A system for GIS polygonal overlay computation on Linux Cluster - An experience and performance report

Dinesh Agarwal, Satish Puri, Xi He, and Sushil K. Prasad[1]
Department of Computer Science
Georgia State University
Atlanta - 30303, USA
Email: {dagarwal2,spuri2,xhe8}@student.gsu.edu, sprasad@gsu.edu

***Abstract*—GIS polygon-based (also know as vector-based) spatial data overlay computation is much more complex than raster data computation. Processing of polygonal spatial data files has been a long standing research question in GIS community due to the irregular and data intensive nature of the underlying computation. The state-of-the-art software for overlay computation in GIS community is still desktop-based. We present a cluster-based distributed solution for end-to-end polygon overlay processing, modeled after our Windows Azure cloud-based Crayons system [1]. We present the details of porting Crayons system to MPI-based Linux cluster and show the improvements made by employing efficient data structures such as R-trees. We present performance report and show the scalability of our system, along with the remaining bottlenecks. Our experimental results show an absolute speedup of 15x for end-to-end overlay computation employing upto 80 cores.**

Keywords: Irregular Data Intensive Computation, Dynamic Load Balancing, R-tree, Overlay operation, Vector Data, GML files

I. Introduction

Scalable vector data computation has been a challenge in Geographic Information Science and Systems (GIS). When large volumes of data are deployed for spatial analysis and overlay computation (see Figure 1), it is a time consuming task, which in many cases is also time sensitive. For emergency response in the US, for example, disaster-based consequence modeling is predominantly performed using HAZUS-MH, a FEMA-developed application that integrates current scientific and engineering disaster modeling knowledge with inventory data in a GIS framework [7]. Depending on the extent of the hazard coverage, datasets used by HAZUS-MH have the potential to become very large, and often beyond the capacity of standard desktops for comprehensive analysis, and it may take several hours to obtain the analytical results. Although processing speed is not critical in typical non-emergency geospatial analysis, spatial data processing routines are computationally intensive and run for extended periods of time. In addition, the geographic extents and resolution could result in high volumes of input data.

In this paper, we present a parallel system to execute traditional polygon overlay algorithms on a Linux cluster with InfiniBand interconnect using MPI framework.

We present the algorithm for carrying out the overlay computation starting from two input GML (Geography Markup Language) files, their parsing, employing the bounding boxes of potentially overlapping polygons to determine the basic overlay tasks, partitioning the tasks among processes, and melding the resulting polygons to produce the output GML file. We describe the software architecture of our system to execute the algorithm and discuss the design choices and issues. Our instrumenting of the timing characteristics of various phases of the algorithm rigorously point out portions of the algorithm which are easily amenable to scalable speedups and some others which are not. The latter is primarily related to file related i/o activities. The experiments also point out the need for input and output GML files to be stored in a distributed fashion (tranparent to the GIS scientists) as a matter of representation to allow efficient parallel access and processing.

Our specific technical contributions are as follows:

- Porting the Windows Azure cloud-based spatial overlay system to Linux cluster using MPI
- Implementing and improving an end-to-end overlay processing system for a distributed cluster
- An absolute speedup of over 15x using 80 cores for end-to-end overlay computation over moderate sized GML data files of 770 MB intersected with 64 MB file with skewed load profile.

The rest of this paper is organized as follows: Section II reviews the literature briefly and provides background on GIS raster and vector data, various operations that define parallel overlay, and R-tree and general polygon clipper (GPC) library. Section III describes two flavors of task partitioning and load distribution. Several key implementation related issues are discussed in Section IV. Our experimental results and other experiments are in V. Section VI concludes this paper.

[1]This work is partially supported by NSF CCF 1048200 project.

