THESIS

TOWARD EFFECTIVE HIGH-THROUGHPUT GEOREFERENCING OVER VOLUMINOUS OBSERVATIONAL DATA IN THE DOMAIN OF PRECISION AGRICULTURE

Submitted by

Maxwell L. Roselius

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2018

Master's Committee:

Advisor: Sangmi Lee Pallickara

Shrideep Pallickara
John McKay

ABSTRACT

TOWARD EFFECTIVE HIGH-THROUGHPUT GEOREFERENCING OVER VOLUMINOUS OBSERVATIONAL DATA IN THE DOMAIN OF PRECISION AGRICULTURE

Remote sensing of plant traits and their environment facilitates non-invasive, high-throughput monitoring of the plant's physiological characteristics. Effective ingestion of these sensing data into a storage subsystem while georeferencing phenotyping setups is key to providing timely access to scientists and modelers. In this thesis, we propose a high-throughput distributed data ingestion framework with support for fine-grained georeferencing. The methodology includes a novel spatial indexing scheme, the nested hash grid, for fine-grained georeferencing of data while conserving memory footprints and ensuring acceptable latency. We include empirical evaluations performed on a commodity machine cluster with up to 1TB of data. The benchmarks demonstrate the efficacy of our approach.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Sangmi Lee Pallickara for her constant support and unrivaled expertise during this project. This work would not have been possible without her. I would also like to extend my gratitude to Dr. Shrideep Pallickara and Dr. John McKay, who both provided excellent feedback during many stages of this work. Additional thanks to ARPA-E, the Department of Homeland Security, and the National Science Foundation who provided funding for this project.

Much of the material in this thesis was presented in [1], which was authored by the author, and co-authored by Dr. Sangmi Pallickara.

This document is typeset in LaTeX using a document class designed by Leif Anderson and modified by Elliot Forney.

DEDICATION

To my late grandfather, Ted Roselius

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: Introduction

Advances in remote and automated geo-sensing allow scientists to monitor dynamic, geospatial phenomena comprehensively. As the resolution and frequency of the data survey increases, the amount of data collected by sensing devices has grown explosively. The collected data with associated geocoordinates are often organized based on the geographical extent that they belong to such as political boundaries, roads, or even experimental geospatial shapes predefined by scientists. Georeferencing is a term that is widely used in geographical information systems to describe the process of associating a physical map with spatial locations [2]. Georeferencing may be applied to any kind of geospatial objects or structure such as points of interest, roads or place [3]. Data points collected from geo-sensors are mapped to the corresponding shape of the area (e.g. city boundaries) for subsequent research such as air quality analysis, traffic monitoring, or surveillance of an emerging, infectious diseases. In this study, we describe our methodology to facilitate high-throughput georeferencing of voluminous sensor datasets collected from a high-throughput phenotyping platform (HPP). HPP is an emerging technology to identify genetic variations underlying important quantitative trait loci. Thousands of recombinant inbred lines with various environmental treatments often create a massive set of plots and automated sensor arrays enable highly frequent data acquisitions throughout the lifecycle of the plant [4, 5]. Timely georeferencing between the newly collected voluminous sensor data and experimental setup is key to enabling researchers to understand traits and patterns as early as possible in the course of an experiment.

There are existing GIS tools and geospatial database management systems supporting georeferencing. PostgreSQL [6], MySQL [7], ArcGIS [8], and MongoDB [9] support geospatial relationship queries that includes a query evaluating whether a geospatial object (e.g. point or polygon) is inside another geospatial object. However, these features were not originally designed for large-scale of georeferencing query evaluations with frequent data staging and ingestions. These systems require the ingested data to be indexed before the georeferencing query is evaluated. There are geospatial analytics frameworks [10, 11], that are based on distributed, open-source frameworks

such as Apache Hadoop [12] or Spark [13]. These software frameworks leverage additional indexing schemes over HDFS to avoid disk I/O that, in turn, result in notable overheads in processing in addition to ineffective memory utilizations for the continuously arriving voluminous data.

To address these challenges, we propose RADIX, a framework for accomplishing high-throughput, effective and scalable georeferencing for sensor-based monitoring environments. Our framework supports: (1) on-line and off-line high-throughput georeferencing for vector point datasets, (2) effective generation and indexing of advanced metadata, and (3) data availability for monitoring and explorative analytics. RADIX is also interoperable with the cloud-based data analytics and archival systems.

1.1 Scientific Challenges

Georeferencing and storing large volumes of sensor data in an efficient manner introduces a set of unique challenges.

- **Voluminous Data** With sensor-equipped autonomous vehicles able to record hundreds or thousands of observations per second, and the ability to easily add additional sensors to an existing array, the data generated increases dramatically.

- **Scalability** As data volumes increase, additional storage and computing power may be needed. A proposed system must be able to easily scale with the addition of new resources.

- **High-throughput Georeferencing** The system must be capable of processing large volumes of data quickly so that users may explore and analyze it without waiting for excessive periods of time.

1.2 Research Questions

An effective georeferencing scheme is vital to organizing raw observations from sensors with the scientific context specified in the experiment. Research questions that guide this study include the following:

- **RQ1:** How can we perform the high-throughput georeferencing while accounting for the customizable shapes of the areas being monitored while preserving accuracy?
- **RQ2:** How can we effectively orchestrate the georeferencing with other data preprocessing required during the raw data ingestion such as metadata extraction and/or data indexing?
- **RQ3:** How can the system monitor the process of data georeferencing and provide explorative analytics features to understand characteristics of recently ingested data?

1.3 Approach Summary

RADIX is a distributed georeferencing engine for organizing voluminous sensor observations. RADIX does not require any grouping or filtering of a dataset prior to performing georeferencing. We propose a nested hash grid that is a distributed bitmap indexing scheme to encode the observations and the targeted geospatial shape. The nested hash grid is a hierarchical bitmap hash that maintains multiple resolutions of geospatial blocks that overlap over a region. Based on the shape that users specify, RADIX determines the resolution levels for the observations. Small areas or complicated shapes often cause higher levels of the nested hash grid to achieve better accuracy. The georeferenced data points are indexed, grouped and stored based on the nested hash grid of the associated geospatial coordinates.

A distributed protocol was designed and developed that performs georeferencing with the nested hash grid over Galileo, a distributed storage framework for the voluminous spatiotemporal observations. RADIX pipelines metadata extraction, indexing, and staging with georeferencing to allow users to access the index of the metadata immediately after a data point is georeferenced.

Users can browse and search data with extended metadata (e.g. statistical summary) while the data is being ingested. This feature is useful for monitoring data ingestion and diagnostic activities that are often required during a project. Our empirical evaluation shows that we can achieve ingestion rates of 48MB/sec in a cluster of comprising 25 nodes and our system outperforms Apache GeoMesa, a distributed geospatial analysis platform leveraging Apache Spark, 8-fold when contrasting latency for ingestions.

1.4 Thesis Contribution

RADIX enables ingestion of voluminous observation data while supporting high-throughput georeferencing, metadata extraction, indexing and staging. Users are allowed to access continually arriving observations. Our nested hash grid provides a balance between accuracy and latency. Specific contributions of RADIX include: 1) high-throughput and customizable georeferencing, 2) a streamlined metadata extraction and indexing scheme to reduce the latency and data movements within the storage cluster, and 3) real-time monitoring of data ingestion and diagnostic analyses.

Chapter 2: Related Work

There are two major software architectures that can be compared to this work: Databases and GIS platforms, and distributed geospatial frameworks. The former includes software such as MySQL, MongoDB, ArcGIS, and PostgreSQL. The latter includes works such as GeoMesa, SpatialHadoop, VegaIndexer, and Sphinx. Each of these architectures share many similarities with this work, but all suffer from scalability issues or lack of pipelined data insertion to maximize throughput. Additionally, some efforts have explored the use of sketching algorithms to manage spatiotemporal data volumes that arise in observational settings. The current state of the art is the Synopsis sketch [14] which extracts information from observational data and organizes it an in-memory, distributed tree that scales (in and out) dynamically based on the density of the observational space. The Synopsis system uses a stream processing layer Neptune to facilitate high-throughput data ingestions [15, 16].

