RDebug: A New Debugging Technique for Distributed R-Trees

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

saviotetes@gmail.com, jkairos@gmail.com

Abstract. *The growing availability of data and the number of users have motivated the emergence of distributed algorithms to process spatial operations efficiently. Those distributed algorithms are based on distributed indexes for efficient processing. Researches and ongoing work use R-Trees as distributed spatial index structure for indexing and retrieval of objects. However, distributed indexes based on R-Trees have a challenge: How debugging a distributed index based on an R-Tree? In the last years many researches have been published on distributed algorithms and distributed processing. However none of them has addressed the debugging of a distributed R-Tree index. This paper presents a new algorithm for debugging a distributed R-Tree index, hereinafter RDebug. This algorithm has been used on DistGeo, a spatial platform, and to visualize the algorithms' output a graphic tool was created.*

1. Introduction

The Internet has revolutionized the computer and Geographical Information Systems (GIS) like nothing before. In fact the Internet brought big changes how systems store, retrieve and analyze spatial data. The ever-increasing of the large spatial datasets and the widely application of the complex computation make the parallel processing of GIS an important component of high-performance computing. Thus, spatial distributed applications came on the scene to process the spatial operations in a cluster.

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 is the most commonly used spatial indexing on GIS databases. Generally, the R-Tree index is built and processed in a single machine. However, process an R-Tree in a single machine is not feasible because of the huge size of the spatial datasets. Thus, many researches such as [An et al. 1999, de Oliveira et al. 2011, Zhong et al. 2012], show that a distributed index structure spanning the workstations can provide an efficient shared storage structure that can be used to gather geographic information more efficiently.

A big challenge though has arisen of spanning the R-Tree index structure among computers: How to debug the building of the distributed R-Tree?

Debugging is an essential step in the development process, albeit 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 been developed some helpful debugging techniques for distributed environment. Nevertheless, we have not found any technique to debug a distributed R-Tree.

In this paper, we propose a new debugging technique for distributed R-Tree building. The debug method, 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 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. Section 2 describes the distributed processing of spatial algorithms, Section 3 presents our approach for distributed R-Tree debugging. Section 4 presents the evaluation of RDebug algorithm in the DistGeo platform. In Section 5, we briefly give an overview of the use of debugging techniques for distributed environments and the view of the distributed spatial algorithms. Finally, we close the paper with some concluding remarks in Section 6.

2. Distributed Processing of Spatial Algorithms

2.1. Data Structures for Spatial Data Processing

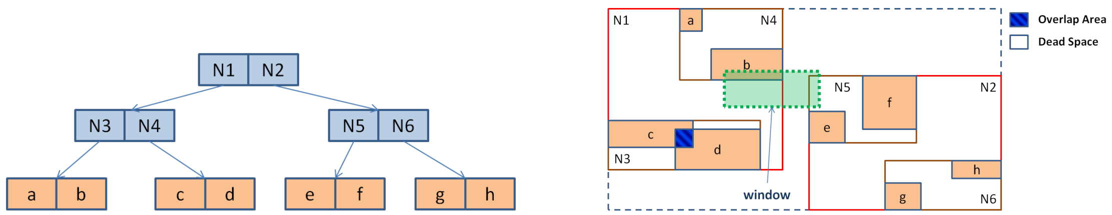
A number of structures have been proposed for handling multi-dimensional point data, such as: KD-Tree [Bentley 1975], Hilbert R-Tree [Kamel and Faloutsos 1994] and R-Tree [Guttman 1984]. The R-Tree has been widely used to index the datasets on GIS databases. It is used to index the datasets in our work due to performance on spatial queries [Guttman 1984].

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 ($N3...6 \subset a...h$). 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].

The Window Query is one of major query algorithms in R-Tree. The input is a search rectangle (Query box). The search starts from the root node of the tree. 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 pointer to some spatial object can be there).

For each rectangle in a node,it has to be decided either it overlaps the search rectangle or not. If yes, the corresponding child node has to be searched also. Searching is done recursively until all overlapping nodes have been traversed. When a leaf node is



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

Figure 1. R-Tree Structure

reached, the contained bounding boxes (rectangles) are tested against the search rectangle and their objects (if there are any) are put into the result set if they lie within the search rectangle.

