2014 IEEE 34th International Conference on Distributed Computing Systems

Exploring the Use of Diverse Replicas for Big Location Tracking Data

Ye Ding†, Haoyu Tan†‡, Wuman Luo†‡, Lionel M. Ni†‡
† *Department of Computer Science and Engineering* ‡ *Guangzhou HKUST Fok Ying Tung Research Institute*
The Hong Kong University of Science and Technology, Hong Kong SAR
{*valency, hytan, luowuman, ni*}*@cse.ust.hk*

***Abstract*—The value of large amount of location tracking data has received wide attention in many applications including human behavior analysis, urban transportation planning, and various location-based services (LBS). Nowadays, both scientific and industrial communities are encouraged to collect as much location tracking data as possible, which brings about two issues: 1) it is challenging to process the queries on big location tracking data efficiently, and 2) it is expensive to store several exact data replicas for fault-tolerance. So far, several dedicated storage systems have been proposed to address these issues. However, they do not work well when the query ranges vary widely. In this paper, we present the design of a storage system using diverse replica scheme which improves the query processing efficiency with reduced cost of storage space. To the best of our knowledge, we are the first to investigate the data storage and processing in the context of big location tracking data. Specifically, we conduct in-depth theoretical and empirical analysis of the trade-offs between different spatio-temporal partitioning schemes as well as data encoding schemes. Then we propose an effective approach to select an appropriate set of diverse replicas, which is optimized for the expected query loads while conforming to the given storage space budget. The experiment results confirm that using diverse replicas can significantly improve the overall query performance. The results also demonstrate that the proposed algorithms for the replica selection problem is both effective and efficient.**

I. Introduction

With the popularity of location services in billions of electronic devices including mobile phones, tablet computers, vehicle GPS navigators, and a wide variety of sensors, it is now possible to obtain their location logs of a huge number of users or objects. To name just a few, telecom operators continuously record the locations of active mobile phones; taxi companies monitor the locations of taxis; Location-based service (LBS) providers keep the location information of the users whenever they use the services. Such datasets are valuable for many emerging applications such as human behavior analysis, urban transportation planning, customized routing recommendation, location-based advertising and marketing, etc.

Although the above datasets are generated from different sources, we collectively refer to them as *big location tracking data* because they share the following three common characteristics. First, all these datasets have at least three core attributes: object ID, timestamp, and location. They may as well contain other attributes called common attributes that can vary among datasets. Second, most queries on these datasets are associated with spatial and temporal ranges. Therefore, efficient indexing schemes for range data filtering is required to improve overall query performance. Third, mainstream big data storage and management systems (e.g., HDFS, parallel RDBMSs, NoSQL datasets, etc) are not suitable for storing and processing these data. This is because these systems do not naturally lend themselves to dealing with spatio-temporal range queries, especially when the number of the result records is very large. The main reason is that they cannot physically co-cluster records according to spatial and temporal proximity, which leads to too many slow random disk accesses.

In recent years, several dedicated storage systems have been proposed to store big location tracking data. Examples include TrajStore [2] and CloST [1]. In these systems, data are partitioned in terms of spatial and temporal attributes. The records in the same partition are physically stored together. To process a range query, we only need to sequentially scan the partitions whose range intersects with the query range. It is demonstrated that this approach is much more efficient than fetching a large number of non-consecutive disk pages. In addition, these systems can achieve high data compression ratio by leveraging specialized storage structures and encoding schemes.

However, the existing dedicated systems do not work well when the query ranges vary widely. The fundamental reason is that there is only one set of configuration parameters to organize (i.e., partition and compress) the data. It is obvious that we cannot find a single configuration that is optimized for all possible queries. For example, consider that data are partitioned into many small partitions (whose size, in the extreme case, can be as small as a disk page). On one hand, queries with small ranges can be processed efficiently because we can prune most of the partitions. On the other hand, queries with large ranges will incur high I/O costs because a large number of partitions will be involved and locating each of them will invoke a random page access. In this context, these systems have to choose the parameters optimized for the overall performance of the expected query workloads. Note that the expected query workloads can be either derived from historical queries [2][1] or known as a priori knowledge [2].

In this paper, we explore the use of *diverse replicas* in the context of storage systems for big location tracking data. In big data storage systems, e.g., Hadoop HDFS, replication is mainly used for data availability and durability, but not yet for optimizing the performance of query processing. Hence, the use of diverse replicas is a novel approach. The implications of diverse replicas are two-fold. First, data are partitioned and compressed in multiple ways such that different queries can pick the best-fit configuration to min-

1063-6927/14 $31.00 © 2014 IEEE
DOI 10.1109/ICDCS.2014.17

imize the processing time. Second, in spite of the diversity of physical data organizations, diverse replicas can recover each other when failures occur because they share the same logical view of the data. Since we can replace the exact replicas with diverse ones, the gain of query performance does not necessarily come at the cost of more storage space. Though the potential advantages of using diverse replicas are prominent, it is non-trivial to determine which replicas to use. Concretely, given a large location tracking dataset, a representative workload and a constraint on storage space, we need to find an optimal or near-optimal set of diverse replicas in terms of overall query performance. To address this problem, we make the following contributions.

- We propose BLOT, a system abstraction that describes an important class of location tracking data storage systems. Based on the BLOT system abstraction, we conduct general discussions on how to integrate diverse replicas into existing systems.
- We formally define the *replica selection problem* that finds the optimal set of diverse replicas in the context of BLOT systems. Besides, we prove that this problem is at least NP-complete.
- We propose two solutions to the replica selection problem, including an exact algorithm based on integer programming and an approximation algorithm based on greedy strategy. In addition, we propose several practical approaches to reduce the input size of the problem.
- We design a simple yet effective cost model to estimate the cost of an arbitrary query on an arbitrary replica configuration. The parameters of the cost model can be either calculated by closed-form formula or measured accurately by a few low-cost experiments.
- We evaluate our solutions using two typical deploy environments of BLOT systems. The experiment results confirm that using diverse replicas can significantly improve the overall query performance. The results also demonstrate that the proposed algorithms for the replica selection problem is both effective and efficient.

The rest of this paper is organized as follows. Section II presents the common designs of BLOT systems as well as the general use of diverse replicas. Section III defines the replica selection problem, proves its hardness, and describes the solutions. Section IV presents the query cost estimation model for BLOT systems. Section V shows the experiment results and conducts analysis. Section VI briefly summarizes the related works and Section VII concludes the paper.

II. BLOT Systems and Diverse Replicas

In this section, we introduce *BLOT*, a system abstraction that reflects common designs of an important class of dedicated systems for storing big location tracking data. We refer to such systems as *BLOT systems*. Figure 1 shows an overview of how data are organized and queried in BLOT systems.

