R-Grove: Growing a Family of R-trees in the Big-Data Forest

Tin Vu
Computer Science and Engineering
University of California, Riverside
tin.vu@email.ucr.edu

Ahmed Eldawy
Computer Science and Engineering
University of California, Riverside
eldawy@ucr.edu

ABSTRACT

The rapid growth of big spatial data urged the research community to develop several big spatial data systems. Regardless of their architecture, one of the fundamental requirements of all these systems is to partition the data efficiently across machines. A widely-used technique for big spatial indexing is to reuse existing search trees as-is, e.g., the R-tree family, by building a temporary tree for a sample of the input and use its leaf nodes as partition boundaries. However, we show in this paper that this approach has major limitations that make it unsuitable for the big data environment. This paper studies the use of three popular trees from the R-tree family to index big spatial data, namely, the original R-tree by Guttman, R*-tree, and RR*-tree. We show that the entire family of R-trees is not ready to grow in the big data forest due to fundamental limitations in their design. To overcome these limitations, we propose three new indexes, namely, R-Grove, R*-Grove, and RR*-Grove, which are fundamentally modified to work with big data while inheriting the main characteristics of their traditional index counterparts. With all the proposed indexes publicly available as open source, we hope that these new indexes will be adopted by the community to better serve big spatial data research.

CCS CONCEPTS

• **Information systems** → Data structures;

KEYWORDS

Spatial Partitioning, Big Data, Indexing

ACM Reference Format:
Tin Vu and Ahmed Eldawy. 2018. R-Grove: Growing a Family of R-trees in the Big-Data Forest. In *26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18), November 6–9, 2018, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3274895.3274984

1 INTRODUCTION

The recent few years witnessed a rapid growth of big spatial data collected by different applications such as satellite imagery, social media analytics, smart phones, and VGI. Traditional Spatial DBMS technology could not scale up to these petabytes of data which led to the birth of many big spatial data management systems such as SpatialHadoop [3].

A common need for all distributed big spatial data systems is spatial indexing which partitions the data across machines in a spatial-aware manner. A common method that we first introduced in SpatialHadoop [3], is the sample-based STR partitioner. This method picks a small sample of the input to determine its distribution, packs this sample using the STR packing algorithm [5], and then uses the boundaries of the leaf nodes to partition the entire data. Figure 1(a) shows an example of an STR-based index where each data partition is depicted by a rectangle. The method was later generalized to other types of indexes such as Quad-tree and Hilbert R-trees. That STR-based index was very attractive due to its simplicity and excellent load balancing which is very important for distributed applications. Its simplicity urged many other researchers to adopt it in different big spatial systems including systems for in-memory processing, spatial SQL processing, visualization, and spatial join.

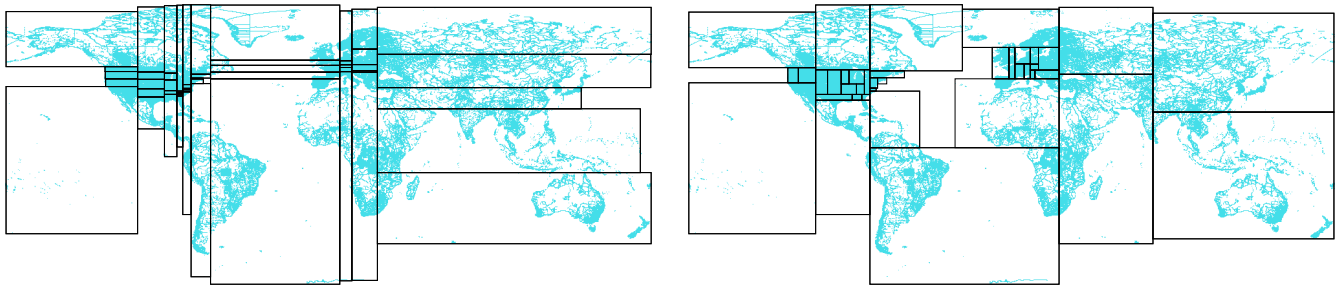
Despite its wide use, the STR-based index has major limitations in its quality as a spatial index which is apparent in Figure 1(a). In fact, anyone who is familiar with spatial indexes would immediately spot the very thin and very wide partitions which reduce the quality of the index, i.e., the query performance. Therefore, we revisit the distributed spatial indexing and study the use of three popular R-tree-based indexes to index big spatial data, namely, the original R-tree [4], the R*-tree [1], and its successor RR*-tree [2].

