RDebug: A New Debugging Technique for Distributed R-Trees

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

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Alameda Palmeiras, Quadra D, Câmpus Samambaia
131 - CEP 74001-970 – Goiânia – GO – Brazil

[2]GoGeo
Rua Leopoldo Bulhões, esquina com a Rua 1014
Quadra 31, Lote 07, Sala 9 Setor Pedro Ludovico
CEP 74820-270 – Goiânia – GO – Brazil

savio.teles@gogeo.io, jkairos@gmail.com, vagner@gogeo.io,
marcelo.cardoso@gogeo.io, sergio@inf.ufg.br

Abstract. *The high data availability and the increasing number of GIS users have motivated the emergence of distributed algorithms to process spatial operations efficiently. They are based on distributed indexes for an high performance processing. Researches and ongoing work use R-trees as a distributed spatial structure for indexing and retrieval of geo objects. However, these indexes have shown a challenge, that is, how to debug an index which is based on an R-Tree? In the past few years researches have been published on both distributed algorithms and distributed processing. Though none of them has proposed a debugging technique to a distributed R-Tree index. This paper presents a new algorithm for debugging a distributed index based on R-Tree which is called RDebug. This algorithm was used on DistGeo, a platform to process distributed spatial operations. A graphic tool, named RDebug Visualizer, was developed to show the output of the RDebug algorithm.*

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 applications taking advantage of all available data is through distributed and parallel GIS processing. But develop 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 will help it retrieve data items quickly according to their spatial locations. 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, Zhong et al. 2012], show that a distributed index structure can provide an efficient mechanism of spatial operations processing.

However, distributed R-Trees indexes for Big Spatial Data are very complex to be developed and so it demands novel approaches to debug and check stability and this is the main issue investigated in this work.

Debugging 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 [WH Cheung 1990]. In recent years, researches have developed some helpful debugging techniques for distributed environment. Nevertheless, we have not found in the literature any work that have addressed the problem of debugging a distributed R-Tree.

In this paper, we propose a new debugging algorithm for distributed R-Tree building. The debugging algorithm, called RDebug, uses the distributed index structure to aggregate debugging information. RDebug is used on DistGeo, a shared-nothing platform for distributed spatial algorithms processing. We have also created a graphical tool to visualize the debugging information and the R-Tree index structure, called RDebug Visualizer.

The main contributions of this paper are as follows:

- RDebug - A debugging technique for distributed R-Tree building.
- DistGeo - A peer-to-peer platform, with no single point of failure, to process distributed spatial algorithms of an R-Tree.
- RDebug Visualizer - A graphical tool to visualize debugging information and the distributed R-Tree index.

The rest of the paper is structured as follows. In Section 2, we briefly give an overview of the use of debugging techniques for distributed environments and the view of the distributed spatial algorithms. Section 3 describes the distributed processing of spatial algorithms, Section 4 presents our approach for distributed R-Tree debugging. Section 5 presents the evaluation of RDebug algorithm in the DistGeo platform. Finally, we close the paper with some concluding remarks in Section 6.

2. Related work

Researches on distributed spatial data either show techniques to debug distributed applications in general or techniques for R-tree distributed processing, but none addressed both issues. The Section 2.1 shows the distributed debugging researches and 2.2 describes researches of platforms for processing distributed spatial algorithms.

2.1. Distributed Debugging Techniques

In [G. et al. 2011] the author breaks down debuggers in two main families: log-based debuggers (also known as post-mortem debuggers) and breakpoint-based debuggers (also known as online debuggers). Log-based debuggers insert log statements in the code to be able to generate a trace log during its execution. Breakpoint-based debuggers, on the other hand, execute the program in the debug mode that allows programs to pause/resume the execution at certain points, inspect the state and the perform step-by-step execution.

Several breakpoint-based debuggers have been designed for parallel programs using message passing communication including p2d2 [Hood 1996], TotalView [Gottbrath 2009], and Amoeba [Elshoff 1989]. These debuggers offer the traditional commands to stop, inspect and execute step-by-step a running program. Some of them allow to set breakpoints on statements of one process (e.g. [Gottbrath 2009]) or a set of processes (e.g. [Hood 1996], [Elshoff 1989]).

