In Proc. 10th Int. Conf. on Database Systems for Advanced Applications (DASFAA 2005), Beijing, China, 2005.

Distributed Intersection Join of Complex Interval Sequences

Hans-Peter Kriegel, Peter Kunath, Martin Pfeifle, Matthias Renz
University of Munich, Germany
{kriegel, kunath, pfeifle, renz}@dbs.ifi.lmu.de

Abstract. In many different application areas, e.g. space observation systems or engineering systems of world-wide operating companies, there is a need for an efficient distributed intersection join in order to extract new and global knowledge. A solution for carrying out a global intersection join is to transmit all distributed information from the clients to a central server leading to high transfer cost. In this paper, we present a new distributed intersection join for interval sequences of high-cardinality which tries to minimize these transmission cost. Our approach is based on a suitable probability model for interval intersections which is used on the server as well as on the various clients. On the client sites, we group intervals together based on this probability model. These locally created approximations are sent to the server. The server ranks all intersecting approximations according to our probability model. As not all approximations have to be refined in order to decide whether two objects intersect, we fetch the exact information of the most promising approximations first. This strategy helps to cut down the transmission cost considerably which is proven by our experimental evaluation based on synthetic and real-world test data sets.

Keywords

Distributed intersection join, probability model, interval sequences.

1 Introduction

After two decades of temporal and spatial index research, the efficient management of one- and multi-dimensional extended objects has become an enabling technology for many novel database applications. The interval, or, more generally, the sequence of intervals, are basic datatypes for temporal and spatial data. Interval sequences are used to handle finite domain constraints [10] or to represent periods on transactions or valid time dimensions [11]. Typical applications of one-dimensional interval sequences include the temporal tracing of user activity for service providers. In general, any time series may be aggregated to an interval sequence, such as periods of "high" stock prices for technical chart analysis (cf. Figure 1).

Further examples of interval sequences residing on different, independently working computers which are connected to each other via local or wide area networks (LANs or WANs) comprise distributed mobile networks, sensor networks or vehicle manufacturers, where the development agencies are located at different places, distributed all over the world. For instance, international companies such as DaimlerChrysler have some development agencies which are located in Europe, some in Asia, and some located in the US. When applied to space-filling curves, interval sequences naturally represent

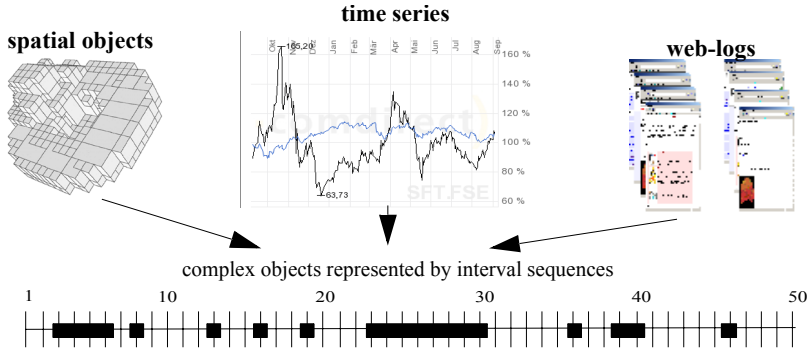

Figure 1. Interval sequences.

spatially extended objects with even intricate shapes (cf. Figure 1). By expressing spatial region queries as interval sequence intersections, vital operations for the digital mock-up of vehicles and airplanes [5], haptic simulations in virtual product environments [8] or engineering data management can be supported. In these areas as well as in the areas of two-dimensional GIS and environmental information systems [7] the locally collected data can only, with great difficulty, be transmitted to a central site to be joined centrally there. Meeting the need of all these application ranges, we will present a distributed interval intersection join in this paper which extracts global knowledge while taking limited bandwidth and security aspects into account.

The remainder of this paper is organized as follows. In Section 2, we present the related work on distributed interval intersection joins. In Section 3, we shortly sketch our general idea, followed by the presentation of the basic definitions and theorems in Section 4. In Section 5, we show how to group different intervals together to coarser approximations on the client site in order to reduce the overall transmission cost. After having transmitted these approximations to the server, we present in Section 6, the server-side join algorithm trying to avoid as many as possible further client accesses for fetching the exact interval sequence objects. In Section 7, we will present convincing experimental results demonstrating the superiority w.r.t. low transmission cost of our new algorithm compared to less sophisticated algorithms. We will close this paper in Section 8 with a short summary and a few remarks on future work.

2 Related Work

Several different approaches to provide efficient interval joins already exist in the literature, especially in the field of temporal applications [3]. For instance, Seidl et al. proposed an interval intersection join based on the Relational Interval Tree which can easily be implemented on top of any relational database system [2]. Furthermore, there exist specialized index structures suitable for detecting intersecting interval sequences [6] which can be used as foundation for an index-based nested loop join.

