Experimental Study of Big Raster and Vector Database Systems

Samriddhi Singla [*], Ahmed Eldawy [*], Tina Diao [†], Ayan Mukhopadhyay [‡], Elia Scudiero [§]
[*] Computer Science and Engineering, [§] Environmental Sciences
University of California, Riverside
{ssing068, eldawy, elias}@ucr.edu
[†] Management Science & Engineering [‡] Aeronautics and Astronautics
Stanford University
{tdiao, ayanmukh}@stanford.edu

***Abstract*—Spatial data is traditionally represented using two data models, raster and vector. Raster data refers to satellite imagery while vector data includes GPS data, Tweets, and regional boundaries. While there are many real-world applications that need to process both raster and vector data concurrently, state-of-the-art systems are limited to processing one of these two representations while converting the other one which limits their scalability. This paper draws the attention of the research community to the research problems that emerge from the concurrent processing of raster and vector data. It describes three real-world applications and explains their computation and access patterns for raster and vector data. Additionally, it runs an extensive experimental evaluation using state-of-the-art big spatial data systems with raster data of up-to a trillion pixels, and vector data with up-to hundreds of millions of edges. The results show that while most systems can analyze raster and vector concurrently, but they have limited scalability for large-scale data.**

I. Introduction

The recent decade has seen an explosive increase in the amount of spatial data. The advancement in remote sensing technology has led to the availability of petabytes of satellite imagery. In the meantime, the proliferation of smart devices and GPS technology has led to highly accurate geographical features such as water bodies, city boundaries, roads, agricultural fields, and others. Spatial data can generally be modeled in two representations: raster and vector. Satellite imagery is an example of raster data and is usually represented in form of multi-dimensional arrays. Vector data is represented as a set of points, lines, and polygons, and is used to represent geographical features such as GPS locations, regional boundaries, and roads.

Domain scientists are using both raster and vector data, concurrently, in various interesting research discoveries and applications such as areal interpolation [1], wildfire risk assessment [2], [3], the effect of vegetation and temperature on human settlement [4], [5], analysis of terabytes of socio-economic and environmental data [6], [7], and land use classification [8]. Through our collaboration with domain scientists, we were surprised that most of them ignore big spatial data systems and use hand-crafted Python scripts or out-of-the-box GIS systems.

In this paper, we study three real applications and use them as a benchmark to evaluate existing big spatial data systems, namely, combating wildfires, crop yield mapping, and areal interpolation. By analyzing these applications and existing systems, we found that the main limitation is the concurrent processing of *raster* and *vector* data, e.g., compute average temperature per city or maximum vegetation per agricultural field. Most existing systems are optimized to process one spatial data representation. On one hand, raster-based systems use the array data model to store and query the data, e.g., SciDB [9], RasDaMan [10], GeoTrellis [11], ChronosDB [12], and Google Earth Engine [13]. To process vector data, raster systems need to rasterize vector data into raster layers with the same resolution of the raster data which results in a significant increase in the data size as the resolution increases. On the other hand, vector based systems, e.g., SpatialHadoop [14], Sedona (formerly GeoSpark) [15], and Simba [16], use a relational data model and relational algebra to process vector data represented as points, lines, and polygons. When it comes to raster data, it needs to convert each pixel to a point to combine it with vector data which, again, increases the data size significantly for high-resolution data. Additionally, there have been recent efforts to develop algorithms that combine raster and vector data efficiently for specific problems such as zonal statistics [17]–[19].

In this paper, we study and evaluate the state-of-art systems based on how they process the combination of raster and vector data. First, we classify existing systems in one of three categories, raster, vector, and rater+vector. Then, we compare them based on their (1) loading phase, that is the time taken to ingest data, and (2) running time. We use large scale satellite data with up-to a trillion pixels, and big vector data with up-to hundreds of millions of edges to compare Raptor Zonal Statistics [19], Beast [20], Sedona [15], Adaptive Cell Trie (ACT) [21], GeoTrellis [11], Rasdaman [10] and Google Earth Engine [13]. We also describe the computation of three real world applications that need to concurrently process raster and vector data.