Source	Example Type	Description	File Size
US Census [2]	Block Centroids	Block centroids for entire US	705 MB
	Block Polygons	2000 Block polygons for the state of Georgia	108 MB
	Blockgroup Polygons	2000 Blockgroup polygons for the state of Georgia	14 MB
GADoT [3]	Roads	Road centerlines for 5-county Atlanta metro	130 MB
USGS [4]	National Hydrography Data set	Hydrography features for entire US	13.1 GB
	National Landcover Data set	Landcover for entire US	3-28 GB
JPL [5]	Landsat TM	pan-sharpened 15m resolution	4 TB
Open Topography [6]	LIDAR	LIDAR point clouds 1-4 pts/sq. ft	0.1-1 TB

Fig. 1: Example GIS data sets and typical files

II. Background and Literature

A. *Data Types in GIS*

In GIS the real world geographic features are prominently represented using one of the two data formats: raster and vector. Raster form stores the data in grids of cells with each cell storing a single value. Raster data requires more storage space than vector data as the representation cannot automatically assume the inclusion of intermediate points given the start and end points. In vector data, geographic features are represented as geometrical shapes such as points, lines, and polygons. Vector data is represented in GIS using different file formats such as GML and shapefile, among others. In our experiments, we use GML file format (XML-based) to represent input and output GIS data.

B. *Map Overlay*

Map Overlay is one of the key spatial operations in GIS. It is the process of interrelating several spatial features (points, lines, or polygons) from multiple datasets, which creates a new output vector dataset, visually similar to stacking several maps of the same region together. These overlays are similar to mathematical Venn diagram overlays. For instance, one map of Japan representing population distribution and another map representing the area affected by Tsunami can be overlaid to answers queries such as “What is the optimal location for a rescue shelter?” Clearly, it is often needed to combine two or more maps according to logical rules called overlay operations in GIS. A *union* overlay operation combines the geographic features and attribute tables of both inputs into a single new output. An *intersection* overlay operation, similarly, defines the overlapping area and retains a set of attribute fields for each. In case of raster data format, the operation is called grid overlay, and, in the case of vector data format, the operation is called polygon overlay. It should be noted here that the process of map overlay in case of raster data is entirely different from that of vector data and our solution deals with vector data only. Since resulting topological features are created from two or more existing features of the input map layers, the overlay processing task can be time consuming and CPU intensive.

For identification of overlaying-map-features, different algorithms based on uniform grid, plane sweep, Quad-tree, and R-tree have been proposed and implemented on classic parallel architectures [8]–[10]. Franklin et al. [11] presented the uniform grid technique for parallel edge-intersection detection. Their implementation was done using Sun 4/280 workstation and 16 processor Sequent Balance 21000. Waught et al. [12] presented a complete algorithm for polygon overlay and the implementation was done on Meiko Computing Surface, containing T800 transputer processors using Occam programming language. Data partitioning in [11]–[14] is done at spatial level by superimposing a uniform grid over the input map layers. Armstrong et al. [15] presented domain decomposition for parallel processing of spatial problems. While a wealth of research shows gains in performance over sequential techniques [16], [17], its application in mainstream GIS software has been limited [18], [19]. There has been very little research in high volume vector spatial analysis [20] and the existing literature lacks an end-to-end parallel overlay solution.

C. *R-tree*

R-tree is an efficient spatial data structure for rectangular indexing of multi-dimensional data; it performs well even with non-uniform data. R-tree data structure provides standard functions to insert and search polygons by their bounding box co-ordinates. We use R-tree for intersection detection among polygons required for efficient overlay processing. Searching for overlap in R-tree is typically an $O(log_m n)$ operation where n is the number of nodes and m is the number of entries in a node, although it can result in $O(n)$ complexity in the worst case. We use a third party library for sequential R-tree [21] which implements Guttman’s algorithm [22] for R-tree construction and search operation.

D. *Crayons system on Azure cloud*

We have earlier developed a distributed cloud-based framework named Crayons [1] to execute traditional polygon overlay analysis algorithms on Windows Azure cloud platform. Windows Azure platform is a computing and service platform hosted in Microsoft data centres. Its programming paradigm employs two types of processes called web role and worker role for computation. For communication between web role and worker roles, it provides queue-based messaging and for storage it provides blobs and tables. Three load balancing mechanisms showing excellent speedup in Crayons are discussed in the paper [1].

However, Windows Azure platform lacks support for traditional software infrastructures such as MPI and map-reduce.

Our current work generalizes Crayons by porting it to a Linux cluster with support for MPI framework. Porting a cloud application has its own set of challenges and we discuss some of them in this paper. We also made improvements on top of the Crayons' design by incorporating R-tree - an efficient spatial data structure. We believe that cluster based faster and efficient end-to-end GIS solution will aid GIS community as a whole.

