A DISTRIBUTED WORKLOAD-AWARE APPROACH TO PARTITIONING GEOSPATIAL BIG DATA FOR CYBERGIS ANALYTICS

BY

KIUMARS SOLTANI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Informatics
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Doctoral Committee:

Professor Shaowen Wang, Chair
Professor Jiawei Han
Professor Jana Diesner
Professor Aditya Parameswaran

ABSTRACT

Numerous applications and scientific domains have contributed to tremendous growth of geospatial data during the past several decades. To resolve the volume and velocity of such big data, distributed system approaches have been extensively studied to partition data for scalable analytics and associated applications. However, previous work on partitioning large geospatial data focuses on bulk-ingestion and static partitioning, hence is unable to handle dynamic variability in both data and computation that are particularly common for streaming data.

To eliminate this limitation, this thesis holistically addresses computational intensity and dynamic data workload to achieve optimal data partitioning for scalable geospatial applications. Specifically, novel data partitioning algorithms have been developed to support scalable geospatial and temporal data management with new data models designed to represent dynamic data workload. Optimal partitions are realized by formulating a fine-grain spatial optimization problem that is solved using an evolutionary algorithm with spatially explicit operations. As an overarching approach to integrating the algorithms, data models and spatial optimization problem solving, GeoBalance is established as a workload-aware framework for supporting scalable cyberGIS (i.e. geographic information science and systems based on advanced cyberinfrastructure) analytics.

To my family, who taught me that the darkest hour is just before the dawn and gave me the strength to fight for life.

ACKNOWLEDGMENTS

I have been very fortunate to complete my Ph.D. at the great University of Illinois at Urbana-Champaign and work under the supervision of Dr. Shaowen Wang. Dr. Wang has always encouraged me to work hard and pursue my goals. I am forever grateful for his support and his wisdom, which has inspired me to improve both my academic and personal life. I would also like to thank my committee members Drs. Jiawei Han, Jana Diesner and Aditya Parameswaran for their valuable insights and guidance throughout my Ph.D.

Moreover, I would like to thank my colleagues at the CyberGIS Center and Cyberinfrastructure and Geospatial Information (CIGI) Laboratory for helping me to shape my research ideas. Particularly, my deepest appreciation to Drs. Yan Liu and Anand Padmanabhan who played an important role to bring my research visions to life.

I would like to dedicate this work to my family, who unconditionally loved me and believed in me even when I did not believe in myself. I am extremely grateful to my dad, Kambiz, who raised me to believe honesty is the greatest quality in a human being. He taught me that life is a daring adventure and gave me the courage to explore my own journey. I owe this achievement to my loving and caring mom, Shahindokht, an angel who has been with me every step of the way. Her kindness gave me the confidence to become who I am today. I am very thankful to Navid, the best brother-in-law that one can hope for. I know that he is always on my side and I cherish his advice and care every day.

Above all, I am exceptionally grateful to my lovely sister Atousa, who is the most influential person in my life. She has always stood by my side through thick and thin and taught me to be a better version of myself. I could not imagine my life without her kindness and support.

Atousa has always cheered me up when I was down and taught me that giving up is never an option. I love her more than anything and anybody in the world and will work hard to live a life that makes her proud.

GRANTS

This research is supported in part by the National Science Foundation (NSF) under grant numbers: 0846655, 1047916, 1354329, 1429699 and 1443080.

Computational experiments used the Extreme Science and Engineering Discovery Environment(XSEDE) (resource allocation Award Number SES090019), which is supported by the National Science Foundation with grant number 1053575; ROGER supercomputer, which is supported by NSF under grant number 1429699 and Chameleon testbed supported by the National Science Foundation.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Geospatial sciences and technologies have witnessed a tremendous growth of geospatial data produced over the past several decades. Sensors, GPS-enabled devices and satellites collect data in accelerated rates with a variety of data models and formats. In this context, cyberGIS (that is, geographic information science and systems based on advanced cyberinfrastructure) and data-driven geography have emerged as new paradigms of data-intensive geospatial research and education [Miller and Goodchild, 2015,Wang and Goodchild, 2019,Wang, 2016, Wang, 2010]. However, extensive research is required to innovate geospatial algorithms and cyberGIS analytics that can resolve the data intensity, particularly from the perspective of dynamic and streaming data. Therefore, this thesis focuses on novel algorithms and architectures to enable data-intensive cyberGIS analytics, where data velocity is the primary challenge.

Previous work demonstrated that centralized architecture where data is stored on a single powerful server is not suitable to resolve this challenge while distributed architecture for geospatial data management is viable [Wang et al., 2015]. In such a distributed environment, efficiently managing data is crucial and to achieve this goal a two-level indexing scheme [Eldawy and Mokbel, 2015] is often employed, where (a) a global index determines how data is partitioned among different nodes; and (b) local indexes manage storage of actual data in each node (Figure 1.1).

Though data partitioning is critical to efficiently manage geospatial big data in a distributed environment, existing algorithms are not susitable for partitioning large and dynamic geospatial data. Particularly, previous work focuses on data quantity and static

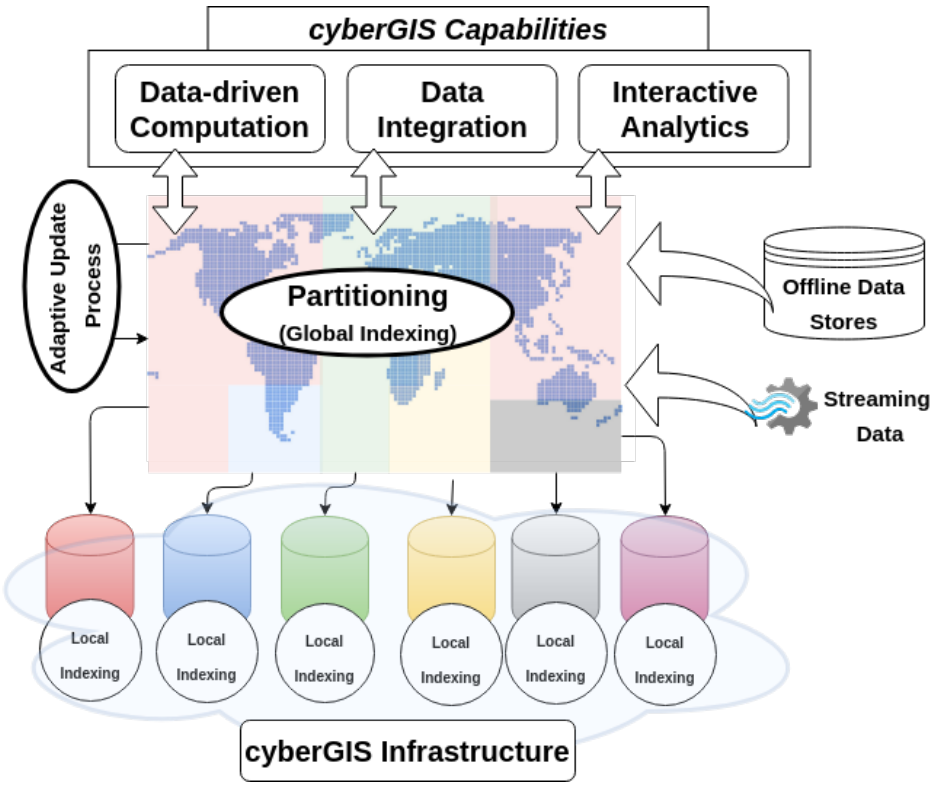


Figure 1.1: Adapting two-level indexing scheme for cyberGIS applications.

partitioning methods designed for workloads (requests made by users to the system [Jain, 1990]) that rarely change over the time [Fox et al., 2013, Malensek et al., 2016, Eldawy, 2014]. This approach is not applicable to large real-time geospatial data that exhibits high variety and comes in at high velocity, since static partitioning may lead to skewed partitions which in turn result in high latency and poor resource utilization [Serafini et al., 2016]. Hence, an effective partitioning approach for such data should address evolvability of workload fluctuations over time by considering spatial and temporal changes in the workload in a holistic fashion.

We have established a workload-aware approach for efficiently indexing, partitioning, and replicating geospatial big data based on cloud computing architecture for achieving scale-out, elasticity and high availability features of cyberGIS analytics [Cooper et al., 2010]. Specific contributions of this research can be summarized as:

- A set of workload-aware load-balancing algorithms for cyberGIS applications that re-

quire data-intensive computation. These algorithms are implemented using the well-known MapReduce model to take advantage of cloud computing infrastructure.

- A fast spatial indexing approach to high-velocity multi-tenant applications for indexing and storing spatial objects with arbitrary shapes.

- A novel adaptive workload-aware framework for handling high velocity real-time geospatial data. This framework applies an evolutionary algorithm to tune how data is spatially distributed over time without re-partitioning data from scratch.

These contributions focus on three critical aspects of data-intensive software systems: scalability, maintainability and reliability [Kleppmann, 2017]. In the first phase of this research, we have developed algorithms for efficiently distributing data and computation among cloud resources that can adaptively respond to inevitable changes in workloads. In the second phase, we design and implement cloud-based architecture to enable seamless integration of the algorithms that were studied in the first phase. The overarching goal of the proposed framework is to a) resolve the growing complexity of data-intensive cyberGIS applications [Gupta et al., 2016]; and b) achieve horizontal scalability to increase resource utilization [Hasselbring, 2016].

1.1 Research Problems

A typical cyberGIS-based knowledge discovery workflow includes at least one of the following components deployed on advanced cyberinfrastructure:

1. **Data management:** includes efficient data storage, indexing, publishing, browsing, search and transfer of data.

2. **Data integration:** Enables interoperable and scalable access to distributed data resources and services.

3. **Data-driven computing:** Using parallel and distributed algorithms to process geospatial data.

4. **Interactive data analyses:** Fast interaction capabilities for end-users.

These components have been extensively investigated in the context of various data-intensive geospatial applications [Padmanabhan et al., 2014, Soltani et al., 2015a, Soltani et al., 2016, Soltani et al., 2015b]. However, prior to implementing such capabilities, we should address two main strategies regarding distributed data processing: domain decomposition and task scheduling [Wang and Armstrong, 2009]. The two strategies are particularly critical in geospatial applications, since real-world geospatial applications are often computationally and data-intensive [Huang et al., 2002].

To design efficient domain decomposition and task scheduling strategies, we should consider three main characteristics [Wang and Armstrong, 2009]. First, solely focus on operation or data cannot handle the reality of geospatial applications. Therefore, the computational intensity should be considered as the main criteria in designing such strategies. Second, domain decomposition and task scheduling strategies should be adaptable to inevitable changes in the characteristics of spatial data and operations. Finally, the strategies should not be tied to a specific parallel computer architecture.

In this thesis, novel workload-aware algorithms are developed to distribute computational tasks and data services of cyberGIS applications among a multi-node cluster. The new knowledge of workload allows us to a) consider integrated intensity of geospatial data and operations; and b) adapt to changes in characteristics of data and operations.

1.1.1 Workload-aware Load Balancing for Data-intensive Geospatial Applications

The 3V (variety, velocity, and volume) of geospatial big data impose significant computational and data challenges. In addition, geospatial data demonstrates unique characteristics

that need to be considered within an scalable cyberGIS software environment. As aforementioned, devising efficient strategies for domain decomposition and task scheduling is critical for designing scalable geospatial applications. In this part of the thesis, we study these strategies for geospatial applications that perform batch processing of massive geospatial data. The main characteristic of such applications is the volume of data that are processed, as the latency of computation and communication is not the main concern. We implement our approach using the MapReduce model [Dean and Ghemawat, 2008] due to its wide adoption [Eldawy and Mokbel, 2015].

Previous research proposed multiple strategies for partitioning geospatial data in distributed environments [Aji et al., 2013a, Cary et al., 2009a, Eldawy, 2014, Eldawy et al., 2015b]. However, we can not solely rely on data partitioning to achieve scalability in such applications.

First, in offline processing applications it is often unrealistic to assume any specific partitioning on the incoming data, since the data is usually stored in data lakes that include unstructured or semi-structured data from multiple data sources. In addition, such massive data is ingested by multiple applications with significantly varying characteristics. Therefore, one application cannot force an specific partitioning on the data.

We address the aforementioned scenario by proposing an approach that does not assume any particular pre-existing partitioning. We use either a theoretical analysis of the problem or observed computational intensity to devise a computational plan that is customized for each problem. The computational plan is a coarse-level mapping of individual records to different nodes in the cluster, which is used to distribute the tasks.

We evaluate our approach using two well-known geospatial operations: multi-resolution data aggregation and approximate point-in-polygon. The approximate point-in-polygon algorithm demonstrates the effects of application parameters in adapting the load balancing strategy. Both operations are evaluated in terms of load balancing capabilities, as well as scalability when the problem size or number of nodes increases.

Adaptive Workload-aware Partitioning of Geospatial Data

Data-driven scientific research has suggested that a fourth paradigm should be added to the three previously identified science paradigms: empirical science, theoretical science and computational science [Miller and Goodchild, 2015]. This new paradigm has initiated changes in fundamental understanding and practices of scientific discovery. Particularly, the ever-increasing availability of various real-time geospatial data has led to the emergence of "data-driven geography", which entails a unique set of challenges, methods and practices. However, current state-of-the-art tools and products are not well suited for handling geospatial big data which is, in turn, limiting new research questions from being asked and answered. Specifically, existing distributed geospatial data management systems are designed for static and predictable data sources [Aly et al., 2015]. However, previous research has indicated that dynamics of data and query load is inevitable in geospatial applications and should be addressed properly [Kleppmann, 2017].

Therefore, this thesis focuses on the following question: how should realtime geospatial big data be managed in order to efficiently support a large magnitude of concurrent requests under dynamically changing workload (data and query distributions). To efficiently store and query in distributed architecture, data and associated services should be distributed in multiple nodes that communicate to each other over the network. However, in current literature there is a lack of efficient algorithms for partitioning large dynamic geospatial data. Previous work mainly addresses the data volume challenge and exploit static partitioning methods that are designed for predictable workloads that rarely change over time [Fox et al., 2013, Malensek et al., 2016, Eldawy, 2014]. These methods often do not address the velocity and variety aspects of big geospatial data. Particularly, when geospatial data exhibits high variety and come in at high velocity, static partitioning may lead to skewed partitions, resulting in high latency and poor resource utilization [Serafini et al., 2016]. Hence, an effective partitioning approach for dynamic data should consider spatial workload fluctuations over

time.

To address such requirements, we propose a workload-aware partitioning approach for real-time geospatial data. Our approach addresses two specific research questions regarding adaptive partitioning of geospatial data. The first question is how to model the performance of an existing partitioning scheme in terms of load imbalance as data changes over time. The model also serves as a guide for the evolution process of the partitions. Second, after an existing partition becomes imbalanced, how to repartition data in an incremental and non-disruptive way such that the data system and associated applications do not experience interruptions or compromise the quality of service (maintains *availability*). This process should preserve spatial contiguity of partitions to minimize distributing single data insertion/retrieval requests into multiple nodes (*distributed transactions*), which is proven to be as critical as balancing load [Serafini et al., 2016].

Our approach computes statistics for data and query distribution over time and space and uses such statistics to evaluate the current partitioning scheme according to a fitness function that measures load variation among partitions [Arzuaga and Kaeli, 2010]. We quantify such partitions loads by representing the spatial extent of data and queries using a geohashing algorithm [Niemeyer, 2008]. Geohash provides a hierarchical grid-based model, based on a Z-order space filling curve, by interleaving bits from coordinate space of latitude and longitude into a one dimensional string representation. Geohash has been extensively used to encode and index large geospatial data since it can be efficiently computed, stored and retrieved in key-value stores [Malensek et al., 2013a]. However, previous work has mainly focused on simple spatial structures such as point data [Moussalli et al., 2015] or bounding boxes [Malensek et al., 2013a]. Therefore, we have developed two algorithms to provide fast geohashing of arbitrary-shaped line and polygon data.

