Indexing RFID data using the VG-curve

Author

Terry, J, Stantic, B, Sattar, A

Published

2012

Conference Title

Conferences in Research and Practice in Information Technology Series

Indexing RFID data using the VG-curve

Justin Terry **Bela Stantic** **Abdul Sattar**

Institute for Integrated and Intelligent Systems (IIIS)
Griffith University, Gold Coast, Queensland, Australia,
Email: J.Terry, B.Stantic, A.Sattar@griffith.edu.au

Abstract

Existing methods for the management of multidimensional data typically do not scale well with an increased number of dimensions or require the unsupported augmentation of the kernel. However, the use of multidimensional data continues to grow in modern database applications, specifically in spatio-temporal databases. These systems produce vast volumes of multidimensional data, and as such, data is stored in commercial RDBMS. Therefore, the efficient management of such multidimensional data is crucial. Despite it being applicable to any multidimensional vector data, we consider Radio Frequency Identifications (RFID) systems in this work. Due to RFID's acceptance and rapid growth into new and complex applications, together with the fact that, as with commercial applications, its data is stored within commercial RDBMS, we have chosen RFID as a pertinent testbed. We show that its data can be represented as vectors in multidimensional space and that the VG-curve combined with Multidimensional Dynamic Clustering Primary Index, which can be integrated into commercial RDBMS, can be used to efficiently access such data. In an empirical study conducted on three, five and nine dimensional RFID data we show that the presented concept outperforms available off-the-shelf options with a fraction of the required space.

Keywords: Access Method, Multidimensional Data, Radio Frequency Identification - RFID

1 Introduction

There are many multidimensional indexes proposed in the literature, but very few have been adopted by the major database vendors due to their complexity and costs of integration. Many require unsupported access to the block manager so they cannot be readily constructed. Others are only possible as external indexes that do not inherit the industrial strength concurrency and recovery of the database system.

The vast majority of proposed multidimensional indexes require the unsupported augmentation of the kernel and are thus not readily available to support current RFID applications in commercial database systems. Currently to support and access multidimensional vector data in a commercial RDBMS it is possible to use bitmaps indexes, inbuilt R-Tree methods, approximation methods or one dimensional transformation methods. Bitmap indexes have been proposed for handling RFID data (Hu et al. 2005). Space requirements are reduced by using a bitmap data type that compactly represents a collection of identifiers. However, they have significant update costs and it may not work well when the data in the same cluster are not continuous or in applications that do not lend themselves well to grouping based on a common property (Lin et al. 2007).

Some commercial RDBMS has inbuilt R-Tree indexes, e.g., Oracle spatial, which has only recently supported intersections on three dimensional data. It is well known that the R-Tree's performance deteriorates significantly above 4 dimensions. They suit a variety of objects and thus have overheads in complexity, as well as not supporting intersection queries beyond three dimensions (Murray 2005).

Approximation methods like the VA-file (Weber et al. 1998) and the IQ-tree (Berchtold et al. 2000) are based on the belief that above a certain dimensionality a full scan is more efficient than an index, so it is best to improve the scan. The IQ-tree integrates compression into its index based query processing, using a three-level index structure to combine a tree with a scan using quantization. The VA-file is a simple vector approximation method that uses an array of compact geometric approximations. Queries are answered by excluding most vectors through an approximate filtering step on the entire VA-file itself. The VA-file reduces the number of disk accesses, however it incurs higher computational cost in decoding the bit-string and computing bounds. Another problem with the VA-file is that it works well for uniform data, but not for skewed data (Jagadish et al. 2005), due to the pruning effect of the approximation vectors deteriorating.

One dimensional transformation methods utilize the available single dimensional index structures, e.g., B-Trees, to index the data based on a scalar key. They employ a two stage query filter, similar to spatial querying, to extract approximate results that then have any false hits removed to produce a final result set. Examples include The Pyramid-Tree technique (Berchtold et al. 1998), iMinMax (Ooi et al. 2000), The P+-tree (Zhang et al. 2004) and i-distance method (Jagadish et al. 2005).

Single dimension transformation methods are typically not bijective functions and are lossy in nature, thus they rely more heavily on the exact filter than bounded regions in the feature space. Their performance is affected by false hits occurring due to objects being far apart in the original space but close in the transformed space. For distance transformations such as i-Distance there may be many objects similarly close to a reference point, but it is unlikely they are all close together and many will need to be put through the exact filter.

Space Filling Curves (SFC) are bijective transformation functions that can produce a scalar key. The most well known SFC method is the UB-Tree (Berchtold et al. 1999), which integrates a SFC and a B-Tree creating a primary index for multidimensional data. It is an efficient paginated index where each leaf node represents a page of data on a segment of the curve. However, like other SFCs, the segments are typically not hyper-cubic and may even represent disjoint space, this typically increases the number of pages read that do not contribute to the answer and more importantly the UB-Tree requires changes to the kernel for integration so cannot be readily used as an internal index (only integrated in Transbase database).

One of the most prominent d dimensional point data structures is the K-D-Tree (Bentley 1975) and its variants: the *hB-Tree* (Lomet & Salzberg 1989), the *BD-Tree* (Ohsawa & Sakauchi 1983), the *hybrid tree* (Chakrabarti & Mehrotra 1999) and the quad-Tree. The K-D-Tree is a binary search tree that uses a recursive subdivision of the data space into partitions by means of (d - 1)-dimensional hyperplanes. A disadvantage common to all K-D-Tree methods is that for certain distributions, no hyperplane can be found that divides the data objects evenly. Like the K-D-Tree, the quad-tree (Samet 1984) decomposes the universe by means of iso-oriented hyperplanes. An important difference however, is the fact that quad-trees are not binary trees anymore. The subspaces are decomposed until the number of objects in each partition is below a given threshold. Quad-trees are therefore not balanced and the subtrees of densely populated regions need to be deeper than sparsely populated regions, giving a bad worst case behavior. Disadvantages of space partitioning methods in general are that they can suffer from poor minimum node utilization, or have a high space complexity. Tree based space partitioning methods are typically unbalanced, increasing the worst case performance.