E. Clipper Library

For computing map overlay over a pair of polygons, we use the GPC library which is an implementation of polygon overlay methods as described in [23]. The GPC library handles polygons that are convex or concave and self-intersecting. It also supports polygons with holes or polygons comprising of several disjoint contours. The usage of this widely used library shows the interoperability and accuracy of our approach for our polygon overlay solution. GPC library supports intersection, union, difference, and X-OR. The output may take the form of polygon outlines or tristrips. We analyze and report *intersection* overlay operation throughout this project since it is the most widely used and is a representative operation. Nevertheless, our system can be extended to other operations as well without any change.

III. Our MPI-based System Design

We have implemented two versions of our system employing static and dynamic load balancing. The two versions further differ in the way task creation is carried out. Our system has a four-step workflow which consists of input file parsing, task creation, overlay computation, and output file creation.

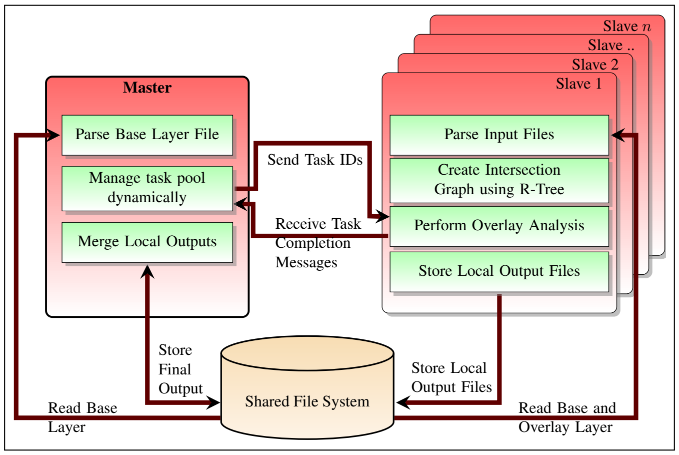


Fig. 2: Architecture with dynamic load balancing

A. Architecture with dynamic load balancing using R-tree

Figure 2 shows master-slave architectural diagram for dynamic load balancing version with task creation by employing R-tree. The first step of our workflow starts with both the master and the slave processes parsing the GML files - only base layer for master process and both the base and overlay layer files for slave processes. This redundancy later helps in task creation and processing.

Once parsing is done, each slave process builds its R-tree using the bounding boxes of the overlay layer polygons. Master process determines the total number of base layer polygons and dynamically partitions those polygons into small chunks. The start and end indices of each chunk are sent to the slave processes. Once a slave process receives the polygon indices, the next step is to search its local R-tree for potentially intersecting overlay layer polygons. Each slave process creates an intersection graph where each polygon from the base layer is connected with all of the polygons from overlay layer that can potentially intersect with it. Algorithm 1 describes the steps involved in creating the intersection graph based on R-tree. Once the intersection graph has been created, each polygon from base layer and the connected polygons from overlay layer are stored together as a primitive overlay task for a slave process for processing. The overlay processing itself is carried out by invoking the built-in overlay function of the GPC library. The output from this library is a polygon structure that is converted to its equivalent GML representation. Once a slave worker finishes processing its current set of tasks, it sends a completion message to the master, who in turn sends indices of the next chunk to be processed. Finally, when all tasks are completed, the master process merges all the local output files to generate the final output GML file.

Algorithm 1 R-tree based algorithm to create intersection graph

INPUT: Set of Base Layer polygons S_b and Set of Overlay Layer polygons S_o
OUTPUT: Intersection Graph (V,E), where V is set of polygons and E is the set of edges among polygons with intersecting bounding boxes.

```
Create an R-tree R using the bounding boxes of $S_o$
for all base polygon $B_i$ in set $S_b$ do
    for each polygon $O_j$ in R-tree R with bounding box intersecting with that of $B_i$ do
        Create an edge ($B_i$, $O_j$) in graph G
    end for
end for
```

We experimented with different grain (chunk) sizes guided by the distribution of polygons in a map layer. For non-uniform load, grain size should be smaller to account for load imbalance in comparison to the uniformly distributed data. The master process can be visualized as the owner of a pool of tasks from which it assigns tasks to slave processes. The slave processes continuously check with the master process for new tasks once they are done processing their individual tasks. The master-slave communication is message-based, handled by MPI send/receive primitives. The message size is intentionally kept small to lower the communication overhead while the message count (and thus the grain size) is determined by empirical data.