The rest of this paper is organized as follows: Section II

This work is partly funded by USDA NIFA AFRI (Grant Number: 2019-67022-29696)

covers the related work in literature. Section III discusses three applications that process the combination of raster and vector data. Section IV describes the systems that we study in this paper. Section V provides a comparison of the described systems. Section VI concludes the paper and discusses future work.

II. Related Work

A. Big Raster Data Processing

Raster data is usually represented as multidimensional arrays and it is analysed using operations [22] such as map algebra and map overlay. Map algebra can be described as a raster-raster join operation that combines two or more multidimensional arrays to produce another multidimensional array possibly of a different size. It may join a pixel in one raster layer to many pixels in other raster layers. On the other hand, a map overlay operation often joins a pixel in one raster layer to a pixel in another raster layer based on their locations. If the two raster layers do not have matching resolutions, a regridding operation is applied on one dataset to match the other dataset. Systems such as SciDB [9], RasDaMan [10], GeoTrellis [11], ChronosDB [12], and Google Earth Engine [13] implement algorithms for raster operations that can process large amounts of raster data. However, none of these systems can directly join raster and vector data. Rather, they usually rasterize the vector data with a matching resolution and apply the map overlay operation.

B. Big Vector Data Processing

Vector data is represented as a set of points, lines, and polygons, which are all represented as a set of coordinates. Vector spatial data systems, e.g., PostGIS [23], extend the relational data model with a geometry data type that can store one of the above geometries. Further, it adds new operations, such as spatial join, and indexes, such as R-tree, to facilitate the processing of this data. The same approach is adopted for big data systems such as SpatialHadoop [14], Hadoop-GIS [24], Sedona [15], and Simba [16]. To bring in raster data, these systems convert each pixel to a record with a point location and pixel value. Once both data sets are in the same format, an appropriate spatial join algorithm can be used such as R-Tree join [25], Spatial Hash join [26], Partition Based Spatial Merge join (PBSM) [27], and many more [21], [28]–[30]. Efforts have been made to port some of these algorithms to distributed systems to process big data [14], [31]–[33]. Unfortunately, the data size increases dramatically in the conversion process and hence the spatial join operation does not scale.

C. Big Raster + Vector Data Processing

An early work on combining raster and vector data processing [34] proposed a hybrid data structure to store both raster and vector data. It requires an offline preprocessing step that converts both datasets to an intermediate form before it performs any processing. Another work on querying raster and vector data [35], [36] focuses on compact representation of raster in memory using a new tree-like data structure and performing range queries with vector data represented by an R-tree. Scanline algorithm [17], [37] was a first step in efficiently processing the zonal statistics problem by combining vector and raster data but it was limited to a single machine. A followup work [19] extended that algorithm to a fully distributed algorithm that runs on Hadoop.

This paper is the first effort to extensively evaluate all the above system types for real scientific applications that require the concurrent processing of vector and raster data.

III. Applications

In this section, we discuss three real-world applications that process the combination of raster and vector data.

A. Combating Wildfires

Wildfire is a natural disaster which causes massive damage to property and human life. It is a recurring phenomenon, especially in North America, which has led to research in ways to prevent, detect, and combat the spread of wildfires. In the current wildfire season in California so far, more than four million acres have already burned due to more than 8,000 wildfires. At one point in August 2020, the entire northern half of the state had been instructed to prepare for evacuation. This has made it crucial to model the spread of wildfire and predict how to efficiently allocate resources to minimize loss of life and property.

Data-driven modelling of wildfire spread needs to combine the information about occurrences of fire with their corresponding geographical factors [3], [38]. The occurrences of $fire$ are available in vector format as a collection of geographical points where the fire occurred. Each point also has a variety of attributes associated with it such as the timestamp of fire, its intensity, etc. Geographical factors such as vegetation, elevation, wind direction and fuel levels, that affect the duration and direction of wildfire spread are available as rasters in form of satellite imagery. Apart from these two datasets, data-driven modelling also requires to divide the target geographic region into numerous polygons called $zones$, which are available as polygons in vector format. At the core of this application, the fire $zones$ need to be joined with several raster layers to calculate various statistics such as mean, median, standard deviation, min, and max, of contributing geographical factors for each zone. The computed statistics may or may not have associative and commutative properties, e.g., median is not associative or commutative. After that, it joins the result with the $fire$ dataset and utilizes machine learning to build a model for wildfire spread [3], [38], [39].

