An R*-tree Based Semi-Dynamic Clustering Method for the Efficient Processing of Spatial Join in a Shared-Nothing Parallel Database System

Kevin Shaw[1], John Sample[1], Mahdi Abdelguerfi[2],
Gayatri Ganpaa[2], and Maik Flanagin[2]

[1] Nava Research Laboratory, Stennis Space Center, Mississippi
{shaw, john.sample}@nrlssc.navy.mil
[2] Computer Science Department, University of New Orleans, Louisiana
{mahdi, gayatri, maik}@cs.uno.edu

Abstract. The spatial join is a computationally expensive operator to implement. The efficient implementation of the spatial join operator is, thus, desirable. This paper discusses a new parallel algorithm that implements the spatial join in an efficient manner. The proposed algorithm is compared to an existing parallel spatial join algorithm, the clone join. Both algorithms have been implemented on a Beowulf cluster and compared using real datasets. An extensive experimental analysis reveals that the proposed algorithm exhibits superior performance both in declustering time as well as in the execution time of the spatial Join query.

Keywords: Spatial join, cluster, parallel processing, declustering, Beowulf cluster.

1 Introduction and Related Work

Geospatial data sets are often large and are being constantly gathered by numerous satellites and other data collection devices. In order for the data collected to be useful, it needs to be processed and analyzed. This data is typically stored in a spatial database to facilitate processing and analysis. However, due to the massive amount of data being stored, several problems can arise. The ability to store and query this enormous amount of data is critical but may lead to performance degradation. Therefore, faster data retrieval and computation mechanisms are now required.

Performance problems with large databases have been widely documented by researchers and several techniques [10,11] have been devised to cope with this. Indeed, traditional database management architectures have difficulty meeting the I/O and compute performance levels needed to handle large volumes of geospatial data. To achieve acceptable performance levels, database systems have been increasingly required to make use of Parallelism [10,11]. One form of parallelism involves the use of compute clusters.

A cluster is simply a collection of compute nodes interconnected via some sort of network. Clusters can improve the performance of geospatial queries by exploiting

parallelism. The most popular type of compute clusters in use today are based on the Beowulf paradigm. Beowulf compute clusters are shared nothing machines in which all of the compute nodes, which are called slave nodes, are isolated on a high-speed private network that is not directly visible to the outside world. A single computer connected to the outside world (called the master node) lets a user login to the cluster and submit jobs for processing i.e. by spawning processes that will execute on the slave nodes. Beowulf clusters help speedup program execution time, which is made possible by splitting a task into several sub-tasks that can run in parallel on the slave nodes.

Parallel Database Systems employ partitioning strategies to distribute database relations across multiple processing nodes. Many schemes have been developed to distribute data across several databases (nodes) [5,6].

Various spatial join algorithms have been proposed for evaluating the spatial join operator. Most of the proposed algorithms decluster the relations into a number of fragments. The spatial join is then performed by pair-wise joining of these small fragments. Spatial partitioning functions for spatial Join algorithms are usually categorized into two types: *static* and *dynamic* declustering schemes.

In the *static* declustering technique, the space is initially decomposed into regions. Each region is mapped to a disk and the features inside a region are stored on the disk the region corresponds to. The tiling technique of [3] is an example of a static declustering technique. Other static partitioning techniques for spatial join are investigated in [12].

In the *dynamic* declustering technique, the features are inserted into a spatial index. The leaves of the spatial index are mapped to disks. In this method, the space is decomposed into regions recursively. There are a minimum and maximum number of features a region can have. Once a region exceeds the number of features specified, the region is split and the features are re-assigned to two new regions. This process is done recursively until all the features are inserted into a spatial index. In [4], a dynamic declustering technique based on the use of an R*-tree [9] for each of the two join relations is proposed. The algorithm starts from the roots of both trees and traverses both of the trees in a depth first order. For each intersecting pair of directory rectangles (minimum bounding rectangle of the data rectangles in the corresponding subtrees), the algorithm follows the corresponding references to the nodes in the lower level of the trees. Results are found when the leaf level is reached.