Some methods for efficient management of temporal data which can be incorporated within commercial database management systems have been presented in literature (Stantic, Topor, Terry & Sattar 2010), (Stantic, Terry, Topor & Sattar 2010). However, these methods cannot efficiently support high dimensional queries.

RFID data is naturally spatial and temporal in nature having time and location as two of its basic elements and additional attributes can be seen as additional dimensions. Therefore, it is well suited to being represented as vectors in multidimensional space where multidimensional queries as well as spatial and temporal predicates can be applied in a straight forward manner. Data warehouse applications will also need an efficient multidimensional access structure to support the growing demand for ad-hoc querying (M.Stonebraker & U.Cetintemel 2005, Stockinger et al. 2002). Typical multidimensional sample ad-hoc query could be: "Find containers that were picked up by fleet F for district D in the last 7 days and delivered by today". In order to efficiently answer these kinds of queries an efficient access methods that scales well in volume and dimensions and is available within commercial RDBMS is crucial where these data records will be stored.

To be able to suit a variety of applications and data models we must consider medium to high dimensional data. The most commonly used access methods for medium to high dimensional vector data are access methods available off-the-shelf in commercial RDBMS (Rudolf Bayer and Volker Markl 1998). RFID deployments are most commonly within a commercial relational database where the best database services such as industrial strength concurrency and recovery are available. Using off-the-shelf indexes to manage RFID data guarantees these services to the application. Thus we do not consider index methods that require kernel modification such as the UB-Tree method or access method that are known not to scale well with increasing dimensions like the R-Tree in Oracle's Spatial Index.

In this work, we have built on top of the concept proposed in VG-Curve (Terry et al. 2011) and have shown that the VG-Curve combined with Multidimensional Dynamic Clustering Primary Index can efficiently manage and access RFID data. The proposed concept is not sensitive to the number or order of dimensions restricted in the query and is easily constructed within existing commercial database management systems and can efficiently manage the large volume of multidimensional RFID data. Storing the data in its multidimensional feature space allows processing of a wide variety of queries. The most important of these for spatial and temporal predicates is the multidimensional interval query that allows many spatial and temporal queries to be applied in a straight forward manner. In empirical evaluation, we demonstrate the performance of presented concept and show its superiority to the currently available off-the-shelf index methods for multidimensional RFID data.

The remainder of the paper is organized as follows; Firstly we review some key background information on RFID technology focusing on the tags and requirements of managing the data. In section 4 we describe our experimental study and in section 5 we analyze the results, finishing with a conclusion and further work in section 6.

2 Multidimensional Nature of RFID

Tagged objects moving through a RFID-based pervasive environment are automatically sensed and observed with their identifications, locations and movement paths. These observations are filtered and recorded in a database producing spatial, temporal and many additional dimensions (attributes) of data.

RFID, having time and location as two of its most basic elements, is naturally spatial and temporal in nature. Therefore, it is well suited to being represented in multidimensional space where spatial and temporal predicates can naturally be applied. Using a multidimensional access method that preserves the original feature distances allows many spatial and temporal predicates to be applied in a straight forward manner to the data without further data transformation.

Interval queries are an integral requirement for both spatial and temporal predicates in multidimensional data space. The efficiency of spatial and temporal queries relies on an efficient interval access method for multidimensional space. Thus any efficient multidimensional access method must be efficient at answering interval queries.

Fixed scanning devices have a known location and mobile scanners can be combined with global positioning to give accurate physical positioning of the location an object scan occurs. The spatial co-ordinates can then be stored and spatial predicates such as `inside` can then be run against the RFID data. This can be combined with the scanning time to answer spatio-temporal predicates such as `inside x` during **T** where `T` is the time interval.

Besides spatial and temporal attributes, RFID data can have many other attributes of interest. Current RFID systems with three to seven dimensions have been identified in the literature (Lin et al. 2007, Wang & Liu 2005) and the use of more complex application-specific tags requires even higher dimensionality. RFID applications will need an access method that can efficiently cope with data from a variety of dimensions to handle both simple and complex tag data as well as a higher level of data modelling such as those produced from data compression (Darcy et al. 2007).

The difficulties associated with multidimensional data grow with the number of dimensions. Once data has more than three or four dimensions, additional problems begin to arise which is loosely termed the *'curse of dimensionality'* that can severely deteriorate an access method's performance. Higher dimensionality characteristics include: exponential volume growth with additional dimensions, high probability of objects being near an edge, data sparsity, pages remaining unsplit in some dimensions and not cubic in shape and queries having very large extensions in each dimension (Berchtold et al. 1998, Bohm 2000, Weber et al. 1998). Due to these characteristics, the performance of traditional multidimensional access methods deteriorates rapidly as the dimensions increase (Orlandic & Yu 2002). Thus the majority of proposed access methods do not scale well to higher dimensions.

Same as for any spatio-temporal data the efficient management of RFID data is challenged by its large volume and multidimensional nature. The current and future needs of RFID applications will require an access method that is not only efficient, both in space complexity and in query performance, but is able to scale well with an increasing amount of volume and dimensionality. A reluctance to go above low dimensionality due to efficiency concerns may limit the design of RFID applications to low dimensional models, even when a higher dimensional data set has proven to be more effective and efficient (Lin et al. 2007).