A great body of concurrent and parallel debugging techniques are event-based. Event-based debuggers [C. E. Mcdowell 1989] conceive the execution of a program as a sequence of events. The debugger records the history of the events generated by the application, which can then be used to either browse the events once the application is finished [Fonseca et al. 2007, Stanley et al. 2009], or to replay the execution to recreate the conditions under which the bug was observed.

[WH Cheung 1990] describes a process for distributed debugging in general and does not focus on a specific debugger or a particular technique, the paper focus is on defining a step-by-step approach to tackle distributed debugging independent of the environment.

2.2. Distributed Spatial Algorithms

This Section describes briefly the researches which 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 paper, which shows a shared-nothing architecture, with a master and several workstations connected to a LAN network. The master machine can be a bottleneck because it handles and client requests and moreover merges the answers of the slaves and send to client machines. A similar technique was found on MC-RTree [Schnitzer and Leutenegger 1999] and [An et al. 1999], which show the same problems on master machine.

Hadoop-GIS [Kerr 2009] shows a scalable and high performance spatial data warehousing system for running large scale spatial queries on Hadoop. However, it does not use index to process the spatial operations. [de Oliveira et al. 2011] presents a platform to process distributed spatial operations. Although, the solution proposed in [de Oliveira et al. 2011] implements a distributed index, it is not scalable, since every message go through the replicated master node. [de Oliveira et al. 2013] shows a hybrid peer-to-peer platform, which comprehends a set of machines for naming resolution that could be a bottleneck in the system.

[Xie et al. 2008] introduces a two-phase load-balancing scheme for the parallel GIS operations in distributed environment. [Zhang et al. 2009] describes MapReduce and shows how spatial queries can be naturally expressed in this model. However, it is only indicated for non-indexed datasets.

A number of techniques and platforms have been proposed for handling spatial big data. Nevertheless, none of the researches propose a technique for distributed spatial index debugging of an R-Tree. Besides, none of them propose a platform using a peer-to-peer approach for processing distributed spatial algorithms as found on DistGeo platform (Section 3.1).

3. Distributed Processing of Spatial Algorithms

A number of structures have been proposed for handling multi-dimensional spatial data, such as: KD-Tree [Bentley 1975], Hilbert R-Tree [Kamel and Faloutsos 1994] and R-Tree [Guttman 1984]. 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 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.

The Figure 1(b) shows MBRs grouping spatial objects of $a...h$ in sets by their co-location. The Figure 1(a) illustrates the R-Tree representation. Each node stores at most M and at least $m \leq M/2$ entries [Guttman 1984]. Our work uses the formula for M value calculation presented in [de Oliveira et al. 2011].
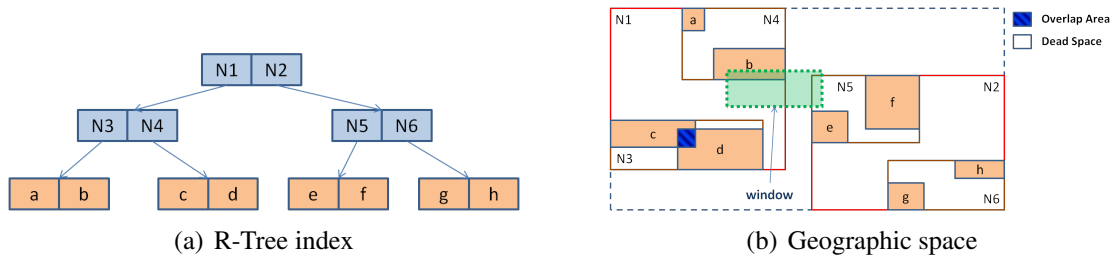

(a) R-Tree index   (b) Geographic space

Figure 1. R-Tree Structure

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.

