Using Spatial Semijoins Over Multiple Sites in Distributed Spatial Query Processing

Utilisation de semi-jointures spatiales sur plusieurs sites dans le traitement des requêtes spatiales distribuées

Wendy Osborn, *Member, IEEE*, and Saad Zaamout

***Abstract*— We present a strategy for geographically distributed spatial query optimization that involves multiple sites. Previous work in the area of geographically distributed spatial query processing and optimization focused only on strategies for performing spatial joins and spatial semijoins, and geographically distributed spatial queries that only involve two sites. We propose a strategy for optimizing a geographically distributed spatial query, which uses spatial semijoins and can involve any number of sites in a geographically distributed spatial database. It identifies and initiates spatial semijoins from the smaller spatial relations in order to reduce the larger spatial relations. By doing so, the data transmission and I/O costs are significantly reduced. We compare the performance of our strategy against the naïve approach of shipping entire spatial relations to the query site. We find that our optimized strategy minimizes the data transmission cost and I/O cost in all cases, and significantly in specific situations. In addition, the CPU cost is not significantly affected by our strategy.**

***Résumé*— Nous présentons une stratégie pour l'optimisation des requêtes spatiales réparties géographiquement qui impliquent plusieurs sites. Les travaux antérieurs dans le domaine de la répartition géographique des traitements de requête spatiale et l'optimisation axée uniquement sur les stratégies pour effectuer des jointures et semi-jointures spatiales, et les requêtes spatiales réparties géographiquement qui impliquent uniquement deux sites. Nous proposons une stratégie pour optimiser une requête spatiale répartie géographiquement, qui utilise semi-jointures spatiales et peut impliquer un certain nombre de sites dans une base de données spatiales distribuées géographiquement. Cette stratégie identifie et initie les semi-jointures spatiales des relations spatiales les plus petites afin de réduire les relations spatiales les plus grandes. Ce faisant, la transmission de données et d'E/S coûts sont sensiblement réduits. Nous comparons les performances de notre stratégie de lutte contre l'approche naïve de l'expédition entière des relations spatiales sur le site de la requête. Nous constatons que notre stratégie optimisée minimise le coût de transmission de données et le coût d'E/S dans tous les cas, et de façon significative dans des situations particulières. En outre, le coût en terme de processeur n'a pas été significativement affecté par notre stratégie.**

***Index Terms*— Distributed databases, optimization, spatial data.**

I. Introduction

NOWADAYS, different users and organizations from around the world must work with spatial data that is geographically distributed. In several cases, the spatial data are geographically distributed over a very long distance. For example, let us consider emergency and disaster management. One example of a distributed disaster management application is for earthquake detection across many Canadian cities at risk [2]. Another example is a distributed emergency management system for Australia [3]. In both papers, it is stated that storing and managing spatial data across several local sites that are geographically distributed (as opposed to managing it centrally in one repository) will significantly improve the response times in emergency situations. This is because in both Canada and Australia, many cities are separated by a very long distance of up to thousands of miles. If one city is in an emergency situation, and is required to wait for the spatial data it needs to arrive from far away, a disastrous outcome may be the result. Having the spatial data stored locally is a better solution. Therefore, the geographic distribution of spatial data has become an important matter globally, and therefore, managing and querying it are very important.

A distributed spatial database system [5], [6] consists of several individual spatial database sites that are interconnected by a network. Each site manages its own collection of spatial

Manuscript received June 24, 2014; revised February 12, 2015 and June 6, 2015; accepted July 30, 2015. Date of current version April 20, 2016.

W. Osborn is with the Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, AB T1K 3M4, Canada (e-mail: wendy.osborn@uleth.ca).

S. Zaamout was with the Department of Mathematics and Computer Science, University of Lethbridge Lethbridge, AB T1K 3M4, Canada. He is now with SCA Interactive, Calgary, AB T2H 0R5, Canada (e-mail: s.zaamout@uleth.ca).

Associate Editor managing this paper's review: Vahid Garousi.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/CJECE.2015.2463753

data, but work collectively for processing intersite data requirements. One such type of a distributed spatial database system is a geographically distributed spatial database system, where the individual database sites are dispersed over a significant geographical distance.

An important requirement of a geographically distributed spatial database system is the ability to efficiently process a query that requires spatial data from the multiple geographically distributed sites. This is a requirement for planning purposes. For example, referring to the emergency management system for Australia above [3], some important queries can include: 1) how environmental concerns are related to buildings and infrastructure considerations or 2) do natural hazards have the appropriate emergency services for handling them in certain regions. Each of these spatial queries can only be answered by performing a geographically distributed spatial query because the spatial data for emergency services, infrastructure, natural hazards, and the environment are each managed in individual spatial database sites that are dispersed geographically.

Initially, the research in distributed relational databases focused on generating query execution plans that minimized the cost of data transmission over the network [6]–[10]. Later strategies also considered minimizing other cost factors, such as I/O and CPU [8], [11]–[14]. Relational data consist of alphanumeric values that can be matched using equality or inequality when joined. However, spatial data are more complex than alphanumeric data. Different types of spatial data include objects [or their representations using minimum bounding rectangles (MBRs)], points, lines, and any combination of these types [5]. In addition, these spatial data lead to different joining predicates such as overlap, containment, and adjacency, which in turn increases the complexity of joining spatial relations. Therefore, CPU and I/O costs must always be considered when processing a geographically distributed spatial query [5].

Most existing strategies that process a geographically distributed spatial query are only proposed for two sites [15]–[19]. One exception to this [20] does not currently support spatial joins. Therefore, this paper helps to address this shortcoming by proposing a strategy for processing and optimizing a geographically distributed spatial query that utilizes spatial data from more than two sites. Our strategy, called optimized, focuses on minimizing not only the data transmission cost of a query, but also the I/O and CPU costs, by applying spatial semijoins in a cost-effective manner.

An empirical evaluation of the optimized strategy versus the naïve strategy (i.e., initial feasible solution (IFS) based on [21]) shows significant reductions in the data transmission cost and the I/O cost over all queries when optimized strategy is utilized. We find a 50% reduction in most cases, in both the amount of data being transmitted and the amount of I/O being incurred. In particular, when the query involves both smaller and larger relations, the reductions in both the data transmission and I/O costs are very significant, at $\sim$90% or greater. With respect to CPU, the optimized strategy achieves a slightly higher number of spatial predicate comparisons than the naïve strategy. However, for all cases it is $<$10% and for many it is $<$5%. This is a small price to pay for the significant achievements in the data transmission and I/O costs.

This paper proceeds as follows. Section II presents some background that is necessary. Section III presents related work in the area of geographically distributed spatial query processing. Section IV presents our algorithm, called the optimal algorithm, for processing a geographically distributed spatial query that can handle multiple (and more than two) sites. Section V presents an example that illustrates our new algorithm. Section VI presents our experimental methodology and cost estimation formulas. Section VII presents the results and discussion of our experiments. Finally, Section VIII concludes this paper and presents future research directions.

II. Background

This background section presents some definitions that are required for our proposed strategy and for the section on related work (Section III). These include definitions for spatial join, spatial semijoin, Bloom filter, naïve approach, and R-tree.

