A Strategy for Optimizing a Multi-site Query in a Distributed Spatial Database

Saad Zaamout and Wendy Osborn

Department of Mathematics and Computer Science, University of Lethbridge, Lethbridge, Alberta, T1K 3M4, Canada

Abstract. In this paper, we present a novel strategy for distributed spatial query optimization that involves multiple sites. Most previous work in the area of distributed spatial query processing and optimization focuses only on strategies for performing spatial joins and spatial semijoins, and distributed spatial queries that only involve two sites. We propose a new strategy, called the Restricted strategy, for optimizing a distributed spatial query. It uses spatial semijoins and can involve any number of sites in a distributed spatial database. The Restricted strategy improves upon an existing strategy by sending group approximations, instead of sending approximations for all objects, in order to reduce the number of comparisons between objects and thereby minimize the CPU and data transmission cost. A performance evaluation of our strategy finds that it significantly minimizes the number of data comparisons and CPU time of distributed spatial queries.

Keywords: distributed query processing, spatial data.

1 Introduction

A distributed spatial database system such as a Geographical Information System [13] contains several spatial database sites that are dispersed geographically. Each site manages its own collection of spatial data, but work collectively for processing inter-site queries and transactions. For example, suppose we have three sites in a province, where one site manages regions of population growth, one manages regions of Lyme disease, and one manages concentrations of wildlife. Each site would be managed separately, but could be use collectively to answer queries such as, "How many areas of population could be affected in the future with Lyme disease given animal migration patterns?".

An important requirement of a distributed spatial database is the ability to efficiently process a query that requires spatial data from multiple sites. Historically, research in distributed relational databases focused on generating query execution plans that minimized the cost of data transmission over the network [3,11]. However, spatial data is more complex than alphanumeric data, which thereby increases the complexity of joining spatial relations. CPU and I/O costs should also be considered when processing a distributed spatial query [13].

Most existing strategies for distributed spatial queries only work for two sites. One exception to this does not handle spatial joins, while the other exception has only looked at minimizing data transmission cost. Therefore, we propose a new, improved multi-site distributed spatial query processing strategy. We find through our performance evaluation significant improvements in the number of comparisons that take place for a semijoin, which lowers both the CPU and data transmission costs.

This paper proceeds as follows. Section 2 presents related work in distributed spatial query processing. Section 3 presents our novel strategy for optimizing distributed spatial queries. Section 4 presents a performance evaluation of our strategy versus a recently proposed strategy that works for multiple sites. Finally, Section 5 concludes the paper and presents directions of future work.

2 Related Work

Previously proposed strategies in distributed spatial query processing focus on the use of spatial joins, spatial semijoins, and Bloom filters for processing queries. For all strategies below, with the exception of [6,12], all proposed strategies for query processing work for two sites only.

2.1 Spatial Join

A spatial join [13] takes two spatial relations R and S, and relates pairs of tuples between them, using a spatial predicate that is applied to the spatial attribute values. An example of a spatial join is an overlap spatial join, where each pair of objects is tested for overlap. Most spatial join algorithms are designed for a centralized system [7]. One application of a distributed spatial join comes from Kang et al. [8]. Their parallel strategy has two phases. The first phase is data redistribution, where on each site, the space that contains objects is partitioned into regions. A subset of regions is transmitted between sites so that each site has the same regions for both data sets. This subset is chosen so that the lowest overall response time is achieved. The second phase is filter and refinement, where the spatial join is performed on each site. An experimental evaluation shows that the parallel spatial join technique has a 33% faster response time over a semijoin-based strategy. One limitation of the strategy is no discussion of an extension to more than two sites.

2.2 Spatial Semijoin

A spatial semijoin [2] 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 second site are shipped back to the first site for joining. Existing work in distributed spatial semijoins focuses on modifying the operator [2,14,10] for a two-site query, with one additional work applying the spatial semijoin in a multiple-site query [12].

