Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 4217432, 17 pages
https://doi.org/10.1155/2018/4217432

Research Article

Diverse Mobile System for Location-Based Mobile Data

Qing Liao,[1] Haoyu Tan,[2] Wuman Luo,[2] and Ye Ding[2]

[1]*Department of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), China*
[2]*Guangzhou HKUST Fok Ying Tung Research Institute, Hong Kong University of Science and Technology, Hong Kong*

Correspondence should be addressed to Ye Ding; dingye@ust.hk

Received 23 March 2018; Accepted 10 July 2018; Published 1 August 2018

Academic Editor: Ziming Zhao

The value of large amount of location-based mobile data has received wide attention in many research fields including human behavior analysis, urban transportation planning, and various location-based services. Nowadays, both scientific and industrial communities are encouraged to collect as much location-based mobile data as possible, which brings two challenges: (1) how to efficiently process the queries of big location-based mobile data and (2) how to reduce the cost of storage services, because it is too 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 ranges of queries vary widely. In this work, we design a storage system based on diverse replica scheme which not only can improve the query processing efficiency but also can reduce the cost of storage space. To the best of our knowledge, this is the first work to investigate the data storage and processing in the context of big location-based mobile data. Specifically, we conduct in-depth theoretical and empirical analysis of the trade-offs between different spatial-temporal partitioning and data encoding schemes. Moreover, 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 show that using diverse replicas can significantly improve the overall query performance and the proposed algorithms for the replica selection problem are both effective and efficient.

1. Introduction

With the development of data collection capabilities, it is much easier to collect a huge number of location-based mobile data of users or objects via billions of electronic devices such as mobile phones, tablet computers, vehicle GPS navigators, and a wide variety of sensors. For example, taxi companies monitor the mobility information of taxis; telecom operators continuously record the locations of active mobile phones; location-based service (LBS) providers keep the mobile information of the users whenever they use the services. Such large amount of location-based mobile data is valuable for many research fields such as human behavior analysis [1], urban transportation planning [2], customized routing recommendation [3, 4], and location-based advertising and marketing [5].

We called the datasets as *location-based mobile data* because they share the following three common characteristics. Firstly, all these datasets have at least three core attributes: object ID, timestamp, and location. They may as well contain other attributes which are called common attributes that can vary among datasets. Secondly, most queries on these datasets are associated with spatial and temporal ranges. Hence, efficient indexing schemes for range data filtering are required to improve overall query performance. Thirdly, mainstream big data storage and management systems (e.g., HDFS, parallel RDBMSs, and NoSQL databases) are not suitable for storing and processing these data. This is because these systems do not naturally lend themselves to dealing with spatial-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-based mobile data, such as TrajStore [6], CloST [7], and Panda [8]. Data are partitioned in terms of spatial and temporal attributes in

the above system. 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 nonconsecutive disk pages. In addition, these systems can achieve high data compression ratio by leveraging specialized storage structures and encoding schemes.

However, the above existing dedicated systems do not work well when the ranges of queries 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 [7] or known as a priori knowledge [6].

In this paper, we explore the use of *diverse replicas* in the context of storage systems for big location-based mobile 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 twofold. First, data are partitioned and compressed in multiple ways such that different queries can pick the best-fit configuration to minimize 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 nontrivial to determine which replicas to use. Concretely, given a large location-based mobile 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:

(i) We propose BLOT, a system abstraction that describes an important class of location-based mobile data storage systems. Based on the BLOT system abstraction, we conduct general discussions on how to integrate diverse replicas into existing systems.

(ii) 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.

(iii) 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.

(iv) 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.

(v) 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 are both effective and efficient.

The rest of this paper is organized as follows. Section 2 briefly summarizes the related works and Section 3 presents the common designs of BLOT systems as well as the general use of diverse replicas. Section 4 defines the replica selection problem, proves its hardness, and describes the solutions. Section 5 presents the query cost estimation model for BLOT systems. Section 6 shows the experiment results and conducts analysis and Section 7 concludes the paper.

2. Related Work

There is a plethora of works on storing spatial-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 [9], quadtree [10], R-tree [11], and TB-tree [12]. These data structures incur many random reads which are inefficient when the number of records in the query result is large. To address this issue, TrajStore [6] and PIST [13] 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 nondistributed environments. CloST [7] and SpatialHadoop [14] are two Hadoop-based systems which aim at providing scalable distributed storage and parallel query processing of big location-based mobile data. SATO [15] is a spatial data partitioning framework that can quickly analyze and partition spatial data with an optimal spatial partitioning strategy for scalable query processing. Note that TrajStore, PIST, CloST, SpatialHadoop, and SATO 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 [16]. Most of the existing works [17–23] 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 [20, 21]. An earlier work introduces the technique of Fractured Mirrors [24]

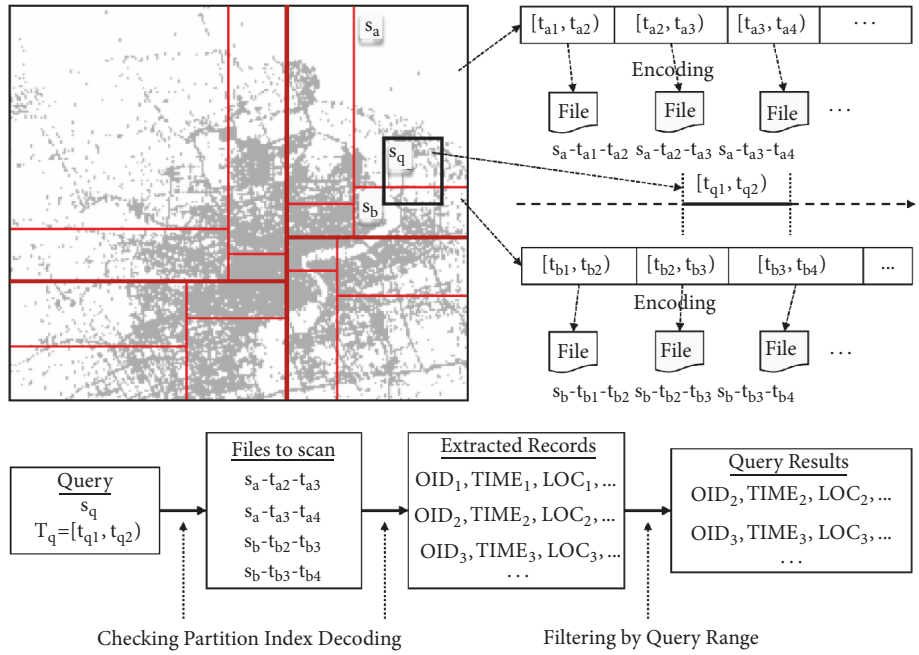


Figure 1: Overview of BLOT systems.

to store data in both row-fashion and column-fashion. For data partitioning, it has been proved in [25] that finding the optimal vertical partitioning is an NP-hard problem. Therefore, the are a number of works that focus on heuristic algorithms for vertical partitioning optimization [26–30]. For workload size reduction, the authors of [31] propose a workload compression method to reduce the size of SQL workloads. A more scalable workload grouping method is proposed in [20]. 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-based mobile data.