3 Efficient Management of RFID Data

RFID data is generated quickly and automatically, and can be used for real-time monitoring or accumulated for object tracking. To filter and clean the high volume of real-time RFID data efficient methods are essential, especially for real-time applications.

RFID applications require a dynamic scalable access method that enables the spatial, temporal and additional dimensions of data to be queried in any combination and independently of each other. The use of multiple single dimensional or compound indexes to achieve this is not an efficient solution in terms of space complexity and update costs, though it is the most common solution to the problem.

3.1 The Primary Multidimensional Index

We build on top of VG-curve concept where variable regions that are formed are influenced by the distribution of data and blocking factor. It is important to mention that the concept does not enforce one-to-one relationship between regions and database blocks. However, it relies on the Primary index data being clustered on the cluster key, which in our case is unique region identification number - *RegionID*. To achieve the benefits that storing data in order gives to query efficiency, we have combined one dimensional transformations into clusters based on a SFC, with multi-stage query process. Producing two structures, a primary index managed by the DBMS, that stores the data in order and a summary structure that manages the keys given to the data in form of directory.

We use a linear ordering of dynamic (locally defined) data clusters that are stored contiguously so that nearby clusters have a high probability of being in nearby blocks, i.e., a Multidimensional Dynamic Clustering Primary Index (MDCPI).

A **Cluster** is the group of similar (nearby) objects sharing the same *cluster key* (in our case *RegionID*) that will be stored contiguously in the base relation. We denote population *pop* as the number of objects located in the cluster. The *cluster factor* is the maximum number of objects allowed in a cluster before it is typically split into two clusters.

A **Base Relation** is an indexed relation altered and ordered by a cluster key (*RegionID*) plus a unique object key. It also contains one column for each dimension of the object and may contain other none-indexed attributes. This relation, typically a B^+-Tree, is maintained and kept in order by the DBMS thus is sometimes known as an Index Organized Table. The base relation can be thought of as the access structure and leaf pages of the access method.

A **Directory** is a compressed representation of the base relation containing its *RegionIDs* and their population.

Control Processes are coded algorithms that manage and query the clusters including defining the order of clusters and hence data.

A MDCPI can retain the advantages of a primary access method while being able to exploit the particular characteristics of a variety of clustering schemes compared to a traditional clustering index.

The concept of MDCPI differs from existing transformation methods as it determines partitions at the local level which are managed by the directory using flexible user defined control processes as well as multistage query processing.

Three major benefits of using a MDCPI are:

- **Efficient Interval queries:** Interval queries benefit from the primary index organization as it is likely that several groups of sequentially stored clusters will be accessed to answer any interval query. These sequential clusters will be stored contiguously and can thus be efficiently retrieved by performing a range scan on the base relation for each sequence of clusters.
- **Low Space complexity:** the size of directory records is small, two fields, and the directory only stores the populated clusters. If the average population is 500, the directory will be less than 0.2% of the base relations size. We address the space complexity problems associated with block oriented (paginated) space partitioning methods as a cluster consumes only the physical space needed to store its objects, plus free space reserved for updating of the ordered base relation. This contrasts with block oriented methods that store each region's data in a block regardless of block utilization.
- **Simple Integration:** The MDCPI is suitable for any relational or object-relational database system that offers an ordered relation and procedures. The access method is constructed with standard objects from off-the-shelf DBMS without the need for kernel modification, and as such, inherits database services such as industrial strength concurrency and recovery.

3.2 The Concept of Proposed Method

We present a space partitioning method that has common characteristics with Space Filling Curves (SFC). It suits a MDCPI as it forms a linearly ordered set of multidimensional clusters (regions). Previously proposed SFC methods employ a curve that passes through all points in multidimensional space, however, a Variable Granularity Curve (VG-curve) (Terry et al. 2011) connects regions of various granularity. An additional benefit is that the space filling curve only connects regions populated with objects.

Though the partitioning is similar to quad-tree partitioning the VG-curve is not a paginated index and only one dimension at the time it is split, which is a considerable advantage as the number of dimensions grow. Splitting on additional dimensions to enhance the discriminatory power of the index is a characteristic we share with the TV-tree (Lin et al. 1994). Our method has three main differences to the TV-tree; we use a space partitioning strategy that creates a unbalanced virtual tree (the directory) representing the occupied data space, our method uses a multi stage filter process on separate data and directory structures and as it is not paginated it can be constructed without kernel modification.

Though the base relation stores the data in a balanced structure, the VG-Curve regions (in the directory) can be considered as an unbalanced binary tree where empty leaf and internal regions are not required to be stored. Not requiring internal nodes to be stored is a property that we share with the linear quad-tree (Gargantini 1982) but presented concept with VG-curve is not paginated but built using the MDCPI components.

Three major benefits of using variable size regions are:

- **Regions can efficiently partition the data:** The volume r_v of a region decreases exponentially with its address length, L:

$$r_v = v * (2^{-L})$$

  allowing a fine partition with a relatively short address length.

- **Intersection calculations are simpler:** Intersecting regions will be fewer than intersecting points, particularly as it is only populated regions, and thus it is faster to calculate the intersecting regions compared to calculating the intersecting points of other SFC based methods.

- **The regions are hypercube like in shape:** Unlike all other SFC's segments the VG-curve regions will have side lengths that differ at most by a factor of two making them hypercube or hyper-rectangular in shape. Having hypercube like shaped regions is widely recognized to reduce the number of regions that overlap with a query interval improving efficiency for query processing.