B. Crop Yield Mapping

The problem of crop yield mapping in agriculture [40] studies the crop yield of various agriculture fields using Normalized Difference Vegetation Index (NDVI), which can be used as a proxy for crop health, growth status, and yield. NDVI is calculated using the red and near-infrared spectral reflectance (SR) measurements captured by satellites which are available as rasters. To study crop yield, NDVI needs to be calculated for all pixels that overlap the agricultural fields under study for a period of time ranging over multiple years. The agricultural fields are available in vector format as a set of polygons, where each polygon represents one agricultural field. Since crop yield is affected by various environmental conditions, it makes it necessary to study various statistics about the crop yield both in-field and across all the fields under study. These statistics also need to be calculated across different windows of time to make sure whether the factors contributing towards decrease in crop yield are local or global (such as drought).

This application starts by sorting the NDVI layers by time to logically form a three-dimensional cube where each pixel contains a time series of NDVI values in one year. Then it calculates the standard deviation of each time series which represents the temporal variability in vegetation in that location. After than, it combines this result with the agricultural fields, represented as polygons, and computes the average and 90^{th} percentile value per field. This whole process is repeated for each year in the study and these values are then used to classify the agriculture fields according to their crop yield and stability over the years.

C. Areal Interpolation

Areal Interpolation is the problem of estimating a function in arbitrary areas, e.g., city boundaries, based on values in other non-aligned areas, e.g., census tracts. One application of this problem is to estimate the population of arbitrary regions using landcover data [1]. The problem is that the US Census Bureau reports the population at the granularity of *census tracts* which are regions chosen by the Bureau to keep the privacy of the data. Areal interpolation transforms these counts from source polygons, i.e., tracts, to target polygons, e.g., ZIP Codes, with unknown counts. One accurate method [1] uses the National Land Cover Database (NLCD) [41] raster dataset as a reference to disaggregate the population counts into pixels and then aggregate them back into target polygons.

This application starts by calculating the histogram of NLCD values of each known region, e.g., census tract. This step estimates how much of each region is covered by each land type, e.g., road, urban area, and water. After that, it uses Poisson regression to estimate the contributing factor of each land type to the true population in these regions. This is called the disaggregation step since it breaks down the population into pixels. In the next step, it processes the unknown regions, e.g., ZIP codes, with the NLCD dataset to compute the histogram of each known region. Finally, it uses the regression parameters to estimate the population of each known region. The most time consuming step in this process is to calculate the histogram of the (raster) NLCD dataset for each (vector) region.

IV. Systems

In this section, we describe the state-of-art systems that belong to either of the three categories: raster-based, vector-based and raster + vector based.

A. Raster-Based Systems

Raster-based systems represent images as multidimensional arrays, which can typically contain trillions of entries. Most standard file structures, e.g., GeoTIFF and HDF5, partition this large matrix into smaller equi-sized *tiles*, where each tile is stored as one block and is typically small enough to load entirely in main memory. The query processing model for these systems is based on linear algebra. Below we describe the three raster-based systems that we study in this paper.

GeoTrellis: It is a Scala library and framework that uses Apache Spark to work with raster data. It implements many Map Algebra operations as well as vector to raster or raster to vector operations. It stores raster data as a RDD (Resilient Distributed Datasets) of key-value pairs, where each tile is a value and the key is its unique identifier. It does not require a separate loading phase to ingest data and can work with data in HDFS(Hadoop Distributed File System). It does not require vector data to be rasterized before ingestion and can do so on-the-fly while processing queries.

Google Earth Engine (GEE): It is a cloud-based platform for planetary-scale geospatial analysis backed by Google Cloud Engine. It implements various raster-based algorithms that can be used to solved real world problems and also provides limited functionality with vector data. Though, it is open to public and free to use, there is no information about how it implements its algorithms or on how many machines these algorithms run. Generally, the number of these machines can vary from a few hundreds to thousands [13]. Although it provides a public data repository, the user needs to upload data that is not available in this repository, using a web interface, which is a time-consuming and cumbersome process. It also imposes a 10 GB restrictions on the size of vector data that can be uploaded.