Similarly, considerable work has been done in the area of distributed data management [9], for instance in the area of Distributed Data Mining (DDM) [4]. Generally, distributed databases constitute a very important and emerging research area which crucially depends on efficient query processing.

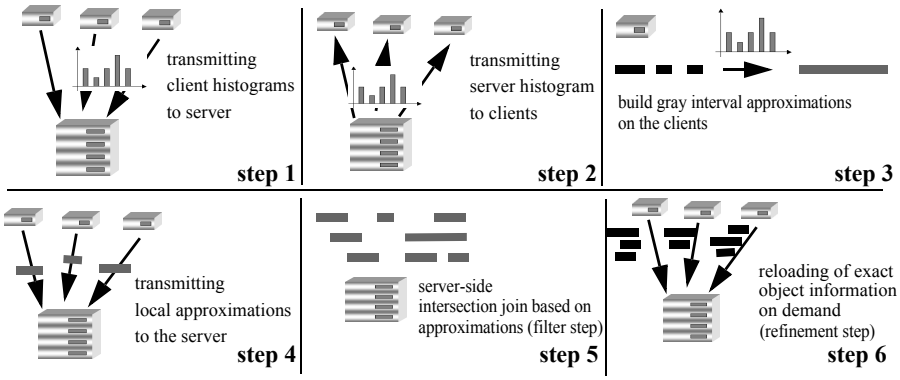

Figure 2. Distributed Intersection Join on Interval Sequences.

Unfortunately, to the best of our knowledge, there has been no work published which brings the two independent research areas of "distributed databases" and "join processing of interval sequences" together.

3 General Idea

The goal of this paper is to present a distributed algorithm which detects intersecting interval sequences residing on different local clients. Note that determining pairs of intersecting interval sequences located at the same local site is a rather straightforward task which can be handled independently by the corresponding local clients. These locally determined result sets can easily be combined with the global result set determined by the distributed intersection join presented in this paper. In this section, we shortly sketch the complete distributed intersection join process (cf. Figure 2).

At first all clients collect statistical information reflecting the interval distributions of the data residing at their own local site. Then, the clients send this statistical information to the server (step 1). At the server site the local client statistics are merged into a global statistic reflecting the interval distribution of all local clients. This global statistic is sent back to each client (step 2). Each client groups "black" intervals belonging to the same interval sequence together to coarser approximations called "gray" intervals (step 3). This grouping process is decisively based on the data distribution of the join partners residing on the other local clients which is reflected by the global statistic minus the own local client statistic. The resulting statistic is not only used for the grouping process but also for a fast filter step on the client sites. If by means of this statistic a global intersection of a gray interval cannot be ruled out, the hull of the gray interval along with additional aggregated information, i.e. the density of the gray interval and the number of bytes required for sending the corresponding (compressed) exact information, is sent to the server (step 4). The server detects all intersecting gray intervals based on their hulls (step 5). Based on the intersection length of the hulls and the density of the gray intervals the server computes a probability that not only the gray intervals intersect but also the corresponding black intervals. The pairs of gray intervals with intersecting hulls are ranked ascendingly according to a combination of the determined probability value and the transmission cost of the exact information. The server iteratively refines the top-listed pairs by fetching the exact information, i.e. the (compressed) black inter-

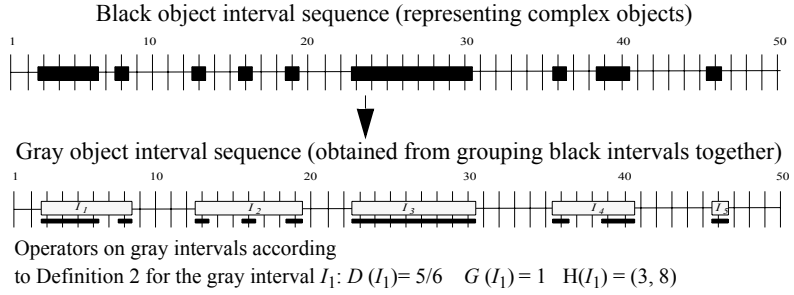

Figure 3. Gray object interval sequence.

vals, belonging to the corresponding gray intervals from the local clients (step 6). Gray interval pairs which belong to object pairs already known to intersect do not have to be refined. Thereby, we can enormously save on the overall transmission cost.

4 (Gray) Interval Sequences

In this section, we formally introduce interval sequence objects. Furthermore, we propose an intersection probability model, which is used for the client-side grouping approach (cf. Section 5) as well as for the server side join processing (cf. Section 6). We start with the definition of some notions.