Regions can hold up to the blocking factor (BF) number of objects. If the number of objects exceeds the BF the region is split along the next dimension as splitting is done in circular order of dimensions. The Algorithm assumes a fixed ordering for dimensions, but any ordering can be used, since the approach is almost non-sensitive to it (only the number of not-empty regions can slightly vary depending on such an ordering). An overfull region may require a child region to be further split if the first split does not produce two under full regions (Terry et al. 2011). This occurs when the data objects are contained wholly within one child of the region. In this case, splitting of the overfull child will continue until two under full child regions are produced or until the pixel size region is reached. When a pixel becomes overfull it is not split and it's population is allowed to grow beyond the clustering factor, similar to the concept of super-nodes for X-tree high-dimensional indexing (Berchtold et al. 1996). This is possible as the physical storage of a region is not limited to a page.

Multiple splits to partition a cluster do not increase the number of directory regions, as only populated regions are stored in the directory. When a split creates an empty child region, the density of the index increases as the sibling child has the same population as its parent in half the volume, thus the empty regions make a denser index.

The different SFC partitions derived from UB-Tree and VG-Curve methods are shown in Figures 1. They shows the top half of a data space containing identical sets of data where each block contains 8 objects. The three UB-Tree areas in Figure 1 are (0,2,1,0),(1,1,2,1),(2,0,0,0). The VG-Curve regions in Figure 1 are derived using the above mentioned partitioning scheme which is in detail explained in (Terry et al. 2011).

Unlike the UB-Tree partitioning, our SFC partitioning scheme allows the identification and exploitation of empty areas to improve query processing. This can be seen in Figures 1 where the query interval (shown as a dotted rectangle) intersects three UB-Tree pages but only one VG-Curve page.

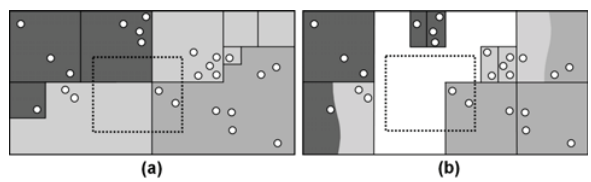

Figure 1: Three (shades) pages using (a) UB-Tree partitioning and (b) VG-Curve partitioning, two of the nine VG-Curve regions contain two shades representing regions stored in two pages. The query intervalis shown as a dashed line.

The percentage of data space that a region represents is not dependent on the dimensionality, but on the length of its address. Thus the space can be efficiently divided regardless of the dimensionality.

3.3 Query Processing Method

There are several query types of interest for point objects stored in multidimensional space. Relevant examples are Interval Queries (IQ) and Exact Match Queries (EMQ). If p is a point in a d dimensional space and i_q is a d-dimensional query interval then the above queries can be represented as:

- Interval Query (IQ)
  - $p \in i_q$, find all objects that are contained within the query interval
- Exact Match Query (EMQ)
  - $p = i_q$, find all objects that have the same value as the target point for each d dimension.

In this paper, we focus on the efficient processing of interval queries on medium dimensional point data (d = 2-9) as well as the exact match query, as it is a specific type of interval query. Multidimensional

range searching, such as interval queries, plays an important role in the way modern applications query their data. It covers many different query predicates in different data models (e.g, temporal, spatial etc).

Queries are performed by identifying the regions intersecting the query interval and then adding a restriction to the original query that region must be in the intersecting set. The query is then efficiently answered with a range scan of the base relation.

Queries are processed in three stages, the directory is preprocessed to remove some regions that cannot contain answers then a primary (spatial) filter is used to select intersecting regions from the remaining regions and a secondary filter to remove false hits in regions overlapping the query interval (see Figure 2). Intersecting regions can be either Contained regions C or Overlapping regions O. Contained regions only have answer objects whereas overlapping regions will need to have their objects checked for false hits.

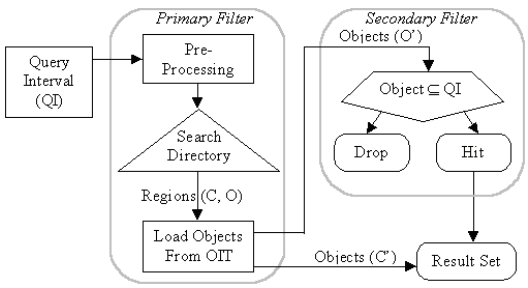

Figure 2: The query filtering stages; Preprocessing reduces the regions considered. Primary filter is a spatial query on the regions while secondary filtering is to remove false hits from the contributing regions data.

In the preprocessing of the directory, we identify the first and last directory region that can contribute to the result set. We then consider only these regions and the regions between them.

Similarly to spatial filtering, the intersecting query evaluation algorithm will return all intersecting data objects. As can be seen in Algorithm 1. In the primary filter all regions intersecting the query interval are returned. At first, all preprocessed regions from the directory are loaded into a cursor in a depth first order. Each region is then tested for intersection with the query interval. Contained regions are added to the result list $A1$. Overlapping regions are added to the result list $A2$. The secondary filter tests all objects in $A2$ and adds the positive hits to all objects in $A1$ for the final output A'.

In the secondary filter, we check each overlapping object for containment in the query window. All false hit objects are removed to create the final result set. It is important to note that the secondary filter uses CPU only and does not cause any further disk I/O's.

We also have the option of performing the primary filter and calculating the intersecting regions without reading the directory. We have investigated and proposed algorithm which can calculate the set of all possible regions of interest that intersect with the query interval without any I/O's on the directory index. These regions exist between the minimum region depth and the maximum tree depth (meta data parameters). The existing regions will be a subset of these regions of interest. Joining the regions of interest to the data will produce the same result as the primary filter. However, this concept is only efficient for exact match queries and when the tree is roughly balanced or the query interval is small. If the tree is highly unbalanced or a query interval is large it may cause the CPU time to blowout, in a similar but less dramatic fashion then standard SFC methods. Therefore, in cases with a highly unbalanced tree or a large query interval, consulting the directory is more efficient.