B. Architecture with dynamic load balancing using sorting-based algorithm

We used sequential R-trees for creating intersection graphs in the previous version. As suggested in the literature [10], we also used a different algorithm for creating intersection graph, which is based on sorting the polygons on their bounding boxes. In this algorithm, each slave process sorts all the overlay polygons using the x coordinates of their bounding boxes (first sort on left x coordinate and then over right). Then, iteratively, for each sets of base polygons assigned by the master to work on, each slave searches over this sorted list of overlay polygons to create its intersection graph. Algorithm 2 has the details of the sorting-based algorithm for detecting polygon intersection. The time taken of this intersection graph algorithm is much higher compared to R-tree based search.

Algorithm 2 Sorting-based algorithm to create intersection graph

INPUT: Set of Base Layer polygons S_b and Set of Overlay Layer polygons S_o
OUTPUT: Intersection Graph (V,E), where V is set of polygons and E is edges among polygons with intersecting bounding boxes.

```
Quicksort set $S_o$ of overlay polygons based on X co-ordinates of bounding boxes
for all base polygon $B_i$ in set $S_b$ of base polygons do
  Find $S_x \subset S_o$ such that $B_i$ intersects with all polygons in set $S_x$ over $X$ co-ordinate (binary search over $S_o$)
  Quicksort $S_x$ on y coordinates of bounding boxes
  for each polygon $O_j$ in $S_x$ that $B_i$ intersects with in Y co-ordinate do
  Create an edge ($B_i$, $O_j$ ) in graph G
  end for
end for
```

C. Our system with Static Load Balancing

Based on the method of intersection graph creation, we have two flavors that employ static load balancing. One of them makes use of R-tree (Algorithm 1) and the second one simply uses binary search on polygons sorted on bounded boxes (Algorithm 2). The rationale behind developing static version is to assess the communication cost between master and slave processes involved in the dynamic version. Although the basic workflow is similar to dynamic version, task partitioning in the static version is straightforward. After independently parsing both of the input GML files, the slave processes equally divide the base layer polygons among themselves based on their process-IDs and perform task creation only for their portion of base layer polygons, thereby obviating any need for master-slave communication. Master process is only responsible for merging the files created by different slave processes.

Output file creation is initiated by master process once all the slave processes finish their tasks and terminate. Termination detection differs in static and dynamic version and is handled by master process. In static version, once master process receives task completion messages from all slave processes, it merges the partial (local) output GML files created by respective slave process to yield final output and finally terminates. On the other hand, in the dynamic version, master process interactively tracks the completion of tasks and once all tasks are finished, it sends termination message to each slave process and generates an output GML file.

IV. Implementation Issues

A. MPI related Issues

As seen in section III, each and every slave process reads input files. The parsing of files is a sequential bottleneck here and it is performed redundantly. This problem worsens, due to i/o contention for the shared file system, when the number of processors increase. It is, therefore, more intuitive to let only master process parse files once, create task and schedule them dynamically for slave processes to execute. We tried this approach initially, but the underlying communication and packing/unpacking overheads made it impractical.

1) MPI Send/Receive issue: In order to distribute work among slave processes, the master needs to communicate the tasks using MPI send/receive primitives. The vector data needed for the GPC library is a nested structure containing polygon information including number of contours, number of vertices, bounding box information, etc. The polygonal data is non-contiguous data of mixed data types. Even though MPI provides derived data type for user-defined data structures and packing routines, these do not work for dynamically allocated nested structure that we had to use.
2) Cost of serialization: To overcome the MPI Send/Receive issue mentioned above, we serialized the polygonal data as a text stream. In this version, master process creates tasks, serializes and sends to slave processes. Each slave process receives the message and deserializes to get the task and performs overlay computation. Vector data tends to be large in size and experimentally we found that the cost of serialization/deserialization and message communication is huge even for smaller grain sizes. Thus, we chose the redundant file reads by all the slaves.

