1998

An efficient parallel algorithm for high dimensional similarity join

Khaled Alsabti
Syracuse University

Sanjay Ranka
Syracuse University

Vineet Singh
Hitachi America, Ltd.

An Efficient Parallel Algorithm for High Dimensional Similarity Join*

Khaled Alsabti
Syracuse University

Sanjay Ranka
University of Florida

Vineet Singh
Hitachi America, Ltd.

Abstract

Multidimensional similarity join finds pairs of multidimensional points that are within some small distance of each other. The ϵ-k-d-B tree has been proposed as a data structure that scales better as the number of dimensions increases compared to previous data structures. We present a cost model of the ϵ-k-d-B tree and use it to optimize the leaf size.

We present novel parallel algorithms for the similarity join using the ϵ-k-d-B tree. A load-balancing strategy based on equi-depth histograms is shown to work well for uniform or low-skew situations, whereas another based on weighted equi-depth histograms works far better for high-skew datasets. The latter strategy is only slightly slower than the former strategy for low skew datasets. Further, its cost is proportional to the overall cost of the similarity join.

1. Introduction

Multidimensional similarity join finds pairs of multidimensional points that are within some small distance of each other. Many important emerging applications require the number of dimensions to be quite large — possibly in the tens or hundreds, even thousands. Application domains include multimedia databases [7], medical databases [5], scientific databases [9], and time-series databases [1].

A pair of points is considered similar if the distance between them is less than ϵ for some distance metric, where ϵ is a user-defined parameter. In this paper, we use L_p-norm as the distance metric.

Several data structures have been proposed for multidimensional similarity join. Most of the well-known structures are not efficient for performing similarity joins on high-dimensional points because their time and/or space complexity increase rapidly with dimensionality [2].

*This work was supported by the Information Technology Lab (ITL) of Hitachi America, Ltd. while K. Alsabti and S. Ranka were visiting ITL. The authors can be reached at kaalsabt@top.cis.syr.edu, ranka@cise.ufl.edu, and vsingh@hitachi.com respectively.

The ϵ-k-d-B tree is a new multidimensional index structure that has been proposed for performing similarity join on high-dimensional points [2]. It has been shown to be considerably superior to other structures for performing the similarity join on high-dimensional points.

In this paper, we present a cost model for performing similarity join using the ϵ-k-d-B tree. We use our cost model to dynamically determine the leaf size threshold. This threshold has a significant effect on the cost of the similarity join operation. Our experimental results show that our model is reasonably effective. This cost model is also useful for its parallelization.

The parallelization of similarity join is difficult because of skewed amounts of work required in different parts of the tree. The amount of work required for different parts of the tree can be a superlinear function of the number of associated points. In this paper, we present a novel sampling-based scheme for the parallelization of this problem. Our scheme uses a subset of data to estimate the amounts of work required based on the cost model discussed earlier. A comparison with a simplistic scheme based on assigning approximately equal numbers of points to different numbers of processors shows that our scheme performs significantly better in the presence of data skews, even for 16 processors.

The rest of this paper is organized as follows. In Section 2, we describe how to determine the optimal or near optimal leaf size of the ϵ-k-d-B tree. In Section 3, we describe several parallel algorithms for computing the similarity join and a novel load-balancing strategy suitable for parallelizing problems which are sensitive to the presence of data skew and are not iterative in nature. Section 4 presents experimental results. Section 5 presents our conclusions.

2. The Sequential ϵ-k-d-B tree

In this section, we present our cost model for performing similarity join using the ϵ-k-d-B tree. We use our cost model to estimate the leaf size threshold.[1] The performance of the similarity join algorithm using the ϵ-k-d-B tree is strongly dependent on the size of the leaf node. The size

[1] Many of the details have been omitted because of the space limitations. For more details, the reader is referred to [4].

of the leaf node affects the depth of the tree as well as the number of join tests performed. A small leaf size will increase the depth of the tree which will result in a decrease in the number of join tests performed in most cases. On the other hand, it might increase the cost of traversing the tree.

The optimal value of the leaf size depends on the total number of points, the dimensionality of the points, the value of ϵ, and the distribution of the dataset. There are two approaches to determine the leaf size: *Static* and *Dynamic*. In the static approach, the leaf size is determined statically regardless of any available information about the problem instance. Whereas in the dynamic approach, the leaf size is determined dynamically using the available information about the problem instance.