A. Distributed Spatial Query Operators

A spatial join [5], [22]–[24] takes two spatial relations Y and Z, each with a spatial attribute, and relates pairs of tuples between Y and Z using a spatial predicate that is applied to the spatial attribute values. Examples of spatial predicates include the overlap, containment, and adjacency of two spatial attributes. This paper assumes that the overlap predicate is used, but will work for any other spatial predicate.

A spatial semijoin [17] is performed by projecting the spatial attribute from one relation, transmitting it to the site that contains the other spatial relation, and performing a spatial join between the spatial projection and relation. Then, the qualifying tuples from the second site are shipped back to the first site and joined with the spatial relation on that site. We used a modified version of this semijoin strategy, which will be presented in Section IV.

A Bloom filter [25] is a hashed bit array that provides a compact but imprecise representation of the values of a joining attribute. A 1 b represents the possible existence of a joining attribute value, while a 0 b represents the absence of the value.

B. Naïve Approach

Our new optimized strategy for geographically distributed spatial query processing will be compared for performance against a naïve approach, which is also known as an IFS. Our naïve approach is based on the IFS utilized in [21] and used with spatial data instead of standard relational data. In the naïve strategy, all spatial relations are transmitted in their original, unreduced form, to the query site. In addition to the potentially large size of the relations, many tuples that will not participate in the final result are being sent unnecessarily to the query site.

C. R-Tree

The R-tree [4], [26] is a spatial access method (i.e., spatial index) that indexes a set of spatial objects or MBRs by their location in multidimensional space. The leaf level contains the spatial objects or MBRs, while the nonleaf levels contain

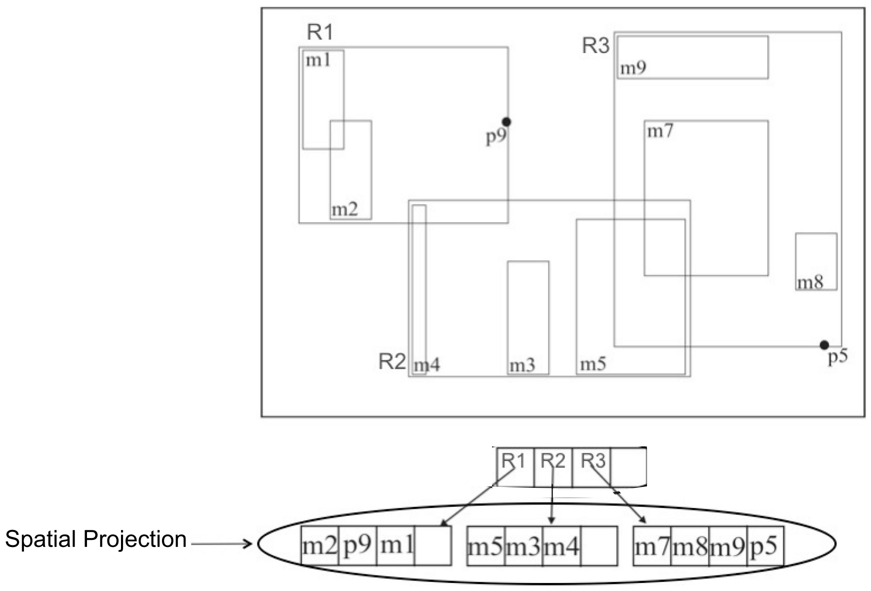

Fig. 1. Spatial projection from an R-tree (data from [4]).

MBRs that encompass other MBRs in the lower levels. In this paper, we utilize the R-tree for obtaining the spatial attribute projection required for a semijoin. The leaf level nodes can be scanned and the MBRs can be extracted. This allows the algorithm to avoid having to scan entire spatial relations in order to only obtain the spatial objects and corresponding identifiers. Fig. 1 shows this strategy.

Many variants of the R-tree have been proposed (see [4] for a comprehensive survey). Some recent adaptations include indexing in peer-to-peer systems [27] and in temporal data [28].

III. Related Work

In this section, we focus on previously proposed approaches and operators used to process geographically distributed spatial queries. Most research in geographically distributed spatial query processing focuses on spatial join algorithms, spatial semijoins algorithms, and the use of Bloom filters for processing geographically distributed spatial queries. With the exception of [20], all the proposed strategies for processing geographically distributed spatial queries are proposed for two sites only. We summarize these works below.

A. Spatial Join

A significant majority of spatial join algorithms are designed for a centralized system (see [23], [24]) or for a parallel system that lacks geographic distribution (see [29], [30]). Jacox and Samet [22] present a comprehensive survey on these algorithms for further information. This section focuses on spatial join strategies for geographically distributed spatial queries.

Kang *et al.* [15] proposed a parallel spatial join strategy that is adapted to a geographically distributed spatial database. Their strategy has two phases: 1) data redistribution and 2) filter and refinement. In the data redistribution phase, on each site, the space that contains objects is partitioned into regions (i.e., buckets). A subset of regions is transmitted between sites so that each site has the same corresponding regions from both spatial data sets. This subset is chosen by estimating which subset will result in the lowest overall response time (although it is unclear if I/O costs are considered). Then, the filter and refinement phase is carried out on both sites by performing a spatial join. An experimental evaluation shows that the parallel spatial join technique has a significantly faster response time—up to a 33% improvement over a semijoin-based strategy. However, it is unclear if this strategy can be extended efficiently to multiple sites.

Kalnis *et al.* [31] proposed a spatial join strategy between two spatial data sets R and S on noncooperating sites. For their strategy, the query site is any mobile device. Given the limited memory on a mobile device, the goal of their work is to reduce the cost of data transmission by pruning objects that will not be a part of the final result before they are sent to the mobile device for the final join. This is accomplished through the recursive partitioning of the overall space on both sites. If one subregion on one site contains no objects, then the corresponding subregion on the other site will be pruned as well. Related subregions between relations R and S that qualify for transmission to the query site are transmitted and can be joined with any spatial join technique, such as a nested loop join. An experimental evaluation shows that this approach achieves a lower data transmission cost in many cases, with the exception being in a comparison against the spatial semijoin. One limitation of their work is the consideration of spatial data sets from two sites only, although it is mentioned by the authors that joins of three or more sites will be considered in the future.

B. Spatial Semijoin

Tan *et al.* [16] and Abel *et al.* [17] proposed a spatial semijoin operator that combines the conventional semijoin operator with the filter stage of spatial query processing in order to reduce the data transmission, I/O cost, and CPU cost.

Their work explores two adaptations of the spatial semijoin. In the first, a projection of a set of MBRs from one spatial relation is transmitted to another site and applied its spatial relation using a spatial join. In the second, the projection is a single-dimensional mapping that represents the objects in each spatial relation. A performance evaluation between these approaches shows that: 1) for object sets with very large spatial descriptions, both strategies have similar performance; 2) for object sets with smaller spatial descriptions, a spatial semijoin that uses single-dimensional mapping works best; 3) using the R-tree for retrieving the MBRs incurs significant CPU costs; and 4) single-dimensional mapping causes more false drops than the MBRs.

Karam and Petry [19] proposed a spatial semijoin, which differs from [16] and [17] in that the MBRs from different levels of the R-tree are chosen for the spatial semijoin, instead of requiring that all come from the same level. A performance evaluation shows that their spatial semijoin outperforms the naïve spatial join (i.e., the whole relation is shipped to the other site for joining) when applied to real world spatial data, but not when applied to randomly generated rectangle sets. Limitation of their work is: 1) no comparison versus other strategies and 2) no consideration of CPU time.