2.1 Databases and GIS Platforms

MySQL [7] is a relational database. Relational databases are organized as a table, or set of tables with a particular schema. Often, data stored in one table may have dependencies or relationships with data in other tables. SQL databases are particularly powerful for structured data, but do not perform well with unstructured data, such as text documents or video files due to the fact that they are not easily converted into row-column format. Additionally, SQL databases are only vertically scalable, meaning that they can be scaled only be using more powerful hardware. This is not always feasible.

MongoDB [9] is a NoSQL database program that utilizes JSON-like documents with schemata. Instead of storing data as tables, it is stored as collections comprised of individual documents. This model works well for data that can be grouped together without dependencies on other documents. For example, a large number of agricultural plots could be represented as a set of documents,

one for each plot. This allows for fast retrieval of plot-level data given that the queries include knowledge of the exact plot numbers being searched for. The primary downfall of this architecture is that relational and polygon queries are not efficiently evaluated unless the proper schemas are in place, which places a heavy work-load on developers.

ArcGIS [8] is a geographic information system designed for working with maps and managing geographic information in a database. It offers powerful analytics and mapping capabilities for geospatial data but is not designed for large-scale raw sensor data. ArcGIS differs from RADIX primarily in that it was built for more complex analysis of spatial data, although it is capable of georeferencing.

PostgreSQL [6] is SQL database that also supports unstructured data in the form of JSON structures. PostgreSQL is an ORDBMS (Object-relational database management system). PostgreSQL, like MySQL, supports spatial indexing of data, but requires an additional package. Since PostgreSQL is a relational database, it suffers from scalability issues much like MySQL.

2.2 Distributed geospatial frameworks

GeoMesa [11] is a tool that enables large-scale spatiotemporal querying and analytics on distributed computing systems. It is designed to run atop distributed storage systems such as HDFS and Apache Accumulo. By default, GeoMesa creates indices on the geospatial features of datasets. These indices are implemented by creating a space-filling curve atop a Geohash index. Query polygons are disseminated into geohash coverages, which are recursively broken down into finer-grained geohashes in order to evaluate points intersecting the query polygon. The key difference between RADIX and GeoMesa is that RADIX allows a user-defined granularity of greater than 4 bits for the Geohash index. This allows large amounts of data concentrated in a small geospatial area to be more efficiently indexed and queried.

VegaIndexer [17] is a spatio-temporal indexing scheme designed to run on top of an existing distributed file system such as HDFS, or a distributed NoSQL database such as HBase [18]. This indexing scheme utilizes both time and space attributes to index incoming data points to be queried

at a later time. Rather than using a bitmap index for indexing geospatial data, this system utilizes a MDR+ (Multi-version Distributed enhanced R+) tree. This structure is based off of an R-tree, but was developed to be distributed across a compute cluster.

SpatialHadoop [19] is another spatial indexing scheme built on Hadoop. This approach utilizes a two-tiered indexing scheme, in which a global index is created to index partitions across nodes, and a local index is created to index data in individual partitions. Local indexing can be performed with either an R-tree or a grid index. This grid has the disadvantage of being single-layered, unlike the nested hash grid. SpatialHadoop is intended for data sets covering large geographic areas, as opposed to small areas requiring precise indexing.

Sphinx [20] is a full-fledged distributed system which uses a standard SQL interface to process big spatial data. It adds spatial data types, indices and query processing, inside the code-base of Cloudera Impala. To index spatial data, Sphinx utilizes a two-layered index on HDFS-resident tables. Much like SpatialHadoop, the master node is responsible for maintaining the global index, which defines how records are partitioned across machines, and the local index on each node defines how records are internally organized on that node. Both R-trees and R+-trees are utilized for these indices. Due to the master-slave architecture of these systems, they are prone to failure or other issues if the master node fails. RADIX avoids this issue by acting as a distributed hash table, in which no master node is required.

[21–26] provide the distributed storage based approaches to achieve fast and effective spatiotemporal analytics features. In these approaches, data are indexed using a hierarchical distributed hash table, and the analytics queries are evaluated without any centralized coordinator. However, the query evaluation is performed over the data set already ingested with a pre-defined indexing scheme in these systems. The large scale data ingestion to these systems is challenging due to the misalignment between the patterns of data acquisition and the indexing schemes. Radix provides a high-throughput data ingestion methodology that is well aligned with the underlying data retrieval system. Also, Radix dynamically orchestrates the computations required during the georeferencing for the sensor data collections.

Chapter 3: Background

3.1 Plant Phenotyping

A plant phenotype is the ensemble of all observable traits of an organism (e.g kernel color and kernel texture for corn). Phenotyping is described as the identification of effects on the phenotype resulting from genotype (genetic code) differences (G) and environmental conditions (E). This can be summarized as P = $G \times E$ [27]. A more detailed description would be P = $Genotype + Environment + G \times E$, which includes both the additive and non-additive interactions of environment and genotype. Phenotyping is typically hypothesis driven, aiming to answer specific questions about gene function. As technology continues to advance, collection of phenotypic information is becoming increasingly automated and less labor-intensive, allowing for large volumes of data to be collected at a high rate, and with high accuracy. For example, Tanger et. al [4] developed a high throughput phenotyping (HTP) platform capable of collecting data at a rate of 3000 plots per hour for a population of recombinant rice. Not only does HTP provide a larger body of data to research, it also allows for time-series characterization of quantitative trait loci over plant lifetime. In our study, phenotypic data is collected by a sensor array mounted on an autonomous vehicle traveling through agricultural plots.

3.2 Georeferencing

Georeferencing is a broad term that may be refer to one of several specific meanings. In any sense, georeferencing is an important process when dealing with any geospatial information, as it provides meaning to arbitrary points in space, often for very specific purposes. For example, an individual pixel in a raster image may be georeferenced according to a specific projection scheme, or an individual coordinate may need to be georeferenced to some form of landmark or boundary, such as a particular address or a county boundary. Generally, georeferencing is the process of

associating something with a location(s) in physical space. Any object that can be related to a geographic location is able to be georeferenced (e.g a building, landmark, or data point). There are several variations of georeferencing. The most common description is raster to geolocation georeferencing [2]. This refers to the process of aligning raster images to a map coordinate system. Most modern satellite images and aerial cameras already contain spatial reference information by using GPS, but scanned images and historical data must often be georeferenced. This type of georeferencing requires ground control points in order to correlate individual pixel values in raster image to a pre-existing coordinate system and may additionally require the use of a map-projection, which is a method of projecting the earth's curved surface onto a flat surface. Direct georeferencing [28] is similar but does not require the use of ground control points. This type of georeferencing is designed for autonomous drones in order to maintain real-time knowledge of its location during flight. This is achieved with a combination of GPS and aerial imagery. Nørremark et. al [29] refer to georeferencing as mapping individual plants to geolocations. Additionally, georeferencing may refer to the process of associating geographic coordinates to polygons on the earth's surface, e.g. mapping points to a particular county in a state. For this study, we refer to georeferencing as the process of mapping a data point associated with a geographic coordinate to a pre-defined polygon on the earth's surface. These polygons represent agricultural plots, which we will refer to as plots throughout the remainder of this paper.