In Figure 1, the search starts on root node, where window intersects with N1 and N2 nodes. Then, the algorithm analyzes N4 and N5 entries in N1, which only N4 intersects with the window. Analyzing N4, the algorithm returns the object namely 'b', that is the single object that intersects the window.

In 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.

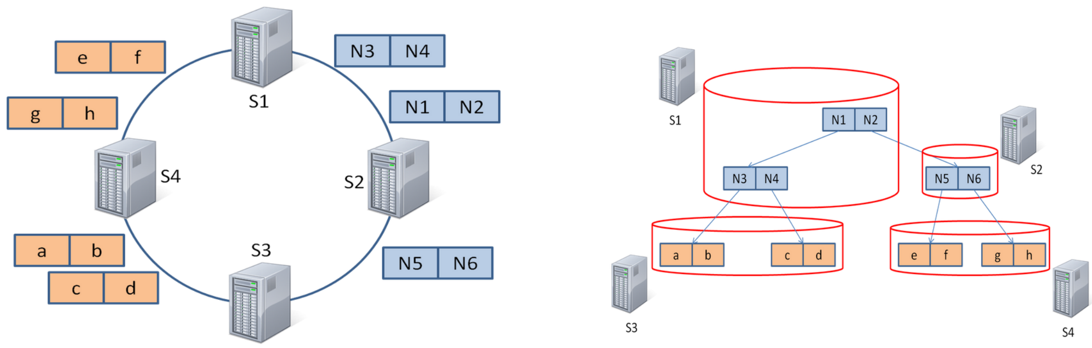
2.2. DistGeo: A Platform of Distributed Spatial Operations for Geoprocessing

This work developed a platform, namely DistGeo (Figure 2), to process the spatial operations in a computer cluster. DistGeo is based on the 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, with the data manage by the cluster presented as a ring topology. It is divided in ranges of keys, which are managed for each node of the cluster. In order to a node join the ring it must be assigned a range first.

The range of keys are known by each node in the cluster. 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 node 2.

Every replica of an object is equally important, in other words, there is not a master replica. Read/Write operations may be performed in any node of the cluster. When a request is made to a cluster's node, 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 node exchanges information among themselves for service discovery and knowing the status



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

Figure 2. DistGeo Platform

of the servers. In the Gossip protocol every second a message is exchanged among three servers in the cluster, consequently every cluster's node 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 in DistGeo.

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.

Some optimizations proposed in [Beckmann et al. 1990] to reduce the overlapping and the dead area were adapted and implemented on DistGeo. Reducing the overlapping and dead area minimizes the number of messages exchange in network on search algorithms, because the query access less nodes during the tree traversal (see Sub-Section 2.1). This work implements a new distributed debug algorithm of the R-Tree index (RDebug) that helps to reduce the overlap and dead area. We cover the RDebug algorithm in more details in Section 3.

3. A Technique for Debugging A Distributed R-Tree

Ensure that the index was built correctly is a non-trivial task. This section describes RDebug, a new technique for index debugging, which allows debugging the index creation of a distributed R-Tree 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 2. Furthermore, it is possible to access index data to help in optimization and minimizing the dead space and overlapping area.

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; 2) The algorithm does the inverse of a search in an R-Tree appending information to the distributed index.

RDebug has been implemented in 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 access of the R-Tree, the traversal might go to a node of the cluster with less workload increasing RDebug performance.

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. Its purpose is to spread the debugging algorithm. 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 to this node 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 every entry E in the node, 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 the node is a leaf, the second step, named S2 [Aggregation] is started.

Second step aim (lines 12 - 39) is to aggregate the information used for future debugging. This step receives the debugging information of every child node of T. Therefore, for a given node T with n children, the second step is invoked n times in the node T.

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.