4.1 Definitions

Interval sequences representing complex objects often consist of very short intervals connected by short gaps (cf. Figure 3). For instance, in the case of high resolution linearized spatial objects the interval sequences may contain several hundred thousands of small intervals per object. When the server request the objects from the clients, the huge amount of interval data would lead to high transmission cost due to a low transfer rate of the network connection. In order to overcome this obstacle, it seems promising to pass over some "small" gaps and approximate the interval sequence by much less intervals. In this paper, we confine ourselves to integer boundary values for intervals, as in all investigated domains, e.g. for time series, stock charts, or spatially extended objects linearized via space-filling curves, only integers are used due to the use of a finite resolution.

Definition 1 (*gray interval sequence objects*). Let *id* be an *object identifier* and $W = \{(l, u) \in IN^2, l \le u\}$ be the domain of intervals which we call *black* intervals throughout this paper. Furthermore, let $b_1 = (l_1, u_1), \ldots, b_n = (l_n, u_n) \in W$ be a sequence of intervals with $u_i + 1 < l_{i+1}$ for all $i \in \{1, \ldots, n-1\}$. Moreover, let $m \le n$ and let $i_0, i_1, i_2, \ldots, i_m \in IN$ such that $0 = i_0 < i_1 < i_2 < \ldots < i_m = n$ holds. Then, we call $O_{gray} = (id, \langle \langle b_{i_0+1},\ldots,b_{i_1} \rangle, \langle b_{i_1+1},\ldots,b_{i_2} \rangle, \ldots, \langle b_{i_{m-1}+1},\ldots,b_{i_m} \rangle \rangle)$ a *gray interval sequence object* of cardinality *m*. If *m* equals *n*, we denote O_{gray} also as a *black interval sequence object* O_{black}. We call each of the $j = 1, \ldots, m$ groups $\langle b_{i_{j-1}+1},\ldots,b_{i_j} \rangle$ of O_{gray} a *gray interval* I_{gray}. If $i_{j-1}+1$ equals i_j, we denote I_{gray} also as a *black interval* I_{black}.

In the next definition, we introduce a few useful operators on *gray intervals* which we will use frequently throughout the remainder of this paper.

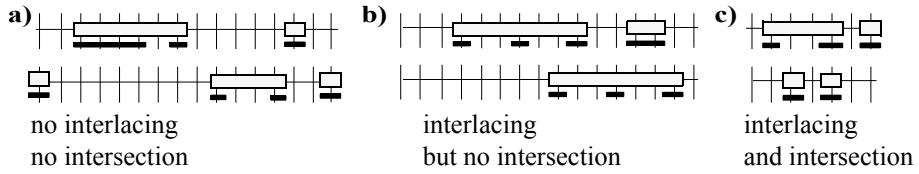

Figure 4. Gray object interval sequences.
a) non interlacing, **b)** interlacing but no intersection, **c)** intersection

Definition 2 (*operators on gray intervals*). For any gray interval $I_{gray} = \langle (l_r, u_r), \ldots, (l_s, u_s) \rangle$ we define the following operators:

Density: $D(I_{gray}) = \sum_{i=r \ldots s} (u_i - l_i + 1) / (u_s - l_r + 1)$

Gap: $G(I_{gray}) = \begin{cases} 0 & r = s \\ \max\{1 + l_i - u_{i-1},\ i = r+1, \ldots, s\} & \text{else} \end{cases}$

Hull: $H(I_{gray}) = (l_r, u_s)$

Figure 3 exemplarily demonstrates the values of these operators for a gray interval I_1. In the following, we define intersect predicates for *intervals*, *interval sequences* and *gray intervals*:

Definition 3 (object intersection). Let $W = \{(l, u) \in IN^2, l \le u\}$ be the domain of intervals, and let $I = \langle b_1, \ldots, b_n \rangle$ and $I' = \langle b'_1, \ldots, b'_{n'} \rangle$ be two gray intervals. Furthermore, let $O = (id, \langle I_1, I_2, \ldots, I_m \rangle)$ and $O' = (id', \langle I'_1, I'_2, \ldots, I'_{m'} \rangle)$ be two gray interval sequence objects. Then, the notions *intersect* and *interlace* are defined in the following way (cf. Figure 4):

1a. Two intervals, $b = (l, u)$ and $b' = (l', u')$ *intersect* if $l \le u'$ and $l' \le u$.
1b. The intersection length $intersect_{length}((l, u), (l', u'))$ of two intervals is equal to $max(0, min(u, u') - max(l, l')+1)$.
2a. Two gray intervals I and I' *intersect* if for any $i \in \{1, \ldots, n\}, j \in \{1, \ldots, n'\}$ the black intervals b_i and b'_j intersect.
2b. Two gray intervals I and I' *interlace*, if their hulls, $H(I)$ and $H(I')$ intersect.
3a. Two objects O and O' *intersect* if for any $i \in \{1, \ldots, m\}, j \in \{1, \ldots, m'\}$, the gray intervals I_i and I'_j intersect.
3b. Two objects O and O' *interlace* if for any $i \in \{1, \ldots, m\}, j \in \{1, \ldots, m'\}$, the gray intervals I_i and I'_j interlace.

