# Indexing

Jianzhong Qi and Rui Zhang

## Synonyms

Indexing big spatial data; Big spatial data access methods.

## Definitions

Consider a set of $n$ data objects $O = \{o_1, o_2, ..., o_n\}$. Each object is associated with a $d$-dimensional vector representing its coordinate in a $d$-dimensional space ($d \in \mathbb{N}_+$). *Indexing* such a set of data is to organize the data in a way that provides fast access to the data, for processing spatial queries such as *point queries*, *range queries*, *kNN queries*, *spatial join queries*, etc.

## Overview

The rapid growth of *location-based services* (LBS) has accumulated a massive amount of spatial data such as user GPS coordinates, which calls for efficient indexing structures to provide fast access to such data. Typical applications of spatial data include digital mapping services and location-based social networks, where spatial queries are issued by users such as *finding restaurants within 3 kilometers around Alice* or *finding other users within 300 meters around Alice*. Spatial indices have been studied extensively to support such queries. In the big data era, there may be millions of spatial queries and places of interests to be processed at the same time. This poses new challenges in both scalability and efficiency of spatial indexing techniques. Parallel spatial index structures are developed to address these challenges, which are summarized in the following section.

## Key Research Findings

Traditional spatial indexing techniques can be classified into *space-partitioning* techniques and *data partitioning* techniques. Space-partitioning techniques such as *quad trees* (Finkel and Bentley 1974) partition the data space into non-overlapping partitions where data objects in the same partition are indexed together. Data-partitioning techniques such as *R-trees* (Guttman 1984) partition the data set directly where data objects nearby are grouped into the same partition and indexed together. We focus on recent developments of the indexing techniques. Readers interested in the traditional spatial indices are referred to a detailed survey on this topic by Gaede and Günther (1998).

Spatial indices have been extended to parallel computing environments to cope with the scalability and efficiency issues. For example, the R-trees have been implemented over a shared-nothing (client-server) architecture. Koudas et al (1996) store the inner nodes of an R-tree on a server and the leaf nodes on clients. In this architecture, the inner nodes stored on the server forms a *global index*. At query processing, the server uses this index to prune the search space and locate the clients that contain the data objects being queried. Schnitzer and Leutenegger (1999) further create local R-trees on clients. This forms a two-level index structure where the global index is hosted on the server while the *local indices* are hosted on the clients. This is illustrated in Fig. 1, where there is a global index stored on the server and four local indices stored on four client machines $C_1$, $C_2$, $C_3$, and $C_4$.

More recent work on parallelizing spatial indices use the *MapReduce* and the *Spark* frameworks which have become standard parallel computation frameworks in the past decade. These frameworks hide the parallelism details such as synchronization and simplifies parallel computation into a few generic operations (e.g., *Map* and *Reduce*). The two-level index structure as described above is still commonly used, although some of the techniques proposed only use either the global index or the local indices. We describe a few representative index structures using these two frameworks next.

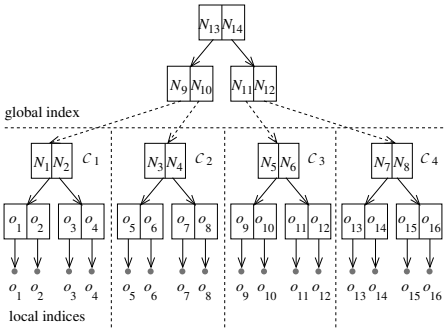


**Fig. 1** Two-level distributed indexing.

### MapReduce based Indexing

*Hadoop-GIS* (Aji et al 2013) is a *Hadoop* (an open-source implementation of the MapReduce framework) based spatial data warehousing system. It builds a two-level (global and local) index structure over the data objects to support parallel processing of spatial queries. The global index is pre-built by first partitioning the data space with a regular grid, and then further partitioning every grid cell containing more than a predefined number of data objects $n_{max}$ into two equi-sized cells at the middle

recursively until each cell contains no more than $n_{max}$ objects. Data objects in the same cell is associated with the same unique id and stored in the same block. Multiple cells are grouped into a block for storage to suit the block size in the *HDFS* (a distributed file system used by Hadoop). The *minimum bounding rectangles* (MBR) of the grid cells are stored in the global index. A local index is built at query time over the data objects assigned to the same machine for query processing.

*SpatialHadoop* (Eldawy and Mokbel 2013) extends Hadoop to provide built-in support for spatial queries. It also uses a global index and a local index. Grid based and R-tree based indices can both be used for the global index, where the grid cells and leaf nodes respectively correspond to partitions that are stored in HDFS blocks. A local index is build for each partition, which can also use grid or R-tree based indices. *AQWA* (Aly et al 2015) builds a global index only, which uses the k-d tree. This index also stores the number of data points of each partition to help query processing.