B. Clipper Library Related Issues

GPC library is a well-known open-source library for basic polygon clipping operations but it has some limitations. First and foremost, the GPC library only supports four operations - intersection, X-OR, union, and difference. It does not support *Equals*, *Crosses*, *Within*, *Contains*, *Disjoint*, *Touches*, and *Overlap*. Moreover, the library does not preserve the relationship between a hole and the polygon that contains this hole. Since we work with only one polygon pair at a time, this is a non-issue as holes must belong to the resultant polygon.

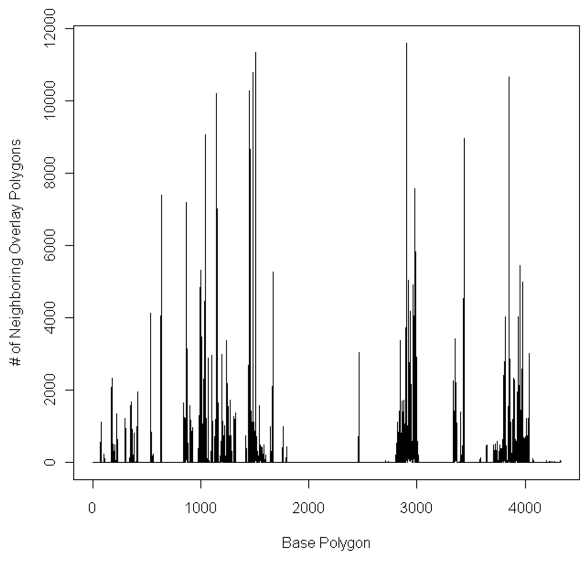


Fig. 3: Skewed load distribution for smaller data set

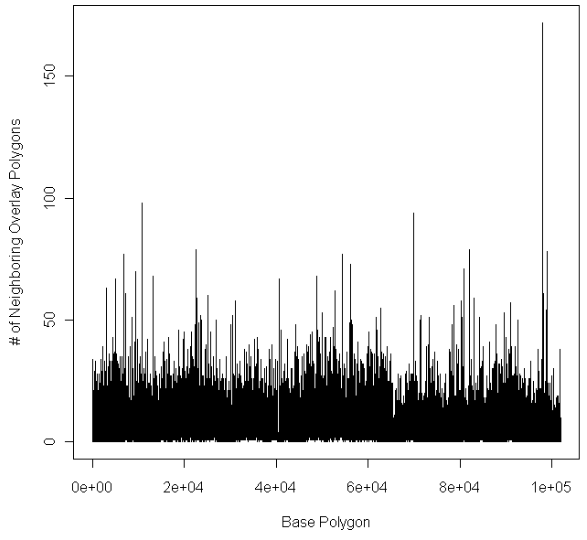


Fig. 4: Comparatively uniform load distribution for larger data set

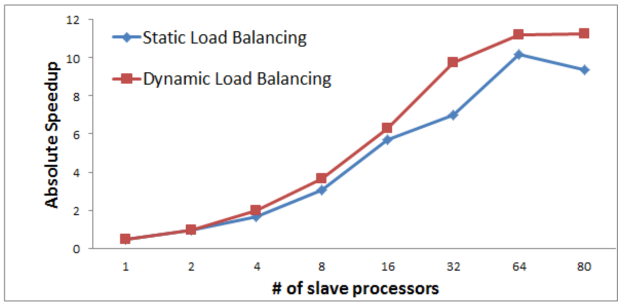


Fig. 5: Performance of sorting-based algorithm

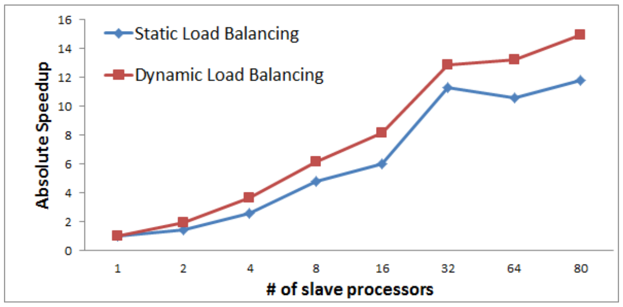


Fig. 6: Performance of R-tree based algorithm

However, our open-architecture allows replacing GPC library with other libraries.

