Scalable Dataspace Construction

Shibwabo K. Bernard, Wanyembi N. Gregory, Ateya L. Ismail and Omwenga O. Vincent

Abstract— This paper proposes the design and implementation of scalable dataspaces based on efficient data structures. Dataspaces are often likely to exhibit a multidimensional structure due to the unpredictable neighbour relationship between participants coupled by the continuous exponential growth of data. Layered range trees are incorporated to the proposed solution as multidimensional binary trees which are used to perform d-dimensional orthogonal range indexing and searching. Furthermore, the solution is readily extensible to multiple dimensions, raising the possibility of volume searches and even extension to attribute space. We begin by a study of the important literature and dataspace designs. A scalable design and implementation is further presented. Finally, we conduct experimental evaluation to illustrate the finer performance of proposed techniques. The design of a scalable dataspace is important in order to bridge the gap resulting from the lack of coexistence of data entities in the spatial domain as a key milestone towards pay-as-you-go systems integration.

Index Terms— Dataspaces, Machine learning, Systems integration, Spatial databases, Range Trees, Scalability.

1 INTRODUCTION

As the amount and complexity of structured and non-structured data increases in a variety of applications, there is a growing need to make available a unified approach of managing data that is contained in existing heterogeneous application data sources. One modern way of managing such heterogeneous data is through dataspaces, which are an abstraction in data management that aim to overcome some of the problems encountered in data integration system. They provide a powerful abstraction for accessing and managing data that resides in divergent data sources. Dataspaces have been proposed [1], [2] as a more appropriate way to provide a co-existing system of heterogeneous data. Further, the importance of dataspace systems has already been acknowledged and emphasized in handling heterogeneous data [3], [4], [5], [6], [7], [8]. In fact, examples of interesting dataspaces are currently prevalent, particularly on the Web [9], which include Google Base and Wikipedia.

Dataspaces are characterized by various aspects. First, dataspaces must manage all the data that exist in a space. Second, we expect that dataspaces consist of heterogeneous data and applications that could possibly be unstructured. Third, users of dataspaces need best-effort services without the concerns for setup time. Finally, the dataspace support platform (DSSP) which constitutes the infrastructure that manages a dataspace, does not have full control over the data unlike database management systems. They only have access to the data. Dataspaces therefore, provide data co-existence and not really data integration which implies that all these services must be provided without the need for semantic mappings as prevalent in other data integration approaches.

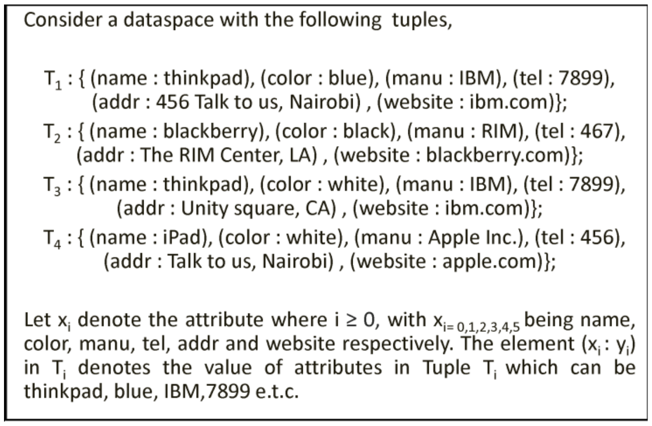
Dataspaces can be categorized in the same way as information tends to be classified. Information is often categorized in relation to various characteristics. Two possible interrelated characteristics for this categorization are access and control. We have Personal Information, Group Information or Public Information. The same categorization can be applied to dataspaces as Personal Dataspaces for Personal Information Management which provide easy access and updates of all of the information existing on a user's desktop, Group Dataspaces for an organization or group, Public Dataspaces for the global audience.

Personal dataspace is defined by [10] as a dataspace in which users interact with a set of personal data repositories. These repositories may be such as private file systems existing on a user’s desktop and private emails of a specific user. In personal dataspaces just like in other dataspaces, users have a hard time understanding which items spread across their sources are related to each other in the same context. While users may search their data sources with search engines, the results returned by these systems are not enriched with contextual information. Users may want to access all other versions of a given file that exist in their dataspace, see files and emails worked on around the same time, or retrieve emails in the same project of a given document.

A dataspace available as a public dataspace also called Google Base is described by [9] as a very large, self-describing, semi-structured, heterogeneous database. This database consists of a set of tuples with attribute values where each entry T_i (see Fig. 1 for illustration) is considered to consist of several attributes with corresponding values and can be regarded as a tuple in an existing dataspace. Due to the heterogeneity of data, which are contributed by users around the world, the data set is extremely sparse.

- *K.B. Shibwabo is with the Faculty of Information Technology, Strathmore University. E-mail: bshibwabo@ strathmore.edu.*
- *N.G. Wanyembi is with the Department of Computer Science, Masinde Muliro University of Science and Technology. E-mail: g.wanyembi@gmail.com.*
- *L.I. Ateya is with the Faculty of Information Technology, Strathmore University. E-mail: iateya@ strathmore.edu.*
- *O.V. Omwenga is with the Faculty of Information Technology, Strathmore University. E-mail: vomwenga@ strathmore.edu.*

Manuscript received (insert date of submission if desired). Please note that all acknowledgments should be placed at the end of the paper, before the bibliography.



Consider a dataspace with the following tuples,

T_1 : { (name : thinkpad), (color : blue), (manu : IBM), (tel : 7899), (addr : 456 Talk to us, Nairobi) , (website : ibm.com)};
T_2 : { (name : blackberry), (color : black), (manu : RIM), (tel : 467), (addr : The RIM Center, LA) , (website : blackberry.com)};
T_3 : { (name : thinkpad), (color : white), (manu : IBM), (tel : 7899), (addr : Unity square, CA) , (website : ibm.com)};
T_4 : { (name : iPad), (color : white), (manu : Apple Inc.), (tel : 456), (addr : Talk to us, Nairobi) , (website : apple.com)};

Let x_i denote the attribute where $i \geq 0$, with $x_{i=0,1,2,3,4,5}$ being name, color, manu, tel, addr and website respectively. The element $(x_i : y_i)$ in T_i denotes the value of attributes in Tuple T_i which can be thinkpad, blue, IBM,7899 e.t.c.

Fig. 1. An Example of Dataspace

Also notable in Fig. 1 is that there is need to recognize the attribute associations in a dataspace, this way it can be said that the keywords in attributes together with associations are neighbors in schema level. For example, keywords, 456 in attributes tel and addr are neighbor keywords in T_1 and T_4, since there is apparent correspondence between the attributes tel and addr. Therefore, a query with keyword neighborhood in schema level is supposed to not only search for the keywords in the set of attributes specified in the query, but also match the neighbor keywords in the attributes with correspondences. For example, a query predicate (tel : 456) should search keyword 456 in both the attributes tel and addr, according to the correspondence between tel and addr [9]. The process of obtaining information from the available sources can be achieved by the application of information extraction techniques [11].

A third example of dataspaces also considered as a public dataspace is observed from Wikipedia where each available article typically has a tuple with a set of attributes and values that describe the fundamental structured information of the entry [9]. For instance, a tuple describing Justy Abuti may contain typically attributes like (Born: Nairobi Kenya 1995), (age: 18 years), (Likes: swimming) . . .}. Once more, the attributes of tuples in diverse entries are various, while each tuple may only contain a limited number of attributes. Thereby, all these tuples from heterogeneous sources form a huge dataspace in Wikipedia [9].

Dataspace systems have been envisioned [1] to be an effective technique for data management which extends the services that were traditionally offered by legacy systems which included data integration and data exchange. It is further considered that the goal of dataspace support systems is to provide base functionality over all data repositories, in spite of how integrated they are.

The introduction of dataspaces addresses the assumption by traditional data integration approaches that there exists intimate familiarity in semantics of the available data sources which actually does not hold in practice. It is therefore not necessary to have upfront data integration and a better strategy should be able to allow for users and administrators to decide whether to invest in identifying semantic relationships or not [1], [12], [13]. Therefore, dataspaces present a data co-existence approach that emphasizes on providing base functionality over all data sources; regardless of how integrated they are, in an incremental fashion.