If a shift in the data/query distribution results in a partitioning scheme that is deemed inefficient, we use a spatial evolutionary algorithm to incrementally modify the existing partitions until the balance is regained. Our algorithm uses a spatial mutation operator

that preserves contiguity at each step to avoid expensive contiguity checks and contiguity repair operations [Liu et al., 2016]. This incremental approach to tuning existing partitions avoids creating partitions from scratch, which is clearly inefficient [Aly et al., 2015] and works by moving a limited number of spatial units among partitions to re-balance them. Hence, our approach is able to autonomously scale-in when some partitions are underloaded and scale-out when some partitions are overloaded. The overall process of our adaptive partitioning algorithm is explained in Figure 1.2.

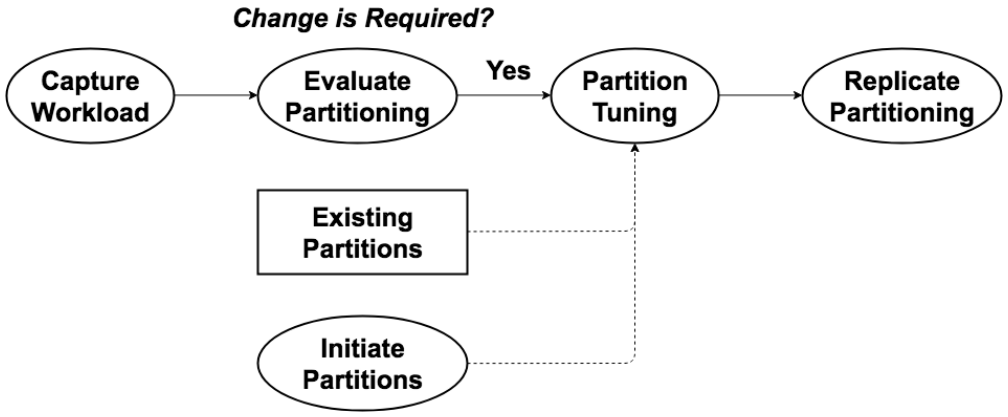


Figure 1.2: Adaptive workload-aware partitioning process.

To avoid disrupting the data ingestion/retrieval process while migration is happening, we provide a rolling migration process among partitions by simultaneously holding two versions of the partitioning (existing and future partitioning schemes), each differentiated by the timestamps of the data they are holding. By using this technique, we do not interrupt the operations of the framework while partition modification is in progress.

We evaluate the effectiveness of our partitioning approach by using an integrated dataset that contains one year of Twitter geo-tagged tweets. In addition, we inject randomly generated hotspots into the dataset to simulate inevitable changes in the underlying workload. Our experiments demonstrate the advantage of our adaptive partitioning approach over previously used static partitioning methods such as k-d tree, in terms of load balance metrics and cost of changing partitions. Finally, we evaluate scalability by measuring latency and throughput of our approach using a varying number of partitions and nodes.

1.1.2 Thesis Layout

In the first part of the thesis, we explain the motivating case studies for the algorithmic and architectural innovations that are elaborated in Chapter 2. The case studies include multiple data-driven geospatial applications encompassing the four main components of cyberGIS data-driven workflows, i.e. data management, data integration, data-driven computing and interactive analytics.

Chapter 3 explains our approach to workload-aware load-balancing of data-intensive geospatial applications. We evaluate our load-balancing approach using two case studies: movement aggregation and point-in-polygon. We report the load statistics for both operations to demonstrate the load balancing effect of our approach. In addition, for both operations we evaluate the "interactive scalability" of the associated workflow by performing stress test with multiple types of queries.

In the next two chapters, we shift our focus into adaptive partitioning of data for applications that handle streaming data. In Chapter 4 describes a geohash-based model for indexing geospatial data. This model is particularly beneficial for solving our problem since it can be efficiently stored and retrieved using key-value data stores and can be updated concurrently for high-velocity. Since geohash is primarily used to index points, we have developed two efficient algorithms to index polygons and lines using geohash.

Chapter 5 describes a novel algorithm for adaptive partitioning of geospatial data. We model the problem of finding the optimal partitions as a spatial optimization problem with the multi-criteria objective function including load imbalance deviation and spatial compactness (Section 5.3). This algorithm can gradually modify the partitions by performing a spatially tuned mutation operation (Section 5.6). Section 5.7 highlights the advantages of our adaptive partitioning approach compared to a static partitioning method and other adaptive partitioning methods (e.g. Kd-Tree).

Chapter 6 explains the GeoBalance framework that takes advantage of the algorithms

described in Chapter 5 to efficiently manage high-velocity real-time geospatial data. Particularly, GeoBalance is based on a microservice-based architecture (Section 6.2) that integrates a number of distributed and loosely connected services. In addition, we articulate how spatiotemporal workload is modeled in GeoBalance (Section 6.3) according to the statistics reported from multiple nodes.

Finally, in Chapter 7 we conclude this dissertation by summarizing the overarching research contributions. In addition, we discuss the future research directions related to optimal partitioning of geospatial data for cyberGIS analytics.

CHAPTER 2

MOTIVATING CASE STUDIES

This chapter reviews multiple related cyberGIS applications, which serve as the motivation for the algorithmic and architectural innovations that will be discussed in the following chapters. While each application serves specific analytical purposes, its associated algorithms are designed in a generalized fashion and may be applied to similar problems. For instance, UrbanFlow [Soltani et al., 2016] reveals intra-city human mobility patterns by combining real-time social media data with authoritative landuse maps through a generic spatial data synthesis approach that addresses one of the major problems in distributed spatial data processing [Eldawy, 2014]. All of the following applications have been implemented within a cutting-edge CyberGIS Gateway software environment [Liu et al., 2015, Wang et al., 2016].

2.1 MovePattern

MovePattern [Soltani et al., 2015a] is designed to achieve interactive and scalable visualization of massive movement datasets, where movement is defined as a trajectory between source S and destination D in time T. For instance, consider the movement of Twitter users throughout the United States in the August of 2014 which was close to 76 million movements. If we would visualize all the movements on a map, the end result could be too cluttered (and computationally expensive to generate) to present any useful information [Liu et al., 2013]. MovePattern aims to resolve this challenge by efficiently generating a multi-resolution view of the movements, providing various levels of details as users interactively explore patterns of data visualization (Figure 2.1).

The aggregation and summarization methods of geographical movements are implemented using the well-known distributed MapReduce model [Dean and Ghemawat, 2008] to provide seamless scaling for massive movement datasets. However, real-world geospatial datasets usually have heterogeneous spatial distributions [Soltani et al., 2015a]. For instance, Figure 2.2 reveals that the frequency of geo-tagged Twitter data in different locations follows a long tail distribution. Therefore, our MapReduce algorithms take the skewed distribution of data into consideration and devise an efficient scheme to balance the data and computation load among parallel computing units. In Section 3.2, we will detail our approach to address the spatial skew of movement data.

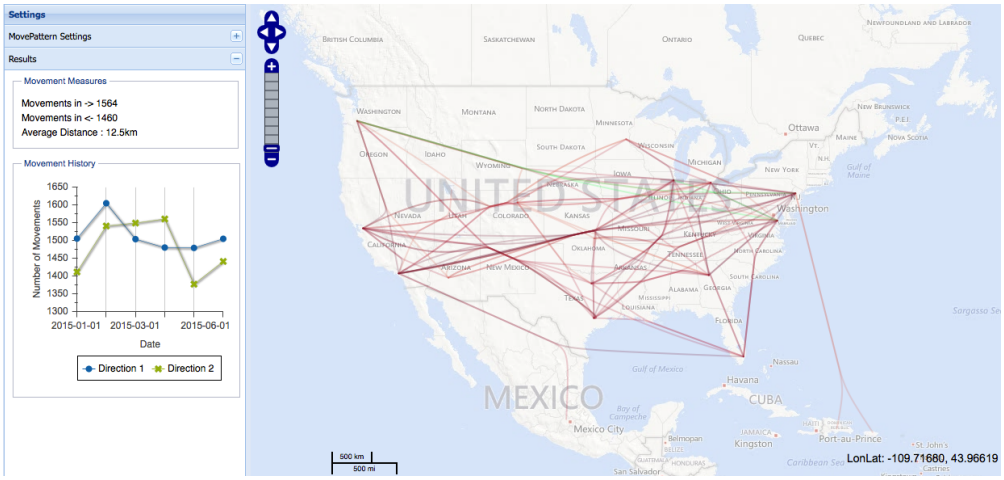


Figure 2.1: MovePattern web application

In addition, MovePattern emphasizes the importance of considering the workload of handling user-driven visualization requests (Section 3.2.3), which was generally overlooked in previous work [Zinsmaier et al., 2012, Daae Lampe and Hauser, 2011]. For instance, to evaluate the interactivity of the framework we used two different access patterns: 1) random access pattern and 2) focused access pattern. While the first pattern focuses on randomly generated user requests, the second one is modeling real-life scenarios where many users access a focused section of data, e.g., due to hotspots caused by common attention to certain locations.

MovePattern employs a vector-based visualization framework [Gansner et al., 2011] as

opposed to pixel-based approaches that were used in previous work [Zinsmaier et al., 2012]. In a pixel-based approach movement information is not straightforward to be linked back to the original data, making it impossible for users to get specific information about individual nodes/edges after a visualization product is presented. Therefore we use a vector-based approach to increase the analytical capabilities of the framework.

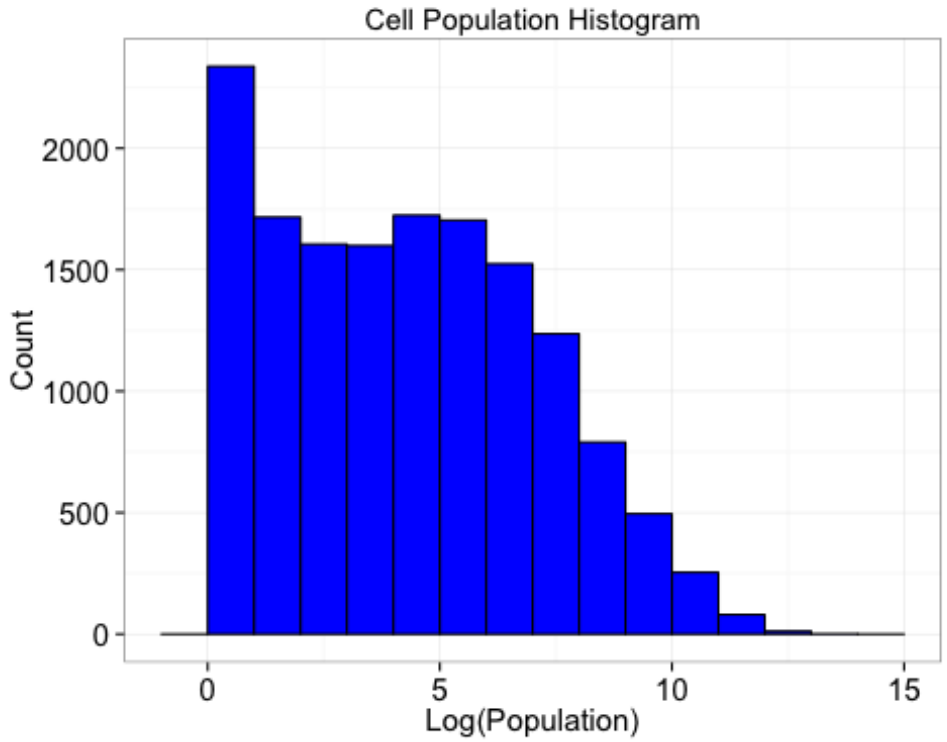


Figure 2.2: Number of Tweets from Oct-Dec 2014 in a Uniform Grid with $32km \times 32km$ Cells covering the Continental United States.

2.2 GeoHashViz

GeoHashViz [Soltani et al., 2015b] provides interactive analytics for understanding spatiotemporal diffusion of Twitter hashtags (Figure 2.3). Recent research studies have confirmed that Twitter hashtags can be used effectively to track diffusion of ideas including social memes, political trends and social movements [Kamath et al., 2013]. Hashtags are

proven to be more explicit and less noisy than the actual text of tweets, thus an effective medium to study collective diffusion of ideas in the virtual world [Romero et al., 2011,Kamath et al., 2013, Chang,]. GeoHashViz provides two types of analytics: 1) hashtag-based (spatial spread, focus points and spread metrics) and 2) location-based (Top-k popular hashtags, compare regions). Such analytics requires a series of pre-processing steps, implemented using Apache Hadoop and a highly interactive module that queries the result of pre-processing steps and generate analytical outcome.

A key challenge of this research is to model diffusion of ideas using spatially-aware metrics. For instance, to find the top-k popular hashtags in a selected region, we should query the selected region, aggregate the hashtags by count and select the most popular ones. However, this often might lead to a limited set of hashtags which are globally popular (e.g. #jobs) and does not provide any valuable insight into that region. Therefore, we employed a Tf-idf like metric [Sheng et al., 2010] to find the hashtags that are specifically popular in that region. The first part of the metric, called $CF - IRF_{h,C}(t)$ is to define hashtag frequency. Suppose that $O_l^h(t)$ is the frequency of hashtag l in location h at time t. Then, the hashtag frequency for geographical bound C is defined as:

$$CF_{h,C}(t) = \frac{\sum_{l \in C} O_l^h(t)}{\sum_{l \in C} O_l(t)} \tag{2.1}$$

Now we focus on the importance of hashtag h in C which relates to the distribution of locations which have used h over the entire region of study. We lay a $g_x \times g_y$ uniform grid on top of data points. Similar to the definition of O_l^h we define O_g^h as the occurrences of hashtag h in the gird cell g. Now we can define the hashtag importance as:

$$IRF_h(t) = \log \frac{|g_x \times g_y|}{|\{O_g^h(t) | O_g^h(t) \neq 0\}} \tag{2.2}$$

The final metric is then formulated as:

$$CF - IRF_{h,C}(t) = CF_{h,C}(t) \times IRF_h(t) \tag{2.3}$$

By calculating this metric for all the hashtags that have appeared in bound C at time period

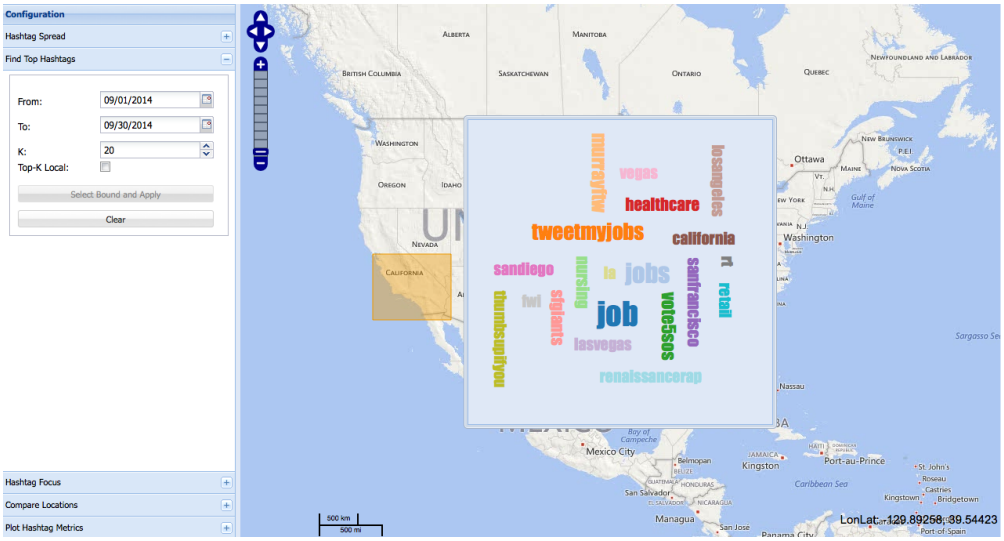


Figure 2.3: Visualizing top-k frequently used hashtags in a selected region using GeoHashViz web application.

of t, we can find the $top - K$ locally significant hashtags.

