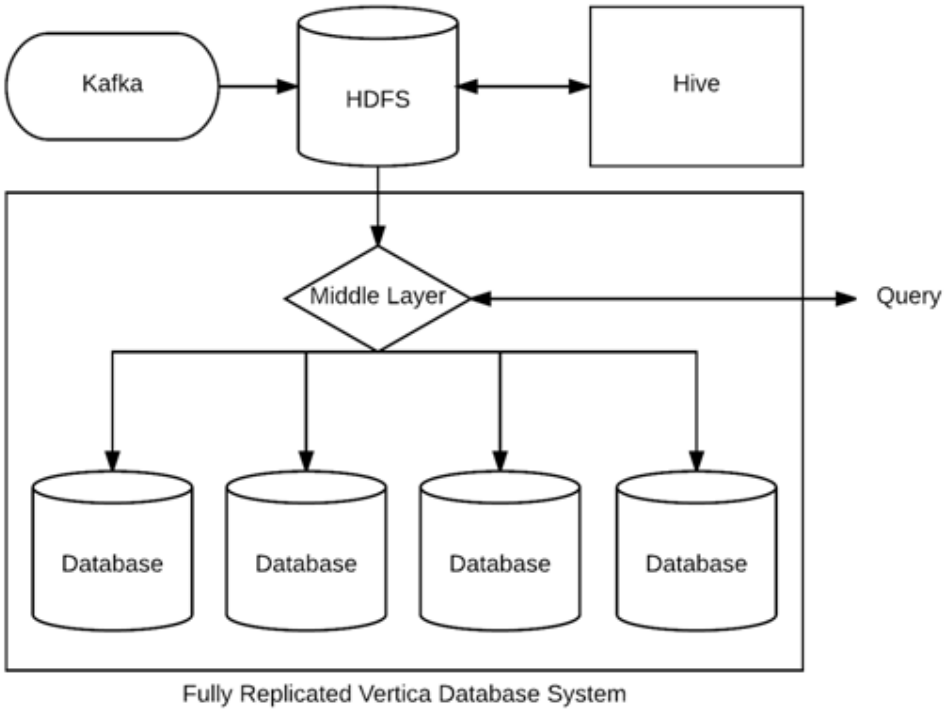

Figure 2. Existing data infrastructure setup at Uber. Data flows from Kafka to HDFS and eventually replicated across multiple isolated Vertica databases. Clients connect to the databases through a middle layer that helps distribute query load across available databases.

In the case of Uber, the compute scalability requirement mainly drives the need for multiple fully replicated isolated Vertica databases system. But, as discussed before, this compute scalability comes at the cost of storage scalability. Apart from optimizing for database availability and compute scalability, achieving a high degree of storage scalability is critical for Uber for two reasons. First, as discussed in Section 1, the additional storage required due to full replication of all the data elements cost millions of additional dollars in terms of licensing fee. Second, there is a limit on how much data can be managed by a single Vertica database to provide near real-time interactive query experience. To address these two major issues, we explored the option of partially replicated isolated databases. The architecture diagram below shows the major component of our solution.

As compared to Figure 2, one key difference is an addition of “data element assignment manager” that informs other components about the placement of the data elements to different databases. The middle layer communicates with the data element assignment manager to determine and route the query to one of the candidate databases that contains all the data elements needed by the query. Every week, based on last four weeks of historical queries, the data element assignment manager generates a new assignment matrix. The assignment matrix is then communicated to the data loader which is responsible for migrating databases from the existing state to the new state.