Furthermore, due to the potential growth of data, a scalable dataspace would be ideal in handling heterogeneous data sources. The growth of the dataspace should be properly managed through the provision of efficient and effective information management techniques. The development of scalable dataspaces would require a scalable algorithm that would maximize on the heterogeneity in dataspaces.

The aspect of constructing a scalable dataspace requires the development and usage of scalable data structures and algorithms. The concept of scalability in data structures was initially identified [14] through the approach of Range Tree. It is considered that Range trees are better than quad-trees and k-d trees, whose application almost require that they be implemented in random access memory, since the node sizes are smaller than any common physical disk storage device data block, and are therefore inefficient for disk resident indices [14]. He also notes that K-D-B trees cannot index data elements of finite spatial extent. Furthermore, Range trees have been proposed as the only indices in current use that are readily extensible to more than two dimensions for special applications [14].

Franklin, Halevy and Maier [2] describe dataspaces as consisting of participants and relationships (see Fig. 2). A dataspace should contain all of the information relevant to a particular organization regardless of its format and location, and model a rich collection of relationships between data repositories. The components of a dataspace support infrastructure interact to provide search and query over the Range Tree implementation.

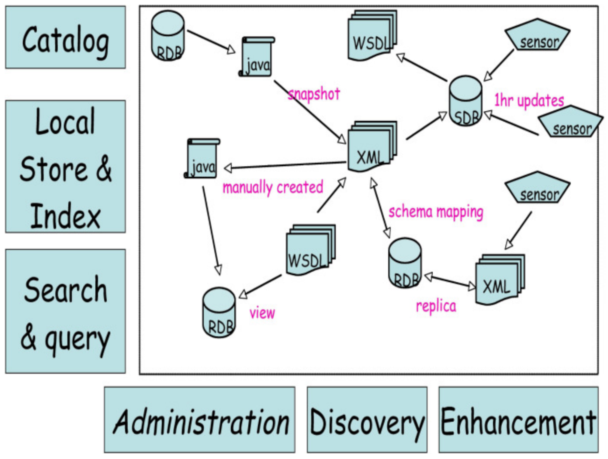


Fig. 2. An Example Dataspace and the Components of a Dataspace System.

Franklin, Halevy and Maier [1] summarizes, the distinguishing properties of dataspace systems to be:

1. A DSSP must deal with data and applications in a wide variety of formats accessible through many systems with different interfaces. All the data and applications must be supported through a com-

mon interface as an abstraction of the underlying differences.

2. A DSSP is not in full control of its data. Although a DSSP offers an integrated means of interacting with the dataspace, often the same data may also be accessible and modifiable through an interface native to the system hosting the data. Database Management Systems (DBMS) on the other hand usually have full control of their data.
3. Queries to a DSSP may offer varying levels of service, and in some cases may return best-effort or approximate answers. For example, when individual data sources are unavailable, a DSSP may be capable of producing the best results it can, using the data accessible to it at the time of the query.
4. A DSSP must offer the tools to create tighter integration of data in the space as necessary.

1.1 Contribution

This paper endeavors to extensively address the challenge of scalability on dataspaces from the standpoint of a range tree. Special consideration is given towards the design and implementation of a dataspace coupled with the fact that data volumes grow exponentially. Following the contributions made by Blackwell [14], the range tree is utilized in this study. The proposed approach is suitable for representing both structured and semi-structured data. The world is short of a global dataspace and so is the case of large corporations. Fulfilling this need is important in order to ensure transparent access to all the available data. This paper addresses this need extensively. Our main contributions in this paper are summarized by:

1. We study the representation of important components of a dataspace. Dataspaces are known to consist of participants and their relationships [1], [2], [3], [4]. Modelling these components for extensibility requires a thorough examination of each of these components. We provide a scalable way of representing participants as well as further work on representing the relationships.
2. We determine and examine the characteristics of a dataspace. Important aspects which are critical to the design and implementation of a scalable dataspace are identified and applied to the solution.
3. We critically examine and introduce a new categorization of dataspaces that improves on the existing dataspace classification that has been found to be insufficient.
4. We explore an application of the range tree which is a special data structure that is best suited to address the current challenges in storage processing. This data structure has not been fully utilized in most common applications, and therefore, we demonstrate how useful and necessary this data structure can be.
5. We propose a technique of designing dataspace support systems with key scalability and access concerns being addressed. In order to achieve this, we map the dataspace components and characteristics to real data structures and associate the appropriate algorithms to the process. As a starting point, we evaluate the existing attempts to construct a dataspace support system and summarize the challenges with more specifically in relation to scalability.
6. We report an extensive experimental evaluation. Both the performance and completeness of dataspace support systems are evaluated. More importantly, the search and query as well as space complexity are experimented in order to guarantee a more reasonable solution. Our approach which combines the appropriate data structures and algorithms together can always achieve the best performance and scales well under large data sizes. Moreover, the experimental results also confirm our conclusions of cost analysis, that is, we can improve the query performance by deploying the use of range trees. Modern techniques like fractional cascading have been confirmed to reinforce the performance.

The remainder of this paper is organized as follows: first, in Section 2 we present background material that is used in subsequent sections. Section 3 develops the design and implementation of a dataspace support system using scalability parameters. In Section 4, we present reports on extensive experimental evaluation of the proposed techniques. We also prove that the proposed techniques are optimal. We discuss the related work in Section 5. Finally, we recap, summarize, and describe directions for future work in section 6 as a conclusion to this paper.

2 PRELIMINARIES

This section introduces some preliminary settings of existing literature and relates them to the purpose of this paper. The settings include the data model, the search and query model and Participant associations. Definitions, notations, and background are further provided as prerequisites for subsequent sections.

2.1 Data Model

A data model is an abstract model that describes how data are represented and accessed. Definition 2.1 provides the data representation as a starting point for representing the data fetched into the dataspace from the various data repositories. For the purpose of this paper, a set is a collection of distinct yet related objects (further presented as nodes), considered as an object in its own right.

Definition 2.1 (Data Representation). *The data in the dataspace is defined by a set D represented by a range tree which is consists of v points in d-space. Each node in a range tree is a set of attribute-value pair. Each value can take any form of data type. The dimensions of any D or a sub-tree of D is denoted as d. A key assumption is that no* *two or more points in the set D have equal coordinates in any dimension at the same time. The data range or range tree is denoted as R and $R \in D$, Such that $R \subseteq D$ (v_d), for which $(v_d) \subseteq N$, where N is a set of dataspace nodes i.e. $N \subseteq D$.*

For example, when representing a database as our dataspace D, the number of records is represented by v while the number of fields is represented by d.

2.2 Search and Query Model

We consider queries with a set of attribute and keyword predicates of which the query inputs can resemble tuples in data repositories. For the purposes of this paper, we will use the following simple keyword and path language.

Definition 2.2 (Query Model). *Queries are typically considered to run through an axis-aligned range. A query Q is typically an expression that selects a set Q (N) $\subseteq$ D. The query inputs are typically n points in d dimensions.*

As discussed by [10], it is expected that a Keyword query expression should return the set containing nodes such that the keyword exists in any of their attribute-value pairs. Given an attribute X_i, an operator op and a value dt, an attribute-value expression on the other hand denoted X_i op dt should return the set containing nodes such that the condition on the attribute X_i with operator op and value dt is true.

It is important to describe the query building process as a critical component of the query model. An example to demonstrate the process of building queries can be presented as to consider a two dimensional (*2D*) plane with X and Y as the axis. Assuming a range tree can be build from the plane, queries executed on the range tree for the plane will constitute the set of coordinate values given by $[x_i, x_j]$ X $[y_i, y_j]$ where x_i and x_j denote values on the X axis while y_i and y_j denote values on the Y axis. Combining these four values gives us the range. The execution of these queries will use the following approach: execute a search for $[x_i, x_j]$ first on the main level tree; then for each node v belonging to the total $O(\log v)$ nodes that constitute $[x_i, x_j]$, execute a total of two binary searches for the points y_i and y_j that are inside $Y(v)$ in order to find the adjacent sequence of points that are contained inside the range $[y_i, y_j]$. We can then report or count all these points.