3. 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-based mobile 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 spatial-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 [13], TrajStore [6], CloST [7], and Spatial Hadoop [14]. 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.

3.1. Data Model. A BLOT system stores a large number of location-based mobile records. Each record is in the form of (`OID`, `TIME`, `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-based mobile data.

3.2. 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 are twofold. First, queries with large spatial-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

Table 1: Comparison of involved partitions and estimated scans in Figure 2.

	Case 1	Case 2	Case 3
Involved partitions	4	3	8
Estimated data to scan	100%	30%	50%

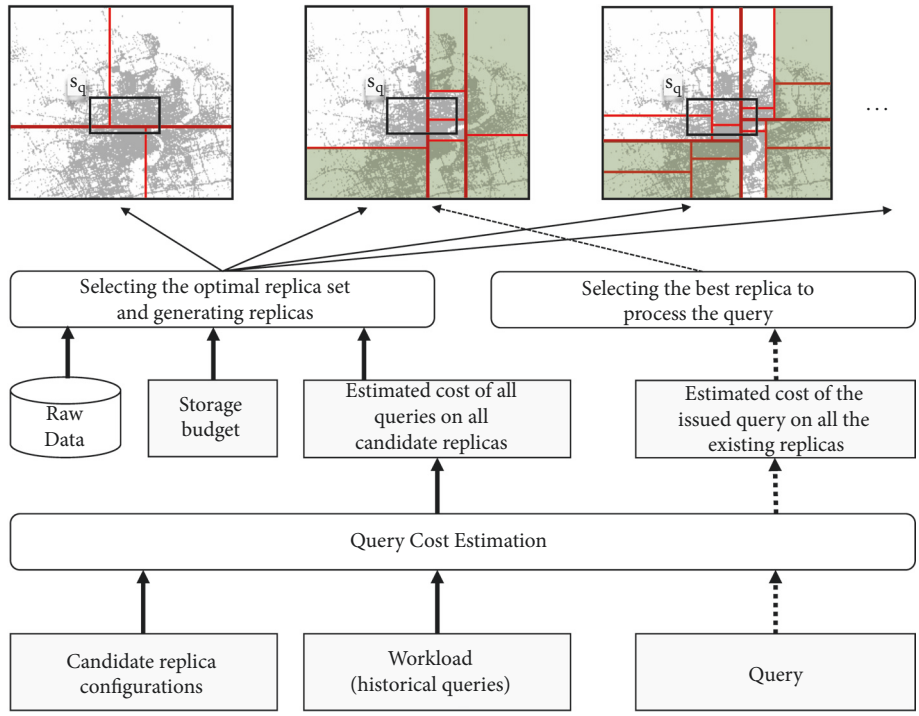


Figure 2: Using diverse replicas in BLOT systems.

partitioning index, a small global data structure to index the spatial-temporal ranges of all data partitions.

3.3. 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 uses 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 to compress the entire partition;

(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.

3.4. 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 partitions that are not involved. Table 1 compares the number of involved partitions and the estimated percentage of data to scan among the three cases.

In this example, it is obvious that the middle case has the lowest query cost because both the scan cost and the extra 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 5.

3.5. 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 has 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.

4. Replica Selection Problem

Given a very large location-based mobile 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.

4.1. Problem Definition. Before formalizing the replica selection problem, we first give the formal definitions of several important concepts mentioned in Sections 3 and 4.

Definition 1 (partitioning scheme). Let U denote the spatial-temporal bounding box of D. A spatial-temporal partitioning scheme $P = \{p_1, p_2, \ldots, p_n\}$ is a spatial-temporal partition of U, where

$$\bigcup_i p_i = U, \tag{1}$$

and

$$p_i \cap p_j = \emptyset, \quad \forall i, j \in \{1, 2, \ldots, n\}, \ i \neq j. \tag{2}$$

Particularly, p_i is called the i-th spatial-temporal 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 spatial-temporally contained by p_i. In addition, we define

(1) $D(p_i) = d_i$;
(2) $P(d_i) = p_i$;
(3) $D(P) = \{d_i \mid P(d_i) \in P\}$.

By Definition 1, we have

$$\bigcup_i d_i = D, \tag{3}$$

and

$$d_i \cap d_j = \emptyset, \quad \forall i, j \in \{1, 2, \ldots, n\}, \ i \neq j. \tag{4}$$

Since it is usually clear from the context, we often use the term *partition* to indicate both spatial-temporal partition (i.e., p_i) and data partition (i.e., d_i). In addition, we use $\mu(p)$ and $\mu(d)$ to denote the spatial-temporal range of a spatial partition p and that of a data partition d, respectively.

Definition 3 (encoding scheme). Given a data partition 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 $R = \{r_1, r_2, \ldots, r_m\}$ 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 $\eta(r)$, is the size of storage space required to store all encoded partitions in r. The storage size of a replica set R, denoted by $\eta(R)$, is the total storage size of all replicas in R, i.e.,

$$\eta(R) = \sum_{r_j \in R} \eta(r_j). \tag{5}$$

Definition 6 (query and workload). A (range) query q is a process that extracts all records in D that are contained by a cuboid whose size is specified by $\langle \delta_x, \delta_y, \delta_t \rangle$ and centroid is specified by $\langle x, y, t \rangle$. 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 $\mu(p)$ and $\mu(d)$, we use $\mu(q)$ to denote the spatial-temporal range of q, i.e., $\langle x, y, t, \delta_x, \delta_y, \delta_t \rangle$. 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

$$\sum_{i=1}^{n} w_i = 1. \tag{6}$$

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 3.4.

Definition 7 (query cost and workload cost). Given a replica $r \in R$ and a query $q \in Q(W)$, the query cost of q on r is denoted as $\rho(q, r)$. Therefore,

$$\rho(q, R) = \min_{r_j \in R} \rho(q, r_j), \tag{7}$$

and

$$\rho(W, R) = \sum_{(q_i, w_i) \in W} w_i \cdot \rho(q_i, R). \tag{8}$$

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

Definition 8 (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 $R = \{r_1, r_2, \ldots, r_m\}$, and a storage budget b, find a replica set R^* such that

(1) $R^* \subseteq R$;

(2) $\eta(R^*) \leq b$;

(3) $\rho(W, R^*) \leq \rho(W, R')$ for all $R' \subseteq R$ such that $\eta(R') \leq b$.

In most situations, R 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 $\rho(q_i, r_j)$ and the storage size $\eta(r_j)$ for all $q_i \in Q(W)$ and $r_j \in R$ in the first place. For $\eta(r_j)$, 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 $\rho(q_i, r_j)$ and we will propose a highly accurate cost model in Section 5 to estimate query cost without generating actual replicas.

