A New Technique for Verifying the Consistency of Distributed R-Trees

Sávio S. T. de Oliveira[1], José F. de S. Filho[2], Vagner J. do Sacramento Rodrigues[1], Marcelo de C. Cardoso[1], Sérgio T. Carvalho[2]

[1] goGeo
{savio.teles, vagner, marcelo.castro}@gogeo.io
[2] Federal University of Goiás, Brazil
jkairos@gmail.com, sergio@inf.ufg.br

Abstract. The ever-increasing of spatial datasets and the widely application of the complex computation have motivated the emergence of distributed algorithms to process spatial operations efficiently. The R-tree index is broadly used by researches as a distributed spatial structure for indexing and retrieval of spatial objects. However, a big challenge has arisen, that is, how to check the consistency of distributed R-Trees. In the past few years researches have been published on both distributed R-Tree and verification of distributed systems. Though none of them has proposed a technique to check the consistency of distributed R-Trees. This article presents a new approach for verifying the consistency of distributed R-Trees, which is called RConsistency. It allows collect information about the distributed R-Tree once it has been created. RConsistency also collects information about the distribute R-Tree and can helps to reduce the overlapping and dead area. It can be used with any index similar to R-Tree, since the RConsistency algorithm uses the nodes organization of the R-Tree to collect consistency information. The algorithm was used on DistGeo, a platform to process distributed spatial operations. A graphic tool, named RConsistency Visualizer, was developed to show the output of the RConsistency algorithm.

Categories and Subject Descriptors: C.2.4 [**Computer-communication networks**]: Distributed Systems—*Distributed databases*; *Distributed applications*; H.2.8 [**Database Management**]: Database Applications —*Spatial databases and GIS*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Distributed debugging*; D.2.4 [**Software Engineering**]: Software/Program Verification

Keywords: Distributed Algorithm, Distributed Indexes, R-Tree, Verification

1. INTRODUCTION

The increasing of large spatial datasets demands high performance engine in order to process complex spatial models. The best cost-benefit to provide innovative GIS[1] applications that take advantage of all available data is through distributed and parallel GIS processing. However, the development of a high performance engine to distributed spatial computing is very complex and challenging.

In order to handle spatial data efficiently, a database system needs an index mechanism that helps it retrieve data items quickly according to spatial objects location. The R-Tree typically is the preferred method for indexing spatial data. Many researches, such as An et al. [1999], de Oliveira et al. [2011] and Zhong et al. [2012], show that a distributed index structure can provide an efficient mechanism to processing spatial operations. However, distributed R-Trees indexes for Big Spatial Data are very complex to be developed and so it demands novel approaches to check consistency.

[1]Geographical Information Systems

Debugging and verifying is an essential step in the development process, though often neglected in the development of distributed applications due to the fact that distributed systems complicate the already difficult task of debugging and verifying [H.Cheung et al. 1990]. In recent years, researchers have developed some helpful techniques for debugging and verifying distributed systems. Nevertheless, we have not found in the literature any work that have addressed the problem of verifying the consistency of distributed R-Trees.

In this article, we propose a new technique for verifying the consistency of distributed R-Trees. The technique, called RConsistency, uses the distributed index structure to aggregate information about distributed R-Tree consistency. RConsistency can be used with any index similar to R-Tree, since the RConsistency algorithm uses the nodes organization of the R-Tree to collect consistency information. RConsistency also collects information about the distribute R-Tree that can helps to improve the quality of the R-Tree. We have also created a graphical tool to visualize the information about R-Tree index structure, called RConsistency Visualizer.

The main contributions of this article are as follows:

—RConsistency - A new technique for verifying the consistency of distributed R-Trees.
—DistGeo - A peer-to-peer platform, with no single point of failure, to process distributed spatial algorithms of an R-Tree.
—RConsistency Visualizer - A graphical tool to visualize information about the distributed R-Tree index.

The rest of the article is structured as follows. In Section 2, we briefly give an overview of the use of verifying approaches for distributed systems and the view of distributed spatial algorithms. Section 3 describes the distributed R-Tree implementation on DistGeo platform. Section 4 presents our approach for verifying the consistency of distributed R-Trees. Section 5 presents the evaluation of RConsistency algorithm in the DistGeo platform. Finally, we close the article with some concluding remarks in Section 6.

2. RELATED WORK

While distributed algorithms is a highly active area, there have been few efforts to achieve consistency verification of distributed algorithms, especially on distributed spatial algorithms. Researches on distributed spatial data either show techniques to verify the consistency of distributed applications in general or techniques for R-tree distributed processing, but none addressed both issues. The Section 2.1 shows researches about distributed consistency verifying and 2.2 describes researches about distributed spatial algorithms.

2.1 Techniques for Verifying System Consistency

Designing and verifying the correctness of reliable distributed systems is a big challenge. Several toolkits, such as SMV [McMillan 1999], Mocha [de Alfaro et al. 2000], AsmL [Gurevich et al. 2001], TLC [Lamport and Yu 2001] and DiVinE [Barnat et al. 2006], support verification and execution of concurrent and distributed systems. The execution is used mainly for understanding the behavior of a system.