GeoHashViz requires access to massive records on a hashtag or a location. Therefore, to ensure the interactivity of the framework, we pre-compute some analytical features of hashtags and locations and combine them on-demand to answer user queries (Figure 2.4). The result of the pre-computation step is stored using a denormalized data model. The data model employs the denormalization optimization principles to reshape the data and store multiple copies of data grouped and indexed differently (according to the users workload) to provide faster read performance and avoid costly joins.

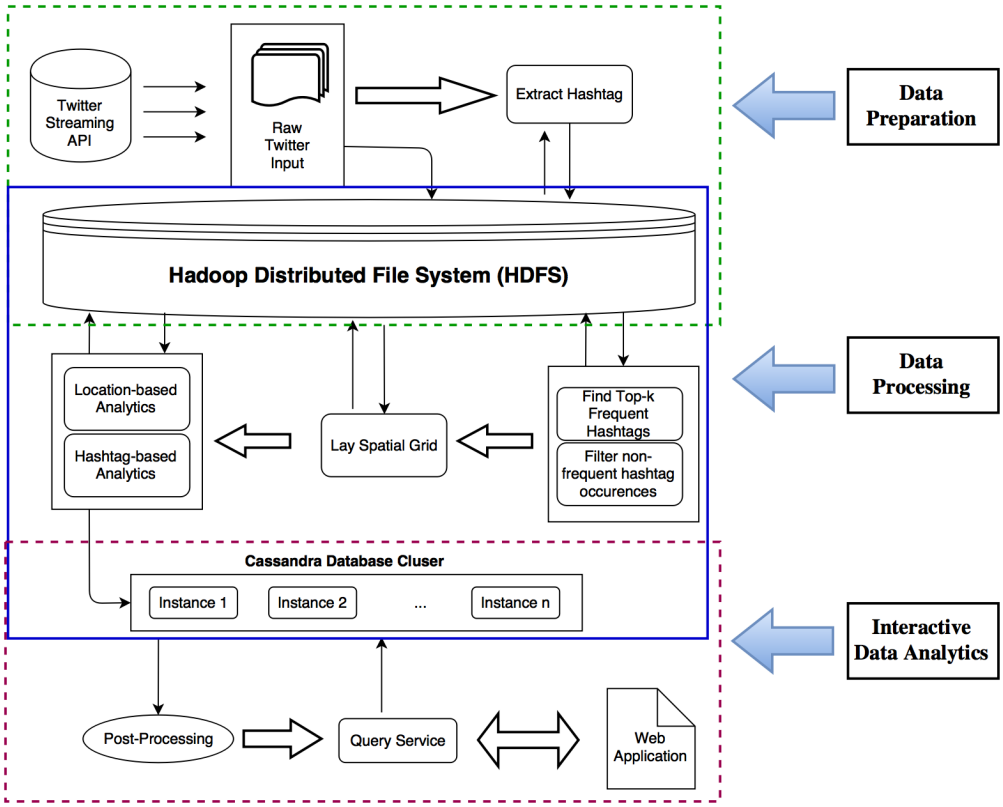


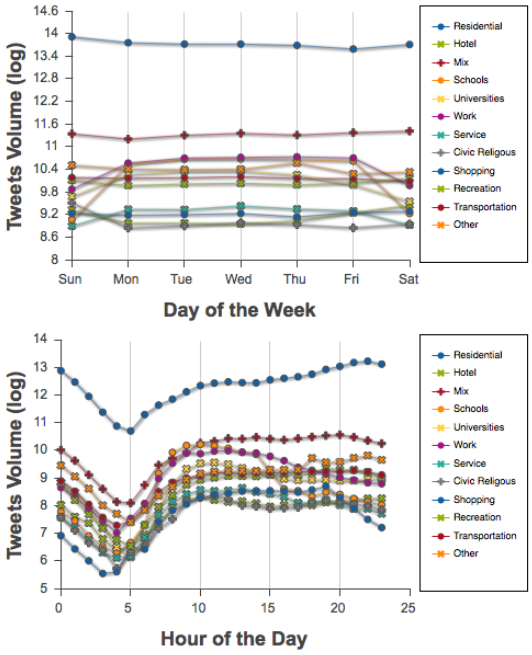
Figure 2.4: Architectural pipeline of GeoHashViz that includes pre-computing steps and on-demand services.

2.3 UrbanFlow

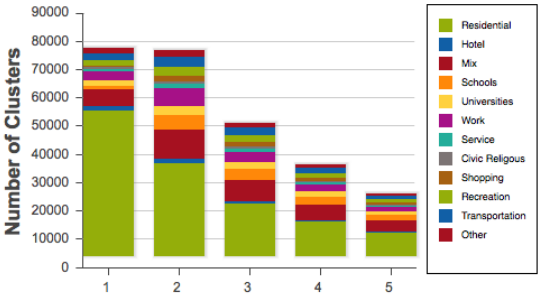
Recent work suggested that geo-tagged tweets are complementary sources of information to characterize urban landuse types [Frias-Martinez and Frias-Martinez, 2014]. Patterns extracted from geo-tagged tweets such as relative changes in number of tweets, number of users and user movements were found to correlate with the urban activity patterns [Wakamiya et al., 2011]. These interesting results motivate the development of open platforms for analyzing massive geo-located datasets. The demand for these platforms is high because of the need to monitor and understand urban dynamics.

However, the lack of accurate user activity context, makes it challenging to use geo-tagged social media data in urban dyanmics studies. UrbanFlow [Soltani et al., 2016] is designed to solve this problem by integrating social media data with traditional authoritative data sources such as landuse maps. For example, using UrbanFlow, researchers are able to find users' most frequently visited locations through spatial clustering and determine their context by integrating those locations with their corresponding landuse types (e.g. home, work, education, etc.). UrbanFlow provides visual insights to help understand urban spatial networks based on identifying common frequent visitors between different urban neighborhoods (Figure 2.5).

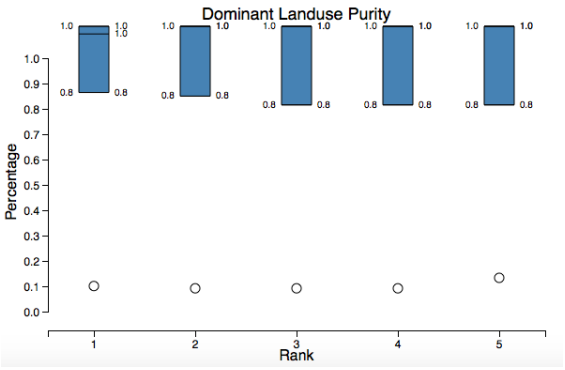
UrbanFlow includes a novel distributed approach to synthesizing spatial data in a distributed fashion using the MapReduce programming model. The integration problem in UrbanFlow can be solved through a classic point-in-polygon algorithm, while each geo-located tweet is represented as a point and each land parcel is represented as a polygonal area. This algorithm resolves the limitation of previous approaches [Zhang et al., 2015, esr, 2013] that only treat small or modest sizes of landuse datasets. In addition our approach sustains the inaccuracy in both polygon mapping and GPS locations by integrating points with their nearest polygon if an exact match does not exist. In Section 3.3, we describe the distributed approximated point-in-polygon algorithm used in UrbanFlow.



(a) Time series plot for number of tweets per hour of the day and day of the week for each landuse type



(b) Stack plot of dominant land use for top visited locations grouped by rank, confirming the high preferential bias to tweet from residential specially for top locations (rank1)



(c) box plot of clustering purity with most of the clusters corresponding to a single land use parcel (purity = 1.0)

Figure 2.5: Visual analytics in UrbanFlow's web application [Soliman et al., 2017]

UrbanFlow has been used by other researchers as an example of complex distributed geospatial applications to examine cloud operation management platforms and study scalability patterns. In one example [Keahey et al., 2017], researchers used UrbanFlow to test their operation management platform for multi-cloud environments. As another example, UrbanFlow was profiled in detail [Fu et al., 2018] to evaluate multiple dynamic data redistribution mechanisms in Apache Hadoop.

2.4 CyberGIS-Fusion

CyberGIS-Fusion is developed to demonstrate CyberGIS capabilities for a large number of users to perform computing and data-intensive, collaborative geospatial problem solving enabled by advanced cyberinfrastructure. The main contributions of CyberGIS-Fusion are:

- Provide a fast and scalable framework to insert/retrieve spatiotemporal data.
- Synthesize heterogeneous geospatial datasets in a single integrated environment.
- Facilitate seamless integration of multiple microservices to take advantage of cloud infrastructure.

To evaluate CyberGIS-Fusion, we chose the Beijing Taxi dataset [1] which includes more than 15 million taxi trips in Beijing from Feb 3rd to Feb 9th 2008 (Figure 2.6).

CyberGIS-Fusion is an example of real-time geospatial applications, where the ingestion-rate is high and the data should be immediately available to be queried. This is fundamentally different than batch-processing systems such as Apache Hadoop [Shvachko et al., 2010a]. Batch processing systems, while capable of dealing with massive datasets, are not tailored to handle real-time data where bulk loading of the data is not feasible.

One major challenge in designing a scalable data management framework for handling highly dynamic real-time data is related to how the data is distributed among different

[1] https://www.microsoft.com/en-us/research/publication/t-drive-trajectory-data-sample/

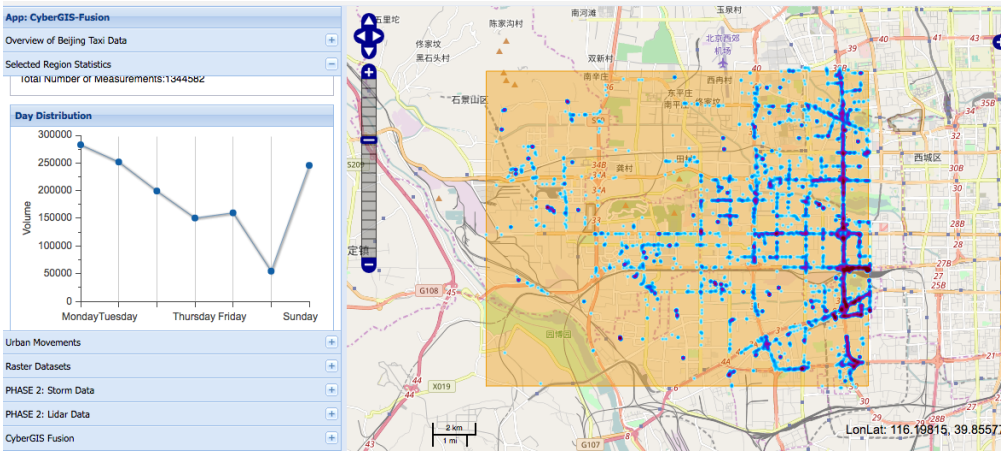


Figure 2.6: CyberGIS-Fusion web application for accessing Beijing taxi data.

nodes. Throughout the development of CyberGIS-Fusion we observed that geospatial data distribution may vary significantly over time (Figure 2.7), hence an effective partitioning scheme can dramatically improve the resource utilization and throughput of CyberGIS-Fusion. Particularly, we focus on adaptive workload-aware partitioning that a) uses both query and data distribution in determining the optimal partitions and b) adapts to changes in the data and query distribution over time. In Chapter 5, we discuss the algorithms for adaptive workload-aware partitioning of real-time geospatial data.

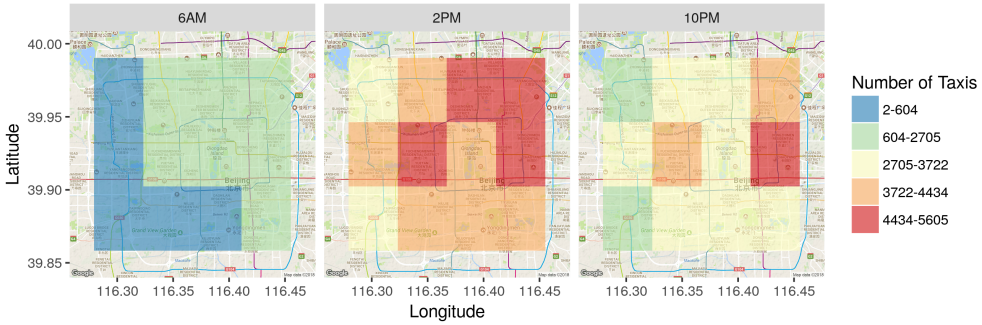


Figure 2.7: Changes in spatial distribution of taxi data over different hours of the day in the City of Beijing.

CHAPTER 3

STATIC WORKLOAD-AWARE LOAD BALANCING FOR DATA-INTENSIVE GEOSPATIAL APPLICATIONS

3.1 Overview

Geospatial scientists have taken advantage of advanced cyberinfrastructure to solve computation- and data-intensive problems [Wang et al., 2005]. Traditional research in this area focused on using high performance computing to design scalable algorithms and frameworks [Tang et al., 2011, Liu and Wang, 2015]. However, the focus has shifted dramatically in the past decade to cloud-based approaches [esr, 2013, Soltani et al., 2016]. Particularly in the case of data-intensive applications, cloud computing provides scalable functionalities to distribute geospatial data and parallelize spatial operations through exploiting data parallelism.

MapReduce has been adopted as a major programming paradigm of cloud computing [Dean and Ghemawat, 2008]. The paradigm contains two main steps: map and reduce. In the map phase input data is converted into series of intermediate (key, value) pairs. Then after shuffling and sorting the intermediate pairs, the reduce phase will collect all the (key, value) pairs that have the same intermediate key (Figure 3.1). This programming framework can support automatic parallelization as the map operations are considered independent and can be done in parallel. In addition, MapReduce can scale horizontally by seamlessly adding new computing nodes for data hosting and processing.

Hadoop [Shvachko et al., 2010b] is an open-source implementation of MapReduce, which is built on top of a distributed file system called Hadoop Distributed File System or HDFS. Hadoop facilitates data-intensive computing on large commodity clusters by providing capabilities such as data distribution, parallel computing and fault tolerance. From early

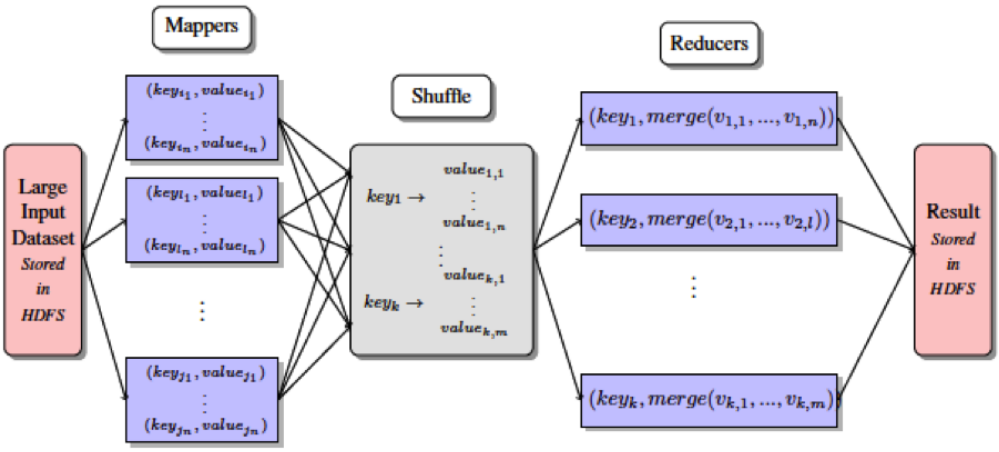


Figure 3.1: MapReduce programming model.

days of Hadoop, geospatial communities have embraced the framework and used it to solve data-intensive problems [Cary et al., 2009b, Liu et al., 2010]. However, previous work has also proved that adapting spatial algorithms to the MapReduce programming model is a challenging task [Eldawy and Mokbel, 2015]. First, Hadoop takes advantage of functional programming model which requires fundamentally different thinking on how to design optimized algorithms. In addition, Hadoop does not natively exploit spatial data characteristics. Therefore, customized techniques need to be designed to handle distributed processing of spatial data in the Hadoop ecosystem.