3.3 Distributed Spatiotemporal Data Storage

To perform georeferencing on voluminous phenotyping data while providing maximum data locality, we leverage a distributed storage framework Galileo [30–33]. Galileo is a high throughput distributed storage framework designed for large, geospatial and time-series datasets. It is modeled as a hierarchical distributed hash table and utilizes the underlying host file system for storage of data on physical media. A Galileo cluster is logically organized into groups, essentially turning a cluster into a cluster of smaller clusters. This organization allows similar data to be grouped together rather than on random nodes. We utilized Galileo to store data once it had been processed

and georeferenced. The basic unit of storage in Galileo is a block. A block is a multi-dimensional array of data that is ultimately stored on disk as a stream of bytes that may represent any type of data. Every block stored in Galileo is accompanied by its corresponding metadata. The metadata for a block is stored in an in-memory metadata graph and is also stored as a file alongside the block in the underlying filesystem. Upon failure or shutdown, a given node is able to restart and utilize the metadata stored on disk to rebuild the graph. This metadata graph allows for individual nodes to process queries for blocks without having to read from disk, drastically decreasing query processing time. Galileo supports user-configurable data partitioning schemes. In order to partition data amongst nodes within a group, a two-tiered hashing scheme is used. The first hash function determines the group a data point belongs to, while the second hash function determines the node within that group. When a query comes in to a node in the system, the metadata graph is evaluated to determine if the node has any data matching the query. This allows for queries with no matching data to be quickly ignored, avoiding disk I/O. If information in the metadata graph matches the query, the corresponding block can either be returned as a whole or evaluated for specific portions of data matching the query. In the process of georeferencing, we utilize the Geohash [34] geocoding scheme to avoid a brute-force search of polygons to associate a data point with. A Geohash is a 1-dimensional Base32 string representation of a 2-dimensional bounding box on the earth's surface. A longer Geohash string indicates a higher geographical precision, i.e a finer-grained bounding box. A Geohash is created by intertwining the bits of latitude-longitude pairs. For example, the decimal coordinates 40.58532°N, -105.084379°W would map to 9xjqbsrfs, with each character represented by five bits. Additional characters may be added or removed to decrease or increase respectively the size of the bounding box that the string represents. This geocoding scheme allows for a coarse-grained Geohash to be defined as a grid of more fine-grained Geohashes. This grid can be converted into a bitmap, which is utilized by the nested hash grid discussed in Chapter 3.

Chapter 4: Methodology

To achieve high-throughput data ingestion that entails compute-intensive preprocessing, our methodology targets concurrent georeferencing and metadata extractions while pipelining computationally intensive operations to reduce the data communication and I/O access. Our data ingestion consists of three stages: stamping, georeferencing, and metadata extraction. The indexing scheme involved in both the stamping and georeferencing phases is the nested hash grid. The nested hash grid is a distributed spatial indexing scheme that converts geographic coordinates to an array of bits to provide coarse-grained data mapping by grouping spatially neighboring data points. We specify this grouping based on geographical proximity by leveraging the geohash algorithm [34]. The geohash represents a two-dimensional location (based on latitude and longitude) on Earth as one-dimensional Base-32 string where the geographical extent is controlled by the length of string: longer strings represent smaller, contiguous spatial regions.

We use this hash grid for indexing pre-defined shapes (e.g. the HPP plots, or state boundaries) and ingested data points. Figure 4.1 depicts the bitmap representation of two spatial shapes. The cells that intersect with the shapes are set to 1. Once the shapes are indexed, newly ingested data points are indexed with a separate bitmap representation to perform geospatial queries. To evaluate intersections between a shape and a set of data points, we perform a bitwise AND of the two

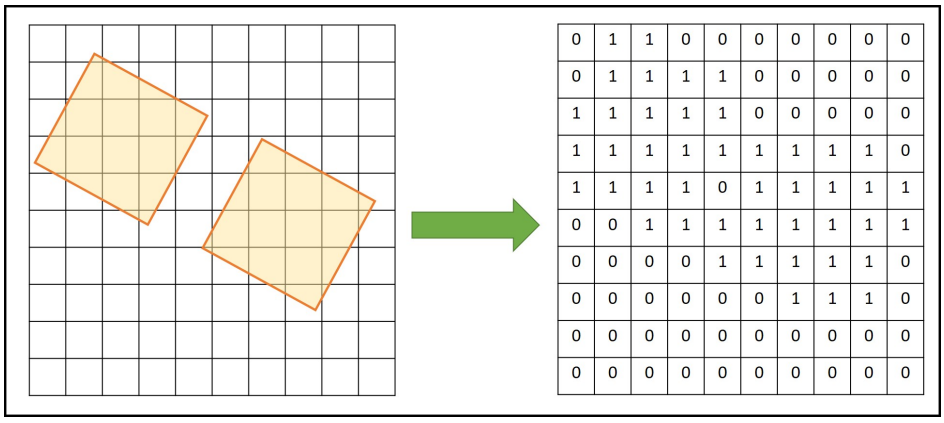

Figure 4.1: Conversion of plot shapes to bitmap index representation.

bitmaps; this reduces the query evaluation time considerably compared to the traditional R-tree based query scheme [35]. To reduce compute times and memory requirements, the system dynamically lowers the precision of the bitmap index. The challenge of this approach is to manage ambiguity during query evaluation. If the data points within the same cell map to the multiple shapes, the system needs to evaluate entire data points within the cell. To address this issue, we propose the nested hash grid that is a hierarchical bitmap indexing scheme with configurable resolutions. RADIX inherits the architecture of our underlying storage network topology - distributed hash table: Any subset of nodes in the RADIX cluster can be used as the ingest nodes to provide concurrent data ingestion.

4.1 Stamping with the Nested Hash Grid

To reduce in-cluster data movement during georeferencing, RADIX identifies the corresponding data node and forwards the batch of data points. RADIX reads raw data from disk or network as a chunk, a group of adjacent points within the sequence of data stream. The size of these chunks is tunable, but smaller (< 500 KB) chunks show relatively low latency. Several message sizes were tested: 100 KB, 500 KB, 1 MB, and 5 MB. Each individual record within a message consisted of 16 features, including 11 double-precision floating point decimals, two strings, one UTC time stamp, one integer, and one boolean. As seen in Figure 4.2, message size increased, throughput tended to decrease. This is caused by a combination of increased compression overhead for larger messages as well as increased processing time to compute metadata.

Figure 4.3 displays the multi-tiered hashgrid with 3 layers. RADIX maintains hash grid indices of shapes with multiple resolutions in the main memory of nodes. The hash grid with the lowest resolution includes complete coverage of bitmap information of the shapes. If there is a cell of the bitmap that intersects with multiple shapes, RADIX calculates a finer resolution of the hash grid only for those cells. Adjusting the length of the geohash value enables nested coverage between the finer and coarser resolutions of the hash grids. The levels of nesting are extensible based on the dataset. Stamping involves two steps, identification of the storage node and data forwarding.

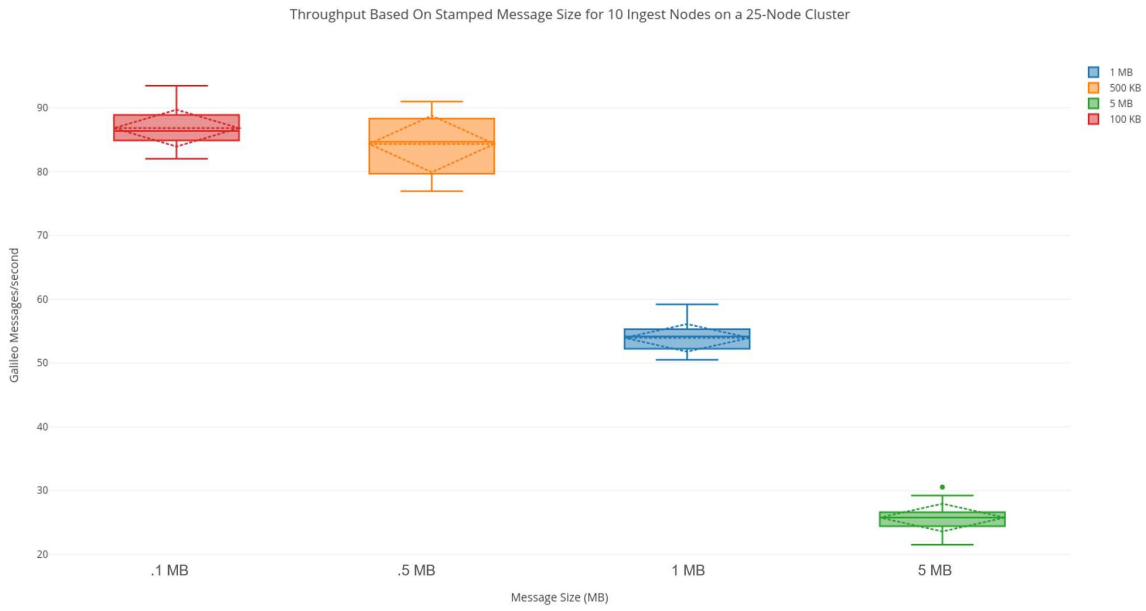

Figure 4.2: Throughput in messages per second for varying stamped message sizes.