In this paper, we study two general approaches for adopting existing R-trees, a black-box approach, and a gray-box approach. In the black-box approach, we pick a sample of the input, load it into any of the standard R-trees, and use its leaf nodes as Minimum Bounding Rectangles (MBRs) of partitions for the big index. This approach is simple to use and has the advantage of using the existing tree with all its optimizations. However, it has two major limitations that urged us for proposing the novel gray-box implementation described shortly.

The first limitation of the black-box implementation is that all R-trees, like most traditional tree-based indexes, can produce leaf nodes with a huge variance in size which result in a load imbalance in the distributed index. While some indexing techniques, including STR, can overcome this limitation, they produce indexes of a low quality such as the one in Figure 1(a). On the other hand, the R-tree family can produce higher-quality indexes but they only guarantee that the size of each node is in the range $[m, M]$. By design, $m \leq M/2$ and m is usually set to a smaller number, e.g., $m = 0.3M$ in R*-tree and $m = 0.2M$ in RR*-tree. This configuration results in many underutilized nodes which were not bad in traditional single-machine systems; in fact, they were even desirable as they can accommodate future inserts efficiently. However, distributed indexes do not support in-place inserts due to a common limitation in distributed file systems. On the other hand, they result in a huge

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11.
https://doi.org/10.1145/3274895.3274984



(a) STR-based index [3]. All the thin and wide partitions reduce the query efficiency.

(b) The proposed R*-Grove index with square-like and balanced partitions

Figure 1: An OpenStreetMap dataset indexed with the existing STR-based index and the proposed R*-Grove index

variance in partition sizes which hurts the load balance. Therefore, near 100% utilization of partitions is important for efficiency.

The second limitation is that R-trees are constructed on a sample which does not necessarily cover the entire input space. As a result, while partitioning the actual data, there will be records that do not fall in any of the partitions and cannot be assigned. Assigning these records to any of the existing partition will cause these partitions to expand and they are likely to produce overlapping partitions. While this is not a serious issue for simple operations such as range query and kNN, some operations require disjoint partitioning for correctness such as visualization and computational geometry.

In order to overcome the limitations of the black-box implementation, this paper also proposes a gray-box method for R-tree, R*-tree, and RR*-tree that inherit their main characteristics while overcoming the above limitations; we call them R-Grove, R*-Grove, and RR*-Grove, to distinguish them from the traditional trees. Figure 1(b) gives an example of an R*-Grove index which avoids all the thin and wide partitions that degraded the performance of the previous STR-based index. In addition, the R*-Grove produces an excellent load balancing between the partitions and disjoint partitions.

The key idea of the R-Grove index is to start with one partition that contains all sample points and then use the node split algorithm in the corresponding R-tree to split it into smaller partitions. While splitting, we introduce new constraints to ensure that the utilization of leaf nodes will not fall below a prespecified threshold, e.g., 95%. In addition, instead of computing the MBR of the leaf nodes based on the sample, we keep track of the history of all node splits in an efficient and concise in-memory data structure that guarantees any record is always assigned to a partition while keeping the partitions disjoint.

Given the wide adoption of our previous STR-based index, we believe the proposed indexes will be widely used for standard spatial indexes in big spatial data systems and for distributed spatial algorithms such as spatial join, visualization, and computational geometry.

The rest of this paper is organized as follow. Section 2 gives a background about big spatial indexing then provides the black-box R-tree indexes. Section 3 describes the gray-box R-Grove indexes. Section 4 gives some preliminary results of the proposed work. Finally, Section 5 concludes the paper.

2 BLACK-BOX R-TREE INDEXING

In this paper, all the indexing algorithms rely on sampling-based indexing method, which consists of three phases: sampling, boundary computation, and physical partitioning. Phase 1 draws a random sample of the input to infer its distribution. Phase 2 uses this sample to divide the space and define partition boundaries. Phase 3 partitions the data by assigning each record to one of the partitions. More details can be found in [3]. The work in this paper focuses on Phases 2 and 3.

