Spring 5-13-2016

A study of three paradigms for storing geospatial data: distributed-cloud model, relational database, and indexed flat file

Matthew A. Toups
University of New Orleans, mtoups@cs.uno.edu

A study of three paradigms for storing geospatial data: distributed-cloud model, relational database, and indexed flat file

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements of the degree of

Master of Science
in
Computer Science

by
Matthew Toups
B.S. Carnegie Mellon University, 2008
May 2016

Acknowledgments

I must express my deep gratitude to Dr. Elias Ioup and Dr. Mahdi Abdelguerfi who not only supported me during this work, but without whom my studies in this program would not be possible. I additionally thank Dr. Shengru Tu and Dr. Md Tamjidul Hoque for their service on my thesis committee.

Finally, I thank my mother, Merry Toups, for her support throughout.

Matthew Toups

Contents

List of Figures

List of Tables

Abstract

Geographic Information Systems (GIS) and related applications of geospatial data were once a small software niche; today nearly all Internet and mobile users utilize some sort of mapping or location-aware software. This widespread use reaches beyond mere consumption of geodata; projects like OpenStreetMap (OSM) represent a new source of geodata production, sometimes dubbed “Volunteered Geographic Information.” The volume of geodata produced and the user demand for geodata will surely continue to grow, so the storage and query techniques for geospatial data must evolve accordingly.

This thesis compares three paradigms for systems that manage vector data. Over the past few decades these methodologies have fallen in and out of favor. Today, some are considered new and experimental (distributed), others nearly forgotten (flat file), and others are the workhorse of present-day GIS (relational database). Each is well-suited to some use cases, and poorly-suited to others. This thesis investigates exemplars of each paradigm.

Keywords:

geospatial index, spatial database, OpenStreetMap, Accumulo, GeoMesa, PostGIS

1 Introduction

The reach and importance of Geographic Information Systems (GIS) have grown since the 1980s, from a niche software market for a few groups of professionals, to a widely-used technology present on almost any Internet-connected computer or mobile device. Access to the technology for both creating and consuming GIS data and related content has made these systems and the data that they process a part of daily life.

One way that both the creation of use of geospatial data has changed has been the advent and growth of Volunteered Geographic Information (VGI)[18]. The largest and best-known example of this is the OpenStreetMap (OSM) dataset, which is both generated by and available to the public, and is used as raw data for these experiments.

The work presented here will consider the data that powers GIS, in particular, *vector* data. What this data is, how it is structured, and three fundamentally different ways of processing it will be detailed in Section 2. These methods, or *paradigms*, span a wide range of topics in computing. The simplest is the *flat file*, indexed for spatial searches but stored simply on a single filesystem and accessed using a simple API. The most complex (and newest) is the *distributed-cloud key-value store*, built atop many levels of software platforms and a possibly large amount of computing hardware. Somewhere in between these, and a longtime player in geospatial computing, is the *relational database*, a powerful tool for managing many types of data. One exemplar of each type is selected and described; brief information on other forms not tested here is also provided.

Section 3 details the process used to deploy these three systems, and how testing was performed in the early part of this investigation. Based on lessons from the initial tests, a more rigorous experimental design is presented, along with three different types of measurements to apply to each of the three exemplar systems.

Section 4 presents the results of these tests, from a broad generalized comparison of the three systems, to more specific questions about what types of queries yield what results, and how they behave under a simulated load of parallel requests.

Finally, Section 5 summarizes the advantages and disadvantages of each system, relates these to their respective design choices and philosophies, and comments on future directions for work on this topic.

This thesis compares three distinct paradigms for systems that manage vector data, in particular those used in common web-based mapping services. Over the past few decades, as GIS software has evolved, these methodologies have fallen in and out of favor. Today some are considered new and experimental (distributed), others nearly forgotten (flat file), and yet others are the (possibly under-appreciated) workhorse of present-day GIS (relational database). Each of these is well-suited to some use cases, and poorly-suited to others. This thesis investigates that suitability.

2 Background

This thesis is concerned with *vector* data, often contrasted with *raster* (or bitmap) data. Raster data typically (but not necessarily) consists of a 2-dimensional image suitable for human viewing. Vector data is not an image, but a geometric representation of a spatial feature; the most common examples of geometric primitives would be a point, a line, and a polygon. Vector data requires less space to store than raster data, but often requires more processing in order to be used in a GIS application. [32, pp. 35,193]

Uses of this data include visualizing in a desktop GIS application, display in a web-based application, or more complex analysis of geospatial data such as routing. As GIS applications become increasingly web-based, vector data management has followed. The flat files of vector data used in desktop GIS are typically replaced by relational databases in web applications. According to Sample and Ioup,

> There are two primary methods of storing vector data for tiling: database storage and file system storage. Database storage of vector data is more common than file storage when the data is to be retrieved using geospatial queries. File storage is more commonly used for archival and distribution of vector data as fixed data sets. [32, p. 196]

The newer distributed-cloud key-value stores introduced in recent years[1] could also be considered a type of "database" storage within that dichotomy, but since they introduce many new advantages and disadvantages, the distributed-cloud model will be considered a distinct third paradigm here.

The authors go on to describe a file format which, perhaps counter-intuitively, achieves some of the index and query advantages of a database. This idea would become the Vector Cluster format, detailed in Section 2.3.

Prior to that, in the following two sub-sections, the OGC data model and a common implementation of it, PostGIS, will be described.

Later in this section, the recently popular distributed-cloud approaches will be described, with a focus on GeoMesa. Then, more information is presented on the OSM data set, and finally the growing need for privacy and security features in geospatial data storage systems.

2.1 Open Geospatial Consortium data model and services

Much recent work in geospatial computing utilizes some part of the standards defined by the Open Geospatial Consortium (OGC), from basic data models and types (such as Simple Features) to file formats/markup languages (such as KML[2]) to widely-used publishing services (such as WMS, WMTS, WFS, WPS[3]) and others.

The OGC defines open standards for data and services to ensure "geospatial interoperability". With this goal in mind, the OGC standards organization began as a collaboration of government,

[1]At the time of [32]'s publication in 2010, new distributed-cloud frameworks did yet not support geospatial data, so that work only addresses Hadoop as an ill-suited platform for processing raster tiles, not vector data; that would be made possible by the advent of GeoMesa and similar systems in later years.

[2]KML stands for Keyhole Markup Language, popularized by Google Earth, a software package purchased by Google in 2004 and subsequently distributed at no charge on the Internet.

[3]These each stand for Web {Map, Map Tile, Feature, Processing} Service.

academic, and private-sector members of the GIS community in the early 1990s. Much of the effort's origins can be traced farther back to the GRASS GIS community which grew around the seminal GRASS (Geographic Resources Analysis Support System) software suite created by the U.S. Army Corps of Engineers in the early 1980s.[8]

The OGC standard of interest in this work is the *Simple Feature*, a model for the storage and access of geographic information. [7] This industry standard went on to become an international standard: ISO 19125. [23] The standard outlines a model for storing geometries, attributes, and spatial reference systems, specifying names and types for data. Both GeoMesa and PostGIS implement this standard. Vector Cluster does not.

Software described herein uses the Java implementation of the `SimpleFeature`[4] and other standards from OGC's Geotools version 11.2. OGC standards also provide standardized abstractions for common access methods and types, such as `DataStore`, `FeatureStore`, and `Query`.

2.2 Spatial data in a Relational database

Before considering two radically diverging paradigms for geospatial data storage (flat files and distributed key-value cluster), it is worth examining what could be considered a middle ground: the relational database (or RDBMS). This is certainly the most widely used of the three methods considered, and therefore is well studied.

Two components together make up the RDBMS geodata software which will be tested: PostgreSQL, a very well known and mature general-purpose RDBMS; and PostGIS, an extension for geospatial data.

Other relational databases support spatial indexes and queries. Oracle has produced a spatial option for its relational database since the 1990s, known as "Spatial Data Option", then "Oracle Spatial", and most recently "Oracle Spatial and Graph". More recently, MySQL added extensions for spatial data, following the OGC specitications, but this is not a mature or widely-used system. Only PostgreSQL/PostGIS will be tested here.

2.2.1 PostgreSQL

PostgreSQL is used here as exemplar of a traditional relational database. Created in 1986 by Stonebraker at the University of California at Berkeley (UCB), PostgreSQL is the successor to Ingres[35] (implemented in 1975-1977, also at UCB) and has been widely influential in the design of relational databases.

PostgreSQL has traditionally been used as a single-node database server, serving requests from one or many clients. These clients may be software running on the same host, or connecting from other systems over a network.

PostgreSQL does have some support for clustering, but this is usually in the form of simple replication for the purposes of load balancing and high availability.[10] Historically the PostgreSQL project policy was to avoid supporting replication in the core of the database, and instead to encourage third parties to develop competing approaches as add-ons. But starting in 2008, the PostgreSQL core team did introduce simple replication support into the core project. [24]

[4]Referred to in Java as `org.opengis.feature.simple.SimpleFeature`

2.2.2 PostGIS

PostGIS is an extension to PostgreSQL which adds support for geospatial data and analysis.[6] It is an open-source project under the Open Source Geospatial Foundation [36] (OSGeo) and independent from the PostgreSQL project. Despite yet another similar name, OSGeo is distinct and unrelated to OGC and OSM. However each of these organizations contribute to the same ecosystem of Free Open Source Software for Geospatial (FOSS4G), which is also the name of an annual conference organized by OSGeo.[28]

PostGIS uses the standard OGC Simple Features for GIS objects[5] as well as functions that operate over them, and OGC-compliant metadata on Spatial Reference Systems (SRS). The PostGIS spatial index is an R-tree over GiST (Generalized Search Tree). [6]

2.3 Spatial data in an indexed flat file

The oldest method of storing geospatial data is the simple “flat” file. In this context, a flat file is not necessarily devoid of any structure, but is distinguished by the following properties:

- consists of a single file (can be easily transferred)
- accessible through standard filesystem operations (and stored on virtually any device)
- does not require a server or other software devoted solely to handling requests

Strong proponents of relational databases may consider the flat file to be an obsolete idea, but some recently created geospatial data file types challenge that notion: Vector Cluster, and GeoPackage. They both push the boundaries of the notion “flat” by utilizing features from relational databases (especially spatial indexes), but still meet the three criteria given above. Perhaps geospatial data has come full circle and returned to consider the flat file again.

The newer format, GeoPackage, will be described briefly in Section 2.3.3; Vector Cluster will be examined with more detail, and will be used for the experiments in this work.

2.3.1 Vector Cluster History

Vector Cluster is a geospatial data storage format created by the Geospatial Computing Section at the United States Naval Research Laboratory (NRL). NRL’s products require large amounts of vector data with fast access times in order to support real-time on-the-fly generation of WMS tiles based on many data sources. [22]

Prior to the creation of Vector Cluster, many government projects stored vector data in a format called Vector Product Format (VPF). This format was developed by the National Geospatial-intelligence Agency (NGA) (previously known as National Imagery and Mapping Agency, and prior to that known as Defense Mapping Agency) in the late 1980s, and was later adopted into the Digital Geographic Exchange Standard (DIGEST) standard.[4] [3]

NRL researchers discovered that VPF was an unsuitable format for the modern, dynamic GIS system they were building. In order to render raster tiles on-demand from vector data, random-access queries must be practical. VPF had no support for geospatial queries and indices, a major

[5]Technically PostGIS implements a superset of the OGC Simple Features; PostGIS developers extended the model to support 3d and 4d coordinates.[6]

limitation. This means that, for example, in order to find objects that match a certain bounding box, every object must be scanned and compared to the bounding box. So any search required a full scan, greatly limiting the scalability of any GIS service.

At the time that NRL hit this limitation in the late 2000s, they also found the query performance for geospatial data in relational databases to be poor. More importantly, NRL needed a file format without the additional requirements of an RDBMS.

Vector Cluster was born from this need, and derives its name from another NRL product, the Tile Cluster[6]. A Vector Cluster consists of a single flat file with a format further detailed in Section 2.3.2.

The Vector Cluster format is notable for its simplicity, from which it derives much of its advantages. The format is also much more limited in what it supports, compared to other data storage approaches, as it was tailored to solve a narrowly defined problem.

NRL Geospatial Computing needed a way for clients to query a large set of vector data to dynamically generate maps on demand. These clients may not be strongly connected to a network, for example those in underwater craft. The client programs may also be running on smaller systems, such as those running the Android mobile operating system. Vector Cluster supports these use cases.

Vector Cluster does not support many other features that would be expected in something like a database. Vector Clusters can not edit or delete records, it is essentially a read-only format[7]. Updates to map data are performed periodically by regenerating the vector cluster entirely. One example given is the Digital Nautical Chart (DNC) data, from which a new Vector Cluster is generated every two weeks. [22]

Vector Cluster has its own complete API for vector geometries, features, queries, and more. It does not use any of OGC's GeoAPI, but implements much of the functionality independently. One difference is that OGC Simple Feature uses a fixed attribute schema, whereas NRL's Feature type can be schema-less. Other than that difference, much of the Vector Cluster API duplicates OGC's GeoAPI.