C. Bloom Filters

Karam [18] proposed a 2-D bit-matrix approach for performing a spatial semijoin of two relations that attempts to minimize the data transmission, I/O cost, and CPU cost. A 2-D space is partitioned into equal-sized regions, with each region mapping to a bit in a 2-D array. If a region contains any objects, the corresponding bit is set to 1. This bit-matrix is transmitted to the site containing the other spatial relation, and is applied by testing each region that contains objects for the existence of a 1 bit in the bit-matrix. Any qualifying objects are sent to the first site. A performance evaluation shows that this approach shows the best improvement when applied to real world spatial data. Limitation of this paper is: 1) an evaluation against the spatial join, and not versus a spatial semijoin, which in functionality is a closer match to the bit-matrix approach and 2) no compression of the bit-matrix—a bit-matrix that contains many zeros is still transmitted in its entirety.

Hua *et al.* [20] proposed the BR-tree, which is an R-tree that is augmented with Bloom filters to support an exact-match spatial query. Each node entry contains: 1) an MBR that approximates an object or a subset of objects and 2) a Bloom filter that also represents one or more objects. In a leaf node, a Bloom filter is created by taking each object and producing k bits in the filter using different hash functions. In a nonleaf node entry, a Bloom filter is created by intersecting the Bloom filters in its child node. Although the BR-tree supports exact-match spatial queries using Bloom filters, it still requires the MBRs for region and point queries. A general strategy for processing distributed region, point, and exact-match queries is proposed. The algorithm duplicates the root of every BR-tree across every site in the geographically distributed spatial database. Any objects that pass the test against a root node is shipped to the site containing the original BR-tree. This strategy works for any number of sites. A significant limitation is a lack of support for spatial joins.

IV. Optimized Strategy

In this section, we present our optimized strategy for processing a geographically distributed spatial query. The focus is to reduce the data transmission, I/O cost, and CPU cost by utilizing the semijoin operator. We first present our preliminaries, followed by the optimized algorithm itself.

A. Preliminaries

We attempt to reduce all of the cost factors—primarily data transmission, but also the I/O and CPU costs—by utilizing the spatial semijoin operator. The original semijoin was proposed for the use in optimizing a geographically distributed relational query [6], [8], [32]. Its primary focus was to significantly reduce the cost of data transmissions across network lines by eliminating the data that were not needed for the final result before they were shipped. The spatial semijoin has the same primary objective in a geographically distributed spatial database system.

We utilize a greedy approach in our strategy. Spatial semijoins are applied in the following manner. We transmit the smaller spatial attributes to other sites and apply them to the larger spatial relations in order to eliminate a significant amount of data that will not participate in the final result. The transmission of the spatial projection from the smaller spatial relations to the larger ones, instead of the other way around, can significantly reduce the amount of spatial data that participates in the query, because it is more likely that the smaller spatial relation will have more participating tuples than the larger spatial relation. Reducing the larger spatial relation, therefore, is more advantageous. This not only reduces the amount of spatial data that must be transmitted, but also reduces the amount of spatial data that must be read and written, thus, reducing I/O. Although this approach has been applied for different types of spatial joins and strategies for processing geographically distributed relational queries [6], [8], it has not been applied in processing a geographically distributed spatial query that involves multiple (i.e., more than two) sites. Given that many sites (and spatial relations) are now involved, this affects the choice of which smaller spatial attribute projections to send to which larger spatial relations.

We chose to apply this technique for reducing the cost factors, and not to use estimations of selectivity for the following reasons. First, since our spatial attributes are representing random sets of objects, we felt it was not realistic to expect duplicate objects (or at least duplicate MBRs) in a spatial attribute that would be removed during a projection operation. Therefore, selectivity methods that are based on a low ratio of distinct values to the total number of values in an attribute would not be applicable. Second, we also consider this to be a worst case scenario, since an entire spatial attribute must ultimately be shipped to another site in order to perform a spatial semijoin.

We utilize a modified version of the approximation-based spatial semijoin that is proposed in [16]. In our implementation of the spatial semijoin, we use the following approach. We have two sites Y and Z that each contains a spatial relation. First, we obtain the spatial attribute projection from relation Y (or from an R-tree index if available). Then, the spatial projection is transferred to the site containing relation Z and joined on its spatial attribute. Our spatial semijoin differs after this point. Instead of sending the qualifying tuples from Z back to Y for the final join, we send back to Y the identifiers that correspond to the objects from the spatial attribute projection that participated in the spatial semijoin. These identifiers are used to select tuples from relation Y to ship to the final query site. In addition, the tuples on site Z that qualified in the spatial semijoin are also shipped to the query site. Using this spatial semijoin strategy allows us to incorporate more than two sites when processing a geographically distributed spatial query. In addition, sending all qualifying tuples from relation Z directly to the query site for a final join, as opposed to sending them back to Y first, performing a spatial join and then sending the partial result to the query site, will further reduce the cost of data transmission.

We make the following assumptions in this paper.

1) Every spatial relation has one spatial attribute. We use this as a starting point for processing geographically distributed spatial queries, because the focus of this paper is to propose a strategy that can be applied to more than two sites. Extensions to two or more spatial attributes in a spatial relation will be a future consideration.
2) The spatial attribute for every spatial relation is already indexed by an R-tree (or a similar index that places the MBRs for all spatial objects in its leaf level). Many database vendors provide the R-tree for indexing spatial data [33]. Therefore, it is safe to expect one to be available and used for indexing the spatial attribute of a spatial relation.
3) Every spatial object is represented using its MBR. R-tree-based structures store object approximations in the form of MBRs, with a reference to the actual object in the spatial database. Although utilizing MBRs instead of the actual objects themselves will result in some false positives, utilizing them will also result in faster spatial join times, since testing for the overlap of two MBRs is faster than testing for overlap of arbirarily shaped objects.
4) All spatial objects (or corresponding MBRs) in all spatial attribute are drawn from the same spatial domain (i.e., the same region of space). This is to ensure that the potential for overlap between MBRs from different spatial relations exists. However, our strategy will work across different spatial domains if they overlap, or some domains are contained in others. Furthermore, our strategy will determine via spatial semijoins if no overlap between spatial relations exists, and would avoid sending any spatial data to the query site.
5) Every site that participates in the geographically distributed spatial query has one spatial relation that is required for the query. If a site contains other spatial relations that are required for the query, it is assumed that all local processing has taken place and one spatial relation remains. Since, by the definition of a database, all data are related in some way [6] it is fair to say that the data can be processed locally with joins and selections to produce one overall spatial relation.
6) The cardinality of each spatial attribute is equal to the number of MBRs in the relation. That is, we assume that all the MBRs in a spatial attribute are distinct and represent a random collection of spatial objects. Unless several cases exist where two or more objects happen to have the same exact MBR representation, it is fair to say that all MBRs will be distinct.
7) Finally, the number of sites participating in the geographically distributed spatial query is a multiple of two. The reason for this is to be able to reduce the spatial relations on all participating sites using our strategy below. However, in the worst case, if there is an odd number of sites, then only one spatial relation will not be reduced, and therefore be shipped to the query site in its entirety. This could be the smallest spatial relation in order to keep this cost minimal.