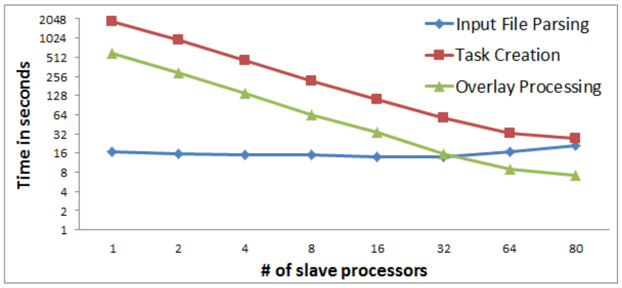
V. Timing Characteristics and Experiments

We have performed our experiments on a Linux cluster that has 80 cores distributed among 9 compute nodes interconnected by an InfiniBand network. The cluster contains 1) four nodes with each having two AMD Quad-Core Opteron model 2376 (2.3 GHz), 2) one node with four AMD Quad-Core Opteron model 8350 (2.0 GHz), and 3) four nodes with each having two Intel Xeon Quad-Core 5410 (2.33 GHz) . In our cluster, all the nodes share the same file system hosted at the head node.

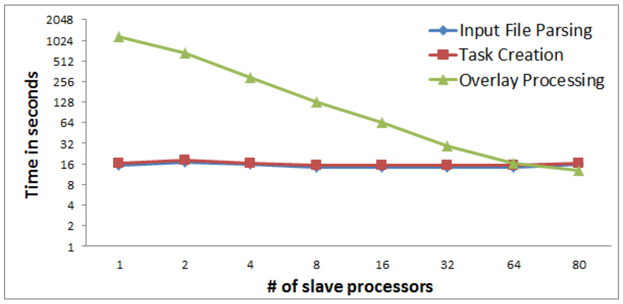
We experimented with two different sets of data. First set consists of files of size 770 MBs containing 465,940 polygons (overlay layer) and 64 MBs containing 8600 polygons (base layer). This data set has skewed load distribution. The second data set consists of files of size 484 MBs containing 200,000 polygons and 636 MBs containing 250,000 polygons. This is the larger data set but the load distribution is uniform here. Figure 3 and Figure 4 shows the load distribution plots for a sample of the base layer polygons used in experiments. To calculate absolute speedup against the sequential time without any parallel overhead, unless otherwise stated, all benchmarking has been performed over the end-to-end 1-core time (the process of taking two GML files as input, performing overlay processing, and saving the output as a GML file) using R-tree based algorithm on the smaller data set and executed on AMD Quad-Core Opteron model 2376 (2.3 GHz).

Figure 5 shows the absolute speedup when we use sorting-based algorithm. For dynamic version using R-tree, the overall end-to-end acceleration is about 15x as shown in Figure 6. R-tree based version cleary shows better performance in comparison to the sorting based version. Dynamic version works better than static version due to the non-uniform distribution of polygonal data in small data set as we mentioned earlier. Moreover, master-slave communication time is relatively small.

Figure 7 shows the execution time breakdown of subprocesses for the static versions. Task creation step involves intersection graph creation based on either R-tree or sorting,

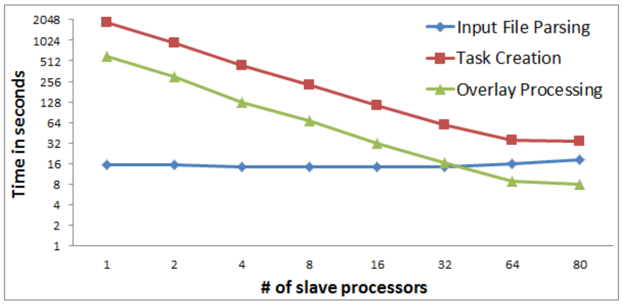


(a) Static Load Balancing (Sorting-based algorithm)

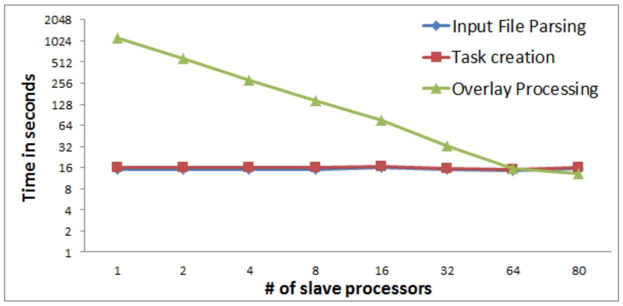


(b) Static Load Balancing (R-tree based algorithm)