Rasdaman: It is developed using C++ and Java, and uses an array data model to store raster data. It is available as a distributed version for commercial use but the public version only works on a single machine. It suffers from the overhead of an ingestion phase where it restructures the raster layer according to its array data model. It provides limited support for vector data and implements few algorithms, accessible through its web interface, that can work with only a single geometry at a time. Otherwise, it would require the user to rasterize the vector data and ingest it into the system before queries can be performed. It has the advantage of providing a SQL-like query language which makes it easier to use.

B. Vector-Based Systems

Below we describe two vector-based systems that we study in this paper.

Sedona: It is a cluster computing framework that can process vector data at scale. Its core component is a SpatialRDD that consists of data partitions that are distributed across the Spark cluster. It uses this SpatialRDD to store and process vector data. It has the advantage of processing data in-situ from HDFS, however, it can only work with vector data. This makes it necessary for the user to vectorize raster data before it can be processed. It implements various partitioning and indexing techniques to make query processing more efficient. Users can easily query data in Sedona either using SpatialRDDs directly or its SQL interface.

Beast: It is another Spark-based system for Big Exploratory Analytics on Spatio-Temporal data [20]. It extends the Spark RDD API by adding a geometry data type, spatio-temporal input formats, multidimensional indexes, query processing, and visualization. In this evaluation, we use Beast since it provides an optimized spatial join algorithm that is useful for this problem.

Adaptive Cell Trie (ACT): ACT is an efficient in-memory index that can be used to process point-in-polygon queries. This index is built for the vector data and minimizes the amount of computationally expensive point-in-polygon queries that need to be run. Though the original implementation of this system does not work with raster data, we use GDAL library to load pixels from the raster file, convert each one to a point, and search for overlapping polygons in the index.

C. Raster + Vector Based Systems

We study *Raptor Zonal Statistics (RZS)* [19] as a candidate for systems that process raster+vector data. It is a fully distributed system implemented in Hadoop using the MapReduce paradigm and can process data in-situ. It can ingest rasters in form of GeoTiff and HDF5 formats, and vectors in form of shapefiles, WKT and CSV. It can work with data in their native formats by computing an intermediate data structure *intersection file*, which defines a mapping between raster and vector data. However, its limitation is that it can only perform the zonal statistics operation on raster+vector data.

V. EXPERIMENTS

This section provides an extensive experimental evaluation of Google Earth Engine (GEE) [13], GeoTrellis [11], Rasdaman [10], Adaptive Cell Trie (ACT) [21], Sedona [15], Beast [20], and Raptor Zonal Statistics (RZS) [19].

Section V-A describes the setup of the experiments, the system and the datasets used. Section V-B compares the systems in section IV based on the total running time. Section V-C discusses the vector and raster dataset ingestion time for each of these methods. Section V-D discusses how RZS and GEE was used to implement the applications discussed earlier.

A. Setup

We run Sedona, GeoTrellis, Beast and RZS on a cluster with one head node and 12 worker nodes. The head node has Intel(R) Xeon(R) CPU $E5-2609$ v4 @ 1.70GHz processor, 128 of GB RAM, 2 TB of HDD, and 2×8-core processors running CentOS and Oracle Java 1.8.0_131. The worker nodes have Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz processor, 64 GB of RAM, 10 TB of HDD, and 2×6-core processors running CentOS and Oracle Java 1.8.0_31-b04. RZS and Beast are implemented using the open source GeoTools library 17.0. GEE runs on the Google Cloud Platform on up-to 1,000 nodes [13] but it does not make available the actual resources used by each query. Rasdaman and ACT are run on a single machine with Intel(R) Core $i5-6500$ CPU @ 3.20GHz × 4, 32 of GB RAM, 1 TB of HDD running Ubuntu 16.04.