The proponents of Vector Cluster's simple flat-file design argue that the guarantees commonly provided by relational databases (atomicity, consistency, isolation, and durability) and related features (locking, rollbacks) are not useful for geospatial vector data when generating map tiles. By eliminating these features, the goal is to achieve a more efficiency for the more frequent use: reading data using a spatial query.[32]

The Vector Cluster format is in production use in several government contexts. The public Geospatial Computing Tile Server[8] uses vector clusters as its vector data source when rendering tiles. [22]

[6]The tile cluster is also a flat-file approach, but for storing raster tile data. It is designed to work around filesystem limitations with regard to many small files.

[7]Robert Owens of NRL says that it would be possible to add edit/delete support to the Vector Cluster library, but this has never been implemented.

[8]`http://geoint.nrlssc.navy.mil/`

2.3.2 Vector Cluster Format

While a Vector Cluster is a single file, it is well-structured and indexed for efficient access.[9] A Vector Cluster can contain one or many feature layers, which can be found quickly by way of a B+ Tree index on the layers.

Each layer has its own header and is independently indexed. The spatial index is a 2-dimensional R* Tree. Accordingly, nodes in the tree are ordered by Minimum Bounding Rectangle (MBR). The Vector Cluster can support attributes with a schema, or use a schema-less key/value pair system. [22]

2.3.3 GeoPackage

GeoPackage (defined by OGC in 2014) is a newer file format with very similar goals to Vector Cluster. It aims to provide a single file suitable for disconnected mobile devices, and as an interchange format in a heterogenous software environment. GeoPackage is "serverless", with clients accessing the file directly (using a software library). And like Vector Cluster, it also uses an R Tree as a spatial index. [27]

One notable difference is that, despite the simple single-file, serverless model, GeoPackage does consider itself an RDBMS with the transactional guarantees that typically carries:

> all changes to data in the container are Atomic, Consistent, Isolated, and Durable (ACID) despite program crashes, operating system crashes, and power failures. [27]

These are the same guarantees eschewed by the Vector Cluster designers.

It may seem surprising at first: a simple file format which also promises database-level guarantees and accessibility. It turns out GeoPackage is based on a non-geospatial file format which provides both of these things: SQLite. It may not be as conspicuous as PostgreSQL, but SQLite is actually used more broadly. Its creators make a convincing argument that it is "Most Widely Deployed and Used Database Engine", due to its common use as a configuration file format in widely used[10] software such as the browsers Firefox, Chrome and Safari and even the operating systems Windows 10, Mac OS, and Android.[34]

This new file format is quite promising and is being widely adopted, from the open-source library GDAL, to commercial ArcGIS suite from Esri, to government software from the National Geospatial-Intelligence Agency (NGA). GeoPackage supports both vector and raster data.

Flat-file purists may not be enamored with GeoPackage due to its use of SQLite, but it delivers the desired flat-file features: single file, serverless, indexed; and unlike Vector Cluster, GeoPackage is cross-platform compatible with many types of GIS software. The performance of GeoPackage is not yet well studied, however, and will not be tested in this work.

2.4 Spatial data in a distributed-cloud key-value store

The advent and publication of the Google File System [16] (GFS) in 2003 and the related MapReduce computation model[9] in 2004 has inspired the development of distributed systems and large

[9] Arguably the index means the file is no longer "flat" — but it is still far simpler than the other methods studied, so relatively speaking, "flat" (as the term is used here) would still apply.

[10] This remarkably wide use is made possible because SQLite is dedicated to the public domain and can be embedded in any program.

scale computation frameworks which are now used widely. The most prominent project inspired by those publications is Hadoop [17], an open-source software framework under the Apache Software Foundation, with major contributions from engineers at Yahoo Inc.[11] and other large web-based organizations. Although research papers describing the software have been released to the public, GFS and MapReduce[12] are proprietary software not available outside of Google[13]. Hadoop and related projects are open-source and freely available, so despite being a later arrival, they see more use in both research and production.

Many "big data" oriented efforts utilize the Hadoop framework. Many software systems have been built atop the Hadoop framework, most of which will not be named here. Two prominent examples are Hive, a data warehouse system, and HBase, a distributed key-value store. HBase has been used in some geospatial computing work, but will not be used here.

The Hadoop-based datastore used in this work is Accumulo, which is extended for spatio-temporal data by GeoMesa. These will each be detailed in this section. First, the layer which lies under these systems, Hadoop, will be described.

2.4.1 Distributed computing framework: Hadoop

At it's simplest, Hadoop can be broken down into two key components: HDFS and YARN (a MapReduce implementation).

HDFS is the Hadoop Filesystem, a distributed filesystem over many nodes, which provides fault tolerance, parallel access and load balancing. HDFS is closely paired with the MapReduce framework (the 2^{nd} generation of which is called YARN). This tight relationship between distributed data storage and distributed computational power lies at the core of Hadoop's large-scale data analytics design.

When building distributed systems at a large enough scale, hardware failures become a regular occurrence and must be accounted for in the system design. The GFS design (and thus Hadoop) have built-in redundancy as well as the ability to add and remove nodes dynamically.

While it is possible for HDFS and YARN to run on a single computer, doing so negates any benefits the distributed-computing framework offers and is only done for testing purposes. At their core, HDFS and YARN are designed to run on a large number of (possibly low-cost, commodity) computers, referred to as "nodes". Some of these nodes include:

- Name Nodes, which manage and provide HDFS filesystem metadata
- Data Nodes, which store HDFS data blocks
- Yet Another Resource Negotiator (YARN): tracks resources and orchestrates work (such as MapReduce jobs) across many nodes
- Zookeeper (similar to the Paxos algorithm): manages configuration, naming, and provides synchronization

[11] Yahoo and Google are both Internet advertising firms based in California.

[12] Originally MapReduce referred to Google's proprietary implementation, but now the term is used generically to refer to computation using that model.

[13] See footnote 11.

2.4.2 Key-value store: Accumulo

Accumulo is a distributed key-value datastore based on the BigTable design from Google [33]. As with GFS and MapReduce mentioned above, the publication of the BigTable paper [5] describes proprietary software available only within Google, but has spurred new developments in open source software, especially those under the Apache Software Foundation.

An Apache project since 2011, Accumulo splits tables up into "tablets" of contiguous, sorted rows which are stored in HDFS, which provides data redundancy and high throughput I/O. [33] Accumulo also extends the BigTable design in two significant ways:

- Column visibility: this optional value restricts access to each individual cell based on the authentication tokens specified. Using this value, Accumulo can offer fine-grained data security/privacy capabilities with this cell-level security label.[14]
- Iterators: similar to reducers in MapReduce frameworks, iterators allow for server-side processing. They are typically used to filter and transform data, possibly chained or in parallel across many tablet servers [33] [15]

Accumulo's multidimensional key includes the visibility feature for security, as well as various other fields. Table 1 shows this unique key format, which offers notable advantages. In addition to the fine-grained access-controls, the timestamp value in the key allows for temporal filtering on the server side using iterators.

Key					Value
Row ID	Column Family	Column Qualifier	Column Visibility	Timestamp	

Table 1: Accumulo's key/value structure[15]

Unlike searches on a relational database, which can utilize the index from multiple columns to expedite results, in a "NoSQL"[14] key/value store such as Accumulo, only one index is built on the lexicographically-ordered records.

2.4.3 Spatiotemporal index: GeoMesa

GeoMesa enables indexing and querying of spatiotemporal data on Accumulo, similar to the relationship between PostGIS and PostgreSQL. Like PostGIS, GeoMesa is not a standalone piece of software, but a library added-on to Accumulo[15] to enable support for spatiotemporal data.

GeoMesa uses and extends the OGC SimpleFeature data model and supports spatiotemporal indexing (within Accumulo's single key) by interleaving a record's geometry with its associated datetime string. This index is then split up between the three components of Accumulo's key structure: Row ID, Column Family, and Column Qualifier (as shown in Table 1). [15]

[14]This term originally referred to non-relational databases, but more recently has been associated with the phrase "not only SQL".

[15]GeoMesa is designed to work on other key/value stores such as HBase. But Accumulo is best supported; support for HBase is "rather new" and incomplete (as of October 2015), according to author Jim Hughes.[20]

2.4.4 Other approaches to geospatial computing on Hadoop

GeoMesa is but one of many approaches to spatial data indexing on the Hadoop framework. Whitman et al. reviewed GeoMesa and other Hadoop-based spatial data approaches before proceeding with implementing a Point Matrix Region (PMR) quadtree index with support for *range* and *k nearest node* (k-NN) spatial queries. GeoMesa does not support k-NN queries. Whitman et al.'s implementation does not use Accumulo to split and store data, but uses a MapReduce job to build the quadtree. One advantage they claim is that a MapReduce job is *not* required for access, providing "efficient, random-access queries" because "neither a full table scan, nor any MapReduce overhead is incurred when searching".[39]

Hadoop-GIS/RESQUE [1] is built on Hive, a Hadoop-based package supporting SQL-like queries on spatial data, as opposed to Accumulo's NoSQL key/value approach. Hadoop-GIS was motivated by medical imaging uses, supports range and self-join queries, and employs an R*-tree.

SpatialHadoop is built on Hadoop MapReduce, uses R+ tree indexing, and supports range, k-NN, and more computational geometry operations. Unlike Whitman et al.'s approach, SpatialHadoop does require a MapReduce job for each query. Eldawy and Mokbel's work includes a MapReduce extractor for OpenStreetMap data. [12]

2.5 The OpenStreetMap dataset

OpenStreetMap[16] (OSM) is a notable project that collects, stores, and displays map data contributed by users ("crowd-sourced").[19] Founded in 2004 in the UK, OSM's mission is to create map data available to all under a Free and Open license, powered by free software and by a community of volunteer editors. It has grown into a complete and vibrant mapping platform with global coverage and, in some locations, very rich features. Often considered a geographic analog of Wikipedia, the Free Encyclopedia, OSM introduced a novel approach to the collection of map data. Geospatial data collection was a domain that was previously limited to professional surveyors. OSM, enabled by widely available an inexpensive GPS and computing equipment, moved geodata collection into the hands of the public. Goodchild in 2007 dubbed this broader phenomenon Volunteered Geographic Information (VGI).[18]

OpenStreetMap is not only unique in how it collects its crowd-sourced data, but also in the infrastructure for storing and displaying the data. The OSM database and software stack were designed by newcomers to the geospatial computing world who paid little attention to the decades-long legacy of GIS. In fact, Haklay describes an outright rejection of this legacy:

> OSM's developers deliberately steered away from using existing standards for geographical information from standard bodies such as the Open Geospatial Consortium (OGC) — for example, its WMS standard. They felt that most such tools and standards are hard to use and maintain [...] and a lack of adaptability of OGC-compliant software packages to support wiki-style behavior. [19, pp. 14-15]

This break from the GIS past freed the OSM project to innovate with a completely free-form database: there is no fixed schema of allowable attributes, any key/value pair (called "tags" in OSM parlance) is valid. This schemaless design is similar to that of NoSQL storage, described in

[16]Despite the similar name, OpenStreetMap and the Open Geospatial Consortium are not related.

Section 2.4.2. OSM was not built with NoSQL in mind, however; this schemaless design predates the rise of today's NoSQL systems. In fact, OSM's tag system is quite easily used in PostgreSQL thanks to the Hstore data type[26]. The schemaless freeform "tagging" system was the perfect fit for a project composed of many disparate volunteers, upon whom no fixed schema or strict rules would be applied. OSM's contributors and users may not necessarily share a common language or culture, so they are not forced into a shared data schema either. OSM's tagging conventions were able to grow organically in ways that could not have been predicted by a fixed schema ahead of time; in that way, this new way of creating geodata is paired with a new data model.

OSM also has no "layers" in the traditional GIS sense[17]. This also frees the OSM users from a defined schema of layers. Yet in other ways, OSM's stark break from the GIS legacy can be disadvantageous: OSM uses "nodes", "ways", and "relations" which mostly duplicate the points, lines and polygons of traditional GIS. It is more than a semantic difference, as a way can be either a line or a polygon ("closed way"). OSM's divergence from the past can make use of data in traditional GIS software cumbersome.

While the Open Geospatial Consortium grew out of the longstanding government-academic-corporate GIS community, OpenStreetMap finds its roots and inspiration elsewhere: in the Internet-based movements for free software (perhaps most associated with the GNU project and the Linux kernel) and free information (exemplified by Wikipedia)[18].

OSM's data is not only licensed in such a way to enable sharing and reuse; the project encourages use of the full dataset by producing daily complete copies of the worldwide dataset for convenient download at the website `http://planet.openstreetmap.org/`. As of March 2016, the full dataset with current data is 48GB of bzip2-compressed XML. The full history is also available, at 74GB of bzip2-compressed XML.

OSM's database stores data in the WGS84 (EPSG 4326) spatial reference system, which is what will be used throughout this work.

2.6 Privacy and Security in geospatial data

Recently Palmieri et. al. introduced the Spatial Bloom Filter (SBF)[29] which enables privacy-preserving computation of location information. A common use of personal location information is to determine if a user is in an area of interest to a service provider. Smartphone users would sacrifice privacy by providing their exact location to service providers, and service providers do not wish to provide their entire dataset to users. Palmieri et. al[29] provide two protocols for secure multi-party computation using the SBF.

With this and other work, security and privacy concerns are increasingly being used in the design and selection of geospatial data storage systems. GeoMesa, when used with Accumulo's fine-grained visibility key features, is the only example used here which addresses this need. Either the old systems will need to adapt to the future demand for privacy and security features in geospatial storage systems, or these systems will be replaced by those which do provide that capability.