BLOT systems are primarily aimed at providing a storage layer that supports efficient data filtering by spatio-temporal ranges for high-level data analytical systems such as RDBMSs and Hadoop. They can be also used as standalone systems to dedicatedly answer range queries. The advantages of BLOT systems have been demonstrated by a number of existing works such as PIST, TrajStore, CloST and Spatial Hadoop. Compared with other solutions (e.g., using the original Hadoop or NoSQL databases), the speed of range queries in a BLOT system can be up to one to two orders of magnitude faster while using a much smaller storage space (typically 20% or less). In the rest of this section, we will first describe the general design of BLOT systems and then explain why using diverse replicas can significantly improve the overall system performance.

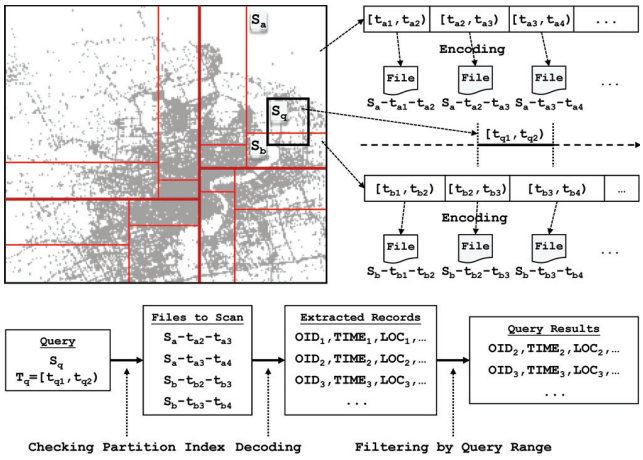


Figure 1. Overview of BLOT systems

A. Data Model

A BLOT system stores a large number of location tracking records. Each record is in the form of $(\texttt{OID}, \texttt{TIME}, \texttt{LOC}, A_1, \ldots, A_m)$, where `OID` is an object ID, `TIME` is a timestamp, `LOC` is the location of object `OID` at time `TIME`, and A_1 through A_m are other attributes that can vary among different datasets. We refer to the first three attributes as *core attributes* and the others as *common attributes*. Any dataset that naturally fits into this data model, i.e., containing and emphasizing core attributes, can be viewed as location tracking data.

B. Data Partitioning

Based on the data model, BLOT systems split a large dataset into relatively small partitions using core attributes. In TrajStore and CloST, for example, data are first partitioned by location (`LOC`) and then further partitioned by time (`TIME`). Records in the same partition are stored together in a storage unit which is optimized for sequential read. For instances, a storage unit can be an object stored in Amazon S3, a file on HDFS, a segment of a file on a local file system, etc. Typically, the size of a storage unit in BLOT systems is much larger than that of a disk page, ranging from hundreds of kilobytes to several megabytes. The advantages of using relatively large storage units is two-fold. First, queries with

large spatio-temporal ranges can be efficiently processed because data are mostly accessed sequentially. Second, it makes the number of storage units sufficiently small such that we can easily maintain the *partitioning index*, a small global data structure to index the spatio-temporal ranges of all data partitions.

C. Data Encoding

A data partition can be stored in any format. A popular approach is to store each partition as a CSV file with each line specifying a record. While this format is easy to process, the storage utilization is low. It is therefore undesirable for huge datasets, especially when using cloud storage systems that charge for every bit stored. To reduce the storage size, a BLOT system usually use various compression techniques to encode records in a partition. For example, we can 1) use binary format instead of text format, 2) apply a general compression algorithm such as Gzip to compress the entire partition, or 3) organize the data in column fashion and then apply column-wise encoding schemes (e.g., delta encoding and run-length encoding). Moreover, we can use the combinations of the above techniques to further reduce the storage size. Note that higher compression ratio comes at the cost of longer decompression time which may degrade the performance of query processing.

D. Query Processing

To process range queries in a BLOT system, we first search for *involved partitions*, i.e., the partitions whose range intersect with the query range. Next, we read and decompress each involved partition to extract all the records. Finally, we check the extracted records and output the ones within the query range. Note that it is straightforward to conduct parallel query processing by scanning multiple partitions simultaneously.

In general, the cost of processing an involved partition consists of two parts: scan cost which includes the cost of extracting and filtering the records, and extra cost which includes the cost of initializing the procedure, locating the partition, loading the decoder, cleaning up the procedure, etc. In a typical BLOT system, scan cost is usually proportional to the total number of records in the partition while extra cost is usually a constant decided by the corresponding encoding scheme. Therefore, for a specific query, the query cost is determined by the total amount of records to be scanned and the total number of involved partitions. Consider three partitioning schemes and a query shown in the upper part of Figure 2. For illustration purpose, we omit the temporal dimension and highlight the not involved partitions. The table below compares the number of involved partitions (N_p) and the estimated percentage of data to be scanned (S) among the three cases.

	Left Case	Middle Case	Right Case
N_p	4	3	8
S	100%	30%	50%

In this example, it is obvious that the middle case has the lowest query cost because both the scan cost and the extra

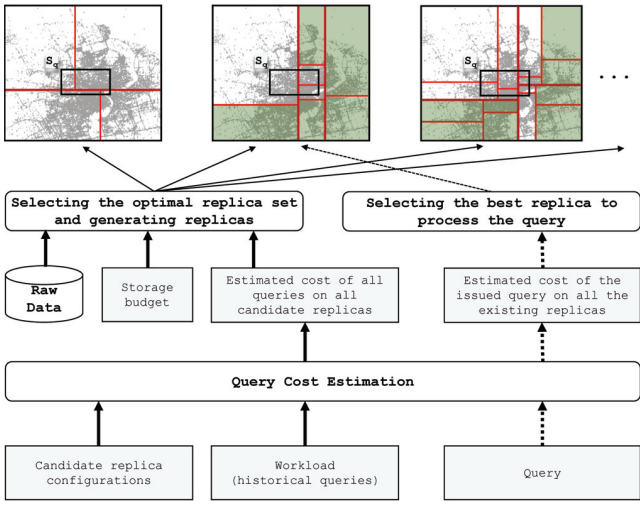


Figure 2. Using diverse replicas in BLOT systems

cost are the lowest. However, it is unclear whether the query cost of the left case is higher or lower than that of the right case. To answer that question, we will develop an effective cost estimation model in Section IV.

E. Diverse Replicas

From Figure 2 we can see that the cost of a query may vary a lot with different partitioning schemes. Undoubtedly, encoding scheme also has a significant influence on query performance. Most existing BLOT systems can adaptively optimize the configuration of the physical storage organization, such as spatial and temporal partition sizes, based on analyzing the historical queries. However, in the cases when the range of queries have high variation, the optimal configuration may still be far from satisfactory in terms of overall query performance. It is intuitive that using multiple copies of data with different physical organizations can mitigate the "one-size-does-not-fit-all" problem. Traditionally, this is a typical performance tuning approach that trades space for time. However, in the context of big data storage systems where data are replicated for fault-tolerance, we can make better use of the storage space by replacing exact replicas with diverse ones. As a result, the overall query performance can be improved without necessarily using more storage space.