In recent years, extensive research has been conducted in parallel and distributed model-checking. In this regard, some works [Lamport 2011; Muller 1998; Win et al. 2004; Hendriks 2005; Konnov et al. 2012; Tsuchiya and Schiper 2011] have addressed algorithms for model checking of distributed algorithms and verifying the correctness of the distributed systems properties. Parameterized model checking of ring-based systems has been studied in [Emerson and Namjoshi 2003; Aminof et al. 2014;

Aiswarya et al. 2015]. The paper [Bollig et al. 2012] pursued a symbolic model-checking approach to systems involving data.

System Level Formal Verification (SLFV) was used by Zuliani et al. [2013], Cavaliere et al. [2011], Mancini et al. [2013] and Mancini et al. [2014] to show system correctness notwithstanding uncontrollable events. Considerable effort has been devoted to the verification of fault-tolerant algorithms, which have to cope with faults such as lost or corrupted messages (e.g., [Chaouch-Saad et al. 2009; Konnov et al. 2014]).

There are several efforts to address the hard problem of Verification the B trees consistency. A mechanized verification was proposed in [Malecha et al. 2010] and an automatic verification in [Ernst and andWolfgang Reif 2015; Chang and Rival 2008; Distefano and Parkinson 2008]. Herter [2008] verifies some properties and Loginov et al. [2006] developed a shape analysis specification of B trees.

2.2 Spatial Algorithms

There are researches that present the use of parallelism in order to improve the response time of the spatial algorithms. M-RTree [Koudas et al. 1996] was the first published article, which shows a shared-nothing architecture, with a master and several workstations connected to a LAN network. A similar technique was found on MC-RTree [Schnitzer and Leutenegger 1999] and [An et al. 1999].

Hadoop-GIS [Kerr 2009] shows a scalable and high performance spatial data warehousing system for running large scale spatial queries on Hadoop. A platform to process distributed spatial operations is presented in [de Oliveira et al. 2011]. The work in [de Oliveira et al. 2013] shows a hybrid peer-to-peer platform, which comprehends a set of machines for naming resolution.

In [Xie et al. 2008], a two-phase load-balancing scheme is introduced for the parallel GIS operations in distributed environment. MapReduce is described in [Zhang et al. 2009], which shows how spatial queries can be naturally expressed in this model.

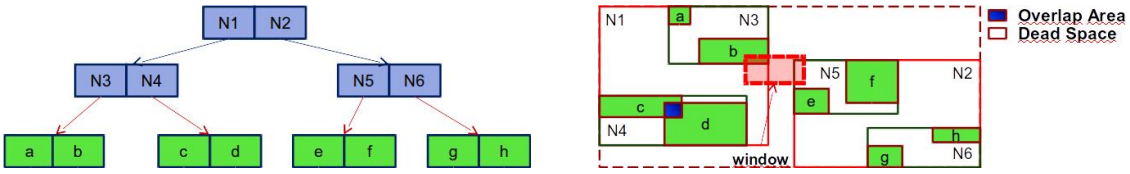
Some techniques and platforms have been proposed for handling distributed spatial data. Nevertheless, none of the researches propose a technique for verifying the consistency of distributed R-Trees. There are some tools to visualize different spatial indexes and to animate the execution of some common spatial operations on them. An applet was proposed in [Brabec and Samet 1998; 1999] to enable users to visualize some hierarchical spatial data structures, like R-tree, Quadtree and Kd-tree, on the worldwide web. R-Tree visualization for high dimensional data is addressed by Giménez et al. [2010]. Heid [Agrawal et al. 2011] allows visualize high dimensional data clusters grouped and ordered according to R-tree structure. However, these tools do not provide consistency information about distributed R-Tree, but could be integrated with a solution of gathering consistency information.

3. ALGORITHMS FOR SPATIAL DISTRIBUTED PROCESSING

A number of structures have been proposed for handling multi-dimensional spatial data, such as KD-Tree [Bentley 1975], R-Tree [Guttman 1984] and its variants [Kamel and Faloutsos 1994]. The R-Tree has been widely used to index the datasets on GIS databases and it has been used as an index data structure in this work.

An R-Tree is a height-balanced tree similar to a B-Tree [Comer 1979] with index records in its leaf nodes containing pointers to data objects. The key idea of the data structure is to group nearby objects and represent them with their minimum bounding rectangle (MBR) in the next higher level of the tree.

Figure 1(a) illustrates the hierarchical structure of an R-Tree with a root node, internal nodes ($N1...2 \subset N3...6$) and leaves ($N3...6 \subset a...h$). Every internal node contains a set of rectangles and pointers to the corresponding child node and every leaf node contains the rectangles of spatial objects.



(a) R-Tree index (b) Geographic space

Fig. 1. R-Tree Structure

Figure 1(b) shows MBRs grouping spatial objects of $a...h$ in sets by their co-location. Each node stores at most M and at least $m \leq M/2$ entries [Guttman 1984].

The Window Query is one of major query algorithms in R-Tree. The search starts from the root node of the tree and the input is a search rectangle (Query box). For each rectangle in a node, it has to be decided whether it overlaps the search rectangle or not. If so, the corresponding child node has to be searched too.

Searching is done recursively until all overlapping nodes have been traversed. When a leaf node is reached, the contained bounding boxes (rectangles) are tested against the search rectangle and the objects that intersects with the search rectangle are returned.

