Uniwersytet Ekonomiczny w Katowicach

Studia Ekonomiczne. Zeszyty Naukowe
Uniwersytetu Ekonomicznego w Katowicach
ISSN 2083-8611 Nr 296 · 2016

Informatyka i Ekonometria 6

Grace L. Samson

University of Huddersfield, UK,
Department of Computer Science,
gracedyk@yahoo.com

Joan Lu

University of Huddersfield, UK,
Department of Computer Science,
j.lu@hud.ac.uk

PaX-DBSCAN: A PROPOSED ALGORITHM FOR IMPROVED CLUSTERING

Summary: We focused on applying parallel computing technique to the bulk loading of X-tree in other to improve the performance of DBSCAN clustering algorithm. We have given a full description of how the system can be archived. We proposed a new parallel algorithm for DBSCAN and another algorithm to extend the X-tree spatial indexing structure. Spatial database systems incorporate space in database systems, they support non-traditional data types and more complex queries, therefore in order to optimise such systems for efficient information processing and retrieval, appropriate techniques must be adopted to facilitate the construction of suitable index structures.

Keywords: X-tree, spatial index, partition, parallel computing, bulk-loading, spatial database, clustering.

Introduction

According to Lungu and Velicanu [1], spatial objects consisting of lines, surfaces, volumes and higher dimensions objects are frequently used in applications such as computer-aided design, cartography, geographic information systems etc. A single **spatial data** contains observations **with locations**, they identify features and positions of objects on the earth's surface and they present us a framework for putting our observations on the map [2]. In this paper, we describe the design of a system for spatial query processing (suitable for managing large datasets) that fully exploits the parallelism that is typical of modern multi-core CPU. The notion is to design a system that parallelises the indexing of spatial data and spatial query execution. We base this work on the shared-nothing platform as a platform to solve the problem of parallel bulk loading of X-tree in

a parallel spatial database context. We assume that an adjusted X-tree (aX-tree − which we proposed) access method is constructed, from a spatial relation that is distributed to a number of processors. The main intension is to exploit parallelism in order to achieve both high quality of produced index and efficient index generation. As such, we did a deep study of parallel techniques for bulk-loading while assuming that the environment is composed of a number of processors based on a shared-nothing architecture, in which each processor manages its own disk(s) and main memory. We have assumed that there would be no reorganization of the data taking place after the completion of the index construction process, that is to say, the data remain assigned to the same processor. It is important that some processors need to transmit the spatial information of the objects to other processors, without transmitting the whole record (i.e., the objects' detailed geometry). This approach would guarantee load balance during index construction. Most research on spatial databases focuses on either the performance or the space utilization therefore, by dividing large problems into smaller ones, big problems can be solved concurrently saving time and resources with an improved performance. Parallel computing describes a process where computation involving many calculations or the execution of multiple processes are carried out simultaneously [3]. This kind of computing (in the form of multi-core processors) has become the dominant paradigm in computer architecture. In this work, we have focused on applying parallel computing technique to the bulk loading of X-tree.

1. Spatial database systems

According to Güting [4] Spatial Database Systems (SDBS) are database systems for the management of spatial data, including point objects or spatially extended objects in a 2D or 3D space or in some high-dimensional feature space. In Velicanu Belciu and Olaru [5], spatial database is described as a collection of spatial and non-spatial data that is made up of data descriptions and links between data. Spatial databases incorporate space in database systems, they support non-traditional data types and more complex queries, therefore in other to optimise such systems for efficient information processing and retrieval in a large multidimensional spatial dataset environment, appropriate techniques must be adopted to facilitate the construction of suitable index structure for these database systems. A number of spatial access methods have been proposed because the idea of improving large spatial databases is a way to empower them to effi-

ciently support applications that require non-conventional data. The most important distinguishing factor of SDBSs s their ability to answer in answering queries (involving spatial relationships between objects efficiency).

2. Clustering

Clustering real world data sets according to Kailing et al. [6], Verma and Jain [7], J. Liu [8] and T. Liu [9], is often hampered by the so-called curse of dimensionality and it is a fact that many real world data sets consists of very high dimensional feature space. According to Han and Kamber [10], different types of clustering methods exist including hierarchical, partition, Density Based method and Grid based method. The DBSCAN algorithm discussed below is an example of a density based clustering method. In Fayyad et al. [11], clustering is described as a data mining technique that groups data into meaningful subclasses, known as clusters, such that it minimizes the intra-differences and maximizes inter-differences of these subclasses. Several clustering algorithms including: K-means, K-medoids, BIRCH, DBSCAN, STING, Wave-Cluster, etc. [12].

2.1. DBSCAN algorithm

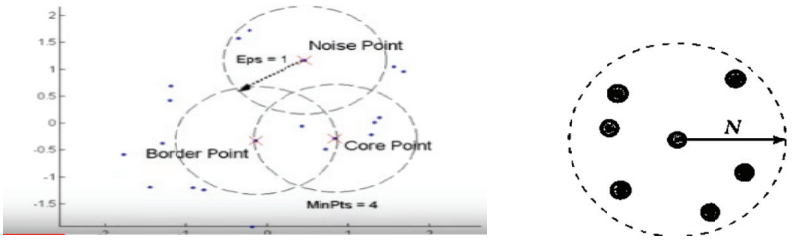
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is an effective clustering algorithm for Spatial Database Systems, which has the ability to detect noise and outlier, cluster arbitrary shaped point dataset and (contrary to some other clustering algorithm like the k-means), does not require the number of clusters a priori. Notwithstanding the performance of the algorithm deteriorates when the data size becomes too large and the algorithm may not perform optimally if the wrong values are chosen for *minpts and eps* (i.e., *radius neighbourhood*), which are two vital components of the algorithm. In this paper, we propose a new algorithm that can improve the efficiency of the DBSCAN clustering algorithm. The motivation is to improve the performance of the algorithm in terms of analysing huge spatial databases and in its process of choosing the right *minpts and eps* values. Density based algorithms reserve the notion that two objects in space are similar to each other, if the space between them is small. The DBSCAN algorithm proposed by Ester et al. [13] is described below:

1. The algorithm takes as an input:
   - A set of **points P** in space **(2d)**.
   - A neighbourhood **N** and a neighbourhood value **eps (see figure 1 below)**.

- And a parameter **minpts**, which determine when a cluster can be taken as dense.

2. The algorithm starts with an arbitrary unvisited starting point.
3. Then extracts the neighbourhood of that point using the **eps** value.
   // all points within the **eps** distance are in the same neighbourhood.
4. Clustering process begins when enough points (**minpts**) are found around the neighbourhood with a distance not more than **eps** between each points.
5. For all points that belongs to the cluster (including its **eps neighbourhood**), repeat **steps** 3 through 5.
6. Then new unvisited points are extracted and processed (this might lead to the discovery of further clusters or even noise).
7. The process terminates only when all points are visited.

Figure 1. Diagram showing DBSCAN core point border point, noise point and eps neighbourhood (N)

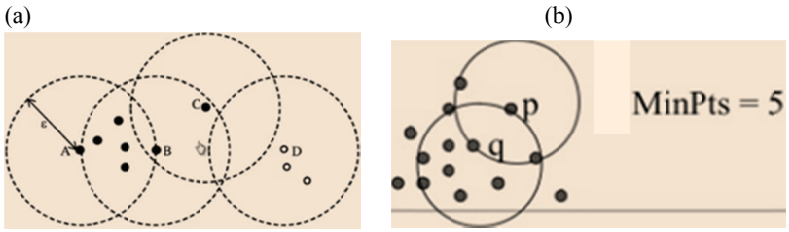


2.2. DBSCAN characteristics

The characteristics are as follows:

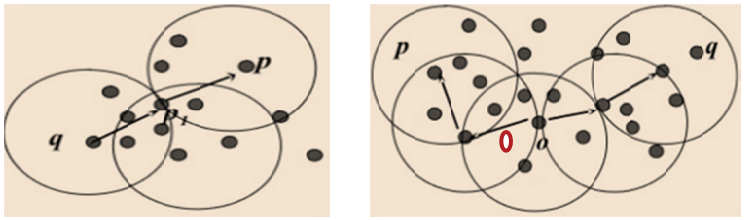
1. **Directly Density Reachable.** A point **p** is directly density reachable from another point **q**, if **p** is within the ***eps*** (Figure 2a) neighbourhood of **q** and **q** is a core point (core because it has at least ***minpts*** within its neighbourhood – see Figure 2b).

Figure 2. DBSCAN directly density reachable points



2. **Density reachable**. A point **p** is density reachable from **q**, if there exist a sequence of points $\mathbf{p_1}$….$\mathbf{p_n}$, $\mathbf{p_1} = \mathbf{q}$ and $\mathbf{p_n} = \mathbf{p}$ so that $\mathbf{p_{i+1}}$ is directly density reachable from $\mathbf{p_i}$ (see Figure 3).

Figure 3. DBSCAN reachable



3. **Density connected**. A point **p** is density connected to point **q** if there is another point **O** so that both **p** and **q** are density reachable from **O.** It is also worthy to note that the shape of the ***eps*** neighbourhood can be derived by the choice of a distance function for two any points **p** and **q**, denoted by ***distance* (p, q)** and in each case, this shape appears different**.** Using the Manhattan distance in 2D space for example, the shape of the ***eps*** neighbourhood is rectangular. The description of the DBSCAN (expressed in 2D space using the Euclidean distance) in Ester et al. [13] indicates that the algorithm works with any distance function and so an appropriate function can be chosen for any given application.