TABLE I: Vector and Raster Datasets

Vector datasets

Dataset	$\lvert V\rvert$	Points	File Size	Type	Coverage
Counties	3k	52k	978 KB	Polygons	US
States	49	165k	2.6 MB	Polygons	US
Boundaries	284	3.8m	60 MB	Polygons	World
TRACT	74k	38m	632 MB	Polygons	US
Parks	10m	336m	8.5 GB	Polygons	World

Raster datasets

Dataset	Image Size	File Size	Coverage
glc2000	$40,320 \times 16,353$	629 MB	World
MERIS	$129,600 \times 64,800$	7.8 GB	World
US-Aster	$208,136 \times 89,662$	35 GB	US
Tree cover	$1,296,036 \times 648,018$	782 GB	World

We test these systems for the zonal statistics problem and perform a raster-vector join to compute the four aggregate values, minimum, maximum, sum, and count for the resulting tuples. We measure the end-to-end running time as well as the performance metrics which include reading both datasets from disk and producing the final answer. Table I lists the datasets that are used in the experiments along with their attributes. All these datasets are publicly available with the vector datasets being available on UCR-Star [42], [43]. More information about these datasets can be found in [18].

We used (1) Rasdaman version 10.0, single machine implementation as the distributed version is not publicly available; (2) *geotrellis-spark* version 1.2.1, as described in its documentation; (3) Sedona 1.3.2-SNAPSHOT and followed the official documentation; (4) Beast 0.9.1 with the default parameters; and (5) Google Earth Engine (GEE) which runs on Google Cloud Engine. Due to the lack of transparency on how jobs are queued and executed in GEE, we run each operation 3-5 times at different times and report the average to account for any variability in the load. All the running times are collected as reported by GEE in the dashboard.

B. Execution Time

This section compares the systems based on their end-to-end execution time for the zonal statistics problem.

Vector-based Systems: We first compare the vector-based systems with RZS. This experiment is run for the smaller raster datasets, GLC2000 and MERIS with all the vector datasets. As can be observed in Figure 1, Sedona and ACT are not able to scale for all vector datasets even with smaller raster datasets. Even though Beast is able to scale for smaller raster datasets, RZS achieves an order of magnitude performance

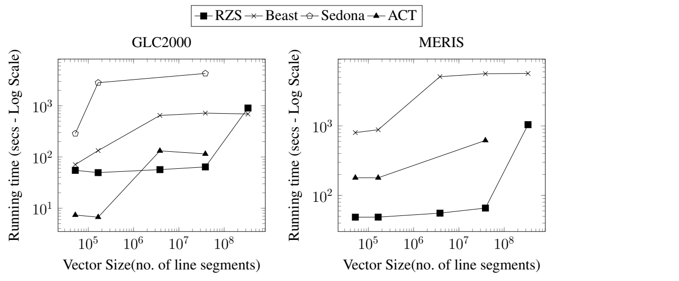


Fig. 1: Comparison of total running time for Beast, Sedona, ACT, and RZS.

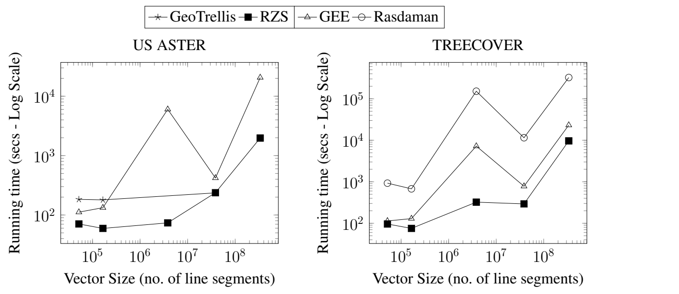


Fig. 2: Comparison of total running time of Rasdaman, GeoTrellis, Google Earth Engine, and RZS

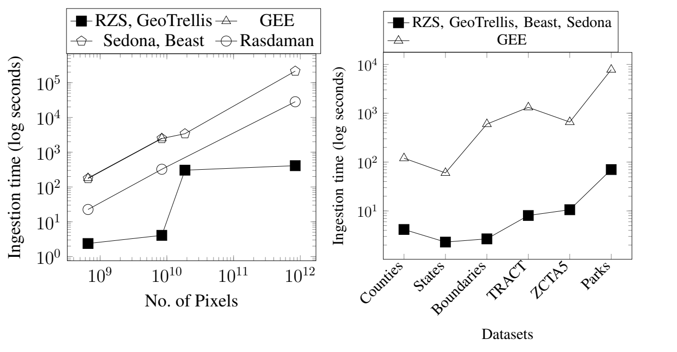
gain over it. We omit results for bigger raster datasets since the vector-based systems were not able to scale to them.