We have developed a cost model for optimizing the leaf size under the assumption of uniform distribution for the dataset. Even when the dataset distribution is not uniform, the cost model can be significantly better than the static approach to determine leaf size. The cost C of performing the join algorithm can be modeled based on the parameters of the problem instance as follows:

$C = \frac{\text{number of leaf nodes}}{2}$[number of visited leaf nodes per a leaf$+k\times$number of visited leaf nodes per a leaf$\times 2\epsilon\times$ (size of leaf node)^2 $\times$ number of dimensions]
The first term above refers to the traversal cost, and the second term refers to the join cost.

We use the following notation: b is the branch factor ($= \frac{1}{\epsilon}$), d is the depth of the tree, n is the number of points, D is the dimensionality of the points, and k is a positive constant. Since we are assuming that the data points follow the uniform distribution, each leaf node is expected to have $\frac{n}{\text{number of leaf nodes}}$ points. The leaf size $LS \in [\frac{n}{b^d}, \frac{n}{b^{d-1}} + 1]$. In the following, we derive an optimal or near-optimal value of the depth of tree, d, with n D-points. From the characteristics of the ϵ-k-d-B tree, the upper bound of the number of visited leaf nodes during the course of joining one leaf node is 3^d [2]. Thus, the cost formula can be simplified as follows:

$$C \simeq \frac{b^d}{2}[3^d + k3^d 2\frac{1}{b}(\frac{n}{b^d})^2 D] \quad (1)$$

To find the optimal value of d, we need to differentiate C with respect to d and then equate the result to zero and solve it for d. After simplifications, we obtain the following:

$$d \simeq \frac{1}{2}\log_b(-\frac{\ln\frac{3}{b}}{\ln 3b}) + \log_b n + \frac{1}{2}\log_b D - 0.5 \quad (2)$$

We have conducted several experiments on datasets with different parameters. These experiments (not reported here due to space limitations) show that our cost model is reasonably effective and it is significantly better than using an arbitrarily fixed leaf size.

3. The Parallel Similarity Join

In this section, we describe the parallelization of our algorithms on coarse-grained shared nothing (message passing) parallel machines.

We present two algorithms for parallelization of the similarity join algorithm. Both of these algorithms consist of four main phases. In the partitioning phase (Phase 1), the space is partitioned into disjoint regions. These regions represent the global part of the parallel ϵ-k-d-B tree. Ideally, the regions should be assigned to processors such that the load across them is balanced. The local ϵ-k-d-B tree is built in the second phase. Each processor requires non-local data to perform the computation of similarity join. In phase 3, each processor determines the processors with which it needs to exchange some data points. It also computes the subregions it needs to communicate. Additional data structures are required for this computation. The join algorithm is performed on the local tree and the tree consisting of non-local regions to obtain all pairs of points within ϵ distance (Phase 4). The computation is performed such that duplicate pairs (e.g., (a,b) and (b,a)) are not generated.

Data Partitioning We have developed two data partitioning strategies for partitioning the data across several processors assuming that the n D-dimensional points are assigned among p processors such that each processor has approximately $\frac{n}{p}$ points.

The global part of the tree is built by assigning disjoint subsets of points to each processor. Each subset of points corresponds to a region in the space. To achieve this, the space is partitioned into p regions. Each of these p regions is assigned to a unique processor.

PQ Algorithm This algorithm, Partitioning based on Quantiling (PQ), uses the entire dataset to partition the space into p disjoint regions. The PQ algorithm uses d ($d \geq 1$) dimensions for partitioning the space. An equi-depth histogram is generated with z bins at every level. This is done recursively for d levels. Each bin is assigned to z disjoint subsets of processors. The equi-depth histogram is generated by using a quantiling algorithm, OPAQ, which has been proposed in [3]. OPAQ generates a good bounded approximation of an equi-depth histogram using one pass over the entire data.

After estimating the quantiles, the points are redistributed among the sub-regions such that each processor belonging to a sub-region will receive approximately the same number of points as the other processors in the same sub-region. In each of the following iterations of the algorithm, we perform the same process in each sub-region using some common unused dimension. After iteration i, there will be $p^{\frac{i}{d}}$ sub-regions, each having $p^{\frac{d-i}{d}}$ processors.

The PQ algorithm can be approximated by random sampling. This random sample is used by the PQ algorithm to build the global part of the tree. The global part of the tree determines the processor regions. After determining the processor regions, the entire set of data points is redistributed once only among the processors. We call this version of the PQ algorithm the PQ^{Est} algorithm. The experiments reported in this paper have used PQ.