Algorithm 1: $RDebug(T)$

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

S1 [Search subtrees]
if *T is not leaf* **then**
    stores the number of children entries in each replica server of T
    for *each entry E in T* **do**
        $server \leftarrow$ choose one server, randomly, that stores one replica of E
        send msg to $server$ to process the node's child of E on step S1
    end
else
    verify the consistency of T in others replicas
    Invoke step S2 [Aggregation]
end
S2 [Aggregation]
$information \Leftarrow$ the child's information stored on shared memory by replicas of T
$replica\_consistency \Leftarrow$ verify the consistency of T in others replicas
$node\_consistency \Leftarrow$ verify the consistency of M and m values of T
$overlap \Leftarrow$ overlap area of T
$dead\_area \Leftarrow$ dead area of T
$bound \Leftarrow$ MBR of T
add in $information$: $replica\_consistency$, $node\_consistency$, $overlap$, $dead\_area$, $bound$
if *T is leaf* **then**
    if *T is root* **then**
        send response with R-Tree nodes information to app client
    end
    send msg with $information$ to parent of T
else
    $entry\_info \Leftarrow$ information sent by child node
    $mbr\_consistent \Leftarrow$ verify if the bound of the child node is equal to bound of entry of T that points to this child
    add in $information$: $entries\_info$, and $mbr\_consistent$
    $count \Leftarrow$ retrieve the number of child entries, which did not send a debugging response and decrement by 1
    if *$count$ == 0* **then**
        if *T is root* **then**
            send response with $information$ to client
        else
            send msg with $information$ to parent of T
        end
    else
        store $information$ on shared memory
    end
end

The information of the children nodes are stored in a shared memory, with concurrency control, by the replicas of T. Hence, line 13, those information are 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 for each node of the R-tree, respectively. Those information help the insertion algorithm designer analyze the quality of the built index. Line 18 get the MBR of the T. This data are inserted in $information$ on line 18. This information will be used as an input to a tool capable of visualizing the index of the R-Tree.

If the aggregation step is being executed in the leaves (lines 20 - 24), then if T is the root node (line 22), the node information are sent to the client application. If T is not the root node, in line 24, the information are 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 the values is stored on shared memory to let the other replicas know.

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 are sent to the client application, otherwise, those information are sent to the parent node of T. If the variable $count$ is greater than 0, then the client information are stored in the shared memory to be used until every child node send replies.

The algorithm 1 was implemented in the DistGeo platform to collect the debugging information of the built distributed R-tree. Those information are used in the platform to find out indexing issues and to optimize the R-tree index for searching. With the aid of RDebug 1 algorithm, it is possible debug the searching algorithms of an R-Tree. E.g: The Window Query algorithm shown on Section 2.1. To tweak RDebug to Window Query, it is only needed add an window query in the first step and gather the aggregation information of the accessed nodes. Whereas, the algorithms that access diverse R-Trees, such as Spatial Join, need a deep change, as the algorithms can go through different paths.

The algorithm RDebug have collected debugging informations about the R-Tree index built during the insertion of the dataset. Figure 3 shows a graphical tool (RDebug Visualizer) created in this 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, so the user can access the path of the node and visualize the node's inconsistent information.

4. 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.

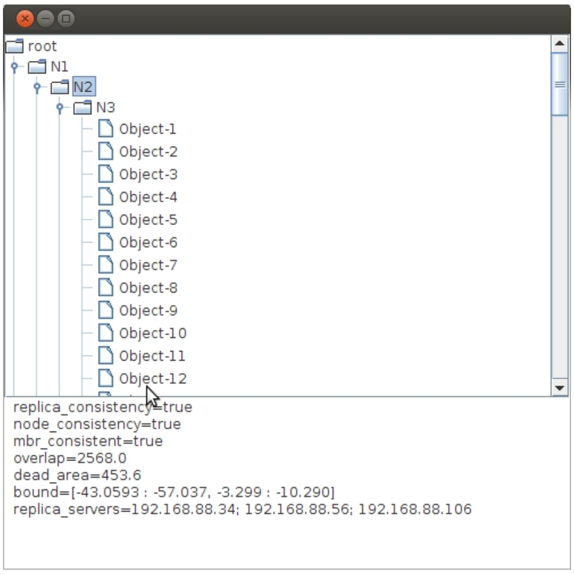


Figure 3. RDebug Visualizer

Each node has 16 GB of main memory. The experiment results were achieved with 1, 2, 4 and 8 servers on DistGeo platform.