Raster-based Systems: We now compare the raster-based systems to RZS, using the larger raster datasets, US Aster and Treecover. Figure 2 shows the results for this experiment. It can be observed that RZS and GEE are able to scale for larger datasets with comparable performance. Geotrellis is not able to scale for the Treecover dataset since it suffers from the limitation of rasterizing vector data on-the-fly while Rasdaman fails to ingest and structure US Aster dataset according to its data model.

C. Ingestion Time

Figure 3 shows the ingestion time of the raster and vector datasets for GeoTrellis, Rasdaman, GEE, Sedona, Beast and RZS. RZS, GeoTrellis, Beast and Sedona require both raster and vector datasets to be loaded into HDFS. However, Beast and Sedona require raster data to be vectorized before loading it into HDFS. Rasdaman only ingests raster data while GEE requires both raster and vector data to be uploaded to its web interface. We do not upload US Aster and Treecover to GEE as they are available in its data repository, while Rasdaman was not able to ingest US Aster. We do not show ingestion time for ACT as it is not disk-optimized.

RZS and GeoTrellis is two to three orders of magnitude faster than others for ingesting raster data as shown in Figure 3a. For vector data, all the syatems are one to three orders of magnitude faster than GEE as can be observed from



(a) Raster Dataset (b) Vector Dataset

Fig. 3: Ingestion Time

Figure 3b. The reason for that is that RZS, GeoTrellis, Beast and Sedona implement an in-situ approach.

D. Applications

This section shows how we implemented the applications discussed in Section III. The first application for *combating wildfires* as implemented in [38] requires to join over 3 million polygons with 23 rasters from *landfire.gov* each containing over a billion pixels, using the predicate $\theta_{polygon}$. It computes statistics such as max, min, sum, count, mean, mode and median to be calculated for combination of each raster and each polygon. It then requires to be spatially joined with *VIIRS* fire dataset containing 4 million points representing actual fire locations. The total time taken to calculate statistics and perform the spatial join using RZS was about 2 hours while it took GEE 17 days.

For the second application of *crop yield mapping*, we calculated the NDVI for 360k agricultural fields in California using USGS Landsat 8 ARD surface reflectance images. There were 20,000 images amounting to about 1.8 TB, each for the red and NIR bands (required to compute NDVI) ranging over the temporal period of 6 years from 2014 - 2019, each having a resolution of 5000 $\times$ 5000 pixels per file. We computed NDVI for each pixel in each agricultural field and computed the standard deviation for the resulting time series of NDVI values per pixel. RZS algorithm was able to complete this in 3 hours while it took GEE over 30 hours to do the same.

The third application of *areal interpolation* required to join the National Land Cover (NLCD) raster dataset, of resolution 161,190 $\times$ 104,424, with 74k TRACT polygons to estimate their population. Since, the baseline used by the developers of this application was a vector-based Python implementation, we tested RZS on single machine as well. The entire process completed in 10 seconds for the state of Pennsylvania while the Python-based script took over 100 minutes to complete. Given that impressive speedup, the authors of that work were able to scale their work to the entire US which took under 2 hours on a single machine.

VI. Conclusion and Future Work

The paper studies and evaluates state-of-art systems based on how they process queries that require concurrent processing of raster and vector data. These systems can be classified in three categories based on the data model they use to implement algorithms: raster-based, vector-based and raster+vector based. Raster-based systems use the array data model to represent raster data and therefore require vector data to be rasterized before they can process it. Similarly, vector-based systems require raster data to be vectorized before processing since they use the relational data model. On the other hand, most raster+vector based systems do not require to convert data from one from to another but compute an index or intermediate data structure to facilitate query processing. However, they are limited to only particular queries. This paper experimentally evaluates three raster-based systems (Google Earth Engine, Rasdaman, and GeoTrellis), three vector-based systems (Adaptive Cell Trie, Sedona, and Beast) and one raster+vector based system (Raptor Zonal Statistics). It shows that raster+vector based systems are more efficient at processing queries involving both raster and vector data with the raster-based system Google Earth Engine being a close competitor. However, other raster-based and vector-based systems fail to scale for big raster+vector data.