In this section, we propose a first-cut solution to using R-tree indexes for big data. This method customizes Phase 2 in the sampling-based indexing by using any existing R-tree index as a black box. It starts by initializing an empty R-tree (or R*-tree or RR*-tree) while setting the maximum node capacity $M = C$. The minimum node capacity m is set to $0.5M$, $0.3M$, and $0.2M$ for R-tree, R*-tree, and RR*-tree, respectively, as recommended for each index structure.

All the R-tree implementations share two common limitations. First, the leaf nodes can contain any number of points in the range $[m, M]$. For example, in RR*-tree $m = 0.2M$ which means that the ratio of the size between the largest and smallest leaf nodes can be up-to 0.2. By R-tree design, the largest allowed value for m is $\lfloor M/2 \rfloor$. If m is set to a larger value, the node split method will always fail when splitting a node of size $M + 1$. In traditional single-machine systems this is usually a desirable feature as all the partially filled nodes can accommodate future inserts without any need for splitting. However, since big-spatial indexes are static by nature, a near 100% utilization of all nodes is desirable as it ensures an optimum balance load across machines.

The second limitation is that all those indexes are data-partitioning indexes. This means that each record belongs to one leaf node which results in potential overlap between leaf nodes. This is fine for simple database operations such as range query, kNN, and spatial join. However, for some analytical jobs that run on big-data systems, disjoint space partitioning is required. For example, many computational geometry operations rely on disjoint partitions to be able to merge the partial answers in each partition. Similarly, big-spatial data visualization requires disjoint partitioning to smooth the data in each partition without worrying about overlapping regions.

3 GRAY-BOX R-TREE INDEXING

This section describes the novel gray-box method to implement the three indexes, R-tree [4], R*-tree [1], and RR*-tree [2]; for clarity, we call them R-Grove, R*-Grove, and RR*-Grove. First, all three indexes overcome the first limitation (huge disparity in partition sizes) by employing a novel improvement that allows us to freely set the minimum split size m to any value in the range $(0, M)$, even larger than $M/2$. Second, the R*-Grove and RR*-Grove employ an additional improvement that allows them to produce disjoint indexes while maintaining the high quality of the indexes. Below, we describe the two key ideas that we propose to support the three Grove indexes.

3.1 Key Idea 1: Balanced Partitions

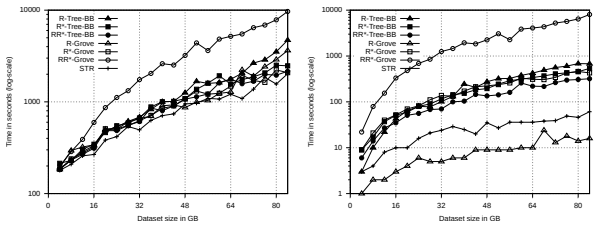
The first key idea is used to produce balanced partitions by allowing the minimum node size m to be set to any values including $m > M/2$ which is not allowed in traditional R-trees. This idea is employed in the three Grove indexes to produce balanced partitions by setting $m = 0.95M$, for example. The limitation $m < M/2$ in traditional R-trees is required only to ensure that the node splitting algorithm would succeed. For example, if $M = 10$ then a node is split when it contains 11 records. In this case, to split it into two nodes, at least one of them should have five elements or less, i.e., $m \leq 5$. To overcome this limitation, we observe that we do not have to insert records one-by-one. Rather, we can partition the space in a top-down manner where we look at all the (sample) points and partition them recursively until we produce the leaf partitions. Therefore, we add a condition while splitting the data to ensure that we eventually produce leaf partitions that are in the range $[m, M]$. For example, assume that we have 28 points and we would like to partition them into three partitions with $m = 9$ and $M = 10$. Obviously, there is a correct answer with three partitions of sizes 9, 9, and 10. However, if we apply the traditional node splitting algorithm as-is, it might start by producing two partitions each of size 14 which is considered incorrect because they cannot be further split. In this case, we say that the size $S = 28$ is valid because it can be partitioned correctly, while $S = 1$ is invalid because it cannot be further partitioned. In the same way, we can say that according to $m = 9$ and $M = 10$, the partition sizes 26 and 62 are invalid while 27 and 63 are valid. Obviously, if we keep the partition size valid as we recursively partition the sample points, we will end up with valid partitions with sizes in the range $[m, M]$. For a partition size S to be valid, it has to satisfy the inequality $\lceil S/M \rceil \leq \lfloor S/m \rfloor$. The proof is omitted for the space limitation.