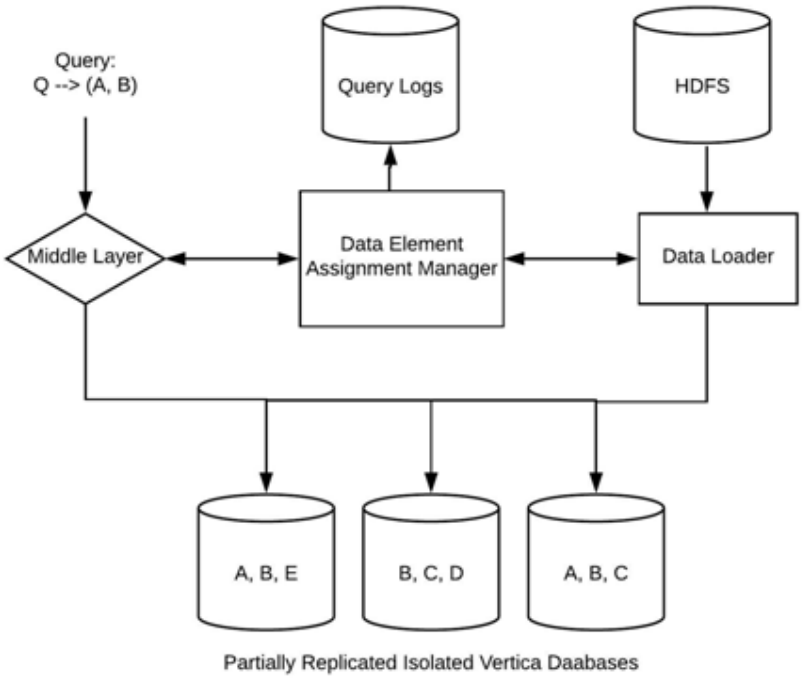

Figure 3. Architecture of our ephemeral partially replicated isolated Vertica databases. The key component is data element assignment manager that re-computes optimal assignment for each data element every week.

In order to evaluate our approach, we ran a ten-week simulation on historical queries. For the simulation, we assumed five homogenous Vertica databases (i.e. N = 5) and a replication factor of 2 (i.e. r = 2). As described in Section 3.1, historical queries are grouped into different query classes based on the set of referred tables. For each query class, we further used the average daily number of queries based on the last four weeks of historical queries as a representation for the query class weight. Further, we approximate compute capacity of each database P_i to be equal to the total query workload distributed equally across N databases i.e. $P_i = \frac{\sum_{i=1}^{|Q|} W_i}{N}$

One of the challenges we faced was the scale of the optimization problem formulated in eqn. 6. With hundreds of tables and thousands of query classes, solving a mixed integer quadratic constraint optimization problem with thousands of decision variables is a non-trivial task. To scale down the problem, we relied on empirical observations to make the optimization problem formulated in eqn. 6 substantially simpler, thereby obtaining a near optimal solution within a few minutes using the commercial mixed integer linear programming solver, Gurobi[8]. First as shown in figure 4, the top 10% biggest tables account for almost 90% of the total data. Thus most of the disk efficiency comes by an optimal placement of these big tables. Based on this observation we define our set of data elements T to include only these top tables and thereby significantly reduce the size of the X matrix. The remaining 90% of the tables are fully replicated as the potential gains in terms of disk utilization are insignificant.

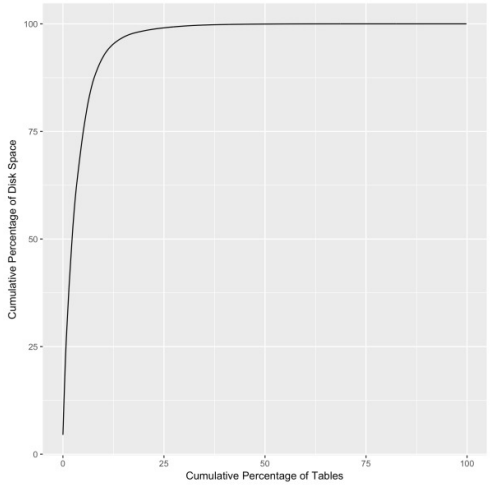

Figure 4. Cumulative percentage of data by percentage of tables in descending order by size. Thus, biggest 10% account for almost 90% of the total data.

Second, a side effect of restricting the set of data elements to the top 10% biggest tables is that it significantly reduces the number of query classes one has to consider. Since query class definition is based on data elements in T, any data element not in T is dropped from the query class definition. If all the data elements needed by the query class are outside the scope of T, then we drop the query class itself from consideration. This is justified since data elements outside of T are fully replicated across all the databases and hence the queries can be executed on any of the N databases. Figure 5 shows the number of query classes as a function of the percentage of tables included in T. Thus, by considering only the top 10% biggest tables (the same as the set tables considered within our first improvement), the number of query classes decreases from almost 100K to 40K. This significantly reduces the size of the Y matrix.