First, each chunk is placed into a global queue and each data point in a chunk is indexed in a hash grid object. Once the hash grid for the chunk is ready, it is evaluated by mapping with a lowest resolution hash grid of the shapes. Overlapping between a shape and data chunk indicates that the portion of data chunk may belong to the spatial shape. If the cell in the shape hash grid intersects with multiple shapes, the data is evaluated with one level finer hash grid of the shapes. Second, the chunks are then forwarded to the RADIX node that is responsible for the majority of the chunk. The node that received the data chunk performs further processing locally. All chunks are compressed with the Snappy [36] compression scheme to minimize bandwidth consumption. Once compressed, the message is sent to the appropriate destination as an unprocessed store message.

4.2 Distributed Georeferencing

The process of georeferencing is performed asynchronously over the storage cluster. We call this design as a relaxed georeferencing. With a relaxed georeferencing, data points are evaluated in a data-parallel way and merged at the end of the data ingestion process or query evaluation.

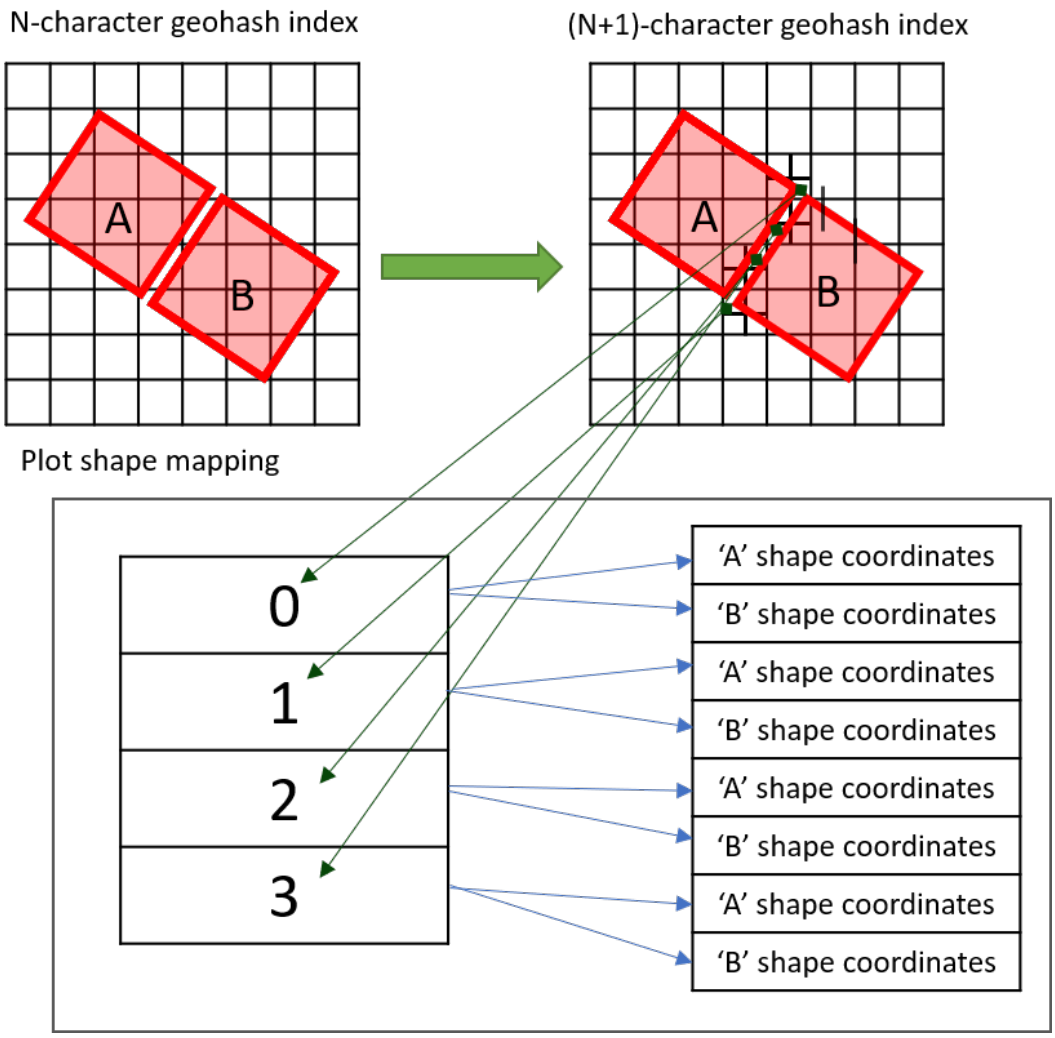

Figure 4.3: Georeferencing plot shapes with the nested hash grid .

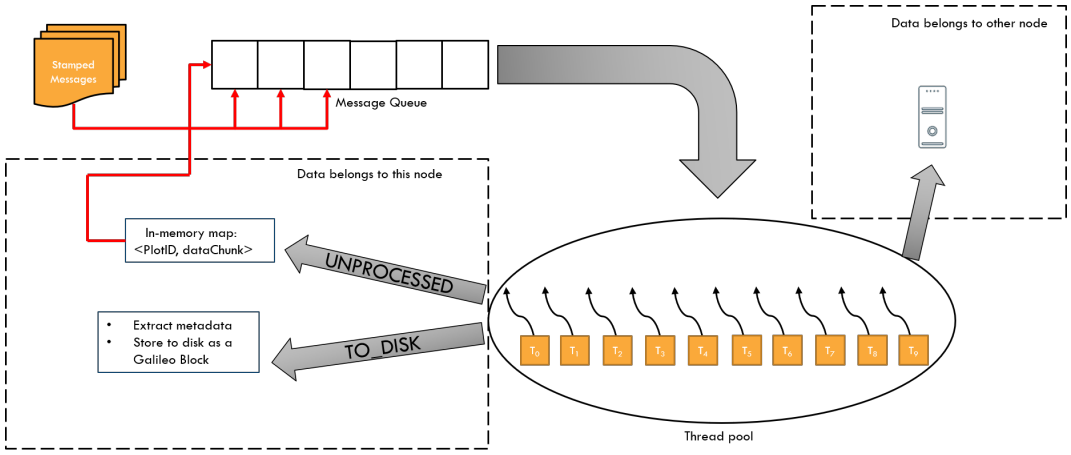

Figure 4.4: Message handling with thread pool for stamped messages.