2.3. Problems of the existing DBSCAN algorithm

Clustering algorithms for spatial databases according to Ester et al. [13] specifically deals with task of class identification, however, it does not come cheap as such an algorithm must satisfy the three basic requirements of **a)** having a basic domain specific knowledge to be able to determine the input parame-

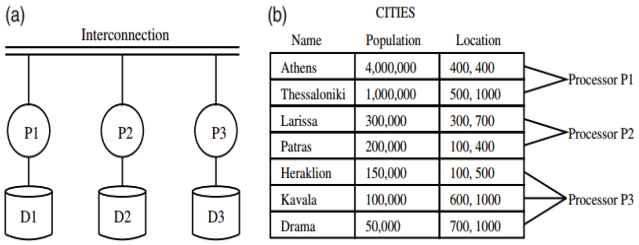
ters, **b)** discovery of clusters with arbitrary shape, and **c)** having a good efficiency on large databases. Despite all the abilities of the existing DBSCAN algorithm, it is well known to possess some major limitations, which include high time consumption for finding neighbourhood (***eps***) of a given data point [14], performance degeneration with increase in dataset size [15]. The DBSCAN algorithm clusters data points based on density and the underlying idea of density is based on the two parameters (***Eps and Minpts***). According to Berkhin [16], for a spatial database, clustering algorithms can easily be improved for fast nearest neighbour search if they are indexed, because the indexes serve as good substitutions for poor performance caused by dimensionality. Spatial index structures like the R-trees Ester et al. [17] are normally used in a spatial database management system to speed up the processing of queries such as region queries or nearest neighbour queries. When the SDBSs is indexed by an R-tree (or any other indexing structure), then the R-tree nodes helps to accelerate the search operations [18]. Notwithstanding, the basic limitations of the existing DBSCAN algorithm is compounded by the fact that though the R-tree based index structure do not require point transformation in other to store spatial data and also proves efficient for spatial clustering which is a vital issue in the performance of tree based indexing structures according to Berchtold et al. [19], they are not adequate for high-dimensional data sets as the index structures supports high overlap of the bounding boxes in the directory, which increases with growing dimension. The problem with this is that most large spatial databases are often represented using high-dimension **feature vectors**, thus because feature spaces most often tend to contain multiple instances of similar objects (Samet, 2006), then the database built using such a feature space is bound to be clustered thus if the database is indexed with an R-tree there would be cases of redundant search of rectangles due to the high overlap between MBRs of the R-tree nodes. According to Mamoulis [18] several new index structures (including the A-tree, VA-tree and the X-tree) have been proposed that outperforms the R-tree for indexing high dimensional data but most of them show degraded performance as dimension increases [16], [19], [18]. Thus based on these premises we propose an improved DBSCAN algorithm that is accelerated using an adjusted ***X-tree*** (***aX-tree***) and scalable for large datasets through the power of parallel computing technology.

3. Parallel programming

3.1. Parallel programming architecture

According to Taniar et al. [21], the whole essence of parallelism is to be able to reduce data size by partitioning the data into a number of processors, whereby each focuses on processing its partition of the data. By the completion of these individual task by the various processors, then all the results are combined to form the final result. Due to the increase in the amount of data accumulated daily nowadays, single processor database management systems are becoming inefficient in data management, thus the diversion to parallel databases [22]. A parallel database is equipped to manage data in 10^{12} bytes or above in a very short period of time. According to Papadopoulos and Manolopoulos [23], the benefits of parallel database management systems can easily be understood by taking into consideration the large computational power and the huge amounts of data that modern applications require.

Figure 4. A parallel database system architecture



Source: [23].

Figure 4 shows a parallel database system architecture [23] with three processors with range partitioning of cities relation with respect to attribute population. In Qin et al. [24], there are basically two standard architecture for constructing a parallel computing architecture for processing big spatial including the shared-memory and distributed-memory systems (shared nothing) both of which are adopted to achieve higher availability and better computing performance, and also take advantage of the GIS resource-hungry application domain that still makes good use of parallel techniques for processing spatial data attributes. Among these frameworks, the shared nothing architecture according to Achakeev et al. [25] tends to outperform the rest in terms of low cost data processing. Even Hadoop, one of the MapReduce frameworks that allows for deve-

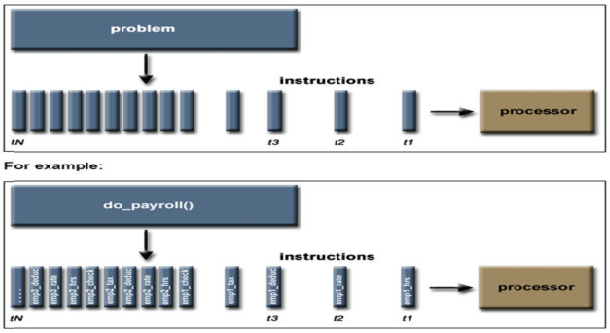
loping conceptually simple and scalable algorithms for processing big data, uses this architecture. For a wide range of application domains such as environmental assessments and medical imaging, time-consuming computational geometry algorithms make the applications very slow, in addition there is a rapid expansion in available processing cores, through multicore machines and cloud computing, therefore, the combination of these trends demands effective parallelization of spatial query processing. Parallel databases present the opportunity to solve variety of problems and they are very useful in a wide range of fields including bioinformatics (e.g., protein folding and sequence analysis). The main differences between parallel and traditional (serial) programming given by Barney [26] are illustrated below.

3.2. Serial/traditional programming

In serial programming (computation) software are basically written by:

- braking a problem into discrete series of instructions,
- executing instructions sequentially one after another,
- using a single processor,
- executing only one instruction any moment in time.

Figure 5. Diagram of a serial mode of computation/programming



Source: [26].

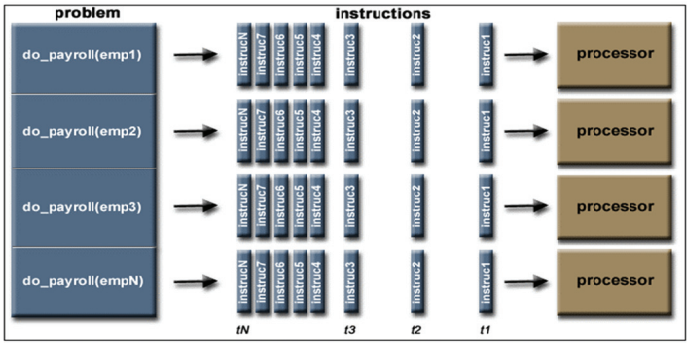
3.3. Parallel technology

In parallel computing, multiple computer resources are used to solve a computational problem simultaneous this techniques involve:

- breaking a problem into discrete parts that can be solved concurrently with each part further broken down into a series of instructions,

- executing instructions from each part simultaneously on different processors,
- the use of overall coordination mechanism.

Figure 6. Diagram of a parallel mode of data processing



Source: [26].

3.4. Parallel programming present state

In recent times, parallel systems has become the order of the day in terms of managing large multidimensional databases. According to Provost and Fawcett [27] Big Data are datasets that are too large for traditional data-processing systems, that requires new technologies, like Hadoop, Hbase, MapReduce, MongoDB or Couch-DB. Notwithstanding, the obvious challenges militating against the success of spatial database management, is almost becoming a thing of the past with the evolution of the Hadoop [28] one of the implementations of MapReduce [29]. This trend hoped to provide analyst with the opportunity to efficiently process large scale data sets by exploiting parallelization. Recent research [30], [31], [32], [33] has indicated that the importance and significance of parallel and distributed programming for handling big data sets in the general context or even in the geospatial context cannot be over emphasized. A significant aim of parallel DBMS according to Zhao et al. [34] is to provide input/output parallelism so as to get a high performance parallel data processing.

3.5. Parallel programming for big spatial data

According to Zhao et al. [34], there are two main parallel GIS design architectures for efficient management of large spatial databases: the one is a framework based on high performance computing cluster (which is what we have

adopted for this work) and the other is based on Hadoop cluster (that implements Mapreduce). In support of this, Maitrey and Jha [35] has established that MapReduce has emerged as the most prevalent computing paradigm for parallel, batch-style and analysis of large amount of data. VegaGiStore was proposed by Zhong et al. [36] as an advanced tool that provides efficient spatial query processing over big spatial data and numerous concurrent user queries. The system creates a geography-aware module to organise spatial data in terms of geographic proximity, then designs a two tier distributed spatial index for efficient pruning of the search space in order to improve data retrieval efficiency, finally the system builds an "indexing + MapReduce" data processing architecture to improve the computation capability of spatial query. Tang and Feng [37] proposed a map projection cloud based parallel framework that possesses a coupling of the capabilities of cloud and high performance computing that is GPU-enabled for managing large spatial databases. Their system is a parallel paradigm for map projection of vector-based big spatial data that couples cloud computing with graphics processing units. Tan et al. [38] established an efficient mechanism which stands as a general framework for parallel R-tree packing using MapReduce. Other advanced techniques have also been proposed and designed and we have provided a detailed description of these systems under parallel bulk-loading techniques for managing large spatial databases in a later section. Li et al. [39] gave an overview of the most recent literature and technologies on the management of large spatial databases.

3.6. Parallel DBSCAN existing systems

Though a little bit different from the scope of this work but of interest to our line of discussion, Ogden et al [40] proposed the AT-GIS which is a highly parallel *spatial query processing association mining system* (which operates on raw spatial datasets) that can scales linearly to a large number of CPU cores by integrating the parsing and querying of spatial data using a new computational abstraction called associative transducers (ATs). The new system has the ability to form a single data-parallel pipeline for computation without requiring the spatial input data to be split into logically independent blocks. The AT-GIS also has the ability to execute in parallel, spatial query operators from raw input data in multiple formats without any pre-processing. The interesting thing about their work is that it does not build an index for spatial query which is a little bit away from the general assertion that a databases not indexed normally perform very

poorly for query processing. Never-the-less, Chen et al [41] proposed the P-DBSCAN, a novel parallel version of the existing DBSCAN algorithm which is applied in a distributed environment by implementing a priority R-tree. In Welton et al. [42] the extended CUDA-DClust algorithm was applied, the system implements a block tree indexing structure to extend the functionality of the existing DBSCAN. Their DBSCAN clustering algorithm version (Mr. SCAN) is designed to handle extreme cases in density based clustering using a hybrid parallel tree-based implementation to combine a network of GPGPU-equipped nodes with an MRNet tree-based distribution network. MR-IDBSCAN was proposed by Noticewala and Vaghela [43] as an efficient parallel and incremental method that improves the existing DBSCAN Algorithm using MapReduce. Xu et al. [44] proposed a fast parallel clustering algorithm for large spatial databases called PartDBSCAN based on a dR*-tree indexing structure. Their system modifies the DBSCAN algorithm by finding clusters w.r.t. a given space constraint S. the main highlight of their system is that it has a very good performance w.r.t. speedup, scale-up and size-up and most of all could be applied to extend other spatial access methods of the R-tree family (such as the X-tree) to distribute spatial index structures for high-dimensional data. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure (PDSDBSCAN) was proposed by Patwary et al. [12]. The algorithm uses a tree-based bottom-up approach to construct clusters with a better-balanced workload distribution and it is implemented on both a shared and a distributed memory architecture.