Previous research has extensively studied the issue of data partitioning and indexing on Hadoop [Eldawy and Mokbel, 2015]. The main challenge regarding distributing spatial data in Hadoop is the fact that data in HDFS works in an append-only fashion and data in HDFS is immutable. To address this limitation, researchers have suggested to use a sample of the data to build the index prior to storing the data in HDFS [Eldawy, 2014]. The index is built in two different granularities: the global index that is accessible to all parallel workers and the local index that is built locally for faster spatial operations. Using the two-level indexing scheme, whenever a spatial operation has to be performed, the system can find the associated data blocks using the global index and find corresponding answers by looking into the local index of the data blocks. While the overall design stays the same, different

researchers have used different indexing algorithms such as uniform grid [Aji et al., 2013a], Z-curve [Cary et al., 2009a], R-tree [Eldawy, 2014] or quadtree [Eldawy et al., 2015b].

Compared to extensive work on partitioning and indexing methods for static geospatial applications, limited work has been done to consider expected workload into such approaches. As previous researchers indicated the tight coupling between spatial domain decomposition and task scheduling is problematic due to the inability of this approach to scale for new data sources, operations and architectures [Wang and Armstrong, 2009]. Therefore, solely focusing on data and operations is "logically inappropriate" for designing efficient domain decomposition and task scheduling strategies [Wang and Armstrong, 2005]. We attempt to address this issue by introducing workload and focusing on problems that are sensitive to workload and more concerned with the variability of the computational tasks that are applied to the data, rather than the distribution of the data. Specifically, we study the methods and algorithms for spatial computational domain representation, using two well-known examples: 1) multi-resolution data aggregation and 2) approximate point-in-polygon problems. This study will complement the previous effort on using Hadoop for geospatial problems by shedding light on a new dimension of distributed processing of geospatial data.

One of the most significant issues in MapReduce applications is to deal with skew [Kwon et al., 2012]. While in Hadoop, input data is evenly distributed among mappers, based on the configuration variable *dfs.block.size*, the applications can still suffer considerably from skew among reducers. The reason for existence of such skew is the inability of Hadoop to dynamically balance the load among reducers. Particularly real-world spatial data are usually highly skewed and therefore a custom partitioner is required to balance Hadoop-based computation. This skew is beyond the distribution of the original data and more related to how the computation model is defined. Similar to the definition of computational intensity, the problem of skew relates to spatial domain decomposition, spatial operations as well as the distribution of data. Therefore, to avoid load imbalance we have to consider the computational intensity of the problem in designing distributed algorithms [Soltani et al.,

2015a].

Our approach has two main steps. First, we use either a theoretical analysis of the problem or a sample of workload to generate a computation plan. This will define how the computation is going to be divided among reducers and is similar to the global index which have been used in previous research. This step may be accompanied by spatially aware partitioning of input data [Soltani et al., 2016]. Second, we use the computation plan to run its corresponding MapReduce job. In the mapper stage, we use the global index to determine which reducers the data should be assigned to. Furthermore, the reducers will receive their portion of data, performs pre-processing steps such as spatial indexing and and compute the final result.

The generation of a computation plan can be understood as a spatial computational domain decomposition problem that is defined based on the nature of the problem and computational intensity analysis of it [Wang and Armstrong, 2009]. This analysis may vary from problem to problem. Therefore we cannot solely rely on a fixed data distribution scheme, as done by some previous work [Eldawy, 2014, Aji et al., 2013a], and thus have to consider a mechanism that can provide the decomposition for each problem (workload-awareness). The computational intensity relies on the spatial distribution of data, complexity of spatial operations, as well as surrounding geographical regions in some cases. Therefore, the approach should be able to handle possibly overlapping regions for the decomposition. In the rest of this chapter, we elaborate our approach by using two case studies.

We evaluate our approach from three different perspectives. First, we study the advantage that our approach provides in terms of reducing the computational and data skew among reducers. Particularly, we study how reducers' data and computational loads are affected by taking advantage of our spatial computational domain representation. Our second experiment is focused on determining the optimal granularity for our spatial domain decomposition, which is a critical issue for large-scale geospatial applications [Wang and Armstrong, 2009]. On one hand, the granularity should be coarse enough that justifies the computational ben-

efits of decomposition, compared to the overhead of parallelization. On the other hand, the granularity should be sufficiently fine to allow for sub-domain to be processed in parallel. Therefore, we evaluate our approach in both examples against multiple granularities to determine the most efficient setting. Finally, since both examples are designed to be interactive applications, we conduct experiments by simulating expected workload on the applications. The goal of these experiments is to ensure that interactivity of the applications stay intact under both temporally and spatially intense query workloads.

3.1.1 Geo-tagged Twitter Data

In both applications we heavily rely on geo-tagged Twitter data to demonstrate the value of our approach. We collect geo-tagged tweets using a custom crawler specifically designed for collecting tweets containing geographic coordinates or place information using Twitter's open access streaming API. The crawler records user id, location, time, text and some associated flag variables (e.g. whether the tweet is a retweet). We exclude data outside of the North American continent and collections over 2.5M tweets on a daily basis.

3.2 Movement Aggregation and Summarization

3.2.1 Background

Large amounts of data pose unique requirements for generating visualization in a scalable fashion. Liu, et al. [Liu et al., 2013] formulate this problem by distinguishing the "interactive scalability" from "perceptual scalability". Perceptual scalability means avoiding plotting every possible point so that users are not overwhelmed. Interactive scalability means that the visualization provides fast responses to multiple users queries and eliminates long latencies in user interaction. Data reduction techniques and detail-on-demand capability [Shneiderman, 1996] can provide an answer to the perceptual scalability problem [Zinsmaier et al., 2012].

However, we still may experience significant latency in the user queries that negatively affect the user experience. Therefore, it is a common practice in large-scale visualization systems to reduce query time by offloading some computation to the server side and adding a pre-processing step [Keim, 2005, Liu et al., 2013].

Previous studies explore three groups of reduction methods for spatial movement data: clustering, density-based and binned aggregation. Clustering approaches [Guo, 2009, Jia et al., 2011] re-model the problem as a graph community detection problem which enables them to define multiple level of abstractions for the movements. However, these approaches tend to perform poorly on large datasets, since they require calculating modularity between large numbers of nodes. Density-based approaches use kernel density estimation with adaptive bandwidth to provide multi-scale view of the movement graph [Zinsmaier et al., 2012, Daae Lampe and Hauser, 2011]. In contrast to clustering methods, density-based approaches are usually very fast due to the fact that they can be efficiently implemented using GPU. However, these methods employ a pixel-based approach where the visualization cannot be linked back to the original data. Therefore the user cannot get specific information about the edges after the visualization is produced. Binned aggregation is another approach which defines the aggregates of the data based on pre-defined spatial bounds [Liu et al., 2013]. One main advantage of this approach is that it can be easily parallelized over multiple computing nodes [Eldawy et al., 2015b]. In addition, binned aggregation can form multiple views of data using a hierarchical definition of bins. For demonstration of this work we adapted a form of binned aggregation, known as hierarchical spatiotemporal data cube [Cao et al., 2015] (3.2), which has been used to model multi-resolution movements [Padmanabhan et al., 2014]. However, the framework is not limited to this data model and can be easily applied to other binned aggregation data schemes as well.

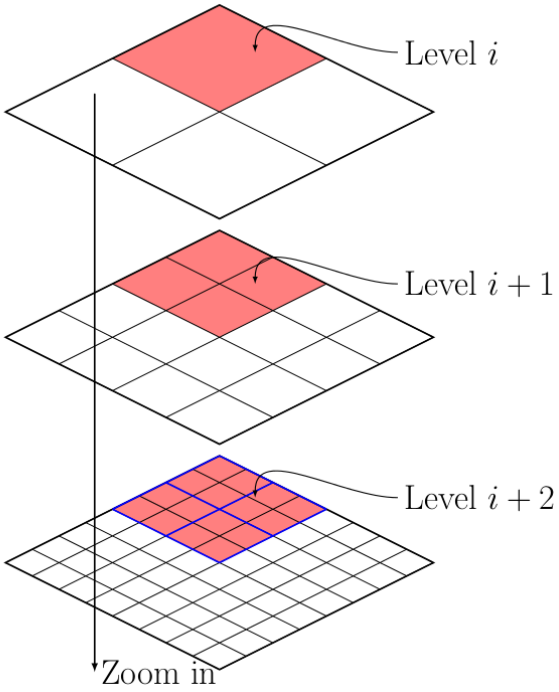


Figure 3.2: MovePattern employs hierarchical data cube model.

3.2.2 MapReduce Algorithms for Movements Aggregation and Summarization

In the MovePattern framework [Soltani et al., 2015a], there is an important need for processing geographical movements and produce a multi-resolution view of collective movements. This includes three main steps:

- Aggregate the movements based on a hierarchical data model.
- Summarize the aggregated graph by eliminating nodes that are not deemed as "significant" compared to their local neighborhood.
- Filter edges associated with nodes that are filtered out in the previous step (using Bloom Filter [Bloom, 1970]).

To build the spatial decomposition scheme, we used the recursive bisection approach that partitions a space into a set of rectangles [Berger and Bokhari, 1987]. In this method at each

step, we divide the region into two sets to minimize the imbalance. One main variation of our approach from the original recursive bisection method is that instead of alternating the cut axis at each level, we choose the axis that achieves better balance among two sets.

The result spatial decomposition scheme is saved on HDFS and can be loaded at the beginning of each MapReduce job, and indexed using R-tree, to determine how the computation is assigned to corresponding reducers. Algorithms 1 and 2 are examples of mapper and reducer stage using this approach.

Algorithm 1: Mapper for spatial aggregation

```
function mapper(movement)
    s ← movement.source ;
    t ← movement.target ;
    for l = 1...L do
        id1 ← getCellId(s, l) ;
        id2 ← getCellId(t, l) ;
        merge(graphs[l], id1, Node(s, 1)) ;
        merge(graphs[l], id2, Node(t, 1)) ;
        if d(s, t) < threshold(l) then
            merge(graphs[l], id1, Edge(id1, id2, 1)) ;
function cleanup()
    for l = 1...L do
        foreach key in graphs[l] do
            p ← rtree.search(key) ;
            emit(p, [l, graphs[l].get(key)]) ;
```

Algorithm 2: Reducer for spatial aggregation

```
function reducer(key, Node[] values)
    foreach value in values do
        merge(graphs[value.level], value.id, value) ;
    for l = 1...L do
        foreach key in graphs[l] do
            x ← graphs[l].get(key) ;
            write("Nodes", x.level, x.coor, x.count) ;
            write("Edges", x.neighbors) ;
```

While multi-resolution spatial aggregation provides a generalized view of data, it can be still too large to convey any clear patterns to users in a visualization interface. For instance, if we divide North America into $512km \times 512km$ cells, we will end up with 468 grid cells which can have up to 109278 edges among them. Therefore, even for very coarse-level view of data, we get a very cluttered graph; hence the result from node aggregation step needs to be summarized for a less cluttered visualization.

Our summarization technique filters less "significant" nodes (grid cells) by assigning them a score, reflecting how large their degree is comparing to neighboring cells. Then by comparing the score to a pre-defined threshold we can decide whether to keep or drop a node from the graph. As previous research has pointed out [Padmanabhan et al., 2014] geographical distribution of real-life location-based data is highly skewed, with a small number of places contributing most of the activities. Therefore, the definition of "significant" nodes should be localized to their surrounding sub-regions as opposed to using the same significance measure for the whole region of study.

The local neighborhood of point p is defined as $\{q \in P : d(p,q) < r\}$. Here P is the set of all points in the graph (in the same spatial resolution as p), $d(p,q)$ is the great-circle distance between p and q and r is the neighborhood radius (Figure 3.3). By reducing r we will have a more strict definition of a neighborhood which will lead to having more points in the final graph. The value of r can be adjusted for each spatial resolution.

Using the neighborhood formulation, calculating local rank can be simply defined as the percentile rank of point p in $neighborhood(p)$:

$$localranking(p, neighborhood(p)) = \frac{\Sigma_{q \in neighborhood(p)} cf(p,q)}{|neighborhood(p)|} \tag{3.1}$$

Where:

$$cf(p,q) = \begin{cases} 1 & \text{if } degree(p) \geq degree(q) \\ 0 & otherwise \end{cases} \tag{3.2}$$

To model this problem using MapReduce, we have to be cautious to avoid unnecessary communication among different nodes. In the naive approach each node send their information to every other node, and help them find about their neighborhoods. However, this will lead to very expensive communication overhead. Instead we take advantage of the partitioning scheme that was built in the initial stage of the data processing module to prune many choices and end up with only a small set of cells as neighbor candidates. The uniform structure of the grid enables us to easily extract the set of cells, which are in a certain distance from the current cell. The outline of this MapReduce based approach is explained in Algorithms 3 and 4. The input of this job is the "Node" output of the aggregation step.

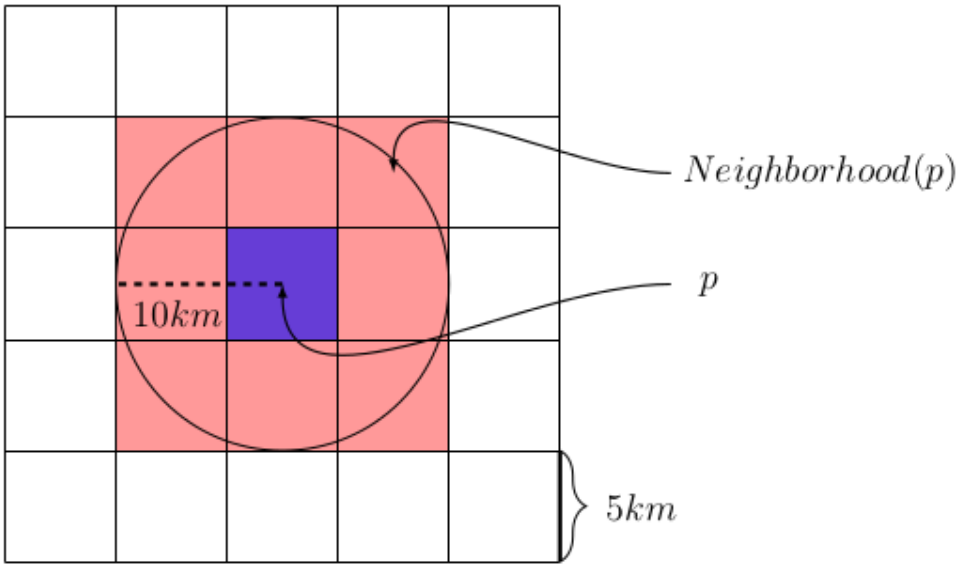


Figure 3.3: MovePattern requires a summarization step to eliminate insignificant geographical nodes.