Algorithm 1 Query primary filter algorithm

```
begin
Input: preprocessed regions from Directory Dir ,
Query Interval Q
Output: Containing Regions A1, Overlapping Regions A2
Add all regions in Dir to LIST in order
Let length L be 1
while LIST is not empty do
  Let F be the first region in LIST
  Let R be the ancestor of F
  whose identifier consists
  of the first L digits of F's address
  if R is contained within Q then
    Move
    from LIST to A1
    Set L to 1
  else if R is disjoint from Q then
    Remove all regions a with a(L) ⊂ F
    from LIST
    Set L to 1
  else if R equals F then
    Add R to A2
    Remove R from LIST
  else
    Increment L
  end if
end while
end
```

4 Experiment

In order to empirically prove the efficiency of the method presented in this paper on medium dimensional data (3-9 dimensions), we have conducted an extensive empirical evaluation against the most common indexes used to support medium to high dimensional vector data (Rudolf Bayer and Volker Markl 1998). Our aim is to follow commercial RFID applications and efficiently manage RFID data within a commercial relational database where the best database services are available.

In line with the study performed to evaluate the efficiency of the UB-Tree (Rudolf Bayer and Volker Markl 1998) and VG-Curve (Terry et al. 2011), we compare with the best available methods in off-the-shelf commercial RDBMS for medium to high dimensional data, i.e., multiple secondary indexes, compound indexes and table scans. As it has been shown that even for six dimensions a compound index outperforms secondary indexes when the result set is greater than 0.000015 % of the relations population (Rudolf Bayer and Volker Markl 1998).

Unfortunately, we could not compare with the UB-Tree method, as it requires kernel modification. Also, we did not compare with access methods that are known not to scale well with increasing number of dimensions like the R-Tree in Oracle's Spatial Index. The performance of other SFC methods (e.g., Z-curve) deteriorate rapidly above a few dimensions and as the query interval grows due to a blow out in CPU operations, which we confirmed in initial testing, thus they were found to be unsuitable for this experiment.

Currently, the most widely used technique to handle multidimensional interval queries is the use of a

secondary index for each dimension (Rudolf Bayer and Volker Markl 1998). The performance of multiple secondary indexes however deteriorates rapidly as the dimensions grow and is only useful for very small result sets, e.g., in (Rudolf Bayer and Volker Markl 1998) for six dimensions a compound index outperforms secondary indexes when the result set is greater than 0.000015 % of the relation's population. Multiple secondary indexes have an additive behavior whereas the VG-curve has a multiplicative behavior.

Assume N objects in the universe and that $p_i\%$ of objects lay in the query intervals restriction of the ith dimension. Then additive behavior means we need to fetch $N * p_i$ objects (or object identifiers) for the ith dimension. Requiring in total $\Sigma_{i=1}^{d}\ N * p_i\%$ objects to be fetched.

For the VG-curve the amount of data to be fetched is approximately proportional to the interval query result set i.e., $N * \prod_{i=1}^{d} p_i\%$. As we are focussing on interval queries at medium to high dimensionality and as we do not wish to restrict the result sets to very small we have focussed on comparisons with the compound index and table scan.

To efficiently index multidimensional RFID data with compound indexes requires the full set of combinations of compound indexes covering from 2 dimensions up to the dimensionality of the data. This set of compound indexes can then efficiently answer a query restricting any combination of dimensions. The number of compound indexes required for a data set is given by:

$$\sum_{r=2}^{d} \left(\frac{d!}{(d-r)!r!}\right) \tag{1}$$

where d = the number of dimensions of data. For three dimensions, this means four indexes i.e., ((D1,D2), (D2,D3), (D1,D3) and (D1,D2,D3)). For five dimensions twenty-six compound indexes are needed and for nine dimensions five hundred and two compound indexes are required. This explosion of combinations of required indexes is in line with the curse of dimensionality hypothesis that renders most indexes inefficient due to the exponential space and complexity that each additional dimension contributes (Bohm 2000).

4.1 Data Set

We have tested the performance of VG-Curve on RFID data with three different dimensions:

For three dimensional *Raw* RFID data we have considered the following schema: `(EPC, ScanTime, LocationID)`. Data represents a chain of 50 stores where every store has 20 scanners monitoring 20 product lines with 500 items in each line.

For 5 dimensional data, we used the container-location table from (Lin et al. 2007), with data on five fleets of 20 trucks servicing 10 districts containing 100 stores over a 3 month period. Five dimensional data has the following schema: `(EPC, LocationID, TimeStart, TimeEnd, TransportMeansID)`

The application specific 9 dimensional scenario proposed is for a Shipping Consortium running an international container shipping application monitoring the movements of containers in the past 100 days across 25 nations with 40 ports and each serviced by 4 fleets of 25 ships using 1000 container terminals with 20 readers in each. Containers are ordered by 20,000 importers from 10,000 exporters. The following schema has been considered: `(EPC, Scantime, LocationID, ShipID, DepartPortID, ArrivePortID, ExporterID, ImporterID, ShipDate)`

All VG-curve relations used a blocking factor of 1,000 and a max depth of 85.

4.2 Query Set

One dimensional queries like the tracking query: "Given the EPC find the history of object" can be efficiently answered with a standard one dimensional index so are not considered. However, two dimensional queries such as: "Find all objects at a certain location at a certain time" can not be efficiently answered with a one dimensional index.