[17]OSM does have a "layer" tag, but this is only used to distinguish what objects should be considered above or below other objects when rendered.

[18]It is interesting to note that Wikipedia was originally licensed under the GNU Free Documentation License (GFDL) until 2009, when it adopted the Creative Commons Attribution-ShareAlike License (CC BY-SA). OpenStreetMap was originally licensed under the CC BY-SA, but changed to the Open Database License (ODbL) in 2012.

3 Methodology

The methodology described below developed over the course of investigating the behavior of various geospatial data services. Early tests of Geomesa began by using it as a data store for Geoserver, a software server from the OGC for viewing and editing geospatial data from a multitude of sources. From there, GeoMesa support was added to NRL's vector server by creating a GeomesaFeatureProvider class. Measurements of this configuration were initially promising, but did not yield useful results for reasons to be explained below.

Regardless, these early experiences are worth examining as they shaped what would become the final experimental design. Problems with bottlenecks in irrelevant parts of the software stack motivated the development of focused, specific tests which could eliminate extraneous bottlenecks in order to test only the data store component. This experience will be detailed further in Section 3.3.

But prior any testing or measuring, the data store services themselves must be set up and deployed. Section 3.1 details this process.

3.1 Configuration and deployment of Geospatial data services

Each of the three systems under consideration were configured in simple, standard configurations similar or identical to those used by many common web-based tile services or GIS vector services.

3.1.1 Vector Cluster

This is by far the simplest data deployment process: copy a single file, to read via NRL's Java class, on the same system as the TestRunner. This is the typical use case for a vector cluster, which has been designed to minimize requirements and maximize efficiency. These design efforts have succeeded, making setup trivial, other than copying the large data file into place.

On the other hand, this ease in setup has a cost, in that essentially no public-available software supports the Vector Cluster, so time is required to modify or write software to utilize the format. The process of writing a class to support Vector Cluster surely took more time then setting up an existing software package.

3.1.2 GeoMesa/Accumulo/Hadoop platform

On the other end of the complexity spectrum, by far the most involved and intricate configuration process in preparing for this work was for GeoMesa. It is not a standalone piece of software, but a Java library used by Accumulo, which in turn requires a Hadoop cluster. The three major components used versions shown in Table 2.

Hadoop is the first component to put into place when building what will become the GeoMsea platform. Cloudera's distribution of Hadoop was used (denoted by the cdh in the version shown in Table 2). This includes core components of Hadoop: Name Nodes, Data Nodes, YARN managers, and Zookeeper nodes. See Section 2.4.1 for details on the functions of each of these components.

With a working Hadoop cluster up and running, Accumulo setup can proceed. This is a matter of building and deploying using standard documented processes, then adding proper Hadoop addresses and credentials to the Accumulo configuration. With these credentials, Accumulo will be able to access HDFS storage, which Accumulo depends on. With the software deployed and configured, Accumulo processes can be launched and accessed.

Finally, unlike Hadoop and Accumulo, GeoMesa does not have its own java process, but is merely a library for Accumulo to use. Once GeoMesa is built, the JAR file is deployed into `/opt/accumulo/lib/ext`, thus adding Geospatial support to Accumulo's distributed key/value store.

Hadoop	2.0.0-cdh4.6.0
Zookeeper	3.4.5-cdh4.6.0
Accumulo	1.5.1
Geomesa	1.1.0-rc.2

Table 2: Software versions used for cluster infrastructure

Five servers is not a typical size for a production Hadoop system. Usually the benefits of a cluster environment don't outweigh overhead until a certain scale is reached, often involving many servers. However this provides enough of a cluster to simulate the various roles played by nodes in the cluster: zookeeper nodes, HDFS namenodes (primary/secondary), HDFS datanodes, Accumulo tablet servers, Accumulo monitor node, Accumulo master node, Accumulo tracer node. Some of these overlapped, their distribution is given in Table 3. Since each physical system has 8 CPU cores and 32GB RAM (more detail is shown in Table 5), running multiple services on each is not expected to cause significant performance problems.

For historical reasons, these systems are numbered 03 through 07.

Service	Function	Server node				
		03	04	05	06	07
Hadoop	name node	✓				
	secondary name node		✓			
	data node	✓	✓	✓	✓	✓
	YARN resource manager	✓				
	YARN node manager	✓	✓	✓	✓	✓
Zookeeper			✓	✓	✓	
Accumulo	Master					✓
	Monitor					✓
	Garbage Collector				✓	
	tracer				✓	
	slaves	✓	✓	✓	✓	✓

Table 3: Service distribution across the available hardware nodes

3.1.3 PostgreSQL/PostGIS server

This project's "traditional" single-node PostgreSQL system was set up on a single Dell R415 system. Although PostgreSQL does have support for various replication and clustering add-ons, this is not the typical case, so PostgreSQL systems set up that way will fall outside the scope of this work.

This system runs PostgreSQL version 9.3 and PostGIS version 2.1, as provided by Ubuntu version 14.04.

3.2 Building data stores

3.2.1 Generating a Vector Cluster file

The vector cluster file used for testing was provided by collaborators at NRL-SSC, and was built with OpenStreetMap data from planet.openstreetmap.org. Since vector clusters are a write-once read-many format, this process must be manually repeated in order to update the map data with newer information.

After processing the `planet.osm` file, the resulting vector cluster `OSM_WORLD-0-0-0.vcluster` is 74GB. There are approximately 3 billion objects in OSM dataset.

3.2.2 Ingestion into GeoMesa

In order to make the comparison as unbiased as possible, the identical OpenStreetMap-derived data set was used, by reading through the entire `OSM_WORLD` vector cluster file and inserting each object into GeoMesa.

The NRL layer scheme was preserved, in order to hold that consistent with the Vector Cluster design[19]. There is a one to one mapping between layers in vcluster to tables in geomesa.

One difference between the way schemas work in GeoMesa and Vector Clusters must be accounted for. The Vector Cluster file uses NRL's feature type (referred to in Java as `mil.navy.nrlssc.commons.vectorData.api.features.Feature<Geometric>`, while GeoMesa uses the Open Geospatial Consortium's feature type (`org.opengis.feature.Feature`). The relevant difference here is that NRL-style features can support variable numbers of attributes, while OGC-style features must have attributes defined in the schema. When working with data from OpenStreetMap, highly variable attributes are a common situation, because of the free-form structure of OSM and due to variations in how millions of users add their ("crowd-sourced") data.

The solution to this disparity is to create an OGC schema with a single `OSMAttributes` field, which can be a string of varying length. Then the NRL attributes are concatenated into to a single string, with key/value pairs separated by `:::` characters. The conversion is simple and only happens during the data insertion, not during the critical path being tested in the next section. (The full SimpleFeatureType schema, including "schema-less" `OSMAttributes`, is shown in Table 4.)

When looking at storage requirements, due to the way data is replicated in the HDFS storage on which Accumulo and GeoMesa are built, the total disk space required is much higher. This is normal and expected for a distributed, cloud-based cluster system like GeoMesa, and is another (small) part of the higher cost associated.

[19]However, further testing revealed no significant performance change for individual queries when forgoing layers and using a single large table

SimpleFeatureType Metadata key	**Value**
SRS	EPSG:4326
Name	*layerName*
Attribute	**Type**
Type	String.class
geomesa_index_geometry	Geometry.class
OSMAttributes	String.class
Layer	String.class
geomesa_index_start_time	Date.class
geomesa_index_end_time	Date.class

Table 4: SimpleFeatureType schema used in GeoMesa

3.2.3 Insertion into PostgreSQL

Since the programs to populate the data for both PostGIS and GeoMesa are both written in Java, the schemas (and Java classes) used as our SimpleFeatureTypes are identical. Likewise, one PostGIS layer (and thus one PostgreSQL table) is created for each layer in the Vector Cluster.

The main difference from the GeoMesa insert is that instead of writing a Java object to a "FeatureStore", the testing software needed to generate SQL statements to insert the data into PostgreSQL. This is easily accomplished using the common JDBC API in Java.

Before inserting the data, it is necessary to create a database in PostgreSQL and then enable PostGIS with the statement: `CREATE EXTENSION postgis`

In addition to the GiST R-tree index, PostgreSQL provides a module for the `hstore` data type, which allows for schema-less key-value pairs with support for indexing and querying. This is crucial for storing and searching OpenStreetMap tags in a PostGIS database, as the schemaless key/values map perfectly onto the hstore type. Like PostGIS, `hstore` is not built-in to the standard PostgreSQL types, so it must be loaded on each database with `CREATE EXTENSION hstore;`.

However, the tests performed in these experiments were limited to spatial queries, not attribute queries, so while the hstore type was used when inserting OSM data, its features were not userful or relevant to this work.

Once the groundwork is laid by loading the extensions above, the bulk insertion can commence. The Java program `InsertFromVectorClusterIntoPostgres` makes a replica of the data stored in the `OSM_WORLD` Vector Cluster used in the experiment, by iterating through each feature of the file and writing it to the database with a SQL `INSERT` statement.

After all rows of data are inserted, the GiST index is created with: `CREATE INDEX [`*layername_idx*`] ON [`*layername*`] USING GIST ( geom );` for each feature layer. (While these GIS-style layers are not generally used for OSM's database, in order to maintain consistency with the Vector Cluster, the same layers/tables will be used here.)

After the insertion and indexing is complete, it is advisable to perform a `CLUSTER` operation to optimally reorder the data in the spatial index. However this step was not performed before the measurements described here were made. The `CLUSTER` command takes considerable time to run on large tables, but it does increase query performance somewhat.

The PostgreSQL DB for the `OSM_WORLD` data ends up consuming around 79GB of disk space on the database server. This is larger than the the 74GB Vector Cluster, so the various database structures and indices add an overhead of around 7% compared to the simpler flat file.

3.3 Early Experiments

This effort began by working to adapt existing web-based GIS services to utilize a novel storage service, GeoMesa. These services, which provide both raster tiles and vector data, traditionally would use a relational database (for example, PostgreSQL) with geospatial-aware indexing (PostGIS). The goal was to measure how the same service behaved when the backing storage system changed. In addition to testing against the new and highly complex GeoMesa cluster, a system with nearly opposite design philosophy was also tested: NRL's Vector Cluster format.

Once a working GeoMesa datastore was built, the first test was to use that datastore for tile serving using OGC's Geoserver (version 2.5.2), which can be modified to use GeoMesa backend. This was useful to verify the validity of the data store, but Geoserver is a large complex Java application which can be difficult to measure. A simpler visualization of the data was required to proceed.

NRL's vector server was a better system for testing, with a much simpler interface to the data store. Once this server was adapted to use a GeoMesa backend, it proved stable and useful. Initially, GeoMesa introduced considerable latency and performance cost. But was that cost worthwhile above a certain scale?

To address that question, jmeter was used to benchmark under heavy load. The drive was to answer these questions: at what scale does vector cluster "top out"? How much more scalability does a cluster like GeoMesa provide?

Using jmeter and a randomly-generated list of query bounding boxes (using similar methods as those described in Section 3.6), it quickly became clear that vector cluster performance was excellent up to a certain level of load. But when the simultaneous request load went above a certain level[20], performance fell off dramatically.

This seemed like a promising result, identifying an inflection point in a curve appeared to be a clear limit of scalability. However, upon further investigation, it turns out that the scalability limit was not in the vector cluster, but in the Tomcat engine handling the HTTP/WMS queries. Another bottleneck came into play before the component of interest was fully utilized.

This was an important lesson in testing and measuring complex multi-component services: drawing meaningful conclusions requires focusing narrowly on the specific part of a system, and eliminating other parts of the system as much as possible. Doubtless many researchers have experienced similar lessons with misleading results, but this experience bears repeating, and also informs the subsequent decisions made in this research effort's experimental design.

Tile servers are intricate collections of software engaged in a complex interaction with networks, buffers, caches, operating systems, hardware limitations, and much more. Figure 1 shows the parts of the system which testing revealed to influence or interfere with our ability to measure the back-end storage components: GeoMesa and Vector Cluster.

In order to make useful measurements of these systems and find real insights into geospatial data storage practices, it is necessary to eliminate the bottlenecks in other unrelated parts of the

[20] Approximately 1024 threads, on the desktop computer used for testing at this early stage.

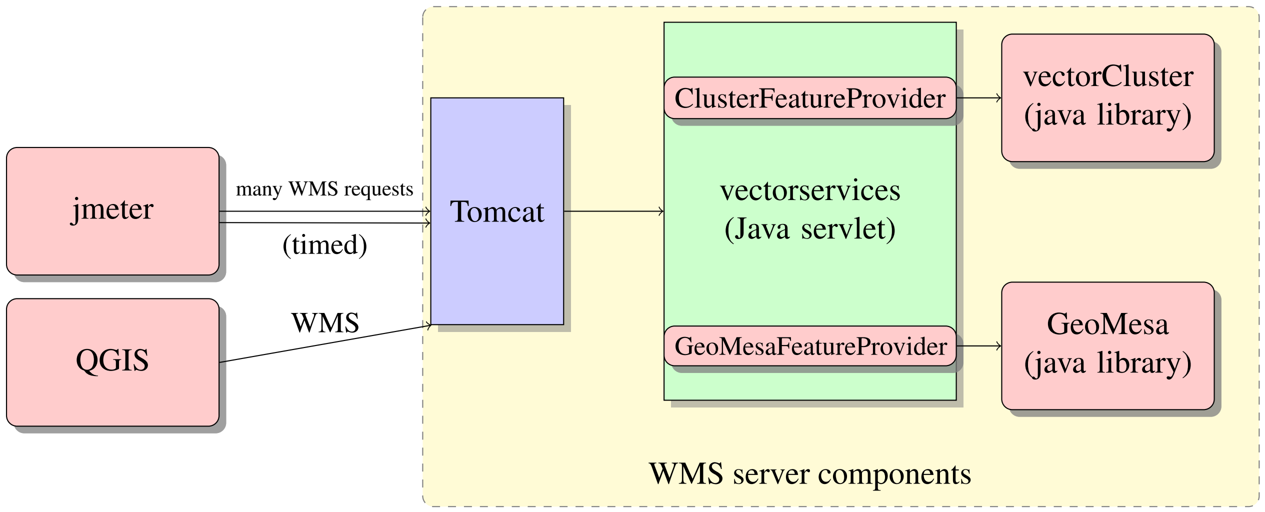

Figure 1: Early testing apparatus

system. This consideration is what motivated the development of a test harness, described in Section 3.4.