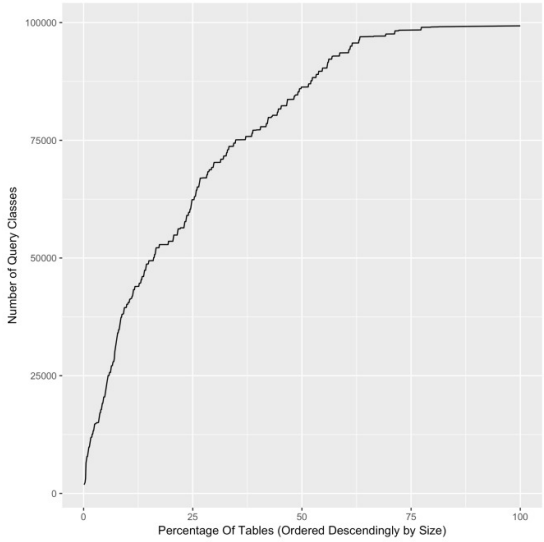

Figure 5. Number of query classes as a function of number of tables considered as part of T. Thus, if we consider the top 10% biggest tables as part of T, the total number of query classes drop from almost 100K to 40K.

Apart from the above considerations, there are new tables that require special handling. Due to a lack of historical data, it is difficult to assign new tables to databases in an optimal way. Hence, for the first two weeks since the creation of a table, we fully replicate new tables across all the databases. This approach helps reduce migration cost associated with changing query patterns related to new tables. From the third week, new tables are treated as regular tables and considered for optimal allocation only if it makes it into top 10% biggest tables.

For the simulation, we generated a new data element assignment matrix X at the beginning of each week and evaluated the performance of our recommended assignment based on proceeding weeks queries. As described below, we measured the performance of the assignment on four different criteria:

1. Savings In Disk Utilization

One of the main objectives of a partially replicated isolated database is to minimize percentage disk utilization of individual databases. Since, our objective is to measure savings in disk space as compared to a fully replicated databases, we define savings in disk utilization as:

$$DiskUtilization_i = 1 - \frac{\sum_{t=1}^{|T|} X_{t,i} O_t}{\sum_{t=1}^{|T|} O_t} \quad (6)$$

In eqn. 7, the numerator represents the size of data elements assigned to database i. The denominator represents the total size of all the data elements. Based on the simulation results, we observed that the median percentage disk savings on an individual database ranged between 58% and 62%. Thus, as compared to fully replicated database, we are able to recover almost 40% disk space. Additionally one can notice from the range that the disk savings is almost evenly distributed across databases.

2. Migration Cost:

As described in Section 3.2, migration cost is concerned with the size of new data elements that were previously not assigned to a given database and are required in the new state and can be computed as:

$$MigrationCost_i = \sum_{t=1}^{|T|} (1 - X'_{t,i}) X_{t,i} O_t \quad (7)$$

Over the 10 week simulation, the median migration cost for the five databases ranged from 0.07 TB to 1.5 TB. This was well within our acceptable level of migration Δ that was set to 5TB. Note that the migration cost for the first week is zero because we are migrating from the fully replicated database to partially replicated database and hence one has to only delete data elements in order to move to a new state.

3. Query Load:

It refers to the percentage of queries from the following week that will be assigned to a database. While, in the actual implementation, the placement of a query is influenced by the current load on candidate databases, for the simulation we randomly assigned the query to one of the candidate databases. The median query load for the five databases ranged from 19.14% to 21%.

4. Dropped Queries:

One of the dangers of a partially replicated isolated database is the lack of all the required data elements by a query on a single database. In this case, the query cannot be satisfied by any of the databases. Over the 10 week simulation, the maximum number of queries dropped in a week was 11. As compared to almost half a million queries that were successfully handled, missing 11 queries was insignificant and not a concern.