Abel *et al.* [2,14] combine the spatial semijoin operator that with the filter stage of spatial query processing to reduce the data transmission, I/O and CPU costs. Two adaptations of the spatial semijoin are proposed. The first is a "projection" of a set of MBRs (minimum bounding rectangles) from one spatial relation that is transmitted to the second site and applied to the other relation. The second is a "projection" consisting of a single-dimensional mapping that represents the objects in each relation. A performance evaluation between both approaches shows that: 1) for datasets with very large spatial descriptions, both strategies perform the same, 2) for datasets with smaller spatial descriptions, a semijoin that uses single-dimensional mapping works best, 3) using the R-tree for retrieving MBRs incurs significant CPU costs, and 4) single-dimensional mapping causes more false drops than MBRs.

The version from Karam and Petry [10] takes MBRs from different levels of the R-tree for the spatial semijoin, instead of 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 data, but not when applied to randomly distributed rectangle sets. Other limitations of their work are no comparison versus other strategies and no consideration of CPU time.

Osborn and Zaamout [12] propose a distributed query processing strategy that works across multiple sites. Their approach is to apply semijoins by shipping smaller spatial projections to the sites of larger relations. A performance evaluation shows significant improvements in data transmission costs over the Naïve approach of shipping all relations directly to the query site. However, a limitation of this work is the lack of CPU and I/O cost evaluation.

2.3 Bloom Filters

A Bloom filter [1] is a hashed bit array that provides a compact but imprecise representation of the values of a joining attribute. A '1' bit represents the possible existence of a joining attribute value, while a '0' bit represents the absence of the value. Two proposed strategies [9,6] propose augmentations to the Bloom filter in a distributed environment.

Karam [9] propose a 2-dimensional bit-matrix approach for performing a semijoin of two relations that focuses on minimizing the data transmission, I/O and CPU costs. A 2-dimensional space is partitioned into equal-sized regions, with each region mapping to a bit in a 2-dimensional array. If a region contains objects, the corresponding bit is set to 1. This bit-matrix is transmitted to the site containing the other relation, and is applied by testing each region containing 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 data. One Limitation of this work is no evaluation against the original spatial semijoin.

Hua et al. [6] propose the BR-tree, which is an R-tree augmented with Bloom filters to support exact-match queries. Along with an MBR, each node contains a Bloom filter that also represents the same set of objects represented by the MBR.

In a leaf node, a Bloom filter is created by taking each object and producing k filter bits. In a non-leaf node, a Bloom filter is created by intersecting the Bloom filters in its child node. Although the BR-tree supports exact-match queries by using Bloom filters, it still requires the MBRs for region and point queries. To process distributed region, point, and exact-match queries, the root of every BR-tree is duplicated across every site in the distributed 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. However, a significant limitation is a lack of support for spatial joins.

3 Restricted Strategy

In this section, we present our strategy for processing and optimizing a distributed query. The Restricted strategy works for any number of sites, with the only requirement being that the number of sites must be a multiple of two. In addition, it improves upon the Optimized strategy [12] by reducing the number of object comparisons required in order to minimize both the data transmission cost and the CPU cost of the query. This is achieved by utilizing an optimization technique from the SpatialJoin2 join algorithm, which is proposed by Brinkhoff *et al.* [4]. We first provide a brief summary of both the Optimized strategy and SpatialJoin2 algorithm before presenting the Restricted strategy.

3.1 Background

The Optimized algorithm [12] uses spatial semijoins to reduce the data transmission cost and I/O cost of a distributed spatial query. This is accomplished by sending smaller projections of spatial attributes to larger relations for semijoin processing. To reduce I/O costs, the spatial projection is obtained by traversing the leaf level of an R-tree [5], which is used to index the spatial attribute of the relation on the site. This is significantly cheaper in terms of I/O than retrieving the entire relation and obtaining the spatial attribute from each tuple.

Given n sites that each contains a spatial relation, the Optimized strategy has four main steps [12]:

1. *Sorting and grouping by increasing spatial attribute cardinality.* After sorting all participating relations by increase spatial attribute cardinality, two groups of relations is formed - one containing the smaller cardinality relations, and the other containing the larger cardinality relations.
2. *Transmission of spatial attributes.* The spatial attribute with the smallest cardinality relation from the first group is sent to the site with the smallest cardinality relation from the second group. This strategy is further applied to the next smallest cardinality relations in both sets, and so on until all relations are processed.
3. *Semijoin execution.* Next, spatial semijoins are performed in parallel, and the identifiers of the spatial projection that qualified in the spatial semijoin are sent back to the originating site.