To apply this idea in the three Grove indexes, we start with all the sample points in one large partition. If the sample size is invalid according to the validity test, the algorithm fails right-away as there is no valid partitioning for the size. In this case, m might need to be reduced to relax the validity condition. Otherwise, we apply the appropriate node splitting algorithm.

For R-Grove, we use the linear-time node splitting algorithm [4][1]. After the node splitting algorithm is done, we test the two partition sizes for validity. If any of them is invalid, we start moving records one-by-one from the larger partition to the smaller partition until both are valid. Since the original non-partitioned size was valid, it is guaranteed that this technique will terminate with two valid partitions.

For R*-Grove and RR*-Grove, we employ their node splitting algorithms. For both, the algorithm works by first choosing a split axis and then a split point. The split-axis algorithm is used as-is with no changes. However, the split-point algorithm is changed to produce two valid partitions. The split-point algorithm simply scans over the points along the chosen axis and gives a cost to each possible split. Finally, it chooses the split with the minimum cost. We modify this algorithm to increase the cost to ∞ if at least one of the two partition sizes is invalid. This way, the algorithm will always produce two valid partitions.

[1]We also tried the quadratic-cost algorithm but it was impractical due to the large sample size.



(a) Total indexing time

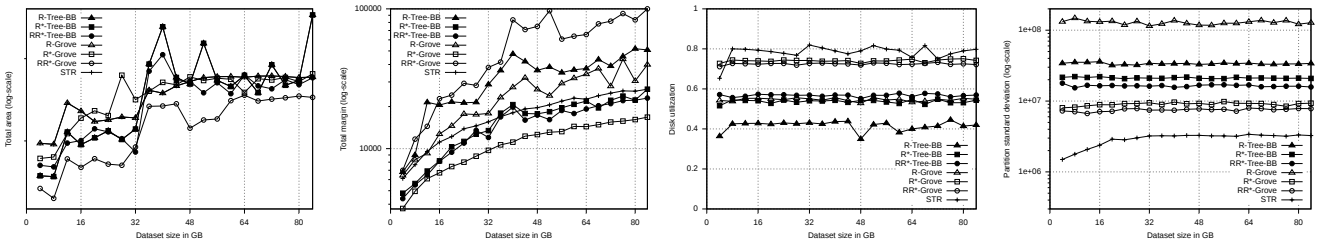
(b) Phase 2: Boundary computation time

Figure 2: Indexing time as the index size increases

3.2 Key Idea 2: Disjoint Partitions

All the three indexes that we consider in this paper are *data partitioning* indexes. This means that each record is assigned to exactly one leaf node while the boundaries of the nodes are allowed to overlap. In some big data analytics algorithms, it is required to produce disjoint partitions while replicating some records to all overlapping partitions; this is sometimes known as *space partitioning*. Unfortunately, this requirement cannot be easily satisfied because the partition boundaries are built on sample points which do not necessarily cover the entire input space. In other words, while partitioning the actual records in Phase 3, there will be some records that do not overlap any of the existing partitions but they still need to be assigned to at least one partition without producing any overlaps.

To overcome this limitation, we employ an idea that does not use partition boundaries during the partitioning phase. Rather, it keeps the history of all the split-axis and split-point algorithms employed during Phase 2. This history of splits is kept in an efficient data structure that is similar to the K-d tree index structure. While partitioning the records, if a disjoint partitioning is desired, the minimum bounding rectangle (MBR) of each record is compared to the *split data structure* to find all the overlapping partitions. Unlike the partition boundaries which do not necessarily cover the entire input space, this auxiliary search data structure is designed to always cover the entire input space. Hence, it can always be used to produce disjoint partitions.