3.4 Refined experimental design

After adapting both NRL's vector server and Open Source Geospatial Foundation's Geoserver to use GeoMesa, performance testing began, but it was soon discovered that the bottleneck being measured was not the data back-end, but in the web-front end.

It was then decided to remove all other parts of the tile generation process and focus on isolating and measuring only the query portion, the part of the process where the choice of geospatial storage paradigm is felt.

A new testing harness was written in Java, with a TestRunner class, a QueryTest interface, and three classes implementing QueryTest: VectorClusterTest, GeoMesaTest, and PostGISTest. These classes each implement a standard `query()` method, such that each can be measured under identical conditions.

Full tile rendering was simulated by querying for all features within a bounding box, for all layers. This results in many queries (turns out to be a bad design decision for a tile service)...

A note on filesystem caching: the Linux kernel's Virtual Filesystem (VFS) uses an in-memory cache to hold frequently accessed data structures from the storage medium: disk pages, directory entries, and inodes. [37] The page cache certainly plays a role in how any storage system performs, as memory accesses are orders of magnitude faster than disk access. In the course of this investigation, tests were run under both "cold" and "warm" cache conditions (the "cold" cache being simulated by using the familiar filesystem interface `/proc/sys/vm/drop_caches` described in [37]). As expected, the differences are dramatic. However the "cold" cache data was set aside after considering that a busy tileserver (being simulated here) would only be under "cold cache" conditions in extremely rare cases (after a reboot, for example). So the "warm cache" case ends up

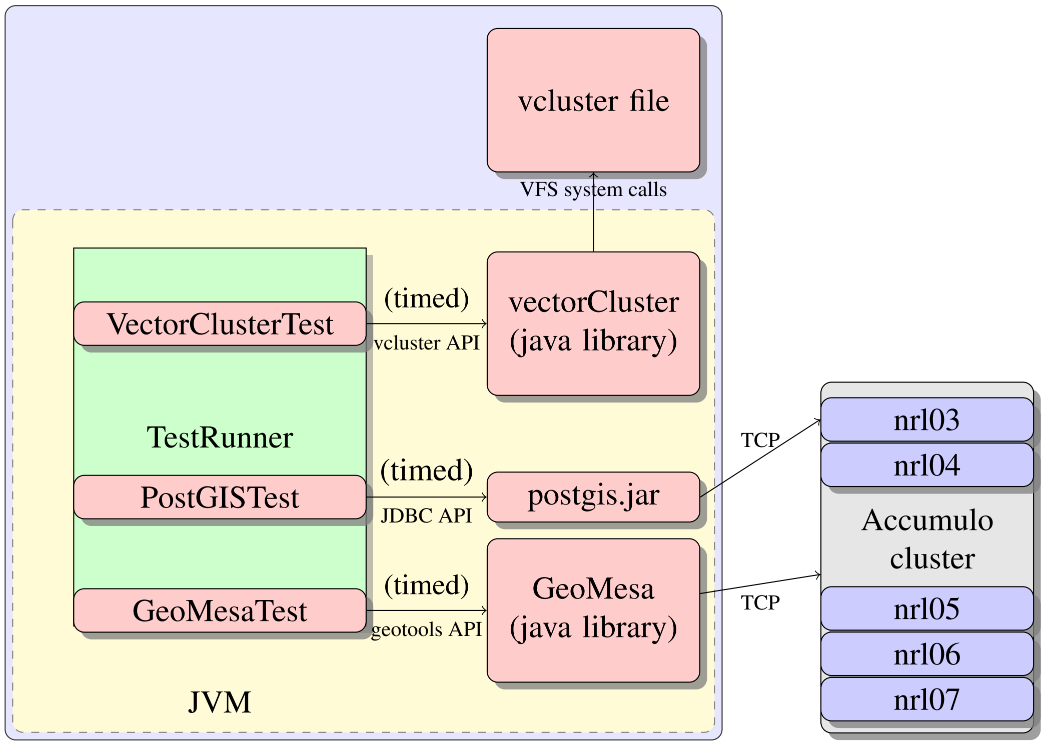

Figure 2: Refined testing apparatus

being closer to the real conditions of an active tile server.[21]

3.5 Measurements

The initialization time for each QueryTest object (GeoMesaTest, VectorClusterTest, or PostGIS-Test) is performed before the stopwatch timer begins. The stopwatch begins when it is time to call the `query()` method and stops immediately after `query()` returns.

Each of the three models tested use a form of lazy evaluation for the results returned from queries. After a query is performed, an iterable Collection[22] is returned but this object may not be fully populated. So in order to truly test the time needed to fetch data from storage, the test harness must iterate through the collection. This effectively "touches" the data, forcing the evaluation of each object in the collection.[23]

3.6 Simulating tile server workload using randomly chosen bounding boxes

A typical GIS database will contain many layers of vector data. Tile servers generate raster tiles for display by querying the GIS database for vector data, often from many (or all) layers. For a human

[21]However, without simulated traffic in geographically disparate areas, the test apparatus may gain an unrealistically high advantage from the page cache. This issue is addressed by the use of the random bounding boxes described in Section 3.6

[22]`https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html`

[23]This is analogous to what functional programmers call "forcing a thunk".

audience, a layer of data (for example, port facilities) is much more meaningful with context (for example, combining port facilities with coastlines and waterways).

In the dataset used for this experiment, 191 layers have been generated from OpenStreetMap data. As noted earlier, OSM's database does not use feature layers. But it is not unusual for GIS data consumers to organize data extracted from OSM into GIS layers, as NRL has in their Vector Cluster.

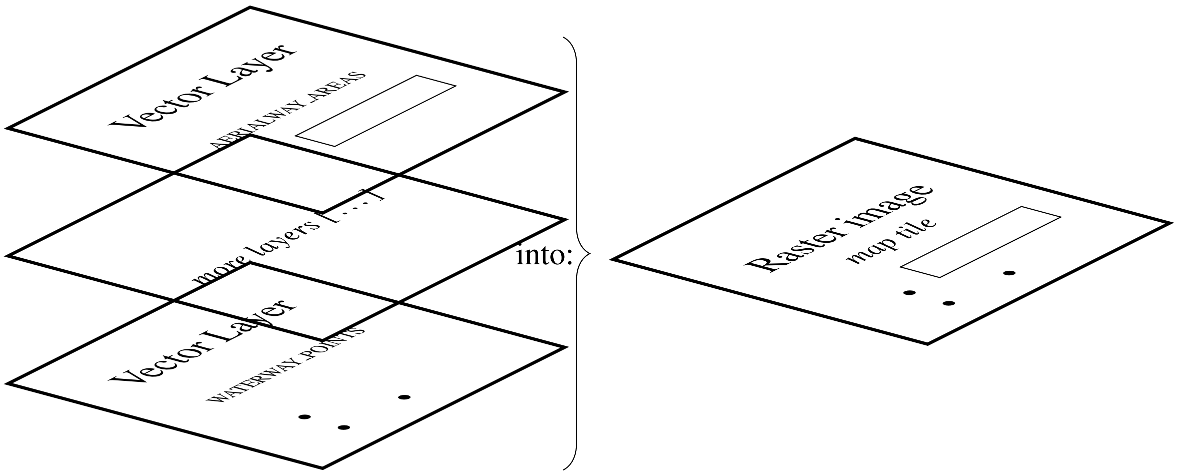

Figure 3: Example of building a tile from queries of many layers

An approximation of the process of generating a raster tile from many vector layers is illustrated in Figure 3.

In order to simulate the workload of a busy map server serving many simultaneous users spread across a large area, a common approach is to scatter requests around a large geographic area. Using an existing tool, simulated map query sample sets (referred to below as "bounding boxes") were generated for use in testing.

These bounding boxes were generated using `wms_request.py` [24], a python program originally written by Frank Warmerdam for the Open Source Geospatial Foundation's 2009 FOSS4G conference. At that conference, GeoServer, MapServer, and ArcGIS Server competed for the title of "Fastest Web Map Server (WMS)". [13]

In each test case, one of two sets of 2000 randomly-generated bounding boxes are used. One set of 2000 bounding boxes is across a wide area (described in Section 3.6.1, the other in a smaller, feature-dense region (described in Section 3.6.2).

3.6.1 Within contiguous North America (all-layer tests)

For our tests that generated full-tiles from 191 layers, bounding boxes were used which spread across a contiguous section of North America. The sparseness of features across the area is less of a problem when all layers are queried; empty tiles are less common. This also allows us to simulate more closely a real workload, where queries come in for disparate geographical areas so a server cannot simply cache one region.

[24] https://svn.osgeo.org/osgeo/foss4g/benchmarking/wms/2010/scripts/wms_request.py

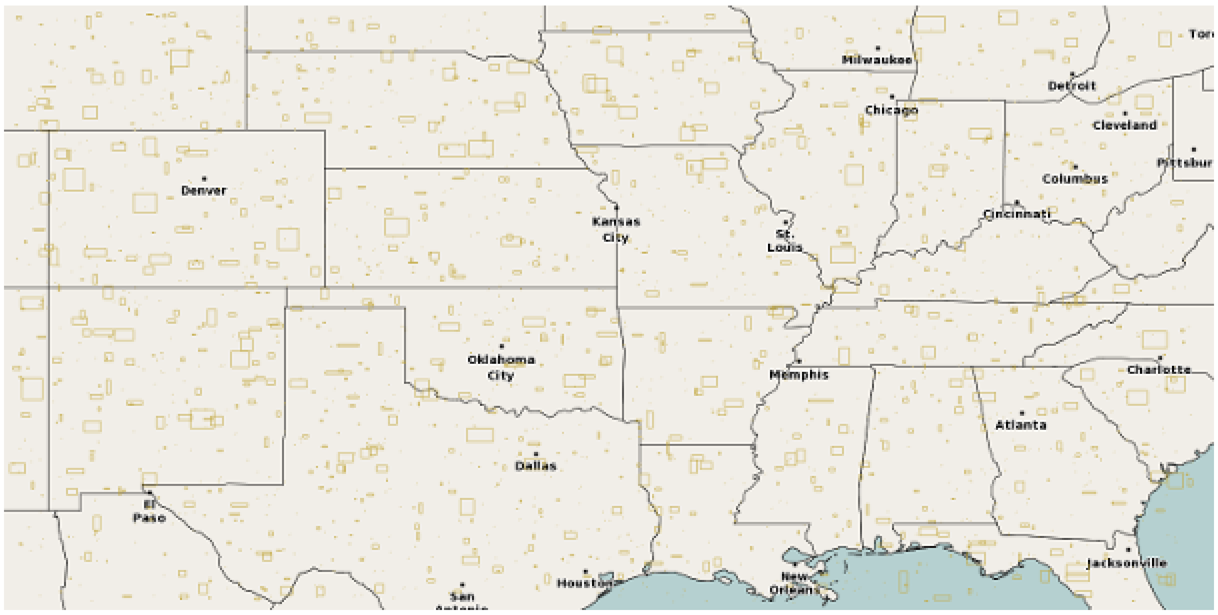

Figure 4: Randomly generated bounding boxes across continental North America

3.6.2 Within feature-dense areas (single-layer tests)

In performing tests across North America, it was observed that many randomly generated bounding boxes contained few features, and queries on many layers would yield zero results. Given the vast size and relatively low density of human settlement in North America, it is not surprising that many randomly chosen bounding boxes will end up in a feature-poor region.

The problem of frequent zero-result bounding boxes was addressed by limiting single-layer tests to areas known to be feature-rich. Originally chosen for experiments were: Berlin, Germany, very heavily mapped by OSM volunteers; New Orleans, USA, a much smaller and less dense city; and San Jose, California, roughly somewhere in between the first two cities in terms of size and density. Chosen for presentation in this document are results from the San Jose queries.

Figure 5 shows a visualization of the 500 bounding boxes in and around San Jose, California, used in the single-layer testing. Results from these tests are presented in Sections 4.2.1 and 4.2.2.

3.7 Testing with multithreading (all-layer tests)

The TestRunner class created for this research effort also supports querying all layers within a given bounding box, using a configurable number of simultaneous threads.

Java, as is common for a widely-used, mature language and runtime environment, can be parallelized using a number of different methods. Some of these include standard components of the Java^{TM} Platform, such as `Timer`, `ExecutorService`, and `ForkJoinPool`. There are also third-party libraries such as ParallelJava and Javolution.