A few other studies (e.g., Nishimura et al (2011); Hsu et al (2012); Zhang et al (2014)) use *space-filling curves* such as the *Z-curve* (Orenstein and Merrett 1984) to map multidimensional coordinates to one dimensional values. This enables storing the data objects into *NoSQL* databases such as *HBase*, where the mapped coordinate value and the id of a data object are stored as a *key-value* pair. Data update and spatial queries can then be handled via the APIs of NoSQL databases.

When a hierarchical index such as the R-tree or the k-d tree is built with MapReduce, dynamic index building procedures that insert the data objects into the index individually lack efficiency. A common assumption is that the entire data set is available. In this case, the entire index structure over the data set can be *bulk-loaded* at once. A straightforward approach is to bulk-load the hierarchical index level by level, where each level incurs a round of MapReduce computation (Achakeev et al 2012). Sampling has been used to reduce the number of MapReduce rounds needed. Papadopoulos and Manolopoulos (2003) propose a generic procedure for parallel spatial index bulk-loading. This procedure uses sampling to estimate the data distribution, which guides the partition of the data space. Data objects in the same partition are assigned to the same machine for index building. SpatialHadoop uses a similar idea for R-tree bulk-loading, where an R-tree is built on a sample data set and the MBR of the leaf nodes are used for data partitioning. Agarwal et al (2016) also use a sampling based technique, but bulk-load k-d trees. This technique involves multiple rounds of sampling. Each round uses a sample data set small enough to be processed by a single machine. A k-d tree is built on the sample data set, and data objects are assigned to the partitions created by the k-d tree. For the partitions that contain too many data objects to be stored together, another round of sampling and k-d tree based partitioning is performed. Repeating the procedure above, it takes $O(\texttt{poly}\log_{\texttt{s}}\texttt{n})$ MapReduce rounds to bulk-load a k-d tree, where $s$ denotes the number of data points that can be processed by a single machine in the MapReduce cluster.

Spark based Indexing

*Spark* models a data set as a *resilient distributed dataset* (RDD) and stores it across machines in the cluster. Spatial indices on Spark focus on optimizing with the RDD storage architecture.

*GeoSpark* (Yu et al 2015) builds a local index such as a quad-tree on-the-fly for the data objects in each RDD partition, while the partitions are created by a uniform grid partitioning over the data space. *SpatialSpark* (You et al 2015) also builds a local index for data objects in each RDD partition, which can be stored together with the data objects in the RDD partition. SpatialSpark supports binary space partitioning and tile partitioning in addition to uniform grid partitioning over the data space. *STARK* (Hagedorn et al 2017) uses R-trees for the local indices while uniform grid partitioning and cost based binary space partitioning for creating the partitions. *Simba* (Xie et al 2016) uses global and local indices (e.g., R-trees). The global index is held in-memory in the master node while the local indices are held in each RDD partition.

## Examples of Application

Spatial indexing techniques are used in spatial databases to support applications in both business and daily scenarios such as targeted advertising, digital mapping, and location-based social networking. In these applications, the locations (coordinates) of large sets of places of interests as well as users are stored in a spatial database. Spatial indices are built on top of such data to provide fast data access, facilitating spatial queries such as *finding customers within 100 meters of a shopping mall* (to push shopping vouchers), *finding the nearest restaurants for Alice*, and *finding users living in nearby suburbs who share some common interests* (for friendship recommendations).

## Future Directions for Research

The indexing techniques discussed above does not consider handling location data with frequent updates. Such data is becoming more and more common with the increasing popularity of positioning system-enabled devices such as smart phones, which enables tracking and querying the continuously changing locations of users or data objects of interest (Zhang et al 2012; Ward et al 2014; Li et al 2015). How to update the spatial indices and to record multiple versions of the data (Lomet et al 2008) with a high frequency is non-trivial. While the parallel frameworks such as MapReduce scale well to massive data, the indices stored in the distributed data storage may not be updated as easily as indices stored in a standalone machine. A periodic rebuilt of the indices is an option, but may not keep the indices updated between rebuilds, and hence may provide inaccurate query answers. More efficiency index update techniques are in need. Another important direction to pursuit is efficient indexing techniques to support spatial data mining. Many data mining algorithms are computational intensive, which require iterative accesses to (high dimensional) data. Spatial indices designed to cope with high dimensional data (Zhang et al 2004; Jagadish et al 2005) and the special accessing patterns of such

algorithms would have a significant impact in improving the usability of such algorithms.