For the window query on Figure 1(b), the search starts on root node (Figure 1(a)) as the window intersects with nodes $N1$ and $N2$. Then, the algorithm analyses node $N1$, which only $N3$ intersects with the window. Analyzing node $N3$, the algorithm returns the spatial object namely b, that is the single object that intersects the window.

In node $N2$, we do not have any entry intersecting with the window due to the dead space. In other words, the window intersects with a space, which does not contain any data. The dead space should be minimized to improve the query performance, since decisions about which paths have to be traversed can be taken on higher levels.

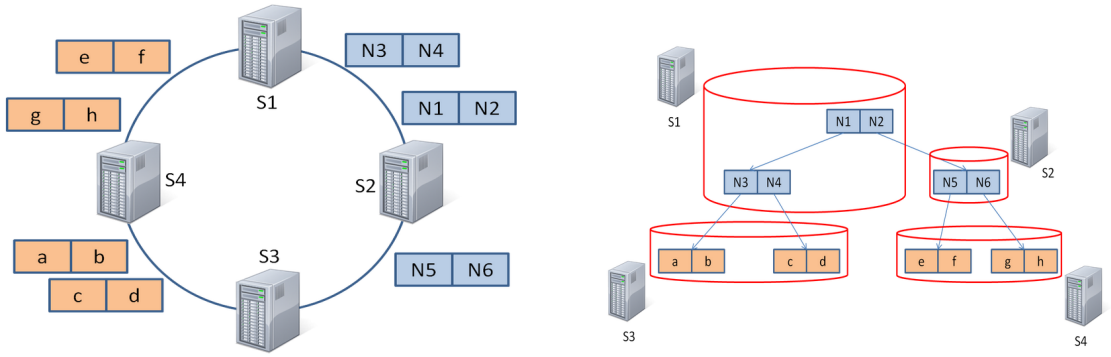
The overlapping area between rectangles should be minimized as well, as it degrades the performance of R-Tree [Beckmann et al. 1990]. Less overlapping reduces the amount of sub-trees accessed during R-tree traversal. The area between c and d in Figure 1 is an example of overlapping.

3.1 DistGeo: A Platform of Distributed Spatial Operations for Geoprocessing

DistGeo is a platform to process spatial operations in a cluster of computers (Figure 2). It is based on a shared-nothing architecture, which the nodes do not share CPU, hard disk and memory and the communication relies on message exchange. Figure 2(a) show DistGeo Architecture, which is based on peer-to-peer model presented as a ring topology. It is divided in ranges of keys, which are managed for each server of the cluster. In order to a server to join the ring it must be assigned a range first.

The range of keys are known by each server in the cluster using a Distributed Hash Table (DHT) to store the mapping of the keys to servers. For instance, in a ring representation whose keys range from 0 to 100, if we have 4 nodes in the cluster, the division could be done as shown below: a) 0-25, b) 25-50, c) 50-75 e d) 75-100. If we want to search for one object with key 34, we certainly should look on the server 2.

Since there is not a master replica, every replica of an object is equally important. Therefore, read and write operations may be performed in any server of the cluster. When a request is made to a cluster's server, it becomes the coordinator of the operation requested by the client. The coordinator works as a proxy between the client and the cluster servers. DistGeo uses the Gossip protocol [Demers et al. 1988], which every cluster server exchanges information among themselves for everyone to know the status of each server.



(a) DistGeo Architecture (b) R-Tree Partitioning in DistGeo

Fig. 2. DistGeo Platform

Figure 2(b) illustrates the structure of a Distributed R-Tree in a cluster. The partitioning is performed by creating the indexes according to the R-Tree structure. The lines in Figure 2(b) show the need for message exchange to reach the sub-trees during the algorithm processing.

Insertions and searching in a distributed R-Tree are similar to the non-distributed version, except for: i) The need of message exchange to access the distributed partitions and ii) Concurrency control and consistency due to the parallel processing in the cluster. Both were implemented on DistGeo platform.

The distributed index has been built according to the taxonomy defined in [An et al. 1999], as follows: i) Allocation Unit: block - A partition is created for every R-Tree node; ii) Allocation Frequency: overflow - In the insertion process, new partitions are created when a node in the tree needs to split; iii) Distribution Policy: balanced - To keep the tree balanced the partitions are distributed among the cluster servers.

Reliability and fault-tolerance were implemented on DistGeo storing the R-Tree nodes in multiple servers in the cluster. The DistGeo uses Apache Cassandra [2] database to store the distributed R-Tree index nodes on cluster servers. Each R-Tree node N receives a key, which is used to store the node in a server S responsible for ring range, replicating the node N to the next two servers in S (clockwise). If a message is sent to N, one of the servers that store a replica of N is selected. The query requests are always sent to one of the cluster's server that stores the root node of the R-tree.

As discussed earlier in this Section, reducing the overlapping and dead area on R-Tree minimizes the number of R-Tree nodes accessed during the tree traversal on search algorithms. The growth of the number of nodes accessed increases the network traffic because the R-Tree nodes are stored in several servers on cluster, as shown in Figure 2(b). Our article implements a new algorithm that collects information about a distribute R-Tree and can helps to reduce the overlapping and dead area. We cover this algorithm in more details in Section 4.