2.3 Participant Associations

Entities existing in a dataspace are called participants. The relationships between participants are defined by the associations. An association in this case is therefore, a mapping between entities in a dataspace. The design and implementation of a DSSP requires the definition and modeling of these associations. By using the range tree, it is possible to represent each entity by a node. Recall from definition 2.1 that, a specific node is denoted v_i, which also represents an entity. Every entity is characterized by a set of attributes used to build associations using schema mapping techniques defined in [15]. The attribute correspondence for existing entities in dataspaces has been described to be developed in a pay-as-you-go fashion [6] incrementally. One way to develop this is through learning based on feedback from users.

Let $X_{d\text{-}i}$ be an attribute for any node v where $d \geq 1$for all values of X. Ideally, attributes are identified by an index beginning from zero to $d\text{-}1$. Given another attribute $X_{d\text{-}j}$ and a keyword $w(X_{d\text{-}i,}, X_{d\text{-}j})$ that is used to associate the attributes $X_{d\text{-}i}$ and $X_{d\text{-}j}$ we proceed to express the matching between the two attributes as $X_{d\text{-}i} \leftrightarrow X_{d\text{-}j.}$ Any keywords $w(X_{d\text{-}i,}, X_{d\text{-}j})$ occurring in $X_{d\text{-}i,}$ $X_{d\text{-}j}$ are said to be neighbors.

2.4 Indexing

An index is a list of data that is stored typically in plain text format for easy scanning by a search algorithm. With indexes, searches are done via the index instead of reading through all the data since indexes contain metadata or keywords. Indexing therefore is a technique that uses indexes to speed up searching and sorting operations on a set of data referenced by the index.

Indexing is an important aspect of any dataspace system as earlier indicated in Fig. 2. This requires the development and usage of indexing structures. In order to facilitate indexing, we adopt structures that are used in modern search engines [16].

The proposed approach will make use of the inverted index, also known as inverted files or inverted lists [16], [17], [18] which are described as a mapping from keyword $w(X_{d\text{-}i,}, X_{d\text{-}j})$ to the list of node identifiers of nodes containing that keyword. For each item in the inverted lists, we maintain a pointer denoted p that associates to a specific list of tuples, where the item occurs. For the purpose of supporting both attribute-value and keyword expressions, it is possible to implement the inverted index by concatenating keywords with the attributes in which those keywords occur. Keyword expressions are then later converted to prefix queries [19]. The keywords can constitute any data type with a total order including floats, strings, dates and integers.

3 Dataspace Construction with Scalability Parameters

3.1 Node Construction

The construction of a scalable dataspace requires an approach that guarantees for scalability. One of the main goals for this paper is therefore to develop an adaptive algorithm for dataspace construction that offers scalability. The range tree is a known data structure that can represent d-dimensional data in space. Various additional reasons are provided to support the importance of this data structure including the ability to implement the data structure on disk [14]. Combining the power of this structure with the need to incorporate dataspace components as proposed by [13] can lead to a higher degree of success in dataspace construction.

Critical to the design of a dataspace is the representation of participants (nodes) and their interrelationships (edges). Participants in a dataspace are called Entities and they can as well be represented as indicated in Fig. 2 based on the assumption that the basic unit of a dataspace is an entity. It is however important to note that the representation proposed in Fig. 2 is largely generic with little if any focus on the practicability of the implementation. The best approach is to define the structure of a basic participant of a dataspace. The node can be represented as consisting of the attribute and data types as shown on Table 1.

TABLE 1
DATASPACE NODE (N) ATTRIBUTES

Attribute	**Data Type**
Index – *i*	**double**
Data - *dt*	**double**
TimeCreated - *t*	**Date/Time (double)**
Dimension - *d*	**integer**
Parent[] –$P[P_1,P_2....P_n]$	**double ptr[]**

An entity is therefore represented as a node in the Range tree. Each node carries the same attributes or properties as indicated on Table1. Ideally, a node will have links called pointers to other nodes. The application of nodes has been used widely on linked lists among other structures with much greater success. This approach makes use of a unique node for each entity regardless of the relationships between nodes. The uniqueness can be in any of the attributes of the node but a more efficient attribute for this is the index. The nodes contain pointers to data objects and can further be made to correspond to disk pages for disk-resident indexing.

By applying this approach of defining nodes, the range tree structure can then be used to model the relationships between dataspace participants with much greater success. The following is a description of the attributes of a dataspace node.

1. Index: Defines the location of the Node within the Range tree. This requirement is introduced and supported by Fig. 2., for traversing the dataspace and to perform further operations like search and query.
2. Data: Describes the actual set of value(s) that a node contains.
3. Time created: Refers to a timestamp (t) which serves the purpose to indicate time zero for a dataspace node.
4. Dimension: Describes the various perspectives that a node in the dataspace may possess or lead to.
5. Parent: Is a set of nodes that a particular node inherits attributes from. In more direct terms, these are nodes that have association with a particular node as described by w_i occurring in A_i, B_i, given that A and B are schemas.

3.2 Constructing Participant Relationships

Participants in a dataspace are related in one way or the other. The relationship can exist in one of three ways: one way, two ways or transitive. Traversing the dataspace in either direction over a set of participants requires a degree of trust. In order to ensure that the general dataspace relationship aspect is constructed in a more practical sense, it is possible to design a model that combines entities and their relationships considering the security component. Fig. 3 presents a proposed model that has a security element as a special consideration. A set of three nodes are used for demonstration purposes. The relationship between any two nodes v_i and v_j is logical as opposed to physical.

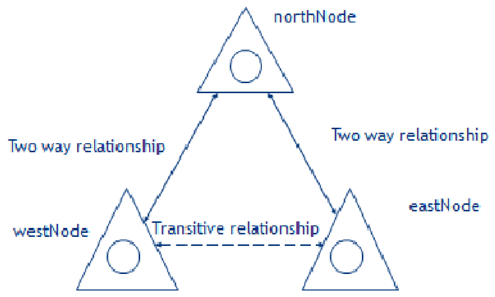


Fig. 3. Dataspace Participant Relationships.

The representation of participant relationship as proposed in Fig. 3 is necessary because it determines the ability of any two nodes v_i and v_j to exchange messages. The same behavior is exhibited with files and classes whereby files that contain defined classed may exist within the same directory but there are defined restrictions within each class using the private, public keywords.

It can be said that a node e.g. $v_{northNode}$ belonging to a dataspace is an element of another node e.g. northNode. This aspect is common in a multidimensional scenario. The corresponding expression is represented next.

Given: $v_{northNode} \in \text{northNode}$

$v_{eastNode} \in \text{eastNode}$

$v_{westNode} \in \text{westNode}$

By using the model presented in Fig. 3, it can be stated that the relationship between any two nodes $v_{westNode}$ and $v_{eastNode}$ referred to as $v_{westNode}$ <---> $v_{eastNode}$ is transitive if and only if there exists some commonality in attribute/value combination between the set of nodes {$v_{westNode}$ and $v_{northNode}$} and {$v_{northNode}$ and $v_{eastNode}$} in such a way that a mapping can henceforth be established between $v_{westNode}$ and $v_{eastNode}$ through $v_{eastNode}$ This combination is such that:

Given:

$dt_{westNode} \in v_{westNode}$ and,

$dt_{eastNode} \in v_{eastNode}$ and,

$dt_{northNode} \in v_{northNode}$ then,

$dt_{westNode} \leftrightarrow dt_{northNode}$
$dt_{northNode} \leftrightarrow dt_{eastNode}$

Where, dt represents the attribute for an arbitrary node and $\leftrightarrow$ denotes an existing mapping due to the commonality.

It is important to observe that the relationship can take any direction. It is expected that two separate nodes may be similar in all aspects except in two key aspects. These aspects are node attributes called time and space. Time refers to the timestamp that the node started to exist. Space on the other hand is the address or location in which the entity (node) exists. There is the possibility of the entity changing its location, and therefore, we need to allow for location updates. The start time of a node on the other hand is considered as fixed for the entire lifetime of the dataspace.

