High Performance Spatial Queries for Spatial Big Data: from Medical Imaging to GIS

Fusheng Wang[1], Ablimit Aji[2], Hoang Vo[3]
[1]Department of Biomedical Informatics, Department of Computer Science, Stony Brook University, USA
[2]HP Labs, USA
[3]Department of Mathematics and Computer Science, Emory University, USA

Abstract

Support of high performance queries on large volumes of spatial data has become increasingly important in many application domains, including geospatial problems in numerous disciplines, location based services, and emerging medical imaging applications. There are two major challenges for managing massive spatial data to support spatial queries: the explosion of spatial data, and the high computational complexity of spatial queries. Our goal is to develop a general framework to support high performance spatial queries and analytics for spatial big data on MapReduce and CPU-GPU hybrid platforms. In this paper, we introduce Hadoop-GIS – a scalable and high performance spatial data warehousing system for running large scale spatial queries on Hadoop. Hadoop-GIS supports multiple types of spatial queries on MapReduce through skew-aware spatial partitioning, on-demand indexing, customizable spatial query engine RESQUE, implicit parallel spatial query execution on MapReduce, and effective methods for amending query results through handling boundary objects. To accelerate compute-intensive geometric operations, GPU based geometric computation algorithms are integrated into MapReduce pipelines. Our experiments have demonstrated that Hadoop-GIS is highly efficient and scalable, and outperforms parallel spatial DBMS for compute-intensive spatial queries.

1 Introduction

The proliferation of cost effective and ubiquitous positioning technologies has enabled the capturing of spatially oriented data at an unprecedented scale and rate. Volunteered Geographic Information (VGI) such as OpenStreetMap [1] further accelerates the generation of massive spatial information from community users. Analyzing large amounts of spatial data to derive values and guide decision making has become essential to business success and scientific discovery.

The rapid growth of spatial data has been driven by not only industrial applications, but also emerging scientific applications that are increasingly data- and compute- intensive. With the rapid improvement of data acquisition technologies, it has become more efficient to capture extremely large spatial data to support scientific research. For example, digital pathology imaging has become an emerging field in the past decade, where examination of high resolution scanned images of tissue specimens enables novel and more effective methods for disease diagnosis and therapy. Pathology image analysis offers a means of rapidly carrying out quantitative, reproducible measurements of micro-anatomical features in high-resolution images. Regions of micro-anatomic objects such as nuclei and cells are computed through image segmentation algorithms, represented with their boundaries, and image features are extracted from these objects. Exploring the results of such analysis involves complex queries such as spatial cross-matching or overlay, spatial proximity computations between objects,

and queries for global spatial pattern discovery. These queries often involve billions of spatial objects and extensive geometric computations. For example, spatial cross-matching is often used to compare and evaluate image segmentation algorithm results [14] (Figure 1). In particular, the spatial cross-matching/overlay problem involves identifying and comparing objects belonging to a wide range of different observations.