4. RCONSISTENCY: TECHNIQUE FOR VERIFYING THE DISTRIBUTED R-TREE CONSISTENCY

In a distributed environment, it is hard to find bugs on insertion algorithms due to the difficult to synchronize the insertion, since it must be done concurrently. Even in cases where the implementation is correct, it is not easy to improve the insertion algorithm's performance (for example, reducing the

[2]http://cassandra.apache.org

overlapping) due to the complexity of collecting information about the spatial index. In other words, it is not a trivial task to ensure that a distributed spatial index has being built accordingly.

This section describes RConsistency, a new technique for verifying the consistency of distributed R-Trees and its variants. The following information are collected by RConsistency algorithm: i) whether each replica of a R-Tree node is consistent; ii) whether the MBR of each parent node intersects with the MBR of their children, iii) whether there are duplicated nodes on R-Tree or nodes being referenced by more than one parent node, and iv) whether the value M and m of the nodes are compliant with the R-Tree descriptions as shown in Section 3. Furthermore, it is possible to access index data to help in optimization and minimizing the dead space and overlapping area.

Algorithm 1 shows the RConsistency technique for collecting consistency information about the distributed spatial index, using the index structure itself. The algorithm has two steps: 1) S1 (lines 1-11): The algorithm processing is similar to the search in an R-Tree with a top-down traversal; 2) S2 (lines 12-39): The algorithm does a bottom-up traversal on R-Tree aggregating the result.

The RConsistency algorithm is based on R-Tree structure, which is used to index the spatial datasets on DistGeo platform, presented in Sub-section 3.1. RConsistency can be applied to any R-Tree variant, such as the dynamic Hilbert R-Tree [Kamel and Faloutsos 1994], since the RConsistency algorithm uses the nodes organization of the R-Tree to collect the information. RConsistency has been implemented on DistGeo platform, so it can be processed without bottlenecks and point of failures. Besides, the R-Tree replicated nodes in the cluster allow load-balancing in the distributed R-Tree index traversal. The traversal might go to servers with less workload.

In the first step, called S1 [Search sub-trees] (lines 1 - 11), the Algorithm 1 go to every node of the R-Tree starting from the root node to the leaves. The first request is sent to any server which stores a replica of the root node. If node T is not a leaf (lines 2 - 8), the Step 1 is called recursively to children entries E in T, sending message to one server that holds a replica of E. In addition, the number of children entries in T is stored to control the number of expected answers in the second step of the algorithm (line 3).

If T is a leaf (line 10), the second step, named S2 [Aggregation] is started (lines 12 - 39). Second step aims (lines $12-39$) to aggregate the information about the index to be used for future consistency verification. The index itself is used to aggregate this information using the cluster computational resources to improve the algorithm's performance. Each node of the R-Tree is responsible to aggregate only the information of its children. The consistency information about each node T of R-Tree is stored in a shared memory that can be accessed by any server that stores a replica of T.

In line 13, the information is retrieved from the shared memory. Line 14 verifies the consistency of the replicas of T and in the line 15 is verified the M and m values. Lines 16 and 17, in turn, calculate the overlap and the dead space area of T and line 18 gets the MBR of T. All these information are inserted in $information$ on line 19.

When in an internal node (lines 26 - 39), the algorithm aggregates the information of the children nodes. In line 29, the algorithm receives the information sent by the child node. Line 27 verifies if the MBR of the entry that points to the child node is indeed the same MBR sent by the child node.

Line 28 adds the data processed from lines 26 and 27 in $information$. Line 29 retrieves the number of child nodes, which did not send a response. This number is stored in the variable *count*, which is decremented and updated on shared memory.

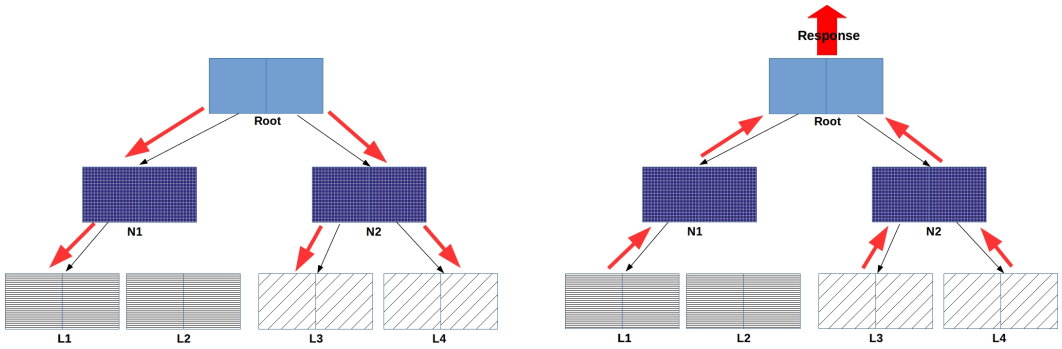
If every node has sent the answer, the variable *count* then will hold the value 0 and lines 30-35 are processed. If T is the root node, then the information is sent to the client application, otherwise, all information collected is sent to the parent node of T. If the variable *count* is greater than 0, then the client information is stored in the shared memory to be used until each reply is received by child nodes.