4. Big spatial data management

In spatial database management, objects are not single-valued and in most cases, they range from points in a multidimensional space to complex polygons. New technologies are evolving for the management and manipulation of large datasets, so some improvements and advancement benefitting large spatial databases include the presented below solutions.

4.1. Cloud computing technologies

Cloud computing is a necessity for big spatial data management and the efficiency of spatial indexing for huge datasets at cloud computing environment cannot be over emphasized [45]. According to Song et al. [46], the main goal of implementing the cloud based platform is to solve the issues faced by traditional

geospatial information platform, such as data-intensive, computing-intensive, and concurrent-intensive problems, this would in turn enhance the implementation of big geo-data analytics and management, provide geospatial information services for multi-departments of government, and facilitate information sharing. Cloud computing according to Wang et al. [33] is the use of resources that are delivered as a service over a network and due to the flexibility and scalability in cloud computing, now cloud computing plays an important role to handle a large-scale data analysis.

4.2. Spark technology

The spark technology Zaharia et al. [47] is designed to exploit large main memory capacities, it is built on the notion of Resilient Distributed Dataset and implemented using *Scala*, it utilizes built-in data parallel functions for vectors/collections (such as map, sort and reduce), which not only makes the programs more concise but also makes them parallelization friendly. You et al. [48] proposed the SpatialSpark which supports indexed spatial joins based on point-in-polygon test and point-to-polyline distance computation and has been designed for large-scale spatial join query processing in cloud.

4.3. Indexing spatial data

Spatial data objects in most cases often cover areas in multidimensional or high dimensional spaces. They are often not well represented by point location thus; an indexing method that can support N-Dimensional range queries based on the object's spatial location is required. The main goal of indexing is to optimize the speed of query according to Singh and Garg [49]. When needing to represent large spatial data, it normally requires a lot of resources in terms of storage and time costs therefore, optimizing the database is one of the most important aspects when working with such large volumes of data [5]. Notwithstanding, Akkaya and Yazici [50] stated that a number of multi-dimensional access methods have been proposed by various researchers in order to support spatial search operations in databases. These methods are used to store and retrieve extended and complex objects. In Velicanu Belciu and Olaru [5], spatial indexes are the best way to improve the optimization of spatial databases. According to Mamoulis [18], when a spatial relation is not indexed there would be need for the nearest neighbour algorithm (for clustering purpose) to access all

objects in the relation, in order to find the nearest neighbour to a query object *q*. Building an indexing structure for spatial data is a mechanism that decreases the number of searches, and a spatial index (considered logic) is used to locate objects in the same area of data (window query) or from different locations [1]. In Gaede and Günther [51], Lee and Lee [52] it is established that since spatial data object are composed of a single point or several thousands of polygons randomly distributed across space, constructing a spatial index is very important. Generally, data mining tasks (e.g., clustering algorithms) for a spatial database can easily be enhanced for fast nearest neighbour search if they are indexed, because the indexes serve as good substitutions for poor performance caused by dimensionality [16]. There are basically two approaches for building a spatial access method. In the first technique individual insertion of the spatial objects is applied, meaning that the access method must be equipped to handle insertions. However, the second technique involves building the access method based on the knowledge of the original dataset (bulk-loading), which means that the data must be available in advance. Fundamentally, the availability of data *a priori* occur quite frequently in various application environments for instance, data can be archived for many days in data warehouses and in order to answer queries efficiently, access methods must be constructed. According to [18] good bulk loading method would build fast for static objects and will ensure a lesser amount of wasted empty spaces on the tree page.

Data mining tasks like clustering require the spatial relation to be indexed otherwise there would be need for certain procedures (like neighbourhood finding for clustering purpose) to access all objects in the relation in order to find the nearest neighbour to a query object. The DBSCAN clustering algorithm is an effective clustering algorithm for Spatial Database Systems, which has the ability to detect noise and outlier, cluster arbitrary shaped point dataset and does not require the number of clusters a priori, but the performance of the algorithm begins deteriorate when the data size becomes too large and the algorithm may not perform optimally if the wrong values are chosen for ***minpts and eps***. Therefore the new algorithm is geared toward overcoming these limitations.

4.4. Bulk loading

Another way forward for managing large spatial dataset is by the use of bulk loading methods. Since most spatial applications are based on *write once read many* access model according to Liu et al. [53], the large amounts of spatial

data could be quickly imported into storage systems for rapid deployment of spatial information services. However, bulk-loading of spatial data is time-consuming and cannot satisfy the desire of the applications dealing with massive spatial data as such, the *parallel technique of bulk loading* proposed by Qin et al. [24], is designed to accelerate the processing of spatial data bulk loading for building tree-based in parallel. Bulk-loading spatial data using the popular *MapReduce framework* is intended to overcome the problems associated with parallel bulk-loading of tree-based indexes which have the disadvantage that the quality of produced spatial index decrease considerably as the parallelism increases [54]. In You et al. [48] bulk loading methods have been described as being more suitable for static read-only data in OLAP (Online Analytic Processing) settings in many applications, where it is assumed that the MBRs of geospatial data can fit into processor memory (which is increasingly becoming practical due to the decreasing prices of memories), the cost of bulk loading is largely determined by in-memory sorting in the order of O (nlogn). The study identified that sorting for bulk loading can be significantly accelerated on GPUs by utilizing the parallel computing power which makes GPU implementations attractive. However, for MBRs with variable sizes of degrees of overlapping, the qualities of constructed R-Trees through bulk loading can be very different which may significantly affect query performance on both CPUs and GPUs.

4.4.1. Serial bulk loading

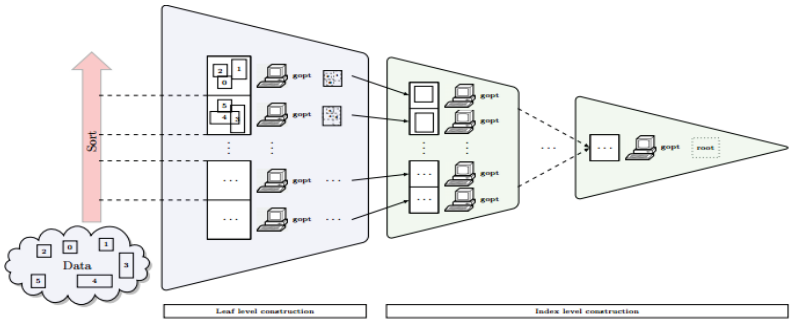
Roussopoulos and Leifker [55] proposed the first sort-based bulk-loading algorithm for R-trees, the method use similar B+-trees methods to build R-trees bottom-up from scratch. The rectangles used as input are first sorted according to one of the dimensions and then the sorted data is scanned and a fixed number of elements is then repeatedly assigned to a node. Kamel and Faloutsos [56] proposed a double-transformation technique which extends the method discussed above. In their approach, a rectangle is mapped to a multidimensional point, and then using a space-filling curve (i.e., the Hilbert-curve) a sorting order is specified. Other serial bulk loading methods include Leutenegger et al. [57] the sort-tile-recursive algorithm which applies a sort and partitioning step for each dimension and Achakeev et al. [58] an optimal query-adaptive algorithms for building R-trees designed for a given query profile.

4.4.2. Parallel bulk-loading

Undoubtedly many bulk-loading algorithm for R-tree has surfaced whether sort-based and non-sort based, nevertheless, our quest in this study is on the improvement of some of these existing technologies for bulk-loading of spatial data by taking advantage of parallel technology. For massive spatial (or none spatial) data, serial/sequential bulk-loading techniques has proven highly inefficient due to being too time-consuming and therefore may not satisfy the computational need of many applications dealing with it. Qin et al. [24] proposed the TGS-based (Top-Down Greedy Split) parallel technique for accelerating the processing of spatial data bulk-loading, by adopting the DCSO (Decompose – Conquer – Stitch − Output) strategy to build the R-tree in parallel. Papadopoulos and Manolopoulos [23], gave a proper description of how to solve the problem of R-tree parallel bulk-loading. Their description is for a generic framework for R-tree bulk-loading on a parallel architecture. In their work, the input rectangles are distributed among the computing nodes so that every machine receives an approximately equal amount of data. This phase utilizes parallel random sampling where a single coordinator machine computes a ***kd-tree*** for partitioning the data space into regions, each associated with a computing node. The regions are then used for rectangles-to-nodes allocation, and then following the above implementation, a local R-tree is bulk-loaded for every node. Lastly, the root entries of local R-trees are sent back to the coordinator where a global root node is then created. Papadopoulos and Manolopoulos [23] also presented various strategies for dealing with R-trees of different heights, in which case an additional post-processing by the coordinator machine is needed in order to obtain the final R-tree. Liu et al. [54] proposed a novel method of bulk-loading spatial data using MapReduce framework, which combines Hilbert curve and random sampling method to parallel partition and sort spatial data. Their technique applies the bottom-up method to simplify and accelerate the sub-index construction in each parallel partition. In Achakeev [25], the problem of parallel loading of R-trees on a shared nothing platform was addressed and a novel scalable parallel loading algorithm for MapReduce was proposed based on a state of the art sequential sort-based query-adaptive R-tree loading algorithm which builds a level-wise R-tree (*In contrast to individual R-tree loading, they created each level of the R-tree in parallel, allowing the scheme to avoid the problem merging local R-tree* – see Figure 7), optimized according to a commonly used cost model. A similar MapReduce technique was adopted by Zhong et al. [36], but in their own case, they implemented a two-tier distributed spatial index for efficient