The mechanism chosen to parallelize these tests is the `ExecutorService` (cite oracle docs java.util.concurrent.ExecutorService). In 2013 Nazario Irizarry Jr. of MITRE Corporation studied[25] each of the mechanisms listed here and concluded that `ExecutorService` performs well but is

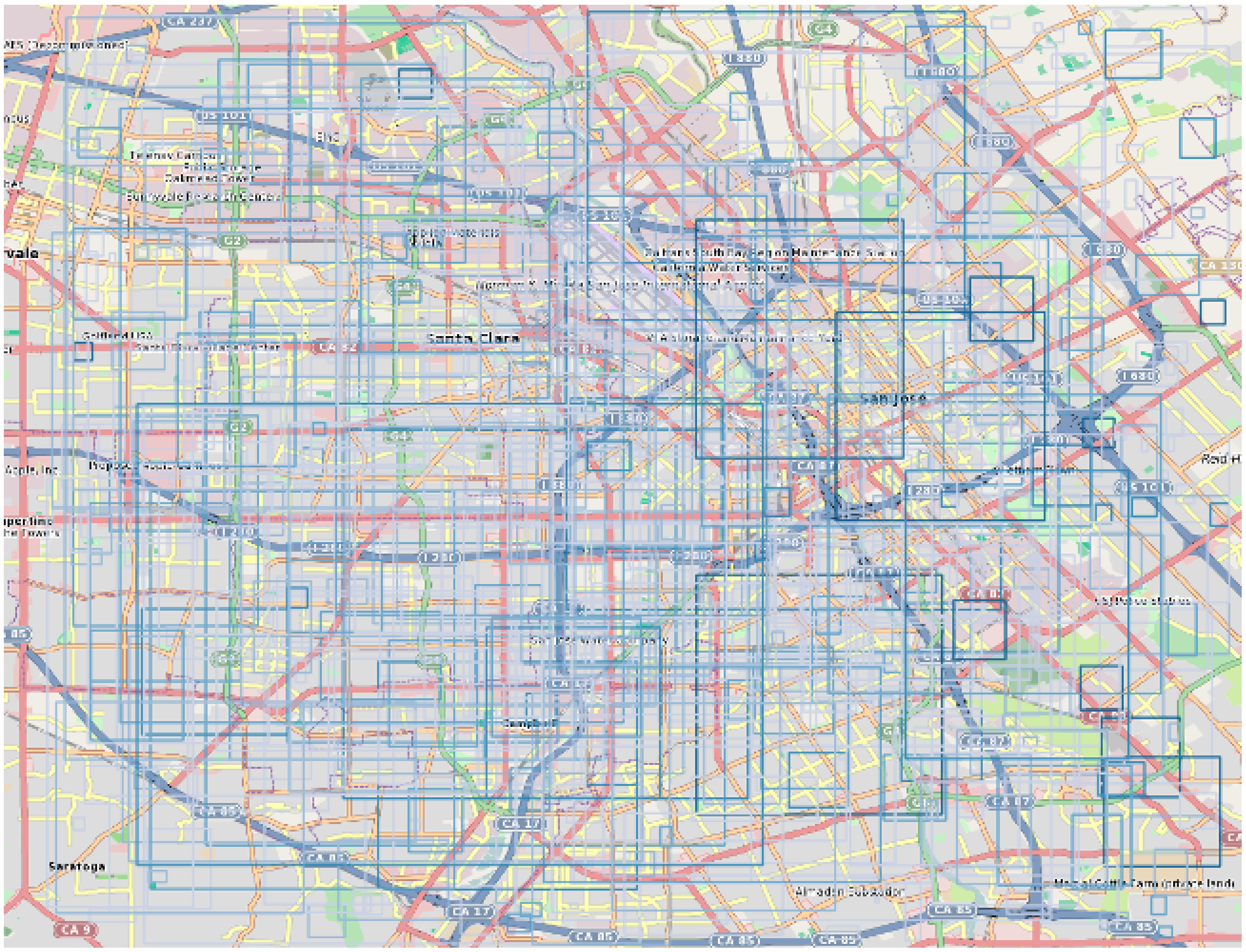
Figure 5: Randomly generated bounding boxes in and around San Jose, California

neither the fastest nor the slowest of those tested.[25, pp. 3-19 to 3-22]

3.8 Hardware

To accommodate the experiments described above, a new geospatial computing cluster was built based on these needs. The Vector Cluster and PostGIS systems had simple hardware requirements, but the cluster on which GeoMesa tests would run required new hardware.

In order to create an environment for simulating use of a distributed Hadoop-Accumulo cloud-based GeoMesa service, a multi-node cluster was needed. A relatively small number of servers were acquired, installed, and dedicated to this task.

The experimental cluster consisted of five nodes, each a Dell server with the specifications given in Table 5.

The Hadoop/Accumulo cluster on which the GeoMesa tests ran consisted of these five nodes (with services distributed across them as described in Table 3). The PostgreSQL and vector cluster tests ran on a single node only, as they represent traditional non-distributed systems.

Make and Model	Dell® PowerEdge R415
CPU	Two 8-core AMD® Opteron™Processor 4386 (total 16 CPU cores)
RAM	32GB DDR3 (1600MHz) Synchronous Registered (Buffered)
Disk	Toshiba 1TB SATA HDD 7200 RPM
OS kernel	Linux 3.13.0 x86_64
OS system	Ubuntu 14.04 LTS

Table 5: Experimental hardware

4 Results

Tests from the custom, finely targeted test harness described in Section 3.4 yield data on the three storage systems under consideration, in order to provide observations on how each performs under varying workloads.

4.1 Generalized performance comparison

Before digging into details on how each system behaves under varying conditions, an overall picture of query performance is quite clear. GeoMesa, the distributed-cloud Accumulo-based system, differs greatly from the others in its design, and the resulting behavior of the system is quite different.

Vector Cluster and PostGIS are both more traditional designs, and their results lie much closer to one another. All of these are summarized in Table 6, which should be paired with Tables 7 and 8 in the next section when considering overall properties of each paradigm.

System	Mean query time (seconds)	standard deviation
GeoMesa	17.50	0.6014
Vector Cluster	1.526	0.0552
PostGIS	0.8024	0.0326

Table 6: Overall performance comparison: North America bboxes, full tile (all layers), 7 threads

These figures are extracted from the data based on running queries on all 191 layers (as defined in the original NRL vector cluster file), with 7 simultaneous threads performing queries in parallel. See Section 4.3 for more on the behavior of these systems when the number of threads is varied.

It bears repeating that this measurement is for a specific simulated task: a tile server (such as a WMS server) making queries on many layers in a GIS data store to retrieve vector data, to be used in generating a raster tile. This is a common task, but very different from other GIS tasks.

4.2 Which parameters impact query performance

Since it is clear that the three paradigms show great differences in query speed (with the workload used in these tests), it is more useful to look for insights in what the performance curve looks like for each system under different conditions.

By looking at the asymptotic behavior of each system, there is insight to be found in how each system scales, and what sort of tasks that system is suited to.

4.2.1 Size of area to be queried

First, for a set of queries over a single feature layer, the size of bounding boxes will be varied, and the impact this change has on query performance is examined.

It may seem intuitive that when searching two regions of the earth for objects (such as building polygons) that intersect these regions, searching the larger region would require more time than searching the smaller one. After all, would you expect “find all buildings in New Orleans” or “find all buildings in Texas” to take longer?

In the thought experiment above, the two regions (New Orleans and Texas) are disjoint and of vastly different size. To generalize this question, also included is the case where the larger region is a superset of the smaller (for example, New Orleans and Louisiana) as well as partially overlapping regions (New Orleans vs. all parts of Louisiana east of the Mississippi River).

It turns out that the intuition about this type of spatial search is wrong. The size of a bounding box does not significantly correlate with the time of a query. This is due to the properties of the R-tree and the fact that the distribution of objects like buildings across the planet is very non-uniform.

To continue the thought experiment, it might be considered trivial to find all buildings north of the 85th north parallel (there are none despite the large area) as opposed to finding all buildings in Manhattan (a small place with many buildings).

The very weak correlation between bounding box size and query performance is almost certainly due to the somewhat increased likelihood that a larger bounding box contains more objects to return. As shown in Section 4.2.2, the number of objects returned is a much more important parameter to consider.

In Figure 6, almost no relationship ($r^2 = 0.0087$) is seen between bounding box size and query performance, when applied to the GeoMesa cluster. This is consistent with what is expected from this type of search: GeoMesa uses a 1-dimensional lexicographically sorted geohash. A bounding box is translated to a set of partitions within the data, represented by prefixes in the key of each row in the Accumulo key-value store. In cases where the data is sparse, fetching all rows with a certain prefix will be fast, regardless of how large that physical area is within the query.

Figure 7 shows the same very weak relationship, with only slightly more effect on PostGIS than in GeoMesa.

And finally in Figure 8 for Vector Cluster, again very low significance. Since Vector Cluster is using an R*tree, it is not surprising to see similar behavior as the R*tree index used in PostGIS. This is also consistent with the understood properties of the R-tree: the depth and complexity of the tree to search is dependent not on physical area, but on the number of leaf nodes (features) within a minimum bouding rectangle (MBR) of arbitrary size.

However, the Vector Cluster data is slightly more significant ($r^2 = 0.02$) than the previous two. In the next section we will explore the way that removing network latency from consideration affects the results for Vector Cluster, the only method we tested that does not involve any network connections.

Overall, for each of the geospatial storage paradigms, bounding box size is not a significant factor in query performance. Since this type of query consists of traversing an R-tree, it can be

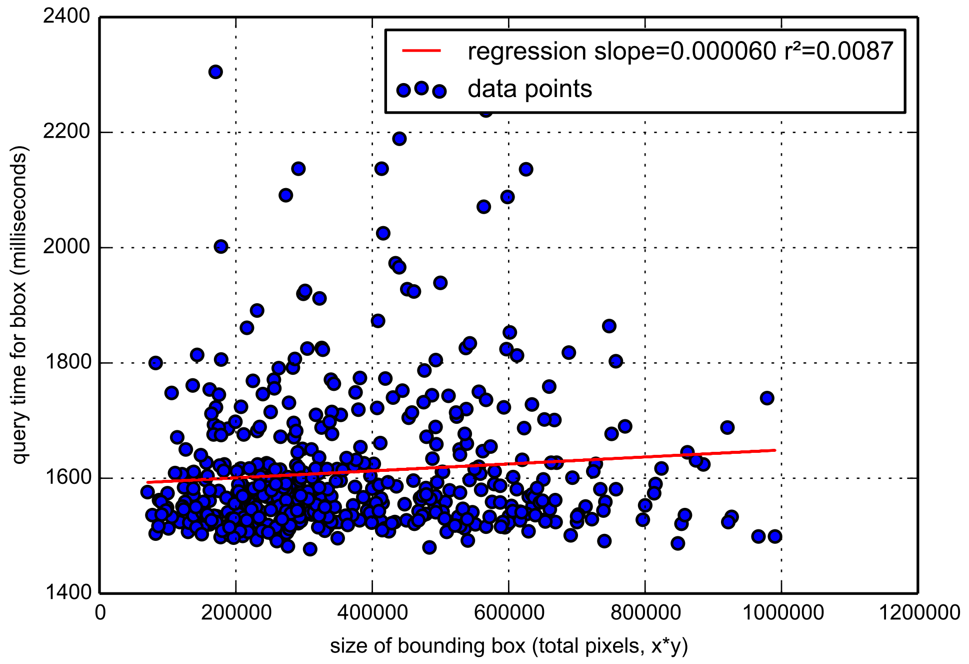

Figure 6: GeoMesa: query times for varying size bboxes (San Jose highway layer)

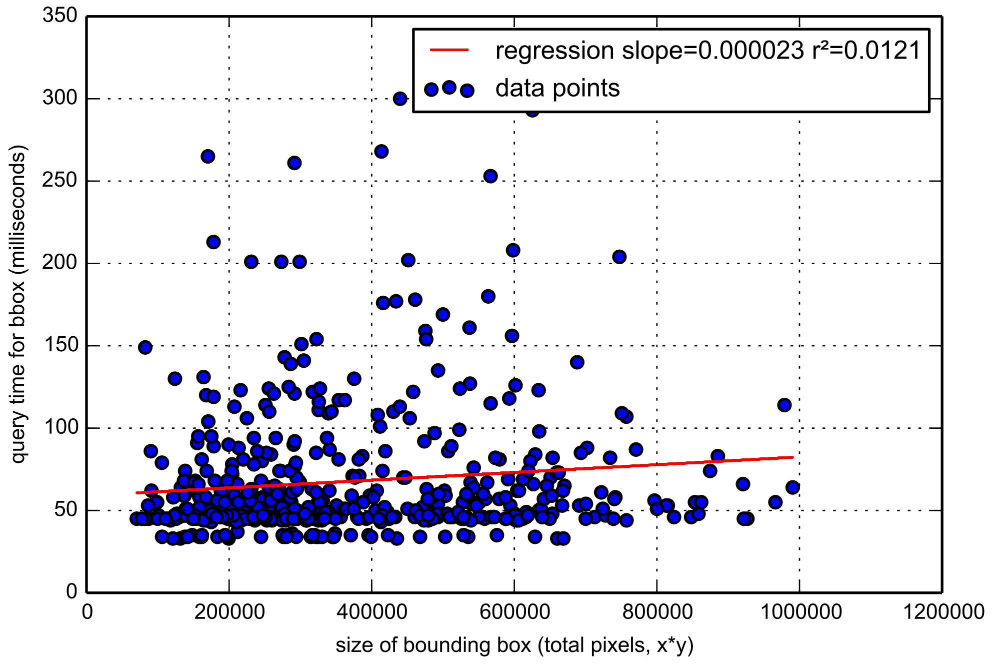

Figure 7: PostGIS: query times for varying size bboxes (San Jose highway layer)