Algorithm 1: *RConsistency*(T)

Data: T reference of the root node of R-Tree *tree*
Result: Consistency information about distributed R-Tree *tree*

```
1  S1 [Search subtrees]
2  if T is not leaf then
3  |  stores the number of children entries in each replica server of T
4  |  for each entry E in T do
5  |  |  server ⇐ choose one server, randomly, that keep one replica of E
6  |  |  send msg to server to process the node's child of E on step S1
7  |  end
8  else
9  |  verify the consistency of T in other replicas
10 |  Invoke step S2 [Aggregation]
11 end
12 S2 [Aggregation]
13 information ⇐ the child's information stored on shared memory by replicas of T
14 replica_consistency ⇐ verify the consistency of T in others replicas
15 node_consistency ⇐ verify the consistency of M and m values of T
16 overlap ⇐ overlap area of T
17 dead_area ⇐ dead area of T
18 bound ⇐ MBR of T
19 add in information: replica_consistency, node_consistency, overlap, dead_area, bound
20 if T is leaf then
21 |  if T is root then
22 |  |  send response with R-Tree nodes information to app client
23 |  end
24 |  send msg with information to parent of T
25 else
26 |  entry_info ⇐ information sent by child node
27 |  mbr_consistent ⇐ verify if the bound of the child node is equal to bound of entry of T that
   |  points to this child
28 |  add in information: entries_info, and mbr_consistent
29 |  count ⇐ retrieve the number of child entries, which didn't send response and decrement by 1
30 |  if count == 0 then
31 |  |  if T is root then
32 |  |  |  send response with information to client
33 |  |  else
34 |  |  |  send msg with information to parent of T
35 |  |  end
36 |  else
37 |  |  store information on shared memory
38 |  end
39 end
```

Figure 3 shows an example of the steps S1 and S2 of RConsistency algorithm. In step S1 (Figure 3(a)), the algorithm starts at root node and traverses to internal nodes $N1$ and $N2$. Starting in $N2$, the algorithm continues to leaf nodes $L3$ and $L4$. Node $N1$ should be linked to children $L1$ and $L2$. However, during insertion algorithm $N1$ dropped the reference to $L2$ due to an implementation error. The RConsistency algorithm identify this cases returning fewer objects than those inserted.



(a) Step 1 - RConsistency algorithm (b) Step 2 - RConsistency algorithm

Fig. 3. Example of RConsistency algorithm steps

When the algorithm reaches the leaves, step S2 starts (Figure 3(b)). Information about consistency and quality of R-Tree index, described above, are collected and sent to parents nodes. Parents nodes are responsible for gathering information about children. The *root* node is responsible for collecting consistency and quality information of entire tree and sending the answer.

Using the RConsistency algorithm is possible to collect information about searching algorithms in a single R-Tree, for example, the Window Query algorithm shown in Section 3. Whereas, algorithms that access many R-Trees, such as Spatial Join, need a deep change, once algorithms can go through different paths.

The algorithm RConsistency have collected consistency information about the R-Tree index built during the insertion of the dataset. Figure 4 shows a graphical tool (RConsistency Visualizer) created to visualize the collected consistency information. RConsistency Visualizer shows the structure of the distributed R-Tree index and allows the analysis of each node of the R-Tree. The output of the RConsistency algorithm shows which nodes are currently inconsistent. The user can access the path of the node and visualize the node's inconsistent information.

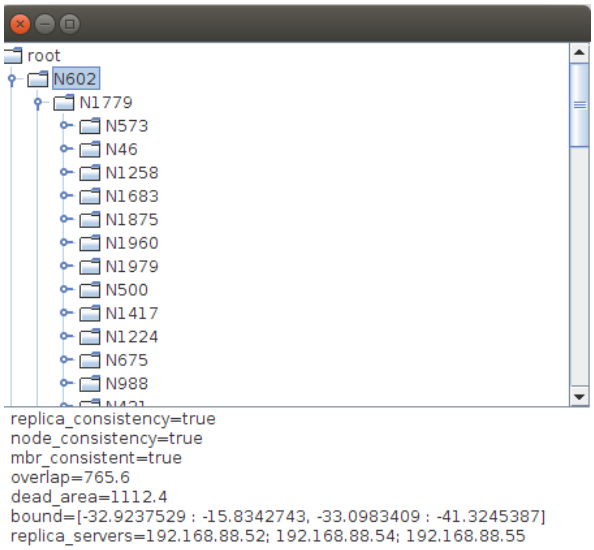
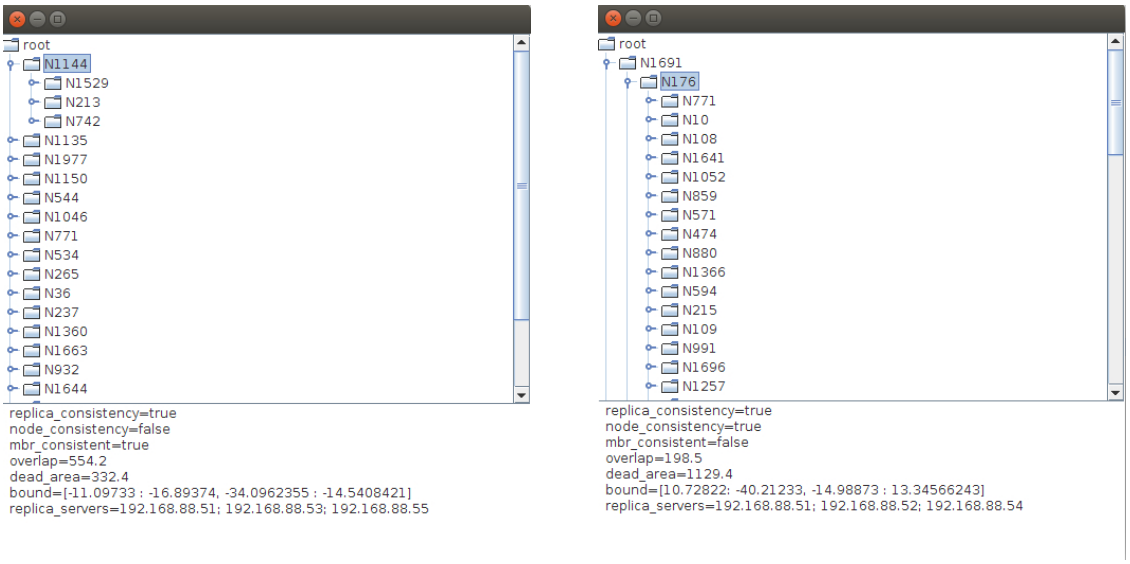