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

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

Theorem 9 (NP-hard). *The replica selection problem is NP-Hard.*

Proof. We prove the theorem by reducing from the minimum weight set cover problem [32] to the replica selection problem. Specifically, given a set of n elements $A = \{a_1, a_2, \ldots, a_n\}$, and a set of m sets $S = \{s_1, s_2, \ldots, s_m\}$, where

$$s_i \subseteq A, \quad \forall s_i \in S, \tag{9}$$

and

$$\bigcup_{s_i \in S} s_i = A, \tag{10}$$

the minimum weight set cover problem is to find a set $S^* \subseteq S$ such that

$$\left|S^*\right| \leq |S|, \tag{11}$$

and

$$\bigcup_{s_i \in S^*} s_i = A, \tag{12}$$

and the cost of S^* is minimum where the cost of S^* is defined as

$$\sum_{i=1}^{|S^*|} c_i. \tag{13}$$

where c_i is the cost (weight) of set s_i.

The minimum weight set cover problem is a well-known NP-hard problem [32]. In this proof we will demonstrate that we can solve any instance of the minimum weight set cover problem by constructing and solving an instance of the replica selection problem.

In correspondence to A, we construct a workload $W = \{(q_1, 1), (q_2, 1), \ldots, (q_n, 1)\}$, where all weights are set to 1. In correspondence to S, we construct a set of candidate replicas $R = \{r_1, r_2, \ldots, r_m\}$ where all $\eta(r_j) \in R$ are set to 0. The query cost is set as follows:

(1) $\rho(q_i, r_j) = 0$ if $\rho(q_i, r_j) = \rho(q_i, R)$;

(2) $\rho(q_i, r_j) = +\infty$ if $\rho(q_i, r_j) \neq \rho(q_i, R)$.

According to Definition 7, we can interpret $\rho(q_i, r_j) = 0$ as that answering q_i on r_j requires the minimum query cost, and $\rho(q_i, r_j) = +\infty$ as that answering q_i on r_j requires more query cost than the minimum.

For the ease of presentation, we use problem α and problem β to denote the instance of the minimum weight set cover problem and the corresponding instance of the replica selection problem, respectively.

Suppose that we have found an optimal replica set R^* in problem β. We can then construct the corresponding set $S^* = \{s_j \mid \text{for all } j \text{ such that } r_j \in R^*\}$ in problem α. To decide whether problem α is feasible, we need to discuss

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

Thus, we have proved that problem α is feasible if and only if the optimal workload cost in the corresponding problem β equals 0. We therefore conclude that the replica selection problem is equally hard to the set covering decision problem. This completes the proof. □

Though Theorem 9 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 [33] 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 $m = |R|$. 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 and y_{ij} be a 0-1 variable indicating whether query q_i is processed on replica r_j. We first list the constraints as follows.

The constraint related to storage size is

$$\eta(R) = \sum_{j=1}^{m} \eta\left(r_j\right) \le b. \tag{14}$$

We use exactly one replica to process each query:

$$\sum_{j=1}^{m} y_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, n\}. \tag{15}$$

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

$$y_{ij} \le x_j, \quad \forall i \in \{1, 2, \ldots, n\}, \ j \in \{1, 2, \ldots, m\}. \tag{16}$$

We can see that (16) 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):

$$\sum_{i=1}^{n} y_{ij} \le n \cdot x_j, \quad \forall j \in \{1, 2, \ldots, m\}. \tag{17}$$

Let $c_{ij} = \rho(q_i, r_j)$; we use the following objective function:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} w_i \cdot c_{ij} \cdot y_{ij}. \tag{18}$$

Putting them together, we need to minimize (18) subject to the constraints specified by (14), (15), and (17). The details are shown in the following:

$$\text{minimize:} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} w_i \cdot c_{ij} \cdot y_{ij}, \tag{19}$$

$$\text{subject to:} \quad \sum_{j=1}^{m} \eta\left(r_j\right) \le b,$$

$$\sum_{j=1}^{m} y_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, n\}, \tag{20}$$

$$\sum_{i=1}^{n} y_{ij} \le n \cdot x_j, \quad \forall j \in \{1, 2, \ldots, m\}.$$

As x_j and y_{ij} are 0-1 variables, this is a well-formed 0-1 MIP problem that can be solved directly by MIP solvers.

4.3. 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 though 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.

4.3.1. 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 spatial-temporal range. Accordingly, we adjust the definition of query in Definition 6 by replacing $\mu(q) = \langle x, y, t, \delta_x, \delta_y, \delta_t \rangle$ with $\mu(Q^G) = \langle \delta_x, \delta_y, \delta_t \rangle$. 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 an 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 5. 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.

Input: W, R, b, $\rho(q_i, r_j)$ for all $q_i \in Q(W)$ and $r_j \in R$
Output: R^*
(1) **begin**
(2) $R^* \longleftarrow \emptyset$;
(3) **while** $\eta(R^*) < b$ **do**
(4) $r^* \longleftarrow$ null;
(5) $score^* \longleftarrow 0$;
(6) **for** $r \in R$ **do**
(7) $score \longleftarrow \dfrac{\rho(W, R^*) - \rho(W, R^* \cup \{r\})}{\eta(r)}$;
(8) **if** $score > score^*$ **then**
(9) $score^* \longleftarrow score$;
(10) $r^* \longleftarrow r$;
(11) **if** r^* *is null* **then**
(12) **break;**
(13) **else**
(14) $R^* \longleftarrow R^* \cup \{r^*\}$;
(15) $R \longleftarrow R \setminus \{r^*\}$;
(16) **return** R^*.

Algorithm 1: A greedy replica selection algorithm.

4.3.2. Reducing the Number of Candidate Replicas. Considering two replicas $r_1, r_2 \in R$ satisfying

$$\eta(r_1) \leq \eta(r_2), \tag{21}$$

and

$$\rho(q_i, r_1) \leq \rho(q_i, r_2), \quad \forall q_i \in Q(W), \tag{22}$$

we refer to this case as replica r_1 *dominates* replica r_2. Obviously, if we use $R\backslash\{r_2\}$ instead of R as the input candidate replicas, it will not change the optimal workload cost $\rho(W, R^*)$. Therefore, we can safely prune r_2 from R. In general, it is more common that a replica is dominated by a set of replicas. Concretely, given a replica $r \in R$ and a replica set $R^D \subseteq R$, we say that replica set R^D *dominates* replica r if

(1) $r \notin R^D$;

(2) $\eta(R) \leq \eta(r)$;

(3) $\rho(q_i, R^D) \leq \rho(q_i, r), \forall q_i \in Q(W)$.

Ideally, we want to find a minimum *dominant replica set* $R^D \subseteq R$ such that R^D dominates any replica $r \in R\backslash R^D$. However, as we can prove that the replica selection problem itself is NP-hard, we do not pursue a minimum R^D in practice. Instead, we use a rough yet effective heuristic algorithm to find a suboptimal dominant replica set.