3.2.3 Experiments

To evaluate our approach, we designed an experiment on processing movements from Twitter users starting August 1st, 2014 through October 31st, 2014 (Table 3.1). The tweets were collected and geo-referenced based on the Twitter Streaming API [Padmanabhan et al., 2014] and movements generated by forming a spatiotemporal trajectory for each user. To

Algorithm 3: Node Summarization Mapper

```
function getNeighborPartitions(node, level)
    offset ← cell_len[level] × r[level] ;
    result-set ← rtree.findNeighbors(
        node.coor, offset) ;
    return result-set ;
function mapper(node)
    foreach p in getNeighborPartitions(node, node.level) do
        emit(p, node) ;
```

Algorithm 4: Node Summarization Reducer

```
function reducer(key, Node[] values)
    for l = 1...L do
        rtrees[l] ← new rtree() ;
    foreach node in values do
        if !rtrees[node.level].contains(node) then
            rtrees[node.level].add(node) ;
    for l = 1...L do
        foreach node in rtrees[l] do
            offset ← cell_len[node.level] × r[node.level] ;
            result-set ← rtrees[node.level].findNeighbors(
                node.coor, offset) ;
            rank ← percentile rank of node.count in result-set ;
            if rank ¿ threshold then
                write("SummaryNode",
                    node.id, node.level, node.coor, node.count) ;
```

divide the data into multiple spatial resolutions, a hierarchical uniform grid is formed with 10 levels, representing different levels of details for the North America continent. At the finest level, a uniform grid with cell size of 30 arc seconds (approximately 1 $km \times 1\ km$) is formed and the cells are iteratively merged to form the uniform grids on the higher level. The merging operation is done in an exponentially increasing fashion forming cell sizes of 2, 4, ..., 512 km. In addition, each cell is presented using the centroid of all the points (location of tweets) within the cell.

The experiments were conducted on the cyberGIS supercomputer called ROGER[1] supercomputer at the University of Illinois at Urbana-Champaign. For the Hadoop cluster, we used 8 nodes each including 2 Intel Xeon E5-2660 2.6 GHz CPU (20 cores total), 256 GB of memory and 800GB of SSD hard drive. In this section, we present only two of the experiments (see the original paper [Soltani et al., 2015a] for more experiments). First, we evaluate our spatial decomposition method to test the load on each reducer. Table 3.2 shows the statistics on reducers load when aggregating data for the 3-month dataset. This result confirms that using the spatial decomposition method, described above, we can divide our study space into multiple regions with similar computational load, which as we explained is a critical issue in designing MapReduce-based algorithms. The next experiments focus on the performance of spatial aggregation and summarization methods against the three test datasets. Table 3.3 shows how aggregation and summarization methods affect the number of locations and movements among them. These two experiments demonstrate the ability of our approach to scale and dramatically reduce the movement graph size by intelligently aggregate movements in multiple resolutions.

Duration	Tweets	Users	Movements	Size(GB)
1-month	107M	2.2M	76M	2.77
2-month	205M	2.9M	136M	5.43
3-month	368M	3.5M	179M	9.88

Table 3.1: Input data statistics

[1]Resourcing Open Geospatial Education and Research

# of Reducers	Avg	Std	Min	Max
8	5,853,967	120,348	5,709,732	6,037,848
16	2,926,984	73,670	2,821,192	3,096,001
32	1,463,492	58,673	1,344,301	1,609,537
64	731,746	45,466	654,939	846,538

Table 3.2: The reducers' load statistics in terms of associated keys to each reducer.

Dataset	Nodes	Edges
1 month	$2.67M \rightarrow 939K$	$33.98M \rightarrow 4.66M$
2 months	$3.18M \rightarrow 1.08M$	$49.27M \rightarrow 6.88M$
3 months	$3.48M \rightarrow 1.16M$	$59.55M \rightarrow 8.44M$

Table 3.3: Combined Effect of aggregation and summarization on graph size

Interactive Scalability

To evaluate interactive scalability, we simulate two categories of queries that represent two most common query patterns of users:

1. **Population Query Pattern**: Queries are distributed in a weighted fashion around the region of study (here North America) with more queries on sub-regions with higher number of tweets.

2. **Hotspot Query Pattern:** Queries are focused on a specific relatively small region resembling occasional situations which an outburst will lead to large number of focused access. For instance, a political visit or a natural disaster can attract user attentions to a certain area.

The underlying assumption in both access patterns is that the framework is likely to get more queries from regions where there are more tweets. Therefore, we generated the query bounding box according to a sample of tweets, where more crowded regions are more likely to be presented in the sample. In addition, the spatial resolution is randomly chosen in a uniform fashion from $\{1, 2, ..., 10\}$.

For our experiments, we generated 3 sets of 2000 queries for overall and focused query pattern. Then Apache JMeter [2], a load testing tool, is used to send this queries to MovePattern with different rates of queries per second and the response time is measured. We launched 2000 queries in the duration of 50, 75 and 100 seconds, on the 3-month dataset. After performing aggregation and summarization, this dataset contains over 1.16M nodes and 8.44M edges. Table 3.4 shows statistics (average, median and 90% percentile) on the response time of queries for both normal and focused patterns. The result shows that MovePattern can sustain relatively large number of simultaneous queries, each based on different resolutions and regions.

Population Pattern

Duration(s)	Average	Median	90% Percentile
50	42	29	96
75	34	23	79
100	31	21	72

Hotspot Pattern

Duration(s)	Average	Median	90% Percentile
50	36	24	82
75	31	21	69
100	28	19	63

Table 3.4: Stress test for 2000 queries on 3-month dataset (in ms)

3.3 Distributed Approximate Point-in-polygon

3.3.1 UrbanFlow Architecture

UrbanFlow collects input data using the Twitter Streaming API [Padmanabhan et al., 2014]. In the first step, two filters are applied to tweets: 1) spatiotemporal filter: to filter tweets

[2] http://jmeter.apache.org/

based on spatial extent and timespan of the study and 2) user filter: to filter tweets from users with very low engagement, indicating a very sparse present over time. The second filter is particularly critical, since we want to capture actual mobility patterns of people and eliminate inactive users with noisy behaviors.

In the next step, the processed tweets dataset is integrated with authoritative dataset using our novel distributed point-in-polygon algorithm (refer to Section 3.3.2). Then we identify frequently visited locations of each user (Section 3.3.1). Finally the clusters are joined with the user-defined partitioning scheme to enable region-based analytics. All the above steps are implemented using Apache Hadoop. Figure 3.4 demonstrates the architecture of UrbanFlow.

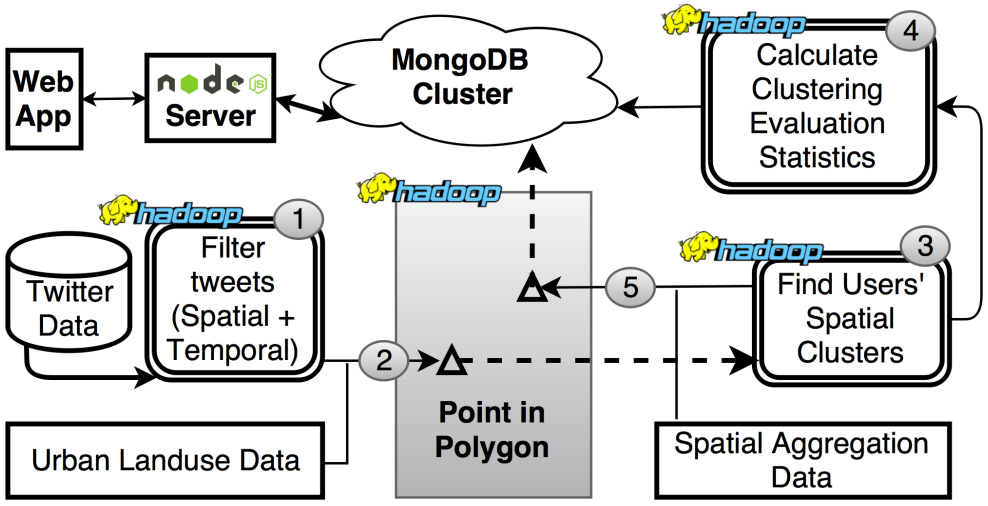


Figure 3.4: UbranFlow Architecture

Identifying Frequently Visited Locations

Previous studies have indicated that human mobility patterns can be explained by preferential return to few frequently visited locations (FLV) such as home, work, school, etc [Soliman et al., 2015]. There are multiple methods to identify the FLVs, ranging from simple solutions such as imposing a fixed grid on the users' locations [Cao et al., 2015] to more advanced methods such as spatial clustering techniques [Bora et al., 2014]. We identify FLVs by using DBSCAN clustering algorithm [Ester et al., 1996] on each user's recorded locations. DB-

SCAN method does not assume a predefined number of clusters or any particular spatial shape. This is particularly beneficial for our application, due to heterogeneous nature of human mobility. The result of this step is a series of clusters per user, marked by their spatiotemporal characteristics as well as their rank in the user's everyday activities. The rank of a cluster demonstrates the tendency of the user to return to that FLV and determined based on the number of tweets in the cluster. For instance, for most of the users the first rank cluster is considered as home, while work/school may appear in the lower ranks [Soliman et al., 2015].

We also introduce the purity metric of clusters, to evaluate the choice of clustering parameters. The purity of a cluster is defined as the frequency of the most common landuse parcel to total number of points in that cluster. For instance, the purity of 1.0 indicates that the cluster is completely located in/near one land use parcel. For a reasonable choice of clustering parameters, we expect the most clusters to have high purity number.

3.3.2 Point-in-polygon Approach

Spatial Decomposition Method

UrbanFlow [Soltani et al., 2016] implements a scalable point-in-polygon algorithm that integrates geo-tagged tweets with fine-resolution authoritative landuse data. There are two main challenges in designing such algorithms in a distributed fashion. First, in addition to the large point dataset (tweets), our polygon dataset is also large and may contain hundreds of thousands of polygons. Previous work on this problem [Zhang et al., 2015, esr, 2013] suffered from the inability to scale when the number of polygons is large. This limitation is due to the fact that these methods attempt to send all the polygons to every worker, which is clearly inefficient for large number of polygons. In other word, these approaches only exploit the parallelism to the points dataset. Second, due to the existence of noise and inaccuracy in both measurements and movement patterns of humans, the approach should be able to

approximate point-in-polygon process to assign points to desirably close polygons, in case an exact match does not exist (Figure 3.5).

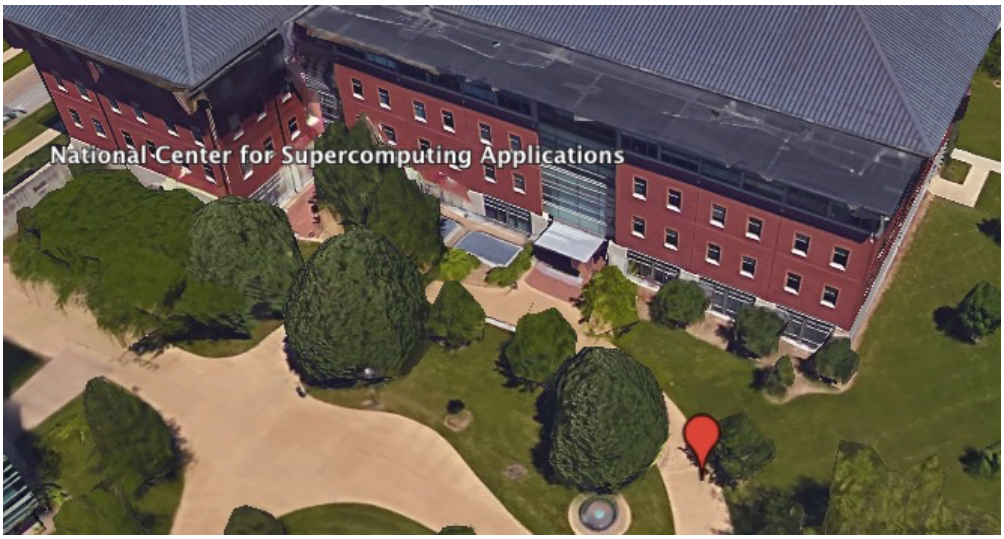


Figure 3.5: Example of inaccuracy in human mobility patterns: the geo-tagged location of tweet may not perfectly match with the building that the user is associated to. Our approach will assign the geo-tagged tweets to its closest land parcel if an exact match does not exist.

To address the above issues, we developed the following approach. First, we pre-process the polygons and break them into smaller spatially contiguous chunks. This step is interleaved with the spatial decomposition method. Therefore, for each decomposition, the corresponding polygons that overlap with the region is determined as saved to a separate file in HDFS.

Second, we expand the original point-in-polygon operation to find the *closest* point-in-polygon. The goal here is to find the polygon that is closest to a given point, considering a distance threshold ($minDist$). Let's consider point p which lies into partition i with bounding box BB_i . If p matches a polygon that is presented in partition i, our work is done (no approximation required). However, in case p does not match any polygon in partition i, we should look for the polygon that has the minimum distance to p. The main issue is that in partitions' boundaries, this polygon may not intersect with BB_i . We resolve this issue by expanding the polygons in partition i to includes any polygon that is located in distance

$minDist$ of bounding box BB_i . While we expand the polygons in partition p beyond BB_i , we will not change the representative bound of partition i. This guarantees that a point will match with one and only one partition. Otherwise an expensive post-processing step is required to handle points that are associated with multiple partitions. Figure 3.6 illustrates this process.

The Algorithm 5 explains the process of splitting the original polygons into smaller sets. The division criteria ($getPartitionWithHighestCI$) is defined by the user according to the desired model for computational intensity of the problem. Some candidates are number of polygons, number of points based on a sample of data or the area of the region. We also takes advantage of Median of Medians algorithm [Blum et al., 1973], which can calculate the median of an array in linear time. In addition, Algorithm 6 explain how the splitting algorithm is employed to generate computational spatial domain decomposition among reducers.

MapReduce Algorithms

Once the spatial decomposition is devised, a MapReduce job is designed that reads the space decomposition index in the mapper stage and determines which reducer should receive the point. The reducers (Algorithm 8) take care of the main point-in-polygon process. Each reducer reads and indexes its associated partition only once and then use that to find the closest polygon associated with each point. The spatial indexing (for bounds and partitions) are done using R-tree.