The rest of this paper is organized as follows. Section II deals with the computationally expensive parallel join operator. The existing clone join parallel algorithm [3] is presented, including the tiling declustering method it uses. In the same section, a new parallel algorithm referred to as *R*-tree based Semi Dynamic Parallel (RSDP) Join* is presented. In section III, a comparative analysis of the two parallel join algorithms using real datasets is performed using a Beowulf cluster. The comparison of the proposed algorithm with the tiling-based Clone Join is motivated by the well-established fact that static declustering techniques perform better [3] than their dynamic counterparts.

Two collections of datasets were used in the experimental analysis. Each collection has two datasets. The first collection's two geospatial data sets were obtained from the Bureau of Transportation Statistics (BTS) [7]: the 2002 National Transportation Data

Hydrographic *(Hydrolin)* and Railway network *(rail)* Features of a collection of adjacent States (Louisiana, Kansas, Mississippi, Arkansas, Texas, Oklahoma, and Missouri). Two datasets from the second collection were obtained from BTS as well: the 2002 National Transportation Data Hydrographic Features of Louisiana State and the 2002 National Transportation Data Railway network of Louisiana State. These datasets are obtained as compressed ESRI Shapefile format files. When imported into a database, the sizes of these datasets are given in Tables 1 &2.

Table 1. Louisiana TIGER data information

Datasets	# of Features	Total size	Type of Features
Hydrolin	31400	20.1MB	MULTILINESTRING
Rail	3543	1.8MB	MULTILINESTRING

Table 2. TIGER data of a collection of adjacent States (Louisiana, Kansas, Mississippi, Arkansas, Texas, Oklahoma, and Missouri) in the US

Datasets	# of Features	Total Size	Type of Features
Hydrolin	99737	65MB	MULTILINESTRING
Rail	35492	19MB	MULTILINESTRING

2 Parallel Spatial Join

Consider the two relations *hydrolin* and *rail.* Each of these relations has the geometry column: *the_geom* of type: MULTILINESTRING. "Find all the railways which are going across a river" is an example of a spatial join query. This example uses a spatial distance function as the join condition. The following SQL statement implements the previously defined query:

```
SELECT h.gid, r.gid FROM hydrolin as h, rail as r
WHERE distance (h.the_geom, r.the_geom) = 0
```

Spatial Joins typically operate in two steps: filter and refinement steps. In the filter step, an approximation of each spatial object (the Minimum Bounding Rectangle (MBR)) is used to eliminate those tuples that cannot be part of the result; this produces a set of candidate pairs for the spatial join. The plane sweep algorithm [8] is a typical algorithm used to implement the filtering step. In the refinement step, each candidate is examined to

check if it part of the result; this is a CPU-intensive computation geometry algorithm since it has to be checked with the exact geometry.

In the case of parallel join, the two relations to be joined are declustered by the master node into smaller fragments. The spatial join is then performed in parallel (in the slave nodes) by pair wise joining of the smaller partitions.

2.1 The Clone Join Algorithm

The Clone Join Algorithm [3] uses a declustering method referred to as *tiling*. The two join inputs are declustered using the tiling scheme. A round robin methodology is utilized to map tiles onto nodes, and then the filter and refinement steps are applied as explained in the previous section.