4.4. Approximation Solution. In this section, we propose several approximate algorithms to select a near-optimal set of replicas based on the reduced problem size as illustrated in Section 4.3. Approximation 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 reselected frequently.

4.4.1. Greedy Strategy. First we give a fast greedy algorithm to solve the replica selection problem. This algorithm is adopted and extended from the minimum weighted set cover algorithm. As shown in Algorithm 1, we add one replica at a time to the replica set R^* such that in each step the added replica r maximizes,

$$\frac{\rho(W, R^*) - \rho(W, R^* \cup \{r\})}{\eta(r)}, \tag{23}$$

until the storage budget is exhausted or the overall workload cost $\rho(W, R^*)$ cannot be further decreased by adding any one of the remaining replicas. Before the storage size is full, each time we add one replica into the replica set, in worst case we need iterate $|R|$ times until the storage space is full. In each iteration, we

(1) score all $|R\backslash R^*|$ replica candidates that are not added to R^* yet;

(2) add the replica with highest score into R^*.

The scoring step computes the gain of each replica candidates that may be added to $R*$; thus in this step all $q \in Q(W)$ are compared with the costs on the current replica and the candidate replicas. Hence, this step takes $O(|R||Q(W)|)$ time, and it will result in an $O(\log n)$ approximation ratio, where n is size of the set of all queries $q \in Q(W)$. The running time of this greedy algorithm is $O(nm^2)$, where m is size of the set of candidate replicas. In Section 6, we will see that the approximation ratio of the greedy algorithm is quite desirable (lower than 1.3 in most cases) in practice.

4.4.2. LP Rounding Strategy. Although the greedy strategy is simple to implement and achieves good approximate result in practice, the best we can hope for the greedy strategy is a logarithmic approximate ratio ($\log n$). When the

Input: $W, R, b, \rho(q_i, r_j)$ for all $q_i \in Q(W)$ and $r_j \in R$, N_i for all $q_i \in Q(W)$
Output: R^*
(1) **begin**
(2) $\Gamma \longleftarrow \emptyset$;
(3) sort $q_i \in Q(W)$ by C_i in ascend order;
(4) **while** $\eta(R^*) < b$ *and* $Q(W) \neq \emptyset$ **do**
(5) choose q_i from $Q(W)$ with smallest C_i;
(6) $\gamma \longleftarrow 0$;
(7) **foreach** $q_{i'} \in \Gamma$ **do**
(8) **if** $N_i \cap N_{i'} \neq \emptyset$ **then**
(9) $\gamma \longleftarrow 1$;
(10) **break;**
(11) **if** $\gamma = 0$ **then**
(12) $\Gamma \longleftarrow \Gamma \cup \{q_i\}$;
(13) $Q(W) \longleftarrow Q(W) \setminus \{q_i\}$;
(14) $R^* \longleftarrow \emptyset$;
(15) **foreach** $q_{i'} \in \Gamma$ **do**
(16) $R^* \longleftarrow r^*$ with $\rho(q_{i'}, r^*) = \rho(q_{i'}, N_{i'})$;
(17) **return** R^*.

Algorithm 2: A LP rounding based algorithm.

quantity of queries goes large, the performance guarantee will drop accordingly. In this section we introduce a constant-factor approximate algorithm based on linear programming rounding [34]. The linear programming rounding strategy consists of three stages:

(1) Formulating the problem to integer linear programming

(2) Relaxing the integral constraints and finding the optimal solution for the relaxed linear programming

(3) Rounding the fractional solution of the linear programming and producing an integral solution.

In the replica selection problem, the LP rounding strategy is based on the MIP proposed in Section 4.2; thus we already finished stage 1. In stage 2, we further relax the MIP by allowing $x_j \leq 1$ and $y_{ij} \geq 0$. Then we can solve the LP in polynomial time resulting fractional x_j and y_{ij}. In stage 3, since general rounding techniques cannot be directly adopted on the replica selection problem, we present the following rounding strategy.

Suppose we have found an optimal solution for the LP in stage 2. For any query $q_i \in Q(W)$, we define the *neighborhoods* of q_i as

$$N_i = \left\{ r_j \in R^* \mid y_{ij} > 0 \right\}. \tag{24}$$

All the replicas r_j that serve q_i fractionally are the neighborhoods of q_i. Further we define *cluster* as a set of queries and replicas with the center $q_i \in Q(W)$. In the LP, we denote

$$C_i = \sum_{r_j \in R^*} c_{ij} \cdot y_{ij}, \tag{25}$$

and thus the total query cost is

$$\rho\left(W, R^*\right) = \sum_{(q_i, w_i) \in W} w_i \cdot C_i. \tag{26}$$

Now we sort queries q_i by C_i in ascending order and then iteratively assign each query and replica to clusters until all queries and replicas are assigned to one cluster. In each iteration, we pick the query q_i with the smallest C_i. If $N_i \cap N_{i'} = \emptyset$ for any existing cluster center $q_{i'} \in \Gamma$, we open a new cluster i and add q_i into the new cluster and denote q_i as the cluster center. If $N_i \cap N_{i'} \neq \emptyset$, we add q_i to cluster i'. Then we can round the fractional solution: for each cluster, we select the cheapest replica r_i for each cluster center q_i in N_i and assign queries in this cluster to replica r_i. The overall constant-factor approximation algorithm is shown in Algorithm 2. Theorem 10 provides the approximate ratio of the LP rounding based strategy.

Theorem 10. *The proposed LP rounding strategy is a 3-factor approximation algorithm.*

Proof. Suppose the optimal solution of the MIP is Θ_0 and the optimal solution of the relaxed LP is Θ_1, since Θ_0 is a feasible solution of Θ_1, we can prove $\Theta_0 \leq \Theta_1$ [35]. In the rounding solution, we select replicas that have the cost at most $4\Theta_1$.

Assuming that $q_{i'}$ is the center of cluster k, we have selected replica r_{j_k} for any query q_i in cluster k. For q_i, there are three types of query cost $\rho(q_i, N_i)$ on replica r_{j_k}:

(1) q_i is in cluster k and $c_{ij_k} \leq c_{i'j_k}$.

(2) q_i is in cluster k but $c_{i'j_k} \leq c_{ij_k}$. Since $N_i \cap N_{i'} \neq \emptyset$, queries q_i and $q_{i'}$ share some replica in common. By triangle inequality we have $c_{i'j} \leq c_{ij_k} + c_{ij} + c_{i'j_k} \leq C_i + 2C_{i'} \leq 3C_{i'}$. The last inequality is because we sort C_i in ascending order and pick the query with smallest C_i each time.

(3) q_i is not in cluster k; in this case, we set c_{ij_k} to ∞ and q_i will not be queried on any replica in this cluster.