References

[1] M. Reibel and A. Agrawal, "Areal interpolation of population counts using pre-classified land cover data," *Population Research and Policy Review*, vol. 26, no. 5-6, pp. 619–633, 2007.

[2] M. B. Joseph *et al.*, "Spatiotemporal prediction of wildfire size extremes with bayesian finite sample maxima," *Ecological Applications*, vol. 29, no. 6, 2019.

[3] O. Ghorbanzadeh *et al.*, "Spatial prediction of wildfire susceptibility using field survey gps data and machine learning approaches," *Fire*, vol. 2, no. 3, p. 43, 2019.

[4] G. D. Jenerette *et al.*, "Regional Relationships Between Surface Temperature, Vegetation, and Human Settlement in a Rapidly Urbanizing Ecosystem," *Landscape Ecology*, vol. 22, pp. 353–365, 2007.

[5] ——, "Ecosystem Services and Urban Heat Riskscape Moderation: Water, Green Spaces, and Social Inequality in Phoenix, USA," *Ecological Applications*, vol. 21, pp. 2637–2651, 2011.

[6] D. Haynes, S. Manson, and E. Shook, "Terra Populus' Architecture for Integrated Big Gepspatial Services," *Transactions on GIS*, 2017.

[7] D. Haynes, S. Ray, S. M. Manson, and A. Soni, "High Performance Analysis of Big Spatial Data," in *Big Data*, Santa Clara, CA, Nov. 2015, pp. 1953–1957.

[8] H. Saadat *et al.*, "Land use and land cover classification over a large area in iran based on single date analysis of satellite imagery," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 66, no. 5, pp. 608–619, 2011.

[9] M. Stonebraker *et al.*, "SciDB: A Database Management System for Applications with Complex Analytics," *Computing in Science and Engineering*, vol. 15, no. 3, pp. 54–62, 2013.

[10] P. Baumann *et al.*, "The multidimensional database system rasdaman," in *SIGMOD*, Seattle, WA, Jun. 1998, pp. 575–577.

[11] A. Kini and R. Emanuele, "Geotrellis: Adding Geospatial Capabilities to Spark," 2014.

[12] R. A. R. Zalipynis, "Chronosdb: distributed, file based, geospatial array dbms," *PVLDB*, vol. 11, no. 10, pp. 1247–1261, 2018.

[13] N. Gorelick *et al.*, "Google earth engine: Planetary-scale geospatial analysis for everyone," *Remote sensing of Environment*, vol. 202, pp. 18–27, 2017.

[14] A. Eldawy and M. F. Mokbel, "SpatialHadoop: A MapReduce Framework for Spatial Data," in *ICDE*, Apr. 2015, pp. 1352–1363.

[15] J. Yu, M. Sarwat, and J. Wu, "GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data," in *SIGSPATIAL*, Seattle, WA, Nov. 2015, pp. 70:1–70:4.

[16] D. Xie *et al.*, "Simba: Efficient In-Memory Spatial Analytics," in *SIGMOD*, Jun. 2016.

[17] A. Eldawy, L. Niu, D. Haynes, and Z. Su, "Large scale analytics of vector+raster big spatial data," in *SIGSPATIAL*, 2017, pp. 62:1–62:4.

[18] S. Singla and A. Eldawy, "(Poster) Distributed Zonal Statistics of Big Raster and Vector Data," in *SIGSPATIAL*, 2018.

[19] ——, "Raptor Zonal Statistics : Fully Distributed Zonal Statistics of Big Raster + Vector Data," in *Proceedings of the 2020 IEEE International Conference on Big Data (IEEE BigData 2020)*. IEEE, Dec. 2020.