4.2 Intersection Detection

In this section, we first present two rather obvious lemmas which state whether two gray intervals intersect or not, based on relatively little information. The first lemma can be used as filter for detecting intersecting interval sequence objects (cf. Figure 4a).

Lemma 1 (non-intersecting gray intervals). Let $I = \langle b_1, \ldots, b_n \rangle$ and $I' = \langle b'_1, \ldots, b'_{n'} \rangle$ be two gray intervals. Then the following statement holds:

$$\neg\, interlace(I, I') \Rightarrow \neg\, intersect(I, I')$$

Proof. First, $\neg$ *interlace*$(I, I') \Rightarrow \neg$ *intersect*(I, I') is equivalent to *intersect*$(I, I') \Rightarrow$ *interlace*(I, I'). Then, *intersect*$(I, I') \Rightarrow \exists\, (b_i, b'_j) \in \{\{b_1, .., b_n\} \times \{b'_1, .., b'_{n'}\}\}$: *intersect*$(b_i, b'_j)$ = *true*. Let $b_i = (l_i, u_i)$, $b_j' = (l_j', u_j')$, $H(I)=(l, u)$, and $H(I')=(l', u')$, then $l \le l_i \le u_j' \le u'$ and $l' \le l'_j \le u_i \le u$ holds which proves that I and I' interlace (cf. case 1a and 2b in Definition 3).n

Let us note that the we cannot pinpoint any intersecting interval sequence objects by means of Lemma 1, as *interlace*$(I, I') \Rightarrow$ *intersect*(I, I') does not hold (cf. Figure 4b). Thus, a refined evaluation of the intersect predicate is necessary when two gray intervals interlace. Only in the rare case where both gray intervals have maximum density, we can abstain from this refinement step.

Lemma 2 (*intersecting gray intervals*). Let $I = \langle b_1, \ldots, b_n \rangle$ and $I' = \langle b'_1, \ldots, b'_{n'} \rangle$ be two gray intervals. Then the following statement holds:

$$(D(I) = 1 \wedge D(I') = 1 \wedge \textit{interlace}(I, I')) \Rightarrow \textit{intersect}(I, I')$$

Proof. According to Definition 1 and 2: $D(I) = 1 \Rightarrow n = 1$ and $D(I') = 1 \Rightarrow n' = 1$. According to Definition 3: *interlace*$(I, I') \Rightarrow$ *intersect*$(b_1, b'_1) \Rightarrow$ *intersect*(I, I'). n

Lemma 2 shows that we can sometimes pinpoint whether two gray interval pairs intersect based on relatively little information. Unfortunately, as in most cases the precondition of Lemma 2 does not hold, we will not be able to apply it very often. Nevertheless, it is still helpful if we can predict how probable an intersection of interlacing gray intervals might be.

4.3 Intersection Probability

The probability model introduced in this section is easy to compute and will be applied in various different forms throughout our approach. The model is equal to the *coin-toss experiment*, i.e. it is a *Bernoulli experiment*. It assumes that the intervals and gaps covered by a gray interval are equally distributed.

Theorem 1 (*intersection probability*). Let I and I' be two gray intervals with densities $D = D(I)$ and $D' = D(I')$. Furthermore, let $L = \textit{intersect}_{\textit{length}}(H(I), H(I'))$. Then the probability $P(I, I')$ that the two gray intervals I and I' intersect is equal to:

$$P(I, I') = 1 - (1 - D \cdot D')^L$$

Proof. Let x be one of the points in the interlacing area. Obviously, the probability that this point is covered by an interval contained in I is equal to the density D. Subsequently, the probability that two intervals I and I' intersect at the point x is $P_x = D \cdot D'$. The probability, that either x or another point $y \neq x$ is covered by intervals from I and I' is $P_{\{x,y\}} = D \cdot D' + (1 - D \cdot D') \cdot D \cdot D'$. As we assume that the interval bounds are mapped to discrete integer values, the probability that I and I' share at least one point can be computed as follows:

$$P(I, I') = \sum_{i=0}^{L-1} D \cdot D' \cdot (1 - D \cdot D')^i = D \cdot D' \cdot \frac{1 - (1 - D \cdot D')^L}{1 - (1 - D \cdot D')} = 1 - (1 - D \cdot D')^L \quad \text{n}$$

Note that Lemma 1 and 2 can be derived from the above theorem by setting the interlacing length L to 0 (Lemma 1) or setting D and D' to 1 (Lemma 2). Similar to the above reasoning, we are going to derive the probability that one gray interval I_0 intersects at least one of n other gray intervals $I_1, .., I_n$.