This allows data to be split arbitrarily amongst nodes within a cluster to exploit a high level of concurrency. The process of georeferencing relies entirely on the global hash grid. As incoming unprocessed messages are received at a node, they are placed into a message queue from which 10 message handler threads pull. These threads handle one of two types of store messages: unprocessed and to_disk. Figure 4.4 depicts the process of message handling at each node. An unprocessed message is one which has been stamped and transferred to the node responsible for storing most of its contents. When a thread receives an unprocessed message, each data point in the message is examined to determine whether this node is responsible for storing it. If a point indeed belongs on this node, it is georeferenced by the hash grid and placed in an in-memory map. This map tracks all data points belonging to a particular polygon, and groups them together. If new data has not been observed for some polygon for more than a time threshold, all existing data for that polygon is removed from the mapping and transformed into a to_disk message and placed in the queue. A temporary map is created for handling of data points that do not belong to the node processing them. Points are added in the same fashion as they are added to the previously mentioned map, although data in this map is lumped together and packaged as a new unprocessed message and sent to the node it belongs to. Clearing portions of the in-memory map at a regular interval is crucial to managing memory consumption, especially when data is processed at the terabyte or petabyte scale. The 30 second interval was chosen to ensure that most or all data for a

given polygon is aggregated before extracting metadata and storing to disk. This prevents excessive disk I/O during this phase of ingestion.

4.3 Metadata Extraction

Once data is aggregated at the polygon level, it is packaged into a to_disk message, and added to the same queue that receives unprocessed messages. When a thread receives a to_disk message, metadata is extracted for that message and added to the Galileo metadata graph. The type of metadata computed is specified by the user before ingestion begins. Typically, the metadata is comprised of simple summary statistics. These are generally lightweight computations that provide a general summary of a data block and also provide a reasonable query scope. Once the metadata has been added to the metadata graph, it is stored on disk alongside the raw data. The on-disk metadata acts as a checkpoint in the case that a node fails and must rebuild its metadata graph.

4.4 Interactive Diagnoses

To facilitate discovery analytics over data that has been ingested, we have created a web interface that allows users to visually explore and query the data stored in the system. Upon login, a user is presented with a Google Maps [37] view that displays all the current polygons for which data is held. Figure 4.5 shows the data exploration interface, which we have named RADIX. Two types of queries are currently supported: polygon bounded geospatial query and feature query. Polygons may be drawn on the interactive map to specify a particular spatial region to query for. When the back end receives this type of query, a coverage map is created, consisting of all geohashes that cover the area of the polygon. The precision of these geohashes is dependent on the precision specified during the ingestion phase. This coverage map is then compared with the global hash grid to determine which polygons are contained or intersect with the query polygon, and all matching blocks are returned to the user. A feature query is more specific. This type of query is similar to a SQL-style query in that the user specifies a specific feature or set of features, a value to

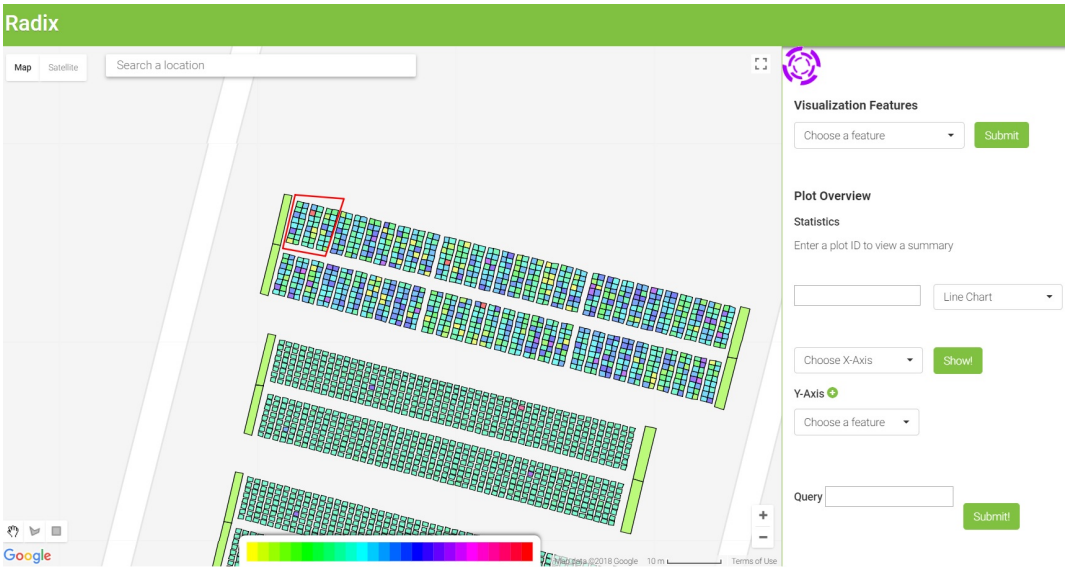

Figure 4.5: RADIX data visualization dashboard with polygon selection.

compare to, and a relational operator for comparison. Existing polygons may be visualized based on feature. This is particularly important for recognizing patterns and outliers in the domain of precision agriculture. For example, if the user desired to look for areas with below or above average nitrogen levels, the visualization feature could be set to nitrogen to view the difference in nitrogen levels between all plots.

Chapter 5: System Architecture

5.1 Data Flow: From the Field to the Archives

Autonomous sensing vehicles have become more reliable and affordable to obtain and can easily be outfitted with a myriad of sensors. These sensing devices are becoming prevalent in many domains, especially precision agriculture [38]. It is crucial that data that is collected by these vehicles can be indexed and stored so it can be analyzed as soon as possible. Our system was designed for daily bulk uploads of field data. There are three major software components in the RADIX framework: the user interface for analytics, the data ingestion engine, and underlying Galileo storage framework. Once data from the field is collected, it is sent to the data ingestion component. The data ingestion engine runs on top of Galileo and communicates for metadata extraction and storage. This architecture allows for the Galileo framework to be constantly up and running, and data can be ingested at any number of nodes at any time with a simple Netcat [39] command. This design also streamlines the process of georeferencing, metadata extraction and archiving into a single pipeline. Data collected by autonomous vehicle is stored onto an external storage media and transported to the physical location of the servers to be transferred locally. Web uploads of data are also supported; however, this method incurs a high latency cost. Once the media is inserted and the command issued, each ingest node will begin stamping data chunks as they are read and transfer them to their corresponding destination. As soon as metadata is calculated and placed in the metadata graph, it is immediately available to query, regardless of whether the raw data has been dumped to disk. This allows for data to be queried in the least amount of time possible after collection and ingestion.

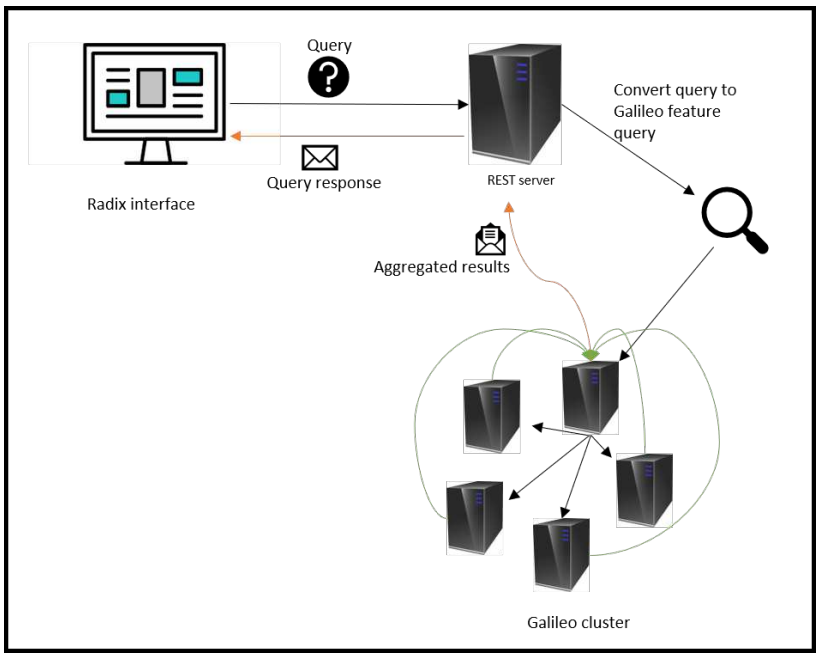

Figure 5.1: Query evaluation process from RADIX interface to back-end Galileo servers.