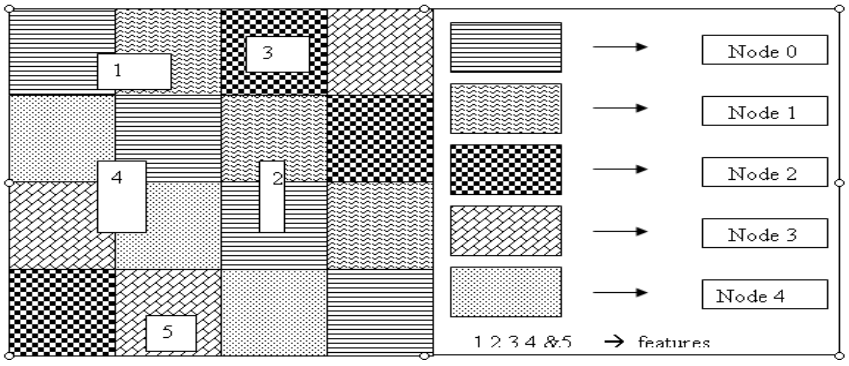

Fig. 1. Tiling declustering technique

In the tiling declustering method, the universe[1] of the relation to be distributed is divided into a number of tiles of the same size. Each tile is mapped to a node according to a round robin hashing function. Spatial objects within a tile are stored on the node that tile is mapped to. Spatial objects which overlap multiple tiles are stored on the nodes that correspond to these tiles. If a spatial object overlaps more than one tile, it is stored in all nodes that map to these tiles. The number of tiles chosen should be no less than the number of nodes. Data distribution skew is usually decreased by increasing the number of tiles. However, because the universe is divided into more tiles, many features may overlap more than one tile resulting in increased replication.

The example of Figure 1 assumes 5 nodes and a universe composed of 16 tiles. A relation is being declustered across 5 nodes using 16 tiles. In the Figure, 1, 2, 3, 4 & 5

[1] Minimum Bounding Rectangle (MBR) that covers all the spatial attributes of the relation.

represent the 5 features of the relation. Using round robin as a mapping function, features are assigned to nodes as shown in Table 3.

As it can be seen from Table 3, there is both data replication and data distribution skew. Feature 4 is stored on nodes 0, 3 & 4. Also Feature 1 is stored on node 0 and node 1, which shows a feature is replicated on more than one node. Also, node 0 has three features whereas node 2 has just one feature which shows some level of data distribution skew.

Table 3. Tiling scheme data distribution

Nodes	Features
0	1,2,4
1	1,2
2	3
3	4,5
4	4

The above steps are performed to decluster one relation. For spatial join, each of the two relations to be joined is declustered using the previously described method. In this case, the universe should now be the MBR that covers all the spatial features of both the relations.

In the next section, two variants of the clone join algorithm are investigated.

2.1 Clone Join, Variant 1 (CJV1)

The master node distributes the data to the slave nodes according to the tiling technique described previously. The filter step is performed on each node. The output of the filter step is a set of Object Identifier (OID) pairs. Each OID pair contains the Object Identifier (Feature-id) of the feature in the first relation and the OID of the feature in the second relation. This output is sent to the master node which removes the duplicate OID pairs. The duplicate OID pairs are removed from all the nodes which have the OID pair, except from one node. The resulting OID pairs, which are retained on each node after removing the duplicates, are sent back to the respective node to perform the refinement step. The refinement step is performed on each node and is intended to remove the false hits. The output, which is again a set of OID pairs, is sent to the master node.

In this method, before the refinement step, the master node has to remove the duplicate OID pairs in the slave nodes and send back the remaining OID pairs to the respective nodes. Therefore, the master node should maintain information about which OID pair belongs to which node.

2.2 Clone Join, Variant 2 (CJV2)

After receiving each database fragment obtained (via tiling) from the master node, each slave node performs the filter step locally. The refinement step is performed on each slave node and then the resulting candidate pairs are sent to the master node which removes the duplicates.

In this method, the master node doesn't have to remember which OID pair belongs to which node, but the problem with this method is that the refinement step is performed on all the OID pairs prior to removing duplicates.

2.3 R*-tree Based Semi Dynamic Parallel (RSDP) Join

The RSDP join algorithm builds an R*-tree on one of the two relations to be joined. The leaves of this R*-tree are distributed using one of two hashing functions described below. The features of the second relation are distributed statically using a tiling like approach. The leaves of the R*-tree built on the first relation are treated as tiles and, thus, the features of the second relation that overlap the leaves are stored on the nodes corresponding to the leaves. Since the leaves of an R*-tree do not overlap, there would not be any replication for the first relation, though there would be some replication for the second relation. It is emphasized that for the second relation, only the features overlapping the leaves of the R*-tree structure of the first relation are distributed since such features are the only candidates for the Join.

Two hashing functions are used to map the leaf of the R*-tree onto compute nodes. For the first hash function, k (where $k = \lceil$Total leaves/ # of Nodes$\rceil$) successive leaves are taken and stored on each compute node. Consider an R*-tree built on a relation that has 21 leaf nodes. The number of compute nodes the data has to be distributed onto is 3. Then k =7 successive leaves are stored on each compute node. For the second hash function, each leaf is stored on a node in a round robin fashion. For example, leaf number p is stored on the node ($p \bmod n$), where n is the number of compute nodes.