Fig. 4. RConsistency Visualizer



(a) Node inconsistency (b) Bound inconsistency

Fig. 5. RConsistency algorithm on business listings dataset

5. EVALUATION

The RConsistency algorithm has been evaluated on 3500 MHz Intel(R) Core(TM) i7-2600 CPU workstations connected by 1 GBit/sec switched Ethernet running Ubuntu 14.04. Each node has 16 GB of main memory. The experiment results were achieved with 1, 2, 4 and 8 servers on DistGeo platform.

The experiments were performed using three datasets with different characteristics. The first contains 1000000 points of business listings and points of interest (POIs) from SimpleGeo[3]. The second dataset comprises 226964 lines representing the rivers on Brazil from LAPIG[4]. The third contains 220000 polygons of the census of USA from TIGER/Line[5].

The RConsistency was executed on DistGeo platform after the indexing of each dataset. The algorithm was able to collect information about the R-Tree index, such as dead space and overlapping area. Furthermore, RConsistency algorithm has succeeded to collect the index structure allowing to visualize each data set R-Tree index on RConsistency Visualizer tool.

Three inconsistencies were deliberately inserted in the index to evaluate the RConsistency: i) inconsistencies between parent and child nodes bounding, ii) nodes filled with more than M entries and iii) duplication of a node on R-Tree. The RConsistency algorithm was able to identify this inconsistencies in every distributed R-Tree related to datasets.

Figure 5 shows the result of RConsistency algorithm with the business listings dataset in RConsistency Visualizer tool. An example of node inconsistency is shown in Figure 5(a), which the R-Tree node $N1144$ contains only three entries. This number of entries violates the m value presented in Section 3. Figure 5(b) shows the bound inconsistency between node $N176$ and one of its children. The duplicated nodes identified on R-Tree are shown on final report by RConsistency algorithm. The user can traverse the R-Tree path on RConsistency Visualizer to identify these duplicated nodes.

[3]https://github.com/simplegeo
[4]www.lapig.iesa.ufg.br
[5]Census 2007 Tiger/Line data

6. CONCLUSION

DistGeo platform presents an approach for processing distributed spatial operations through the distributed R-Tree index. Due to the distributed processing nature on this platform an issue arises: verifying the distributed R-Tree consistency once it has been created.

We have seen researches on spatial data processing and distributed consistency verifying, but none of them propose techniques for collect consistency information of distributed R-Trees. Our article presented the RConsistency algorithm for verifying the consistency of distributed R-Trees. RConsistency uses the R-Tree index itself to gather the consistency information.

The data gathering is achieved in a distributed way, improving the algorithm efficiency. DistGeo, a new peer-to-peer platform, was proposed in our work and has been used to execute the RConsistency algorithm. Since the R-Tree nodes are distributed and replicated over the cluster, RConsistency can be processed without bottlenecks and point of failures.

A graphical tool(RConsistency Visualizer) has been created to visualize the structure of the distributed R-Tree index and the consistency information of R-Trees. Using this information, we can identify discrepancies in the index building. RConsistency also collects information about the distribute R-Tree and can helps to reduce the overlapping and dead area. The RConsistency algorithm can be used to collect consistency information in any index with spatial nodes organization similar to R-Tree (e.g. Hilbert R-Tree [Kamel and Faloutsos 1994]).

Ongoing work includes modify the RConsistency algorithm to collect information about Window Query and Join Query searching algorithms. The RConsistency algorithm can be easily adapted to gather information for Window Query. Whereas, for Join Query algorithm, RConsistency must be changed considerably, since the traversal is processed in two different distributed R-Trees. Another ongoing work is to simulate node replica inconsistencies to evaluate the ability of RConsistency to identify these inconsistencies. On future works, the algorithm RConsistency will be evaluated in larger clusters and performance results will be collected.