In Figure 1, the search starts on root node, where the window intersects with nodes $N1$ and $N2$. Then, the algorithm analyses node $N1$, which only $N4$ intersects with the window. Analysing node $N4$, 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 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) depicts

DistGeo platform 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 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 key set start with 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.
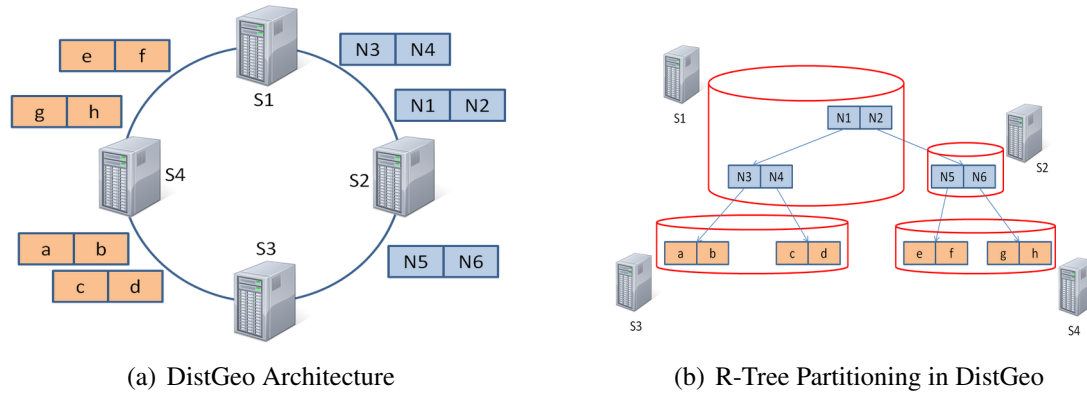

(a) DistGeo Architecture   (b) R-Tree Partitioning in DistGeo

Figure 2. DistGeo Platform

Every replica of an object is equally important, in other words, there is not a master replica. 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. 1987], which every cluster server exchanges information among themselves for everyone knows the status of each server. In the Gossip protocol every second a message is exchanged among three servers in the cluster, consequently every cluster's server have knowledge of each other.

Figure 2(b) illustrates the structure of a Distributed R-Tree in a cluster. The partitioning it is performed grouping the servers in cluster and 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 insert 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 [Cassandra]

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, is selected one of the servers that store a replica of N. The query requests are always sent to one of the cluster's server that stores the root node of the R-tree.

As discussed on Section 3, 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 increase the network traffic because the R-Tree nodes are stored in several servers on cluster, as shown in Figure 2(b). This work implements a new algorithm that collects debugging 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. A Technique for Debugging A Distributed R-Tree

Guarantee that a distributed spatial index has being built accordingly is a non-trivial task. In a distributed environment, it is hard to find bugs on insertion algorithms due 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 overlapping) due the intricacy to collect information about the spatial index.

This section describes RDebug, a new technique for index debugging, which allows collect debugging information about the distributed spatial index once it has been created. The following debug information about building consistency of the R-Tree index are collected by RDebug: i) if each R-Tree node N are consistent between the servers that store any replica of N; ii) if the MBR of each parent node intersects with the MBR of their children, iii) the presence of duplicated nodes on R-Tree or nodes being referenced by more than one parent node, and iv) if 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.

The RDebug algorithm was based on R-Tree structure because it is used to index the spatial datasets on DistGeo platform presented in our work on Sub-Section 3.1. RDebug can be used with any index similar to R-Tree like Hilbert R-Tree [Kamel and Faloutsos 1994], since the RDebug algorithm uses the nodes organization of the R-Tree to collect the debug information.

Algorithm 1 shows the RDebug technique for debugging the distributed spatial index, using the index structure itself. The algorithm has two steps: 1) The algorithm processing is similar to the search in an R-Tree with a top-down traversal; 2) The algorithm does a bottom-up traversal on R-Tree, constructing the result with the debug information.

RDebug has been implemented on DistGeo platform. The R-Tree nodes are distributed and replicated over the cluster. Thus, RDebug can be processed on DistGeo platform 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. During the traversal, at every node accessed the traversal might go to a node of the cluster with less workload, increasing the RDebug algorithm performance.