Theorem 2 (*combined intersection probability*). Let I_0 be a gray interval that intersects with n other gray intervals I_i, $i \in 1..n$, with a probability $P(I_0, I_i)$. Then, the total probability $P(I_0)$ that I_0 intersects with at least one of the other gray intervals can be computed by

$$P(I_0) = 1 - \prod_{i=1}^{n} (1 - P(I_0, I_i))$$

Proof. The probability $P'(I_0)$ that none of the gray intervals $I_1, .., I_n$ intersect with I_0 is $P'(I_0) = (1\text{-}P(I_0, I_1)) \cdot ... \cdot (1\text{-}P(I_0, I_n))$. Consequently, the total probability $P(I_0)$ that I_0 intersects with at least one of the other gray intervals is $P(I_0) = 1\text{-}P'(I_0)$. n

5 Client-Side Approximation of Interval Sequences

The central question is how to group the interval sequence of a client object into gray intervals serving as suitable object approximations. In this section, we first introduce probability histograms which are used to estimate the intersection probability of gray intervals. Secondly, we present our cost model which takes the probability histograms as well as the transmission cost into account. Finally, we present a cost-based grouping algorithm which aims at minimizing the overall transmission cost.

5.1 Estimation of the Intersection Probability

We use simple statistics of the interval sequence objects to estimate the probability $P(I_{gray})$ that any gray interval I_{gray} intersects with at least one other gray interval located on a different client. In order to cope with arbitrary interval distributions, histograms can be employed to capture the data characteristics at any desired resolution. The expected intersection probability $P(I_{gray})$ can be determined by using an appropriate intersection probability histogram which reflects aggregated information over all interval sequence objects distributed over all local clients.

Definition 4 (*intersection probability histogram*). Let $IB = [0, max \in IN]$ be a domain of interval bounds. Let the natural number $\nu \in IN$ denote the *resolution*, and $\beta_\nu = max/\nu$ be the corresponding *bucket size* of the histogram. Let $b_{i,\nu} = [1 + (i - 1)\cdot\beta_\nu, 1 + i\cdot\beta_\nu)$ denote the *span of bucket i*, $i \in \{1, \ldots, \nu\}$. Let further DB be a database of interval sequence objects and the function $\Omega(o_j, b_{i,\nu})$ denotes the sum $\Sigma_\iota intersect_{length}(\iota, b_{i,\nu})$ over all intervals ι of the interval sequence object o_j. Then, $\Psi(DB, \nu) = (n_1, \ldots, n_\nu) \in IN^\nu$ is called the *intersection probability histogram* on DB with *resolution* ν, iff for all $i \in \{1,.., \nu\}$:

$$n_i = 1 - \prod_{\forall o_j \in DB} \left(1 - \frac{\Omega(o_j, b_{i,\nu})}{\beta_\nu}\right)$$

In the above definition, we map an interval sequence object to ν gray intervals congruent to the histogram buckets each having a density $(\Omega(o_j, b_{i,\nu}))/\beta_\nu$. This density corresponds to the probability that one point $x \in b_{i,\nu}$ is intersected by the interval sequence object of o_j. Theorem 2 shows that the value n_i in Definiton 4 reflects the probability that $x \in b_{i,\nu}$ is intersected by at least one interval sequence object of the domain DB.

All local clients send their own intersection probability histogram to the server. The server computes for each client C^j a specific global intersection probability histogram Ψ^j.

Definition 5 (*global intersection probability histogram*). Let $DB_1, .., DB_m$ be the data sets of m different local clients with congruent intersection probability histograms $\Psi(DB_s, \nu) = (n_{1,s}, \ldots, n_{\nu,s})$, $s \in \{1,.., m\}$. Then, the global intersection probability histogram $\Psi^j(\bigcup_{\substack{s=1\ldots m \\ s \neq j}} DB_s, \nu) = (n_1^j, \ldots, n_\nu^j)$ for the client C^j can be computed as follows:

$$n_i^j = 1 - \prod_{\substack{s=1 \\ s \neq j}}^{m} (1 - n_{i,s})$$

Similar to the argumentation following Definition 4, the value n_i^j of Ψ^j in Definition 5 reflects the probability that $x \in b_{i,\nu}$ is intersected by at least one interval sequence object located at a client C^s where $s \in \{1,.., m\} \backslash j$.

5.2 Cost model

The approximation quality has a significant influence on the performance of the multi-step join process. If we adjust the approximation quality too low, for example by taking one-value approximations, the filter step is not very selective, thus many exact object informations have to be requested from the server. On the other hand, if we choose very accurate approximations, the initial transmission cost for sending the aggregated information of the gray intervals to the server are very high.