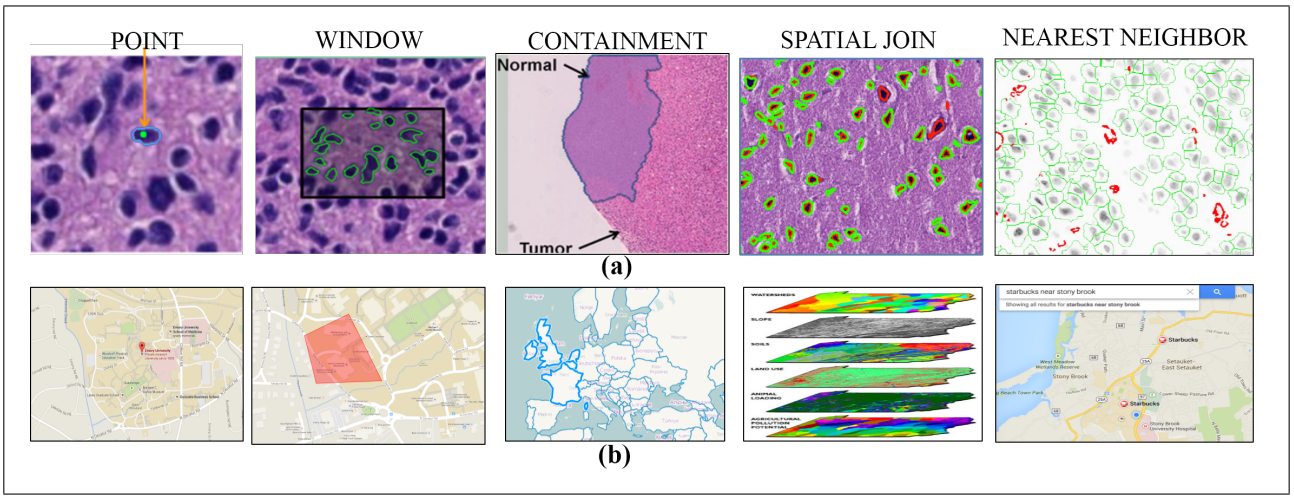

Figure 1: Examples spatial query cases. (a) pathology imaging; (b) GIS applications.

A major requirement for the data intensive spatial applications is fast query response which requires a scalable architecture that can query spatial data on a large scale. Another requirement is to support queries on a cost effective architecture such as commodity clusters or cloud environments. With the rapid improvement of instrument resolutions, increased accuracy of data analysis methods, and the massive scale of observed data, complex spatial queries have become increasingly data- and compute-intensive. A typical whole slide pathology image contains more than 100 billion pixels, millions of objects, and 100 million derived image features. A single study may involve thousands of images analyzed with dozens of algorithms - with varying parameters - to generate many different result sets to be compared and consolidated, at the scale of tens of terabytes. In addition, the aforementioned VGI also enables fast and massive geospatial data collection. Besides the data scale challenge, most spatial queries involve geometric computations that are frequently compute-intensive. While the spatial filtering using minimum bounding boxes (MBBs) can be accelerated through spatial access methods, spatial refinements such as polygon intersection verification are highly expensive operations. For instance, spatial join queries such as spatial cross-matching or spatial overlay could require significant numbers of CPU operations to process. This is mainly due to the polynomial complexity of many geometric computation methods. Such compute-intensive geometric computation, combined with the big data challenge, poses significant challenges to efficient spatial applications. There is major demand for viable spatial big data solutions from diverse fields.

Traditional spatial database management systems (SDBMSs) have been used for managing and querying spatial data, through extended spatial capabilities on top of object-relational database management systems. These systems often have major limitations on querying spatial data at massive scale, although parallel RDBMS architectures [9] are available. Parallel SDBMSs tend to reduce the I/O bottleneck through data partitioning but are not optimized for compute intensive operations such as geometric computations. Furthermore, parallel SDBMS architecture often lacks effective spatial partitioning mechanism to balance data and task loads across partitions. The high data loading overhead is another major bottleneck for SDBMS based solutions [10, 13, 14].

In contrast, MapReduce based computing model provides a highly scalable, reliable, elastic and cost effective framework for storing and processing massive data on a cluster or in cloud environment. While the MapReduce model fits amiably with large scale problems through its key-based partitioning, spatial queries and analytics are intrinsically complex and difficult to adapt into this model due to its multi-dimensional nature. Spatial partitioning poses two major problems to be handled: spatial data skew problem and boundary object

problem. The first could lead to load imbalance of tasks in distributed systems and thus result in long query response time, and the second could lead to incorrect query results if not handled properly. Furthermore, spatial query methods have to be adapted so that they can be mapped into partition based query processing framework while preserving the correct query semantics. Spatial queries are also intrinsically complex which often rely on effective access methods to reduce the search space and alleviate the high cost of geometric computations.