The examples of Figures 2 & 3 assume that Relation 1 has 8 features (*1, 2, 3 … 8*), and Relation 2 has 7 features (*1, 2, 3 …7*). Figure 2 shows the R*-tree built on the first relation: it has 4 leaves: *a, b, c* and *d*. It also shows the features numbered from 1 through 8. Each leaf is mapped to a node according to the hash functions as described above.

So for the first hash function, the features which are contained in leaves *a* and *b* are stored on the first node. The features contained in the leaves *c* and *d* are stored on the

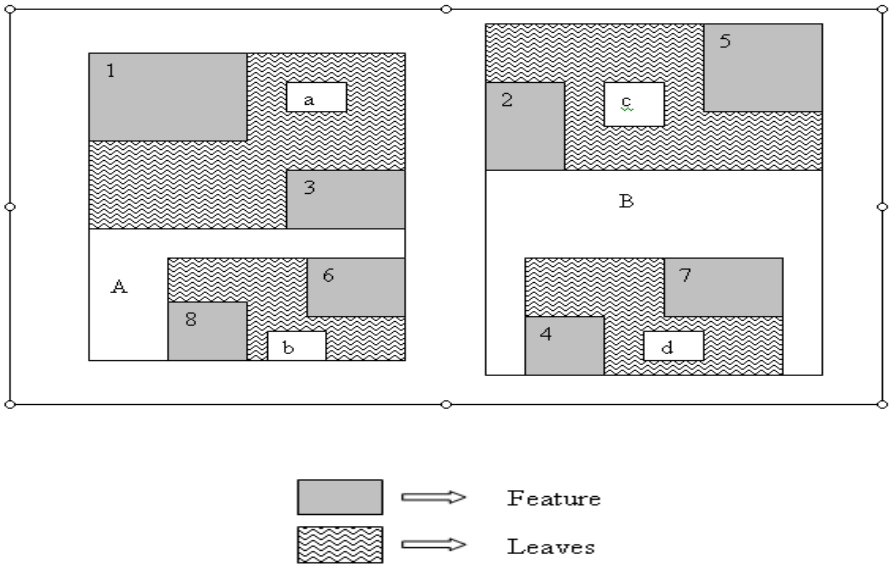

Fig. 2. R*-tree built on the first relation

second node. For the second hash function, the features in leaves *a* and *c* are stored on the first node and the features which are stored on leaves *b* and *d* are stored on the second node.

The R*-tree structure of the first relation is used to decluster the second relation. According to the first hash function, the features that overlap the leaves *a* and *b* are stored on the first node and the features that overlap leaves *c* and *d* are stored on the second node. For the second hash function, features that overlap *a* and *c* are stored on first node and features that overlap *b* and *d* are stored on second node. Figure 3 illustrates the declustering for the second relation based on the R*-tree built on the first relation.

Based on Figures 2 & 3, the features of the two relations are distributed among two nodes as shown in Tables 4 and 5. It can be observed from the two tables that there is no replication for the first relation, but there is replication for the second relation. Indeed, for the second relation in Table 5, feature 5 is stored on both node 1 and node 2. Also for the second relation in Table 4, feature 4 is stored on both nodes.

With the first hashing function, if the answers of the spatial join request are concentrated geographically on one part of the universe, then these answers will be computed by fewer compute nodes, leading to data distribution skew. This disadvantage is reduced with the second hashing function.

Table 4. First hash function

Relation/ Node	Node1	Node 2
Relation1	1,3,6,8	2,4,5,7
Relation 2	1,4,5,7	2,3,4

Table 5. Second hash function

Relation/ Node	Node 1	Node 2
Relation 1	1,2,3,5	4,6,7,8
Relation 2	1,3,4,5,7	2,5

The data is first distributed using the R*-tree based semi-dynamic declustering scheme described above. Then, the RSDP join algorithm performs the filter and refinement steps on each compute node. The resulting OID pairs from all the nodes are merged by the master node which also removes duplicate OID pairs.