The overall join cost $cost_{join}$ related to a gray interval I_{gray} are composed of two parts, the filter cost $cost_{filter}$ and the refinement $cost_{refine}$:

$$cost_{join}(I_{gray}) = cost_{filter}(I_{gray}) + cost_{refine}(I_{gray}).$$

Filter cost. The filter cost $cost_{filter}(I_{gray})$ related to a gray interval I_{gray} depends mainly on the cost required to transmit the aggregated information of I_{gray} to the server. Furthermore, transmission includes the necessary identifier of I_{gray}, the hull $H(I_{gray})$ and the density $D(I_{gray})$. The total size of the transmitted data is constant, thus, we penalize each transmission by a constant c_{trans} which reflects the transmission cost related to one gray interval.

Refinement cost. The refinement cost related to I_{gray} depend on whether the server asks for the exact information of I_{gray} during the join process or not. Obviously, the probability that the server asks for the exact information depends on the probability whether I_{gray} intersects at least one gray interval or not. Thus we can estimate the refinement cost as follows:

$$cost_{refine}(I_{gray}) = \left(1 - \prod_{i=1}^{\nu} (1 - P(I_{gray}, I_{i,\nu}))\right) \cdot cost_{trans}(I_{gray}),$$

```
GroupIS (I_gray, Ψ^j(DB,v)) {
    interval_pair := split_at_maximum_gap(I_gray);
    I_left  := interval_pair.left;   I_right := interval_pair.right;
    cost_gray := cost_join(I_gray);
    cost_dec  := cost_join(I_left) + cost_join(I_right);
    if cost_gray > cost_dec then return GroupIS (I_left,Ψ^j(DB,v)) ∪ GroupIS(I_right,,Ψ^j (DB,v));
    else return I_gray; }
```

Figure 5. Grouping algorithm *GroupIS*.

where $cost_{trans}(I_{gray})$ denotes the cost required to transmit the gray interval I_{gray} from client C^j to the server. Furthermore, $I_{i,\nu}$ denotes a gray interval having the extension of the histogram bucket $b_{i,\nu}$ and a density equal to the value n_i^j of Ψ^j (cf. Definition 5). $P(I_{gray}, I_{i,\nu})$ denotes the probability that I_{gray} intersects at least one gray interval in the bucket $b_{i,\nu}$ stored at a client C^s where $s \in \{1,.., m\} \setminus j$ (cf. Theorem 1). The probability that I_{gray} intersects at least one gray interval in any bucket can be computed by means of Theorem 2 which is reflected in the above equation for $cost_{refine}(I_{gray})$.

5.3 Grouping Algorithm

Our cost-based grouping algorithm depicted in Figure 5 is a greedy approach which is performed in top-down fashion. It starts with a one-value approximation of the input interval sequence, i.e. all intervals are grouped into one large gray interval I_{gray}. At first we search the largest gap of I_{gray} and split it at this gap into two smaller gray intervals I_{left} and I_{right}. As long as the estimated transmission cost of the resulting gray intervals are smaller than the cost according to the unsplitted interval I_{gray}, the algorithm is applied recursively to both gray intervals I_{left} and I_{right}.

6 Server-Side Join Algorithm

The server-side join algorithm is based on the multi-step query processing paradigm. First, we detect all interlacing objects (cf. Definition 3). In order to decide whether an interlacing object pair intersects, it suffices to detect one intersecting gray interval pair of this object combination. Consequently, all remaining intersection tests according to these two objects can be discarded and the corresponding transmission cost can be saved. Therefore, it is desirable to rank interlacing gray intervals according to their intersection probability and their transmission cost. Obviously, the intersection probability should be high and the transmission cost should be low for a top-ranked interlacing gray interval pair. Thus, we compute the ranking value for a pair (I, I') as follows:

$$rank(I, I') = (1 - P(I, I')) \cdot (cost_{transmit}(I) + cost_{transmit}(I'))$$

Thereby, the intersection probability $P(I, I')$ between two interlacing gray intervals is computed according to Theorem 1. Note that all transmitted information is stored on the server site. Therefore, for each gray interval I which has already been transmitted from a local client to the server $cost_{transmit}(I)$ is equal to 0.

The interlacing gray interval pairs are organized in ascending order according to their ranking value in a sorted list *SortList*. The join algorithm iteratively carries out the refinement step for the top-listed gray interval pair. After the exact information of a gray interval I was transmitted from a local client to the server, $rank(I, I_i)$ of all gray interval pairs (I, I_i) stored in *SortList* is updated. Furthermore, if an object intersection is detected during the refinement step of the top-listed gray interval pair, all gray interval pairs belonging to the corresponding object pair are deleted from *SortList*.