Algorithm 1: $RDebug(T)$

Data: T reference of the root node of R-Tree $tree$
Result: Debugging 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       $server \leftarrow$ 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   $information \Leftarrow$ the child's information stored on shared memory by replicas of T
14   $replica\_consistency \Leftarrow$ verify the consistency of T in others replicas
15   $node\_consistency \Leftarrow$ verify the consistency of M and m values of T
16   $overlap \Leftarrow$ overlap area of T
17   $dead\_area \Leftarrow$ dead area of T
18   $bound \Leftarrow$ 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     $entry\_info \Leftarrow$ information sent by child node
27     $mbr\_consistent \Leftarrow$ 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     $count \Leftarrow$ retrieve the number of child entries, which did not send a debugging 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**

In the first step, called S1 [Search sub-trees] (lines 1 - 11), the Algorithm 1 traverses 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 the node T is not a leaf (lines 2 - 8), then the number of children entries is stored to control the number of expected answers associated to T in the second step of the algorithm. This information is stored in a shared memory accessed by all servers with a replica of T. Lines $4 - 7$, show that for each entry E in T, a message is sent (continuing step S1) to any server that holds a replica of the child node of E, carrying on the first step in the children nodes. If T is a leaf, the second step, named S2 [Aggregation] is started.

Second step aims (lines $12 - 39$) to aggregate the information about the index to be used for future debugging. This step returns debugging information about each node of the R-Tree. The index itself is used to aggregate this information using the cluster computational resources to improve the algorithm's performance. The index reverse structure facilitates the collection of the debugging information, as one node of the R-Tree is responsible to aggregate only the information of its children.

The debug information about each node of R-Tree in stored in a shared memory that can be accessed by any server that store a replica of T. The RDebug update the information about the node T that is being analyzed between the lines $13 - 18$.

In the line 13, the information is retrieved from the shared memory. Line 14 verifies the consistency of T in the servers that store any replica of T. Line 15 verifies the consistency of M and m values. Lines 16 and 17 calculate the overlap and the dead space area, respectively, for each node of the R-tree. Line 18 get the MBR of the T. This information is inserted in $information$ on line 18.

If the aggregation step is being executed in the leaves (lines 20 - 24), then there are two options. If T is the root node (line 22), the node information is sent to the client application. If T is not the root node, in line 24, the information is sent to the parent node of T.

If the aggregation step is in an internal node (lines 26 - 39), the algorithm aggregates the information of the children nodes. In the 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 acquires the number of children nodes that not sent debugging information yet. This value 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 until each reply is received by child nodes.

The algorithm 1 was implemented in the DistGeo platform to collect the debugging information of the built distributed R-tree. This information is used in the platform to find out indexing issues and for speed up the searching on an R-Tree. Using RDebug algorithm it is possible debug the searching algorithms in a single R-Tree. For exam-

ple, the Window Query algorithm shown on Section 3. Whereas, algorithms that access many R-Trees, such as Spatial Join, need a deep change, as the algorithms can go through different paths.
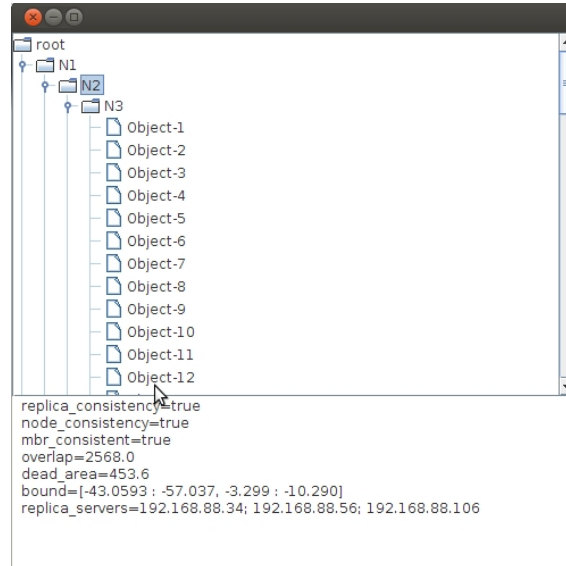

Figure 3. RDebug Visualizer

The algorithm RDebug have collected debugging information about the R-Tree index built during the insertion of the dataset. Figure 3 shows a graphical tool (RDebug Visualizer) created in our work to visualize the collected debugging information. RDebug 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 RDebug algorithm shows which nodes are currently inconsistent. The user can access the path of the node and visualize the node's inconsistent information.

5. Evaluation

The RDebug algorithm have 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[1]. The second dataset comprises 226964 lines representing the rivers on Brazil from LAPIG[2]. The third contains 220000 polygons of the census of USA from TIGER/Line[3].