PW Algorithm Our experimental results show that the most computation intensive part of the algorithm is performing the join tests. The PQ algorithm uses the number of points as a load metric. This may lead to poor load balancing in the case that the dataset has a skewed distribution. For example, PQ algorithm might partition the space among the processors such that each sub-region has approximately equal number of points and all or most of the work required for generating the join output is located in one subpartition; say the first subpartition. In that case, processor p_1 will perform almost all the required work and the speedup of the algorithm will be close to one, regardless of the total number of processors. To overcome this problem, we have developed the PW algorithm which uses the number of join tests as a load metric.

The main idea of the PW algorithm is that it partitions the space into p regions such that the amount of work (number of join tests+ traversal costs) associated with points corresponding to each region is approximately equal. Clearly, this is a chicken and egg problem. Computing the workload corresponding to a given point may require performing the entire similarity join (which is what we are trying to parallelize). We achieve this by performing the computation (in parallel) on a small sample of size s using the PQ partitioning strategy. A high-level description of the partition phase is presented in Figure 1. We only compute the number of join tests for each point. This is used for determining a weight of each point in the sample tree. The weight of a point should be proportional to work required in the region around that point. This information is used to guide the decomposition of the entire dataset. A region tree is built using only the sample points and weights associated with them. The second stage redistributes the points among the processors using the region tree.

In the PQ algorithm, the points were implicitly assigned equal weights. The PW algorithm assigns weights to points based on two factors: the estimated number of required join tests for each point, and the cost of traversing the tree for each point. Formally, we use the following function to assign weights to points: $f(pt) = rD + k3^{Depth}$, where D is the dimensionality of the points, k is a positive constant, r is the estimated number of join tests for point pt, and $Depth$ is the estimated depth of the ϵ-k-d-B tree under the assumption that the dataset has uniform distribution.

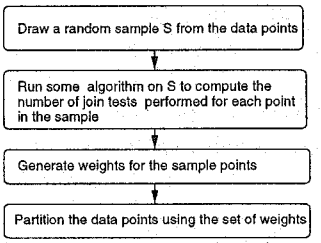

Figure 1. An overview of the PW algorithm

k is determined empirically. Note that 3^{Depth} is the maximum number of visited nodes during the course of the join algorithm (see Section 2).

The partitioning algorithm is similar to the one in the PQ algorithm except that the weights of the points are incorporated in finding the region boundaries.

After partitioning the space into p regions, the entire dataset is redistributed among the processors using the region boundaries. The size of the sample affects the accuracy of load balance as well as the overhead of the partitioning algorithm.

The Local Tree Phase In this phase, each processor builds an ϵ-k-d-B tree using only the local data points.

The Regions Phase Each processor requires non-local data to perform the computation of similarity join. In this phase, each processor determines the processors it needs to exchange this information. Let $Inter_i$ represent the set of processors with which processor i needs to communicate. Each processor i uses the global part of the tree to determine the $Inter_i$ set.

Let processor j belongs to $Inter_i$ set. Processors i and j need to determine the intersected regions between them and then send to each other some of the points which belong to the intersected regions. Our approach to find the intersected region between processors i and j works as follows. The local ϵ-k-d-B trees of the processors are used to identify the intersected regions. One parameter of this approach is a level number l. For this level l, each processor i (j) generates two lists represented by $SimpleLevel_i^l$ ($SimpleLevel_j^l$) and $Level_i^l$ ($Level_j^l$). These lists represent the lth level of the local tree of processor i (j). An entry r of $SimpleLevel_i^l$ represents the r^{th} sub-tree R. The entry is 0 or 1 depending on whether the sub-tree is empty or not respectively. However, an entry r of the $Level_i^l$ list represents the r^{th} sub-tree R and all the "join-able" sub-trees which are needed to be joined with R at the l^{th} level of the tree. A value of 1 is assigned to this entry if either sub-tree R or any of its joinable subtrees are non-empty. Otherwise, a value of 0 is assigned. Note the sizes of the $SimpleLevel_i^l$ and $Level_i^l$ lists are $(\frac{1}{\epsilon})^l$. The value of the l parameter, which is

used in building these lists, affects the performance of the overall algorithm. We intend to determine the value of l empirically.