pruning of the search space instead of the level-wise (level by level parallel R-tree) used in the former method. In the work of Tan et al. [38], the design and implementation of a general framework for parallel R-tree packing using MapReduce was introduced. The framework sequentially packs each R-tree level from bottom up and further presents a partition based algorithm for parallel packing lower levels that have a large number of rectangles. Hua et al. [59] proposes an R-tree bulk loading algorithm that uses the STR strategy (based on the parallel computing powers of GPGPU systems) but applied an overall instead of the usual sorting technique constantly used. You et al. [46] also applied the massive data parallel technologies of graphic processing units (GPUs) to index and query geospatial data based on R-trees. Their paper investigated on the potential of accelerating both R-tree bulk loading construction and R-tree based spatial window query on GPUs. Other works on GPGPU based R-tree indexing of spatial data include: Ogden et al. [40]. According to Ogden et al. [40], current solutions for largescale spatial query processing either rely on extensions to RDBMS (which has to do with expensive *loading* and *indexing* phases when the data changes) or distributed map/reduce frameworks (which runs on resource-hungry compute clusters). Both solutions of which according to them struggle with the sequential bottleneck of parsing complex, hierarchical spatial data formats, which frequently dominates query execution time.

Figure 7. Level by level parallel R-tree



Source: [25].

For efficient query processing in large spatial databases, the R-tree has proven to be a key element, though its creation is costly [25]. Moreover, the R-tree spatial index built by the sort-tile-recursive (STR) techniques has excellent query performance, but low efficiency when building [59]. Notwithstanding, Giao and Anh [60] argues that the Sort-Tile-Recursive (STR) algorithm which is a sort-

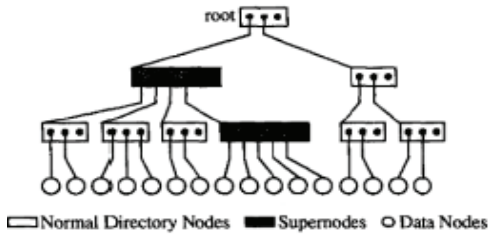
based loading method for managing spatial and multidimensional data remains one of the simple and efficient bulk-loading strategy. Numerous parallel R-Tree construction and query processing algorithms have also been proposed including: Kamel and Faloutsos [61], Hoel and Samet [62], Schnitzer and Leutenegger [63], Apostolos and Yannis [64], Luo et al. [65],You et al. [48]; Hua et al. [59], etc. Most of these algorithms focus on the shared-nothing computer architecture, though some of the recent works implemented R-Tree based construction and query processing on GPUs based on the General Purpose computing on GPUs (GPGPU) technologies. Notwithstanding, in general, though the distributed frameworks with indexing support can offer good query performance, they require substantially more computational resources than single machine deployments [40]. Obviously, a very distinctive characteristics of most of all the existing systems above is that they have all focused on R-Tree based spatial indexing and query processing, in this work we have looked extensively into the design of a different spatial indexing technique the X-tree; which we try to achieve by exploiting the parallelism offered by modern multicore CPUs for parsing and query execution, thereby improving the performance of a computer cluster within a distributed resource environment. In essence, we consider the possibility of benefiting from the influence of parallelism in accelerating the performance of spatial access methods most specifically the X-trees.

The X-tree proposed by Berchtold et al. [66] provides a suitable structure for indexing point and spatial data in high-dimensional space. It is a method for indexing large amounts of point and spatial data in high- dimensional space. Berchtold et al. [19] states that index structures such as the R*-tree are not adequate for indexing high-dimensional data set. X-tree, according to Berchtold et al. [19] and M-tree according to Ciaccia et al. [67], are typically other variants of the R-tree used for multidimensional data. According to the authors of the M-tree article, the construction of M-tree is fully parametric based on some distance function *(d)* and triangle inequality for efficient queries. The M-tree has overlap of regions but no strategy to avoid overlap. Each node there is of radius r, every node n and leaf node l residing in node $\boldsymbol{N}$ is at most distance r from $\boldsymbol{N}$. The M-tree is balanced tree and does not requires periodical reorganization. The X-tree prevents overlapping of bounding boxes which is problem in high dimensionality. Any node that is not split will then result into "*super-nodes*" and in some extreme cases the tree will linearize. The X-tree may be seen as a hybrid of a linear array-like and a hierarchical R-tree-like directory [19]. According to Candan and Sapino [68], an increase in the fan-out of the X-tree is the main positive side effect of the *super-node* strategy. Some advantages of X-tree, as

given by Manolopoulos et al. [69] besides Candan and Sapino [68] shows that the X-tree is a heterogeneous access method because it is composed of nodes of different types. In most cases, whereby it has become impossible to overcome or avoid overlap, super-nodes are created during the processes of inserting new entries into an X-tree. These super nodes account for the advantage of X-trees over all other access methods. Some of the benefits of the super-nodes include:

- increase in average storage utilisation due to fewer splits taking place,
- reduction in height of tree due to increase in average tree fan-out,
- in cases where it is impossible to construct a hierarchical access method with minimised overlap between node bounding regions, then sequential scanning of the dataset is facilitated for very high-dimensional spaces. A diagram of a typical X-tree structure is given in Figure 8 below.

Figure 8. Typical structure of the X-tree



Source: [19].

For low dimensionality, it means that there is no overlap between the triangles, and at first, the X-tree tries to choose an overlap-free (or at least overlap minimal) split axis. When splitting a new node will cause an overlap in rectangles, then the super-node is extended with an additional disk page. A super-node of l pages will have l times more children than a regular node. A super-node consisting of multiple disk pages may require multiple disk accesses (or at least one disk seek operation followed by multiple rotations) therefore, when a given query does not cover the entire MBR of the Super-node, the extra disk accesses result in unnecessary overhead. Nevertheless, this approach diminishes problems with scalability, but cannot solve the problem totally, as in high dimensional data, overlap problem grips the index eventually. The X-tree has also proven very efficient for query processing in large spatial database. We proposed this new scalable parallel loading algorithm for implementing DBSCAN clustering algorithm in parallel. The proposed system would provide a better query performance than R-trees build and other competitive bulk-loading algorithms.

5. Constructing PaX-DBSCAN clustering algorithm

Basically, the approach we have adopted for parallelizing the DBSCAN by implementing the ***aX-tree*** is very simple and it involves the simple logical steps below:

1) Given a large spatial dataset;
2) Store them in a parallel spatial database;
3) Build an aX-tree index on it;
4) Implement the DBSCAN clustering algorithm;
5) Combine the result to get one final output.

We propose to implement the DBSCAN algorithm by applying it on machines that are located at different site individually with a local cluster on each client node (Nc_i *- for* i_s *= 1 to the total number of c clients*). These clusters are then sent to the master node (N_m) from the entire site. On the Master Node we build a global cluster which will synchronise the entire local clusters. The master node takes the job from the different site and aggregates the result for the final output cluster.

The proposed algorithm PaX-DBSCAN which is another novel parallel version of the known DBSCAN clustering algorithm is presented and described in detail. It applies in a distributed computing environment by implementing an adjusted **X-tree** spatial indexing structure. Different from the existing methods we have reviewed above, the algorithm is enhanced by the implementation of the **aX-tree** which has proven to be efficient in high dimensional cases of large spatial data. Apart from the adjusted indexing structure, we have also proposed a new algorithm for the DBSCAN which does not depend on the values of the Eps – neighbourhood (as this is the main factor behind the delayed computational time of the original algorithm). The choice of an adjusted **X-tree** instead of the regular R-tree used as the underlying index structure for DBSCAN, is to improve the algorithm in the terms of managing large spatial dataset. First, we apply a partition technique which provides a paradigm to manage data in database by initially decomposing data into smaller chunks. Secondly, we store the decomposed dataset into different partitions. Thirdly, we derive a function to construct a static X-tree in a parallel modes (so as to compress the construction time). By following this simple procedure, updating the structure or reconstructing the index will be achieved by referencing the partition in the index and not the entire system.

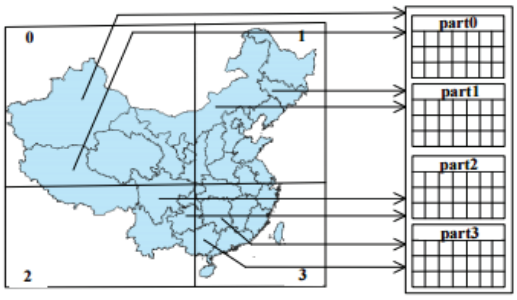
5.1. Partitioning

Unlike relational databases, where the data space can be partitioned using methods like hash partition, list partition, compound partition etc., spatial objects are different from these common databases in the sense that they are multi-dimensional and are co-relational in the space meaning that the longer the distance between two objects, the lesser the influence is [24]. Based on the heuristics above, some of the partitioning techniques for classical data have proven unsuitable. Therefore, we group the spatial data by their spatial locality on the *n*-dimensional (we have used 2-dimensions for simplicity) space by implementing the *str* partitioning strategy and we store different parts of data in different *spaces or disks* with that grouping. With this, it will be easy to get the Minimum Boundary Rectangle (MBR) of the spatial objects in database. Thus, in other to management storage, the partitioned data set can easily be updated or deleted in a relatively small bits without having to rearrange the entire system. The partitioning strategy we have employed ensures that nearby spatial objects are stored unto close partitions and not into different storage partitions which destroys the spatial co-relationship. Sort-tile-recursive algorithm (*str*) splits the space bottom-up recursively, i.e., it partitions the indexing tree recursively to the **m** MBRs of **P** spatial objects where equal amount of **m** are placed in each partition. In this work, this means that the datasets are shared among **C** processors and each has its own $\mathbf{P_C}$ **collection**. Where P_C is the total number of points in each processor, grouped into **m** MBRs.