## References


Achakeev D, Seidemann M, Schmidt M, Seeger B (2012) Sort-based parallel loading of r-trees. In: Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data, pp 62–70

Agarwal PK, Fox K, Munagala K, Nath A (2016) Parallel algorithms for constructing range and nearest-neighbor searching data structures. In: Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS), pp 429–440

Aji A, Wang F, Vo H, Lee R, Liu Q, Zhang X, Saltz J (2013) Hadoop gis: A high performance spatial data warehousing system over mapreduce. Proceedings of the VLDB Endowment 6(11):1009–1020

Aly AM, Mahmood AR, Hassan MS, Aref WG, Ouzzani M, Elmeleegy H, Qadah T (2015) Aqwa: Adaptive query workload aware partitioning of big spatial data. Proceedings of the VLDB Endowment 8(13):2062–2073

Eldawy A, Mokbel MF (2013) A demonstration of spatialhadoop: An efficient mapreduce framework for spatial data. Proceedings of the VLDB Endowment 6(12):1230–1233

Finkel RA, Bentley JL (1974) Quad trees a data structure for retrieval on composite keys. Acta Informatica 4(1):1–9

Gaede V, Günther O (1998) Multidimensional access methods. ACM Computing Surveys 30(2):170–231

Guttman A (1984) R-trees: A dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp 47–57

Hagedorn S, Götze P, Sattler K (2017) Big spatial data processing frameworks: Feature and performance evaluation. In: Proceedings of the 20th International Conference on Extending Database Technology (EDBT), pp 490–493

Hsu YT, Pan YC, Wei LY, Peng WC, Lee WC (2012) Key formulation schemes for spatial index in cloud data managements. In: 2012 IEEE 13th International Conference on Mobile Data Management (MDM), pp 21–26

Jagadish HV, Ooi BC, Tan KL, Yu C, Zhang R (2005) idistance: An adaptive b+-tree based indexing method for nearest neighbor search. ACM Transactions on Database Systems 30(2):364–397

Koudas N, Faloutsos C, Kamel I (1996) Declustering spatial databases on a multi-computer architecture. In: Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology (EDBT), pp 592–614

Li C, Gu Y, Qi J, Zhang R, Yu G (2015) A safe region based approach to moving KNN queries in obstructed space. Knowledge and Information Systems 45(2):417–451

Lomet D, Hong M, Nehme R, Zhang R (2008) Transaction time indexing with version compression. Proceedings of the VLDB Endowment 1(1):870–881

Nishimura S, Das S, Agrawal D, Abbadi AE (2011) Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In: Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management (MDM) - Volume 01, pp 7–16

Orenstein JA, Merrett TH (1984) A class of data structures for associative searching. In: Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS), pp 181–190

Papadopoulos A, Manolopoulos Y (2003) Parallel bulk-loading of spatial data. Parallel Computing 29(10):1419–1444

Schnitzer B, Leutenegger ST (1999) Master-client r-trees: a new parallel r-tree architecture. In: Proceedings of the 11th International Conference on Scientific and Statistical Database Management (SSDBM), pp 68–77

Ward PGD, He Z, Zhang R, Qi J (2014) Real-time continuous intersection joins over large sets of moving objects using graphic processing units. VLDB Journal 23(6):965–985

Xie D, Li F, Yao B, Li G, Zhou L, Guo M (2016) Simba: Efficient in-memory spatial analytics. In: Proceedings of the 2016 SIGMOD

International Conference on Management of Data (SIGMOD), pp 1071–1085
You S, Zhang J, Gruenwald L (2015) Large-scale spatial join query processing in cloud. In: Proceedings of the 31st IEEE International Conference on Data Engineering Workshops, pp 34–41
Yu J, Wu J, Sarwat M (2015) Geospark: a cluster computing framework for processing large-scale spatial data. In: Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL), pp 70:1–70:4
Zhang R, Ooi BC, Tan KL (2004) Making the pyramid technique robust to query types and workloads. In: Proceedings of the 20th International Conference on Data Engineering (ICDE), pp 313–324
Zhang R, Qi J, Lin D, Wang W, Wong RC (2012) A highly optimized algorithm for continuous intersection join queries over moving objects. VLDB Journal 21(4):561–586
Zhang R, Qi J, Stradling M, Huang J (2014) Towards a painless index for spatial objects. ACM Transactions on Database Systems 39(3):19:1–19:42

## Cross-References

- Query Processing – Comp Geometry
- Query Processing – Joins
- Query Processing – $k$NN