The use of directional notations in Fig. 3 is not meaningless but rather is a clear indication of the fact that a node may be related to any other nodes on the Euclidean space of d dimensions where $0 <= d < \infty$. This explains the need to use a data structure that can handle arbitrary n dimensions. The range tree is known to be capable of handling arbitrary dimensions of data representations. A Range Tree is also appropriate for an implementation of a dataspace due to its suitability for disk storage as opposed to a structure that is memory based. This guarantees a higher degree of scalability considering the fact that the dataspace could grow to a single global dataspace. We map the representation in Fig. 3 to a new model in Fig. 4.

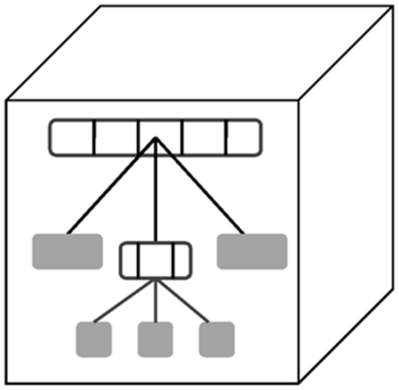


Fig. 4. Dataspace Entity Association in d-Dimensional Eucleudian Space.

A key observation on Fig. 3 is that it is the relationships between dataspace participants that determine the dimensions of the dataspace. For a dataspace support platform, the total number of dimensions is often expected to be unpredictable. This becomes even much complicated by the learning aspect. As relationships encode data semantics, sharp improvements in precision and recall may be achieved through high-quality relationships. Possible approaches that could be employed to obtain those relationships including machine learning techniques.

3.3 Dataspace Implementation

Following the definitions, further analysis and design components described earlier, an implementation of the proposed dataspace is achieved using the range tree as the container representing the set D. Range trees are typically an extension of a one-dimensional data structure called the segment tree. The segment tree is a data structure for intervals on the real line whose extremes belong to a fixed set of an arbitrary number of abscissae. With the range tree, we have the capability to pre-process the data to build a dataspace that allows efficient resolution of dataspace queries earlier defined as $Q\ (N)$. The build algorithm is very comparable to merge-sort. For the construction of the tree representing the dataspace or a dataspace subset $N_d \subset D$, we need to first sort the points. Each point or coordinate on the tree represents the dataspace participant on the constructed tree as it was earlier described as a node (v).

The points being the participants in a dataspace are sorted with respect to the first node and build recursively (from top to bottom) the main tree in linear time. For the associative trees we need not to sort the participants again. We now build the associative trees in bottom-up fashion. Every node merges the sorted lists of its children in linear time starting from the leaves which are trivially sorted.

Kd-trees, typically have $O(\sqrt{n}+k)$ query time, where k is the number of reported points and n is a set of points in the plane. Therefore, in cases where the number of reported points is minimal, the query time is relatively high [20]. The range tree has a better query time, namely $O(\log^2 n+k)$. The cost for this improvement is an increase in storage from $O(n)$ for kd-trees to $O(n \log n)$ for range trees. A technique called fractional cascading can be applied to reduce the query time for a range tree. This technique effectively causes a significant reduction in query time on range trees to $O(\log n + k)$ [21].

The range trees answer a d-dimensional range query in time $O(\log^d n+k)$, where n is the whole set of points and k is the set of reported points. The construction time and the space the tree consume are $O(n\log^{d-1} n)$. The optimal solution to the orthogonal range search problem is proposed by a structure with time complexity $O(\log^c n + k)$ and $O(n(\log n/ \log \log n)^{d-1})$ space consumption, where c is a constant [20].

The updates, insertion and deletion of nodes to the actual tree are ultimately implemented on disk rather than on memory, this enhances the capacity for managing scalability. The approach to access and retrieval is similar to the case of the current computer file system with the consideration of the aspect of multi-dimensionality.

3.4 Sample Dataspace Implementation

During the implementing of the dataspace support system based on the proposed data structures, we considered that the key bottleneck in query performance is not the time for processing but rather the key bottleneck is memory, bandwidth and latency. Therefore, it is possible to hit the memory wall [21] in our application if we are not careful, since it is expected that the disproportional increase between memory and processors is wider today than it was the case three years ago. It has been found that there exists a disproportional increase between memory and processors whereby processor speed has been improving at a significantly higher rate as compared to memory [21].

Our implementation consists of an application that interacts with a database as the dataset which manages a set of records representing the points (V). We expect a similar architecture regardless of whether the data is randomly queried from the dataset or from a geographical dataset. The following code segment illustrates a typical implementation of the dataspace in the C++ language. The Range tree based dataspace is implemented using binary search trees. It assumes no two points have the same x or y coordinate. The example could be made more efficient by assuming points are given all at once. Then could sort by x and by y to ensure tree are perfectly balanced. It is important to note that this implementation is based on disk.

We recommend that dataspace implementation should make use of a powerful programming like C or C++. The sample code provided is modified in order to be more easily understood by object oriented programming language experts.

```
class Dataspace<Key public Comparable<Key>>  {

   private:
           Node root;   // root of the primary BST

  public:
   // BST helper node data type
   class Node {
      Key x, y;                    // x- and y- coordinates
      Node left, right;          // Left and right subtrees
      RangeSearch<Key, Key> bst;  // Secondary BST
      double i;                    // Index
      double t;
      double TimeCreated; // Time created

      Node(Key x, Key y) { // Node Constructor
         this.x   = x;            // x value initialization
         this.y   = y;            // y value initialization
         this.bst = new RangeSearch<Key, Key>();
         this.bst.put(y, x);   // update the tree
      }
   }
}
```

3.5 Improving Query Processing Time

A critical aspect to the design and implementation of a scalable dataspace support platform is the requirement to develop techniques to improve the time taken to process queries. A DSSP will typically process requests that are presented to it in form of queries. The initial stage that requires query processing is building the DSSP.

One way of improving query processing time is through parallelism. The build algorithm for a range tree is very similar to the operation of the merge sort; it is therefore expected that it can be easily parallelized. In order to achieve this, there is the requirement to first sort the available nodes (points). It is considerably practical to either use a specialized algorithm or for a relatively small number of cores there is the possibility of just breaking up the existing data into smaller portions, sort the portions on multiple cores and finally merge the results [22].

In case we have two cores, we break up the data into two smaller portions; in practice, this has resulted in a 42% reduction in running time [22]. The remainder (and a large amount) of the pre-processing time is spent within the recursive build function; an incredibly practical and more efficient way to parallelize the build function is to execute the two recursive calls in multiple threads. This only needs to be done at the top few levels of the recursion (for dual-core, only the first level is adequate). The effect is a general improvement in the query processing time that varies depending on how complex the merge operation is.

The other available strategy for improve query processing time for our approach is through the application of fractional cascading technique. This technique is known to speed up a sequence of binary searches for the same value in a sequence of related data structures. It works by taking logarithmic time duration for the first binary search which is often the case for standard binary searches but any subsequent searches end up taking less time [20].

With the absence of fractional cascading, it is therefore expected that queries that the baseline would answer a d-dimensional range query in time $O(\log^d n + k)$, where n is the whole set of points and k is the set of all reported points. The measured construction time and the space that the tree consume are $O(n \log^{d-1} n)$. By using the technique of fractional cascading, we can gain by a log n factor in the last level of the tree and the result is an improved time complexity of the order $O(\log^{d-1} n + k)$ [20].

4 Experiments

In this section, we evaluate the query processing techniques of the proposed approach. The main goal of the experiments is to understand how our approach compares to the other approaches presented in the reviewed literature. We will evaluate the query performance costs of as we scale on the number of inputs.

4.1 Setup and Datasets

We base the implementation on the Combined Algorithm [23] with the range tree in order to build our dataspace on the available disk. The main evaluation criterion during experimentation is query time cost. The other criteria for evaluation include storage cost and build time. We conduct the experiments in two common data repositories PostgreSQL, and MySQL to test the results on various underlying data models.