Figure 2 illustrates the use of diverse replicas in BLOT systems. There are two components that are key to the success of such systems. First, the system must be able to estimate the query cost both efficiently and effectively. Query cost estimation helps the system to determine which one of the existing replicas is supposed to have the least processing time for the issued query. For example, in Figure 2, the second replica is chosen to answer the given query. Besides, the estimated costs of all queries in the given workload on all candidate replicas are important inputs for the second component which selects a set of diverse replicas (and generates the actual replicas) that is optimized for

a given workload under a storage constraint. The storage constraint is a hard constraint indicating the upper bound of the available storage space. It turns out that selecting the optimal set of diverse replicas in BLOT systems is a challenging problem. To the best of our knowledge, it has not been well investigated in the previous works. Therefore, we will elaborate on this problem in the next section.

III. Replica Selection Problem

Given a very large location-tracking dataset D, we want to choose a set of diverse replicas which conforms to a storage size constraint and optimizes the overall performance for a given workload. In this section, we first formally define the replica selection problem and then propose several practical solutions.

A. Problem Definition

Before formalizing the replica selection problem, we first give the formal definitions of several important concepts mentioned in Section II.

*Definition 1 (**Partitioning Scheme**):* Let U denote the spatio-temporal bounding box of D. A partitioning scheme $P = \{p_1, p_2, \ldots, p_n\}$ is a partition of U, i.e., $\bigcup_i p_i = U$ and $p_i \cap p_j = \emptyset$ ($i, j \in \{1, 2, \ldots, n\}$ and $i \neq j$). Particularly, p_i is called the ith space partition of U.

*Definition 2 (**Data Partition**):* Given a partitioning scheme P, for any partition $p_i \in P$, the corresponding data partition d_i is the set of all records in D that are spatio-temporally contained by p_i. In addition, we define $D(p_i) = d_i$, $P(d_i) = p_i$, and $D(P) = \{d_i | P(d_i) \in P\}$.

By Definition 1, we have $\bigcup_i d_i = D$ and $d_i \cap d_j = \emptyset$ ($i, j \in \{1, 2, \ldots, |P|\}$ and $i \neq j$). Since it is usually clear from the context, we often use term *partition* to indicate both space partition (e.g., p_i) and data partition (e.g., d_i). In addition, we use $Range(p)$ and $Range(d)$ to denote the spatio-temporal range of a space partition p and that of a data partition d, respectively.

*Definition 3 (**Encoding Scheme**):* Given a set of records d, an encoding scheme E is an algorithm that generates a physical storage layout for d.

*Definition 4 (**Replica and Replica Set**):* A replica $r = \langle D, P, E \rangle$ is a physical organization of all records in D in which records are partitioned by P and each partition is encoded by E. A replica set $R = \{r_1, r_2, \ldots, r_k\}$ is a set of diverse (i.e., unique) replicas.

We use $P(r)$ and $E(r)$ to indicate the partitioning scheme and the encoding scheme of r, respectively. Note that the above definition requires that all partitions are encoded by the same encoding scheme. Nevertheless, the essential theoretical analysis in the following can be easily generalized for BLOT systems that allow a separate encoding scheme for each partition.

*Definition 5 (**Storage Size**):* The storage size of a replica r, denoted by $Storage(r)$, is the size of storage space required to store all encoded partitions in r. The storage size of a replica set R, denoted by $Storage(R)$, is the total storage size of all replicas in R, i.e., $Storage(R) = \sum_{r_j \in R} Storage(r_j)$.

*Definition 6 (**Query and Workload**):* A (range) query $q = \langle W, H, T, x, y, t \rangle$ is a process that extracts all records in D that are contained by a cuboid whose size is specified by W, H, T and centroid is specified by x, y, t. A workload $W = \{(q_1, w_1), (q_2, w_2), \ldots, (q_n, w_n)\}$ is a set of unique queries with each query associated with a non-negative weight.

Similar to $Range(p)$ and $Range(d)$, we use $Range(q)$ to denote the spatio-temporal range of q, i.e., $\langle W, H, T, x, y, t \rangle$. The weight of a query in a workload can be interpreted as the importance (frequency, priority, etc.) of the query. In some situations, the weights are normalized such that $\sum_{i=1}^{n} w_i = 1$. In particular, we use $Q(W)$ to denote the set of all queries in W, i.e., $Q(W) = \{q_1, q_2, \ldots, q_n\}$.

Below we define query cost and workload cost based on the query processing mechanism described in Section II-D.

*Definition 7 (**Query Cost and Workload Cost**):* Given r, R, q, and W :

- $Cost(q, r)$ is the cost of evaluating q on r;
- $Cost(q, R) = \min_{r_j \in R}\{(Cost(q, r_j)\}$;
- $Cost(W, R) = \sum_{(q_i, w_i) \in W} w_i * Cost(q_i, R)$.

Now we can formally define the problem of finding an optimal set of diverse replicas.

Replica Selection Problem. Given a dataset D, a workload $W = \{(q_1, w_1), (q_2, w_2), \ldots, (q_n, w_n)\}$, a set of candidate replicas $R^C = \{r_1, r_2, \ldots, r_m\}$, and a storage budget b, find a replica set R^* such that: 1) $R^* \subseteq R^C$; 2) $Storage(R^*) \leq b$; and 3) $Cost(W, R^*) \leq Cost(W, R')$ for all $R' \subseteq R^C$ such that $Storage(R') \leq b$.

In most situations, R^C contains all possible replicas, i.e., if we have m_P partitioning schemes and m_E encoding schemes, then $m = m_P * m_E$.

To find the optimal replica set R^*, we need to know the query cost $Cost(q_i, r_j)$ and the storage size $Storage(r_j)$ for all $q_i \in Q(W)$ and $r_j \in R^C$ in the first place. For $Storage(r_j)$, we can estimate it using the compression ratio of the corresponding encoding scheme $E(r_j)$. Since compression ratio is stable in most situations, it can be effectively measured with a small sample of D. For $Cost(q_i, r_j)$, we will propose a highly accurate cost model in Section IV to estimate query cost without generating actual replicas.

For the rest of this section, we assume that all $Cost(q_i, r_j)$ and $Storage(r_j)$ are already given and focus on designing practical algorithms to solve the problem.

B. Exact Solution

Before presenting the exact solution, we first prove the following theorem.

*Theorem 1 (**NP-completeness**):* The replica selection problem is at least NP-complete.