Meanwhile, hybrid systems combining CPUs and GPUs are becoming commonly available in commodity clusters, but the computational capacity of such systems is often underutilized. There is a general trend towards a simplified programming model such as MapReduce and hybrid computing architectures for processing massive data, but there is a significant research gap in developing new spatial querying and analytical methods to run on such architectures.

We have developed *Hadoop-GIS* [2, 3, 4, 5, 6, 12] – a spatial data warehousing system over MapReduce, to support highly scalable and efficient spatial queries and anaytics on large scale data. Hadoop-GIS provides a framework to parallelize multiple types of spatial queries and convert them into MapReduce based query pipelines. Specifically, Hadoop-GIS offers data skew aware spatial data partitioning to achieve task parallelization, an indexing-driven spatial query engine to process spatial queries, implicit query parallelization through MapReduce, and boundary object handling to generate accurate results. In addition, we have evolved GPU based spatial operators to accelerate heavy duty geometric computation, and integrated them into MapReduce based query pipelines.

2 Overview

The main objective of Hadoop-GIS is to provide a highly scalable, cost-effective, efficient spatial query processing system for data- and compute-intensive spatial applications, that can take advantage of MapReduce running on commodity clusters and CPU-GPU hybrid platforms. We first create new spatial data processing methods and pipelines with spatial partition level parallelism through the MapReduce programming model, and develop multi-level indexing methods to accelerate spatial data processing. We provide two critical components to enable such partition based parallelism by investigating effective and scalable spatial partitioning in MapReduce (pre-processing), and query normalization methods. To maximize execution performance, we fully exploit both thread-level and data-level parallelisms and utilize SIMD (Single Instruction Multiple Data) vector units to parallelize spatial operations to support object level (via grouping of many objects) and intra-object level parallelism (via breaking down an object into many smaller components), and integrate them into MapReduce pipelines. In MapReduce environment, we propose the following steps on running a typical spatial query, as shown in Algorithm 1. In step A, we effectively partition the input to generate tiles. In step B, we assign tile UIDs to spatial objects and store the objects in the Hadoop Distributed File System (HDFS). In step C we pre-process the query, and perform a preliminary filtering based on the global region index derived from the data partitioning in step A. In step D, we perform a tile based spatial query processing in which tiles run as independent MapReduce tasks in parallel. In step E, we process the boundary objects to remove duplicate objects and normalize the query result. In step F, we perform a post processing required for certain spatial query types. In step G, we perform aggregations and any additional operators, and output results to HDFS.

2.1 Real-time Spatial Query Engine

A fundamental component of Hadoop-GIS is its standalone spatial query engine. Porting a spatial database engine for such purpose is not feasible, due to its tight integration with RDBMS engine and complexity on setup and optimization. We developed a Real-time Spatial Query Engine (RESQUE) to support spatial query processing. RESQUE takes advantage of global tile indexes and local on-demand indexes to support efficient spatial queries. In addition, RESQUE is fully optimized, supports data compression, and incurs very low overhead on data loading. Thus, RESQUE is a highly efficient spatial query engine compared to a traditional SDBMS engine.

Algorithm 1: Typical workflow of spatial query processing on MapReduce

A. Data/space partitioning;
B. Data storage of partitioned data on HDFS;
C. Pre-query processing (optional);
D. **for** $tile$ **in** $input\_collection$ **do**
    Index building for objects in the tile;
    Tile based spatial querying processing;
E. Boundary object handling;
F. Post-query processing (optional);
G. Data aggregation;
H. Result storage on HDFS;

RESQUE is compiled as a shared library which can be easily deployed in a cluster environment.

Hadoop-GIS takes advantage of spatial access methods for query processing with two approaches. At the higher level, Hadoop-GIS creates global region based spatial indexes of partitioned tiles for HDFS file split filtering. Consequently, for many spatial queries such as containment queries, the system can efficiently filter most irrelevant tiles through this global region index. The global region index is small and can be stored in HDFS and shared across cluster nodes through Hadoop distributed cache mechanism. At the tile level, RESQUE supports an indexing on demand approach by building tile based spatial indexes on the fly, mainly for query processing purpose, and storing index files in the main memory. Since the tile size is relatively small, index building on a single tile is fast and significantly improves spatial query processing performance. Our experiments show that index building consumes very small fraction of overall query processing cost, and it is negligible for compute-and data-intensive queries such as cross-matching.