Fig. 7: Execution time breakdown for static version (smaller data set)



(a) Dynamic Load Balancing (Sorting-based algorithm)

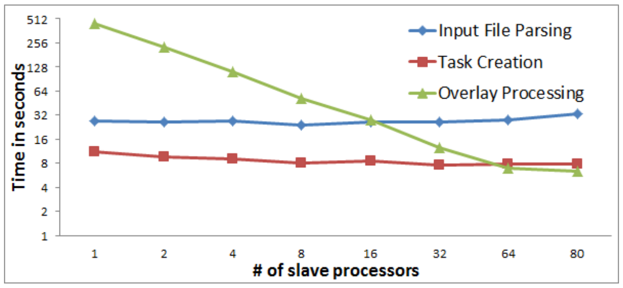


(b) Dynamic Load Balancing (R-tree based algorithm)

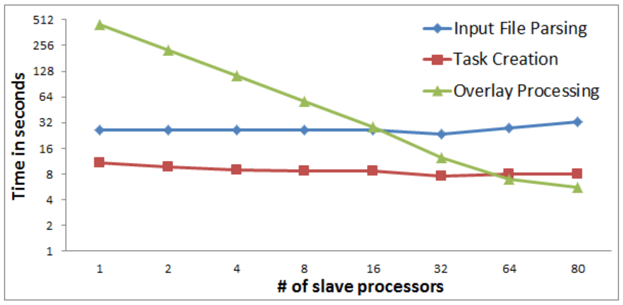
Fig. 8: Execution time breakdown for dynamic versions (smaller data set)

and task partitioning so that slave processors can work on independent tasks. Overlay processing step involves computing overlay and writing the local output polygons to GML files. Figure 8 shows the execution time breakdown of subprocesses for dynamic versions. The reported time in Figure 7, 8 and 9 is the average time recorded by noting the time for each of the three subprocesses, i.e., parsing, task creation, and overlay task processing, for each slave process and then taking an average. Overlay task processing includes assigning the tasks to *GPC* library and is followed by the output storing step where the local outputs are stored in the shared file system as a separate file (one file for each slave process). The overlay processing time in case of R-tree based version is more than sorting-based version for the same dataset as can be seen from Figure 7(b) and Figure 8(b). This is due to the fact that when we use a third party implementation of R-tree data structure, we get more potentially intersecting polygons for a given base layer polygon in comparison to the sorting-based version. It should be noted here that all the potentially intersecting polygons may not actually intersect. So, this does not affect the correctness of our final output.

Figure 9 shows the subprocess timing for R-tree based versions using larger data set with comparatively uniform load distribution. Even though the file sizes and the number of pairs of polygons and thus tasks are higher for this larger data set, the polygons themselves comprise fewer vertices amounting to reduced execution times for overlay computation in comparison with the “smaller” data set. For both the smaller and the larger data sets, the overlay processing task scales very well for static as well as dynamic load balancing as the number



(a) Static Load Balancing (R-tree based algorithm)



(b) Dynamic Load Balancing (R-tree based algorithm)

Fig. 9: Execution time breakdown using R-tree for larger data set

of slave processors increase. The communication cost is small in comparison with the cost of input file parsing and task creation. However, we observed that due to the contention for parallel file access, parsing of the input GML files and writing of output files takes longer with the number of processors growing. Further scaling of this system is challenging unless high throughput file access can be supported and the task creation and partitioning can be scaled.

VI. Conclusions and Future Work

In this paper we have created an end-to-end system for an important class of data intensive applications on Linux cluster. The version with R-tree outperforms the version using sorting and thus shows the efficiency of our approach. The input and output GML file processing phases clearly point to further acceleration possibly based on distributed file storage of these GML files. The platform enables experimenting with third party overlay solutions thus exhibiting flexibility of our system. We are exploring use of parallel filesystems like lustre and parallel virtual file system (pvfs2) to minimize i/o bottleneck.

VII. Acknowledgement

Authors would like to acknowledge Chad Frederick for his help with GIS data partitioning algorithms.