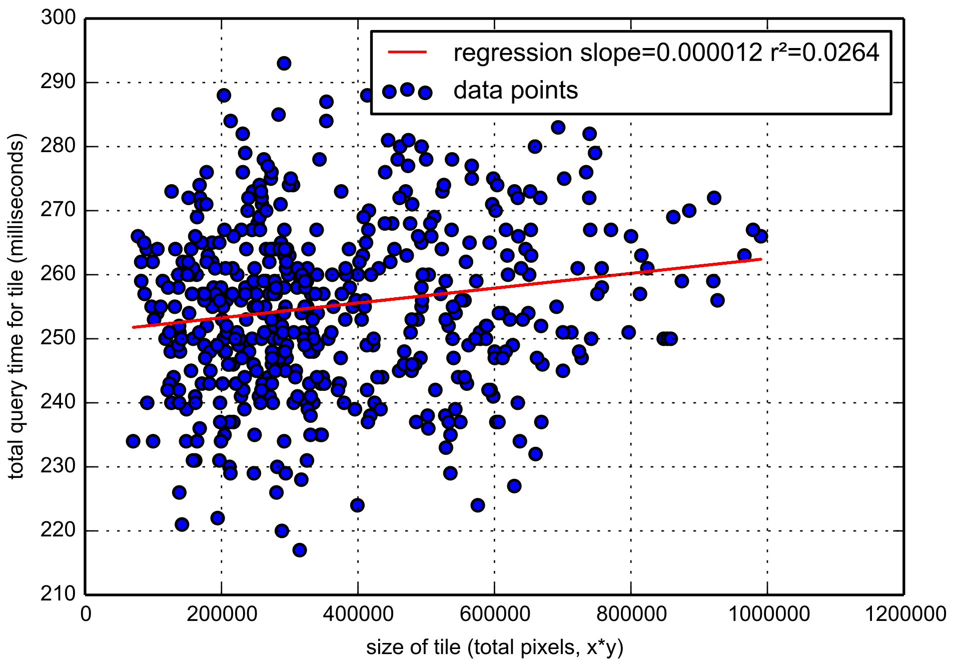

Figure 8: Vector Cluster: query times for varying size bboxes (San Jose highway layer)

considered CPU-bound. And like many other workloads, CPU-intensive portions of the work are insignificant, in terms of time, compared to other bottlenecks like I/O. This result motivated the next test, given below in Section 4.2.2, where that bottleneck will be examined.

4.2.2 Number of objects returned

Once it is clear that bounding box size is not a significant factor in query performance, the same queries (on the highway layer in San Jose, USA) can be measured on another axis: number of objects returned, or put another way: the payload size.

Figures 9 and 10 show that object count is a far more significant factor in query performance than bounding box size. There are some small differences, however, that are worth noting.

Figure 9 shows a less-tightly-fit regression line ($r^2 = 0.85$) than the data from PostGIS in Figure 10. Essentially the GeoMesa data is noisier. This observation seems consistent with the nature of these systems: GeoMesa is the most complex, built on multiple systems with many sources of latency which can be difficult to measure. PostGIS data in Figure 10 is more tightly fit, as the time to retrieve the data has a very linear relationship with payload size. PostGIS is much simpler, with only one network connection, one hard disk, one virtual memory system, etc, coming into play. So those results are more predicable.

Given the structure of the R-trees, and GeoMesa's geohash structure, it is not surprising that this query becomes I/O bound rather than CPU bound.

Figure 11 is notable as it shows the lowest slope and the lowest correlation ($r^2 = 0.278$) with the results from the test on Vector Cluster. So the size of the query payload is much less important

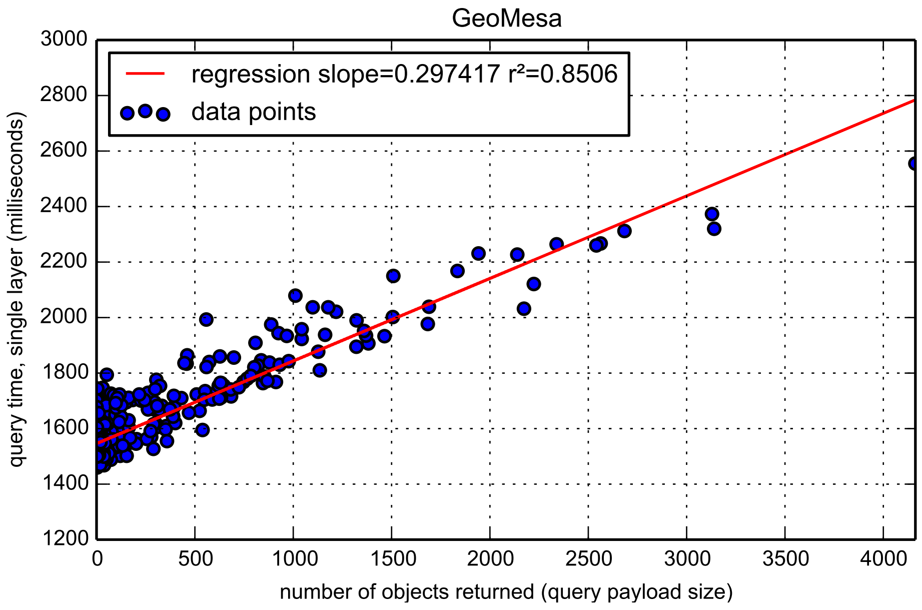

Figure 9: GeoMesa: query times for varying object densities (San Jose highway layer)

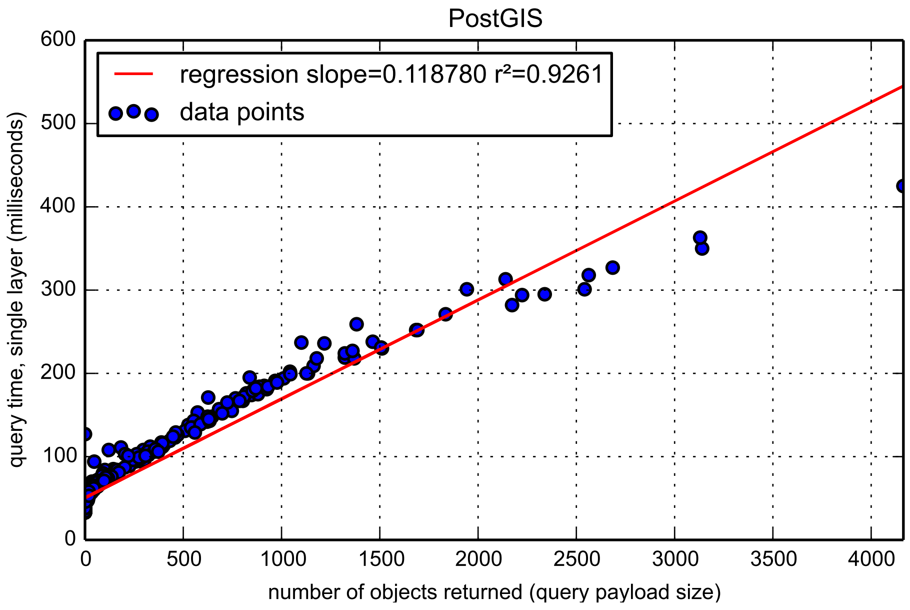

Figure 10: PostGIS: query times for varying object densities (San Jose highway layer)

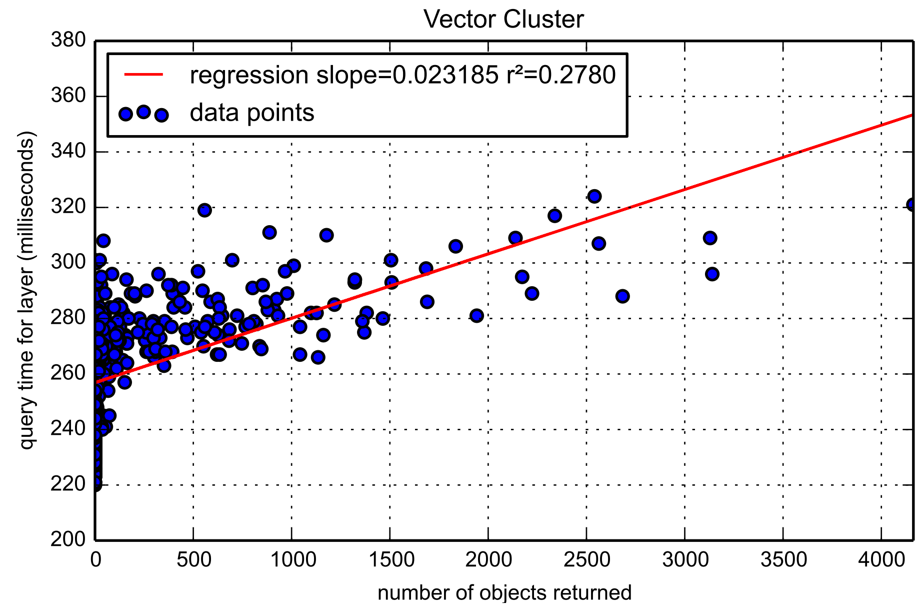

Figure 11: Vector Cluster: query times for varying object densities (San Jose highway layer)

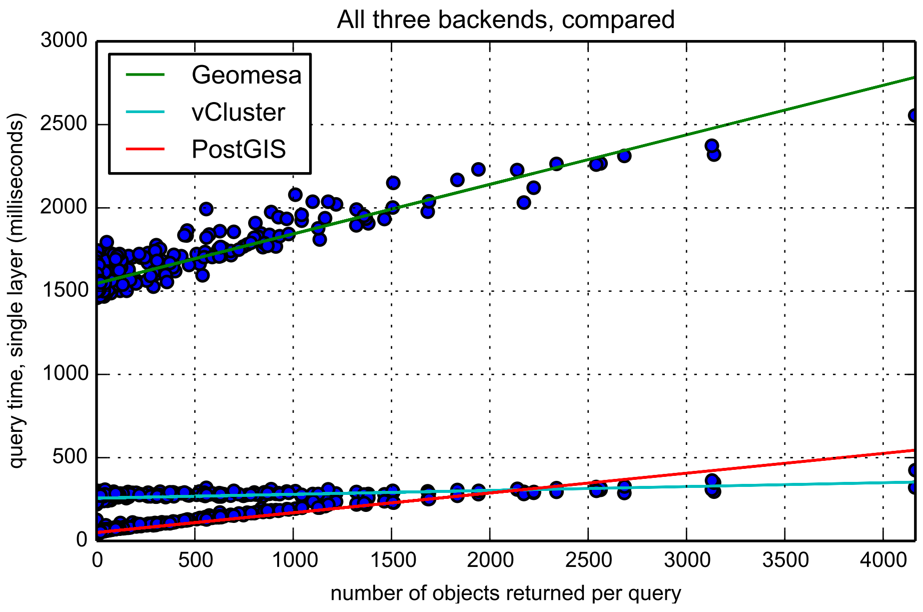

Figure 12: All 3 services, compared

for queries on the Vector Cluster. This observation also agrees well with what is known about the Vector Cluster model: this is the only model in which there is no network connection involved, and the Linux Virtual Filesystem cache likely holds a great deal of the data in memory. So I/O bandwidth effects are much less pronounced here. More of this time is spent actually traversing the R*Tree index, and less of the time is spent actually fetching and transferring data, compared to the other two models. As observed above, bounding box size actually was slightly more significant for Vector Cluster than for the others, again confirming that I/O effects like network and disk are much less impactful on this local, flat-file approach.

Finally, Figure 12 shows the previous three results combined, emphasizing the intersection of the vCluster and PostGIS regression lines. This demonstrations the way that the network I/O effects catch up with PostgreSQL when the dataset rises above a certain size.

4.3 Parallelism

As discussed in Section 3.4 and illustrated in Figure 3, a common workload for a tile server would be to generate a raster tile for display to a user by querying a (possibly large) number of layers of vector data. As observed when testing these storage backends as datastores for Geoserver and NRL's tile server, performing these queries in parallel is essential for any system with query latency (or more simply, any system).

Section 3.7 describes the mechanism used for threading in our test software. The testing approach used was to vary the number of threads used to retrieve the data to build a single tile, in order to find an optimal level of parallelism.

The parallelism tests were conducted over the North America random bounding boxes, shown in Section 3.6.1. For each bounding box, all layers are queried as fast as possible (given the thread concurrency constraint).

In Figures 13, 14 and 15, results are shown for the same workload as the number of threads is increased. The threading levels are colored differently to stand out, and also include a Normal PDF plot.

Like in many other examples of software parallelism, vector cluster performance gains considerably when going from 1 to 2 threads, and returns fall off as the number is increased further.

For the Vector Cluster test in Figure 13, 7 threads is the point at which no further improvement can be made by adding more threads. We do see an increase in variability (indicated by a wider, shorter PDF curve) as more threads are added: more threads essentially adds noise.