Proof: We prove the theorem by reducing from the set covering decision problem [3] to the replica selection problem. Specifically, given a set of n elements $U = \{x_1, x_2, \ldots, x_n\}$, a set of m sets $S = \{s_1, s_2, \ldots, s_m\}$ where $\bigcup_{s_i \in S} s_i = U$, and an integer $k \in \{1, 2, \ldots, n\}$, the set covering decision problem is to decide whether there is a set $S' \subseteq S$ such that $|S^*| \leq k$ and $\bigcup_{s_i \in S^*} s_i = U$. This is a well-known NP-complete problem and we will demonstrate that we can solve any instance of the set covering decision problem by constructing and solving an instance of the replica selection problem.

First, in correspondence to U, we construct a workload $W = \{(q_1, 1), (q_2, 1), \ldots, (q_n, 1)\}$ where all weights are set to 1. Then, for each $s_j \in S$, we construct a replica r_j and set $Storage(r_j) = 1$. Hence, there are m candidate replicas in R^C. The query cost is set as follows: 1) if $x_i \in s_j$, then $Cost(q_i, r_j) = 0$; 2) if $x_i \notin s_j$, then $Cost(q_i, r_j) = +\infty$. We can interpret $Cost(q_i, r_j) = 0$ as that q_i can be answered instantly on r_j and $Cost(q_i, r_j) = +\infty$ as that q_i cannot be answered on r_j (because it will take forever). Last, we set the storage budget b equal to k. For the ease of presentation, we use problem A and problem B to denote the instance of the set covering decision problem and the corresponding instance of the replica selection problem, respectively.

Suppose that we have found an optimal replica set R^* in problem B. We can then construct the corresponding set cover $S^* = \{s_j |$ for all j such that $r_j \in R^*\}$ in problem A. To decide whether problem A is feasible, we need to discuss two cases. On one hand, if $Cost(W, R^*) = 0$ in problem B, then any query in $Q(W)$ can be answered instantly by some replica in R^*. According to our construction process from problem A to problem B, it follows that any element in U must be covered by some set in S^*. In this case, we can safely conclude that problem A is feasible. On the other hand, if $Cost(W, R^*) = +\infty$ in problem B, then we prove that problem A is infeasible by contradiction. Assume S^{**} is a feasible solution to problem A. We can then construct a replica set $R^{**} = \{r_j |$ for all j such that $s_j \in S^{**}\}$ for problem B. We can easily verify that $Cost(W, R^{**}) = 0$, which follows that $Cost(W, R^{**}) < Cost(W, R^*)$. This contradicts the fact that R^* is an optimal replica set in problem B.

Thus, we have proved that problem A is feasible if and only if the optimal workload cost in the corresponding problem B equals 0. We therefore conclude that problem A can be reduced to problem B, which implies that the replica selection problem is at least as hard as the set covering decision problem. This completes the proof. ■

Though Theorem 1 eliminates the possibility of finding the optimal replica set in polynomial time, an exact solution is still useful when the input size is relatively small. Our exact solution is to model the original problem as a 0-1 Mixed Integer Programming (MIP) problem and hand it over to a MIP solver. The challenge here is how to model the problem properly to ensure that the optimal solution of the 0-1 MIP problem is the optimal solution of the original problem.

Let $n = |W|$ and $m = |R^C|$. For any $i \in \{1, 2, \ldots, n\}$ and any $j \in \{1, 2, \ldots, m\}$, let x_j be a 0-1 variable indicating whether replica r_j is present in the replica set R, y_{ij} be a 0-1 variable indicating whether query q_i is processed on replica r_j. We first list the constraints as following.

The constraint related to storage size is:

$$Storage(R) = \sum_{j=1}^{m} Storage(r_j) \leq b \ . \tag{1}$$

We use exactly one replica to process each query:

$$\sum_{j=1}^{m} y_{ij} = 1 \text{ for all } i. \tag{2}$$

Any replica that is chosen to process at least one query must be present in R:

$$y_{ij} \leq x_j \text{ for all } i, j. \tag{3}$$

We can see that Equation (3) specifies $n \times m$ constraints. Because an MIP problem may become extremely difficult in the presence of too many constraints, it is preferable to use fewer constraints. Therefore, we use the following m constraints instead (which are slightly relaxed but do not change the optimal solution):

$$\sum_{i=1}^{n} y_{ij} \leq n \cdot x_j \text{ for all } j. \tag{4}$$

Let $c_{ij} = Cost(q_i, r_j)$. We use the following objective function:

$$\sum_{i=1}^{n} \sum_{j=1}^{m} w_i \cdot c_{ij} \cdot y_{ij} \ , \tag{5}$$

Putting it together, we need to minimize Equation (5) subject to the constraints specified by Equation (1), (2) and (4). As x_{ij} and y_j are 0-1 variables, this is a well-formed 0-1 MIP problem that can be solved directly by MIP solvers (in seconds or minutes, hopefully). It remains to prove that the optimal solution of the formulated MIP problem and the optimal solution of the replica selection problem are essentially the same. Due to the page limitation, we do not include the proof in this paper. Interested readers may refer to the proof of the optimality of the MIP formulation for the well-studied *uncapacitated facility location problem* (UFLP) [4] because our proof has a very similar structure.

C. Reducing the Problem Size

In general, the computation time of solving an MIP problem grows exponentially with the problem size, i.e., the number of decision variables. The total number of decision variables in our formulation is $m(n + 1)$ (all x_{ij} and y_j) which could be very large even both m and n are relatively small. For example, there are more than 10^5 decision variables when we have 20 partitioning schemes, 5 encoding schemes, and 1000 queries in the given workload. Though this is a typical scenario in practice, it already makes the formulated MIP problem computationally infeasible (on up-to-date computers nowadays). Thus, to make the

aforementioned solution more scalable, we propose several practical techniques that can significantly reduce the problem size.

1) Reducing the Workload Size: If we directly use all historical queries recorded in the query log to form the input workload, then m may increase too fast in a working system where new queries are issued frequently. To address this issue, we treat each $q \in Q(W)$ as a group of similar queries. Specifically, we use only one *grouped query*, denoted by Q^G, to represent all the queries with the same size of spatio-temporal range. Accordingly, we adjust the definition of query in Definition 6 by replacing $q = \langle W, H, T, x, y, t\rangle$ with $Q^G = \langle W, H, T\rangle$. This variation reflects the observation that queries with the same size of range often occurs many times in real situations. For example, it is common that users use a equal-sized grid to decompose the space and then conduct simple statistics for each grid cell. It is worth pointing out that estimating the cost of a grouped query is generally more difficult than estimating a single query. We will address this issue in Section IV. In addition, if the number of different range sizes is still large, we can use clustering algorithms such as K-means to cluster the range sizes and only use the cluster centers to construct the input workload. In this way, we have full control of the value of m by manipulating the number of clusters.