MySQL is known to be the most popular open source database. It is the database for the web. MySQL fully supports partial indexing through the use of the InnoDB engine, but not with the MyISAM engine. The MySQL architecture provides Pluggable Storage Engines which enable MySQL to assume the nature of a variety of different databases [24], [25], [26]. PostgreSQL on the other hand is known to be the world's most advanced open source database. It is described as the open source Oracle. It has a very strong security model, provides high levels of flexibility for programming, and generally exceptional OLTP performance as well as scalability [24], [25], [26], [27] all these provided without the need for heavy tuning and workarounds.

In order to establish and manage connections between the DSSP and its underlying data repositories for the implementation, the ODBC client connectors are used. Each of the separate data repository (MyISAM, InnoDB, PostgreSQL) continue to manage its own data storage mechanisms without any modifications including Spatial Index support, sub-queries as well as full text search. Additional connectors supported by the target repositories have been found to be JDBC, .NET, and C++ Client Connectors [27].

For the purpose of obtaining an accurate comparison in performance, a very critical consideration has been to pay special attention to configuration and environment. This is supported by the fact that, in terms of performance, MySQL has been found to do better with simple queries and 2-core machines [27]. PostgreSQL on the other hand performs better with complex queries and multi-core machines. Therefore, it is expected that a dataspace support system will present varying performance levels depending on the target repository. It is possible to optimize database management systems according to the environment in which they have been installed.

The set of the global dataspace *D* is used to represent the data contained on all the described data repositories combined. In this respect, our goal is not to compare performances of databases but the performance of the dataspace support system. PostgreSQL and MySQL both fundamentally make use of various mechanisms to improve performance at the basic level which are generally not compared in this paper.

The actual version of PostgreSQL is 9.1 while the version of MySQL used is 5.5.24. These two repositories are installed on a single machine with each one of them having a database created for the purpose of testing. Each separate database further is comprised of a table consisting of 10.1 million records in two columns (fields or attributes). The two existing fields are pinkness and roundness.

Each and every node (v) in the dataspace *D* is used to represent objects in a repository. Each object in every database has *m* scores, one for each of (*m=2)* attributes. A good example is, an object can have a color score that describes how pink it is, and a shape grade, that describes how round it actually is. The values of the scores are randomly generated using the standard SQL random function and populated to the tables using and during an insert statement. The interaction with the database is enforced via a common web portal that facilitates query execution through the algorithm. The following is a query that is executed to create each table in the separate databases:

```
create table dataspace1 (id int,
                              pinkness INT,
                              roundess INT);
```

The attribute *id* is the primary key for the test table used to maintain the requirements at the database level including indexing. The other fields are populated using the random function as follows:

```
insert into dataspace1(pinkness, roundness
values(FLOOR(1 + RAND() * 2000199), FLOOR(1 + RAND() * 2000199) );
```

The process of populating the values to the database by running the indicated query is performed iteratively in order to generate and populate a total of 5.1 million records. *RAND()* will result to a value like 0.747969849. It basically creates a number in the interval [0;1) (meaning it excludes 1). So *1 * RAND()* yields a number in *[0, 1)*. *2000199 * RAND() + 1* is in *[1;2000199)*. By rounding down we ensure that the resulting integer is within the range *[1;2000199]*.

We associate objects of attribute-keywords with tf*idf weight scores [28] as an aggregation function (scoring function), that combines the pinkness score and the roundness score to obtain an overall score. During the evaluation process, we make use of the Structured Query Language (SQL) random function in order to randomly select 1,000,000 records (tuples) from each data set using instance-level matching as the workload of queries. The mean response time of the executed queries are recorded and presented when the approach is applied on different repositories.

All experiments have been run on a Intel Core 2 Duo CPU (1.5 GHz) T5250 server, with 2 GB of RAM (DDR2 SO-DIMM), and a 160 GB SATA150, 2.5″ 5400 rpm hard disk with 8 MB cache. We run the tests under Windows 7 Ultimate 32 bit, using Microsoft Visual C++ 2008. In the implementation, we have not utilized the aspect of multi-threaded parallelism for querying, and therefore, our code makes explicit usage of only a single core; for the purpose of indexing.

4.2 Query Processing

This section proceeds to define a query plan as an important step for query processing. Given a query Q, with an example described as find the record with the pinkness between 590 and 1000 and roundness between 200 and 300. We have denoted this query as *Q (N)* and it is expected to return a view or a set of views from the set *D* if executed severally. We represent this query as a range query shown in Fig. 5.

Based on the x-coordinate or y-coordinate values, the resulting query range denoted as $[x_i, x_j]$ X $[y_i, y_j]$ where y represents pinkness while x represents roundness becomes:

$[(x_i \mid -\infty) : (x_j \mid +\infty)]$ X $[(y_i \mid -\infty) : (y_j \mid +\infty)]$

Every point v_i is identified by $(v_x,v_y) \in [x_i : x_j]$ X $[y_i : y_j]$ iff

$((v_x \mid v_y) , (v_y \mid v_x)) \in [(x_i \mid -\infty) : (x_j \mid +\infty)]$ X $[(y_i \mid -\infty) : (y_j \mid +\infty)]$

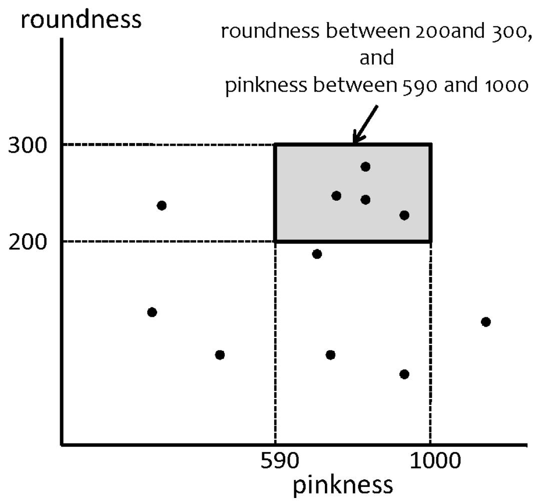

roundness
roundness between 200and 300, and pinkness between 590 and 1000
300
200
590
1000
pinkness

Fig. 5. A Range Query on the two-dimensional axes.

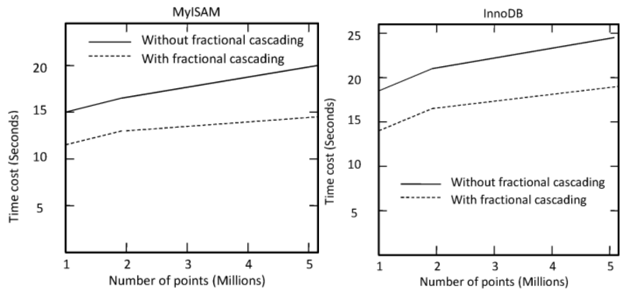


Fig. 6. A Range Query on the two-dimensional axes.

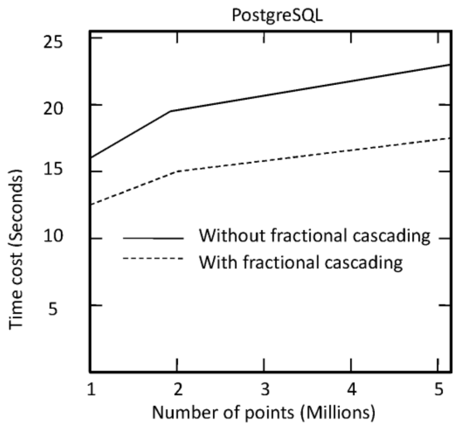


Fig. 7. A Range Query on the two-dimensional axes.

Any query *Q (N)* which is executed on the repository is expected to return a null result denoted as *R* or a subset of *D* where *D* is the set of data in a dataspace. The result is represented by the expression as $R \subseteq D$ which describes that every element in the query result *R* is a member of the dataspace *D* which may include all the nodes present in the dataspace.

4.3 Scalability

The results of the experiments in terms of scalability of a dataspace support system with regards to query response time, storage cost and build time need to be discussed. In order to achieve this, the paper presents further performance results when scaling on the data size. The results existing in this section are recorded and presented using our synthetic dataset, as it allows us to scale on the data size. We use our approach that makes use of the existing range tree construction algorithm with and without fractional cascading as the starting point. Further work can be done to include other techniques.