REFERENCES

Agrawal, S., Vadapalli, S., and Karlapalem, K. Heidi Visualization of R-tree Structures over High Dimensional Data. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*. Portland, OR, USA, pp. 565–567, 2011.

Aiswarya, C., Bollig, B., and Gastin, P. An Automata-Theoretic Approach to the Verification of Distributed Algorithms. In *The 10th International Federated Conference on Distributed Computing Techniques, DiCoTec*. Grenoble, France, pp. 1–14, 2015.

Aminof, B., Jacobs, S., Khalimov, A., and Rubin, S. Parameterized Model Checking of Token-Passing Systems. In *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation*. San Diego, CA, USA, pp. 262–281, 2014.

An, N., Lu, R., Qian, L., Sivasubramaniam, A., and Keefe, T. Storing Spatial Data on a Network of Workstations. *Cluster Computing* 2 (4): 259–270, 1999.

Barnat, J., andIvana Černá, L. B., Moravec, P., Ročkai, P., and Šimeček, P. DiVinE: a tool for distributed verification. In *Proceedings of the 18th International Conference on Computer Aided Verification*. Seattle, WA, USA, pp. 278–281, 2006.

Beckmann, N., Kriegel, H.-P., Schneider, R., and Seeger, B. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record* 19 (2): 322–331, 1990.

Bentley, J. L. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM* 18 (9): 509–517, 1975.

Bollig, B., Cyriac, A., Gasti, P., and Kumar, N. K. Model Checking Languages of Data Words. In *Proceedings of the 15th International Conference on Foundations of Software Science and Computational Structures*. Tallinn, Estonia, pp. 391–405, 2012.

Brabec, F. and Samet, H. Visualizing and Animating R-trees and Spatial Operations in Spatial Databases on the Worldwide Web. In *Proceedings of the Fourth Working Conference on Visual Database Systems 4 (VDB4)*. LâĂŹAquila, Italy, pp. 123–140, 1998.

Brabec, F. and Samet, H. Visualizing and Animating Search Operations on Quadtrees on the Worldwide Web. http://www.cs.umd.edu/~hjs/pubs/brabeceurocg00.pdf, 1999.

Cavaliere, F., Mari, F., Melatti, I., Minei, G., Salvo, I., Tronci, E., Verzino, G., and Yushtein, Y. Model Checking Satellite Operational Procedures. In *Proceeding of the International Conference on Space Engineering. DASIA2011*. Malta, pp. 1–8, 2011.

Chang, B.-Y. E. and Rival, X. Relational Inductive Shape Analysis. *ACM SIGPLAN Notices* 43 (1): 247–260, 2008.

Chaouch-Saad, M., Charron-Bost, B., and Merz, S. A Reduction Theorem for the Verification of Round-based Distributed Algorithms. In *Proceedings of the 3rd International Workshop on Reachability Problems*. Springer, Palaiseau, France, pp. 93–106, 2009.

Comer, D. Ubiquitous B-tree. *ACM Computing Surveys* 11 (2): 121–137, 1979.

de Alfaro, L., Alur, R., Grosu, R., Henzinger, T. A., Kang, M., Majumdar, R., Mang, F. Y. C., Meyer-Kirsch, C., and Wang, B. Mocha: exploiting modularity in model checking. http://embedded.eecs.berkeley.edu/Respep/Research/mocha/doc/j-doc/mocha-tool.ps.gz, 2000.

de Oliveira, S. S., Vagner, J., Cunha, A. R., Aleixo, E. L., de Thiago B. Oliveira, de C. Cardoso, M., and Junior, R. R. Processamento Distribuído de Operações de Junção Espacial com Bases de Dados Dinâmicas para Análise de Informações Geográficas. In *Proceedings of the XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Brasilia, DF, Brazil, pp. 1009–1022, 2013.

de Oliveira, T. B., do Sacramento Rodrigues, V. J., de Oliveira, S. S. T., de Albuquerque Lima, P. I., and de C. Cardoso, M. DSI-Rtree - Um Índice R-Tree Escalável Distribuído. In *Proceedings of the XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Campo Grande, MS, Brazil, pp. 719–732, 2011.

Demers, A., Greene, D., Houser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. Epidemic Algorithms for Replicated Database Maintenance. *ACM SIGOPS Operating Systems Review* 22 (1): 8–32, 1988.

Distefano, D. and Parkinson, M. J. jStar: towards practical verification for Java. *ACM Sigplan Notices* 43 (10): 213–226, 2008.

Emerson, A. E. and Namjoshi, K. S. On Reasoning About Rings. *International Journal of Foundations of Computer Science* 14 (04): 527–549, 2003.

Ernst, G. and andWolfgang Reif, G. S. Verification of B^+ trees by Tntegration of Shape Analysis and Interactive Theorem Proving. *Software and Systems Modeling* 14 (1): 27–44, 2015.

Giménez, A., Rosenbaum, R., Hlawitschka, M., and Hamann, B. Using R-trees for Interactive Visualization of Large Multidimensional Datasets. In *Proceedings of the 6th International Symposium on Advances in Visual Computing*. Las Vegas, NV, USA, pp. 554–563, 2010.