2) Reducing the Number of Candidate Replicas: Consider two replicas $r_1, r_2 \in R^C$ satisfying $Storage(r_1) \leq Storage(r_2)$ and $Cost(q_i, r_1) \leq Cost(q_i, r_2)$ for all $q_i \in Q(W)$. We refer to this case as *replica r_1 dominates replica r_2*. Obviously, if we use $R^C - \{r_2\}$ instead of R^C as the input candidate replicas, it will not change the optimal workload cost $Cost(W, R^*)$. Therefore, we can safely prune r_2 from R^C. In general, it is more common that a replica is dominated by a set of replicas. Concretely, given a replica $r \in R^C$ and a replica set $R \subseteq R^C$, we say that *replica set R dominates replica r* if $r \notin R$, $Storage(R) \leq Storage(r)$ and $Cost(q_i, r_) \leq Cost(q_i, R)$ for all $q_i \in Q(W)$.

Ideally, we want to find a minimum *dominant replica set* $R^C_{dom} \subseteq R^C$ such that for any replica $r \in R^C - R^C_{dom}$, there exists a replica set $R \subseteq R^C_{dom}$ that dominates it. However, as we can prove that this problem itself is at least NP-complete, we do not pursue a minimum R^C_{dom} in practice. Instead, we use a rough yet effective heuristic algorithm to find a sub-optimal dominant replica set. Due to the page limitation, we will not go into more details in this direction.

D. Greedy Strategy

In addition to the exact solution, we also propose a fast greedy algorithm to select a near-optimal set of replicas. This algorithm is suitable in case that the number of candidate replicas is still large after pruning, or the workload is changing rapidly so that the replica set should be re-selected frequently. As shown in Algorithm 1, we add one replica at a time to the replica set R such that the last added replica r maximizes $(Cost(W, R) - Cost(W, R \cup \{r\}))/Storage(r)$, until the storage budget is exhausted or the overall workload cost $Cost(W, R)$ cannot be further decreased by adding any one of the remaining replicas. Though the greedy strategy is simple yet effective, finding an upper bound of the approximation ratio for Algorithm 1 remains an open problem at the time of writing this paper and we leave it to our future work. In Section V, we will see that the approximation ratio of the greedy algorithm is quite desirable (lower than 1.3 in most cases) in practice.

Algorithm 1: A Greedy Replica Selection Algorithm

```
Data: W, R^C, b, Cost(q_i, r_j) for all q_i ∈ Q(W) and r_j ∈ R^C
Result: R
1  begin
2      R ⟵ ∅;
3      size.total ⟵ 0;
4      while StorageSize < b do
5          r.best ⟵ NULL;
6          score.best ⟵ 0;
7          for r ∈ R^C do
8              gain ⟵ Cost(W, R) − Cost(W, R ∪ {r});
9              score ⟵ gain/Storage(r);
10             if score > score.best then
11                 score.best ⟵ score;
12                 r.best ⟵ r;
13         if r.best is NULL then
14             break;
15         else
16             R ⟵ R ∩ {r.best};
17             R^C ⟵ R^C − {r.best};
18             size.total ⟵
               size.total + Storage(r.best);
19     return R;
```

IV. Query Cost Estimation

In this section, we propose an effective model to estimate query cost for the replica selection problem.

A. Cost Model

As described in Section II, to answer a query q on a replica r, a BLOT system scans (the physically stored objects of) all partitions $p \in P(r)$ that satisfies $Range(p) \cap Range(q) \neq \emptyset$ and then filters each record by $Range(q)$. The cost of processing an involved partition p is modeled by:

$$Cost(q, p) = \frac{|D(p)|}{ScanRate} + ExtraTime\ , \qquad (6)$$

where $|D(p)|$ is the number of records in data partition $D(p)$, $ScanRate$ is the scan speed in terms of number of records scanned per unit time, and $ExtraTime$ is the time before and after the actual scan process. For example, if each partition is stored continuously as a regular file on a local disk, then $ExtraTime$ is the seek time of locating the beginning of the file and $ScanRate$ is the transfer rate of the disk (assuming that CPU always waits for IO). As another example, if each partition is stored

as an object on Amazon S3 and queries are processed on Amazon EMR (Elastic MapReduce), then $ExtraTime$ is the time initializing the map task plus the time locating the S3 object before scanning the partition. The value of $ScanRate$ depends on the encoding scheme $E(r)$. In real situations, a high compression ratio generally leads to a slow scan speed.

In the next, we assume that all candidate partitioning schemes will generate non-skewed data partitions. In other words, $|D(p_i)|$ are almost the same for all $p_i \in P(r)$. Non-skewed partitioning is a desirable property when partitions are processed in parallel (e.g., in MapReduce). An example of such partitioning schemes is using a k-d tree to partition the space where data are split equally each time the space is subdivided.

Let $N_p(q, r)$ denote the number of partitions to scan when processing q on r. The query cost can be calculated as:

$$\begin{aligned} Cost(q, r) &= \sum_{p_i \in P(r)} Cost(q, p_i) \\ &= N_p(q, r) \cdot \left(\frac{|D|}{|P(r)| \cdot ScanRate} + ExtraTime \right) \\ &= \frac{N_p(q, r)}{|P(r)|} \cdot \frac{|D|}{ScanRate} + N_p(q, r) \cdot ExtraTime \, . \end{aligned} \tag{7}$$

In Equation (7), we already know the values of $|D|$ and $|P(r)|$. In addition, $ScanRate$ and $ExtraTime$ are stable parameters that can be estimated by empirical study or experimental measurement. Therefore, to complete the cost model, what remains to do is to decide $N_p(q, r)$, the number of partitions to scan.

B. Number of Partitions to Scan

When q is a single query of which both the size and the position of the range are given, it is straightforward to compute $N_p(q, r)$ by iterating over all partitions to count the number of partitions whose range intersects with the range of q. In practice, to reduce the size of the input workload, it is more often that we use grouped queries of which only the range size is specified (see Section III-C). For a grouped query Q^G, we need to estimate the average number of partitions to scan, which will be discussed in the following.

Without loss of generality, let the coordinates of U (see Definition 1) start from zero and the size of U be $\langle W^U, H^U, T^U \rangle$, i.e., $U = \langle W^U, H^U, T^U, x^U, y^U, t^U \rangle$ where $x^U = W/2$, $y^U = H/2$, and $t^U = T/2$. We assume that the range position of the queries with the same range size has a uniform distribution. Concretely, for all queries q_i represented by $Q^G = \langle W, H, T \rangle$, their centroids $C(q_i) = (x_i, y_i, t_i)$ are uniformly distributed within cuboid $CR(Q^G) = \langle W^U - W, H^U - H, T^U - T, x^U, y^U, t^U \rangle$ (CR stands for *centroid range*).