[20] Y. Zhang and A. Eldawy, "Evaluating computational geometry libraries for big spatial data exploration," in *Proceedings of the Sixth International ACM SIGMOD Workshop on Managing and Mining Enriched Geo-Spatial Data*, 2020, pp. 1–6.

[21] A. Kipf, H. Lang, V. Pandey, R. A. Persa, C. Anneser, E. T. Zacharatou, H. Doraiswamy, P. A. Boncz, T. Neumann, and A. Kemper, "Adaptive main-memory indexing for high-performance point-polygon joins." in *EDBT*, 2020, pp. 347–358.

[22] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*. Prentice Hall Upper Saddle River, NJ, 2003.

[23] "Postgis: Spatial and geographic objects for postgresql," 2020, https://postgis.net.

[24] A. Aji *et al.*, "Hadoop-gis: A high performance spatial data warehousing system over mapreduce," *PVLDB*, vol. 6, no. 11, pp. 1009–1020, 2013.

[25] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient processing of spatial joins using r-trees," *ACM SIGMOD Record*, vol. 22, no. 2, pp. 237–246, 1993.

[26] M.-L. Lo and C. V. Ravishankar, "Spatial hash-joins," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, 1996, pp. 247–258.

[27] J. M. Patel and D. J. DeWitt, "Partition based spatial-merge join," *ACM Sigmod Record*, vol. 25, no. 2, pp. 259–270, 1996.

[28] M.-L. Lo and C. V. Ravishankar, "Spatial joins using seeded trees," in *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, 1994, pp. 209–220.

[29] N. Koudas and K. C. Sevcik, "Size separation spatial join," in *SIGMOD*, 1997, pp. 324–335.

[30] S. Ray *et al.*, "Skew-resistant Parallel In-memory Spatial Join," in *SSDBM*, Aalborg, Denmark, Jul. 2014, pp. 6:1–6:12.

[31] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu, "Sjmr: Parallelizing spatial join with mapreduce on clusters," in *CLUSTER*, 2009, pp. 1–8.

[32] I. Sabek and M. F. Mokbel, "On spatial joins in mapreduce," in *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2017, pp. 1–10.

[33] A. Belussi, S. Migliorini, and A. Eldawy, "Cost estimation of spatial join in spatialhadoop," *GeoInformatica*, vol. 24, no. 4, pp. 1021–1059, 2020. [Online]. Available: https://doi.org/10.1007/s10707-020-00414-x

[34] D. J. Peuquet, "A hybrid structure for the storage and manipulation of very large spatial data sets," *Computer Vision, Graphics, and Image Processing*, vol. 24, no. 1, pp. 14–27, 1983.

[35] N. R. Brisaboa *et al.*, "Efficiently querying vector and raster data," *The Computer Journal*, vol. 60, no. 9, pp. 1395–1413, 2017.

[36] F. Silva-Coira *et al.*, "Efficient processing of raster and vector data," *Plos one*, vol. 15, no. 1, p. e0226943, 2020.

[37] S. Singla *et al.*, "Raptor: Large Scale Analysis of Big Raster and Vector Data," *PVLDB*, vol. 12, no. 12, pp. 1950 – 1953, 2019.

[38] T. Diao *et al.*, "Uncertainty aware wildfire management," in *AI for Social Good Workshop, AAAI Fall Symposium Series*, 2020.

[39] S. Singla *et al.*, "WildfireDB: A Spatio-Temporal Dataset Combining Wildfire Occurrence with Relevant Covariates," 2020.

[40] B. Maestrini and B. Basso, "Predicting spatial patterns of within-field crop yield variability," *Field Crops Research*, vol. 219, 2018.

[41] "NLCD dataset," https://www.mrlc.gov/data/type/land-cover, 2020.

[42] S. Ghosh, T. Vu, M. A. Eskandari, and A. Eldawy, "UCR-STAR: The UCR Spatio-Temporal Active Repository," *SIGSPATIAL Special*, vol. 11, no. 2, p. 34–40, Dec. 2019.

[43] "UCR-Star: The UCR Spatio-temporal Active Repository." [Online]. Available: https://star.cs.ucr.edu/