2.2 MapReduce Based Parallel Query Execution

Instead of using explicit spatial query parallelization as summarized in [7], we take an implicit parallelization approach by leveraging MapReduce. This will much simplify the development and management of query jobs on clusters. As data is spatially partitioned, the tile name or UID forms the key for MapReduce, and identifying spatial objects of tiles can be performed in mapping phase. Depending on the query complexity, spatial queries can be implemented as map functions, reduce functions or combination of both. Based on the query types, different query pipelines are executed in MapReduce. As many spatial queries involve high complexity geometric computations, query parallelization through MapReduce can significantly reduce query response time.

2.3 Boundary Object Handling

In the past, two approaches were proposed to handle boundary objects in a parallel query processing scenario, namely Multiple Assignment and Multiple Matching [16]. In Multiple Assignment, the partitioning step replicates boundary crossing objects and assigns them to multiple tiles. In Multiple Matching, partitioning step assigns an boundary crossing object to a single tile, but the object may appear in multiple tile pairs for spatial joins. While the Multiple Matching approach avoids storage overhead, a single tile may have to be read multiple times for query processing, which could incur increase in both computation and I/O. The Multiple Assignment approach is simple to implement with no modification to spatial computation algorithms and fits nicely to the MapReduce programming model. For example, spatial join on tiles with Multiple Assignment based partitioning can be corrected by eliminating duplicated object pairs from the query result set, which can be implemented as an additional MapReduce job [8, 16].

3 Spatial Data Partitioning

Geospatial data tends to be heavily skewed. For example, if OpenStreetMap is partitioned into 1000 x 1000 fixed size tiles, the number of objects contained in the most skewed tile is nearly three orders of magnitude more than the one in an average tile. Such large skewed tiles could significantly increase the response time in a parallel computing environment due to the straggling tiles. Thus effective and efficient spatial data partitioning is essential for scalable spatial queries running in MapReduce.

Spatial partitioning approaches generate boundary objects that cross multiple partitions, thus violating the partition independence. Spatial query processing algorithms get around the boundary problem by using a *replicate-and-filter* approach [9, 16] in which boundary objects are replicated to multiple spatial partitions, and side effects of such replication is remedied by filtering the duplicates at the end of the query processing phase. This process adds extra query processing overhead proportional to the number of boundary objects. Therefore, a good spatial partitioning approach should minimize the number of boundary objects.

We develop SATO [12], an effective and scalable partitioning framework which produces balanced regions while minimizing the number of boundary objects. The partitioning methods are designed for scalability, which can be easily parallelized for high performance. SATO stands for four main steps in the partitioning pipeline: **S**ample, **A**analyze, **T**ear, and **O**ptimize. First, a small fraction of the dataset is sampled to identify overall global data distribution with potential dense regions. Next, the sampled data is analyzed to produce a coarse partition scheme in which each partition region is expected to contain roughly equal amounts of spatial objects. Then these coarse partition regions are passed to the partitioning component that *tears* the regions into more granular partitions satisfying the partition requirements. Finally, the generated partitions are analyzed to produce multi-level partition indexes and additional partition statistics which can be used for optimizing spatial queries.

SATO integrates multiple partitioning algorithms that can handle diverse datasets, and each of the algorithm has its own merits [12]. SATO also provides MapReduce based implementation of the spatial partitioning methods through two alternative approaches: top-down approach with region level parallelization, and bottom-up approach with object level parallelization.

4 MapReduce Based Spatial Query Processing

RESQUE provides the core query engine to support spatial queries, which enables us to to develop a large scale spatial query processing framework based on MapReduce. Our approach is based on spatial data partitioning, tile based spatial query processing with MapReduce, and result normalization for tile boundary objects.