For any $q \in Q^G$, we can rewrite $N_p(q, r)$ as $N_p(Q^G, C(q), r)$. Then, the average number of partitions to scan when processing Q^G on r can be formulated as Equation (8), where $Volume(CR(Q^G)) = (W^U - W)(H^U - H)(T^U - T)$:

$$\begin{aligned} N_p(Q^G, r) &= \int_{C(q) \in CR(Q^G)} N_p(Q^G, C(q), r) p(C(q)) \, \mathrm{d}C(q) \\ &= \frac{\int_{C(q) \in CR(Q^G)} N_p(Q^G, C(q), r) \, \mathrm{d}C(q)}{Volume(CR(Q^G))} \, . \end{aligned} \tag{8}$$

However, it is very difficult to compute $N_p(Q^G, r)$ using Equation (8). The main reason is that $N_p(Q^G, C(q), r)$ is a piecewise integer-valued function with too many small pieces. We argue that it would be infeasible, or at least extremely inefficient, to decompose it into pieces and integrate each piece separately. Hence, we will derive another formula for $N_p(Q^G, r)$ which is easier to compute.

To help understand the idea, we first consider a grouped query Q which is a finite set of queries. For a partition $p \in P(r)$ and a query $q \in Q$, we define a binary function indicating whether their ranges intersect:

$$I(p, q) = \begin{cases} 0 & \text{if } Range(p) \cap Range(q) = \emptyset \, , \\ 1 & \text{if } Range(p) \cap Range(q) \neq \emptyset \, . \end{cases} \tag{9}$$

Then, it is straightforward to have:

$$\begin{aligned} N_p(Q, r) &= \frac{1}{|Q|} \sum_{q_i \in Q} \sum_{p_j \in P(r)} I(p_j, q_i) \\ &= \sum_{p_j \in P(r)} \left(\frac{1}{|Q|} \sum_{q_i \in Q} I(p_j, q_i) \right) \, . \end{aligned} \tag{10}$$

Let $\mathbf{q}$ be a random query in Q and set $P\{\mathbf{q} = q_i\} = 1/|Q|$ for all q_i. We can rewrite the above formula as:

$$\begin{aligned} N_p(Q, r) &= \sum_{p_j \in P(r)} \sum_{q_i \in Q} I(p_j, q_i) \cdot P\{\mathbf{q} = q_i\} \\ &= \sum_{p_j \in P(r)} \mathrm{E}[I(p_j, \mathbf{q})] \\ &= \sum_{p_j \in P(r)} P\{I(p_j, \mathbf{q}) = 1\} \, . \end{aligned} \tag{11}$$

When $Q = Q^G$ which is an infinite set, it is easy to verify that Equation (11) still holds if we set $P\{\mathbf{q} = q_i\} = 1/Volume(CR(Q^G))$ for all q_i.

What remains now is to calculate $P\{I(p_j, \mathbf{q}) = 1\}$, the probability that the range of a random query $\mathbf{q}$ intersects with the range of a fixed partition p_j. Let $CR(Q^G, p_j)$ denote the 3-D region such that if $C(q) \in CR(Q^G, p_j)$ for any $q \in Q^G$, then $Range(q) \cap Range(r_j) \neq \emptyset$. Since $C(q)$ is uniformly distributed in $CR(Q^G)$, we have:

$$P\{I(p_j, \mathbf{q}) = 1\} = \frac{Volume(CR(Q^G, p_j))}{Volume(CR(Q^G))} \, . \tag{12}$$

Note that $CR(Q^G, p_j)$ is a cuboid whose boundaries in the first spatial dimension are:

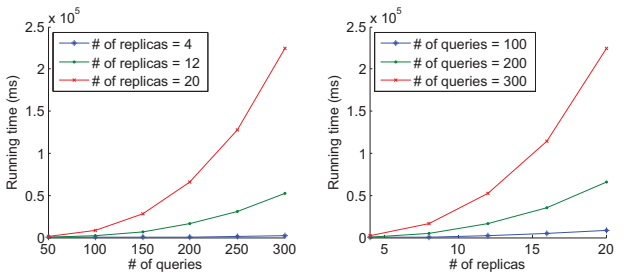
$$\begin{aligned} west(CR(Q^G, p_j)) &= \max\{\frac{W}{2}, west(p_j) - \frac{W}{2}\} \, , \\ east(CR(Q^G, p_j)) &= \min\{W^U - \frac{W}{2}, east(p_j) + \frac{W}{2}\} \, . \end{aligned}$$

Table I
COMPRESSION RATIO

Uncompressed		Snappy		GZip		LZMA2	
Row	Col	Row	Col	Row	Col	Row	Col
1	0.557	0.485	0.312	0.283	0.179	0.213	0.156

Table II
MEASURED $ScanRate$ AND $ExtraCost$

		Amazon S3 and EMR	
		$1/ScanRate$ (ms)	$ExtraCost$ (ms)
Uncompressed	Row	85.02	32689
	Col	N/A	N/A
Snappy	Row	90.24	30187
	Col	56.98	30518
Gzip	Row	90.65	28698
	Col	51.72	28725
LZMA2	Row	54.39	29029
	Col	38.69	29609
		Local Hadoop Cluster	
		$1/ScanRate$ (ms)	$ExtraCost$ (ms)
Uncompressed	Row	606.78	5312
	Col	N/A	N/A
Snappy	Row	598.84	5316
	Col	175.75	4150
Gzip	Row	488.32	5349
	Col	177.15	4427
LZMA2	Row	265.41	5244
	Col	159.98	4551



(a) Varying size of workload (b) Varying size of candidate replicas

Figure 3. Comparison of the computation speed of MIP

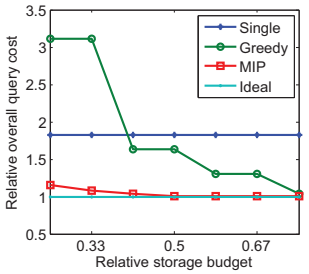


Figure 4. Relative overall query performance of different storage budgets

The formulas for the other boundaries of $CR(Q^G, p_j))$ can be derived similarly, after which we can directly compute $Volume(CR(Q^G, p_j))$.

Putting Equation (11), (12) and (7) together, we can compute the cost of any query on replica r in $O(|P(r)|)$ time. It follows that the time complexity of computing all query costs is $O(|W| \cdot |R^C| \cdot \max_{r_j \in R^C}\{|P(r_j)|\})$.

V. EVALUATION

In this section, we describe the experiment settings and present the evaluation results in detail.