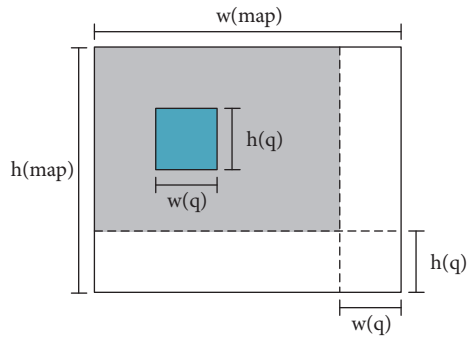


Figure 3: An example of the distribution of queries in this paper.

The total cost of the rounding solution is

$$\rho(W,R) = \sum_{i=1}^{n}\sum_{j=1}^{m} w_i \cdot c_{ij} \cdot y_{ij} \le 3 \cdot \sum_{i=1}^{n} w_i C_i = 3 \cdot \Theta_1, \quad (27)$$

which is at most triple cost of the LP. □

5. Query Cost Estimation

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

We estimate the cost of a query with respect to a replica via the expectation of the running time towards the replica. Since each partition p of a replica r consists of a spatial range $S(p)$ and a temporal range $T(p)$, we will show our estimations of the query cost in both spatial and temporal aspects.

As defined in Definition 6, in this paper, we consider q as a cuboid and we use $\mu(q)$ to denote the spatial-temporal range of q, i.e., $\langle x, y, t, \delta_x, \delta_y, \delta_t \rangle$. To clearly show the proof, in this section, we use $S(q)$ to denote the spatial range of q, where $S(q) = \langle x, y, w, h \rangle$, $\langle x, y \rangle$ is the top-left point of the rectangle and $w(q)$ and $h(q)$ are the width and the height of the rectangle, respectively. Similarly, for each partition $p \in P(r)$, we use $w(p)$ and $h(p)$ to denote the width and the height of the partition.

To clearly address the expected partitions that a query should scan, we consider the queries are uniformly distributed in the space, as shown in Figure 3. In Figure 3, $w(D)$ and $h(D)$ are the width and height of the map, respectively. The query is shown as a blue rectangle, and the top-left point of the query is only allowed to be generated in the gray area, because if a query exceeds the spatial range of the map, such query can be considered as another query with a smaller spatial range. The probabilities of the top-left point being anywhere of the gray area are the same, i.e., uniformly distributed.

5.1. Expected Spatial Partitions. In this paper, given a workload W, the probability of a spatial partition being scanned is clearly the quotient of the number of queries overlapped with the partition, being divided by the total number of queries in W. Since the queries are uniformly distributed, the probability can be written as the quotient of the area within which the queries may overlap with the partition (the orange rectangle in Figure 4), being divided by the entire area that all the queries belong to (the gray area in Figure 3).

Assuming the distances between a partition p and the boundary of the map are $west(p)$, $east(p)$, $north(p)$, and $south(p)$, we define the expected spatial partitions as follows.

Theorem 11 (expected spatial partitions). *Given query q and replica r with partitions $p \in P(r)$, the expected number of spatial partitions $\varepsilon_s(q,r)$ that the query should scan is*

$$\varepsilon_s(q,r) = \sum_{p \in P(r)} \varepsilon_s(q,p), \quad (28)$$

where

$$\varepsilon_s(q,p) = \frac{(w(q) + w(p) - w(\alpha)) \cdot (h(q) + h(p) - h(\alpha))}{(w(D) - w(q)) \cdot (h(D) - h(q))}, \quad (29)$$

where α is the offset of the query, and

$$\begin{aligned} w(\alpha) &= \max(0, w(q) - west(p)) \\ &\quad + \max(0, w(q) - east(p)), \\ h(\alpha) &= \max(0, h(q) - north(p)) \\ &\quad + \max(0, h(q) - south(p)). \end{aligned} \quad (30)$$

Proof. The proof of the denominator of $\varepsilon_s(q,p)$ is trivial; thus we only consider the numerator, denoted by S; i.e., the area within which the queries may overlap with the partition as shown in Figure 4. In Figure 4, the query is colored in blue, and the partition is colored in purple. The orange rectangle shows the area within which the queries may overlap with the partition.

(1) The area of the partition is smaller than the query, as shown in Figure 4(a). From observation, we have $S = (w(q) + w(p)) \cdot (h(q) + h(p))$, and $w(a) = h(a) = 0$. Hence Theorem 11 holds.

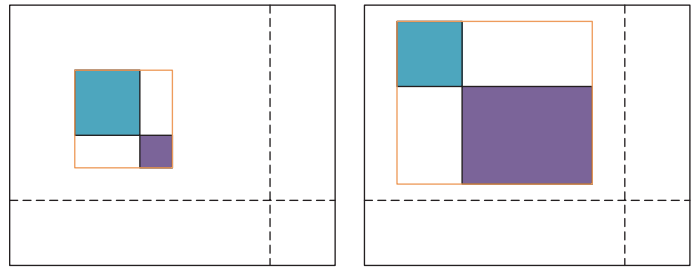
(2) The area of the partition is larger than the query, as shown in Figure 4(b). Similar to the previous situation, Theorem 11 holds.

(3) The partition is in the corner and exceeds the legal range of the query, as shown in Figure 4(c). From observation, we have $S = (w(q)+w(p)-w(a)) \cdot (h(q)+h(p)-h(a))$. Hence Theorem 11 holds.

(4) The partition is adjacent to the boundary, as shown in Figure 4(d). From observation, we have $S = (w(q)+w(p)) \cdot (h(q)+h(p)-h(q)) = (w(q)+w(p)) \cdot h(p)$, since $h(q)+h(p)-h(a) = h(q)+h(p)-0-(h(q)-0) = h(p)$; Theorem 11 holds.

(5) The partition is adjacent to more than two boundaries. This is not possible based on the spatial partition scheme, because the number of partitions ≥ 4.

In conclusion, Theorem 11 holds. □



(a) Partition is smaller than the query

(b) Partition is larger than the query

(c) Partition exceeds the legal range of the query

(d) Partition is adjacent to the boundary

FIGURE 4: Different situations when calculating $\varepsilon_s(q, p)$.

5.2. Expected Temporal Partitions. Similar to the expected spatial partitions, the probability of a temporal partition being scanned is the quotient of the temporal range within which the queries may overlap with the partition, being divided by the temporal range that all the queries belong to. Assuming the intervals between a partition p and the temporal range of all the records $T(D)$ are $top(p)$ and $bot(p)$, we define the expected temporal partitions as follows.

Theorem 12 (expected spatial partitions). *Given query q and replica r with partitions $p \in P(r)$, the expected number of temporal partitions $\varepsilon_t(q, r)$ that the query should scan is*

$$\varepsilon_t(q, r) = \sum_{p \in P(r)} \varepsilon_t(q, p), \tag{31}$$

where

$$\varepsilon_t(q, p) = \frac{T(q) + T(p) - T(\alpha)}{T(D) - T(q)}, \tag{32}$$

where α is the offset of the query, and

$$\begin{aligned} T(\alpha) = {} & \max(0, T(q) - top(p)) \\ & + \max(0, T(q) - bot(p)). \end{aligned} \tag{33}$$

The proof of Theorem 12 is similar to Theorem 11.