The RDebug 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

---

[1]https://github.com/simplegeo

[2]www.lapig.iesa.ufg.br

[3]Census 2007 Tiger/Line data

and overlapping area. Furthermore, RDebug algorithm has succeeded to collect the index structure allowing visualize each data set R-Tree index on RDebug Visualizer tool.

Three inconsistencies were deliberately inserted in the index to evaluate the RDebug: 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 RDebug algorithm was able to identify this inconsistencies in every distributed R-Tree related to datasets. The replica consistency on DistGeo is provided by Apache Cassandra [Cassandra] and no replica inconsistencies was found in any test.
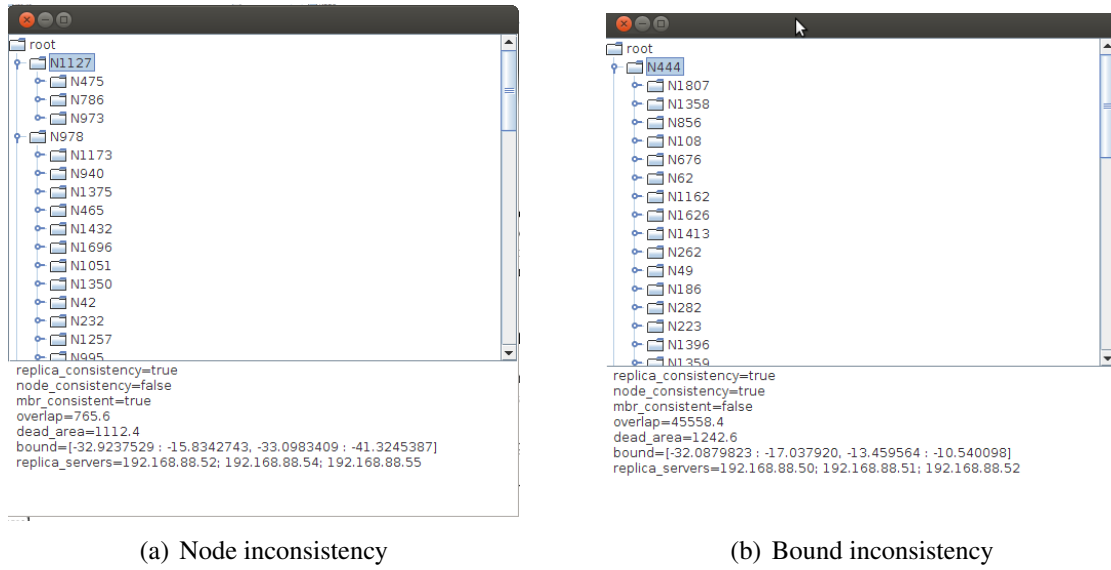

(a) Node inconsistency   (b) Bound inconsistency

Figure 4. RDebug algorithm on business listings dataset

Figure 4 shows the result of RDebug algorithm with the business listings dataset in RDebug Visualizer tool. An example of node inconsistency is shown in Figure 4(a), which the R-Tree node *N1127* contains only three entries. This number of entries violates the m value presented in Section 3. Figure 4(b) shows the bound inconsistency between node *N444* and one of its children. The duplicated nodes identified on R-Tree are shown on final report by RDebug algorithm. The user can traverse the R-Tree path on RDebug Visualizer to identify these duplicated nodes.

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: debugging the R-Tree index distributed in a cluster of computers.

We have seen researches on spatial data processing and distributed debugging, but none of them propose techniques for debugging spatial algorithms in an R-Tree. Our work presents the RDebug algorithm for debugging the building of a distributed R-Tree index. RDebug uses the R-Tree index itself to gather the debug information. The data gathering is achieved in a distributed way, improving the debugging algorithm efficiency.

A new peer-to-peer platform (DistGeo) was proposed in our work to process distributed spatial algorithms. RDebug has been implemented in DistGeo platform. The

R-Tree nodes are distributed and replicated over the cluster. Thus, RDebug can be processed without bottlenecks and point of failures.

A graphical tool(RDebug Visualizer) has been created to visualize the structure of the distributed R-Tree index and the debugging information about the index building. Using this debugging information, we can identify discrepancies in the index building and optimize the R-Tree index too. The RDebug algorithm can be used to collected debug 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 RDebug algorithm to debug the Window Query and Join Query searching algorithms. The RDebug algorithm is easily adapted to gather debugging information for Window Query. Whereas, for Join Query algorithm, RDebug 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 the Rdebug to identify this inconsistencies. On future works, the algorithm RDebug will be evaluated in larger clusters and performance results will be collected.