3.3.3 Experiments

We conducted an experiment using geo-tagged tweets from January 2013 to January 2016 (2.42 billions of tweets for US boundaries) and filtered them to only keep tweets that match Chicago's bounding box (nearly 47 million tweets). To identify the landuse type of each

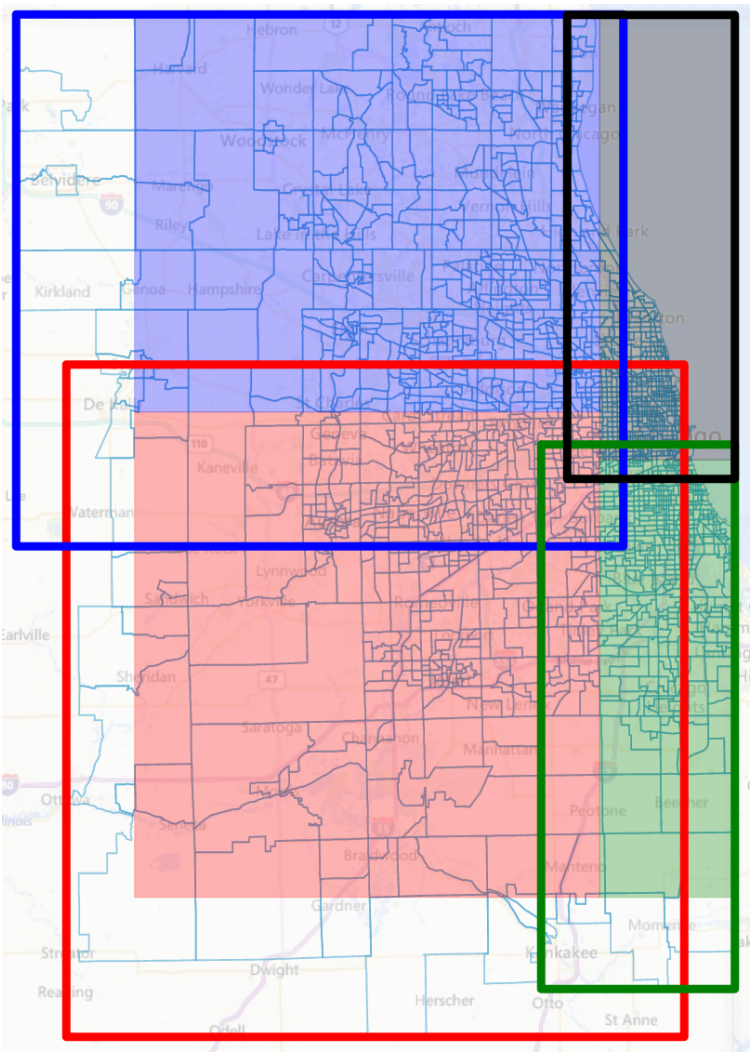


Figure 3.6: Example of splitting space into 4 partitions, illustrating how each partition includes more polygons than its representative bounding box (colored rectangles) to enable *closest* point-in-polygon operation. The non-filled rectangles show the bounding box of all the polygons in each partition.

Algorithm 5: Split original polygons into partitions

```
function divideBounds(partition, divideDirection)
    coordinates ← partition.polygons.centroid[divideDirection] ;
    median ← MedianOfMedians(coordinates) ;
    newBounds ← newArray() ;
    if divideDirection is x − axis then
        newBounds[0] ←
          [partition.minX, partition.minY, median, partition.maxY] ;
        newBounds[1] ←
          [median, partition.minY, partition.maxX, partition.maxY] ;
    else
        newBounds[0] ←
          [partition.minX, partition.minY, partition.maxX, median] ;
        newBounds[1] ←
          [median, median, partition.maxX, partition.maxY] ;
    return newBounds ;
function split(partition, threshold)
    divideDirection ←
      alternate(parition.divideDirection) ;
    newBounds ←
      divideBound(partition, divideDirection) ;
    newPartitions ← new Partition[2] ;
    foreach polygon in partition.polygons do
        if intersect(newBounds[0], polygon, threshold) then
            newPartitions[0].addPolygon(polygon) ;
        if intersect(newBounds[1], polygon, threshold) then
            newPartitions[1].addPolygon(polygon) ;
    newPartitions[0].divideDirection ←
      divideDirection ;
    newPartitions[1].divideDirection ←
      divideDirection ;
    return newPartitions ;
```

Algorithm 6: Generate computational spatial domain decomposition to guide reducers.

```
function generate(polygons, maxPartitions, maxReducers, distanceThreshold)
    partition ← new Partition(polygons) ;
    partitions ← new Array() ;
    partitions.add(partition) ;
    while partitions.length ≤ maxPartitions do
        splitPartition ←
          getPartitionWithHighestCI(partitions) ;
        partitions.add(
          split(splitPartition, distanceThreshold)) ;
        partitions.remove(splitPartition) ;
    reducersOrder ←
      generateRandomPermutation(1, maxReducers) ;
    bounds ← new Array() ;
    for i = 1...maxPartitions do
        partitions[i].reducerId ←
          reducersOrder[i%maxReducers] ;
        writeToHDFS(partitions[i]) ;
        bounds.add(getBounds(partitions[i])) ;
    writeToHDFS(bounds) ;
```

Algorithm 7: Mapper for point-in-polygon

```
tree ← null ;
function setup(jobConfig)
    bounds ← ReadFromHDFS(jobConfig.boundsFile) ;
    tree ← BuildQuadTree(bounds) ;
function mapper(point)
    parition ← tree.find(point.coordinates) ;
    emit(partition.reducerId, point) ;
```

Algorithm 8: Reducer for point-in-polygon

```
tree ← null ;
distanceThreshold ← 0 ;
function setup(jobConfig)
    id ← jobConfig.getReducerId() ;
    polygons ←
     ReadFromHDFS(jobConfig.partitionFiles[id]) ;
    tree ← BuildQuadTree(polygons) ;
    distanceThreshold ← jobConfig.distanceThreshold ;
function reducer(key, Point[] points)
    foreach point in points do
        matchedPolygons ← tree.findByRadius(point, distanceThreshold) ;
        f(p) ← {distance(p, point) | p ∈
         matchedPolygons} ;
        closestPolygon ← arg min_p f(p) ;
        write(closestPolygon.id, point) ;
```

tweet's location, we use a highly detailed landuse map of Chicago [3] that includes 468,641 parcels. The approximation threshold to consider the closest polygon is set to 100m.

We deploy UrbanFlow on the ROGER supercomputer. ROGER's Hadoop cluster for this experiment, includes 11 nodes, each having 20 cores and 256GB of memory. Based on the system's setting, it can run the maximum of 121 mappers or 55 reducers at the same time (the memory requirement for mappers and reducers are different in ROGER). The block size for the HDFS, which determines the number of mappers is set to 64MB, with replication factor of 2 and the number of reducers is set to 64, unless explicitly noted.

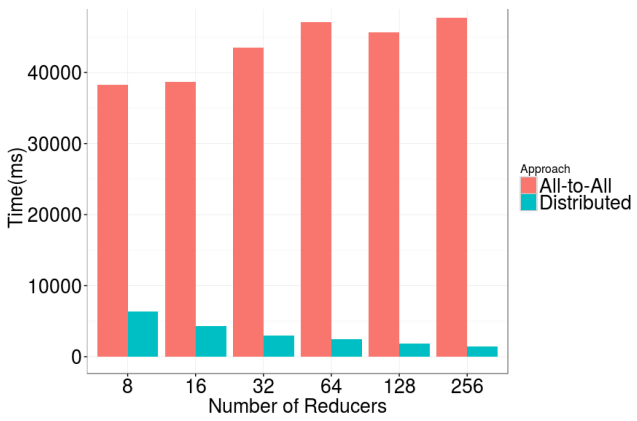
Point-in-polygon Algorithm

First experiment compares our point-in-polygon approach with the existing approach of sharing all the polygons with every worker (from now on we refer to this approach as *all-to-all*). One important note is that based on the Hadoop cluster configurations, running the all-to-all approach may not be possible. This is due to the fact that in a standard

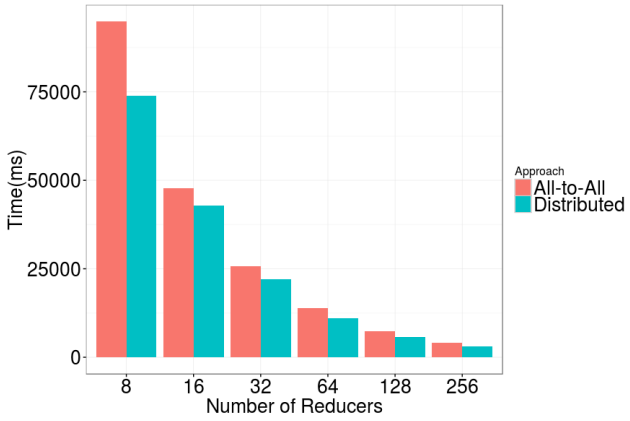
[3] Landuse Inventory for Northeastern Illinois - `http://www.cmap.illinois.gov/data/land-use/inventory`

Hadoop cluster the available memory for one mapper/reducer is quite limited (usually less than 1GB). The limitation is closely related to the nature of Hadoop clusters that are intended as commodity clusters. Therefore, in most of the Hadoop clusters running all-to-all approach for a slightly large number of polygons is impossible. However, since ROGER is designed to provide real-time analytical capabilities for geospatial problems, it is equipped with high memory nodes. This allows us to implement the all-to-all approach and compare the performance with our distributed algorithm.

We compare the two points in polygon approaches using two metrics: a) *spatial indexing time* which measures the average time it takes each reducer to build the spatial index based on its associated set of polygons and b) the average query time for the reducers to find the closest polygon. The time for pre-processing is excluded from this experiment, since it has to be called only once per polygon datasets and accounts for a very small part of the computation process. We set the number of partitions equal to the number of reducers to get one partition per reducer, hence eliminating the effect of scheduling and shuffling stages in MapReduce and focus on the core algorithm. As Figure 3.7a illustrates, using our point-in-polygon approach we can significantly reduce the spatial indexing time, since less data is required to be read and indexed. However, this reduction is not going to be proportional to the number of partitions. The reason lies into the overlapping nature of our partitions , which requires storing one polygon in multiple partitions. This time for the all-to-all approach remains nearly constant since it has to index all the polygons regardless of the number of reducers. In addition, we observe that the query time decreases using our point-in-polygon approach (Figure 3.7b)). This decrease is not going to be as substantial as the spatial indexing part since the querying complexity is $O(\log n)$ where n is the number of polygons. Therefore, for a large n the speed up ratio will be very small.



(a) Time to build the spatial index.



(b) Average query time for reducers.

Figure 3.7: Comparison of our approximate point-in-polygon algorithm with all-to-all approach.

Optimal Number of Partitions

The decision about the optimal number of partitions and reducers is a trade-off. First, while having more reducers usually leads to higher parallelism (if the available resources allow), it will also add considerable overhead to the Hadoop framework. In addition, choosing large number of partitions reduces the risk of highly dense regions and load imbalance. However, by adding more partitions we also increase the number of duplicate polygons in the boundaries. Table 3.5 shows the total computing time of point-in-polygon operation for different number of partitions and reducers. Since each reducer should have at least one partition to process the upper part of the table is undefined. We observe that for a fix number of partitions, we usually get better performance as we move to larger number of reducers. Based on Table 3.5 having 256 partitions with 128 reducers provides us with the best computing time. Here by increasing the number of partitions, we reduce the time required to build the spatial index, as well as reducing the risk of having skewed and dense regions. However, we observe that by moving to 256 reducers the overhead of reducers overcome the extra parallelism and the overall computing time increases.

Partitions/Reducers	8	16	32	64	128	256
8	317	-	-	-	-	-
16	303	158	-	-	-	-
32	331	236	136	-	-	-
64	290	192	131	117	-	-
128	282	173	139	121	118	-
256	279	183	145	114	106	123

Table 3.5: The point-in-polygon computing time for different combinations of number of partitions and reducers (in seconds)

Measuring Interactivity

We evaluate interactivity of UrbanFLow by measuring the query latency for the two most intense query patterns in our framework:

- *getCluster*: Query a region to get all of the clusters which are located in that region.
- *getSpread*: Query a region to find the spread of users from that region.

The pattern *getSpread* is particularly intense, since it requires two levels of queries to get the users of that region and find out other frequently visited locations of those users. For both patterns, we run queries on all the regions (2025) and measured mean, median and 90% percentiles of query latencies. Table 3.6 shows the result for query latencies in pattern 1 and 2. Using 90% percentile of query latencies we can conclude that the interaction of the users is most likely take less than half a second, even for the most intense queries.

Query Pattern/Latency	Mean	Median	90% Percentile
getCluster	152	103	260
getSpread	247	154	466

Table 3.6: Query latency in UrbanFlow for defined patterns in milliseconds

Our experiments confirm that UrbanFlow can efficiently handle processing large datasets while providing a fast interactive environment for the users to analyze the result.

3.4 Concluding Discussion

In this chapter, we explain our approach for balancing the computational and data loads on reducers in the existence of skew which is very common for geospatial applications. Our approach consider the computational intensity of the problems by looking at challenges arising from both data and computational aspects. We demonstrate the approach using two well-known problems of spatial data aggregation and point-in-polygon, which were used in MovePattern and UrbanFlow applications, respectively.

Our experiments confirmed that our approach can effectively balance the load on reducers by devising a computation plan in the pre-processing step. This plan determines the association of keys to reducers and will enable reducers to process a spatially contiguous block of data that we demonstrated to be advantageous in other spatial operations such as spatial nodes summarization.

In addition, we discussed the importance of interactive scalability which is often overlooked in designing user-driven data-intensive geospatial applications. We evaluated the interactive scalability of MovePattern and UrbanFlow by performing stress test, designed according to the real-world workload. Both experiments confirmed that our frameworks can sustain the sudden increase in users query loads and continue providing fast sub-second responses.

CHAPTER 4

SCALABLE INDEXING OF GEOSPATIAL DATA

4.1 Background

The importance of complex geospatial big data in understanding and addressing fundamental problems in many domain sciences is being increasingly recognized. One of the fundamental capability needed for handling big geospatial data is ability to index them for efficient storage and quick access. Though the problem of indexing has been well studied in literature, the approaches presented are not tailored in the context of large big data [Eldawy and Mokbel, 2015]. This limitation is due to the inability of classic systems to scale to today's massive data streams and the lack of sophisticated spatial support on modern data stores.

One well-studied technique to represent spatial data is Geohash. geohash is a form of Z-order space-filling curve that recursively subdivides a space into smaller regions in a hierarchical fashion interleaving latitude and longitude bits to represent a location's coordinate with a single number (Figure 4.1). Over the past few years, geohash has gained particular attention to index large spatial data [Malensek et al., 2013a, Fox et al., 2013] because it provides the following key advantages (1) an effective mapping from coordinate space into a single key, which can be efficiently stored, updated, retrieved and distributed in modern key-value datasets (compared to more classic techniques such as r-trees [Malensek et al., 2013a]); (2) hierarchical nature that facilitates using fast bit operations to support multiple resolution queries; (3) since geohash is a form of Z-order space-filling curves, locations that are close by on coordinate space have close geohash value, and this property can be used to support fast range-based and nearest neighbor queries; and (4) compared to common index

structures, geohash does not affect the indexing structure in a recursive fashion and has only one "stratum" [Liu et al., 2014]. Therefore it facilitates situations where data has high velocity and/or when the index needs to be concurrently updated.

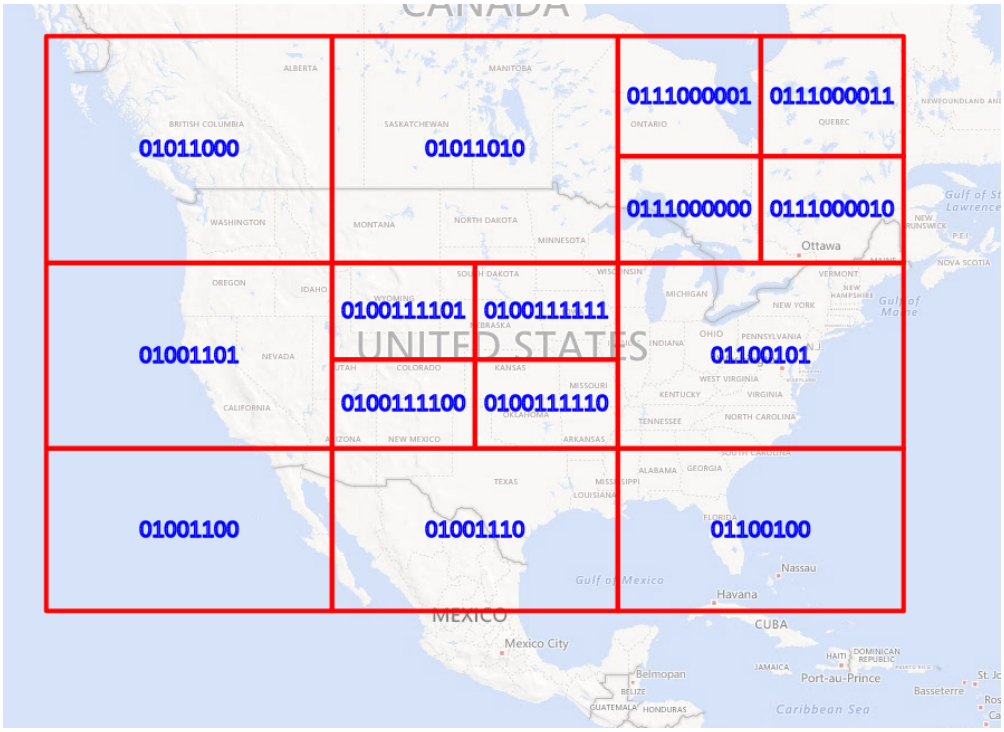


Figure 4.1: Example of using geohashes with different resolutions

While geohash has been used to index point data [Moussalli et al., 2015], there has been little efforts to index other spatial objects using geohash. This is mainly due to the complex process of finding multiple geohashes (often with varying precision) to cover arbitrary spatial objects [Malensek et al., 2013a]. In this chapter, we introduce two new algorithms to index lines and polygons using geohashes. The conducted experiments demonstrates the ability of the algorithms to efficiently index high-precision lines and polygons dataset.