5.3. Expected Query Cost. As described in Section 3, to answer query q on replica r, a BLOT system scans (the physically stored objects of) all partitions $p \in P(r)$ that satisfies $\mu(p) \cap \mu(q) \neq \emptyset$ and then filters each record by $\mu(q)$. Based on the expected number of spatial and temporal partitions that a query should scan, we can combine them as the expectation of desired partitions given query q:

$$\varepsilon(q, r) = \sum_{p \in P(r)} \varepsilon_s(q, p) \cdot \varepsilon_t(q, p). \tag{34}$$

Now given the number of spatial and temporal partitions n_s and n_t of $P(r)$, respectively, we have

$$\rho(q, r) = \varepsilon(q, r) \cdot \left(\frac{\eta(r)}{n_s \cdot n_t \cdot \zeta(r)} + \xi(r) \right) \tag{35}$$

where $\zeta(r)$ and $\xi(r)$ are the scanning speed in terms of number of records scanned per unit time and the time before and after the actual scan process of the replica given its encoding scheme $E(r)$, respectively. For example, if each partition is stored continuously as a regular file on a local disk, then $\xi(r)$ is the seek time of locating the beginning of the file and $\zeta(r)$ is the transfer rate of the disk (assuming that CPU always waits for I/O). As another example, if each partition is stored as an object on Amazon S3 and queries are processed on Amazon EMR (Elastic MapReduce), then $\xi(r)$ is the time initializing the map task plus the time locating the S3 object before scanning the partition. The value of $\zeta(r)$ depends on the encoding scheme $E(r)$. In real situations, a high compression ratio generally leads to a slow scan speed.

In this paper, we assume that all candidate partitioning schemes will generate non-skewed data partitions. In other words, the number of records in each $D(p_i)$ is 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.

Table 2: Compression ratio of encoding schemes.

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

Putting (34) and (35) 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

$$O\left(|W| \cdot |R| \cdot \max_{r_j \in R} \left|P\left(r_j\right)\right|\right). \tag{36}$$

6. Evaluation

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

6.1. 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 pointing 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 and then the time to generate equal-sized (in terms of number of records) partitions. The space is partitioned according to a k-d tree [9] 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 spatial-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 2. Putting the above partitioning schemes and the encoding schemes together, the total number of candidate replicas is $25 \times 7 = 150$.

6.2. Measuring Scan Rate and Extra Time. Since ζ and ξ 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 sizes of partitions within a partition set are the same while they are different across partition sets. We then launch 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 $\rho(q, p)$ in (34). Accordingly, we use the corresponding partition size (in terms of number of records) as $\eta(r)$. We therefore have 5 measured points for (35). In the last step, we perform linear regression to fit the measured points and use the fitted parameters as $1/\zeta$ and ξ.

In Figure 5, the left two subfigures show all the measurement results and the right two subfigures show the fitted lines for three measurements in each of the execution environments. In addition, the measured values of ζ and ξ are listed in Table 3. We can see that $\rho(q, p)$ is well-fitted by (35) especially when the size of partition is relatively large, which demonstrates the effectiveness of our cost model.

6.3. 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 6 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 7 compares the relative query performance for all the queries when the replica set is selected by different approaches. The storage budget is set to be 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 8 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 7. We can see that when the MIP solution is close to the ideal case regardless of the storage budget, which is faster than

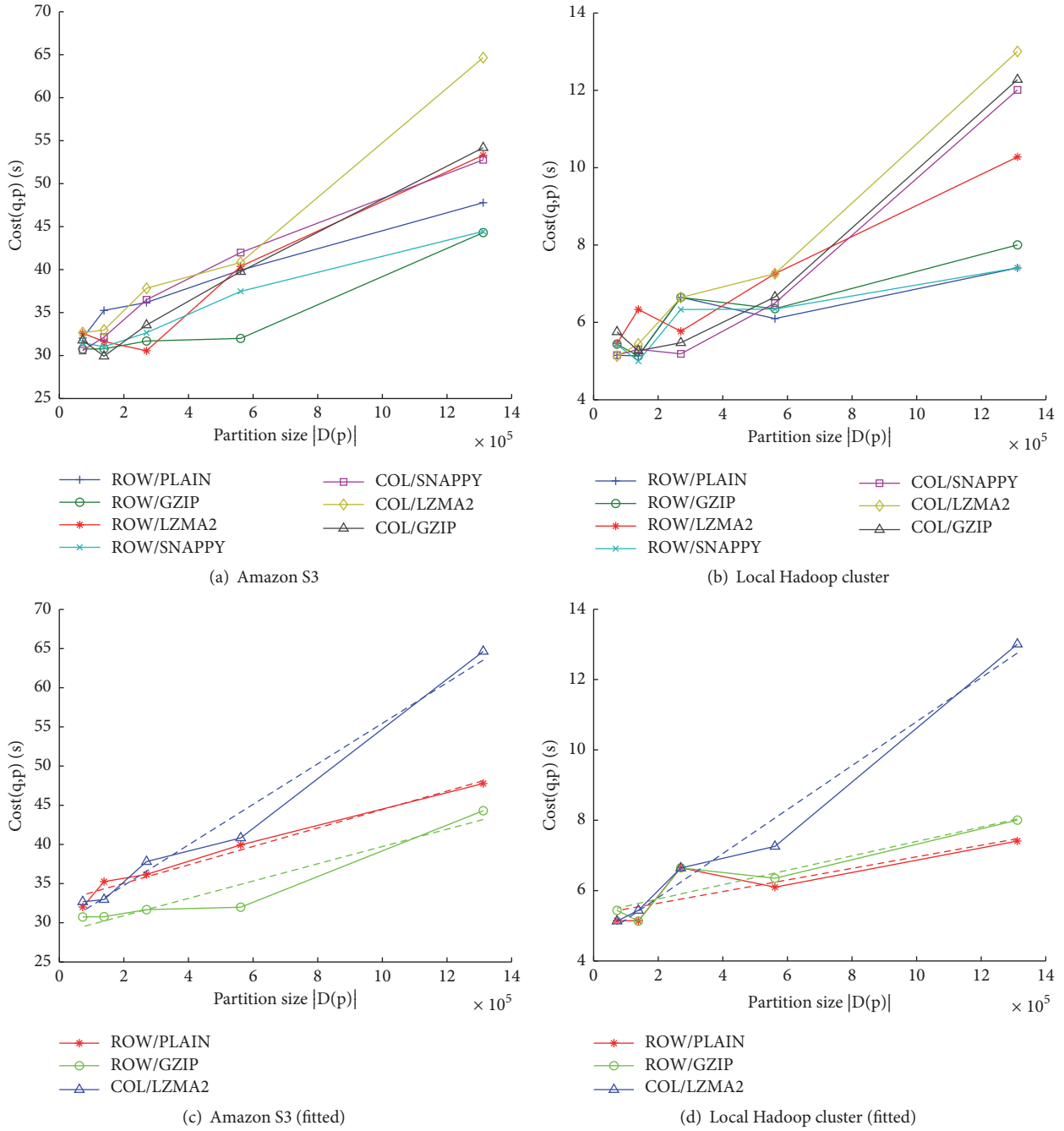


Figure 5: Measurement results of $\rho(q, p)$.