4.1 Spatial Join with MapReduce

Spatial join is among the most frequently used and costly queries in many spatial applications. Next, we discuss how to map spatial join queries into the MapReduce computing model. We first show an example spatial join query for spatial cross-matching in SQL, as shown in Figure 2. This query finds all intersecting polygon pairs between two sets of objects generated from an image by two different algorithms, and computes the overlap ratios (intersection-to-union ratios) and centroid distances of the pairs. The table *markup_polygon* represents the boundary as *polygon*, algorithm UID as *algrithm_uid*. The SQL syntax comes with spatial extensions such as spatial relationship operator *ST_INTERSECTS*, spatial object operators *ST_INTERSECTION* and *ST_UNION*, and spatial measurement functions *ST_CENTROID*, *ST_DISTANCE*, and *ST_AREA*.

For simplicity, we first present how to process the spatial join above with MapReduce ignoring boundary objects, then we return to discuss boundary handling. Input datasets are partitioned into tiles during the data loading phase, and each record is assigned a unique *partition id*. The spatial join query is implemented as a MapReduce query operator processed in following three steps: i) *Map step*: the input datasets are scanned for Map operator, and each mapper, after applying user defined function or filter operation, emits the records

```
1: SELECT
2:  ST_AREA(ST_INTERSECTION(ta.polygon,tb.polygon))/
3:    ST_AREA(ST_UNION(ta.polygon,tb.polygon)) AS ratio,
4:  ST_DISTANCE(ST_CENTROID(tb.polygon),
5:    ST_CENTROID(ta.polygon)) AS distance,
6: FROM  markup_polygon ta JOIN markup_polygon tb ON
8:    ST_INTERSECTS(ta.polygon, tb.polygon) = TRUE
9: WHERE ta.algrithm_uid='A1' AND tb.algrithm_uid='A2' ;
```

Figure 2: An example spatial join (cross-matching) query

with their *parition id* as the key along with a tag to indicate which dataset the records belong to. ii) *shuffle step*: records are sorted and shuffled to group the records having the same key (same *partition id*), and the intermediate results are materialized to local disks. iii) *Reduce step*: each reducer will be assigned to process a single partition, and a spatial join processing algorithm, such as plane-sweep join or index based join, is used to process the single partition. The join algorithm used for processing the single partition can be an in-memory or a disk based depending on the size of the partition. In addition, during the execution, Hadoop-GIS constructs an in-memory R*-Tree for each dataset in a partition, and uses those indexes to process spatial join query.

4.2 Support of other Query Types with MapReduce

Other types of spatial queries follow a similar processing pattern as shown in Algorithm 1. Spatial selection or containment is a simple query type in which objects geometrically contained in selection region are returned. For example, in a medical imaging scenario, users may be interested in the cell features which are contained in a cancerous tissue region. Since data is organized in partitions, containment queries can be processed in a *filter-and-refine* fashion. In the filter step, partitions disjoint from the query region are excluded from further processing. In the refinement step, the candidate objects are checked with the precise geometry test.

The global region index is used to generate a selective table scan operation which only scans the file splits potentially containing the query results. The query would be translated into a map only MapReduce program as shown in [6]. Support of multi-way spatial join queries and nearest neighbor queries follow a similar pattern and are discussed in [5].

For K-nearest neighbors search, Hadoop-GIS provides two algorithms for an application scenario where the query is processed over a set of query objects and the cardinality of one set of objects is much smaller than the other. For example, a query in pathology imaging would, for each stem cell, find the nearest blood vessel, compute the variation of intensity of each biological property associated with the cell in respect to the distance, and return the density distribution of blood vessels around each cell. In this case the number of cells is significantly larger than the number of blood vessels. Both algorithms [5] use a replication strategy to parallelize nearest neighbor queries. Specifically, the larger cardinality dataset is partitioned and distributed over HDFS, and mappers replicate the smaller cardinality dataset to each node. Each reducer builds an in-memory index structure, such as Voronoi diagram or R-Tree, on the smaller dataset, and processes the query over the larger dataset utilizing the index.