Figure 9 shows a simple description of how the partitioning is achieved. The algorithm starts by initially splitting the objects into some sub-sets vertically in ***y*** direction and then horizontally in the ***x*** direction (and does same for all other dimensions) with the splitting satisfying two conditions as below:

- nearby objects are placed in the same vertical or horizontal partition
- each partition contains equal amount (or size) of spatial objects.

Figure 9. A typical example of partitioning of spatial data



Source: [24].

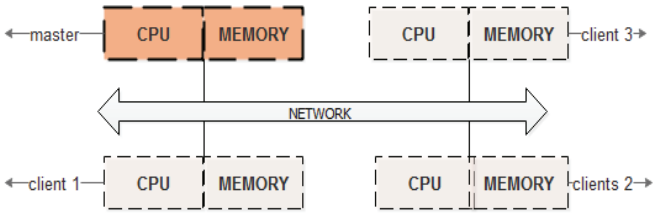
5.2. Storage

In order to preserve the spatial proximity that exist between the spatial objects, the data objects are stored using methods that *improves physical clustering*, this will ensure an improved efficiency for data access. Dimension reduction methods like Morton curves and Hilbert curves typically perform certain functions to map multidimensional data into one dimension while preserving locality of the data points. Following this mapping structure, any one-dimensional data structure such as B-tree can then be used to store the data. In this case we are using the sort-tile-recursive [60] sorting technique. Once the data are sorted into this ordering, we construct a bulk loaded static X-tree to store the data, without needing to do the one dimensional transformation like in the case of Morton and Hilbert curves. Though several researches has gone into storing spatial object by computing an improved natural clustering arithmetic for example, they all focus on dimension reduction and point transformation before applying a general index method into the encoded spatial objects in other to improve performance. We have decided to apply the *str* partitioning technique for large spatial data set storage to reduce index and storage time complexity.

5.3. Architecture

Our choice of the 'shared-nothing' architecture is based on the fact that the framework has high scalability which can go up to hundreds and possibly thousands of computers. Figure 10 is an example overview of the underlying architecture.

Figure 10. Proposed hardware architecture



5.4. Problem identification (PaX-DBSCAN)

The setup consists of a set of computers ***C*** connected via a high speed network, thus a typical problem can be seen as in Figures 11-13.

Figure 11. Problem statement

Initial Problem Statement
Given:
A set of points (*n-dimensional*) in a database say **P** such that **P = {P1, P2,...,P_n}**
A set of computers N such that ***N = {N1, N2,...,Nn***} connected via a high performance computing infra-structural network
Find the clusters (density-based) which obeys a given ***Eps*** and ***MinPts*** constraint.

Figure 12. Sort – tile – recursive algorithm

Sort-Tile-recursive Pseudocode:
P = the count of high dimensional objects in a 2d Cartesian plane.
Let ***N*** = the total number of available computer.
Let **m** = the maximum capacity of a node (number of node entries that can fit leaf or non-leaf node).
Let **n** = dimension
// J = **P** / **m** = the estimated total number of leaves required.
Step 1: by using the x-coordinate as a key; sort the objects (rectangles) based on the x-coordinate of their centre.
Step 2: Determine the maximum node entries.
Step 3: Order the sorted rectangle into **J = ⌈ P / m ⌉.**
Step 4: Divide the sorted rectangles into **r** groups of vertical slices.
- For two dimensions **r =** $\sqrt{J}$.
- For dimensions more than two, let **p** = dimension, **r =** $J^{1/p}$.

step 5: Sort the new group r groups again based on y – coordinate of the rectangles centre into
Output:
After loading the **r** groups of rectangles into nodes (**pages**) the output = (**MBR, Node Id**) for each leaf level node that loaded into a temporary file to be processed in the second phase two of the aX-tree algorithm.

Figure 13. Proposed steps for clustering

Basic clustering steps:
Divide the input (**P**) into **r** partitions such that **k = k1, k2…, k_r** and distribute these partitions to the ***N*** available computers.
Run the proposed DBSCAN clustering algorithm in each partitions concurrently
//the input parameter for the DBSCAN deduced from section 3.1 is (**k_j, EPS, minpts**).
Finally ***combine or merge*** the clusters from the partitions into a global cluster for the entire database.

5.5. Building distributed index

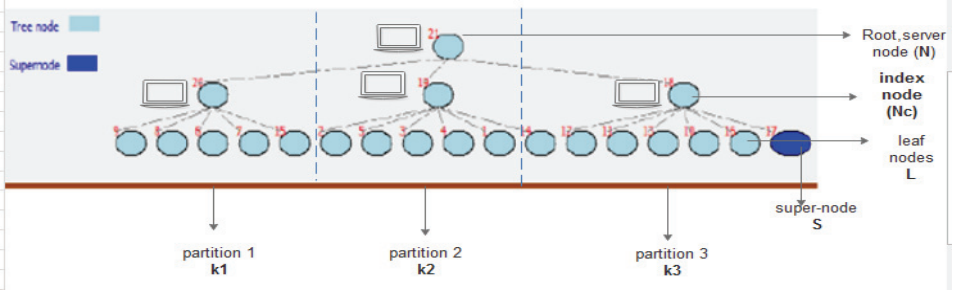
The X-tree (as review in Section 5.5.2) has proven performance measure on high dimensional data and has shown to be robust therefore, we have chosen the X-tree structure as our database link. Access to distributed data on the network can be achieved efficiently by replicating the **aX-tree** index on all the index nodes (computers) based on the assumption that all nodes contains equal amount of points, depending on the value of **m** (maximum capacity of each storage block).

According to Zhao et al. [34], there are several techniques for data *partitioning in parallel DBMS*. Assuming that data will be partitioned onto ***N*** disks, such as D0, D1, … , Dn-1, then the partitioning procedure is such that a map layer like the one shown in section 6.1.A (***ml***) or region (***rn***) or a set of data points (**P**) like the one in Figure 12b is entered onto the *server* and then the size of objects including the real data size is computed in addition, the required indices size is also computed after the partitioning. In this work, we adopted the *range partitioning* strategy. A significant sample of the dataset is selected and the midpoints of complex objects is computed to further reduce data size. We distribute each vertical segment (global leaf node ***gl***) of the partitioning dimension onto different (**r**) disk. We partition the space (according to the available dataset, the storage size and cluster scale − typically set to 64MB) and each represents one sub-region (***srn***). Initially we create the global index **GI** in four (4) simple steps. The *first step* is to sort the rectangles based on the partitioning dimension then *secondly* we calculate the maximum node entry for each disk by computing ***j = P/m***. *Thirdly*, we compute the value of **r** using sort-tile-recursive (str) partitioning strategy and then *partition* the data space into **r** slices. Then we associates each of these sub-region to one of the available computing resource

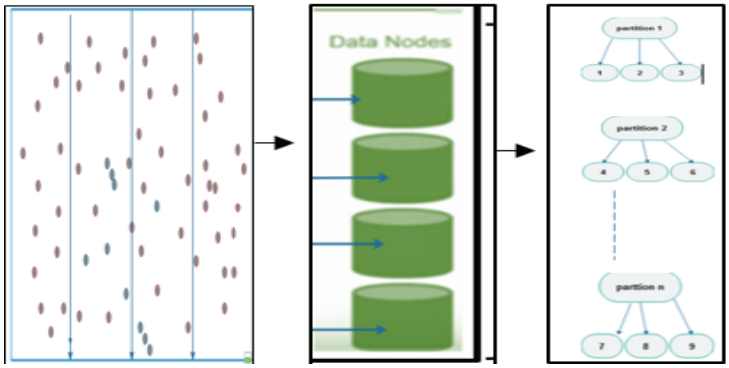
(node). In each partitions the local index (**LI**) is created, the rectangles are further sorted based on the other dimension (or dimensions) and packed in groups of **m** into their minimum bounding rectangles (MBR). The MBRs are further packed into the index or parent nodes in the sizes of 64MB, 128 Mb or whatever the available block of each node. The idea is such that geographically contiguous neighbouring data should be stored into the same node (block). The partitioning allocates ach of the regions to a processor.

Figure 14. Proposed system

(a) partitioning the data into tree nodes



(b) distrusting the tree among the data nodes

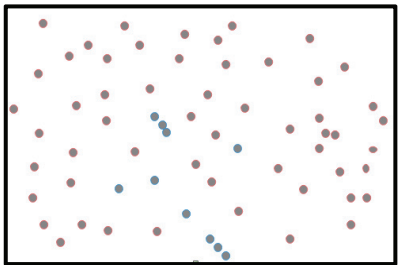


The distributed indexing structure adopted for this work see Figure 14, is similar to what is described in the work of Kamel and Faloutsos [61]. Here the rectangles (data) are distributed by assigning them to the different nodes using a *range* function obtained by simply comparing the vertices of the new data as against the stored leaf of the global index. The space is divided into $\mathbf{k_i}$ partitions, where i = 1 … **r**. The tree is guaranteed to be balanced because there would be equal number of rectangles in each node. The root node remains in main memory of the server while other nodes are distributed across **N** nodes. At the

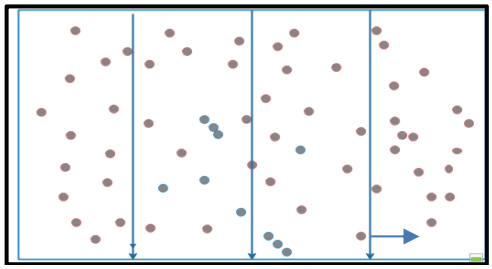
first instance, we take a sample of the data and the CenterPoint for complex objects, with the record values (*spatial attributes and references*) taken into consideration. The partitions of the rectangles are then distributed onto the computers ***C1***, ***C2***, ... , ***CN*** (in the case of ***N*** computers).