4.3.1 Query Response Time

We mainly test the performance of the approach under various data sizes, so as to evaluate scalability. In cases when implementing and testing the query time of the data structures and our algorithms, the O(k) factor existing in the query time is found to be dominant except when we restrict to small queries. For this purpose, during measuring the query performance of our implementation, the reporting of the query results has been disabled. Moreover, queries are confined to only counting the points existing within the query rectangle. However, the capability to additionally report the points (nodes) that do not have any overhead is always maintained. This approach can be useful in practice for scenarios where we need to report the resulting points in the query rectangle, only in cases when they are not too many. This scenario does not however eliminate the need to determine the total count. Sometimes, we may need to obtain the output, but only up to some fixed number of points. More techniques of addressing the counting problem have been described in detail by [29].

As presented in Fig. 6 and Fig. 7, the time cost of deploying the approach increases logarithmically as the data size given an original query *Q* selecting attribute values in each of our datasets. The proposed approach continues to scale well under large sizes.

A very imperative fact is that initial binary searches take less than 12% of the query time. Fractional cascading improves subsequent query time. Another way of improving the query time has been known to be caching. Caching require the use of a cache-friendly data structure. Various experiments have been done when using the range tree with caching [22] that further indicate that CSStrees should be used in place of arrays in order to facilitate caching. This paper does not include the implementation, experiments and the results of the application of caching with the main reason being that the significant proportion of the query response time is always spent on range-tree searching rather than binary searches as it has been claimed by [30].

The average query response time for MyISAM storage engine was found to be quicker than InnoDB and PostgreSQL. Since all the storage engines were installed with their default settings, it is expected that the results would be similar in other experiments. An important observation is that regardless of the underlying storage, the time taken for the same number of points tends to increase logarithmically. This has a benefit on scalability since any further addition of points after a certain threshold (About 2 million for our case) does not proportionally or substantially increase the time taken to generate a response for the query. It is not usual to expect a decrease in query response time with an increase in the number of points, and therefore, comparing our approach with any other approach that uses data structures that consume linear, for example [9], quadratic and other time expensive order yields sufficient justification for adopting the discussed approach.

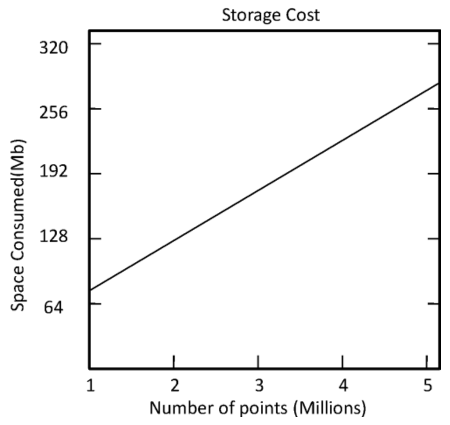


Fig. 8. A Range Query Storage Cost.

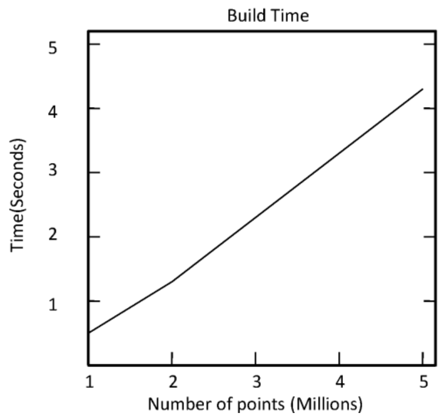


Fig. 9. A Range Query Build time.

4.3.2 Storage Cost

As presented in Fig. 8, it is observed that the storage cost tends to increase in a linear fashion from a certain point of storage. The advantage of using this data structure is that the implementation can be enforced on disk as opposed to memory. Therefore, this can be managed due to the current trends of growth on disk storage as compared to memory. Memory growth has been known to grow at a slower rate as compared to processor speeds as well as disk storage capacity. The asymptotic complexity of space for memory based implementation still needs to consume memory since the data structure needs to be loaded to memory during query processing. Therefore, the number of dimensions is determined to be the only important constraint in case the implementation is done on memory.

4.3.3 Build Time

The findings presented in Fig. 9 indicate the time taken to build the dataspace structure. This structure is ideally a tree structure that maps all the entities (points). The build time is determined to vary linearly. This result is expected in typical computing environments as default. The benefit is that this would not be done every other time. It is expected that the dataspace will typically be initially built once and then any necessary updates follow later on demand. A suggestion on this paper is have further experiments on the deployment of distributed computing techniques on this so as to share the load.

5 Related Work

5.1 Spatial Domain and Data Structures

Entities in the real world are known to exist in some space. Files and objects that exist in devices can in the same way be defined to exist in some space typically defined on memory or disk. We often have a logical mapping to any physical storage area. The process of constructing a DSSP requires defining a logical domain for various entities. This domain is called the spatial domain for dataspaces. This space therefore ranges from the scope of a single entity to the entire dataspace set.

A dataspace support system can therefore be said to manage dataspace domains. The implementation of a dataspace support system requires the development of effective and efficient techniques of defining a dataspace. These techniques are in computer science supported by algorithms and data structures that work together to achieve a common goal. Different data structures have been defined for various applications including the queue, stack, graphs and tree which are often implemented using the array and linked list among other implementations.

The unique organization of a dataspace infrastructure requires the design or selection and application of a suitable data structure that can represent the dataspace with guarantee to scalability, efficiency and practicality. A dataspace consists of entities that must be interlinked in order to have a coherent dataspace infrastructure. The definition of dataspace entities has been discussed by [31] as similar to the mathematical set.

A dataspace implementation can benefit from the developments in spatial data structures in order to make the dataspace vision a reality. Similarly, dataspace design can use some techniques that are used for spatial indexing and classification. However, it is important to point out that a dataspace should contain any information in any format over and above what is contained in geographical information systems. This demand to have anything uniformly makes dataspace design a more challenging task. Some attempts have been made in a theoretical sense to propose the design of a dataspace, however, this paper would focus on the actual low level implementation of a dataspace.

5.2 Dataspace Implementation Attempts

Various efforts have been made towards the design and possible implementation of a dataspace. The discussions in this section analyze the significant attempts and indicate the challenges that this paper intends to address. It is imperative to understand that we envision a scenario where the world can integrate into a single global web dataspace. Desktop dataspaces can still be retained as private. This vision is not to be ignored due to the fact that a fundamental principle of dataspaces is the ability to learn and provide integration on demand. With the development of appropriate integration technologies, the possibility and practicability of integration can surely become a reality.

Wikipedia and Google Base have been explored by [9] as dataspace examples and a possible integration solution provided. Our view is that dataspaces can be broader than this. Therefore, rather than implementing separate code and providing specialized solutions in search engines for each of those examples, we should preferably have a single, exhaustive, and powerful framework to model all of these different integration needs. Moreover, the cost in terms of time for processing queries has been defined by [9] to be linear. There is need to reduce this time considering the possibility that dataspace is expected to grow without limits.

Another dataspace implementation has been presented by [9]. Although the implementation clearly constructs a dataspace, we have found challenges relating to scaling. This is because the provided approach heavily depends on memory and will therefore not be practical for large dataspaces. The approach models a dataspace based on a common mathematical and programming construct called the set. In programming set variables are stored in computer memory.

Salles [19] presents novel breed of information-integration architecture that stands in between search engines and traditional information integration systems. This architecture is specified as for personal and social dataspaces. The first requirement for a Personal Dataspace Management System (PDSMS) is to offer basic query services on all the data in the data sources from the start. In a personal and social dataspace scenario, this requirement implies dealing with a highly heterogeneous collection of data (e.g., files, directories, e-mails, addresses, music) distributed among a variety of data sources (e.g., file systems, email servers, databases, web sites). A key drawback to the suggested approach is that the data structures are implemented in memory.

Additionally, the query-response time of all indexing strategies scales linearly with the number of association trails. Materializing both left-side and right-side queries as materialized views had slightly better processing times than materializing the left side as a B+-tree and the right side as a materialized view for larger number of trails, though differences are not significant.