4.2 Geohash-based Indexing of Lines

To index lines we should find all the unique geohashes that intersect with the line (Figure 4.2). This problem can be modeled as a line drawing problem over a digital grid that is

well-known to the computer graphics community.

We use a modified version of a fast ray-tracing algorithm [Amanatides and Woo, 1987] to find all the geohashes. Our algorithm traces the line beginning from the geohash that covers the starting point of the line, iteratively moving towards the geohash that covers the ending point of the line. At each iteration we move to one of the neighboring geohash cells, which is determined based on the line slope ($stepX$ and $stepY$) and the progress made in the previous iteration ($maxX$ and $maxY$). Algorithm 9 illustrates this process. Our modification to the original algorithm also enables us to overlay the geohash grids which have floating point cells' width and height.

Algorithm 9: Algorithm to encode lines using geohashes

input: (x1,y1), (x2,y2), cellWidth, cellHeight, precision
$x, y \leftarrow 0$,$gHash1 \leftarrow GeoHash(x1, y1, precision)$,$gHash2 \leftarrow GeoHash(x2, y2, precision)$;
$[endX, endY] \leftarrow \frac{gHash2.point - gHash1.point}{[cellWidth, cellHeight]}$;
$deltaX \leftarrow \frac{cellWidth}{|x2-x1|}$ and $deltaY \leftarrow \frac{cellHeight}{|y2-y1|}$;
$stepX \leftarrow sign(x2 - x1)$ and $stepY \leftarrow sign(y2 - y1)$;
$fracX \leftarrow x1 - startGeohash.minX$ and $fracY \leftarrow y1 - startGeohash.minY$;
if $stepX > 0$ **then** $maxX \leftarrow deltaX \times (cellWidth - fracX)$;
else $maxX \leftarrow deltaX \times fracX$;
if $stepY > 0$ **then** $maxY \leftarrow deltaY \times (cellHeight - fracY)$;
else $maxY \leftarrow deltaY \times fracY$;
while $xReached$ *or* $yReached$ **do**
  if $maxX < maxY$ **then**
    $maxX \leftarrow maxX + deltaX$;
    $x \leftarrow x + stepX$;
  else
    $maxY \leftarrow maxY + deltaY$;
    $y \leftarrow y + stepY$;
  $Result.add(GeoHash(x1 + x \times cellWidth, y1 + y \times cellHeight, precision))$;
  if $x \rightarrow endX$ **then** $xReached \leftarrow true$;
  if $y \rightarrow endY$ **then** $yReached \leftarrow true$;
return $Result$;

After the set of geohash cells are produced, we merge geohash cells which are child of the same parent using a greedy algorithm. For instance, suppose we have geohash cells with

binary codes 100011, 100010 and 10000. First, 100011 and 100010 will be merged into 10001 and then merged with 10000 to produce 1000. This approach allows us to reduce the number of geohash cells produced by the algorithm, which results in a smaller indexing structure.

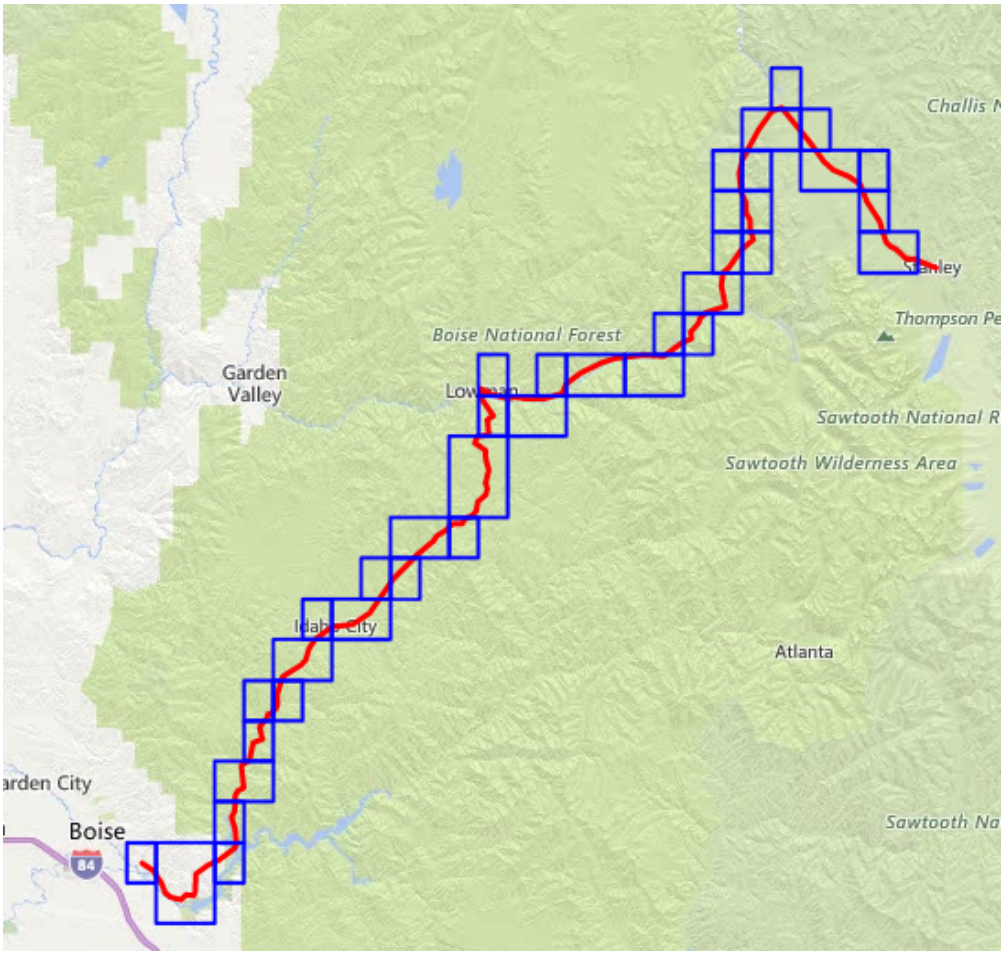


Figure 4.2: Indexing lines using geohashes.

4.3 Geohash-based Indexing of Polygons

Indexing polygons introduces additional challenges since a fixed precision approach [Malensek et al., 2013b], will either lead to too many geohashes for large polygons or very coarse geohash indexes for small polygons. Therefore, an efficient indexing scheme should consider geohashes with various precision (Figure 4.3).

To address this issue, we designed an adaptive algorithm to recursively divides geohashes until it reaches the desired accuracy. Therefore, a polygon will be indexed using geohashes in multiple granularities. Algorithm 10 explains the steps required for finding geohashes that cover a polygon.

Algorithm 10: Algorithm to encode polygons using geohashes

```
function geohash(polygon, maxPrecision)
    coarsestGeo ← find largest geohash that contains polygon ;
    geohashRec(coarsestGeo, polygon, maxPrecision, result) ;
    return result ;
function geohashRec(geohash, polygons, maxPrecision, result)
    if geohash.precision ≥ maxPrecision then
        result.add(geohash) ;
        5 return ;
    if polygon.contains(geohash.boundingBox) then
        result.add(geohash) ;
        return ;
    childs ← geohash.getChilds() ;
    if polygon.intersect(childs[0].boundingBox) then
        geohashRec(childs[0], polygon, maxPrecision, result) ;
    if polygon.intersect(childs[1].boundingBox) then
        geohashRec(childs[1], polygon, maxPrecision, result) ;
```

The performance of this algorithm highly depends on the implementation of *contain* and *intersect* spatial operators, which can be expensive for high precision polygons. The cost of these operations can be reduced using two approaches: (1) probabilistic method (e.g. Monte Carlo methods) as opposed to the exact calculation; and (2) simplifying the polygon to reduce the complexity of these two operations. The drawback of the first method is that we may end up losing some required geohashes for an specific polygon that affects the accuracy and correctness of the indexing process, while the second approach may result in some excess geohashes that needs to be removed in the post-processing step. The choice between these two approaches highly depends on the requirement of the application.

Due to the required accuracy of queries in our platform, we present a solution based on

the simplification approach. For high-precision polygons, we replace the original polygons with their convex hull (using Graham Scan [Graham, 1972]) in the pre-processing step and perform *contain* and *intersect* operations on the replaced simplified polygons. This will result in dramatic reduction in the overall time of querying process (refer to 4.4). The choice of simplification method come with a trade-off between the number of geohash cells which are used to index a polygon and the time complexity of the algorithm. For instance, in most cases by using concave hull we can achieve a more compressed and concise representation of the original polygon. However, concave hull algorithms have much higher time complexity compare to convex hull algorithms.

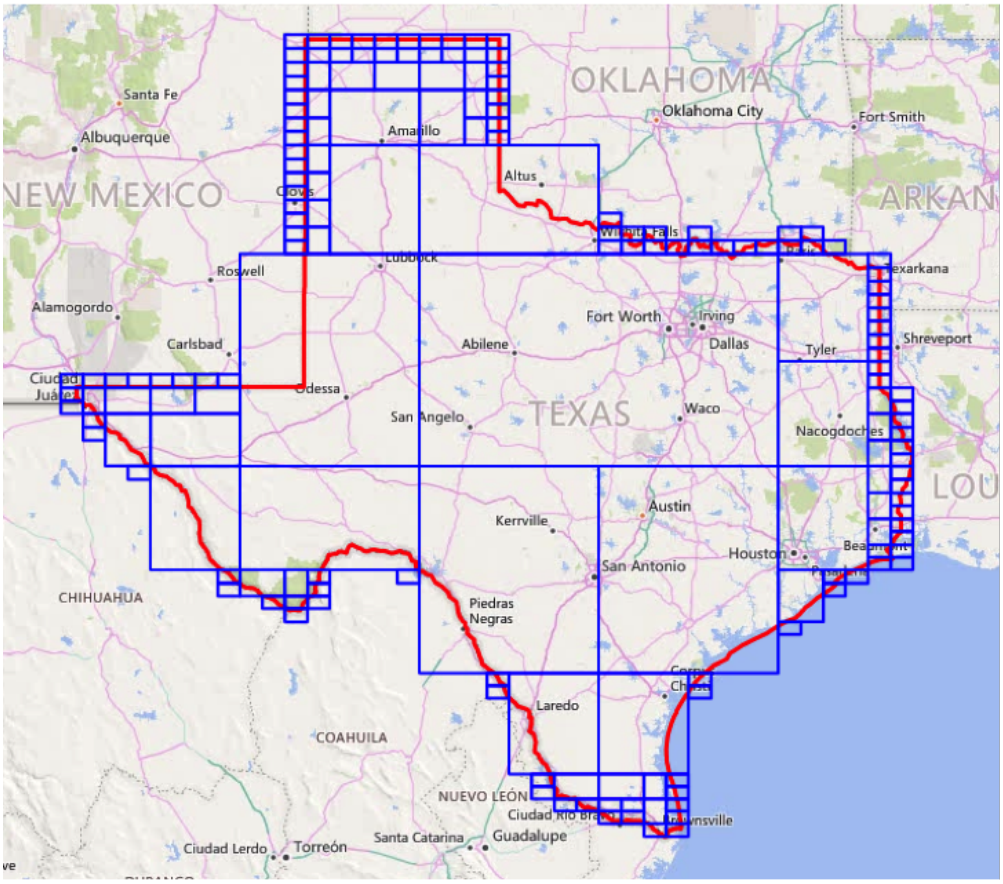


Figure 4.3: Indexing polygons using geohashes.

4.4 Experiments

To evaluate our indexing scheme, we use three datasets representing point, line and polygon objects:

1. **Point Dataset:** USGS dataset for Cities and Towns of the United States, including 38186 cities.

2. **Line Dataset:** List of US Major Road Bases which is gathered by `MapCruzin.com` in collaboration with Federal Communications Commissions. This dataset includes 47013 different roads (each of the roads may include multiple line segments).

3. **Polygon Dataset:** The counties for the entire United States, collected by United States Census Bureau (500k resolution level). This dataset has 3232 counties (each may include multiple polygons).

The result of experiments are presented in Table 4.1. For the polygon indexing we compare the result of the original algorithm, to the modified algorithm (simplifying polygons). To focus on the performance of the indexing approach we exclude the time for reading the original data and writing back the result and only measure the indexing time. In addition, in all the experiment the maximum geohash precision is set to 20 bits. As the result illustrates,

Experiment/Time	Mean	Median	90% Percentile	Geohashes
Point Data	616.5 ns	347 ns	936 ns	38186
Line Data	0.001 ms	0.001 ms	0.003 ms	45363
Polygon Data (Original)	1.90 ms	0.16 ms	2.09 ms	31698
Polygon Data (Modified)	0.29 ms	0.09 ms	0.35 ms	32496

Table 4.1: The performance of our geohashing algorithms on 3 different datasets.

our algorithms can index complex spatial objects in few milliseconds. In addition, we observe that using the modified algorithm, the indexing time decreases dramatically with slight increase in the number of geohashes which is expected.

4.5 Concluding Discussions

In this chapter, we presented two scalable algorithms to index arbitrary geospatial data using geohashes. Geohash is a form Z-order space filling curve that has been increasingly used to index large geospatial data. The main advantages of geohash-based indexing include a) mapping 2D space into a single key which can be efficiently stored and retrieved and b) faster update process, compared to traditional approaches such R-tree, in multi-tenant applications where the index is frequently updated. While geohash has been mainly adapted to index point data, we present two novel algorithms to efficiently index line and polygon data using geohash. We use these algorithms to represent and index geospatial objects in Chapters 5 and 6.

CHAPTER 5

DYNAMIC WORKLOAD-AWARE DATA PARTITIONING

5.1 Background

Distributed frameworks for geospatial data is built on two core processes [Aly et al., 2015]: 1) partitioning data and distributing the resultant chunks among multiple nodes; and 2) storing and indexing of these chunk. This scheme is also known as using global and local indexes to distribute the data [Eldawy and Mokbel, 2015]. However, frameworks can vary significantly when it comes to the details of partitioning and indexing, which is driven by their use cases.

Most of the existing distributed geospatial management systems focus on read-heavy batch-processing scenarios, where data comes in bulk with low velocity [Eldawy, 2014, Kini and Emanuele, 2014, Yu et al., 2015, Aji et al., 2013a]. Such systems can effectively take advantage of frameworks such as Apache Hadoop [Shvachko et al., 2010a] to distribute data among multiple nodes, with a local index formed in each node. On the other hand, some frameworks focus on dynamic data [Nishimura et al., 2013, Malensek et al., 2016, Fox et al., 2013], where the data comes with high velocity and should be indexed and available to query immediately. One main drawbacks of the existing distributed frameworks for geospatial data is that they only support static partitioning of data. In static partitioning, the data is partitioned once, either by a fixed scheme [Fox et al., 2013, Nishimura et al., 2013, Malensek et al., 2016] or based on a data sample [Eldawy, 2014, Yu et al., 2015]. In this case partitions cannot get updated without re-partitioning the whole data from scratch.