PostGIS seems to benefit slightly more from multithreaded processing. The test shown in Figure 14 shows the relational database making steady gains up to 8 threads. After 8 threads, very small gains (less than one standard deviation) are made while the σ value increases: more threads mean more noise and uncertainty. Above 19 threads, the mean time actually gets worse. Between 8 and 19 threads would seem to be ideal; 8 would be preferable, as the benefits of going higher are nearly negligible.

This additional parallelism advantage may be due to PostgreSQL's long history as a high-performance database system under great concurrency and load.

The results for GeoMesa shown in Figure 15 are less clear. The data is erratic, so while there is a definite improvement from 1 thread to many threads, the change is not monotonic and very noisy. This is likely due to factors other than threading level. The Hadoop cluster is far more complex than either the PostGIS server or the Vector Cluster file, so gaining insight into its behavior will require extensive investigation. As clustered systems like Hadoop gain in popularity, there will be more motivation to investigate these performance questions deeply.

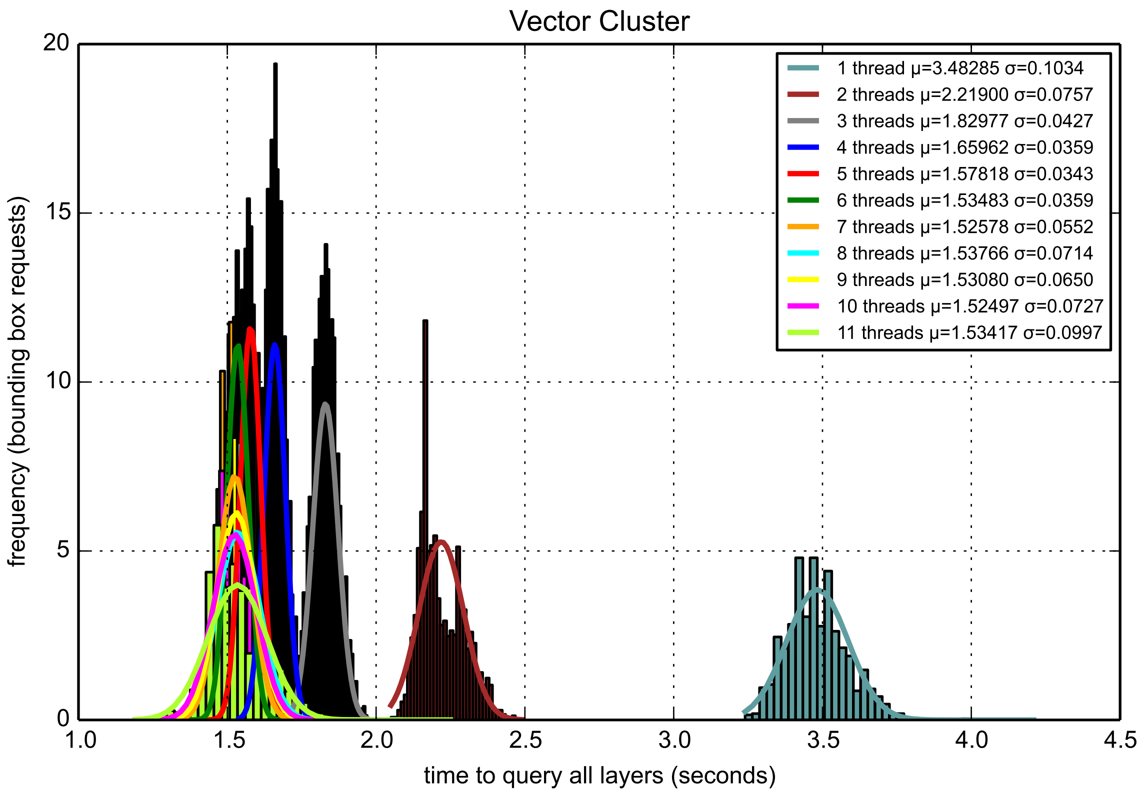

Figure 13: Vector Cluster threading results histogram (all layers)

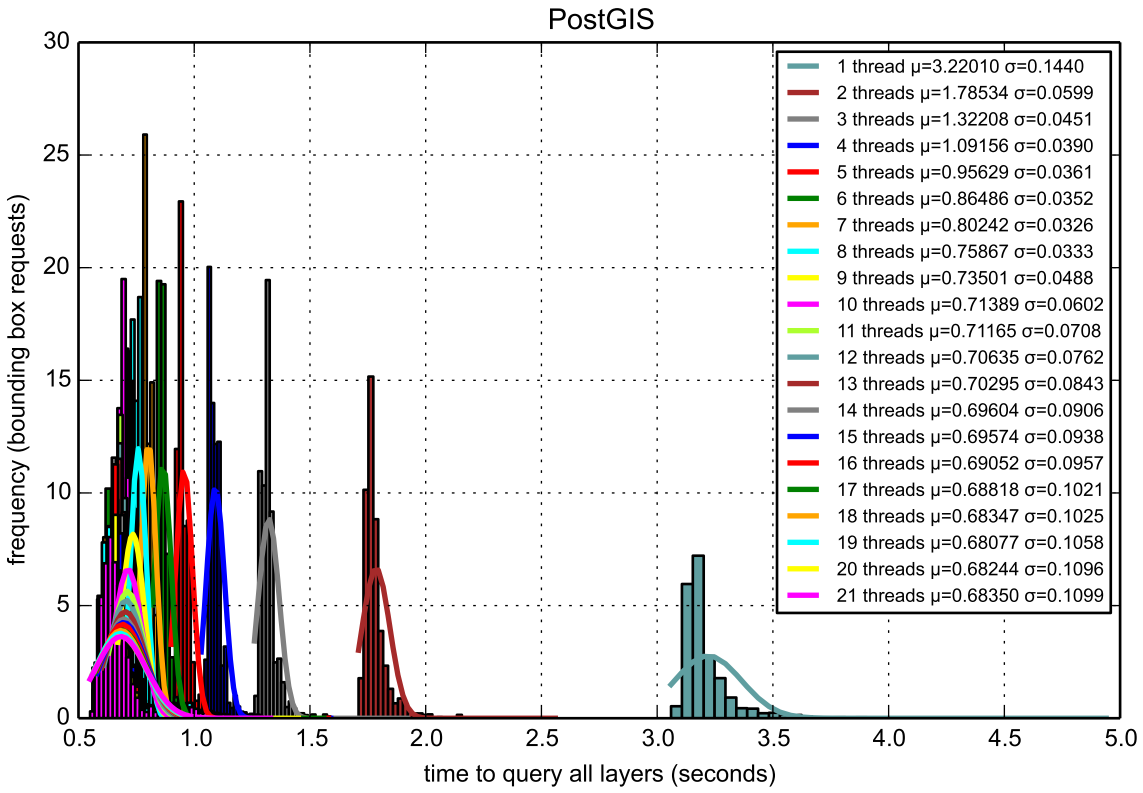

Figure 14: PostGIS threading results histogram (all layers)

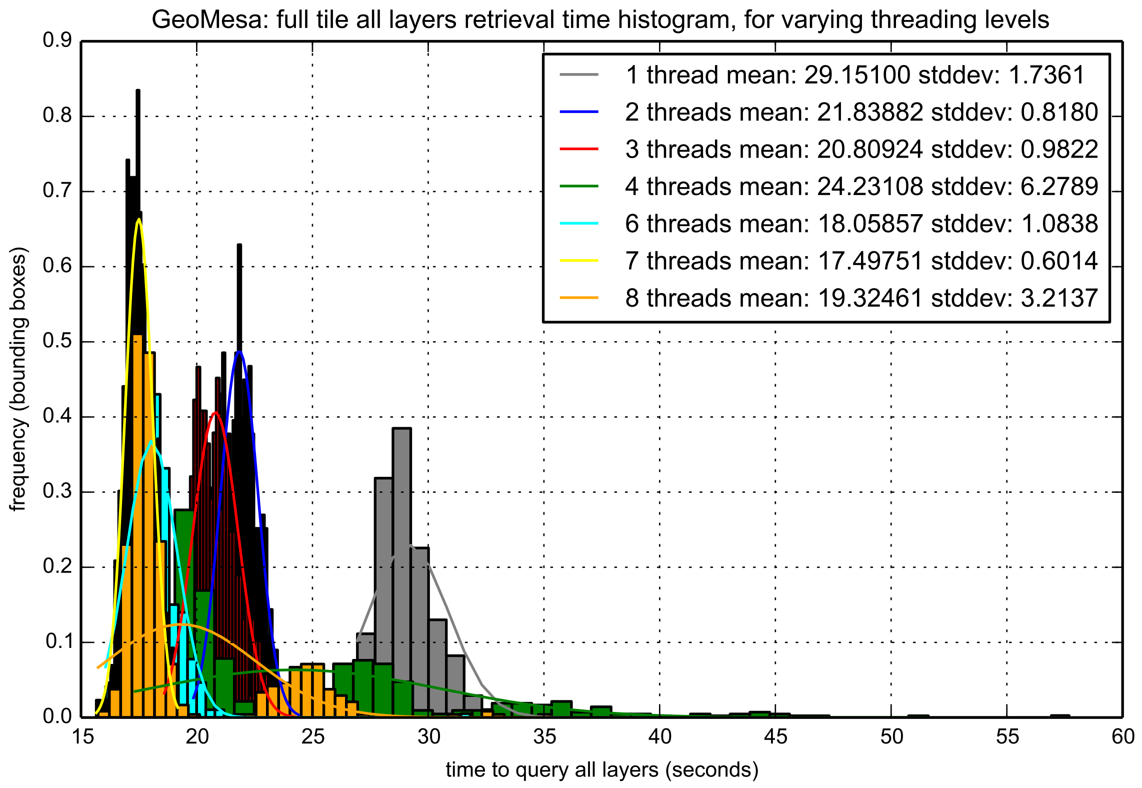

Figure 15: GeoMesa threading results histogram (all layers)

5 Conclusions

The three geospatial data storage systems studied vary greatly in their design, from extremely simple to highly complex.

Based on experiences from running the experiments in Section 4, a qualitative evaluation of the strengths and weaknesses of each paradigm is given in Table 8, along with a general description of each in Table 7.

	underlying paradigm	hardware requirement	index
Vector Cluster	flat file	low: single node	R*tree
PostGIS	relational database	medium: single server	R*tree[a]
GeoMesa	distributed key/value	high: many-node cluster	lexicographical[b]

[a]R*Tree-over-GiST (Generalized Search Tree)
[b]geohash interleaved with time

Table 7: Summary of storage backends and their properties

	latency	scalability	security
Vector Cluster	low	low: limited by filesystem	low: none
PostGIS	low	medium: limited by DB server or cluster	medium: table-level security
GeoMesa	high	high: Hadoop cluster	high: row-level security

Table 8: Summary of storage backends and their advantages/disadvantages

As noted in Section 4.2.2, the results from GeoMesa tests were quite noisy and inconsistent compared to the other two systems. These values should be considered high-uncertainty, and more work would warranted in tuning GeoMesa and Accumulo for performance and studying what other factors come into play with this highly complex system.

Regardless of uncertainty in the GeoMesa query times, they are very certainly significantly worse than the Vector Cluster and PostGIS times. This may not surprise those who have also studied Accumulo and similar systems. As Sawyer, et al., of MIT Lincoln Lab write:

> [N]ew NoSQL databases lack the mature code base and rich feature set of established RDBMS solutions. [...] As a result, optimizing data retrieval in a NoSQL system can be challenging. [...] Because distributed systems are complex, bottlenecks can be difficult to predict and identify. Many open-source NoSQL databases are implemented in Java and use heavy communication middleware, causing inefficiencies that are difficult to characterize.[33]

Sawyer, et al.'s work specifically addresses Accumulo, and these remarks seem to fit very well with the results here as well.

The major drawback seen here in GeoMesa is high latency on query responses. For the case of a tile server, this is a problem — possibly a major problem. In the cases tested here, where a tile is rendered based on 191 queries, that latency is magnified into something unacceptable. This might not be so bad in a system which abandons the traditional GIS notion of feature layers, like many OSM-based tile renderers.

For other use cases, the latency may be more tolerable or even worth the cost. Accumulo's iterators provide it the ability to perform significant computation on data that these tests did not utilize when using it merely as a datastore to be read from. Use cases that depend on high-throughput operations rather than low-latency small queries will be better positioned to utilize the strengths of a system like GeoMesa.

The order-of-magnitude slower performance seen in the GeoMesa results may be somewhat less surprising considering that Accumulo does not seem to prioritize low-latency random-access queries in its design. Instead, many publications about Accumulo's strengths focus on Ingest performance, where the desired metric is throughput, not latency. This static dataset did not take advantage of this strength at all. GeoMesa/Accumulo would be more appropriate for a workload that involved constant or frequent ingestion (ie, writes to the data store).

This emphasis on high throughput rather than low latency is not limited to Accumulo. Other distributed-cloud key-value ("NoSQL") systems such as Apache Cassandra make a similar tradeoff; Rabl, et al. write:

> "Cassandra achieves the highest throughput for the maximum number of nodes [...] This comes at the price of a high write and read latencies. Cassandra's performance is best for high insertion rates."[31]

This tradeoff is seen frequently in many other systems, and should not be ignored when considering this type of software for latency-sensitive applications.