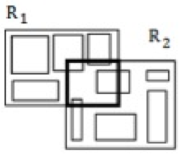

Fig. 1. Intersection Region

4. *Transmission of qualifying tuples to query site for the final join and processing.* From all sites, all tuples that qualified in their respective semijoins are shipped to the query site for final processing.

The algorithm SpatialJoin2 utilizes the concept of *intersection regions*. Given two leaf nodes and the corresponding minimum bounding rectangles (MBRs) that contain the objects in each leaf node, the intersection region is the area that is overlapped by both MBRs. The idea is that any object in either leaf node must overlap the intersection region in order to potentially qualify for a join with any other object. This can significantly reduce the number of comparisons between objects between different sites that must take place. Figure 1 depicts an intersection region with the rectangles that do, and do not, overlap with it.

3.2 Restricted Strategy

Given n participating sites, where each contains one spatial relation, we have the following steps:

1. Sorting and grouping by spatial attribute cardinality,
2. Creation and Transmission of MBRs that bound subsets of objects,
3. Calculation of intersection regions,
4. Identification of objects that intersect the intersection regions,
5. Transmission of qualifying tuples to query site for the final join.

Steps 1 and 5 are the same as those in the Optimized Strategy. We include them in the overall description for completeness.

First, the relations are ordered by increasing cardinality of their spatial attributes, and then are grouped into sets P (lower half of ordering, smaller spatial attribute cardinality) and Q (upper half of ordering, larger spatial attribute cardinality). Second, for each relation in P and Q, a set of MBRs is created for each subset of spatial objects in the relation.

If the spatial attribute is indexed with an R-tree, then the set of MBRs from each leaf node (which can be obtained from the parent node) can be used for the required set of MBRs. Figure 2 depicts an R-tree that is indexing the given set of objects (from [5]). In the Optimized strategy [12], the leaf-level entries (MBRs and identifiers) would be obtained and used as the required spatial projection. In the new Restricted Strategy, the non-leaf-level entries (in the example, the entries in the root node), are the ones obtained and sent as the set of MBRs

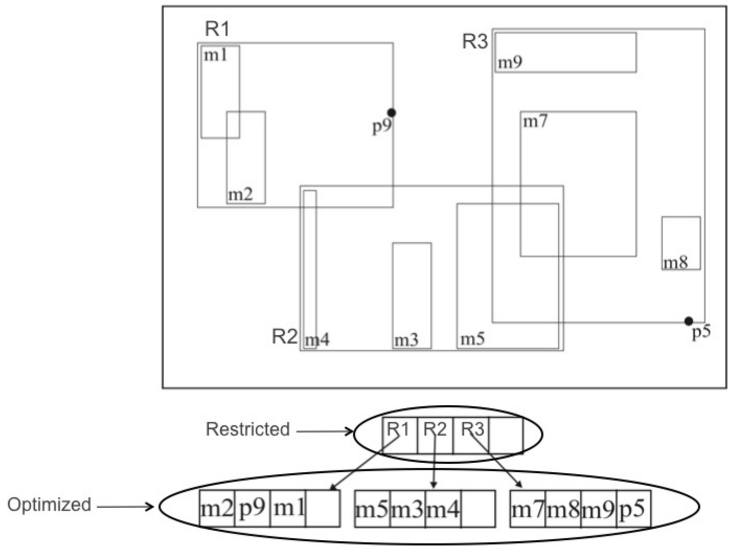

Fig. 2. Obtaining Spatial Projections from R-tree

that bound subsets of objects. In the example, we see that bounding box R1 contains the objects m1, m2 and p9. Similarly, R2 and R3 bound the objects in their respective leaf nodes. Therefore, the set of R1, R2 and R3 are selected as the set of MBRs that represent all objects.