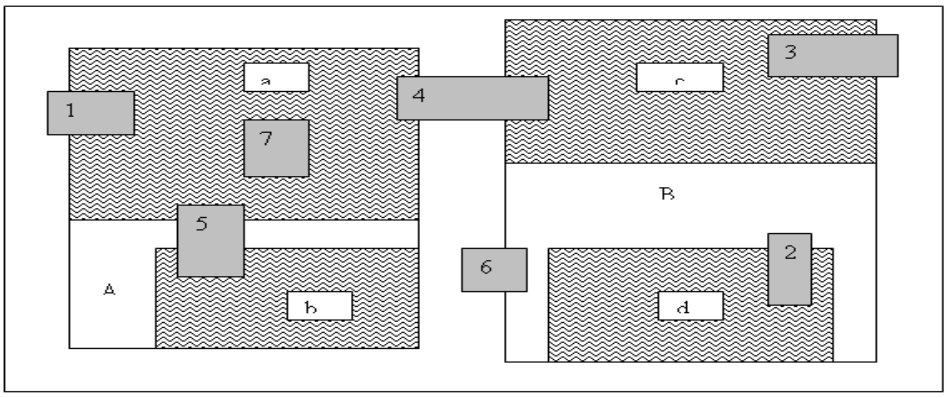

Fig. 3. Declustering the second relation based on the R*-tree structure of the first relation

3 Experimental Analysis

Two collections of data sets (*dataset1* and *dataset2* in Table1/Table2) were used in the comparative performance study. The data obtained as compressed ESRI shapefiles were loaded into PostgreSQL [2] using the PostGIS [1] extension. The Beowulf cluster used consists of slave nodes which are 2.2Ghz Intel Pentium systems with 1 GB of memory, and a master node that has dual 2.2GHz Intel Xeon processors and 2 GB of memory.

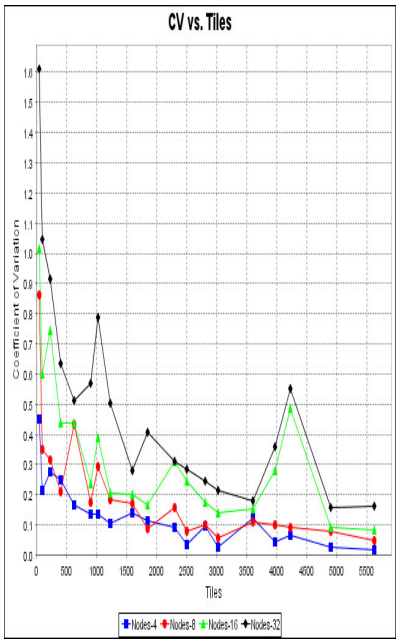

Fig. 4.1. CV vs. tiles for dataset1

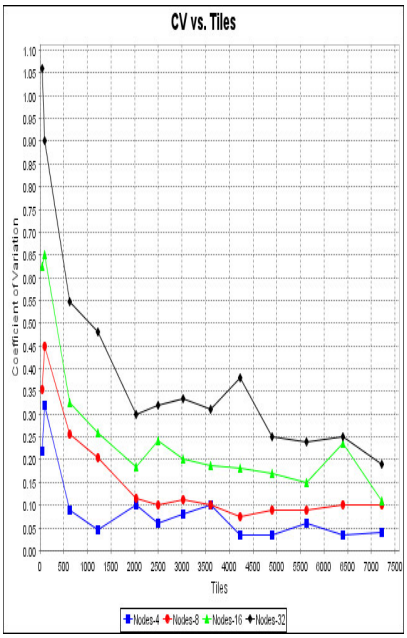

Fig. 4.2. CV vs. tiles for dataset2

Figures 4.1 and 4.2 show the Coefficient of Variation[2] (CV) of the number of features per node versus the number of tiles generated by the tiling declustering scheme. The x-axis represents the number of tiles and the y-axis represents the coefficient of variation. In both figures, each curve represents the graph generated for a specified set of compute nodes. Each curve represents the number of nodes the data was distributed across. Nodes-4 means that the result was taken when the number of compute nodes used was 4.