The points of the subtree r of processor i (j) are part of the intersected region with processor j (i) if $SimpleLevel_i^l[r] = 1$ ($SimpleLevel_j^l[r] = 1$) and $Level_j^l[r] = 1$ ($Level_i^l[r] = 1$). Processors i and j exchange their *Level* lists to determine their intersected regions locally.

For potential result points (a, b) such that a and b belong to the same processor, we assume that the computation is performed by the processor which contains these two points. However, the situation is different for potential result points (a, b) such that a and b belong to different processors i and j respectively. The computation can be performed on processor i or j. Clearly, it is possible to perform this computation on a processor different than i and j. However, this option has not been considered in this paper due to our belief that this will result in extra communication without significantly affecting the load balance.

For skewed datasets, an inappropriate assignment of computation for such points can result in substantial load imbalance, even assuming that the computations that require only local data points are well balanced. Processors i and j need to exchange some of the points of the intersected regions such that a good load-balance is achieved. This problem is an optimization problem where there are p processors. Each of them has an intersected region with the other processors. Let n_{ij} be the number of points which are local to processor i and belong to the intersected regions between processors i and j. We need to assign some of the n_{ij} points to processor j and some of the n_{ji} points to processor i. We have developed local and global assignment strategies for assignment of these computations. These differ in whether they use global or local information. The strategy based on global information (n_{ij} $\forall 1 \leq i, j \leq p$ and $i \neq j$) uses information about all the processors for performing this assignment. The strategy based on local information uses only information gathered by processor i from its $inter_i$ list of processors (i.e., it only uses n_{ij} and n_{ji}).

Using a global assignment algorithm, one can potentially obtain a better load balance. However, this may not be preferable due to potentially higher costs and poor scalability. In the particular global algorithm used in this paper, a series of decisions are made iteratively to assign the work of intersected regions among the processors. In our local assignment method, processors i and j divide the work of their intersected region into two halves such that each processor will perform half of the work.

The Join Phase We need to perform the join on the local trees using the sequential join algorithms [4]. In performing the join for the points of the intersected regions, we ensure that we do not generate duplicates. The assignment methods (of Phase 3) guarantee that no duplicates are generated.

4. Experimental Results

We implemented the different schemes defined in the previous Section on an IBM SP-2 with 16 processors using MPI standard [6]. The clock speed of the processors is 66.7 MHz, the memory size is 128 MB per processor, and the operating system is AIX version 4.

We studied the following four datasets which were generated synthetically.

1. Uniform Distribution (DS1): The point values along each dimension were generated randomly in the range $[0, 2]$.
2. Gaussian Distribution (DS2): Along each dimension, the values were generated according to a Gaussian distribution with mean of 1.0 and standard deviation of 0.25. Note, the range of the points is $[0, 2]$.
3. Mixed Distribution 1 (DS3): The range ($[0, 2]$) along each dimension is divided into two parts: $[0, 0.5]$ and $(0.5, 2]$. 25% of the points were generated in the first range $[0, 0.5]$ with Gaussian distribution (mean = 0.25 and standard deviation = 0.0625) and the rest of the points were generated randomly with a uniform distribution in the other range $(0.5, 2]$.
4. Mixed Distribution 2 (DS4): This dataset is similar to Mixed Distribution 1. 6.25% of the points were generated in $[0, 0.125]$ with Gaussian distribution (mean = 0.0625 and standard deviation = 0.015625) and the rest of the points were generated randomly with a uniform distribution in $(0.125, 2]$.

We have performed a set of experiments to determine appropriate values for the important parameters of the algorithms. The results of these experiments are: the value of k (which is used in the weight function) is set to one, the sample size (used in the PW algorithm) is set to 10%, the value of level l is set to two, and the number of dimensions used for partitioning is set to two. The choice of number of dimensions for partitioning is data-dependent. However, we expect that using two dimensions will be better for larger numbers of processors.

Table 1 presents the time requirements of PW and PQ algorithms using local and global algorithms for assigning the intersected regions. These results are representative of results for other values of the parameters and show that the local assignment strategy is superior to the global assignment strategy. Since the local strategy is more scalable, it should be the strategy of choice for larger numbers of processors. The rest of the experiments presented in this Sec-

tion assume that the local algorithm is used for assigning the intersected regions.

Data Set	PW		PQ	
	Local	Global	Local	Global
DS1	2.36	2.42	2.64	2.31
DS2	8.39	9.12	9.87	10.11
DS3	82.21	121.031	277.57	352.70
DS4	50.36	88.87	572.34	564.74

Table 1. The total execution time (secs) on 16 processors using different assignment strategies. Total number of points is 256k 12-d points. ϵ is 0.1.