7 Experiments

In this section, we evaluate the performance of our approach with a special emphasis on the overall transmission cost which are measured in bytes. All experiments were performed on a Pentium 4/2600 machine with IDE hard drives.

Test data sets. The tests are based on a test data set *CAR* which consists of 200 high-resolution 3D CAD objects provided by our industrial partner, a German car manufacturer. These voxelized objects have been linearized via a space filling curve leading to 200 interval sequence objects. Each of these objects consists of approximately 50,000 black intervals. Furthermore, we used an artificial test data set *ART* consisting of 1,024 interval sequence objects each represented by 10,000 black intervals. The objects are equally distributed in a range $[0..2^{27}\text{-}1]$ and the gap lengths inside an object follow a normal distribution. During the experiments, the objects of both test data sets were equally distributed on the available clients.

Grouping. We used two different grouping strategies for forming the gray intervals. The *MaxGap* approach tries to minimize the number of gray intervals while not allowing that a maximum gap $G(I_{gray})$ of any gray interval I_{gray} exceeds a given *MAXGAP* parameter. By varying this *MAXGAP* parameter, we can find the optimum trade-off between the two opposing grouping goals of Section 5.2, namely accurate approximations but a small number of gray intervals. A one-value interval approximation is achieved by setting the *MAXGAP* parameter to infinite. If the parameter is set to zero, each gray intervals is identical to one black interval. Furthermore, we used the *GroupIS* approach according to Figure 5, where we set the resolution of the used histograms to 10,000 buckets by default.

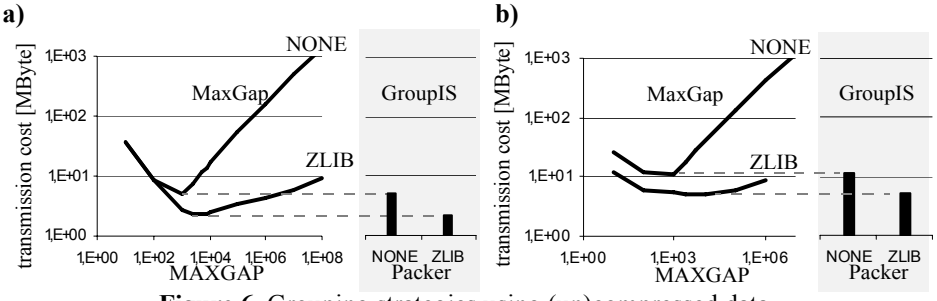

Figure 6. Grouping strategies using (un)compressed data
(a) CAR and **(b)** ART which are equally distributed on 4 local clients

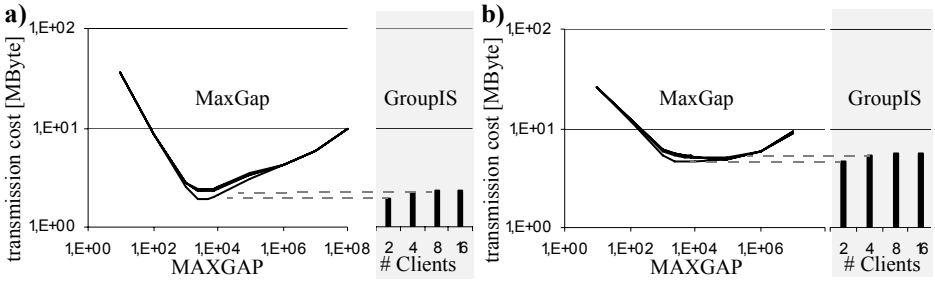

Figure 7. Different grouping strategies on the two datasets (*ZLIB*)
(a) CAR and **(b)** ART which are equally distributed on 2, 4, 8 and 16 local clients

Client-Side Grouping. In a first set of experiments, we compare our different client-side grouping strategies to each other. Figure 6 shows that for the MaxGap approach we have rather high transmission cost when using too small or two large MAXGAP parameters. When the parameter is two small many hulls have to be transmitted. On the other hand, when the parameter is very high the filter selectivity is very bad leading to high transmission cost during the refinement step of the server-side join algorithm. Figure 6 shows that by applying suitable packers for compressing the exact information of the gray intervals, these cost can dramatically be reduced. Note that our GroupIS approach does not produce higher transmission cost than the "optimal" MaxGap approach independent whether a packer (*ZLIB* [1]) is used or not (*NONE*).

In another experiment, we investigated the dependency of the different grouping approaches for a varying number of clients. In this experiment, we transmitted the exact information of the gray intervals in a compressed way. Figure 7 shows that our GroupIS approach yields optimum results independent of the number of used clients. Again, for low MAXGAP values, the number of hulls sent to the server is quite large and dominates the overall transmission cost. High MAXGAP values lead to a very small number of gray intervals per object, thus, almost all join candidates have to be refined.