Then, this set of MBRs from each site in P is transmitted to a site in Q in the following manner:

– The MBR set from the relation with the smallest spatial cardinality in P is sent to the relation with the smallest spatial attribute in Q,
– The MBR set from the relation with the next smallest spatial cardinality in P is sent to the relation with the next smallest spatial attribute in Q,
– and so on... until,
– The MBR set from the relation with the largest spatial cardinality in P is sent to the relation with the largest spatial attribute in Q.

For the third step, for each relation in Q, all MBRs from the transmitted set are intersected with the MBRs from the local set to create the set of intersection regions. Readers can refer back to Figure 1 for an example intersection region.

In the fourth step, the spatial objects from the relation in Q are tested for overlap with the appropriate intersection region. At the same time, the intersection regions are sent back to the relation from P that sent the original MBR set. Then, for each relation in P, each object is tested for overlap with the appropriate intersection region.

Finally, in the fifth step, the tuples of all objects that overlapped with an intersection region are shipped to the query site for the final join.

Table 1. Two-site Restricted Query vs. Optimized Query

Site 1	Site 2	TT-O	TT-R	%Imp	TC-O	TC-N	%Imp
100	400	1876	183	90.24	40000	9650	75.87
100	600	2847	260	90.86	60000	11642	80.59
100	800	3868	317	91.80	80000	11007	86.24
100	1000	4619	316	93.15	100000	7689	92.31
200	400	3782	1460	61.39	80000	51250	35.93
200	600	5635	1526	72.91	120000	47829	60.14
200	800	7447	1163	84.30	160000	47179	70.51
200	1000	9526	1133	88.10	200000	40407	79.79

4 Evaluation

In this section, we present our experimental methodology and results of our performance evaluation of the Restricted strategy. Our evaluation compares the Restricted strategy against the Optimized strategy [12] to determine which achieves both the lower CPU cost and the lower the number of object comparisons. Although we do not directly compare the data transmission costs between the two strategies, the fact that the Restricted strategy is transmitting fewer representative MBRs instead of entire spatial attributes can lead to data transmission cost savings.

Our experimental framework consists of a simulated distributed spatial database. For the purpose of our experiments, we used six relations. Each relation contains 100, 200, 400, 600, 800, and 1000 tuples respectively. Each tuple consists of a spatial object (in the form of a square), and some non-spatial attributes. Also, each relation is indexed with an R-tree [5], which uses nodes that can hold 50 entries. Each relation will be its own 'site' in our simulated system.

Our evaluation criteria will be evaluated using the following. The CPU time will be tracked using the system clock for all semijoin operations. The number of comparisons made during all semijoin operations will be tracked up to (but not including) the final query site join.

4.1 Two-Site Restricted Query Test

The first set of comparisons of our Restricted strategy is for distributed queries involving two sites. Table 1 shows the pairs of relations that were evaluated, along with the total number of comparisons that were made (column TC-O for Optimized, column TC-R for Restricted), and the total CPU cost (column TT-O for Optimized, column TT-R for Restricted). We also record the percentage improvement in the Restricted strategy over the Optimized strategy for both cost factors.

Results show that a very significant improvement is achieved by the Restricted strategy in both the number of comparisons that are being performed and the CPU time of the distributed query. For both the CPU time and comparison count, the improvement is more significant when there exists a significant difference in the size of the spatial relations between the two sites. In addition, when

Table 2. Four-Site Restricted Query vs. Optimized Query

Site 1	Site 2	Site 3	Site 4	TT-O	TT-N	%Imp	TC-O	TC-N	%Imp
100	200	400	600	7497	1388	81.49	160000	57479	64.08
400	600	800	1000	43361	22942	47.09	920000	501520	45.49
100	200	800	1000	13117	1578	87.97	280000	51414	81.64

the smaller of the two sites contains only 100 spatial objects, the improvement in CPU time is always over 90%. The improvements in the number of comparisons that are being carried out are smaller than the CPU time improvements, but are still very significant nonetheless.

4.2 Four-Site Restricted Query Test

For our second set of tests, we compared the evaluation of the strategies for four-site queries. Table 2 shows the sites involved and the CPU times and total number of comparisons. Again, we find that the Restricted strategy outperforms the Optimized strategy in both CPU time and the number of comparisons. We also find in the situation where a significant size difference exists between the relations - again, 100, 200, 800, and 1000 tuples - the greatest improvement is achieved for both factors.