5.3 Range Tree

A tree is a widely used data structure that consists of nodes in a hierarchical tree structure. Different implementations of the tree data structure exist each with specific domains of application. Trees provide a default data structure for the implementation of Dataspace Support Platforms due to their hierarchical nature. Shibwabo, Ateya and Wanyembi [31] design a dataspace as exhibiting this kind of a relationship as well as a possible transitive relationship that can minimize on the computational overhead.

Range trees were discovered separately by several individuals including Bentley [32], who additionally discovered K-D trees and Lueker, who further introduces the technique called fractional cascading for range trees [33]. They were first introduced as a spatial indexing strategy for multi-dimensional data by Guttman [34]. Their development was guided by the inadequacy of other indexing methods for handling data elements of finite extent and of arbitrary distribution in a space most common of two dimensions, but more generally of any number of dimensions. Competing index structures include binary trees, cell methods, quadtrees, k-d trees, and K-D-B trees. All of these suffer from one or more severe limitations in spatial data applications.

Binary trees are based on only one dimension. Even if multiple trees are built to handle more dimensions, retrieval "bands" must be intersected through sequential comparisons to find the desired data elements. All methods require specification of boundaries in advance and are hence inefficient if clustering of data elements occurs, as it commonly happens with spatial data sets [14]. An example of the Range tree Spatial Search Index is presented in Fig. 10.

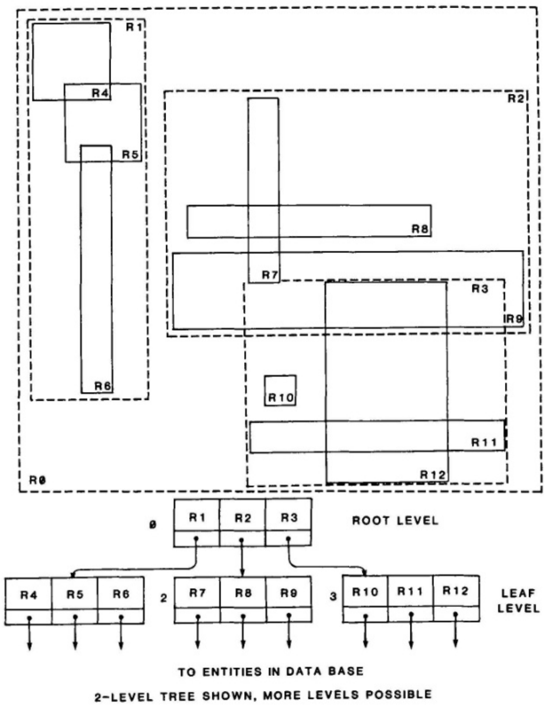


Fig. 10. Range tree Spatial Search Index (Adopted from [14]).

5.4 Fractional Cascading with Range Trees

Fractional cascading is commonly known as a technique that is usually applied to accelerate a sequence of binary searches for an identical value in a sequence of related data structures. A good introduction to orthogonal range searching, range trees and fractional cascading is described in [35, 36]. With this technique, the first binary search in the sequence takes a logarithmic time just as is the case with basic binary searches but any successive searches are considered to take faster time. Various studies have been done towards the issues surrounding the implementation of range trees with fractional cascading [20].

The range tree data structure typically answers any d-dimensional range query in time $O(\log^d n+k)$, where n is the whole set of points and k is the set of reported points. The overall construction time and the space requirement for the tree are $O(n\log^{d-1} n)$. By applying fractional cascading techniques, we can consequently gain by a log n factor in the last level of the tree. This finally leads to a resulting time complexity of $O(\log^{d-1} n+k)$. Intuitively, fractional cascading performs one binary search as opposed to two in the last level. The optimal solution to the orthogonal range search problem is contained in a proposed structure with time complexity $O(\log^c n + k)$ and $O(n(\log n/ \log \log n)^{d-1})$ space consumption, where c is a constant [20].

5.5 Range Tree Operation

A Range tree is a balanced tree structure wherein each Range Tree node contains a number of entries, and each entry consists of a pointer to a child node and the minimum bounding rectangle (MBR) of the child node. The MBR of a node is the least rectangle containing the MBR's of all its children.

Fig. 11 is an example of a portion of a GIS arc-node database. Two levels in the R-tree have special significance. There is a single node at the beginning of the tree called the root. At the end level of the tree, the nodes are called leaves and the child pointers are to database entries themselves rather than to lower level nodes in the tree. Recursive algorithms for initially populating, updating, and searching the R-tree have been well defined [33].

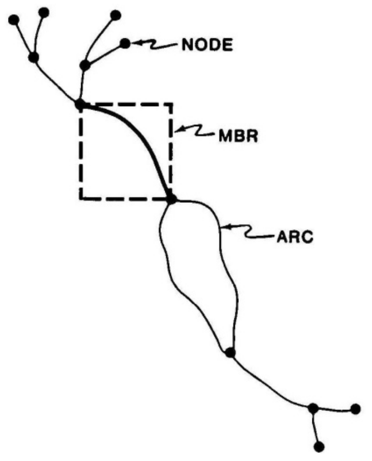


Fig. 11. Range Tree Example (Adopted from [14]).

Although, the CGAL library provides some classes for range trees there is space for optimizations in that package [22]. Firstly, there is a lack of recursive construction of d-dimensional range tree and the only way to construct a range tree of dimension d is to build a tree of dimension 1 and then make this an associative range tree of a new one which will have dimension 2. Then one must build a tree of dimension 3 with this tree as an associative tree and this technique continues until the construction of the whole d-dimensional tree.

In addition to that, the package uses virtual functions, which increases the run time and finally there is no fractional cascading. The proposed approach uses nested templates for the representation of the d-dimensional range tree which is defined in compilation time. The dimension of the tree must be a constant and defined in the compilation time. In the last level a fractional cascading structure is constructed [20].

For example a 4-dimensional range tree of size n with different kind of data at each layer is given by the following nested template definition [20].

```
LayeredRangeTree <DataClass ,
        LayeredRangeTree <DataClass ,
                    LastRangeTree <DataClass>
>
> t r e e (n) ;
```

5.6 Data Storage Architecture

A critical evaluation into the common database management systems architecture indicates that they consist of two parts: logical and physical architecture. The logical Database management system architecture manages techniques to store and present data to the users. The physical architecture on the other hand concerns more on the software building blocks to constitute the system.

Fig. 12 presents the physical architecture of a database management system. In overall, end users use the available Application Programming Interface (API) to connect to the database with different programming languages.

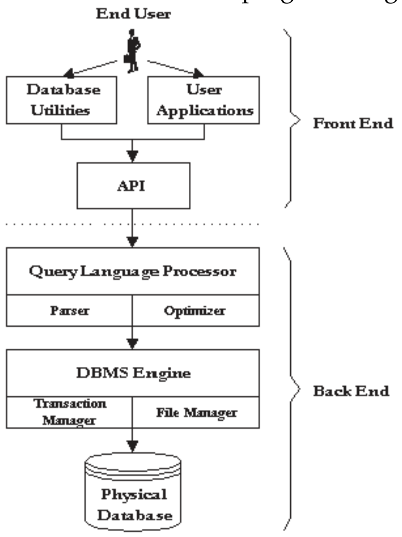


Fig. 12. Physical DBMS Architecture (Adopted from [27]).

Thereafter, data is ideally processed to the Back End through an exchange from Query Language Processor to the DBMS Engine, consequently to the Physical Database. The end users who only interact with the front end usually are not exposed to the back end services which usually run in the background.

6 Conclusion

This paper has shown, through extensive situation analysis, supporting experiments and further existing literature that data sets can coexist by making use of dataspaces. Dataspaces are considered as a new approach to data management that provides a solution to the systems integration challenge. Dataspaces however, are mainly targeted towards ensuring systems co-existence. The construction of dataspaces requires efficient, effective and scalable techniques for managing data partly because data tends to grow exponentially coupled by the continuous nature of storage. Two of the main services that a Dataspace Support Platform (DSSP) should support are search and query. While DBMSs have excelled at providing support for querying, search has emerged as a primary mechanism for end users to deal with large collections of unfamiliar data. A DSSP should enable a user to specify a search query and iteratively refine it, when appropriate, to a database-style query.