4.3 Boundary Handling

In partition based spatial query processing, some spatial objects may lie on partition boundaries. As the partition size gets smaller, the percentage of boundary objects increases. In general, the fraction of boundary objects is inversely proportional to the size of the partition. Boundary objects pose the challenge that they belong logically to multiple disjoint partitions and would generate duplicate results.

Hadoop-GIS remedies the boundary problem in a simple but effective way. If a query requires to return complete query result, Hadoop-GIS generates a query plan which contains a pre-processing task and a post-

processing task. In the pre-processing task, the boundary objects are duplicated and assigned to multiple intersecting partitions (multiple assignment). When each partition is processed independently during query execution, the results are not yet correct due to the duplicates. In the post-processing step, results from multiple partitions will be normalized, e.g., to eliminate duplicate records by checking the *object uid*s, which are internally assigned and globally unique. In the post-processing step, objects will go through a filtering process that eliminates duplicate records.

Intuitively, such approach would incur extra query processing cost due to the replication and duplicate elimination steps. However, this additional cost is very small and insignificant compared to the overall query processing time [6].

4.4 Performance

The RESQUE engine is highly efficient compared to traditional spatial engine [6, 5]. In particular, the on-demand R*-Tree construction cost is less than one percent of overall spatial join cost, and it does not incur any index maintenance overhead as we discard the index after processing the query. The geometric computation is the dominant cost in cross matching spatial joins. While this is difficult to support through I/O optimization oriented parallel spatial database systems, Hadoop-GIS is well adapted for such computations and outperforms parallel spatial DBMS [6]. In particular, Hadoop-GIS achieves high scalability as the on-demand spatial query engine can be easily executed in multiple parallel MapReduce tasks on cluster nodes.

5 GPU Supported Spatial Queries

GPUs employ a *SIMD* architecture that executes the same instruction logic on a large number of cores simultaneously. Many spatial algorithms and geometry computations do not naturally fit into such parallelization model. Two alternative approaches are proposed for GPU based geometric operation, in particular, polygon intersection.
Monte-Carlo Based Method. This approach uses Monte-Carlo method for *rasterization*, which transforms the combined spatial space of two polygons into pixel based representation. After such transformation, the original vector geometric computation can now be performed on the pixel based representation. The intersection area thus can be determined by counting pixels belonging to both polygons. A common approach to check if a pixel is within a polygon is to use ray tracing [11] for point-in-polygon test. As the operation for each pixel is fully independent from each other, they can be effectively executed in parallel by GPU threads [15]. Rasterization resolution is critical for achieving best performance. A high resolution rasterization yields larger number of pixels, and consequently increases the compute intensity of the geometry computations. A low resolution rasterization could increase computation efficiency, but will lead to loss of accuracy.
PixelBox. A more adaptive approach [15] — named PixelBox — can reduce the computation intensity while ensuring the computation accuracy. Specifically, PixelBox first partitions the space into cells or boxes. For boxes containing edges of polygons, rasterization is performed as Monte-Carlo approach. In this way, group of pixels in a box could be tested together for the containment relationship with a polygon, and pixel level testing is performed only for edge crossing areas. Thus, the computational efficiency could be much improved. The experiments demonstrate two orders performance improvement for intersection operation compared to a single thread CPU algorithm.
Integration of GPU Based Geometric Computation into MapReduce. To support a more efficient execution on accelerated systems, we have been extending Hadoop-GIS for execution of spatial query operations with GPUs in distributed memory machines. The goal is to design an efficient bridge interface between the MapReduce program and the GPU program. Many small tasks sent to GPU may incur much overhead on communication and subdue the benefit of GPU. We propose a prediction model to decide the granularity of tasks for GPU invocation, by considering both data communication cost and execution cost for different types of spatial operations. Another goal is to achieve load balancing and data/operation aware task assignment in

the CPU/GPU hybrid environment. We first take a knowledge based approach to decide assignments to CPU or GPU, and then build efficient task migration between the CPU and the GPU in case of an unbalanced task assignment. Preliminary work is reported in [4].