5. CONCLUSION

Traditionally, the design of partially replicated databases has focused on increasing storage scalability but without accounting for the migration cost. In this paper, we formalized the notion of the migration cost and presented a new cost-based objective function that along with other factor optimizes the allocation of data elements for migration as well. Based on 10-week simulation results on actual query load on the Vertica database, we demonstrate that this migration based formalization not only helps achieve significant savings in terms of disk utilization but also optimizes for migration. As compared to a fully replicated isolated databases, disk utilization dropped by almost 40%. Further, the median migration cost ranges from 0.07 TB to 1TB across different databases. Although we did observe 11 dropped queries in a week, the number of dropped queries were insignificant as compared to almost half a million queries that our system is able to handle correctly and all the other potential gains in terms of disk utilization.

REFERENCES

[1] Agrawal S, Narasayya V, Yang B (2004) Integrating vertical and horizontal partitioning into automated physical database design. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data. ACM, pp 359–370

[2] Bernstein PA, Goodman N (1983) The failure and recovery problem for replicated databases. In: Proceedings of the second annual ACM symposium on Principles of distributed computing. ACM, pp 114–122

[3] Bonifati A, Cattaneo F, Ceri S, Fuggetta A, Paraboschi S (2001) Designing data marts for data warehouses. ACM transactions on software engineering and methodology 10:452–483

[4] Ceri S, Negri M, Pelagatti G (1982) Horizontal data partitioning in database design. In: Proceedings of the 1982 ACM SIGMOD international conference on Management of data. ACM, pp 128–136

[5] Coulon C, Pacitti E, Valduriez P (2005) Consistency management for partial replication in a high performance database cluster. In: Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on. IEEE, pp 809–815

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

[7] El Abbadi A, Toueg S (1985) Availability in partitioned replicated databases. In: Proceedings of the fifth ACM SIGACT-SIGMOD symposium on Principles of database systems. ACM, pp 240–251

[8] Gurob Inc (2016) Gurobi Optimizer Reference Manual. In: Gurobi. http://www.gurobi.com

[9] Kemme B, Alonso G (2000) Don't Be Lazy, Be Consistent: Postgres-R, A New Way to Implement Database Replication. In: Proceedings of the 26th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., pp 134–143

[10] Kemme B, Jiménez-Peris R, Patiño-Martínez M (2010) Database replication. Synthesis Lectures on Data Management 5:1–153

[11] Rabl T, Jacobsen H-A (2017) Query centric partitioning and allocation for partially replicated database systems. In: Proceedings of the 2017 ACM International Conference on Management of Data. ACM, pp 315–330

[12] Schiper N, Schmidt R, Pedone F (2006) Optimistic algorithms for partial database replication. In: International Conference On Principles Of Distributed Systems. Springer, pp 81–93

[13] Song S (2017) A Framework for Design of Partially Replicated Distributed Database Systems with Migration Based Genetic Algorithms. International Journal of Database Management Systems 9:01–21

AUTHORS

Ritesh Agrawal is a lead data scientist at Uber with more than 10 years of experience in the field of big data and machine learning. At Uber, he focuses on using machine learning and artificial intelligence techniques to help scale Uber's infrastructure. Before Uber, Ritesh worked on predictive and ranking models at Netflix, AT&T Labs and Yellow Pages, Inc. He received MS in the field of Transportation Engineering from the University of Illinois at Urbana Champaign (UIUC) in 2004 and thereafter worked towards his PhD in the field of Environmental Earth Sciences from Pennsylvania State University (PSU). His PhD thesis focused on computational tools and technologies such as ontologies and concept maps.

Peter Frazier is an Associate Professor in the School of Operations Research and Information Engineering at Cornell University. He works on problems in learning and decision-making in which decisions affect the data available. This includes Bayesian optimization, multi-armed bandits, Bayesian sequential experimental design, optimization via simulation, and incentivized exploration. He is also a Staff Data Scientist at Uber, where he works on pricing and marketplace design. He received a Ph.D. in Operations Research and Financial Engineering from Princeton University in 2009. He is the recipient of an AFOSR Young Investigator Award and an NSF CAREER Award, and is an associate editor for Operations Research, ACM TOMACS, and IISE Transactions.