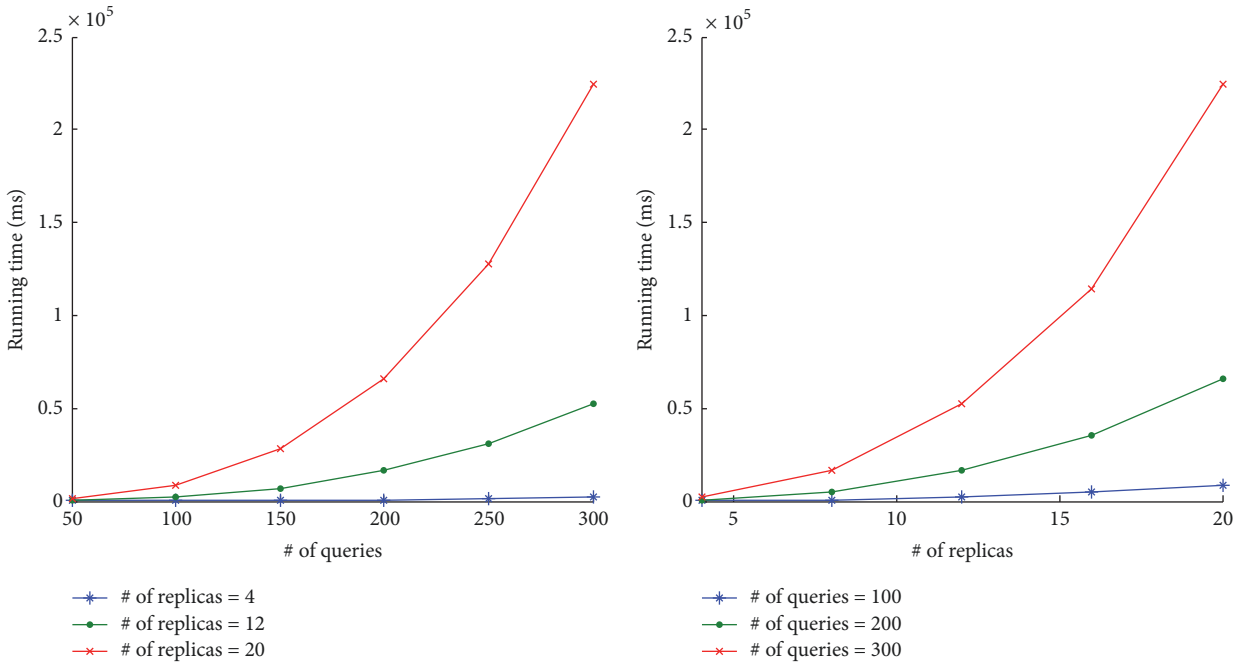
the single replica case by up to 80%, the approximation ratio of the greedy algorithm decreases dramatically as the storage budget increases. When the relative storage budget is greater than 1, the approximation ratio of the greedy algorithm is less than 1.2.

7. Conclusion

In this paper, we explore the use of diverse replicas in the context of storage systems for big location-based mobile data. Specifically, we propose BLOT, a system abstraction that describes an important class of location-based mobile 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

Table 3: Measured ζ and ξ.

		Amazon S3 and EMR	
		$1/\zeta$ (ms)	ξ (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	
		$1/\zeta$ (ms)	ξ (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 6: Comparison of the computation speed of MIP.