We have tested queries where between two and all dimensions are restricted to demonstrate the performance of our multidimensional access method as a replacement for all combinations of compound indexes for the given relation.

For ease of comparison, all queries aim to return approximately 5000 rows. We compare against the ideal compound indexes for each query i.e., compound indexes who's indexed dimensions match the query restrictions. All query subgroup ranges start from a randomly selected identity number. To be able to achieve the a consistent result set size from queries above six dimensions we simply used the a query of the same percentage restriction in each dimension. Due to space constraints we can describe only a few example queries from each of the five and nine dimensional sets.

For the 3 Dimensional relation, we used the following queries:
Q1. Find all items in Product Line `P` scanned at Store `S`.
Q2. Find all items in Product Line `P` scanned at Store `S` for the month `M`.

Example queries for 5 dimensional relation:
Q1. Find containers that were picked up by a contaminated truck `T` in the last 9 days.
Q2. Find containers that were picked up by fleet `F` for district `D` in the last 7 days and delivered by today.

Example queries for 9 dimensional relation:
Q1. Find containers ordered by an exporter `E` shipped during the past 50 days.
Q2. Find containers moved from nations `N` by fleet `F` shipped more than a month ago arriving at a container terminal before the past fortnight.

Each query has its restrictions randomly instantiated 20 times and average performance figures are reported.

4.3 Environment

All experimental results presented in this Section are computed on a Sun Fire V880 server with 8 x UltraSPARC-III 900MHZ CPU using 8GB RAM, running Oracle 10g RDBMS. Database block size was 8K and SGA size was 1GB. At the time of testing database server had no other significant load. We used built-in methods for statistics collection, analytic SQL functions, and the PL/SQL procedural runtime environment. All queries had the buffers flushed before running.

5 Result and Analysis

As anticipated, our results show that the VG-curve combined with MDCPI concept is an effective replacement for multidimensional combinations of compound indexes for a given data set at a fraction of the space. Compared to the best compound index for each query our method is clearly superior for 3 and 5 dimensions with on average reductions in CPU and I/O, as can

VGC Vs Ideal Comp	3D	5D	9D
I/O (%)	−86.7	−86.0	−39.9
CPU (%)	−11.9	−9.8	208.4

Table 1: Average percentage decrease (negative) or increases on 3, 5 and 9 dimensional data sets of the VG-curve compared with the ideal compound index for each query.

be seen in Table 1. In this table best compound index value is considered 100% and therefore -86.7% for I/O means our concept required 86.7% less I/O's that the best compound index.According to the theory of indexability a physical disk accesses are considered to be the most important aspect to be taken into account (Hellerstein et al. 1997). For nine dimensions our method requires less disk I/O's however due to the query algorithm requires more CPU, which is due to the increased overlap at the high dimensions. However, it is important to mention that our method can basically efficiently answer any combination of restricted dimensions. In contrast using compound indexes to be able to answer any combination of restricted dimensions according to the equation 1 would require 502 different compound indexes for nine dimensional data. It is also important to highlight that the compound indexes in our experiment are constructed to have the same dimensions as the query of the same dimensionality, i.e., they are the best compound index for that query. To support this claim, it is important to highlight that in experiments for different queries we used best possible compound index to efficiently answer specific query and in all cases compound index was different, however, for all queries same VG-curve structure was used built on top of MDCPI concept.

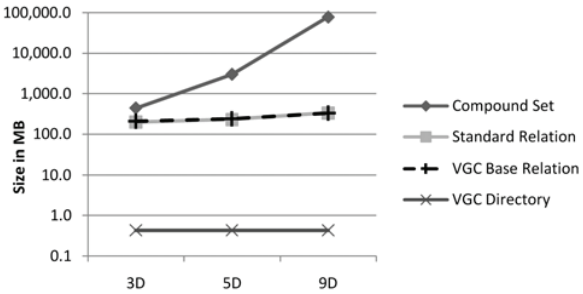

Figure 3: Space Complexity for three to nine dimensional data

As seen in Figure 3, the size of the VG-curve directory is a small fraction of that required for any compound index. The size difference between a directory entry and a compound index entry combined with the group of data represented by a single directory entry means the VG-curve approach is clearly superior in space complexity. When space is taken into consideration the compound index approach fails dramatically and gets exponentially worse as the number of dimensions grows.

This is clearly visible in Figure 3 where the VG-curve directory size is constant across the dimensions, due to the mostly even data distribution, and is between 219 and 566 times smaller than any one of the compound indexes and between 1,000 and 180,000 times smaller than the combination set of compound indexes for one RFID relation. In our method, adding the region field increased the size of the relation by 7% but this pales in comparison to the alternative and size of all combinations of compound indexes. Even for just 3 dimensions adding the compound indexes needed increased the space used by 370%, by 9 dimensions they increase the space needed by 315 times. As can be seen in Figure 3 our method's base relation achieved the storage utilization guarantee of the underlying B^+-Tree.

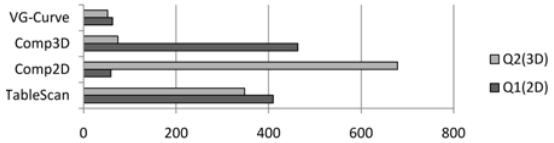

Figure 4: CPU usage for three dimensions

We show the performance comparison in both CPU and I/O's for three dimensions in Figures 4 and 5, in five dimensions in Figures 6 and 7 and in nine dimensions in Figures 8 and 9. The number of dimensions restricted in each query is shown on the legend in brackets next to the query number. Queries are displayed in the charts in the same order as they appear in the legend.