When the number of tiles is low, the CV is high reflecting the fact that data on different nodes varies greatly in size. As the number of tiles increases, the distribution becomes nearly uniform.

For a fixed number of tiles, the CV is less when fewer compute nodes are used. This is because the distribution of tiles that cover dense regions is better with a smaller number of nodes. Also, it is seen that the curves generated are not smooth. The irregularities on the curves were due to the limitations of the round-robin hashing function used. Indeed, when the number of columns was a multiple of the number of nodes, all tiles which have the same column number were stored on the same node; this is equivalent to having less tiles. These findings are in line with results obtained in [3] using different datasets.

[2] The ratio of the standard deviation and the mean.

Figures 5.1 and 5.2 show the effect of increasing the number of tiles (when using the tiling declustering scheme) on the replication of features. The x-axis represents the number of tiles, and the y-axis represents the percentage of replication. As the number of tiles increases, spatial features, which overlap many tiles, are replicated on many nodes. Therefore, as the number of tiles increases, the percentage of replication grows. The irregularities in the two curves are due to the same reason explained previously.

The two versions of clone join algorithms (CJAV1, CJAV2) are compared. The performance of these algorithms is tested using dataset1 and dataset2. The spatial join is performed on the *Hydrolin* and the *Rail* relations. From Figures 6.1 and 6.2, it is seen that for CJAV2, the time taken for the Join of *Hydrolin* and *Rail* starts decreasing as the number of tiles increases, due to a decrease in the coefficient of variation. The time then starts increasing as the number of tiles gets much higher due to an increase in replication. A similar trend was observed with different compute node configurations. The CJAV1 algorithm exhibits a similar behavior.

Next, CJAV1 and CJAV2 are compared using a varying number of compute nodes. The number of tiles was kept constant (3,600 and 8,100 for dataset1 & dataset2 respectively). The join query was tested under four different cluster configurations: 4, 8, 16, and 32 slave nodes for both dataset1 and dataset2. The datasets were also loaded onto one slave node (reference system) and tested.

From Table 6, it is observed that with dataset1, CJAV2 outperforms CJAV1. The performance gain becomes more pronounced as the number of compute nodes is increased. Table 7 shows the same experiment with the larger dataset, dataset2. A similar trend is observed except that the performance gain of CJAV2 over CJAV1 is higher with this larger dataset.

The R*-tree was built on the *hydrolin*, the larger of the two relations. This R*-tree structure was used to distribute the *rail* dataset. This approach was tested with various values for the R*-tree's fan out to find the best set of parameters. The same process is repeated with an R*-tree on the *rail,* the smaller of the two relations. It was observed that building an R*-tree on the smaller relation, *rail,* reduces replication of data. In addition, the declustering time is reduced by building an R*-tree on the smaller relation.

From Tables 8 & 9, the tiling declustering technique takes considerably more time than the proposed declustering method. The performance advantage of the proposed clustering method becomes more pronounced as the size of the data set is increased.

The spatial join query was tested under the following cluster configurations: 1, 4, 8, 16, and 32 slave nodes for each of the two datasets. As previously done, both algorithms were run under their most favorable parameters values.

It is noted from Tables 10 and 11 that RSDP outperforms CJAV2 for all cluster configurations, but as the number of compute nodes increases, the performance gap tends to decrease. It is also observed that round-robin performs better than the range hash function in almost all cases.

The RSDP algorithm is compared to CJAV2, as this algorithm performs better than CJAV1 for both dataset1 and dataset2. It should be noted that in the experiment, both algorithms (Tiling and the RSDP declustering methods) have been run under their most

favorable parameter values. Indeed, as shown previously, the tiling method performs best when 3,600 (8,100) tiles are used with dataset1 (dataset2). Similarly, the R*-tree based semi-dynamic approach performs best when *rail* is declustered using an R*-tree with a fan out M= 40 and 90 (280) for dataset1 (dataset2).