5.2 Interaction Between Components

The Galileo framework interacts with both the user interface and the data ingestion engine. All communication with Galileo is performed with Galileo messages. Message handling is performed with socket connections on a separate thread for each node. There are several types of message, with each message type invoking a particular action and response based on the information it carries. Figure 5.1 shows the interaction between each component during query handling. The web-based interface relies on a translation server that acts as a middle-man for communication between the client and the back-end Galileo cluster. This node could be a member of the cluster or a separate machine. For the purpose of load balancing, we employed an external node of the Galileo cluster. This allows us to orchestrate data traffics to idle nodes or nodes with lower workloads. This node is implemented as a RESTful [40] server, accepting web requests and translating them to Galileo messages, and passing those messages to the back end. When the REST server receives a query, it is translated into the appropriate Galileo message type and sent to a node in the cluster. The Galileo node receiving the query will forward the query to all other nodes, while simultaneously evaluating the query against its local metadata graph. Responses from all other nodes are sent to

the node that received the original query, where they are aggregated into a single message and sent back to the REST server. The REST server then converts the results into a JSON [41] object and returns it to the user. The data ingestion engine communicates with Galileo only during the process of georeferencing and stamping. The actual computations performed during these phases are handled by the data ingestion engine, and the results are passed to the Galileo framework, which updates the metadata graph and handles disk I/O.

Chapter 6: Performance Evaluation

We describe the results of our benchmark by profiling several aspects of RADIX, including memory consumption, CPU utilization, data ingestion/georeferencing performance, and query performance. Several cluster configurations were used and are described in the next section. All query results were validated for 100

6.1 Dataset and Environment Setup

We have used a high throughput phenotyping platform (HPP) dataset that is synthesized from existing HPP data. The data contains 15 attributes including datetime, geographic coordinates, plant genotype, and an array of randomly generated numbers to represent various sensor readings. The majority of the data types are floating point decimals. The total size of data used in this benchmarking ranged from 8 GB to 1 TB and the geospatial coverage of this dataset is approximately 20,000 m^2. Since RADIX is targeting off-line high-throughput data ingestion, the dataset is segmented and loaded to the ingest nodes. Performance evaluations reported here were carried out on a base cluster of 40 HP DL160 servers (Xeon E5620, 12 GB RAM). The test cluster was configured to run Fedora 24, and RADIX was executed under the OpenJDK Java runtime 1.8.0 72. For evaluations involving Apache GeoMesa, we used Apache Spark version 2.1.0 with HDFS 3.1.1 with a 25 node cluster. For large cluster configurations, we combined our baseline cluster of 40 machines with 30 HP DL320e servers (Xeon E3-1220 V2, 8 GB RAM) and 30 HP DL60 servers (Xeon E5-2620, 16 GB RAM). Smaller cluster configurations consisted of subsets of the base cluster.

6.2 Resource Utilization

We first compared memory and CPU utilization for a cluster of 15 nodes, and separate cluster of 160 nodes. We evaluated resource utilization for 2 different data input sizes: 8 GB and 1 TB.

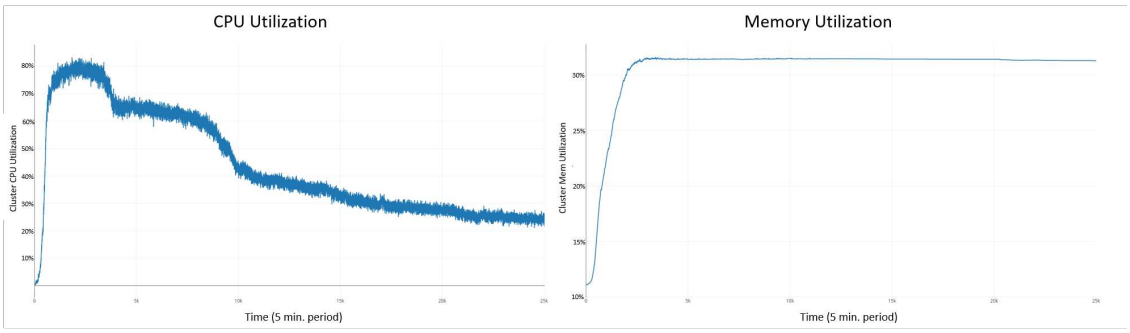

Figure 6.1: Memory and CPU utilization on 160-node cluster for 1 TB ingest file over 5 minute period during ingestion.

For the 15-node cluster, we measured resource use based on varying number of ingest nodes, while the 160-node cluster utilized every node as an ingest node. The ability to use an increased number of nodes as ingest nodes can increase both throughput and resource utilization in most scenarios. Figure 6.1 illustrates CPU and memory utilization on a 160-node cluster during the first 5 minutes of a 1 TB data ingest. In this scenario, each of the 160 nodes acts as an ingest node. The first 45 seconds during ingestion see a large spike in CPU usage, as each machine utilizes several cores for generating stamped messages, and other cores process messages from other nodes. Figure 6.2 compares resource utilization for an 8 GB ingest with 2 (a, d), 4 (b, e) and 8 (c, f) ingest nodes. Memory consumption spikes early during the ingest phase, but gradually drops as more messages are processed and stored. Once ingesting has completed, the metadata graph is still maintained in memory, causing persistently moderate memory usage but allowing for faster query evaluation.

6.3 Throughput

System throughput was measured as the number of messages per second processed for varying numbers of ingest nodes on a 25-node cluster. We measure throughput in terms of messages processed per second. The two message types are unprocessed and to disk, as discussed in Section 3. Message size is user-configurable and can impact performance with larger sizes. We chose a message size of 120 KB for testing purposes.

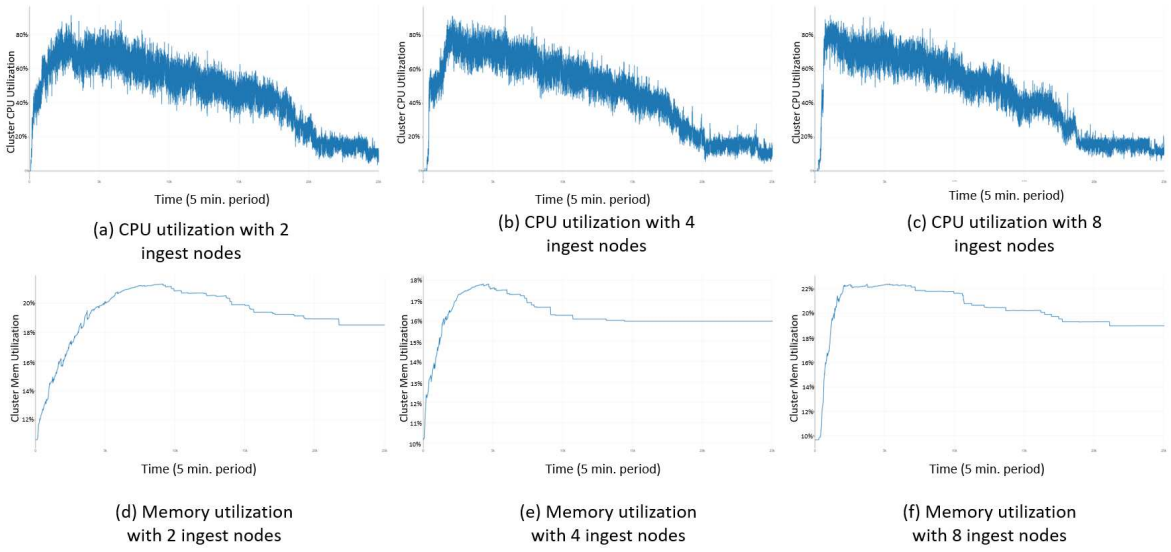

Figure 6.2: Memory and CPU utilization on 15-node cluster for 8 GB ingest file over 5 minute period during ingestion.