References

An, N., Lu, R., Qian, L., Sivasubramaniam, A., and Keefe, T. (1999). Storing spatial data on a network of workstations. *Cluster Computing*, 2(4):259–270.

Beckmann, N., Kriegel, H., Schneider, R., and Seeger, B. (1990). *The R*-tree: an efficient and robust access method for points and rectangles*, volume 19. ACM.

Bentley, J. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):517.

C. E. Mcdowell, D. P. H. (1989). Debugging concurrent programs. *ACM Computing Surveys*, 21:593–622.

Cassandra, A. The apache software foundation. *URL: http://cassandra. apache. org/(visited on 24/08/2013)*.

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

de Oliveira, S. S., Vagner, J., Cunha, A. R., Aleixo, E. L., de Oliveira, T. B., Cardoso, M. d. C., Junior, R. R., Bloco, I., and Campus, I. (2013). Processamento distribuído de operações de junção espacial com bases de dados dinâmicas para análise de informações geográficas. *XXXI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

de Oliveira, T., Sacramento, V., Oliveira, S., Albuquerque, P., Cardoso, M., Bloco, I., and Campus, I. (2011). DSI-Rtree - Um Índice R-Tree Escalável Distribuído. In *XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*.

Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., and Terry, D. (1987). Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12. ACM.

Elshoff, I. J. P. (1989). A distributed debugger for amoeba. In *In Symposium on Parallel and distributed tools*, pages 1–10. SIGPLAN Not.

Fonseca, R., Porter, G., Katz, R. H., Shenker, S., and Stoica, I. (2007). X-Trace: A pervasive network tracing framework. In *4th USENIX Symposium on Networked Systems Design And Implementation*, pages 271–284. Cambridge MA, USA.

G., B. E., V., C. T., C., N., D., M. W., and D, H. T. (2011). REME-D: a Reflective Epidemic Message-Oriented Debugger for Ambient-Oriented Applications. *ACM*, pages 1275–1281.

Gottbrath, C. (2009). Deterministically troubleshooting network applications. In *Technical report, TotalView Technologies*. TotalView Technologies.

Guttman, A. (1984). *R-trees: a dynamic index structure for spatial searching*, volume 14. ACM.

Hood, R. (1996). The p2d2 project: building a portable distributed debugger. In *In Symposium on Parallel and distributed tools*, pages 127–136. ACM.

Kamel, I. and Faloutsos, C. (1994). Hilbert R-tree: An Improved R-tree using Fractals. In *VLDB 20th*, page 509. Morgan Kaufmann Publishers Inc.

Kerr, N. (2009). Alternative Approaches to Parallel GIS Processing. *Arizona State University - Master Thesis*.

Koudas, N., Faloutsos, C., and Kamel, I. (1996). Declustering spatial databases on a multi-computer architecture. *Advances in Database Technology-EDBT'96*, pages 592–614.

Schnitzer, B. and Leutenegger, S. (1999). Master-client r-trees: A new parallel r-tree architecture. In *Scientific and Statistical Database Management, 1999. Eleventh International Conference on*, pages 68–77. IEEE.

Stanley, T., Close, T., and Miller, M. (2009). Causeway: A message-oriented distributed debugger. In *Technical Report HPL-2009-78*. HP Laboratories.

WH Cheung, JP Black, E. M. (1990). A Framework for Distributed Debugging. *IEEE*, pages 106–115.

Xie, Z., Ye, Z., and Wu, L. (2008). A two-phase load-balancing framework of parallel gis operations. In *Geoscience and Remote Sensing Symposium, 2008. IGARSS 2008. IEEE International*, volume 2, pages II–1286. IEEE.

Zhang, S., Han, J., Liu, Z., Wang, K., and Feng, S. (2009). Spatial queries evaluation with mapreduce. In *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*, pages 287–292. IEEE.

Zhong, Y., Han, J., Zhang, T., Li, Z., Fang, J., and Chen, G. (2012). Towards parallel spatial query processing for big spatial data. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 2085–2094. IEEE.