4 Conclusion

This paper proposes a new declustering strategy using a semi-dynamic approach. The proposed scheme builds an R*-tree on one of the two relations to be joined. The leaves of this R*-tree are mapped onto compute nodes. The features of the second relation are distributed statically using a tiling like approach. The leaves of the R*-tree built on the first relation are treated as tiles, and thus, the features of the second relation that overlap the leaves are stored on the nodes corresponding to the leaves. Based on this declustering strategy, a new R*-tree based semi-dynamic parallel join algorithm and two versions of the existing clone join algorithm were investigated. A comparative performance analysis of these algorithms was done using real datasets. These algorithms were implemented and run on a Beowulf cluster. The experimental results show that the proposed algorithm outperforms the clone join algorithm. Its performance is superior both in declustering time as well as in the execution time of the spatial Join query. Future work includes the design and implementation of *adaptive* parallel algorithms for both single scan and multiple scan spatial queries. These algorithms are expected to take into account *the size of the datasets*, the distribution of features, the main memory size of each slave node and other parameters (to be determined) to determine the best execution strategy. These algorithms are expected to form the basis of an *adaptive parallel query processor*.

References

1. PostGIS Manual : <http://postgis.refractions.net/documentation/>
2. PostgreSQL Online Documentation : <http://www.postgresql.org/docs/>
3. Patel, J.M, and Dewitt, D.J, "Clone Join and Shadow Join: Two Parallel Spatial Join Algorithms", Proceedings of the 8th ACM International Symposium on Advances in GIS, ACM Press, 2000, pp.54-61.
4. Brinkhoff, T., Hans-Peter Kriegal, and Seeger, B., "Parallel Processing of Spatial Joins Using R-trees," Proceeding of the 12th international Conference of Data Engineering. Washington – Brussels – Tokyo, Feb 1996, IEEE Computer Society pp. 258-265.
5. Koudas, N., Faloutsos, C., and Kamel,I., "Declustering Spatial Databases on a Multi-Computer Architecture," Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology, 1996. London, UK, pp. 592-614.
6. Zhou, X., Abel, D.J., Truffet, D., "Data Partitioning for Parallel Spatial Join Processing," Proceeding of the 5th Intl. Symp. on Large Spatial Databases, Germany, July 1997. 178-196.

7. Bureau of Transportation Statistics (BTS), "2002 National Transportation Atlas Data Shapefile Download Center Selected States:" http://www.bts.gov/programs/geographic_information_services/download_sites/ntad02/statedownloadform.html
8. Shekar, S., Sanjay, C., "Spatial Databases–A Tour," Prentice Hall, 2003, ISBN 013-017480-7.
9. Beckmann, N., Kreigel, and Seeger, B. "The R*-tree: An Efficient and Robust Access method for Points and rectangles." Proceedings of the ACM SIGMOD International Conference on Management of Data, 1990, pp. 322-331.
10. Abdelguerfi, M., and Wong, K.F., "Parallel Database Technique," Ed. manuscript, IEEE Computer Society Press, 1998. ISBN 0-8186-8398-8.
11. DeWitt, David J., and Jim Gray. "Parallel Database Systems: The future of High performance database processing," CACM, Jan 1992. Vol. 36, No. 6, pp.85-98,
12. Tan, K L., and J. X. Yu. "A Performance Study of Declustering Strategies for Parallel Spatial Databases," Proceedings of the 6th International Conference on Database and Expert Systems Applications, London, United Kingdom, September 1995, pp.157-166.

Table 6. Time (ms) vs. nodes - dataset1 (3,600 tiles)

Nodes/ Time	CJAV2	CJAV1
1	390.3	399.7
4	41.1	44.6
8	18.1	28.0
16	11.8	23.1
32	11.6	28.4

Table 7. Time (ms) vs. nodes - dataset2 (8,100 tiles)

Nodes/ Time	CJAV2	CJAV1
1	6,797.7	6,804.6
4	554.3	567.1
8	158	177
16	56	76
32	26	59