B. Optimized Strategy

Given n sites that will be participating in processing a geographically distributed spatial query, where each site has one spatial relation, our strategy has four main steps:

1) ordering and grouping participating sites (i.e., spatial relations) by spatial attribute cardinality;
2) transmission of spatial attributes;
3) spatial semijoin execution;
4) transmission of qualifying tuples to query site for the final spatial join and processing.

Each step is described in detail next.

1) Ordering and Transmission of Spatial Attributes: First, all participating sites are ordered by increasing cardinality of the spatial attribute contained in its spatial relation. After the sites are ordered, the first $n/2$ sites of the ordered list are placed in a set P, while the remaining $n/2$ sites are placed in a set Q.

Then, the spatial attribute from the spatial relation on each site in P is projected and transmitted to a site in Q in the following manner.

a) The spatial attribute from the site with the smallest cardinality in P is sent to the site with the smallest cardinality in Q.
b) The spatial attribute from the site with the next smallest cardinality in P is sent to the site with the next smallest cardinality in Q.
c) This happens until the spatial attribute from the site with the largest cardinality in P is sent to the site with the largest cardinality in Q.

Because we are assuming that every participating spatial attribute contains unique MBRs, we will not have the benefit of reduced spatial attribute projection sizes. Therefore, sending the smaller spatial attributes to the sites of larger spatial relations will further reduce the amount of spatial data that is being transmitted, especially if the larger spatial relations have fewer participating tuples.

2) *Spatial Semijoins and Final Transmission:* Next, on each site in Q, a spatial semijoin is performed between the existing spatial relation and the spatial attribute sent from the corresponding site in P. The results of the semijoin are: 1) the set of tuples on the site in Q that qualified during the spatial semijoin and 2) a set of identifiers from the spatial attribute from P whose MBRs also qualified during the spatial semijoin. While the spatial semijoin is being executed, any tuples from the spatial relation from the site in Q that qualified are automatically shipped to the query site. At the same time, any identifiers from any qualifying MBRs from the spatial attribute from P are sent back to its corresponding site.

Finally, for all sites in P, the tuples corresponding to any identifiers that are shipped back from the spatial semijoin sites (i.e., sites in Q) are fetched and sent to the query site. At the query site, the final spatial join is performed.

V. Example

Suppose there exists a geographically distributed spatial database with six sites, as shown in Fig. 2(a). Each site contains a spatial relation. R1 contains 100 tuples, while R2, R3, R4, R5, and R6 contain 800, 200, 600, 1000, and 400 tuples, respectively. In addition, each spatial relation contains one spatial attribute. Our strategy for processing a geographically distributed spatial query that involves these sites proceeds as follows. First, the sites are ordered by increasing spatial attribute cardinality. Then, the list of sites is divided into the two sets. The set P will contain the sites R1, R3, and R6, while the set Q will contain the sites R4, R2, and R5.

Next, the spatial attributes from the sites in set P are projected and sent to the sites in Q in the following manner. First, the spatial attribute from the site R1 is first projected from the corresponding relation, and then sent to the site R4. Since we have an R-tree indexing the relation on its spatial attribute, a scan of the leaf nodes will provide us with the spatial projection. Similarly, the spatial attribute from the site R3 is projected and sent to the site R2, while the spatial attribute from the site R6 is projected and sent to the site R5. This is shown in Fig. 2(b).

Then, on each of the sites R4, R2, and R5, a spatial semijoin is performed between the local spatial relation and the spatial attribute projection that was shipped to it. During this process, the identifiers that correspond to the MBRs in the spatial attribute that qualified during the spatial semijoin are sent back to originating site. This is shown in Fig. 2(c). For example, on site R4, the identifiers corresponding to the qualifying MBRs are sent back to site R1, and are used to select the corresponding tuples. In addition, the qualifying tuples from the local relation on sites R4, R2, and R6 are sent to the query site. In Fig. 2(d), R4 will have 280 tuples sent to the query site while R2 and R5 will have 350 and 565 tuples, respectively, sent to the query site.

Finally, all qualifying tuples from sites R1, R3, and R6 are shipped to the query site. In Fig. 2(d), R1 will have

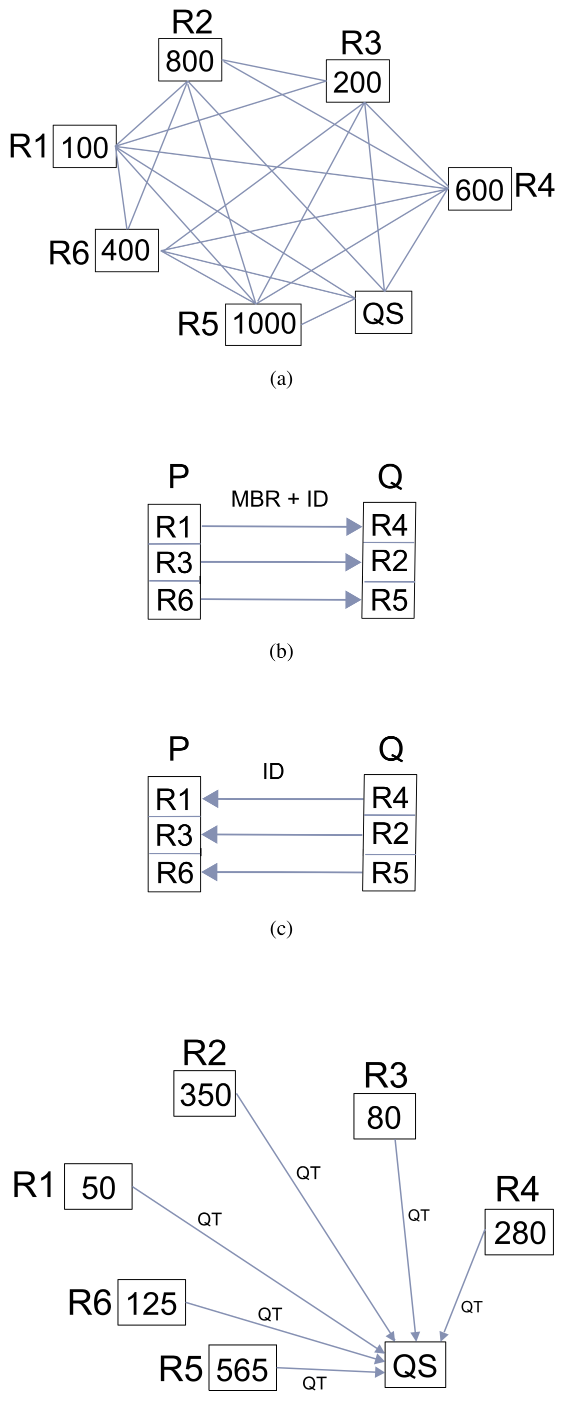

Fig. 2. Example geographically distributed spatial query. (a) Example geographically distributed spatial database. (b) Transmission of MBRs. (c) Transmission of IDs back to originating sites. (d) Transmission of qualifying tuples to query site.

50 qualifying tuples, and R3 and R6 will have 80 and 125 qualifying tuples, respectively, sent to the query site for the final spatial join.

VI. Evaluation Methodology

In this section, we present our empirical approach for evaluating our optimized strategy. We compared our strategy with the naïve approach that we summarized earlier in Section II. Here, we present our geographically distributed spatial database environment, spatial data sets, tests, and all formulas and criteria for estimating the data transmission, I/O and CPU costs.