6 Software

The high adaptability of the framework allows the system to be integrated into computer clusters or cloud computing environments such as Amazon EC2. Hadoop-GIS is available as a set of library functions, including input data transformation, data partitioning, spatial indexing and spatial query processing, and MapReduce based execution. It also includes pipelines for combined multiple query jobs. We implemented the core spatial indexing and querying methods in C++, and implemented the MapReduce programs in Java. We use the Hadoop streaming mechanism to bridge the communication between C++ libraries and Java based MapReduce programs. The pipelines are as set of scripts which can be easily customized. Hadoop-GIS is open source, and it can be downloaded from the web site [2].

Acknowledgments

This work is supported in part by NSF IIS 1350885, by NSF ACI 1443054, by NLM R01LM009239, and by NCI 1U24CA180924-01A1.

References

[1] Open Street Map: `http://www.openstreetmap.org`.
[2] Hadoop-GIS: `http://hadoopgis.org`.
[3] A. Aji, X. Sun, H. Vo, Q. Liu, R. Lee, X. Zhang, J. Saltz, and F. Wang. Demonstration of hadoop-gis: A spatial data warehousing system over mapreduce. In *SIGSPATIAL/GIS*, pages 518–521. ACM, 2013.
[4] A. Aji, G. Teodoro, and F. Wang. Haggis: Turbocharge a mapreduce based spatial data warehousing system with gpu engine. In *BigSpatial '14*, pages 15–20, 2014.
[5] A. Aji, F. Wang, and J. H. Saltz. Towards Building A High Performance Spatial Query System for Large Scale Medical Imaging Data. In *SIGSPATIAL/GIS*, pages 309–318. ACM, 2012.
[6] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *Proc. VLDB Endow.*, 6(11):1009–1020, Aug. 2013.
[7] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Parallel processing of spatial joins using r-trees. In *ICDE*, 1996.
[8] M.-L. Lo and C. V. Ravishankar. Spatial hash-joins. In *SIGMOD*, pages 247–258, 1996.
[9] J. Patel et al. Building a scaleable geo-spatial dbms: technology, implementation, and evaluation. In *SIGMOD*, pages 336–347, 1997.
[10] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker. A comparison of approaches to large-scale data analysis. In *SIGMOD*, pages 165–178, 2009.
[11] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *ACM TOG*, volume 21, pages 703–712. ACM, 2002.
[12] H. Vo, A. Aji, and F. Wang. Sato: A spatial data partitioning framework for scalable query processing. In *SIGSPATIAL/GIS*. ACM, 2014.
[13] F. Wang, J. Kong, L. Cooper, T. Pan, K. Tahsin, W. Chen, A. Sharma, C. Niedermayr, T. W. Oh, D. Brat, A. B. Farris, D. Foran, and J. Saltz. A data model and database for high-resolution pathology analytical image informatics. *J Pathol Inform*, 2(1):32, 2011.
[14] F. Wang, J. Kong, J. Gao, D. Adler, L. Cooper, C. Vergara-Niedermayr, Z. Zhou, B. Katigbak, T. Kurc, D. Brat, and J. Saltz. A high-performance spatial database based approach for pathology imaging algorithm evaluation. *J Pathol Inform*, 4(5), 2013.
[15] K. Wang, Y. Huai, R. Lee, F. Wang, X. Zhang, and J. H. Saltz. Accelerating pathology image data cross-comparison on cpu-gpu hybrid systems. *Proc. VLDB Endow.*, 5(11):1543–1554, 2012.
[16] X. Zhou, D. J. Abel, and D. Truffet. Data partitioning for parallel spatial join processing. *GeoInformatica*, 2(2):175–204, 1998.