(a) Q_1: Total area (b) Q_3: Total margin (c) Q_4: Disk utilization (d) Q_5: Standard deviation of partition size

Figure 3: Index quality as the input size increases

4 PRELIMINARY RESULTS

In this section, we carry out some preliminary results to verify the advantages of the proposed indexes. We choose the STR index as the baseline since it is widely used in many big spatial data indexing systems. Based on that baseline, we will compare the performance of the six indexes proposed in this paper, the three black-box indexes, R-tree, R*-tree, and RR*-tree, and the three gray-box indexes, R-Grove, R*-Grove, and RR*-Grove. We used a 84-GB OSM Point dataset and a 12-node cluster running on Hadoop 2.9.0.

In this experiment, we vary the input dataset size by using random samples of different sizes and measure the total time to index the entire dataset. Figure 2(a) shows the indexing time for the STR as a baseline and the six proposed R-tree implementations. Except for RR*-Grove, all indexes provide a similar indexing time as they are driven mainly by the parallel Phases 1&3 which are very similar for all of them. To further investigate the behavior of the indexing time, Figure 2(b) provides the running time for Phase 2, boundary computation, which runs on a single machine. This figure clearly highlights the difference between the various techniques. R-Grove is clearly the fastest as it builds on the linear-time R-tree splitting algorithm. The STR algorithm requires two rounds of sorting, one for each dimension. The R*-Grove and RR*-Grove indexes require multiple rounds of sorting which is clear in their performance.

Figure 3 shows the index quality in terms of Q_1: total area, Q_3: total margin, and Q_4: disk utilization, and Q_5: standard deviation of partition sizes. In Q_1, Q_3, and Q_5, the lower the value the better, while in Q_4, the higher the better. Glancing through the four figures, it is hard to identify one index that is a clear winner. For example, while the RR*-Grove is the best in Q_1, it is the worst in Q_3. There are three interesting observations, though. First, both R-tree and R-Grove indexes do not seem to be competitive which is expected given the superiority of the R*-tree and RR*-tree. Second, the black-box techniques seem to behave better than their gray-box counterparts especially in Q_3, total margin. The reason is that the black-box implementation compacts all the partitions to the contained sample points while the gray-box implementation uses the auxiliary search structure which implicitly enlarges some partitions to cover the entire space. The third observation is that the gray-box techniques provide the best disk utilization thanks to the high m/M ratio.

5 CONCLUSION

This paper addresses the problem of building R-tree-based indexes for big spatial data. We proposed two approaches, a black-box approach which deals with existing R-tree indexes as a black-box, and gray-box techniques, which utilizes the characteristics of the R-tree indexes while adopting them for big data. We employed both techniques to implement R-tree, R*-tree, and RR*-tree black-box indexes, and R-Grove, R*-Grove, and RR*-Grove, as the gray-box indexes. We highlighted two limitations in black-box indexes, namely, low disk utilization due to the inherently small m/M ratio, and the lack of support for disjoint indexes. We showed that we can overcome both limitations in the gray-box indexes in addition to improving the performance. An extensive experimental evaluation was carried out on big spatial datasets which showed that the proposed indexes can scale well to big data with the gray-box indexes overcoming the two limitations discussed above. In general, we found that, while not always the best, the R*-Grove index is pretty stable across the different experiments and we believe that it can replace the baseline STR algorithm that is currently widely used in big spatial data systems.

ACKNOWLEDGEMENTS

This work is supported in part by the National Science Foundation (NSF) under grant IIS-1838222. The authors would like to thank Norbert Beckmann for his help in clarifying the R*-tree and RR*-tree indexes.

REFERENCES

[1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *SIGMOD*. Atlantic City, NJ, 322–331. https://doi.org/10.1145/93597.98741

[2] Norbert Beckmann and Bernhard Seeger. 2009. A Revised R*-tree in Comparison with Related Index Structures. In *SIGMOD*. Providence, RI, 799–812. https://doi.org/10.1145/1559845.1559929

[3] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE*. Seoul, South Korea, 1352–1363. https://doi.org/10.1109/ICDE.2015.7113382

[4] Antonin Guttman. 1984. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*. Boston, MA, 47–57. https://doi.org/10.1145/602259.602266

[5] Scott T Leutenegger, Mario A Lopez, and Jeffrey Edgington. 1997. STR: A simple and efficient algorithm for R-tree packing. In *Data Engineering, 1997. Proceedings. 13th International Conference on*. IEEE, 497–506.