A. Environment and Data

We simulate a six-site geographically distributed spatial database. In this simulation, we assume that all sites are homogeneous with respect to both hardware and software. In addition, the code used for the simulation is written in the C++ programming language. This includes the implementation for both the overall execution strategy and the spatial semijoin.

We utilize spatial relations with sizes ranging from 1000 to 100 000 tuples. This provides an opportunity to evaluate our strategy for both smaller and larger spatial object sets. Every spatial relation is indexed on its spatial attribute using an R-tree [26].

Each site contains the following spatial relations:

1) R1: 1000 tuples;
2) R2: 2000 tuples;
3) R3: 4000 and 40 000 tuples;
4) R4: 6000 and 60 000 tuples;
5) R5: 8000 and 80 000 tuples;
6) R6: 10 000 and 100 000 tuples.

It must be noted that although sites R3 through R6 have two spatial relations, only one is used at a time, depending on the query being evaluated.

Each spatial relation has one spatial attribute, which consists of four values (lx, ly, hx, hy) that represent the extents of an MBR. In addition, each spatial relation has the following nonspatial attributes: 1) identifier; 2) region name; 3) population; and 4) a line slope indicator. No fault tolerance mechanisms, in particular using data replication, are utilized in our simulated geographically distributed spatial database. We chose to address data replication in the future work because our focus is to determine how well our algorithm performs over multiple sites. However, future considerations of fault tolerance will make our algorithms more robust.

B. Evaluation Tests and Criteria

For all test queries, our performance criteria include data transmission, I/O cost and CPU cost.

For our evaluation, we perform the following sets of tests. For the first test set, we evaluate the optimized strategy using queries that require spatial relations from two sites of our geographically distributed spatial database. We executed 16 queries. The first eight queries process spatial relations that range between 1000 and 10 000 tuples. The remaining eight queries process spatial relations that range between 1000 and 100 000 tuples. The different combinations of spatial relations can be found by consulting Tables I–III.

For the second test set, we evaluated six queries that require data from four of the six sites, using the following spatial relations:

1) 1000, 2000, 4000, and 6000 tuples;
2) 1000, 2000, 8000, and 10 000 tuples;
3) 4000, 6000, 8000, and 10 000 tuples;
4) 1000, 2000, 40 000, and 60 000 tuples;
5) 1000, 2000, 80 000, and 100 000 tuples;
6) 4000, 6000, 80 000, and 100 000 tuples.

Finally, we performed two queries that require data from all six sites, using the following spatial relations:

1) 1000, 2000, 4000, 6000, 8000, and 10 000 tuples;
2) 1000, 2000, 4000, 60 000, 80 000, and 100 000 tuples.

TABLE I
TWO SITES—TRANSMISSION COST

Site 1	Site 2	Optimized	Naïve	%Imp
1000	4000	79024	340000	77
1000	6000	82776	476000	82
1000	8000	79426	612000	87
1000	10000	86796	748000	88
2000	4000	161532	408000	60
2000	6000	172252	544000	68
2000	8000	161398	680000	76
2000	10000	180292	816000	78
1000	40000	67500	2788000	98
1000	60000	83982	4148000	98
1000	80000	81570	5508000	99
1000	100000	83910	6868000	99
2000	40000	129104	2856000	95
2000	60000	156306	4216000	96
2000	80000	149740	5576000	97
2000	100000	153090	6936000	98

TABLE II
TWO SITES—I/O COST

Site 1	Site 2	Optimized	Naïve	%Imp
1000	4000	8438.65	27310.17	69
1000	6000	9815.51	385595.88	75
1000	8000	10541.01	49881.60	79
1000	10000	12351.37	61167.32	80
2000	4000	17563.68	37332.64	53
2000	6000	21362.79	51264.24	58
2000	8000	21750.13	65195.83	67
2000	10000	27372.77	79127.42	65
1000	40000	8084.81	230453.09	96
1000	60000	12142.50	343310.26	96
1000	80000	13597.33	456167.44	97
1000	100000	14972.46	569024.62	97
2000	40000	18300.91	288101.30	93
2000	60000	28092.74	427417.22	93
2000	80000	30579.37	566733.13	95
2000	100000	34912.08	706049.05	95

TABLE III
TWO SITES—CPU COST

Site 1	Site 2	Optimized	Naïve	%Imp
1000	4000	4190512	4000000	-5
1000	6000	6271180	6000000	-5
1000	8000	8332898	8000000	-4
1000	10000	10451130	10000000	-5
2000	4000	8693114	8000000	-9
2000	6000	13103204	12000000	-9
2000	8000	17242751	16000000	-8
2000	10000	21928238	20000000	-9
1000	40000	40164500	40000000	-0.4
1000	60000	60362929	60000000	-0.6
1000	80000	80448010	80000000	-0.5
1000	100000	100521220	100000000	-0.5
2000	40000	80875064	80000000	-5
2000	60000	121946686	120000000	-1.6
2000	80000	162175870	160000000	-1.3
2000	100000	202682875	200000000	-1.3

C. Data Transmission Cost Calculation

For our evaluation, we estimated the cost of data transmission as the total number of bytes that are transmitted. For estimating the cost for transmitting a data item X, the following formula has been proposed [11]:

$$T\text{cost}(\text{X}) = T_{\text{INIT}} + T_{\text{BYTE}} * \text{sizeof}(\text{X}) \quad (1)$$

where T_{INIT} is the time to initiate the transmission and T_{BYTE} is the time required to transmit one byte of data. However, given the homogeneous system we are simulating, we assume that both T_{INIT} and T_{BYTE} are constant, and therefore, are not incorporated into our calculations below.

In addition, we assume an integer size of 2 B, a double-precision floating point size of 8 B, a long integer size of 8 B, and a character size of 1 B. We first present the various costs of data transmission and finish with overall costs for both the optimized strategy and the naïve approach.

The various costs of data transmission are calculated in the following manner. There are several calculations required. First, the cost for transmitting an identifier between two sites is

$$T\text{cost}(\text{ID}) = \text{sizeof}(\text{int}). \quad (2)$$

Next, the cost for transmitting an MBR is equal to the number of bytes used to represent an MBR

$$T\text{cost}(\text{MBR}) = 4 * \text{sizeof}(\text{double}) + T\text{cost}(\text{ID}) \quad (3)$$

which encompasses the coordinate values (lx, ly, hx, hy) of the MBR itself, and the tuple identifier. Similarly, the cost for transmitting a tuple is

$$\begin{aligned} T\text{cost}(\text{tuple}) = &\ T\text{cost}(\text{MBR}) + 20 * \text{sizeof}(\text{char}) \\ &+ \text{sizeof}(\text{longint}) + \text{sizeof}(\text{int}) \end{aligned} \quad (4)$$

which encompasses the region name (up to 20 characters), population, and line slope indicator.