References

[1] D. Agarwal, S. Puri, X. He, and S.K. Prasad, Crayons - An Azure Cloud based parallel system for GIS overlay operations. [Online]. Available: www.cs.gsu.edu/dimos/crayons.html

[2] Census.gov, "US Census Data," http://www.census.gov/, December 2011.

[3] GDOT, "Georgia department of transportation," http://www.dot.state.ga.us/Pages/default.aspx, 1916.

[4] USGS, "U.S. geological survey," http://www.usgs.gov/, 1879.

[5] NASA, "Jet propulsion laboratory," http://www.jpl.nasa.gov/, 1936. [Online]. Available: http://www.jpl.nasa.gov/

[6] Open Topography Facility, "Open topography," http://opentopo.sdsc.edu/gridsphere/gridsphere?cid=geonlidar.

[7] Hazus website. [Online]. Available: http://www.fema.gov/plan/prevent/hazus/

[8] J. D. Hobby, "Practical segment intersection with finite precision output," *Computational Geometry*, vol. 13, no. 4, pp. 199 – 214, 1999. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925772199000218

[9] R. G. Healey, M. J. Minetar, and S. Dowers, Eds., *Parallel Processing Algorithms for GIS*. Bristol, PA, USA: Taylor & Francis, Inc., 1997.

[10] F. Wang, "A parallel intersection algorithm for vector polygon overlay," *Computer Graphics and Applications, IEEE*, vol. 13, no. 2, pp. 74 –81, mar 1993.

[11] W. Franklin, C. Narayanaswami, M. Kankanhalli, D. Sun, M. Zhou, and P. Wu, "Uniform grids: A technique for intersection detection on serial and parallel machines," in *Proceedings of Auto-Carto*, vol. 9, 1989, pp. 100–109.

[12] T. Waugh and S. Hopkins, "An algorithm for polygon overlay using cooperative parallel processing," *International Journal of Geographical Information Science*, vol. 6, no. 6, pp. 457–467, 1992.

[13] S. Hopkins and R. Healey, "A parallel implementation of Franklins uniform grid technique for line intersection detection on a large transputer array," *Brassel and Kishimoto [BK90]*, pp. 95–104, 1990.

[14] Q. Zhou, E. Zhong, and Y. Huang, "A parallel line segment intersection strategy based on uniform grids," *Geo-Spatial Information Science*, vol. 12, no. 4, pp. 257–264, 2009.

[15] M. Armstrong and P. Densham, "Domain decomposition for parallel processing of spatial problems," *Computers, environment and urban systems*, vol. 16, no. 6, pp. 497–513, 1992.

[16] P. K. Agarwal, L. Arge, T. Mølhave, and B. Sadri, "I/O-efficient efficient algorithms for computing contours on a terrain," in *Proceedings of the twenty-fourth annual symposium on Computational Geometry*, ser. SCG '08. New York, NY, USA: ACM, 2008, pp. 129–138. [Online]. Available: http://doi.acm.org/10.1145/1377676.1377698

[17] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha, "Fast computation of database operations using graphics processors," in *ACM SIGGRAPH 2005 Courses*, ser. SIGGRAPH '05. New York, NY, USA: ACM, 2005. [Online]. Available: http://doi.acm.org/10.1145/1198555.1198787

[18] T. M. Chan, "A simple trapezoid sweep algorithm for reporting red/blue segment intersections," in *In Proc. 6th Canad. Conf. Comput. Geom*, 1994, pp. 263–268.

[19] B. Chazelle and H. Edelsbrunner, "An optimal algorithm for intersecting line segments in the plane," *J. ACM*, vol. 39, pp. 1–54, January 1992. [Online]. Available: http://doi.acm.org/10.1145/147508.147511

[20] S. Dowers, B. M. Gittings, and M. J. Mineter, "Towards a framework for high-performance geocomputation: handling vector-topology within a distributed service environment," *Computers, Environment and Urban Systems*, vol. 24, no. 5, pp. 471 – 486, 2000.

[21] R-Tree Source Code. [Online]. Available: http://www.rtreeportal.org/code.html

[22] A. Guttman, *R-trees: A dynamic index structure for spatial searching*. ACM, 1984, vol. 14, no. 2.

[23] GPC clipper library. [Online]. Available: http://www.cs.man.ac.uk/~toby/alan/software/gpc.html