4.3 Six-Site Restricted Query Test

Finally, we performed one test that compares the Restricted strategy and the Optimized strategy when all six sites are involved. We found that the Restricted strategy performed 204,777 comparisons in 8.7 seconds, while the Optimized strategy required 620,000 comparisons and 24.7 seconds to perform them. Therefore, the restricted strategy required 66% fewer comparisons and 70% less time than the original Optimized strategy.

5 Conclusion

In this paper, we propose a new strategy for performing distributed spatial query processing that involves more than two sites. The Restricted strategy improves upon the existing Optimized strategy [12] by applying a technique for minimizing the number of object comparisons that must be carried out, which in turn reduces both CPU and data transmission costs. A performance evaluation studied the improvement of the Restricted strategy over the Optimized strategy with respect to CPU and comparison costs for two-, four- and six-site queries. Results show that the Restricted strategy outperforms the Optimized strategy with respect to the number of comparisons and CPU cost in all cases. In particular, when there is a significant size difference between the participating relations, the improvements CPU costs and the number of comparisons are the most significant.

Some of our directions of future research include the following. The first is investigating the performance (with respect to the number of comparisons) of the Restricted strategy with different numbers of MBRs in the MBR sets. It is probable that sending fewer representative MBRs may result in more savings in CPU and data transmission costs. The second is deriving more strategies for distributed spatial query processing that work for an arbitrary number of sites. In particular, we want to study the use of Bloom filters for reducing the data transmission, CPU and I/O costs of a distributed spatial query. This work will ultimately lead to the best future strategies for processing and optimizing distributed spatial queries.

References

1. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13, 422–426 (1970)
2. Abel, D.J., Ooi, B.C., Tan, K.-L., Power, R., Yu, J.X.: Spatial Join Strategies in Distributed Spatial DBMS. In: Egenhofer, M., Herring, J.R. (eds.) SSD 1995. LNCS, vol. 951, pp. 348–367. Springer, Heidelberg (1995)
3. Apers, P.M.G., Hevner, A.R., Yao, S.B.: Optimization algorithms for distributed queries. IEEE Transactions on Software Engineering 9, 57–68 (1983)
4. Brinkhoff, T., Kriegel, H.-P., Seeger, B.: Efficient processing of spatial joins using R-trees. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, New York, USA, pp. 237–246 (1993)
5. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, pp. 47–57 (1984)
6. Hua, Y., Xiao, B., Wang, J.: BR-Tree: a scalable prototype for supporting multiple queries of multidimensional data. IEEE Transactions on Computers 58, 1585–1597 (2009)
7. Jacox, E., Samet, H.: Spatial join techniques. ACM Transactions on Database Systems 34, 1–44 (2007)
8. Kang, M.-S., Ko, S.-K., Koh, K., Choy, Y.-C.: A Parallel Spatial Join Algorithm for Distributed Spatial Databases. In: Andreasen, T., Motro, A., Christiansen, H., Larsen, H.L. (eds.) FQAS 2002. LNCS (LNAI), vol. 2522, pp. 212–225. Springer, Heidelberg (2002)
9. Karam, O.: Optimizing Distributed Spatial Joins using R-trees. Ph.D. Thesis, Tulane University (2001)
10. Karam, O., Petry, F.: Optimizing distributed spatial joins using R-trees. In: Proceedings of the 43rd ACM Southeast Regional Conference, pp. 222–226 (2006)
11. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Springer, New York (2011)
12. Osborn, W., Zaamout, S.: Multiple-Site Distributed Spatial Query Optimization using Spatial Semijoins. In: Proceedings of the 10th International Baltic Conference on Databases and Information Systems, Vilnius, Lithuania (2012)
13. Shekhar, S., Chawla, S.: Spatial Databases: A Tour. Prentice Hall, New Jersey (2003)
14. Tan, K.-L., Ooi, B.C., Abel, D.J.: Exploiting spatial indexes for semijoin-based join processing in distributed spatial databases. IEEE Transactions on Knowledge and Data Engineering 12, 920–937 (2000)