Server-Side Join. In Figure 8 it is shown how the ranking function influences the transmission cost of the refinement step. Thereby, we compare our cost-based ranking approach (*CBR*) with the following methods which differ in the order in which the join candidates are refined: Ordered exclusively by the intersection probability (*PBR*), by the transmission cost (*LBR*), or in a randomized order (*RND*). This experiment shows that our approach achieves the lowest transmission cost, as well as the lowest number

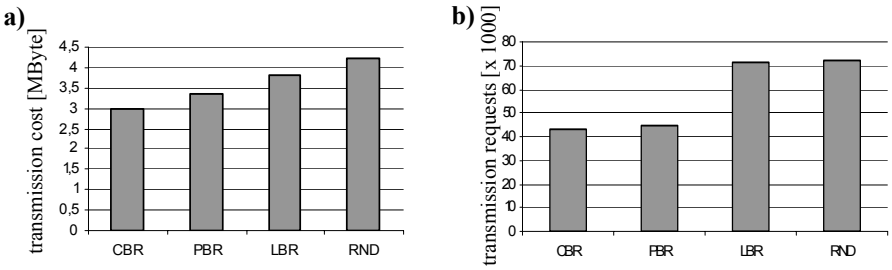

Figure 8. Different join strategies (4 Clients, *ZLIB, GroupIS*, *ART*)
(a) transmission cost **(b)** transmission requests

of transmission requests, i.e. our cost-based ranking approach produces the smallest additional communication overhead. Note that we made similar results for the CAR data set, a varying number of clients, and if we transmit the exact information uncompressed.

8 Conclusion

In this paper, we presented an intersection join for complex objects represented by interval sequences. Thereby the objects are assumed to be distributed on clients located at different sites. The intersection join is executed at a central server which is connected to all clients via local or wide area networks. The main goal of our approach is to minimize the client-server-communication cost incurred by the server side join process. Our proposed solution is based on generating approximations of the interval sequence data which are transmitted from the clients to the server site for a filter step. In contrast to existing solutions, e.g. error-bound approaches, our statistic driven proposal achieves a good trade-off between the communication cost of the filter and the refinement step. It adapts automatically to different client-server characteristics, e.g. different data sets, varying number of clients, or the used compression technique. Another contribution of this work is a cost based strategy for the refinement step. The experiments show that our approach leads to a speed-up of more than one order of magnitude compared to the use of one-value approximations or the use of no approximations at all.

In our future work, we plan to develop an even more efficient approximative distributed intersection join allowing fuzzy query responses.

References

1. Deutsch P.: RFC1951, DEFLATE Compressed Data Format Specification. http://rfc.net/rfc1951.html, 1996.
2. Enderle J., Hampel M., Seidl T.: *Joining Interval Data in Relational Databases*, Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'04), Paris, France, 2004.
3. Gao D., Jensen C. S., Snodgrass R. T., Soo M. D.: *Join Operations in Temporal Databases*, A Time Center Technical Report (TR-71), 2002.
4. Kargupta H., Chan P.: *Advances in Distributed and Parallel Knowledge Discovery*, AAAI/MIT Press, 2000.
5. Kriegel H.-P., Pfeifle M., Pötke M., Seidl T.: *Spatial Query Processing for High Resolutions*, Proc. 8th Int. Conf. on Database Systems for Advanced Applications (DASFAA), Kyoto, Japan, pp. 17-26, 2003.
6. Kriegel H.-P., Pötke M., Seidl T.: *Interval Sequences: An Object-Relational Approach to Manage Spatial and Temporal Data*, Proc. 7th Int. Symposium on Spatial and Temporal Databases (SSTD), LNCS 2121: pp. 481-501, 2001.
7. Medeiros C. B., Pires F.: *Databases for GIS*, ACM SIGMOD Record, 23(1): pp. 107-115, 1994.
8. McNeely W. A., Puterbaugh K. D., Troy J. J.: *Six Degree of Freedom Haptic Rendering Using Voxel Sampling*, ACM SIGGRAPH, pp. 401-408, 1999.
9. Özsu T., Valduriez P.: *Principles of Distributed Database Systems*, Prentice Hall, ISBN 0-13-659707-6, 1999.
10. Ramaswamy S.: *Efficient Indexing for Constraint and Temporal Databases*, Proc. 6th Int. Conf. on Database Theory (ICDT), LNCS 1186, pp. 419-413, 1997.
11. Tansel A. U., Clifford J., Gadia S., Jajodia S., Segev A., Snodgrass R.: *Temporal Databases: Theory, Design and Implementation*, Redwood City, CA, 1993.