A. Experiment Settings

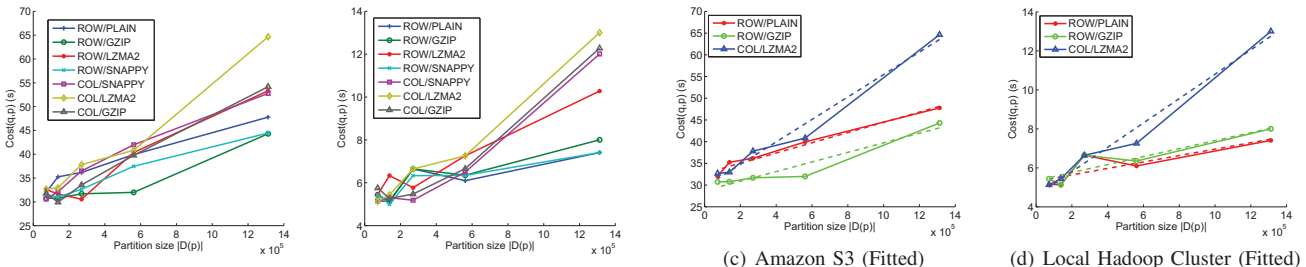
We consider two typical execution environments for BLOT systems. The first one is a local Hadoop cluster where each partition is stored as a separate file on HDFS. The second one uses Amazon S3 to store partitions. To process a query, we lunch a map-only MapReduce job, either in local cluster or in Amazon EMR, with each mapper scanning exactly one of the involved partitions. The dataset we use is a sample of vehicle GPS log collected from more than 4,000 taxis in Shanghai during a month. Each record contains 8 attributes (including the 3 core attributes). The total number of records is around 65 million and the total storage size in uncompressed CSV format is 3.7 GB. The latitude ranges from 30 to 32, longitude from 120 to 122, and time from 11/01/2007 to 11/29/2007. It is worth point out that though the full dataset in our working system is more than 100 GB, we only need a small portion of the data to build the cost model and select diverse replicas for the whole dataset.

For data partitioning, we first partition the space then the time to generate equal-sized (in terms of number of records) partitions. The space is partitioned according to a k-d tree [5] index which recursively decomposes the space by alternatively using each space dimension. The number of spatial partitions is chosen from 4^2, 4^3, 4^4, 4^5, 4^6 and the number of temporal partitions is chosen from 2^4, 2^5, 2^6, 2^7, 2^8. Therefore, there are $5 \times 5 = 25$ candidate spatio-temporal partitioning schemes in total. For data encoding, we store data either by row or by column (with delta encoding), with an option of whether or not using a general compression method chosen from Gzip, Snappy and LZMA2. Since uncompressed column-store has poor performance in terms of both compression ratio and scan speed, we do not use it as a candidate encoding scheme. Therefore, there are $2 \times 4 - 1 = 7$ candidate encoding schemes in total. The compression ratio of each encoding schemes measured on our dataset is listed in Table I. Putting the above partitioning schemes and the encoding schemes together, the total number of candidate replicas is $25 \times 7 = 150$.

B. Measuring $ScanRate$ and $ExtraCost$

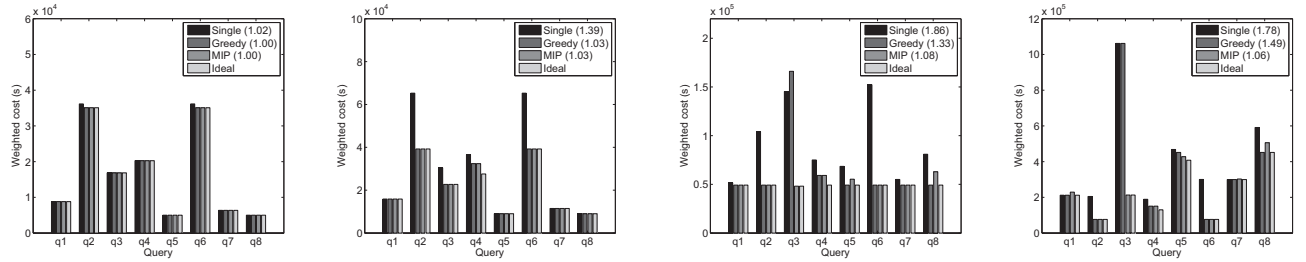
Since $ScanRate$ and $ExtraCost$ are constants with respect to encoding schemes, we conduct $7 \times 2 = 14$ measurements corresponding to 7 candidate encoding schemes in each execution environments, respectively.

For each measurement, we generate 5 sets of partitions with each set containing 20 partitions. The size of partitions within a partition set are the same while they are different across partition sets. We then lunch a map-only MapReduce job with 20 mappers with each scanning a partition. After the job is finished, we compute the average processing time of all mappers and use it as the (measured) value of $Cost(q, p)$



(a) Amazon S3 (b) Local Hadoop Cluster (c) Amazon S3 (Fitted) (d) Local Hadoop Cluster (Fitted)

Figure 5. Measurement results of $Cost(q, p)$



(a) Data size: 3.7 GB (b) Data size: 37 GB (c) Data size: 370 GB (d) Data size: 3,700 GB

Figure 6. Relative overall query performance in Amazon S3 and EMR

in Equation (6). Accordingly, we use the corresponding partition size (in terms of number of records) as $|D(p)|$. We therefore have 5 measured points for Equation (6). In the last step, we perform linear regression to fit the measured points and use the fitted parameters as $1/ScanRate$ and $ExtraTime$.

In Figure 5, the left two sub-figures shows all the measurement results and the right two sub-figures shows the fitted lines for three measurements in each execution environments. In addition, the measured values of $ScanRate$ and $ExtraTime$ are listed in Table II. We can see that $Cost(q, p)$ is well-fitted by Equation (6) especially when the size of partition is relatively large, which demonstrates the effectiveness of our cost model.

C. Performance of Replica Selection

To measure the effectiveness and the efficiency of our replica selection algorithms, we construct a synthetic workload containing 8 grouped queries with wildly varied range size. We conduct all the following experiments in the Amazon S3 and EMR execution environment.

Figure 3 compares the computation time via MIP upon different sizes of workload and candidate replicas. When the size of the given workload or candidate replica set increases, we can see that the computation time of the MIP solution increases exponentially. Hence, when the input workload or the candidate replica set is too large, it is desirable to switch to the greedy algorithm which runs in polynomial time.

Figure 6 compares the relative query performance for all the queries when the replica set is selected by different approaches. The storage budget is set to the same as the storage size of 3 exact copies of the optimal single replica. The approximation ratio of each approach is shown in the brackets of the figure (ideal case is always 1.00). It is clear that when the size of data grows, the performance of the greedy algorithm and the MIP solution is closer to the ideal case than a single replica, thus the advantages of using diverse replicas become more and more prominent. Figure 4 shows the overall query performance relative to the ideal case when varying the storage budget. In this figure, the x-coordinate is the storage budget relative to the storage budget used in Figure 6. We can see that when the MIP solution is close to the ideal case regardless of the storage budget, which is faster than the single replica case by up to 80%. The approximation ratio of the greedy algorithm decreases dramatically as the storage budget increases. When the relative storage budget is greater than 1, the approximation ratio of the greedy algorithm is less than 1.2.