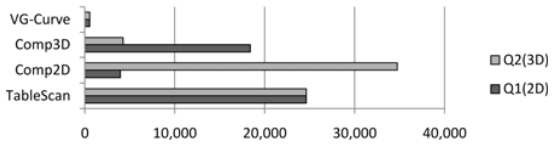

Figure 5: Physical Disk I/O's for three dimensional data

Figures 5 and 7 show our method to be superior for all multidimensional queries on three and five dimensions compared to the best of the compound indexes for each query.

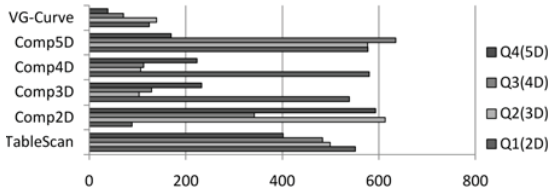

Figure 6: CPU usage for five dimensional data

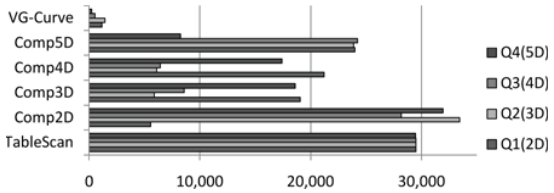

Figure 7: Physical Disk I/O's for five dimensional data

Despite the fact that we build the best possible compound indexes for each query, out of 502 possible for 9 dimensional relations, the physical disk I/O of our method (Figure 9) is superior for 5 of the 6 multidimensional queries. The exception is the 2 dimensional index on the 2 dimensional query on the 9 dimensional data set. The I/O performance of our method demonstrates its suitability for medium to high dimensional data by showing its scalability over several dimensions without the usual severe performance deterioration.

Limiting the combinations of compound indexes to a subset due to space considerations runs the risk

of poor performance as can be seen in the results for all relations and especially Figure 8 for the 2 and 4 dimensional compound index, where queries restricting dimensions not matching the indexes can perform very poorly.

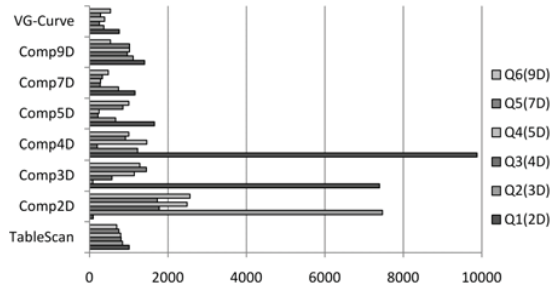

Figure 8: CPU usage for nine dimensional data

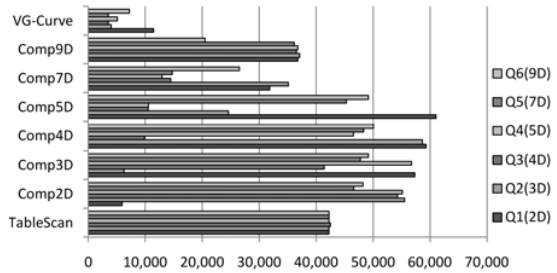

Figure 9: Physical Disk I/O's for nine dimensional data

The CPU performance of our method can be seen in Figures 4, 6 and 8 to be better in 6 of the 12 multidimensional queries compared to the best of the compound indexes for each query. It is worth noting that the max depth (address length) parameter used was 85, though it could have been set to 20 without any region being at pixel depth, improving the reported CPU use for our method.

In other testing on highly clustered data of up to 18 dimensions not shown here, the performance of our method was up to 50% better. This is due to data exhibiting large areas of empty space, as in our method empty regions are not represented. The populated regions indexed cover a smaller volume and the data is more dense compared to more evenly distributed data. Also, the performance benefit grew with larger result sets as the average number of result rows returned per region increased taking better advantage of physical clustering. Changing the cluster factor varies the CPU I/O trade off caused by changing the number of regions used, smaller regions cause higher CPU usage but use fewer I/O's due to less overlap. We varied the cluster factor in earlier experiments widely from 100 to 10,000 and found it to not be a sensitive parameter typically causing less than a 10% variation in performance measures.

In summary, the presented concept differs from the K-D-Tree since in the VG-Curve approach partitions are at predetermined positions and it is a secondary (disk - oriented) not primary memory storage method. It differs from Quad trees since the VG-Curve dimensions are split one at a time. It differs from the grid file since the partitions are applied locally to the node not across the whole dataspace. It differs from the UB tree since regions are hyper cubic, or hyper rectangular with two side lengths of x and 2x. It differs from other SFCs since it uses a directory and a two stage query processing. We differ from the VA file since we use a clustering index entry, and not approximations. We differ from i-distance since we use hyper cubic like regions, not selected points and offsets.

6 Conclusion and Future Work

This study makes the following contributions to the field:

- We have shown that multidimensional data can be organized in a way suitable for employing a primary index structure which guarantees better performance.
- Through an empirical study, we have demonstrated that the VG-curve method combined with the Multidimensional Dynamic Clustering Primary Index is superior to the off-the-shelf compound index, which is tailored for the specific query, for processing range queries on RFID data of medium dimensionality.
- In our experiments, we have efficiently answered many different queries with a single structure obtained following the presented concept.
- Therefore, the presented concept can replace many combinations of compound indexes for a given multidimensional data set with one small directory for RFID data of medium dimensionality.

In future work we intend to investigate the applicability and performance of the VG-curve method on a wider variety of query types and data sets while identifying how to best exploit the parameter blocking factor to suit different data characteristics and local conditions.

References

Bentley, J. (1975), 'Multidimensional binary search trees used for associative searching', *Communications of the ACM* **18**, 509–517.