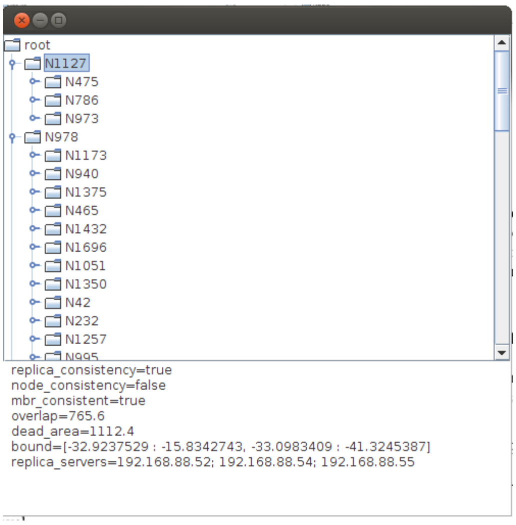
The experiments used three data sets with different characteristics. The first contains 1000000 points of business listings and points of interest (POIs) from SimpleGeo[1]. The second dataset comprises 226964 lines representing the rivers on Brazil from LAPIG[2]. The third contains 220000 polygons of the census of USA from TIGER/Line[3].

The RDebug was executed on DistGeo platform after the insertion and indexing of each dataset. The algorithm was able to collect information from the R-Tree index, such as dead space 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.

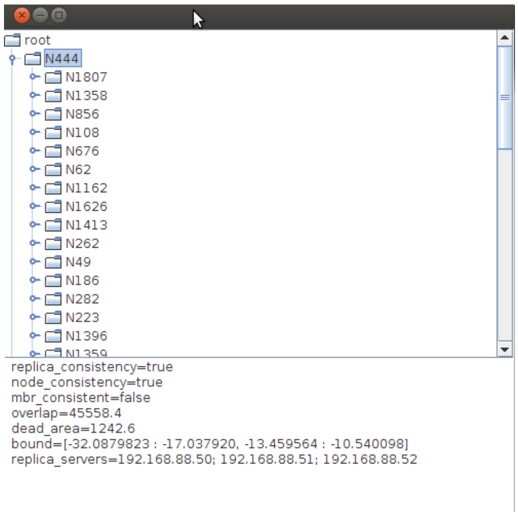
Three inconsistencies were deliberately inserted in the index to evaluate the RDebug: i) bound inconsistencies between parent and child nodes, ii) nodes filled with more than M entries and iii) duplication of a node on R-Tree. The RDebug algorithm was able to identify this inconsistencies on distributed R-Tree. The replica consistency on DistGeo is supported by Apache Cassandra [Cassandra] and no replica inconsistencies was found in any test.

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 2. 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.

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



(a) Node inconsistency



(b) Bound inconsistency

Figure 4. RDebug algorithm on business listings dataset

5. Related work

Researches on distributed spatial data either show techniques to debug distributed applications in general or techniques for R-tree distributed processing. Techniques for debugging a distributed R-Tree index was not found in literature. The Section 5.1 shows the distributed debugging researches and 5.2 describes researches of platforms for processing distributed spatial algorithms.

5.1. 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 of the program 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 program execution at certain points, inspect program state and 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, e.g. stop, inspect and step-by-step execution of 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]). An interesting alternative to traditional breakpoints is message breakpoints [Wismuller 1997].

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.

5.2. Distributed Spatial Algorithms

This Section describes briefly the 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 paper, which shows a shared-nothing architecture, with a master and several workstations connected to a LAN. The master machine handles a high volume of computation, besides of processing some directories of the R-Tree, it merges the answers to the client machines. 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. However the gain running large scale queries, it does not use indexing. [de Oliveira et al. 2011] presents a distributed platform for 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. In [Zhong et al. 2012], an approach is proposed for "indexing + MapReduce" data processing architecture to improve the spatial query performance.

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

6. Conclusion

DistGeo platform presents an approach for processing the 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 the spatial algorithms of an R-Tree. In this paper, we present RDebug, a new technique for debugging the distributed index building of an R-Tree. RDebug, uses the R-Tree index itself to gather the debug information. The information gathering is done in the R-Tree index in a top-down traversal, using the distributed index itself. The data gathering can be achieved in a distributed way, improving the debugging algorithm efficiency.

A new peer-to-peer platform (DistGeo) was proposed in this 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. 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 access of the R-Tree, the traversal might go to a node of the cluster with less workload increasing RDebug performance. The information exchange of the machines statuses is done trough the Gossip algorithm.

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. With these input we can identify discrepancies in the index building and optimize it.

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 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.

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.

Wismuller, R. (1997). Debugging message passing programs using invisible message tags. In *In European PVM/MPI UsersâĂŹ Group Meeting*, page 295âĂŞ302. Springer-Verlag.

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.