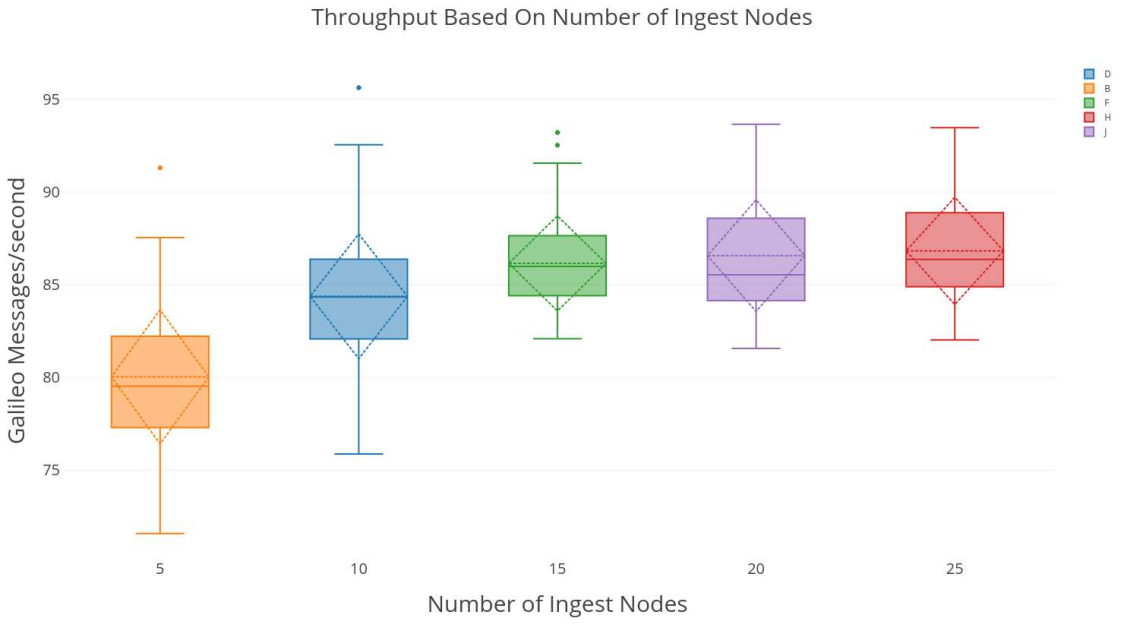

Figure 6.3: Throughput in messages/sec for varying numbers of ingest nodes.

Figure 6.3 depicts throughput in messages per second for varying numbers of ingest nodes on a 25-node cluster. Throughput increases logarithmically as the number of ingest nodes increases and levels off when roughly half of the cluster nodes are used as ingest nodes. This is due to the fact that the ingest nodes devote more processing power to disk I/O and stamping, with less threads able to handle incoming messages from other nodes. Throughput is affected by the number of features for which metadata is computed. A dataset with a significantly large number of features will see diminished throughput as metadata computations are more complex, while a dataset with fewer features will achieve greater throughput.

6.4 Georeferencing Query and Feature Query Evaluation

To demonstrate the low latency of query evaluations with the nested hash grid and metadata graph, we performed a variety of queries against MySQL, GeoMesa and RADIX. We evaluated polygon queries, in which each system must identify all points intersecting a user-defined polygon shape, as well as feature queries, where all features matching the user's query are returned. For MySQL queries, an index was created for the features queried, and a spatial index created on the spatial column. Our GeoMesa configuration consisted of a spatiotemporal index, with no indices on any other features. For query evaluation experiments, the RADIX cluster and GeoMesa Hadoop cluster each consisted of 25 nodes from the base cluster, with 10 ingest nodes for RADIX and 10 threads for GeoMesa. MySQL was run on a single machine with 8 threads to insert data to maximize insertion speed. MySQL required additional time to create indices on the desired columns that was not accounted for. Figure 6.4 shows query evaluation time for each system for both polygon queries (blue) and feature queries (orange). Even with a spatial index, polygon queries are slow for MySQL. Additionally, feature queries are 20 times slower in GeoMesa than for MySQL or RADIX. Figure 6.5 shows the time to fully ingest a 10 GB file in each system. RADIX outperforms both systems by nearly 8-fold. GeoMesa suffers a penalty during ingestion due to the fact that indexing is not pipelined.

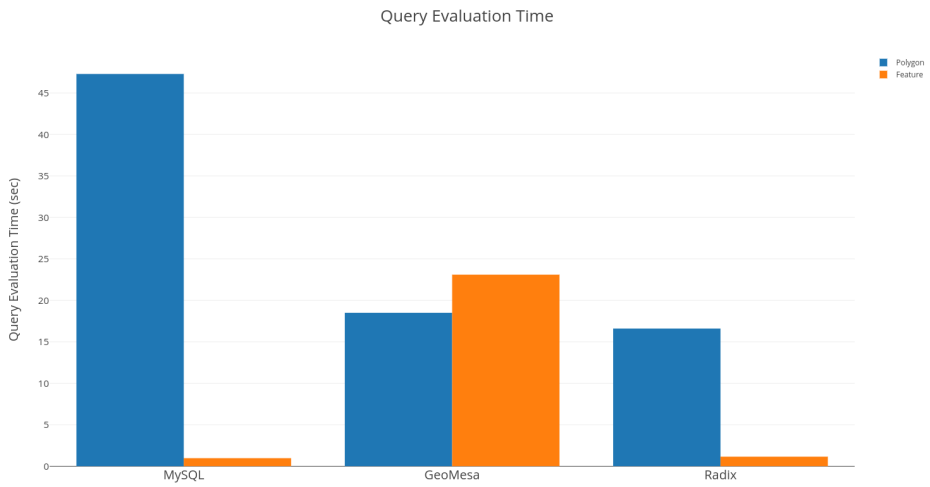

Figure 6.4: Query evaluation time for polygon and feature queries.

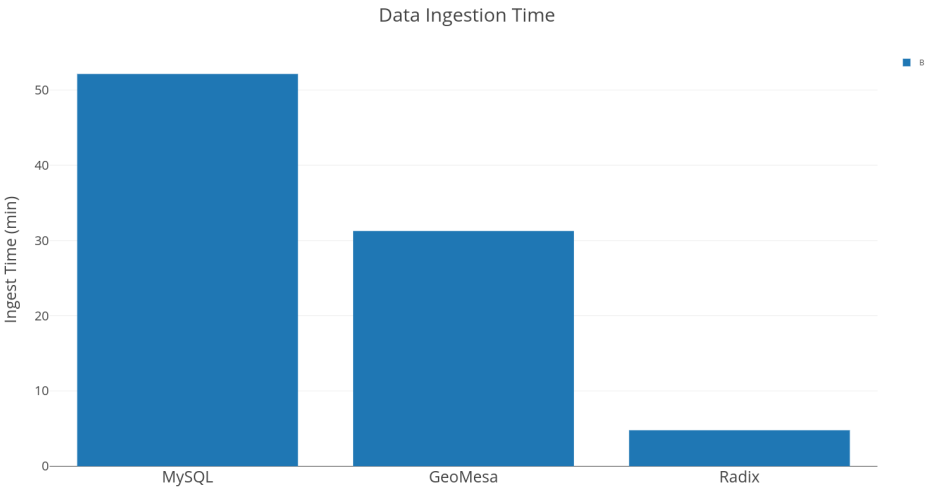

Figure 6.5: Time to fully index and store a 10 GB file.

Chapter 7: Conclusion and Future Work

In this work, we presented a spatial indexing scheme and data processing pipeline for georeferencing and extracting metadata from observational data.

- **RQ1:** The nested hash grid provides accurate and quick georeferencing for large-scale datasets. It allows for extremely fine-grained spatial indexing without excessive time or memory costs. The multi-tiered design allows facilitates high accuracy without high memory cost.

- **RQ2:** RADIX was able to significantly outperform both MySQL and GeoMesa in terms of data ingestion, while also providing near equal query evaluation times. Raw data is georeferenced and aggregated at a plot or shape level so that metadata extraction can occur at that same level of granularity. Additionally, metadata extraction is pipelined in the ingest process, avoiding an extra stage of data processing.