Figure 15. Partitioning

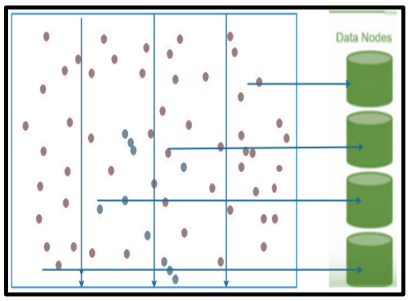
(a) data points



(b) vertical partitions of **r** slices



(c) storing the partitions to the available data nodes



The MBRs of the leaf nodes are partitioned so that nearby rectangles are in the same partition with almost same size for each partition. This partitioning strategy is achieved through the sort tile recursive (*str*) partition algorithm. The *str* algorithm according to [60] is a sort-based loading method for manage spatial and multidimensional data. It is simple and efficient bulk-loading strategy. The algorithm was proposed by Leutenegger et al. [57] and is described below:

- ***Super-node:*** After we partition the area to slices, we group the objects according to the maximum node entries. If the last group is less than the minimum allowed, then we extend to *super node*. But this is only on the *leaf level*. Note, the justification for creating the *super node* is to handle cases of highly skewed distributed data (which is very typical of spatial data), because in the case of uniformly distributed data the MBRs are guaranteed to contains same amount of data.

- **Leaf node entry** → (***O**Id*, ***MBR***): ***O**Id* is the tuple identifier for referring to an object in the database. ***MBR*** describes the smallest bounding ***n*** dimensional region around the data objects (for a *2d* - space, the value of ***MBR*** will be of the form – *xlow*, *xhigh*, *ylow*, *yhigh*, and for *3d* space – *xlow*, *xhigh*, *ylow*, *yhigh*, *zlow*, *zhigh*).
- **Non-leaf node entry** → (***Cp***, ***MBR***, ***P**Id*): ***Cp*** is a (child) pointer to a lower level node and ***MBR*** is the he rectangle that enclosing it (which covers all regions in child node). ***P**Id*** identifies the partition (computing node) where the object is stored.

Figure 16. Pre-processing step

Algorithm 1: Pre-processing

Start:

1. **Take** a sample of data from the large dataset
   // the sample can be chosen as a percentage (1, 2 or any percentage of the data, though 1 is a good choice) of the given dataset for point data **P**, but in the case of spatial object (objects with extent), we could convert shapes (lines, regions, areas) *to points* by obtaining their mid-points.
2. **Find** the centroid of the complex shapes (regions, rectangles, lines etc.), from the sample using the simple equation below. Note the formula considers the bounding rectangle of the spatial **object** only.

   $$C_{x,y} = \left(\frac{x2 - x1}{2}, \frac{y2 - y1}{2}\right)$$

   //In other cases, getting the centroid on a polygon based on the number of **j**-vertices will generally require a different formula.
3. **Calculate r.**
4. **Divide** the sample space into **r** vertical slices.
5. **Bulk load the aX-tree into main memory of the server**
   // the extended node (***super-node***) is applied only in the first level (the bottom of the tree) to avoid the problems of hyper rectangles overlap.
   Step 1: by using the x-coordinate as a key; sort the objects (rectangles) based on the x-coordinate of their centre for complex objects.
   Step 2: Sort the new group r groups again based on y –coordinate of the rectangles centre into.
6. **Output:**
   After loading the **r** groups of rectangles into nodes (**pages**) the output = (MBR, Node Id) for each leaf level node that loaded into a temporary file to be processed in phase two of the aX-tree algorithm.

Assumptions are as follows:

1) The sample data (and their extracted centroids for extended complex objects, e.g. lines, polygons regions, etc.) reduces the size of available data therefore making it possible for a single machine (the master to managing the indexing and partitioning);
2) The data is bulk loaded into an in-memory sort-tile − recursively (str) loaded X-tree for creating the global partitioning – i.e. data block location.

Figure 17. Loading the tree

Algorithm 2: Loading

// loading the collection of the MBRs of sorted spatial objects from the temporary file in phase one.

Start:

Step 1. Create leaf nodes the basement level (level l=0) While (there are more rectangles) Generate a new X-tree node, Assign the next **m** rectangles to this node.
//During node creation prevents any split that cause overlapping, by extend one super-node in the current level.
Step 2. Create nodes at higher level (l + 1) While (there are > 1 nodes at level l) Sort nodes at level l≥0 on ascending creation time Repeat.
Step 3. Return Root.

Figure 18. Distributing data to various partitions

Algorithm 3: Partitioning

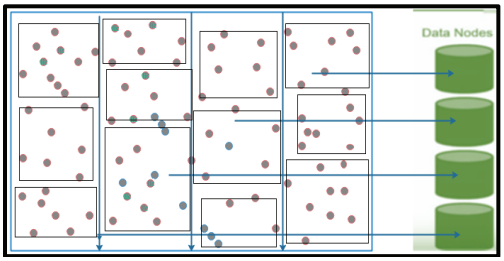
Start

// Using the boundaries of each leaf node as the boundary of the MBR of the leave nodes, then scan the data in parallel and assign each record to its overlapping partition.

i. **Store** the location of each partition (***gl***) and their range in a file in the server in the form (***O*** $_{Id}$, ***MBR***, **N** $_{Id}$) and call it **N** $_{k}$. (**i.e. k** partition in **N** computer)
// ***O*** $_{Id}$ is a (child) pointer to a lower level node.
// ***MBR*** is the he rectangle enclosing it (which covers all regions or points in child node).
//**N** $_{Id}$ ***identifies*** the partition (computing node) where the object is stored.
ii. **Partition** the data into the ***N*** number of computers by computing **r** and mapping out its horizontal range for partitions **k** $_{i}$. **(i = 1….r).**
iii. **For** each new entry point (from the new dataset) the server compares the range its spatial attribute to the range of the **MBR of the global leaf** (***gl***), and sends it to the right node partition as shown in figure 18.

Step iv is repeated until the entire dataset has been fully distributed.

Figure 19. Points grouped into rectangles based on the value of the leaf node capacity (**m**) which could be 64mb or higher



5.6. Point clustering (PaX-DBSCAN algorithm)

Figure 20. The aX-tree

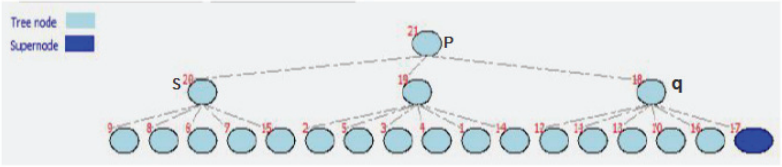


Figure 21. Finding k-nearest neighbour (ϵ)

Algorithm 4: How to find the neighbourhood (ϵ)

A point spatial database **P** partitioned into **N** nodes {assuming that records consists of real number values}
Let
s and **q** be any two point nodes on the tree
k = an arbitrary number
Let $\mathbf{s_i}$ be the query object (spatial).
Then:
For all **aX**-tree node say **v**.
The distance between the minimum bounding rectangles of **v** (**MBRv**) and the query object $\mathbf{s_i}$ given by Distance ($\mathbf{s_i}$, **MBRv**) is ≤ distance between the query point and any other object S indexed under **v** i.e. distance ($\mathbf{s_i}$, **Pj**) for all **j** indexed under the node **v**.
Thus:
If we call the calculated distance between p and another object **b** indexed under node **q** distance ($\mathbf{s_i}$, **b**) and Distance ($\mathbf{s_i}$, **MBRv**) = **t**.
Then
If **t** > distance ($\mathbf{s_i}$, b)
1. **Then**
2. Stop search for **v**
3. Take a different path
4. **Else**
5. **Return** closest distanced neighbours (ϵ)

Figure 22. aX-DBSCAN

Algorithm 5: Clustering (aX-DBSCAN AGORITHM)

```
For each node
   1. Find a neighbourhood value (say ∈ > 0).
   2. Determine Φ > 0.
   // Φ is the parameter that determines which cluster
   is dense.
 // value of Φ = (at least) m +1 so as to avoid clusters
that has only one object.
 // m is the maximum capacity of an aX-tree node.
   3. Compute the value of ∈.
   // the value of ∈ (nearest neighbour to point p) is
   deducted from the aX-tree as follows as in step i
   above:
      i.    Find Bi = {p ∈ S: d(pi, p) ≤∈ }
      // (d = distance between p and pi for all i = 1,
      2….m)
      ii.   If | Bi | ≤ Φ, THEN
      iii.  REJECT Bi
      // Bi is an outlier
            ELSE
      iv.   Find the relationship (Bi ∪ Bj ≠ Ø)
      v.    Repeat iv until there is no more union
   4. Return cluster to the master (cl, Nid, Pb, Pn).
```

For algorithm 5, a temporary file is initially created that stores all the calculated distance between all the points from the distance function: **Distance** (**s**, **q**) ≥ *distance* (MBR (**s**), MBR (**q**)) for any point between and **q** in the tree. Thus, for any two points in **s**, the nearest neighbour of **t** in **s** is defined by e_i : ∀ $e_{i+1} \in$ **s**, $distance\ (t, e) \leq distance\ (t, e_{i+1})$ **and** The **K-nearest neighbour (E)** of t in **s**, is all *e* ***with distances*** ≤ ***kth distance*** *from* ***t; where k = m*** = *maximum points capacity of a node.* This procedure will make it faster to build the clusters by merely comparing distances from the distance table in when processing **Bi** at each iteration. At the end of each computation, all the node directs the sub-result to the master node in other to process the final result and presents the PaX--DBSCAN cluster. This process takes the description below: Each process of DBSCAN executes concurrently on the parameter (P_j , **E**, **Φ**), for j = 1, 2, ... , N,) and finishes at the same time. Then each local ***task identifier*** (**TID**) is read by the master which extracts each unique values. The values are then compared. For each processor **C**, each point $\mathbf{p} \in \mathbf{P_C}$ is a local point, thus other points not found in $\mathbf{P_C}$ are referred to as remote points. Being a distributed memory, any other partition Ki ≠ Kj, 1 ≤ j ≤ N, 1 ≤ j ≤ N is not directly visible to the processor **C**.