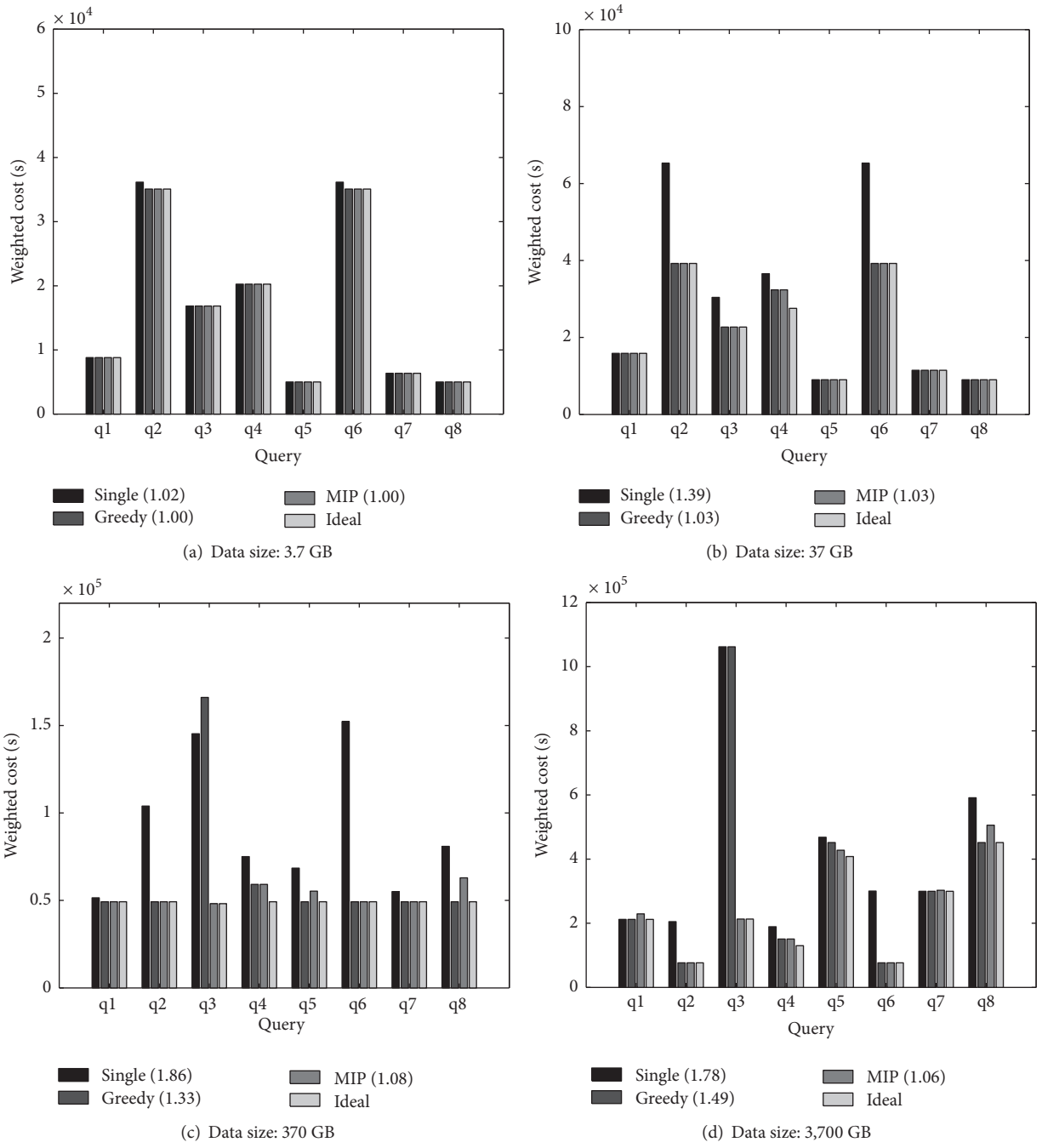


Figure 7: Relative overall query performance in Amazon S3 and EMR.

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.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Disclosure

An earlier version of this work appeared in the Proceedings of IEEE ICDCS [36], June 2014.

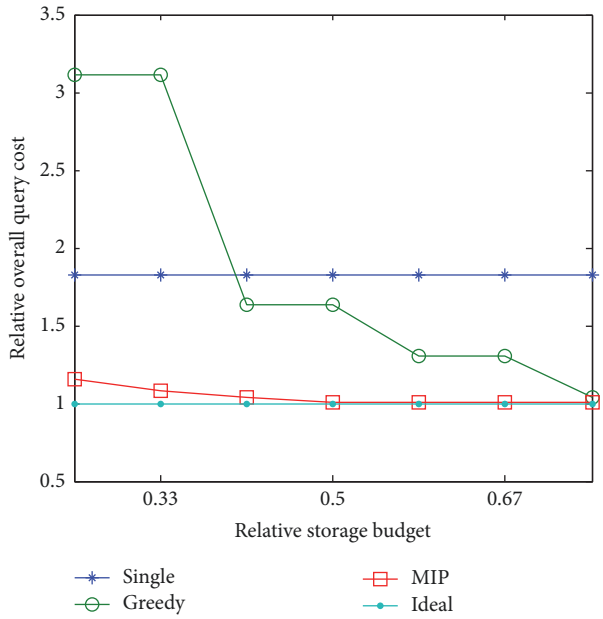


FIGURE 8: Relative overall query performance of different storage budgets.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work is supported in part by the National Key Research and Development Program of China under Grant no. 2017YFB0202201 and National Natural Science Foundation of China under Grant no. U1711261.

References

[1] A. Doshi and M. M. Trivedi, "Tactical driver behavior prediction and intent inference: A review," in *Proceedings of the 14th International IEEE Conference on Intelligent Transportation Systems, ITSC 2011*, pp. 1892–1897, Washington, DC, USA, 2011.

[2] Y. Ding, J. Zheng, H. Tan, W. Luo, and L. M. Ni, "Inferring road type in crowdsourced map services," *DASFAA*, pp. 392–406, 2014.

[3] Y. Ding, S. Liu, J. Pu, and L. M. Ni, "Hunts: A trajectory recommendation system for effective and efficient hunting of taxi passengers," *MDM*, pp. 107–116, 2013.

[4] W. Luo, H. Tan, L. Chen, and L. M. Ni, "Finding time period-based most frequent path in big trajectory data," *SIGMOD*, pp. 713–724, 2013.

[5] C. S. Jensen, H. Lu, and B. Yang, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull*, vol. 33, no. 2, pp. 12–17, 2010.

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

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

[8] A. M. Hendawi and M. F. Mokbel, "Panda: a predictive spatio-temporal query processor," in *Proceedings of the SIGSPATIAL 2012 International Conference on Advances in Geographic Information Systems (formerly known as GIS), SIGSPATIAL'12, Redondo Beach*, vol. 2012, pp. 13–22, CA, USA, 2012.

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

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

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

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

[13] 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.

[14] A. Eldawy and M. F. Mokbel, "A demonstration of spatial-hadoop: An efficient mapreduce framework for spatial data," *PVLDB*, vol. 6, no. 2, 2013.

[15] H. Vo, A. Aji, and F. Wang, "SATO: a spatial data partitioning framework for scalable query processing," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Dallas/Fort Worth*, vol. 2014, pp. 545–548, TX, USA, 2014.

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

[17] N. Bruno and S. Chaudhuri, "Constrained physical design tuning," *PVLDB*, vol. 1, no. 1, pp. 4–15, 2008.

[18] 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.

[19] 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.

[20] A. Jindal, J.-A. Quiané-Ruiz, and J. Dittrich, "Trojan data Layouts: Right shoes for a running elephant," *SOCC*, p. 21, 2011.

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

[22] J. Chen, K. Hu, Q. Wang, Y. Sun, Z. Shi, and S. He, "Narrowband Internet of Things: Implementations and Applications," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 2309–2314, 2017.

[23] L. Wang, X. Qi, J. Xiao, K. Wu, M. Hamdi, and Q. Zhang, "Exploring Smart Pilot for Wireless Rate Adaptation," *IEEE Transactions on Wireless Communications*, vol. 15, no. 7, pp. 4571–4582, 2016.

[24] R. Ramamurthy, D. J. DeWitt, and Q. Su, "A case for fractured mirrors," *VLDB*, pp. 430–441, 2002.

[25] D. Sacca and G. Wiederhold, "Database Partitioning in a Cluster of Processors," *ACM Transactions on Database Systems*, vol. 10, no. 1, pp. 29–56, 1985.

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

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

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

[29] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical Partitioning Algorithms for Database Design," *ACM Transactions on Database Systems*, vol. 9, no. 4, pp. 680–710, 1984.

[30] Y. Tong, L. Chen, Z. Zhou, H. V. Jagadish, L. Shou, and W. Lv, "Slade: a smart large-scale task decomposer in crowdsourcing," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

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

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

[33] R. J. Dakin, "A tree-search algorithm for mixed integer programming problems," *The Computer Journal*, vol. 8, no. 3, pp. 250–255, 1965.

[34] M. Charikar, S. Guha, É. Tardos, and D. B. Shmoys, "A constant-factor approximation algorithm for the k-median problem," *Journal of Computer and System Sciences*, vol. 65, no. 1, pp. 129–149, 2002.

[35] J.-H. Lin and J. S. Vitter, "Approximation algorithms for geometric median problems," *Information Processing Letters*, vol. 44, no. 5, pp. 245–249, 1992.

[36] Y. Ding, H. Tan, W. Luo, and L. M. Ni, "Exploring the use of diverse replicas for big location tracking data," in *Proceedings of the IEEE 34th International Conference on Distributed Computing Systems, ICDCS 2014*, pp. 83–92, 2014.