Berchtold, S., Bohm, C. & Kriegel, H.-P. (1998), 'The Pyramid-Tree: Breaking the Curse of Dimensionality', *Proceedings ACM SIGMOD* pp. 142–153.

Berchtold, S., Bohm, C., Kriegel, H.-P., Sander, J. & Jagadish, H. (2000), 'Independent quantization: An index compression technique for high-dimensional data spaces', *icde* **00**, 577.

Berchtold, S., C.Bohm, Kriegel, H. P. & Michel, U. (1999), Implementation of Multidimensional Index Structures for Knowledge Discovery in Relational Databases, *in* 'Int. Conf. on Data Warehousing and Knowledge Discovery DaWaK', pp. 261–270.

Berchtold, S., Keim, D. & Kriegel, H. (1996), 'The X-tree: An index structure for high-dimensional data', *In Proceedings of the 22nd Int. Conf. on Very Large Data Bases* pp. 28–39.

Bohm, C. (2000), 'A Cost Model for Query Processing in High Dimensional Data Spaces', *ACM Transactions on Database Systems* **25**(2), 129–178.

Chakrabarti, K. & Mehrotra, S. (1999), The hybrid tree: An index structure for high dimensional feature spaces, *in* 'Proceedings of the 15th International Conference on Data Engineering (ICDE'99)', pp. 440–447.

Darcy, P., Stantic, B. & Derakhshan, R. (2007), 'Correcting Stored RFID Data with Non-Monotonic Reasoning', *Principles and Applications in Information Systems and Technology (PAIST)* **1**(1), 65–77.

Gargantini, I. (1982), 'An Effective Way to Represent Quadtrees', *Commun. ACM* **25**(12), 905–910.

Hellerstein, J., Koutsupias, E. & Papadimitriou, C. (1997), 'On the Analysis of Indexing Schemes', *16th ACM SIGMOD Symposium on Principles of Database Systems* pp. 249–256.

Hu, Y., Sundara, S., Chorma, T. & Srinivasan, J. (2005), Supporting RFID-based Item Tracking Applications in Oracle DBMS Using a Bitmap Datatype, *in* 'Proceedings of Very Large Data Bases VLDB', pp. 1140–1151.

Jagadish, H. V., Ooi, B. C., Tan, K.-L., Yu, C. & Zhang, R. (2005), 'iDistance: An Adaptive B+-tree Based Indexing Method for Nearest Neighbor Search', *ACM Trans. Database Syst.* **30**(2), 364–397.

Lin, D., Elmongui, H. G., Bertino, E. & Ooi, B. C. (2007), Data management in rfid applications, *in* 'DEXA', pp. 434–444.

Lin, K., Jagadish, H. & Faloutsos, C. (1994), 'The TVtree an index stucture for high dimensional data', *In VLDB Journal* **3**(4), 517–543.

Lomet, D. & Salzberg, B. (1989), 'The hB-tree: A robust multiattribute search structure', *In Proc. IEEE international conference on data enginerring* **5**, 296–304.

M.Stonebraker & U.Cetintemel (2005), One size fits all: an idea whose time has come and gone, *in* 'Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)', pp. 2–11.

Murray, C. (2005), 'Oracle Spatial User's Guide and Reference, Release 10g'.

Ohsawa, Y. & Sakauchi, M. (1983), 'Bd-tree: A new n-dimensional data structure with efficient dynamic characteristics', *Proceedings of the Ninth World Computer Congress, IFIP* pp. 539–544.

Ooi, B. C., Tan, K.-L., Yu, C. & Bressan, S. (2000), Indexing the Edges - A Simple and Yet Efficient Approach to High-Dimensional Indexing, *in* 'Symposium on Principles of Database Systems', pp. 166–174.

Orlandic, R. & Yu, B. (2002), 'A retrieval technique for high-dimensional data and partially specified queries', *Data Knowl. Eng.* **42**(1), 1–21.

Rudolf Bayer and Volker Markl (1998), The UB-Tree: Performance of Multidimensional Range Queries, Technical report, Institute for Informatik, TU Muenchen.

Samet, H. (1984), 'The quadtree and related hierarchical data structures', *ACM Computing Surveys* **16**(2), 187–260.

Stantic, B., Terry, J., Topor, R. W. & Sattar, A. (2010), Indexing Temporal Data with Virtual Structure, *in* 'Advances in Databases and Information Systems - ADBIS', pp. 591–594.

Stantic, B., Topor, R. W., Terry, J. & Sattar, A. (2010), 'Advanced indexing technique for temporal data', *Comput. Sci. Inf. Syst.* **7**(4), 679–703.

Stockinger, K., Wu, K. & Shoshani, A. (2002), Strategies for processing ad hoc queries on large data warehouses, *in* 'DOLAP '02: Proceedings of the 5th ACM international workshop on Data Warehousing and OLAP', pp. 72–79.

Terry, J., Stantic, B., Terenziani, P. & Sattar, A. (2011), Variable Granularity Space filling Curve for Indexing Multidimensional Data, *in* 'Advances in Databases and Information Systems - ADBIS, (to-appear)'.

Wang, F. & Liu, P. (2005), Temporal management of rfid data, *in* 'VLDB '05: Proceedings of the 31st international conference on Very large data bases', VLDB Endowment, pp. 1128–1139.

Weber, R., Schek, H. J. & Blott, S. (1998), A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces, *in* 'VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases', pp. 194–205.

Zhang, R., Ooi, B. C. & Tan, K.-L. (2004), Making the Pyramid Technique Robust to Query Types and Workloads, *in* 'ICDE '04: Proceedings of the 20th International Conference on Data Engineering'.