Each client reports the new local cluster likely with noise through the TID to the master from their previous iteration, the server stores unto a temporary

file. The server then broadcasts the value of the points at the border and the noise points. Using this information, each processor processes the new data with their existing cluster, by this means the border points may form new clusters with the other unvisited points in each of the processor ($\mathbf{P_C}$) merge with existing cluster. The process continues until all points are visited for all $\mathbf{P_{Cs}}$. At the end of the iteration process, the server compiles the result and filter out noise.

The system is designed in such a way that data within a partitioned region are stored on one of the computer nodes on the network, and all spatial data are then distributed across the different local cluster according to their geographical space. At this point, the original tree has **K** data pages. The pages (nodes) are distributed onto the **N** computers with **m** data pages on each computer. Querying starts on the master node and request is sent across the client computers for any impending task. For each query, the master computes a *cost model* to know which client node contains the required triangle, and sends the query to that node.is performed by starting at the root and computing all entries whose rectangle qualifies. In the underlying architecture adopted for this project, all the network task are identified by an integer ***task identifier*** (TID) – described fully in Geist et al. [70] – and messages are sent to and received from these TIDs across the network. TIDs are supplied by the local node and are not user chosen must be unique across the entire virtual machine. Although system encodes information into each TID, the user is expected to treat the TID as opaque integer identifiers. The system contains several routines that return TID values so that the user application can identify other tasks in the system Geist et al. [70].

Summary covers the following comments:

- The indexing bulk-loads the x-tree and take only the leaf level.
- This procedure ends the global index with a flat partitioning, therefore new records fall into any one of the partitions.
- The global index is stored in main memory (as the boundary of each partition), using this technique, only the partition covering the query point will be processed.
- The second layer index partitions each of the index blocks to the maximum capacity of each of the data node.
- The data is partitioned into processors (**Nc**).
- Each processor builds its own local cluster and sends each intermittent result to the server.
- The server then rebroadcasts the border points and noise, using this information, the clients compare the points and decide to either merge the clusters if there core points and density reachable points or rather create a new cluster.

- The operation of our system can be described as intra-operational parallelism where the **aX-tree** index is parallelized over the N processors. This can be achieved by partitioning the data among the processors. After the initial local processing, each processor returns its results to the host (master) then the result is combined for the eventual result.

Conclusions and future work

Spatial databases keep information about the location and the tasks of spatial data mining need the spatial relation to be indexed before it can perform optimally. In this work, we have studied the usefulness of parallelism in the improvement of clustering task in a spatial database. And we have looked into building a parallel adjusted X-tree (PaX-tree) for this purpose. We discussed indexing structure for large spatial datasets, bulk loading spatial structure and applying the idea of parallelism to their improvement. We proposed an improved algorithm for the DBSCAN clustering and we also proposed a new algorithm for adjusting the existing X-tree.

Bulk-loading spatial data using the popular *MapReduce framework* is intended to overcome the problems associated with parallel bulk-loading of tree-based indexes which has the disadvantage that the quality of produced spatial index decrease considerably as the parallelism increases. Therefore as our future work, we intend to design an algorithm for parallel bulk loading of X-tree using Hadoop technology. Secondly we have to also evaluate our current proposed system in other to ascertain its efficiency.

References

[1] I. Lungu, A. Velicanu, *Spatial Database Technology Used in Developing Geographic Information Systems*, The 9th International Conference on Informatics in Economy – Education, Research & Business Technologies, Academy of Economic Studies, Bucharest, 7-8 May 2009, pp. 728-734.

[2] GIS Geography, *GIS Spatial Data Types: Vector vs Raster*, 2016, http://gis geography.com/spatial-data-types-vector-raster/ (accessed: 22.11.2016).

[3] A. Gottlieb & G.S. Almasi, *Highly Parallel Computing*, Benjamin-Cummings, Redwood City, CA 1989.

[4] R.H. Güting, *C*, "The VLDB Journal − The International Journal on Very Large Data Bases" 1994, Vol. 3(4), pp. 357-399.

[5] A. Velicanu Belciu, S. Olaru, *Optimizing Spatial Databases*, 2011, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1800758 (accessed: 20.11.2016).

[6] K. Kailing, H.P. Kriegel, P. Kröger, *Density-Connected Subspace Clustering for High-Dimensional Data* [in:] *Proceedings 4th SIAM International Conference on Data Mining*, Vol. 4, Lake Buena Vista, FL 2004, pp. 246-257.

[7] P. Verma, Y.K. Jain, *High Dimensional Object Analysis Using Rough-Set Theory and Grey Relational Clustering Algorithm*, "International Journal of Advanced Research in Computer and Communication Engineering" 2016, Vol. 5(5), pp. 404-410.

[8] J. Liu, *New Approaches for Clustering High Dimensional Data*, Doctoral dissertation, University of North Carolina, Chapel Hill 2006.

[9] T. Liu, *Fast Nonparametric Machine Learning Algorithms for High-Dimensional Massive Data and Applications*, No. CMU-CS-06-124, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 2006.

[10] J. Han, M. Kamber, *Data Mining Concepts and Techniques*, Morgan Kaufmann Publishers, San Francisco, CA 2001. pp. 335391.

[11] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, *From Data Mining to Knowledge Discovery in Databases*, "AI Magazine" 1996, Vol. 17(3), p. 37.

[12] M.M.A. Patwary, D. Palsetia, A. Agrawal, W.K. Liao, F. Manne, A. Choudhary, *A New Scalable Parallel DBSCAN Algorithm Using the Disjoint-Set Data Structure* [in:] *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, Salt Lake City, UT, November, 2012, pp. 1-11.

[13] M. Ester, H.P. Kriegel, J. Sander, X. Xu, *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, "Kdd" August 1996, Vol. 96, No. 34, pp. 226-231.

[14] *Rough Sets and Current Trends in Computing*, M. Szczuka, M. Kryszkiewicz, R. Jensen, Q. Hu (eds.), Proceedings of the 7th International RSCTC Conference, LNAI 6086, Springer Verlag, Berlin Heidelberg 2010, pp. 60-69.

[15] S. Vijayalaksmi, M. Punithavalli, *A Fast Approach to Clustering Datasets Using DBSCAN and Pruning Algorithms*, "International Journal of Computer Applications" 2012, Vol. 60(14).

[16] P. Berkhin, *A Survey of Clustering Data Mining Techniques* [in:] *Grouping Multidimensional Data*, J. Kogan, Ch. Nicholas, M. Teboulle (eds.), Springer Verlag, Berlin-Heidelberg 2006, pp. 25-71.

[17] M. Ester, H.P. Kriegel, J. Sander, *Knowledge Discovery in Spatial Databases* [in:] *Mustererkennung 1999*, Springer Verlag, Berlin-Heidelberg 1999, pp. 1-14.

[18] N. Mamoulis, *Spatial Data Management*, 1st ed., Morgan & Claypool Publishers, US 2012.

[19] S. Berchtold, D.A. Keim, H.P. Kriegel, *An Index Structure for High-Dimensional Data*, "Readings in Multimedia Computing and Networking" 2001, pp 451-462.

[20] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, Morgan Kaufmann, Burlington, MA 2006.

[21] D. Taniar, C.H. Leung, W. Rahayu, S. Goel, *High Performance Parallel Database Processing and Grid Databases*, Vol. 67, John Wiley & Sons, New York 2008.

[22] D. Jagli, *Parallel Database*, 2013, http://www.slideshare.net/dhanajagli1/parallel-database> (accessed: 26.10.2016).

[23] A. Papadopoulos, Y. Manolopoulos, *Parallel Bulk-Loading of Spatial Data*, "Parallel Computing" 2003, Vol. 29(10), pp. 1419-1444.

[24] Z. Qin, Z. Ershun, H. Yaohuan, *Research on Parallel Bulk-Loading R-Trees Based on Partition Technology of Database*, "The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences" 2008, Vol. 37, Part B4.

[25] D. Achakeev, M. Seidemann, M. Schmidt, B. Seeger, *Sort-Based Parallel Loading of R-Trees* [in:] *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, ACM, Redondo Beach, CA 2012, pp. 62-70.

[26] B. Barney, *Introduction to Parallel Computing*, "Lawrence Livermore National Laboratory" 2010, Vol. 6(13), pp. 1-34.

[27] F. Provost, T. Fawcett, *Data Science for Business: What You Need To Know about Data Mining and Data-Analytic Thinking*, O'Reilly Media, Sevastopol, CA 2013.

[28] *Apache Hadoop*, 2016, http://hadoop.apache.org/index.html (accessed: 28.10.2016).

[29] R. Lammel, *Google's Mapreduce Programming Model – Revisited*, "Science of Computer Programming" 2008. Vol. 70, pp. 1-30.

[30] K. Lee, R.K. Ganti, M. Srivatsa, L. Liu, *Efficient Spatial Query Processing for Big Data* [in:] *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems – SIGSPATIAL*, Dallas − Fort Worth, TX 2014, pp. 469-472.

[31] S. Shekhar, V. Gunturi, M.R. Evans, K. Yang, *Spatial Big-Data Challenges Intersecting Mobility and Cloud Computing* [in:] *Proceedings of the 11th ACM International Workshop on Data Engineering for Wireless and Mobile Access − MobiDE'12*, Scottsdale, AZ 2012.

[32] S. Shekhar, M.R. Evans, V. Gunturi, K. Yang, D.C. Cugler, *Benchmarking Spatial Big Data* [in:] T. Rabl, M. Poess, C. Baru, H.-A. Jacobsen (eds.), *Specifying Big Data Benchmarks*, Springer Verlag, Berlin-Heidelberg 2014, pp. 81-93.