Gurevich, Y., Schulte, W., and Veanes, M. Toward Industrial Strength Abstract State Machines. Tech. rep., Technical Report MSR-TR-2001-98, Microsoft Research, 2001.

Guttman, A. R-trees: a dynamic index structure for spatial searching. *ACM SIGMOD Record* 14 (2): 47–57, 1984.

H.Cheung, W., Black, J. P., and Manning, E. A Framework for Distributed Debugging. *IEEE Software* 7 (1): 106–115, 1990.

Hendriks, M. Model checking the time to reach agreement. In *Proceedings of the Third International Conference on Formal Modeling and Analysis of Timed Systems*. Uppsala, Sweden, pp. 98–111, 2005.

Herter, J. *Towards Shape Analysis of B-trees*. M.S. thesis, Saarland University, 2008.

Kamel, I. and Faloutsos, C. Hilbert R-tree: an Improved R-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases*. VLDB '94. San Francisco, CA, USA, pp. 500–509, 1994.

Kerr, N. T. *Alternative Approaches to Parallel GIS Processing*. M.S. thesis, Arizona State University, 2009.

Konnov, I., Veith, H., and Widder, J. Who is Afraid of Model Checking Distributed Algorithms. In *Proceedings of the CAV'12 Workshop (EC)2*. Berkeley, USA, 2012.

Konnov, I., Veith, H., and Widder, J. On the Completeness of Bounded Model Checking for Threshold-based Distributed Algorithms: reachability. In *Proceedings of the 25th International Conference on CONCUR 2014–Concurrency Theory*. Rome, Italy, pp. 125–140, 2014.

Koudas, N., Faloutsos, C., and Kamel, I. Declustering Spatial Databases on a Multi-computer Architecture. In *Proceedings of the 5th International Conference on Extending Database Technology*. Avignon, France, pp. 592–614, 1996.

Lamport, L. Byzantizing Paxos by Refinement. In *Proceedings of the 25th International Symposium on Distributed Computing*. Rome, Italy, pp. 211–224, 2011.

Lamport, L. and Yu, Y. TLC: the TLA+ model checker. http://research.microsoft.com/en-us/um/people/lamport/tla/tlc.html, 2001.

Loginov, A., Reps, T., and Sagiv, M. Automated Verification of the Deutsch-Schorr-Waite tree-traversal Algorithm. In *Proceedings of the 13th International Symposium on Static Analysis*. Seoul, Korea, pp. 261–279, 2006.

Malecha, G., Morrisett, G., Shinnar, A., and Wisnesky, R. Toward a Verified Relational Database Management System. *ACM Sigplan Notices* 45 (1): 237–248, 2010.

Mancini, T., Mari, F., Massini, A., Melatti, I., and Tronci, E. System level formal verification via model checking driven simulation. In *Proceedings of the 25th International Conference on Computer Aided Verification*. Saint Petersburg, Russia, pp. 296–312, 2013.

Mancini, T., Mari, F., Massini, A., Melatti, I., and Tronci, E. System Level Formal Verification via Distributed Multi-core Hardware in the Loop Simulation. In *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. Torino, pp. 734–742, 2014.

McMillan, K. L. The SMV Language. http://www.kenmcmil.com/language.ps, 1999.

Muller, O. I/O Automata and Beyond: temporal logic and abstraction in Isabelle. In *Proceedings of the 11th International Conference on Theorem Proving in Higher Order Logics*. Canberra, Australia, pp. 331–348, 1998.

Schnitzer, B. and Leutenegger, S. T. Master-client R-trees: a new parallel R-tree architecture. In *Proceedings of the 11th International Conference on Scientific and Statistical Database Management*. Cleveland, Ohio, USA, pp. 68–77, 1999.

Tsuchiya, T. and Schiper, A. Verification of Consensus Algorithms Using Satisfiability Solving. *Distributed Computing* 23 (5-6): 341–358, 2011.

Win, T. N., Ernst, M. D., Garland, S. J., Kirli, D., and Lynch, N. A. Using Simulated Execution in Verifying Distributed Algorithms. *International Journal on Software Tools for Technology Transfer* 6 (1): 67–76, 2004.

Xie, Z., Ye, Z., and Wu, L. A Two-Phase Load-Balancing Framework of Parallel GIS Operations. In *Proceedings of the IEEE International Conference on Geoscience and Remote Sensing Symposium. IGARSS*. Boston, Massachusetts, USA, pp. 1286–1289, 2008.

Zhang, S., Han, J., Liu, Z., Wang, K., and Feng, S. Spatial Queries Evaluation with Mapreduce. In *Proceedings of the 8th International Conference on Grid and Cooperative Computing. GCC'09*. Lanzhou, China, pp. 287–292, 2009.

Zhong, Y., Han, J., Zhang, T., Li, Z., Fang, J., and Chen, G. Towards Parallel Spatial Query Processing for Big Spatial Data. In *Proceedings of the IEEE 26th International on Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW)*. Hyderabad, India, pp. 2085–2094, 2012.

Zuliani, P., Platzer, A., and Clarke, E. M. Bayesian Statistical Model Checking with Application to Simulink/Stateflow Verification. *Formal Methods in System Design* 43 (2): 338–367, 2013.