Finally, given spatial attribute X from spatial relation Y (i.e., site Y from the set P above) that is shipped to spatial relation Z (i.e., site Z from set Q above), the total data transmission cost for processing the spatial semijoin in the optimized strategy is

$$\begin{aligned} &T\text{cost}(\text{X}, \text{Y}, \text{Z}) \\ &\quad = \#\text{tuples}(\text{Y}) * T\text{cost}(\text{MBR}) \\ &\qquad + \#\text{qualifiers}(\text{X}) * (T\text{cost}(\text{ID}) + T\text{cost}(\text{tuple})) \\ &\qquad + \#\text{qualifiers}(\text{Z}) * T\text{cost}(\text{tuple}). \end{aligned} \quad (5)$$

The first term is the cost of transmitting the spatial attribute X from spatial relation Y to spatial relation Z. The second term is the cost of both transmitting back to Y the corresponding tuple identifiers for the qualifying the MBRs in X, and then transmitting the tuples that correspond to those tuple identifiers to the query site. Finally, the third term is the cost of transmitting qualifying tuples from Z to the query site. This cost is calculated for every pair (Y, Z) of sites that are involved in the query, with all costs summed together to obtain the total cost of the query. The function #qualifiers(relation) return the number of tuples from a relation that participate in the result of a spatial semijoin operation.

For the naïve approach, the total data transmission cost consists of the transmission of all participating spatial relations to the query site. The cost for shipping one spatial relation R to the query is calculated as

$$T\text{cost}(\text{R}) = \#\text{tuples}(\text{R}) * T\text{cost}(\text{tuple}). \quad (6)$$

D. I/O Cost Calculation

The I/O cost will be calculated as time in milliseconds. We use a seek time of 4 ms, a transfer time of 0.3 ms per block, a R-tree node size of 1024 B (i.e., approximately 50 MBRs), and a disk block size of 1024 B. We first present the calculations for various costs and finish with overall costs for both the optimized strategy and the naïve approach.

We estimate the cost of fetching spatial attribute X from spatial relation Y on a site from set P as

$$\text{IOcostA}(\text{X}, \text{Y}) = \#\text{nodes}(\text{X}, \text{Y}) * (\text{seek} + \text{transfer}) \quad (7)$$

where #nodes(X, Y) is the number of leaf nodes in the R-tree that is indexing spatial relation Y.

We estimate the I/O cost for reading in spatial relation Y on a site as

$$\begin{aligned} &\text{IOcostR}(\text{Y}) \\ &\quad = \text{seek} + \text{transfer}\left(\frac{\#\text{tuples}(\text{Y}) * \text{tuple\_size}(\text{Y})}{\text{block\_size}}\right). \end{aligned} \quad (8)$$

The relation is only being read in once because we are assuming the spatial attribute being sent from the other site can be stored in memory.

The cost of reading individual qualifying tuples from spatial relation Y to send to the query site is

$$\begin{aligned} &\text{IOcostRI}(\text{Y}) = \#\text{qualifiers}(\text{Y}) \\ &((\log_{50}(\#\text{tuples}(\text{Y})) + 1) * (\text{seek} + \text{transfer})) \end{aligned} \quad (9)$$

where $\log_{50}(\#\text{tuples}(\text{Y}))$ is the height of the R-tree that is indexing spatial relation Y.

The cost of writing the set of qualifying tuples from spatial relation Y as they arrive at the query site is

$$\text{IOcostW}(\text{Y}) = (\text{seek} + \text{transfer}) * \#\text{qualifiers}(\text{Y}). \quad (10)$$

We assume the worst case, here, that tuples are arriving individually from any site and, therefore, each may require a seek and transfer to store with the correct relation.

Given the above equations, for the optimized strategy, the I/O cost for performing a spatial semijoin between spatial relations Y and Z on spatial attribute X, and sending the intermediate results to the query site is

$$\begin{aligned} &\text{IOcost}(\text{X}, \text{Y}, \text{Z}) = \text{IOcostA}(\text{X}, \text{Y}) + \text{IOcostR}(\text{Z}) \\ &\text{IOcostRI}(\text{Y}) + \text{IOcostW}(\text{Y}) + \text{IOcostW}(\text{Z}) \end{aligned} \quad (11)$$

where the first term is the cost of projecting the spatial attribute X from spatial relation Y, the second term is the cost of reading in spatial relation Z in order to perform the spatial semijoin, the third term is the cost of reading individual tuples from spatial relation Y to send to the query site, and the fourth and the fifth terms are the cost of writing the intermediate result tuples from spatial relations Y and Z, respectively.

Finally, the I/O costs for the final join of n relations at the query site are the following. For the first pair of intermediate result spatial relations YI and ZI, and assuming a block nested loop spatial join is used, we require that each block is fetched in one relation

$$\begin{aligned}&\text{IOcostJ1(YI)}\\&= (\text{seek} + \text{transfer}) * \left(\frac{\text{\#tuples(YI)} * \text{tuple\_size(YI)}}{\text{block\_size}}\right)\end{aligned} \quad (12)$$

and for each block in spatial relation YI, the entire spatial relation ZI must be fetched for comparison

$$\begin{aligned}&\text{IOcostJ2(ZI)}\\&= \text{seek} + \text{transfer} * \left(\frac{\text{\#tuples(ZI)} * \text{tuple\_size(ZI)}}{\text{block\_size}}\right).\end{aligned} \quad (13)$$

Then, for the remaining $n-2$ spatial relations, each is fetched and compared with the intermediate result using (13). It is assumed, here, that the intermediate result is smaller, will fit in memory and therefore does not need to be written out to disk after each join is performed.

For the naïve approach, the total I/O cost will consist of the cost of reading the spatial relation from each site, writing the tuples at the query site, and performing the final spatial join. First, the total I/O cost for shipping spatial relation Y to the query site is

$$\text{IOcost(Y)} = \text{IOcostR(Y)} + \text{IOcostW(Y)}. \quad (14)$$

Finally, for the naïve approach, the total I/O cost for performing the final spatial join is the same as that for the optimized strategy.

E. CPU Cost Calculation

The CPU cost will be determined by calculating, for both the optimized and naïve strategies, the total number of spatial predicate comparisons that are carried out. For the optimized strategy, this will include spatial predicate comparisons for all the semijoins and the final joins at the query site. For the naïve strategy, this will include the spatial predicate comparisons at the query site. In addition, we discuss the CPU running time for the semijoin operations.

VII. Evaluation Results

In this section, we present the results of our empirical evaluation. We present the results for the geographically distributed spatial queries that involves two of the six sites, followed by four of the six sites, and finally, all six sites.

A. Two-Site Query Test

We first present the results of the comparison that involved two sites of the geographically distributed spatial database. Table I shows the results of the data transmission cost comparison. Along with the pairs of spatial relations (i.e., sites) that were evaluated, the cost (in bytes) of both our optimized strategy (column optimized) and the naïve approach (column naïve) are presented. In all cases, the optimized strategy has a lower data transmission cost over the naïve approach. In particular, the most significant improvement is achieved when there exists a significant difference in the size of the spatial relations between the two sites. For example, when the query involves the sites that contain the 1000- and 100 000-tuple spatial relations, we have over 90% less data that are being transmitted when the Optimized strategy is being used to process the query.