- **RQ3:** Queries are evaluated against in-memory data structures such as the hash grid and metadata graph, avoiding disk I/O and providing fast response times. This allows the system to maintain an up-to-date view of all data that has been ingested, including data that was just recently ingested.

The ability to utilize any number of nodes as ingest nodes can significantly increase overall throughput, but only to a certain point. Using n/2 nodes as ingest nodes in an n-node cluster achieved the best results in most cases. The entire system is easily scalable, as nodes can be easily added or dropped from the system at any time. Overall, our approach provides an avenue for handling daily data ingestion at extreme scales, while also providing a simple interface for visually exploring the data.

While the system performs very high-throughput data ingestion and georeferencing, there is room for improvement. In particular, (1) polygon query evaluation method could be modified to

reduce query evaluation time when the polygon covers a large percentage of the total data stored, (2) finer-grained query evaluation such that when a polygon intersects a plot, only data falling within the defined polygon is returned, rather than all data for the intersecting plot, and (3) allowing multi-geometry queries, such that multiple sub-groups of plots may be queried.

References

[1] Maxwell Roselius and Sangmi Pallickara. Radix: Enabling high-throughput georeferencing for phenotype monitoring over voluminous observational data. *(To appear) Proceedings of the IEEE International Conference on Big Data and Cloud Computing (BDCloud 2018), Melbourne, Australia*, 2018.

[2] Linda Hill. *Georeferencing: The Geographic Associations of Information*. The MIT Press, 2006.

[3] A. Hackeloeer, K. Klasing, J.M. Krisp, and L. Meng. Georeferencing: a review of methods and applications. *Annals of GIS*, pages 61–69,2014.

[4] P. Tanger, S. Klassen, J.P. Mojica, J.T. Lovell, B. Moyers, and J. Baraoidan, M.and ... McKay. Field-based high throughput phenotyping rapidly identifies genomic regions controlling yield components in rice. *Scientific Reports*, 7,2017.

[5] D.J. Mulla. Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems Engineering*, 114(4):358–371, 2013.

[6] Regina Obe and Leo Hsu. *PostgreSQL: Up and Running*. O'Reilly Media, Inc., 2012.

[7] Michael Widenius and Davis Axmark. *Mysql Reference Manual*. O'Reilly and Associates, Inc., 2002.

[8] ESRI. Arcgis desktop: Release 10. Software Release, 2011.

[9] Kristina Chodorow and Michael Dirolf. *MongoDB: The Definitive Guide (1st ed.)*. O'Reilly Media, Inc., 2010.

[10] R. Giachetta. A framework for processing large scale geospatial and remote sensing data in mapreduce environment. *Comput. Graph.*, 49:37–47,2015.

[11] James Hughes, Andrew Annex, Chris Eichelberger, Anthony Fox, Andrew Hulbert, and Michael Ronquest. Geomesa: a distributed architecture for spatio-temporal fusion. *Geospatial Informatics, Fusion, and Motion Video Analytics*, 2015.

[12] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media Inc., 2012.

[13] M. Zaharia et al. Spark: Cluster computing with working sets. pages 10–10. Hot-Cloud, 2010.

[14] Thilina Buddhika, Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Synopsis: A distributed sketch over voluminous spatiotemporal observational streams. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2552–2566, 2017.

[15] Thilina Buddhika and Shrideep Pallickara. Neptune: Real time stream processing for the internet of things and sensing environments. *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium*, 2016.

[16] Thilina Buddhika, Ryan Stern, K. Lindburg, K. Ericson, and Shrideep Pallickara. Online scheduling and interference alleviation for low-latency, high-throughput processing of data streams. *IEEE Transactions on Parallel and Distributed Systems*, 28(12):3553–3569, 2017.

[17] J. Fang Y. Zhong and X. Zhao. Vegaindexer: A distributed composite index scheme for big spatio-temporal sensor data on cloud. *IEEE International Geoscience and Remote Sensing Symposium*, pages 1713–1716, 2013.

[18] Apache Software Foundation. Hbase. Web. Available: http://hbase.apache.org.

[19] Ahmed Eldawy and Mohamed F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. *IEEE 31st International Conference on Data Engineering*, pages 1352–1363, 2015.

[20] A. Eldawy, M. Elganainy, A. Bakeer, A. Abdelmotaleb, and M. F Mokbel. Sphinx: Distributed execution of interactive sql queries on big spatial data. *23rd ACM Sigspatial International Conference on Advances In Geographic Information Systems*, pages 3–6, 2015.

[21] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Autonomous data management and federation to support high-throughput query evaluations over voluminous datasets. *IEEE Cloud Computing*, 3:40–49,2016.

[22] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Geometry and proximity constrained query evaluations over large geospatial datasets using distributed hash tables. *IEEE Computing in Science and Engineering (CiSE). Special Issue on Extreme Data*, 16(4):53–60, 2014.

[23] Matthew Malensek, Sangmi Lee Pallickara, and Shrideep Pallickara. Fast, ad hoc query evaluations over multidimensional geospatial datasets. *IEEE Transactions on Cloud Computing*, 5(1):28–42, 2017.

[24] Sangmi Lee Pallickara, Shrideep Pallickara, and Milija Zupanski. Enabling efficient data search and subsetting of large-scale atmospheric datasets. *Future Generation Computer Systems*, 28(1):112–118, 2012.

[25] Sangmi Lee Pallickara, Shrideep Pallickara, Milija Zupanski, and Stephen Sullivan. Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections. *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science, Indianapolis*, 2010.

[26] Sangmi Lee Pallickara and Marlon Pierce. Swarm: Scheduling large-scale jobs over the loosely-coupled hpc clusters. *Proceedings of the IEEE International Conference on e-Science*, 2008.

[27] M. Minervini, H. Scharr, and S. Tsaftaris. Image analysis: The new bottleneck in plant phenotyping. *IEEE Signal Processing Magazine*, 32(4):126–131, 2015.

[28] GPS World Staff. Uav real-time: Data use in a lightweight direct georeferencing system. web, 2015. gpsworld.com/uav-real-time-data-use-in-a-lightweight-direct-georeferencing-system/.

[29] M. Nørremark, H.-W. Griepentrog, and S. Blackmore H. Nielsen. A method for high accuracy geo-referencing of data from field operations. *Proceedings of the 4th European Conference on Precision Agriculture*, 2003.

[30] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Galileo: A framework for distributed storage of high-throughput data streams. *Proceedings of the IEEE/ACM Conference on Utility and Cloud Computing*, 2011.

[31] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Polygon-based query evaluation over geospatial data using distributed hash tables. *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 219–226, 2013.

[32] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Geometry and proximity constrained query evaluations over large geospatial datasets using distributed hash tables. *Computing in Science and Engineering*, (1):1–1, 2014.

[33] Matthew Malensek, Sangmi Pallickara, and Shrideep Pallickara. Analytic queries over geospatial time-series data using distributed hash tables. *IEEE Transactions on Knowledge and Data Engineering*, 28(6):1408–1422, 2016.

[34] Wikipedia Contributors. Geohash. Web, 2018. http://en.wikipedia.org/wiki/Geohash.

[35] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[36] Google. Snappy. Web, 2018. http://google.github.io/snappy/.

[37] Google. Maps javascript api. Web, 2018. Retrieved from: https://developers.google.com/maps/documentation/javascript/reference/3.

[38] N. Stehr. Drones: The newest technology for precision agriculture. *Natural Sciences Education*, 44(1):89–91, 2015.

[39] Jan Jr. Kanclirz. Netcat power tools. *Syngress*, 2008. https://doi.org/10.1016/B978-1-59749-257-7.X0001-5.

[40] Roy Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[41] T. Bray. The javascript object notation (json) data interchange format. *STD 90*, 2017. https://www.rfc-editor.org/info/r.