One surprising result was the very fast performance of PostGIS/PostgreSQL, which easily outperformed Vector Cluster on all queries except the few with the largest payloads (shown in Figure 12). This contradicts the results from a very similar test by Sample and Ioup in 2010 which shows "File Query" outperforming PostGIS (labelled "DB Query"). What the two results have in common are the slopes: in both, PostGIS query time grows at a faster rate, so a regression line would have a larger slope. [32]

One possible explanation is simply the evolution of large, vibrant open-source projects like PostgreSQL and PostGIS. In the years since the Sample book was published, PostgreSQL has released a new major version and many minor versions. According to multiple sources, query performance has become dramatically faster in recent versions. [30] [38] The use of these recent versions, PostgreSQL 9.3 and PostGIS 2.1, likely explains at least some of the difference between this result and the Sample/Ioup result of 2010.

5.1 Implications for cloud-based geospatial data infrastructure

That systems like Hadoop suffer from inefficiency has been well described by others; see Anderson and Tucek's succinctly titled SOSP09 paper "Efficiency Matters!". [2] Yet in spite of this, in all likelihood use of distributed, clustered, and off-site (better known as "Cloud") storage and processing of geospatial data will continue to grow and develop. Technologies like GeoMesa (and the underlying Hadoop cluster) enable many new abilities not previously practical, particularly in the realm of security, scalability, and high availability.

Utilizing a cloud infrastructure in pursuit of these features will come at a cost, and for some classes of problems, a high cost. The workload used here, a non-temporal geospatial query driven by traditional GIS services like tile serving, appears to be one for which the cost is very high, possibly too high.

So in order to adapt to a possible future of cloud-based geospatial data, changes will need to be made to traditional GIS model when it comes to tile servers and vector servers. Do not expect to simple drop in a cloud-based distributed system like GeoMesa as a replacement for a traditional GIS server like PostgreSQL, and certainly not for a ruthlessly simple and efficient file like a Vector Cluster. A fault-tolerant massively distributed system like Hadoop adds power, but also great complexity. These systems will be highly valuable, but our front-end services need to adapt to a different model. Going to the cloud requires a paradigm shift, and is not something to be done piecemeal or half-way. Go big or go home!

Regardless of the performance cost, cloud-based geospatial (and especially spatiotemporal) data systems will certainly play a role in future GIS applications. The security features provided by Accumulo make it uniquely suited to applications which require tight, granular security. As temporal data becomes more common (and orders of magnitude larger), systems like GeoMesa will begin to show their advantages.

Traditional GIS services like tile serving may need to adapt to a changing geospatial datastore landscape. As shown in these experiments, multi-layer queries for tile generation will face serious latency problems. Due to this latency, some of which is unavoidable in cloud-based infrastructure, multiple queries should be avoided. A single bounding box query could easily use attribute filtering to accomplish the same goals, and still provide a "layer" abstraction to the stubborn traditional GIS user, without paying the large latency penalty seen in Figure 15 earlier.

GeoMesa should only be used in narrow cases, not a general purpose replacement for PostGIS. Likewise, Vector Cluster by definition can only support narrow use cases, due to its simplistic read-only structure and limited feature set.

In this work, no temporal data is stored nor queried, so GeoMesa's hybrid spatio-temporal key structure is only a hindrance for the tileserver workload. GeoMesa should be used for data and workloads which are both temporal and spatial.

5.2 When is a problem a "big data" problem?

When considering "big data" oriented solutions like Accumulo (and other Hadoop-based systems), it is important to ask "is this a big data problem?" This question does not only address the question of how many bytes of data are in play, but should consider the "3 Vs of Big Data": Volume, Velocity and Variety.[11]

OSM's *volume* is not terribly large by this standard: less than 100GB of compressed data. This is all vector data, so this does amount to a large number of database rows, and could be considered computationally quite large if viewed as a directed graph. But it is not difficult to store this on a single, or few, modern systems.

OSM data's schemaless key-value tagging system might seem like an example of the sort of *variety* in these problems. But in fact, the data is not terribly noisy in that way. Most OSM map editors follow conventions, if not schemas, making the data fairly well-defined despite the lack of formal schema.

Most map data consisting of roads, buildings, and other large physical features will almost certainly not have *velocity*, at least not at scales anywhere near those of today's maps. The physical world does change constantly, but large features like roads and buildings will likely not change in the millions-of-rows-per-second rates that big data systems expect. OSM is likely the highest-velocity vector map data source around, seeing on the order of 1 million[25] new objects per day, but this still falls short of "big data" velocity.

When very high resolution, real-time spatiotemporal data is being aggregated into GIS systems, then the velocity will call for a big data solution. The "big data" solutions will enable new and powerful abilities in geospatial computing, but they may also be poor solutions for more traditional GIS computing.

Meanwhile, there are solutions which attempt to combine the strengths of traditional RDBMS systems and new distributed systems. One example is Greenplum, a shared-nothing massively-parallel relational database built on PostgreSQL. Greenplum has recently been proposed as a solution for scaling GIS systems while keeping the relational data paradigm in tact.[26]

5.3 Future work

GeoPackage, the recently-introduced "flat file" geospatial data format based on SQLite (described in Section 2.3.3) has RDBMS-like properties not found in Vector Cluster. How does GeoPackage perform, in terms of query latency and throughput?

What changes do large-memory systems introduce, when much or all of a dataset can be held in-memory? The Sample/Ioup test in [32] was performed in the late 2000s, with a dataset of the same order of magnitude as data used here, but on a system with considerably less RAM. So in that case, the database- and file- based storage systems perform well when they succeed in minimizing disk access. In newer systems like those used here, the RAM size has grown faster than the vector data, putting a larger fraction of the data in a database buffer or VFS page cache. How does this change the performance profiles of each system, now that random disk access may not always dominate the result?

How can machine learning techniques be applied to changing geospatial data like OpenStreetMap? Can the computational power of Accumulo contribute to the analysis of spatial data (and more importantly, spatio-temporal data)? As the number of map edits outstrips the number of people to manually check them for errors, an open problem with OSM developers[27] today is the need for tools to detect suspicious edits that are likely to contain errors (or deliberate misinformation, known on Wikipedia as "vandalism").

More generally, further experimentation on these storage paradigms with a wider variety of workloads would open up new chances to weigh their strengths and weaknesses. Studying data with a larger temporal component would open up the questions of streaming updates (high-throughput ingestion), real-time server-side computation on spatial data, and generation of dynamic spatiotemporal content where spatial data changes over time.

[25] `http://wiki.openstreetmap.org/wiki/Stats#New_nodes_and_ways_added_per_day`

[26] `http://boundlessgeo.com/2015/10/scaling-your-gis-with-pivotal-greenplum/`

[27] See `http://neis-one.org/2016/01/suspicious-osm/` for discussion of a tool which applies some heuristics for detecting map editors with little experience. Pascal Neis' solution addresses this problem with a useful tool, but neither this nor other proposed solutions utilize more sophisticated pattern recognition techniques, yet.

6 References

[1] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop-GIS: A high performance spatial data warehousing system over mapreduce. August 2013.

[2] Eric Anderson and Joseph Tucek. Efficiency matters! *Operating Systems Review*, 44(1):40–45, 2010.

[3] P Beaulieu and H Dohmann. The digital geographic information exchange standard and military mapping. *Stockholm: Proceedings of the 18th ICC*, pages 563–570, 1997.

[4] Kelly Chan. *DIGEST A primer for the International GIS Standard*, volume 7. CRC Press, 1998.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. In *Proceedings of the Sixth Symposium on Operating Systems Design and Implementation*, November 2006.

[6] PostGIS Project Steering Committee et al. PostGIS 2.2.3dev Manual. `http://postgis.net/docs/`, 2016.

[7] Open Geospatial Consortium. Simple Feature Access, Part 1: Common Architecture. `http://www.opengeospatial.org/standards/sfa`, 2007.

[8] Open Geospatial Consortium. OGC History (Detailed). `http://www.opengeospatial.org/ogc/historylong`, 2010.

[9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. Sixth Symp. on Operating System Design and Implementation (6th OSDI'04)*, pages 137–150, San Francisco, CA, December 2004. USENIX Association. Google.

[10] PostgreSQL Developers. Replication, clustering and connection pooling. `https://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling`.

[11] Edd Dumbill. What is big data? `https://www.oreilly.com/ideas/what-is-big-data`, 2012.

[12] Ahmed Eldawy and Mohamed F. Mokbel. The Ecosystem of SpatialHadoop. *SIGSPATIAL Special*, 6(3):3–10, April 2014.

[13] FOSS4G Organising Committee. Curious How Various Web Mapping Servers Perform? Press Release, October 2009. `https://wiki.osgeo.org/wiki/FOSS4G_2009_Press_Release_32`.

[14] Apache Software Foundation. Apache Accumulo User Manual: Security. https://accumulo.apache.org/1.5/accumulo_user_manual.html#_security, 2014.

[15] Anthony Fox, Chris Eichelberger, John Hughes, and Skylar Lyon. Spatio-temporal indexing in non-relational distributed databases. In *Big Data, 2013 IEEE International Conference on*, pages 291–299. IEEE, 2013.

[16] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *Proceedings of the nineteenth Symposium on Operating Systems Principles (SOSP'03)*, pages 29–43, Bolton Landing, NY, USA, October 2003. ACM, ACM Press.

[17] Dan Gillick, Arlo Faria, and John Denero. Mapreduce: Distributed computing for machine learning, 2006.

[18] Michael F Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007.

[19] Mordechai Haklay and Patrick Weber. OpenStreetMap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[20] James Hughes. HBase limitations. https://locationtech.org/mhonarc/lists/geomesa-users/msg00667.html, 2015.

[21] James N. Hughes, Andrew Annex, Christopher N. Eichelberger, Anthony Fox, Andrew Hulbert, and Michael Ronquest. Geomesa: a distributed architecture for spatio-temporal fusion. In *Proceedings of SPIE*, volume 9473, pages 94730F–94730F–13, 2015.

[22] Elias Ioup, Norman Schoenhardt, and Robert Owens. Private interview (recording available), March 15 2016.

[23] ISO 19125-1:2004 Geographic information – Simple feature access – Part 1: Common architecture. Standard, International Organization for Standardization, Geneva, CH, August 2004.

[24] Tom Lane. Core team statement on replication in PostgreSQL. http://www.postgresql.org/message-id/26529.1212070375@sss.pgh.pa.us, 2008.

[25] Nazario Irizarry, Jr. Mixing C and Java for High Performance Computing. *MITRE Corporation, Bedford, MA, Technical Report*, 2013.

[26] Regina O. Obe and Leo S. Hsu. *PostGIS in Action*. Manning Publications Co., Greenwich, CT, USA, 2nd edition, 2015.

[27] Open Geospatial Consortium. OGC GeoPackage Encoding Standard. http://www.geopackage.org/spec, 2016.

[28] OSGeo: the Open Source Geospatial Foundation. FOSS4G. https://wiki.osgeo.org/wiki/FOSS4G, 2015.

[29] Paolo Palmieri, Luca Calderoni, and Dario Maio. Spatial bloom filters: enabling privacy in location-aware applications. In *Information Security and Cryptology*, pages 16–36. Springer, 2014.

[30] Michael P Peterson. *Mapping in the Cloud*. Guilford Publications, 2014.

[31] Tilmann Rabl, Mohammad Sadoghi, Hans-Arno Jacobsen, Sergio Gómez-Villamor, Victor Muntés-Mulero, and Serge Mankowskii. Solving Big Data Challenges for Enterprise Application Performance Management. In *Proceedings of the VLDB Endowment (PVLDB), Vol. 5, No. 12, pp. 1724-1735 (2012)*, August 20 2012. VLDB2012.

[32] John T Sample and Elias Ioup. *Tile-based geospatial information systems: principles and practices*. Springer Science & Business Media, 2010.

[33] Scott M Sawyer, B David O'Gwynn, An Tran, and Tamara Yu. Understanding query performance in Accumulo. In *High Performance Extreme Computing Conference (HPEC), 2013 IEEE*, pages 1–6. IEEE, 2013.

[34] SQLite Development Team. Most Widely Deployed and Used Database Engine. `https://www.sqlite.org/mostdeployed.html`.

[35] Michael Stonebraker and Lawrence Rowe. "The Design of Postgres". In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 340–355, 1986.

[36] The Open Source Geospatial Foundation. OSGeo. `https://www.osgeo.org/`, 2015.

[37] Rik van Riel and Peter W Morreale. Documentation for `/proc/sys/vm`. `https://www.kernel.org/doc/Documentation/sysctl/vm.txt`, 2008.

[38] Tomas Vondra. Performance since PostgreSQL 7.4 / pgbench. `http://blog.pgaddict.com/posts/performance-since-postgresql-7-4-to-9-4-pgbench`, 2015.

[39] Randall T. Whitman, Michael B. Park, Sarah M. Ambrose, and Erik G. Hoel. Spatial indexing and analytics on hadoop. In Yan Huang, Markus Schneider 0001, Michael Gertz, John Krumm, and Jagan Sankaranarayanan, editors, *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Dallas/Fort Worth, TX, USA, November 4-7, 2014*, pages 73–82. ACM, 2014.

7 Vita

The author was born in New Orleans, Louisiana. He earned a B.S. in Computer Science from Carngie Mellon University in 2008 and joined the University of New Orleans Computer Science graduate program in 2013.