TABLE IV
FOUR SITES—TRANSMISSION COST

Site 1	Site 2	Site 3	Site 4	Optimized	Naïve	%Imp
1000	2000	4000	6000	251276	884000	72
1000	2000	8000	10000	259718	1428000	82
4000	6000	8000	10000	853890	1904000	55
1000	2000	40000	60000	223806	7004000	97
1000	2000	80000	100000	234660	12444000	98
4000	6000	80000	100000	728734	12920000	94

TABLE V
FOUR SITES—I/O COST

Site 1	Site 2	Site 3	Site 4	Optimized	Naïve	%Imp
1000	2000	4000	6000	28074.10	75004.68	63
1000	2000	8000	10000	35089.12	101975.72	66
4000	6000	8000	10000	105524.10	165277.78	36
1000	2000	40000	60000	33411.46	668016.35	95
1000	2000	80000	100000	182704.39	1247903.95	85
4000	6000	80000	100000	44804.09	898847.504	95

Table II shows the results of the I/O cost comparison. Again, along with the pairs of relations that were evaluated, the cost (in milliseconds) of both our optimized strategy (column optimized) and the naïve approach (column naïve) is presented. We see a trend that is very similar to that for the data transmission costs. The optimized strategy outperforms the naïve approach in all cases, with the most significant differences occurring when there exists a significant difference in size of the spatial relations across the two sites.

Table VI shows the CPU cost results. We do find a slight increase in the number of spatial predicate comparisons for the optimized strategy over the naïve approach. However, this increase is $<10\%$, and is significantly less when the spatial relations vary greatly in size. In addition, the increases are minimal when compared with the significant savings in the data transmission and I/O costs.

B. Four-Site Query Test

We now present the results for the four-site queries. Table IV shows the data transmission results, while Table V shows the I/O results. For both cases, we find that the optimized strategy outperforms the naïve approach. In addition, we also find that the most significant improvement in both data transmission cost and I/O cost occurs where a significant size difference exists between the spatial relations—in this case, 1000, 2000, 80 000, and 100 000 tuples.

Table VI shows the CPU cost results. We do find a slight increase in the number of spatial predicate comparisons for the Optimized strategy over the naïve approach. However, this overall increase is $<10\%$, and is significantly less when the

TABLE VI
FOUR SITES—CPU COST

Site 1	Site 2	Site 3	Site 4	Optimized	Naïve	%Imp
1000	2000	4000	6000	20439828	19146112	-7
1000	2000	8000	10000	37545385	35284249	-6
4000	6000	8000	10000	183096566	169369526	-8
1000	2000	40000	60000	165456586	163345400	-1.3
1000	2000	80000	100000	291076561	287945676	-1.1
4000	6000	80000	100000	1322953243	1282186309	-3.2

spatial relations vary greatly in size. Again, these increases are minimal when compared with the significant savings in the data transmission and I/O costs.

C. Six-Site Query Test

Finally, we present the results of the two six-site queries: 1) from 1000 to 10 000 tuples and 2) from 1000 to 100 000 tuples.

For the first query, we found the data transmission cost from optimized strategy to be 616 550 B and that from the naïve strategy to be 2 108 000 B. This gives an improvement of ∼70%. For the I/O cost, we found that the I/O cost for the optimized strategy is 76978.83 ms, while the cost for the naïve strategy is 142 784 ms, for an improvement of ∼46%. In addition, we found that 26 739 820 231 spatial predicate comparisons were required by the optimized strategy, and 26 732 000 000 were required by the naïve strategy, for a small increase in CPU cost.

For the second query, we found a data transmission cost of 376 896 B from the optimized strategy and 13 940 000 B from the naïve strategy, for a significant improvement of ∼97%. With respect to the I/O cost, the optimized strategy achieved 64618.22 ms while the naïve strategy produced 942219.46 ms, for an improvement of ∼93%. Finally, once again the CPU times are comparable, with a predicate comparison count of 14 054 794 061 for the optimized strategy and 14 050 000 000 for the naïve strategy.

D. Discussion

In all cases, we discovered a significantly lower data transmission cost and lower I/O cost from the optimized over the naïve approach. We discovered the following trends. First, the greater the difference in the number of tuples between the participating relations, the greater the reduction in both the data transmission cost and the I/O cost that the optimized strategy achieves over the naïve approach.

Second, we discovered that, although the CPU cost—with respect to the number of spatial predicate comparisons—is slightly higher for the optimized strategy than the naïve approach, we note that the number of spatial predicate comparisons that are carried out during the final spatial join is a very small fraction of the overall number required by the final spatial join in the naïve approach. The increase in CPU cost by the optimized approach is at most 10% and in many cases, <5%, over the CPU cost by the naïve approach. This is minimal compared with the significant reduction in the other two factors.

We also performed some evaluations that utilized spatial relations containing over 100 000 tuples up to 1 million tuples. We found similar trends with respect to both the data transmission and I/O costs.

However, with respect to actual CPU clock time, for the queries involving two sites, the CPU time for semijoin execution ranged from 1 min for the 1000–4000 tuple query to 50 min for the 2000–100 000 tuple query. When working with spatial relations containing over 100 000 tuples, the time it look to execute the spatial semijoins proved to be a bottleneck. This is due to the choice of using a block nested loop join, which works well for smaller spatial relations but not larger ones. In addition, this is due to running our simulations on a machine that is not high performance. However, further improvement can be gained by considering other options for spatial joins and spatial semijoins.

VIII. CONCLUSION

In this paper, we propose a strategy for optimizing queries in a geographically distributed spatial database that involves spatial relations on more than two sites. Our optimized strategy focuses on minimizing the cost of data transmission, as well as the I/O cost, by applying spatial semijoins. Smaller spatial attributes are chosen for transmission and application to larger spatial relations so that both the overall data transmission and I/O costs are minimized.

We also evaluate empirically our optimized strategy versus the naïve strategy. Our results are obtained through simulation. We show that significant reductions in the data transmission cost and I/O cost over all queries are achieved when then optimized strategy is utilized. We find at least a 50% reduction in most cases, in the amount of data being transmitted and the amount of I/O being incurred. In particular, when the query involves both smaller and larger spatial relations, the reductions in both data transmission and I/O costs are very significant, at ∼90% or greater. With respect to CPU, the optimized strategy achieves a slightly higher number of spatial predicate comparisons than the naïve strategy. However, for all cases it is <10% and for many it is <5%. This is a small price to pay for the significant achievements in the data transmission and I/O costs.

A. Future Work

Some directions of future work include the following. One is to create a real geographical distributed spatial database system with multiple sites, which will provide a means to better evaluate our strategy. Another is to evaluate the two-site version of our strategy (i.e., when only two sites are involved) versus other existing strategies. Although the focus of this paper was to extend the number of sites handled by a geographically distributed spatial query, evaluating the efficiency of our algorithm in the two-site case versus existing strategies is also important and would better identify if our strategy is superior in this situation.

A further research direction includes the consideration of fault tolerance, in particular, using data replication and fragmentation, for our strategy in order to further improve upon its performance. Finally, some final research directions are to develop and evaluate other strategies for processing and optimizing a geographically distributed spatial query. In particular, it is important to consider the following.

1) The case of spatial relations with more than one spatial attribute.
2) The use of join indices [34] and other implementations of the spatial join, spatial semijoin, and a spatial Bloom filter, as ways to improve overall performance, and in particular, the CPU time for processing very large spatial data sets.
3) Estimating the selectivity of spatial attributes when they contain (mostly) distinct objects.

As discussed, very limited work has been proposed, which leads to many exciting opportunities for research in the area of geographically distributed spatial query processing.

ACKNOWLEDGMENT

The authors would like to thank the referees for their thoughtful and constructive reviews of previous drafts of this paper.