Figure 2 gives the speedups for PQ and PW for the four datasets on 4 and 16 processors. These results clearly demonstrate the efficacy of weighted quantiling. For 16 processors, PW is significantly better for all data distributions. For 4 processors, PW is comparable to PQ for uniform[2] and Gaussian distributions, and PW is significantly better for the two mixed distributions. In fact, for mixed distribution 2 (DS4), PQ resulted in no speedup for both 4 and 16 processors. This is because almost all the work has been performed by one of the processors. We conclude that weighted quantiling is more robust as compared to simple quantiling in the presence of data skew and does not deteriorate in the absence of data skew. These are important considerations for any practical algorithm.

5. Conclusions

We have presented a cost model for performing the similarity join using the ϵ-k-d-B tree and used it to optimize the leaf size. This new leaf size is shown to be better in most situations compared to previous work that used a constant leaf size. We have also presented novel parallel algorithms for the similarity join. A load-balancing strategy based on weighted equi-depth histograms was shown to be superior to the one based on unweighted equi-depth histograms. The weights for the weighted strategy were based on the same cost model that was used to determine optimal leaf sizes. Further, the cost of the weighted strategy is proportional to the overall cost of the similarity join process.

We are not aware of any previous work on the parallelization of high-dimensional similarity join except [8], which was done concurrently. A reasonable comparison will require some detailed experimental and/or theoretical analysis.

[2]Both algorithms have obtained a super speed-up for the uniform distribution. We believe that the cache effects as well as the low cost of the parallelization overhead (for the uniform case) are the cause of this phenomena.

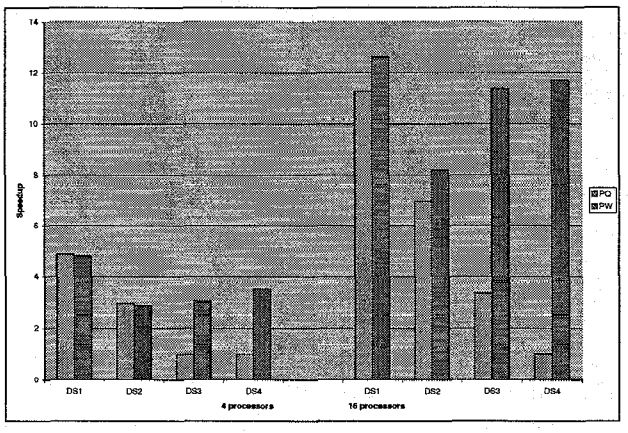

Figure 2. Speedup for the four datasets of size 256k 12-d points. ϵ is 0.1.

References

[1] R. Agrawal, K. Lin, H. S. Sawhney, and K. Shim. Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases. *In Proc. of VLDB*, 1995.

[2] R. Agrawal, K. Shim, and R. Srikant. A Fast Algorithm for high-dimensional Similarity Joins. *In Proc. of the 13th Int'l Conference on Data Engineering*, 1997.

[3] K. Alsabti, S. Ranka, and V. Singh. A One-Pass Algorithm for Accurately Estimating Quantiles for Disk-Resident Data. *In Proc. of VLDB*, 1997.

[4] K. Alsabti, S. Ranka, and V. Singh. An Efficient Parallel Algorithm for High Dimensional Similarity Join. *http://www.cise.ufl.edu/~ ranka/*, 1997.

[5] M. Arya et al. Qbism: a Prototype 3-d Medical Image Database System. *IEEE Data Engineering Bulletin, 16(1):38-42*, 1993.

[6] M. P. I. Forum. *MPI-2: Extensions to the Message-Passing Interface*. http://www.mcs.anl.gov/mpi/, 1995.

[7] W. Niblack et al. The qbic Project: Querying Images by Content Using Color, Texture and Shapes. *In SPIE 1993 Int'l Symposium on Electronic Imaging: Science and Technology*, 1993.

[8] J. Shafer and R. Agrawal. Parallel Algorithms for High-Dimensional Proximity Joins. *In Proc. of VLDB*, 1997.

[9] D. Vassiliadis. The Input-State Space Approach to the Prediction of Auroral Geomagnetic Activity from Solar Wind Variables. *In Int'l Workshop on Applications of AI in Solar Terrestrial Physics*, 1993.