Although Range trees have been existing for decades, the application to solve real world problems has been rare due to the fact that most common applications are relatively smaller than dataspaces and geographical information systems. As complexity in data management increases, the need to design and adopt a practical approach to data management rises as well. We design and implement a dataspace support system supported by range trees. Each participant in a dataspace is described as a node that can itself be in arbitrary number of dimensions. The node is a set of attribute-value pair. We also provide a model to describe the relationships between dataspace participants. We build the associations using schema mapping techniques. The inverted indexes are used to support indexing structures as an important component of a dataspace system. Our approach is suitable for representing both structured and semi-structured data with higher degrees of practicability. An important gain for this Range trees based dataspace construction is that Range trees are in no way restricted to two-dimensional spatial indexing. They generalize readily to a space of any required random number of dimensions.

Finally, we report an extensive experiment to illustrate the performance of proposed methods. In the method of materialization, the general query plans show no worse performance than the standard query plans. When proper negative merge is applicable, the general plan can achieve better performance. The hybrid approach with both views and partitions can always achieve the best performance. Additionally, the results of our experiments also confirm our conclusions of performance analysis, that is, it is very likely that in most applications, the range-tree based implementation will not be the bottleneck; it is expected that considerable amount of time is spent in transferring the real query output or in constructing/fetching queries. The results of this paper offer an interesting example of how aspects that were trivial over time become important in designing, implementing and selecting the right data structures and algorithms.

Acknowledgment

The authors like to thank Strathmore University for the PhD Scholarship. This work was supported in part by a grant from The National Council for Science and Technology (NCST) in the Republic of Kenya grant reference number NCST/ST&I/RCD/4th Call PhD/213 under the Science, Technology and Innovations Grant fund (ST&I). The authors would like to thank NCST.

References

[1] M. Franklin, A. Halevy, and D. Maier, "Principles of dataspace systems", *Proc. of Twenty-Fifth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS 2006)*, ACM Press, p. 1-9., ISBN 1-59593-318-2.

[2] M.J. Franklin, A.Y. Halevy, and D. Maier, "From Databases to Dataspaces: A New Abstraction for Information Management," *SIGMOD Record*, vol. 34, no. 4, pp. 27-33, 2005.

[3] A.Y. Halevy, M.J. Franklin, and D. Maier, "Principles of Dataspace Systems," *Proc. 25th ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems (PODS '06)*, pp. 1-9, 2006.

[4] M.J. Franklin, A.Y. Halevy, and D. Maier, "A First Tutorial on Dataspaces," *Proc. VLDB Endowment*, vol. 1, no. 2, pp. 1516-1517, 2008.

[5] J. Madhavan, S. Cohen, X.L. Dong, A.Y. Halevy, S.R. Jeffery, D. Ko, and C. Yu, "Web-Scale Data Integration: You can Afford to Pay as You Go," *Proc. Conf. Innovative Data Systems Research (CIDR)*, pp. 342-350, 2007.

[6] S.R. Jeffery, M.J. Franklin, and A.Y. Halevy, "Pay-As-You-Go User Feedback for Dataspace Systems," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '08)*, pp. 847-860, 2008.

[7] A.D. Sarma, X. Dong, and A.Y. Halevy, "Bootstrapping Pay-As-You-Go Data Integration Systems," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '08)*, pp. 861-874, 2008.

[8] M.A.V. Salles, J.-P. Dittrich, S.K. Karakashian, O.R. Girard, and L. Blunschi, "Itrails: Pay-As-You-Go Information Integration in Dataspaces," *Proc. 33rd Int'l Conf. Very Large Data Bases (VLDB '07)*, pp. 663-674, 2007.

[9] S. Song, L.Chen, M. Yuan, "Materialization and Decomposition of Dataspaces for Efficient Search," Knowledge and Data Engineering, IEEE Transactions on Knowledge and Data Engineering, vol.23, no.12, pp.1872,1887, Dec. 2011

[10] A. Marcos, S. Vaz, D. Jens, B. Lukas, "Intensional associations in dataspaces,"*In: ICDE 2010*, (2010)

[11] R. Grishman, "Information Extraction: Techniques and Challenges," in SCIE, 1997.

[12] P. Ziegler, K. Dittrich,"Data Integration — Problems, Approaches, and Perspectives," *Springer*, Berlin Heidelberg, 2007.

[13] J. Pokoryn, "Databases in the 3rd Millenium: Trends and Research Directions," *Journal of Systems Integration* vol. 1, no. 1-2, pp. 3-15, 2010.

[14] B. Blackwell, "The use of range-tree spatial indexing to speed GIS retrieval," *Proc. AUTO-CARTO 8, 195-200*, 1987.

[15] E. Rahm and P.A. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *Int'l J. Very Large Data Bases*, vol. 10, no. 4, pp. 334-350, 2001.

[16] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[17] I.H. Witten, A. Moffat, and T.C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, second ed. Morgan Kaufmann, 1999.

[18] J. Zobel and A. Moffat, "Inverted Files for Text Search Engines,"ACM Computing Surveys, vol. 38, no. 2, pp. 1-55, 2006.

[19] M. V. Salles, "Pay-as-you-go Information Integration in Personal and Social Dataspaces," Ph.D. dissertation, ETH Zurich, 2008.

[20] V. Fisikopoulos, "An implementation of range trees with fractional cascading in C++," arXiv, 2011.

[21] T.M. Chilimbi, J.R. Larus and M.D. Hill. Improving pointer-based codes through cache-conscious data placement. Technical report 98, University of Wisconsin-Madison, Computer Science Department, Madison, Wisconsin, 1998.

[22] R. Berinde, "Efficient implementations of range trees," 2007.

[23] R. Fagin, "Combining Fuzzy Information: An Overview," *SIGMOD Record*, vol. 31, no. 2, pp. 109-118, 2002.

[24] EnterpriseDB, "A Comparison of PostgreSQL 9.0 and MySQL 5.5," 2011. [Online]. Available: http://www.arsys-eu-ope.net/EnterpriseDB/White_Papers/White_Papetr_Postgres_Plus_9.0_vs_MySQL_5.5.pdf

[25] WikiVS, "MySQL vs PostgreSQL," 2008 . [Online]. Available: http://www.wikivs.com/wiki/MySQL_vs_PostgreSQL

[26] 2nd Quadrant, "10 reasons why MySQL Users should move to PostgreSQL," 2011. [Online]. Available: http://www.2ndquadrant.com/static/2quad/media/pdfs/10_reasons_why_mysql_users_should_move_to_postgresql.en.pdf

[27] X. Yang, "Analysis of DBMS: MySQL Vs PostgreSQL," 2011. [Online]. Available: http://publications.theseus.fi/bitstream/handle/10024/27471/Final_Thesis_Xiaojie_Yang.pdf?sequence=1

[28] G. Salton, "Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer," Addison-Wesley, 1989.

[29] L. Arge, G.S. Brodal, R. Fagerberg. "Cache-Oblivious Planar Orthogonal Range Searching and Counting," *Proc. of the 21st Annual Symposium on Computational Geometry*, 2005.

[30] J. Rao, K. Ross. Cache Conscious Indexing for Decision-Support in Main Memory. Proceedings of the 25th International Conference on Very Large Data Bases, pages 78-89, 1999.

[31] B. Shibwabo, I. L. Ateya and G. W. Wanyembi, "Modelling Dataspace Entity Association using Set Theorem'" *Computer Technology and Applications*, vol. 3, No. 6, (2012).

[32] J. L. Bentley. Multidimensional binary search trees used for associative searching. Commun. ACM, 18(9):509–517, 1975.

[33] G. S. Lueker. A data structure for orthogonal range queries. In SFCS '78: Proceedings of the 19th Annual Symposium on Foundations of Computer Science (sfcs 1978), pages 28–34, Washington, DC, USA, 1978. IEEE Computer Society.

[34] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *Proc. of ACM SIGMOD Conference on Management of Data*, Boston, June 1984.

[35] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry: Algorithms and Applications. Springer-Verlag, second edition, 2000.

[36] D. M. Mount. Lecture notes: Cmsc 754 computational geometry. lecture 18: Orthogonal range trees, pp. 102–104, 2007.