REFERENCES

[1] W. Osborn and S. Zaamout, "Multiple-site distributed spatial query optimization using spatial semijoins," in *Proc. 10th Int. Baltic Conf. Databases Inf. Syst.-Local*, 2012, pp. 11–19.

[2] R. Abdalla and V. Tao, "Integrated distributed GIS approach for earthquake disaster modeling and visualization," in *Geo-Information for Disaster Management*. Berlin, Germany: Springer-Verlag, 2005, pp. 1183–1192.

[3] T. Hunter, "A distributed spatial data library for emergency management," in *Geo-Information for Disaster Management*. Berlin, Germany: Springer-Verlag, 2005, pp. 733–750.

[4] V. Gaede and O. Günther, "Multidimensional access methods," *ACM Comput. Surv.*, vol. 30, no. 2, pp. 170–231, Jun. 1998.

[5] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2003.

[6] M. Özsu and P. Valduriez, *Principles of Distributed Database Systems*. New York, NY, USA: Springer-Verlag, 2011.

[7] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, Jr., "Query processing in a system for distributed databases (SDD-1)," *ACM Trans. Database Syst.*, vol. 6, no. 4, pp. 602–625, 1981.

[8] P. M. G. Apers, A. R. Hevner, and S. B. Yao, "Optimization algorithms for distributed queries," *IEEE Trans. Softw. Eng.*, vol. 9, no. 1, pp. 57–68, Jan. 1983.

[9] C. Wang, V. O. K. Li, and A. L. P. Chen, "Distributed query optimization by one-shot fixed-precision semi-join execution," in *Proc. 7th Int. Conf. Data Eng.*, Apr. 1991, pp. 756–763.

[10] N. Roussopoulos and H. Kang, "A pipeline N-way join algorithm based on the 2-way semijoin program," *IEEE Trans. Knowl. Data Eng.*, vol. 3, no. 4, pp. 486–495, Dec. 1991.

[11] G. Lohman *et al.*, "Query processing in R*," in *Query Processing in Database Systems*. Berlin, Germany: Springer-Verlag, 1985, pp. 31–47.

[12] P. Legato, G. Paletta, and L. Palopoli, "Optimization of join strategies in distributed databases," *Inf. Syst.*, vol. 16, no. 4, pp. 363–374, 1991.

[13] C. Wang, A. L. P. Chen, and S.-C. Shyu, "A parallel execution method for minimizing distributed query response time," *IEEE Trans. Parallel Distrib. Syst.*, vol. 3, no. 3, pp. 325–332, May 1992.

[14] D. Kossmann, "The state of the art in distributed query processing," *ACM Comput. Surv.*, vol. 32, no. 4, pp. 422–469, 2000.

[15] M.-S. Kang, S.-K. Ko, K. Koh, and Y.-C. Choy, "A parallel spatial join processing for distributed spatial databases," in *Proc. 5th Int. Conf. Flexible Query Answering Syst. (FQAS)*, 2002, pp. 212–225. [Online]. Available: http://portal.acm.org/citation.cfm?id=645424.652610

[16] K.-L. Tan, B. C. Ooi, and D. J. Abel, "Exploiting spatial indexes for semijoin-based join processing in distributed spatial databases," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 6, pp. 920–937, Nov./Dec. 2000.

[17] D. J. Abel, B. C. Ooi, K.-L. Tan, R. Power, and J. X. Yu, "Spatial join strategies in distributed spatial DBMS," in *Proc. 4th Int. Symp. Adv. Spatial Databases*, 1995, pp. 348–367.

[18] O. Karam, "Optimizing distributed spatial joins," Ph.D. dissertation, Dept. Comp. Sci., Tulane Univ., New Orleans, LA, USA, 2001.

[19] O. Karam and F. Petry, "Optimizing distributed spatial joins using R-trees," in *Proc. 43rd ACM Southeast Conf.*, 2006, pp. 222–226.

[20] Y. Hua, B. Xiao, and J. Wang, "BR-tree: A scalable prototype for supporting multiple queries of multidimensional data," *IEEE Trans. Comput.*, vol. 58, no. 12, pp. 1585–1598, Dec. 2009.

[21] E. Wong, "Retrieving dispersed data from SDD-1," in *Proc. 2nd Berkeley Workshop Distrib. Data Manage. Comput. Netw.*, 1977, pp. 217–235.

[22] E. H. Jacox and H. Samet, "Spatial join techniques," *ACM Trans. Database Syst.*, vol. 32, no. 1, 2007, Art. ID 7.

[23] Y.-W. Huang, N. Jing, and E. A. Rundensteiner, "Integrated query processing strategies for spatial path queries," in *Proc. 13th Int. Conf. Data Eng.*, Apr. 1997, pp. 477–486.

[24] L. Arge, O. Procopiuc, S. Ramaswamy, T. Suel, and J. C. Vitter, "Scalable sweeping-based spatial join," in *Proc. 24th Int. Conf. Very Large Databases*, 1998, pp. 570–581.

[25] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[26] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 1984, pp. 47–57.

[27] A. Mondal, Y. Lifu, and M. Kitsuregawa, "P2PR-tree: An R-tree-based spatial index for peer-to-peer environments," in *Proc. EDBT Workshops*, 2004, pp. 516–525.

[28] Y. Tao, D. Papadias, and J. Sun, "The TPR*-tree: An optimized spatio-temporal access method for predictive queries," in *Proc. 29th Int. Conf. Very Large Databases*, 2003, pp. 790–801.

[29] Y. Zhong, J. Han, T. Zhang, Z. Li, J. Fang, and G. Chen, "Towards parallel spatial query processing for big spatial data," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp. Workshops*, May 2012, pp. 2085–2094.

[30] A. Aji *et al.*, "Hadoop GIS: A high performance spatial data warehousing system over mapreduce," in *Proc. VLDB*, vol. 6. 2013, pp. 1009–1020.

[31] P. Kalnis, N. Mamoulis, S. Bakiras, and X. Li, "Ad-hoc distributed spatial joins on mobile devices," in *Proc. 20th IEEE Int. Parallel Distrib. Process. Symp.*, Apr. 2006, pp. 1–10.

[32] P. Bodorik, J. S. Riordon, and J. S. Pyra, "Deciding to correct distributed query processing," *IEEE Trans. Knowl. Data Eng.*, vol. 4, no. 3, pp. 253–265, Jun. 1992.

[33] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-Trees: Theory and Applications*. London, U.K.: Springer-Verlag, 2005.

[34] P. Valduriez, "Join indices," *ACM Trans. Database Syst.*, vol. 12, no. 2, pp. 218–246, 1987.

Wendy Osborn (M'14) received the B.C.S. (Hons.) and M.Sc. degrees from the University of Windsor, Windsor, ON, Canada, in 1996 and 1998, respectively, and the Ph.D. degree from the University of Calgary, Alberta, AB, Canada, in 2005, all in computer science.

She is currently an Associate Professor with the Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, AB, Canada. Her current research interests include distributed spatial query optimization, query processing for location-based services in mobile information systems, spatial indexing, and digital humanities.

Saad Zaamout received the B.Sc. degree in computers and computer information systems from Philadelphia University, Amman, Jordan, in 2005.

He was a Research Assistant and B.Sc. Student in computer science with the University of Lethbridge, Alberta, AB, Canada. He is currently a Software Developer with SCA Interactive, Calgary, AB, Canada.