[33] Y. Wang, S. Wang, D. Zhou, *Retrieving and Indexing Spatial Data in the Cloud Computing Environment* [in:] *The First International Conference on Cloud Computing*, Springer Verlag, Berlin-Heidelberg 2009.

[34] L. Zhao, L. Chen, R. Ranjan, K.K.R. Choo, J. He, *Geographical Information System Parallelization for Spatial Big Data Processing: A Review*, "Cluster Computing" 2016, Vol. 19(1), pp. 139-152.

[35] Maitrey S., Jha, C.K. (2015). *Handling Big Data Efficiently by Using Map Reduce Technique*, Paper presented at the The 2015 IEEE International Conference on Computational Intelligence & Communication Technology (CICT), Ghaziabad, IN.

[36] Y. Zhong, J. Han, T. Zhang, Z. Li, J. Fang, G. Chen, *Towards Parallel Spatial Query Processing for Big Spatial Data* [in:] *2012 IEEE 26th International Parallel*

and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE, Piscataway, NJ 2012, pp. 2085-2094.

[37] W. Tang, W. Feng, *Parallel Map Projection of Vector-Based Big Spatial Data: Coupling Cloud Computing with Graphics Processing Units*, "Computers, Environment and Urban Systems" 2014.

[38] H. Tan, W. Luo, H. Mao, L.M. Ni, *On Packing Very Large R-Trees* [in:] *IEEE 13th International Conference on Mobile Data Management*, Bengaluru, Karnataka 2012, pp. 99-104.

[39] S. Li, S. Dragicevic, F.A. Castro, M. Sester, S. Winter, A. Coltekin, C. Pettit, *Geospatial Big Data Handling Theory and Methods: A Review and Research Challenges*, "ISPRS Journal of Photogrammetry and Remote Sensing" 2016, Vol. 115, pp. 119-133.

[40] P. Ogden, D. Thomas, P. Pietzuch, *AT-GIS: Highly Parallel Spatial Query Processing with Associative Transducers*, SIGMOD'16, , San Francisco, CA June 26 − July 1, 2016.

[41] M. Chen, X. Gao, H. Li, *Parallel DBSCAN with Priority R-Tree* [in:] *The 2nd IEEE International Conference on IEEE Information Management and Engineering (ICIME)*, Chengdu 2010, pp. 508-511.

[42] B. Welton, E. Samanas, B.P. Miller, *Mr. Scan: Extreme Scale Density-Based Clustering Using a Tree-Based Network of GPGPU Nodes* [in:] *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ACM, Denver, CO November 2013, p. 84.

[43] M. Noticewala, D. Vaghela, *MR-IDBSCAN: Efficient Parallel Incremental DBSCAN Algorithm using MapReduce*, "International Journal of Computer Applications" 2014, Vol. 93(4).

[44] X. Xu, J. Jäger, H.P. Kriegel, *A Fast Parallel Clustering Algorithm for Large Spatial Databases* [in:] *High Performance Data Mining*, Springer Verlag, US 1999, pp. 263-290.

[45] L.S. El-Sayed, H.M. Abdul-Kader, S.M. El-Sayed, *Performance Analysis of Spatial Indexing in the Cloud*, "International Journal of Computer Applications", 2015, Vol. 118(4), pp. 1-4.

[46] W.W. Song, B.X. Jin, S.H. Li, X.Y. Wei, D. Li, F. Hu, *Building Spatiotemporal Cloud Platform for Supporting GIS Application*, "ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences" 2015, Vol. 2(4), pp. 55-62.

[47] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, *Spark: Cluster Computing with Working Sets*, "HotCloud" 2010, Vol. 10, pp. 1-7.

[48] S. You, J. Zhang, L. Gruenwald, *GPU-Based Spatial Indexing and Query Processing Using R-Trees* [in:] *BigSpatial'13 Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, Orlando, FL 2013.

[49] A. Singh, D. Garg, *Implementation and Performance Analysis of Exponential Tree Sorting*, "International Journal of Computer Applications" June 2011, Vol. 24, No. 3, pp. 34-38.

[50] K. Akkaya, A. Yazici, *An Indexing Method for Spatial Databases*, "XIV. International Symposium on Computer and Information Sciences (ISCIS'99)" April 1999.

[51] V. Gaede, O. Günther, *Multidimensional Access Methods*, "ACM Computing Surveys (CSUR)" 1998,Vol. 30(2), pp. 170-231.

[52] T. Lee, S. Lee, *OMT: Overlap Minimizing Top-down Bulk Loading Algorithm for R-Tree*, "CAiSE Short Paper Proceedings" June 2003, Vol. 74, pp. 69-72.

[53] X. Liu, J. Han, Y. Zhong, C. Han, X. He, *Implementing Webgis on Hadoop: A Case Study of Improving Small File I/O Performance on HDFS* [in:] *2009 IEEE International Conference on Cluster Computing and Workshops* IEEE, New Orleans, LA August 2009, pp. 1-8.

[54] Y. Liu, N. Jing, L. Chen et al., *Parallel Bulk-Loading of Spatial Data with MapReduce: An R-Tree Case*, "Wuhan University Journal of Natural Science" 2011, 16, pp. 513.

[55] N. Roussopoulos, D. Leifker, *Direct Spatial Search on Pictorial Databases Using Packed R-Trees*, "ACM SIGMOD Record" May 1985, Vol. 14, No. 4, pp. 17-31.

[56] I. Kamel, C. Faloutsos, *On Packing R-Trees*, "Proceedings of the Second International Conference on Information and Knowledge Management" December 1993, pp. 490-499.

[57] S.T. Leutenegger, M.A. Lopez, J. Edgington, *STR: A Simple and Efficient Algorithm for R-Tree Packing* [in:] *Proceedings of 13th International Conference on Data Engineering*, IEEE, Graz, Austra April 1997, pp. 497-506.

[58] D. Achakeev, B. Seeger, P. Widmayer, *Sort-Based Query-Adaptive Loading of R-Trees* [in:] *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, ACM, Maui, HI October 2012, pp. 2080-2084.

[59] S. Hua, J. Nan, H. Bin, L. Heng, Z. Jin, *GPU-Based Parallel Bulk Loading R-Trees Using STR Method on Fine-Grained Model[J]*, "Geomatics and Information Science of Wuhan Universty" 2014. Vol. 39(9), pp. 1068-1073.

[60] B.C. Giao, D.T. Anh, *Improving Sort-Tile-Recusive algorithm for R-tree packing in indexing time series*, "2015 IEEE RIVF International Conference on IEEE Computing & Communication Technologies-Research, Innovation, and Vision for the Future (RIVF)" 2015, pp. 117-122.

[61] I. Kamel, C. Faloutsos, *Parallel R-Trees*, "Research Showcase CMU" 1992, Vol. 21, No. 2, pp. 195-204.

[62] E.G. Hoel, H. Samet, *Performance of Data-Parallel Spatial Operations*, "Proceedings of VLDB Conference" 1994, pp. 156-167.

[63] B. Schnitzer, S.T. Leutenegger, *Master-Client R-Trees: A New Parallel R-Tree Architecture*, "Proceedings of SSDBM Conference" 1999, pp. 68-77.

[64] P. Apostolos, M. Yannis, *Parallel Bulk-Loading of Spatial Data*, "Journal of Parallel Computing" 2003, Vol. 29(10), pp. 1419-1444.

[65] L. Luo, M.D.F. Wong, L. Leong, *Parallel Implementation of R-Trees on the GPU* [in:] *17th Asia and South Pacific Design Automation Conference*, Sydney 2012, pp. 353-358.

[66] S. Berchtold, D. A. Keim, H.-P. Kriegel, *The X-tree: An Index Structure for High--Dimensional Data* [in:] *Proceedings of the 22nd VLDB Conference*, Mumbai, India 1996, pp. 28-39.

[67] P. Ciaccia, M. Patella, P. Zezula, *M-tree An Efficient Access Method for Similarity Search in Metric Spaces* [in:] *Proceedings of the 13th International Conference on Very Large Data Bases*, Athens 1997.

[68] K.S. Candan, M.L. Sapino, *Data Management for Multimedia Retrieval*, Cambridge University Press, Cambridge 2010.

[69] Y. Manolopoulos, A. Nanopoulos, A.N. Papadopoulos, Y. Theodoridis, *R-Trees: Theory and Applications*, Springer Science and Business Media, Berlin-Heidelberg 2010.

[70] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM: Parallel Virtual Machine − A User's Guide and Tutorial for Networked Parallel Computing*, 3rd ed., The MIT Press, Cambridge, MA − London, England 1996.

PaX-DBSCAN: PROPOZYCJA ALGORYTMU DLA DOSKONALONEGO GRUPOWANIA

Streszczenie: W artykule autorzy skupiają swoją uwagę na zastosowaniu techniki przetwarzania równoległego przy wykorzystaniu struktur drzewiastych X-tree i algorytmu bulk loading. Zaproponowano nowy algorytm przetwarzania równoległego DBSCAN i drugi algorytm dla rozszerzania struktur indeksowania przestrzennego.
Algorytm grupowania DBSCAN jest efektywnym algorytmem grupowania dla Systemów Przestrzennych Baz Danych, który ma możliwość wykrywania zakłóceń i nie wymaga znacznej liczby skupień wcześniej ustalonych, jednakże działanie algorytmu zmienia się, gdy rozmiar danych jest duży. Ten algorytm może nie działać optymalnie, jeśli niewłaściwe wartości są wybrane dla minpts i eps. Dlatego nowy zaproponowany algorytm powinien eliminować te ograniczenia.

Słowa kluczowe: struktura drzewiasta X-tree, indeks przestrzenny, rozdzielanie, przetwarzanie równoległe, algorytm bulk loading, przestrzenne bazy danych, grupowanie.