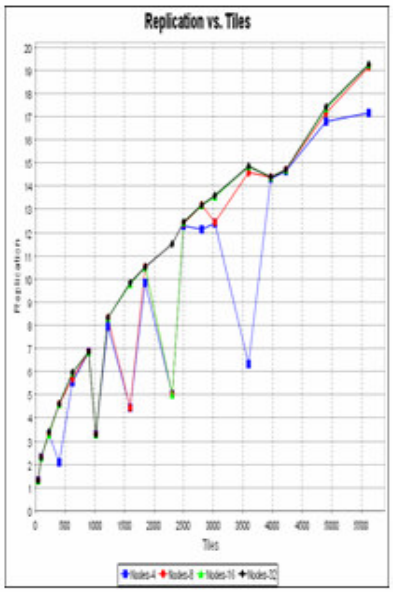

Fig. 5.1. Replication vs. tiles for dataset1

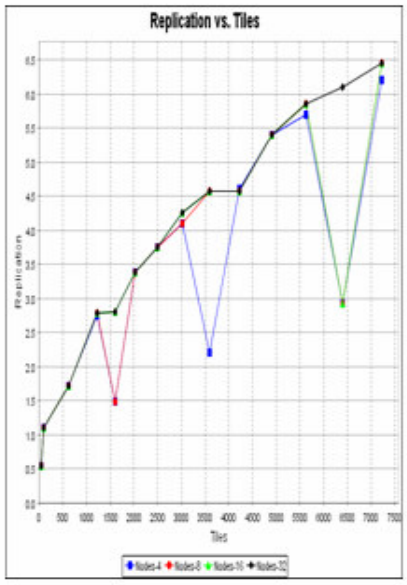

Fig. 5.2. Replication vs. tiles for dataset2

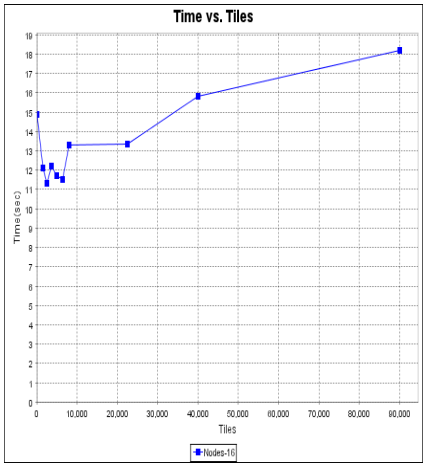

Fig. 6.1. Time vs. # of Tiles for dataset1 CJAV2 using 16 nodes

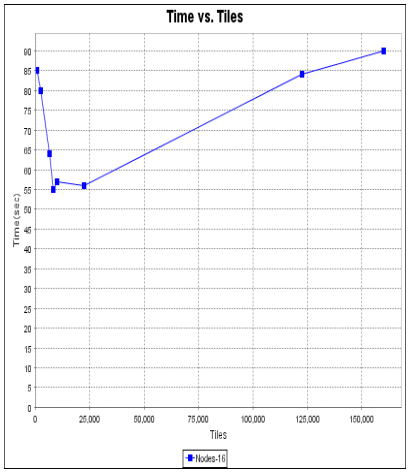

Fig. 6.2. Time vs. # of Tiles for dataset2- CJAV2 using 16 nodes

Table 8. Declustering time – dataset1

Time/ Variants	RSDP using round-robin		RSDP using range-hash		Tiling Technique Tiles=3,600
	M=40	M=90	M=40	M=90	
Time (sec)	116.7	128.7	151.4	152.3	816.8

Table 9. Declustering time –dataset2

Time/ Variants	RSDP using round-robin M=280	RSDP using range hash M=280	Tiling Technique Tile=8,100
Time (sec)	760.6	723.8	5,647.0

Table 10. RSDP vs. CJAV2 – dataset1

Nodes/ Variants	**RSDP using round-robin**	**RSDP using range hash**	**CJAV2 3,600 tiles**
	M=40	**M=40**	
1	99.1	99.1	390.3
4	19.2	17.7	41.1
8	10.9	12.4	18.1
16	11.7	11.8	11.8
32	11.6	13.6	11.6

Table 11. RSDP vs. CJAV2 – dataset2

Nodes/ Variants	**RSDP using round-robin M=280**	**RSDP using range hash M=280**	**CJAV2 8,100 tiles**
1	4,844.4	4,844.4	6,797.7
4	403.8	409.2	554.3
8	134.7	133.1	158.3
16	46.5	50.0	55.7
32	21.7	25.2	25.8

Acknowledgments. The work of Kevin Shaw and John Sample was partially funded by the Oceanographer of the Navy, N84.