VI. Related Work

There is a plethora of works on storing spatio-temporal data and efficient processing of range queries. Early studies, dating back to 1970s and 1980s, mainly focus on indexing individual points or trajectories. Representative works include k-d tree [5], quadtree [6], R-tree [7] and TB-tree [8]. These data structures incur many random reads which is inefficient when the number of records in the query result is large. To address this issue, TrajStore [2] and PIST [9] attempt to co-locate data according to spatial and temporal proximities and use relatively large partition size. Both

TrajStore and PIST cannot scale to terabytes of data because they can only consider non-distributed environments. CloST [1] and SpatialHadoop [10] are two Hadoop-based systems which aim at providing scalable distributed storage and parallel query processing of big location tracking data. Note that TrajStore, PIST, CloST and SpatialHadoop can be viewed as concrete instances of BLOT systems without using diverse replicas.

Recommending a physical configuration for a given workload has been widely studied since 1987 [11]. Most of the existing works [12][13][14][15][16] propose effective methods to estimate the cost of a given workload over candidate physical configurations. However, only a few of them consider the situations where data can be replicated [15][16]. An earlier work introduces the technique of Fractured Mirrors [17] to store data in both row-fashion and column-fashion. For data partitioning, it has been proved in [18] that finding the optimal vertical partitioning is an NP-hard problem. Therefore, the are a number of works focus on heuristic algorithms for vertical partitioning optimization [19][20][21][22]. For workload size reduction, the authors of [23] propose a workload compression method to reduce the size of SQL workloads. A more scalable workload grouping method is proposed in [15]. Most of the above works are based on the relational data model while our work is based on the BLOT data model which is more suitable for big location tracking data.

VII. Conclusion

In this paper, we explore the use of diverse replicas in the context of storage systems for big location tracking data. Specifically, we propose BLOT, a system abstraction that describes an important class of location tracking data storage systems. Then, we formally define the replica selection problem that finds the optimal set of diverse replicas. We propose two solutions to address this problem, including an exact algorithm based on integer programming, and an approximation algorithm based on greedy strategy. In addition, we propose several practical approaches to reduce the input size of the problem. We also design a simple yet effective cost model to estimate the cost of an arbitrary query on an arbitrary replica configuration. Finally, we evaluate our solutions using two typical execution environments including Amazon and local Hadoop cluster. The results demonstrate that the proposed algorithms for the replica selection problem is both effective and efficient. In this paper, we only consider full replication of the entire data. The use of partial replication, where only frequently accessed data ranges are replicated, is one of our future work.

Acknowledgment

This paper was supported in part by Hong Kong, Macao and Taiwan Science & Technology Cooperation Program of China Grant 2012DFH10010; Nansha S&T Project Grant 2013P015; NSFC Grant 61300031; and the National Key Basic Research and Development Program of China (973) Grant 2014CB340303.

References

[1] H. Tan, W. Luo, and L. M. Ni, "Clost: a hadoop-based storage system for big spatio-temporal data analytics," in *CIKM*, 2012, pp. 2139–2143.

[2] P. Cudré-Mauroux, E. Wu, and S. Madden, "Trajstore: An adaptive storage system for very large trajectory data sets," in *ICDE*, 2010, pp. 109–120.

[3] J. M. Kleinberg and É. Tardos, *Algorithm design*. Addison-Wesley, 2006.

[4] Z. Drezner, *Facility location: a survey of applications and methods*. Springer, 2006.

[5] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[6] H. Samet, "The quadtree and related hierarchical data structures," *ACM Comput. Surv.*, vol. 16, no. 2, pp. 187–260, 1984.

[7] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD Conference*, 1984, pp. 47–57.

[8] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *VLDB*, 2000, pp. 395–406.

[9] V. Botea, D. Mallett, M. A. Nascimento, and J. Sander, "Pist: An efficient and practical indexing technique for historical spatio-temporal point data," *GeoInformatica*, vol. 12, no. 2, pp. 143–168, 2008.

[10] A. Eldawy and M. F. Mokbel, "A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data," *PVLDB*, vol. 6, no. 12, pp. 1230–1233, 2013.

[11] S. Khoshafian, G. P. Copeland, T. Jagodis, H. Boral, and P. Valduriez, "A query processing strategy for the decomposed storage model," in *ICDE*, 1987, pp. 636–643.

[12] N. Bruno and S. Chaudhuri, "Constrained physical design tuning," *VLDB J.*, vol. 19, no. 1, pp. 21–44, 2010.

[13] D. Dash, N. Polyzotis, and A. Ailamaki, "Cophy: A scalable, portable, and interactive index advisor for large workloads," *PVLDB*, vol. 4, no. 6, pp. 362–372, 2011.

[14] H. Kimura, G. Huo, A. Rasin, S. Madden, and S. B. Zdonik, "Coradd: Correlation aware database designer for materialized views and indexes," *PVLDB*, vol. 3, no. 1, pp. 1103–1113, 2010.

[15] A. Jindal, J.-A. Quiané-Ruiz, and J. Dittrich, "Trojan data layouts: right shoes for a running elephant," in *SoCC*, 2011, p. 21.

[16] M. P. Consens, K. Ioannidou, J. LeFevre, and N. Polyzotis, "Divergent physical design tuning for replicated databases," in *SIGMOD Conference*, 2012, pp. 49–60.

[17] R. Ramamurthy, D. J. DeWitt, and Q. Su, "A case for fractured mirrors," *VLDB J.*, vol. 12, no. 2, pp. 89–101, 2003.

[18] D. Saccà and G. Wiederhold, "Database partitioning in a cluster of processors," *ACM Trans. Database Syst.*, vol. 10, no. 1, pp. 29–56, 1985.

[19] S. Agrawal, V. R. Narasayya, and B. Yang, "Integrating vertical and horizontal partitioning into automated physical database design," in *SIGMOD Conference*, 2004, pp. 359–370.

[20] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudré-Mauroux, and S. Madden, "Hyrise - a main memory hybrid storage engine," *PVLDB*, vol. 4, no. 2, pp. 105–116, 2010.

[21] R. A. Hankins and J. M. Patel, "Data morphing: An adaptive, cache-conscious storage technique," in *VLDB*, 2003, pp. 417–428.

[22] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical partitioning algorithms for database design," *ACM Trans. Database Syst.*, vol. 9, no. 4, pp. 680–710, 1984.

[23] S. Chaudhuri, A. K. Gupta, and V. R. Narasayya, "Compressing sql workloads," in *SIGMOD Conference*, 2002, pp. 488–499.