However, over the course of an application, the workload may experience a skew due

to existence of hotspots, time-varying skew, load spikes and "honey-stick effect"(sudden increase popularity of an application) [Taft et al., 2014]. In such cases, static partitioning of data cannot address the dynamically changing nature of the data. In addition, previous research highlighted the value of incorporating query workload into the partitioning approach [Curino et al., 2010, Aly et al., 2015] and avoid presuming any assumption on the workload. *Workload-aware partitioning* algorithms provide the solution to unpredictable shifts in the workload by dynamically finding an optimal partitioning given a set of constraints.

In this chapter, we present an approaches for partitioning data based on a high-volume and multi-tenant workload. This approach is optimized for the use cases where the ingestion rate of the data is high and the workload may experience significant shifts on its spatial distribution overtime.

5.2 Adaptive Workload-aware Partitioning

Adaptive workload-aware partitioning refers to a family of approaches that 1) take workload into account in defining the partitions and 2) can dynamically change overtime to reflect the shifts in the workload. There have been extensive studies on designing workload-aware partitioning for Online Analytical Processing (OLAP) databases [Ghosh et al., 2016, Jindal and Dittrich, 2011]. In these cases, the algorithm focuses on increasing parallelism by avoiding co-locating popular data blocks in the same partition. In addition, OLAP-based algorithms work with very coarse-grained data, which require a careful storage utilization approach [Serafini et al., 2016, Jindal and Dittrich, 2011]. However, our approach is focused on real-time spatial data which is related to Online Transaction Processing (OLTP).

One main distinguishing factor of OLTP-based algorithms is the higher cost of distributing a transaction over multiple partitions (inter-partition access), which is known to have a dramatic effect on the overall performance of the framework. For instance, one study showed that even with 10% of the transactions distributed, the throughput may be cut in half [Pavlo

et al., 2012]. The algorithms for workload-aware partitioning for OLTP databases can be differentiated over multiple aspects as follows:

- *coarse-grained* [Quamar et al., 2013] vs. *fine-grained* [Serafini et al., 2016, Taft et al., 2014]: While fine-grained approaches can provide more balanced partitions, gathering usage information for such fine-grained units is quite expensive. One solution to address this issue is to use a two-step approach [Serafini et al., 2016, Taft et al., 2014]. In the first step, some coarse-level statistics, such as cpu/memory usage or partition size, are gathered for each partitions. The first step is continued until a constraint is violated, e.g. the load imbalance ratio of the partitions exceeds a threshold δ. Then the collection module moves into the second step, where specific fine-grained statistics are gathered for a short period of time. The data that is gathered in the second step is used to define the optimal partitions.

- *Optimization approach:* One common optimization approach is hypergraph partitioning [Curino et al., 2010, Quamar et al., 2013], where a graph is formed based on the co-access pattern of entities and a *k-way partition* divides the space into k small components minimizing an objective function. This approach is particularly popular since the problem of graph partitioning is well-studied and efficient implementations of graph partitioning approaches exist. However, this approach does not consider the existing partition and generates a new partition from scratch, which is inefficient due to the cost of moving data and distributing transactions. Another approach is to incrementally tweak existing partitions to derive new partitions [Serafini et al., 2016, Taft et al., 2014, Aly et al., 2015]. Due to the high complexity of the problem, such approaches usually employ greedy methods to optimize the workload.

- *Cost function:* The definition of the cost function varies significantly among different algorithms. Quamar et al [Quamar et al., 2013] minimize the inter-partition edges weights while enforcing an upper limit on the load imbalance ratio. Other approaches

also limit load imbalance ratio, while minimizing different metrics such as the data transfer cost [Taft et al., 2014] or inter-partition access count [Serafini et al., 2016].

While previous researchers proposed effective method to provide adaptive workload-aware partitioning [Taft et al., 2014, Serafini et al., 2016], they are not tailored for spatial data. Spatial data partitioning should guarantee spatial contiguity [Wu and Murray, 2008] of the partitions to minimize transaction distribution, which has proven to be a critical factor in the achieved throughput [Pavlo et al., 2012]. This is particularly challenging since checking contiguity of a partitioning approach can be expensive [Liu et al., 2016]. In addition we cannot run the partitioning algorithm in the individual record level, since a) calculating the optimal partitioning in such fine-grained scale is time-consuming, due to high cost of spatial operations such as intersect; and b) due to 2d nature of the data if the atomic level is defined as spatial objects we may get overlapping partitions. Therefore we need to overlay a fine-grained grid on top of the original data [Aly et al., 2015].

Previous approaches for adaptive workload-aware partitioning of spatial data either focus on a single centralized server model [Tzoumas et al., 2009, Achakeev et al., 2012] or focus on splitting some existing partitions as opposed to modify the current partitions [Aly et al., 2015].

Our approach takes advantage of a distributed architecture and models the partitioning problem as an optimization problem with spatial contiguity as one of the constraints. We formulate the optimization problem as finding the partitioning scheme P' from partition P by:

$$\begin{aligned} &\underset{P'}{\text{minimize}} \quad F_{P'} \\ &\text{subject to} \qquad \forall p \in P' : \quad p \quad is \quad spatially \quad contigious. \end{aligned} \tag{5.1}$$

We define F_P as a metric to quantify the partitioning desirability which we will discuss shortly in Section 5.3.

The spatial contiguity constraint make this optimization problem particularly challeng-

ing, since contiguity check is an expensive operation [Liu et al., 2016]. Therefore, we use an incremental approach that migrate spatial units among partitions in a way that spatial contiguity is preserved. Therefore, starting from spatially contiguous partitions, the contiguity condition will hold in the rest of procedure.

5.3 Partitioning Fitness Evaluation

One of the main challenges in designing an effective partitioning scheme is to define a metric that can capture the desirability of a partition. Particularly, since the focus of our partitioning algorithm is to provide even loads to the nodes in a cluster, we are pursuing metrics that quantify load imbalance. However, different metrics measure load imbalance in various ways and provide different insights on how the load is distributed among different nodes.

In order to choose the best load imbalance metric, we need to consider how this metric is being used in our approach. For instance, after new statistics regarding workload are captured from the nodes, the current partitioning scheme is evaluated to determine whether re-partitioning is required. In this particular scenario, we evaluate whether the Maximum Load Imbalance Ratio surpasses a threshold. This ratio, which is calculated as a percentage and defined as [DeRose et al., 2007]:

$$\lambda = \frac{L_{max} - \overline{L}}{L_{max}} \times \frac{n}{n-1} \tag{5.2}$$

Here L_{max} is the maximum and $\overline{L}$ is the average load among n partitions. Therefore, the partition change process will be triggered if λ exceeds θ, which is the maximum allowed load imbalance.

However, during the evolutionary re-partitioning algorithm we also continuously evaluate the partitioning scheme to guide the optimization process. In this case, we need a metric that is sensitive toward variation in the load distribution. The Maximum Load Imbalance

Ratio does not provide such information since it is only considering the maximum load. For instance in Figure 5.1 both partitioning has the same λ value. However, the right partitioning is considered a better solution in the evolutionary re-partitioning algorithm since it shows less load variation among different partitions.

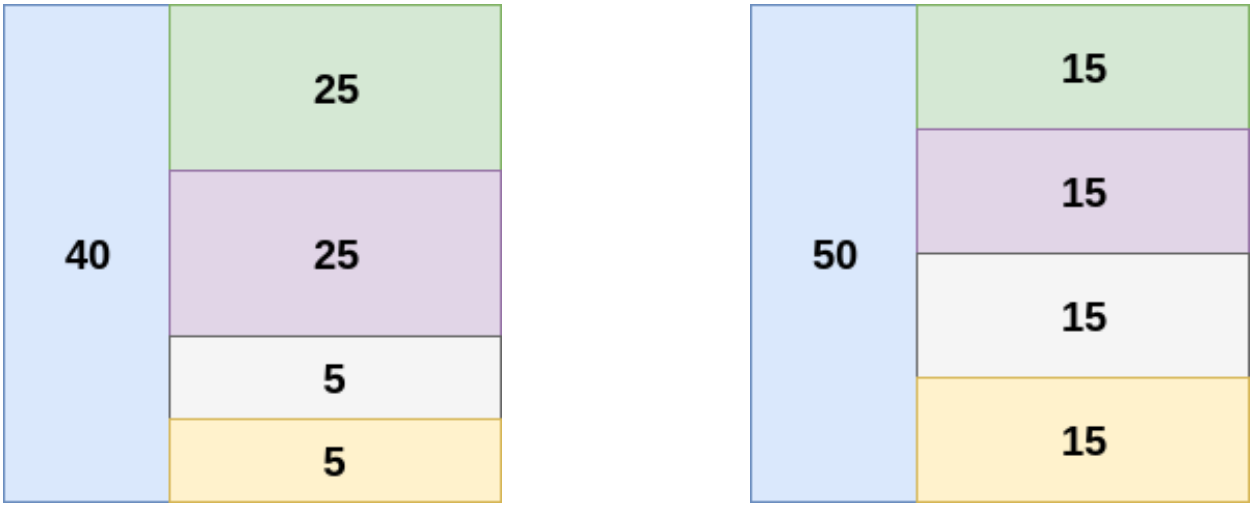


Figure 5.1: Two different partitioning with the same λ value that exhibit different load variation.

To address this issue, we employ a new load imbalance metric as the coefficient of variation [Arzuaga and Kaeli, 2010] of partitioning P:

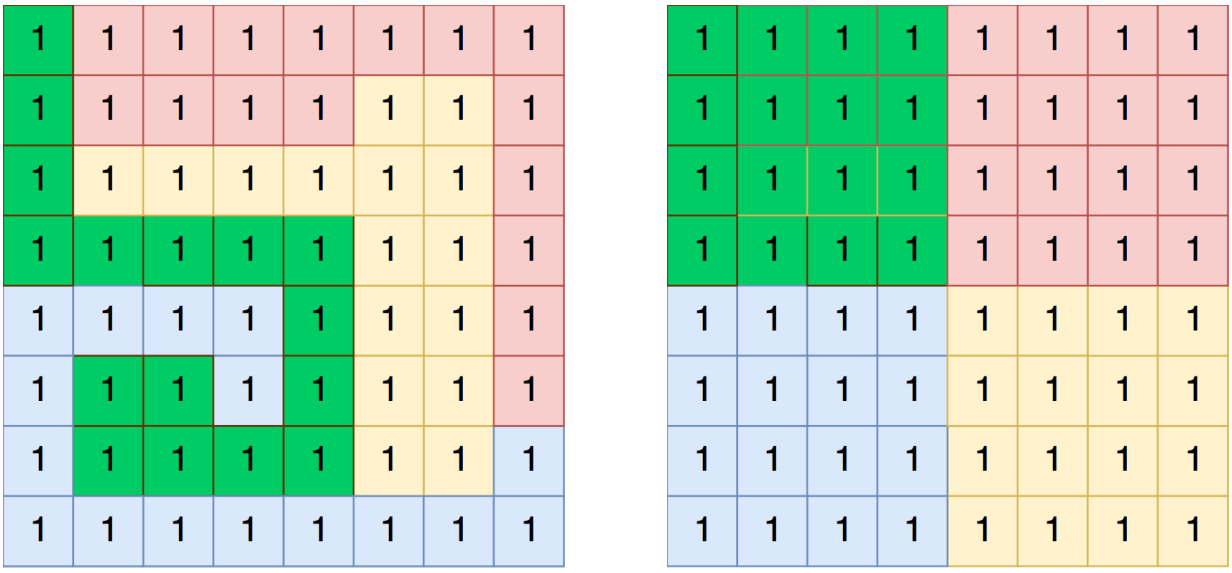
$$C_P = \frac{\sigma_P}{\mu_P} \tag{5.3}$$

Here σ_P is the standard deviation of the load among partitions and μ_P is the mean. This metric does not solely depends on maximum load and considers the overall load distribution among different partitions. Therefore, it can be effectively used in the optimization process.

5.3.1 Spatial Compactness

Spatial compactness is a property related to the shape of a spatial object. While there are many different ways to evaluate compactness [Kai and Boa, 2010], the presence of it

in spatial partitioning is encouraged. For instance, consider Figures 5.2a and 5.2b as two partitioning schemes. It is clear that partitions in Figure 5.2a are less desirable because the probability of a request to be distributed in multiple partitions is significantly higher (due to increased border perimeter).



(a) Low compactness

(b) High compactness

Figure 5.2: Two partitioning where each partition has the same intensity of 16 with varying compactness.

In our approach, we quantify the spatial compactness using *Non-linear Neighbor Method* which measures the number of neighboring cells that shares the same partition assignment [Kai and Boa, 2010]:

$$compactness = \sum_{p=1}^{P} \sum_{i=1}^{N} \sum_{j=1}^{M} A_{ijp} x_{ijp} \tag{5.4}$$

Where

$$A_{ijp} = x_{i-1jp} + x_{i+1jp} + x_{ij-1p} + x_{ij+1p} + x_{i-1j-1p} + x_{i+1j+1p} + x_{i-1j+1p} + x_{i+1j-1p} \quad \forall p = 1, ..., P, i = 1, ..., N, j = 1, ..., M \tag{5.5}$$

Here $x_{ijp} = 1$ if cell x_{ij} has the partition assignment of p and otherwise it is 0.

The *compactness* is value in the $[0, 1)$ range, where larger value exhibits higher compact-

ness which is the more desirable outcome. Therefore we define the combined $fitness(P)$ of a partitioning P as:

$$fitness(P) = w_1 \times C_P + w_2 \times compactness(P) \tag{5.6}$$

The value of C_P can be potentially more than 1, however for such extreme cases with set the value to 1, which indicates very high deviation. We choose w_1 and w_2 in a way that $w_1 + w_2 = 1$. In Section 5.7 we present experiments on the effect of different w_1 and w_2 values on the final load imbalance.

5.4 Approach Layout

Our proposed algorithm is an evolutionary algorithm to solve the optimization problem, explained in Equation 5.1. The algorithm is based on [Liu et al., 2016], that incrementally modify the partitions to find one which satisfy the constraints. Checking for spatial contiguity at each step is expensive, therefore we use an algorithm that given an initially spatially contiguous partitioning scheme, and preserves the contiguity at each step.

The approach can be summarized as:

1. Initialize k new partitioning solutions given the current workload and form the initial population or use an existing partitioning of data.

2. Repeat this process for $maxIterations$ times:

   - Select a member from the population using one of the parent selection methods.
   - For the selected solution, run the random spatial migration procedure (explained in Section 5.6) to derive a new partitioning solution p.
   - If there is a partition p' in the population where $fitness(p') < fitness(p)$ replace it with p.

5.4.1 Parent Selection Methods

One crucial element of any evolutionary algorithm is the parent selection method which is used at each step to select the member(s) of population which the evolutionary operation is applied to. These methods are the gateway of entering solutions from one round of an evolutionary algorithm to the next. In our approach, we provide three methods for parent selection [Beasley et al., 1993]:

- **Random Selection:** In this approach, an individual is randomly selected from the population.
- **Tournament:** This method forms a tournament among K randomly selected individuals and introduce the winner of the tournament (individuals with the highest score) as the parent.
- **Rank Selection:** Ranks individual in the population according to their fitness value. Next, it randomly select a solution from the population where higher rank solutions are more likely to be selected than the lower rank solutions.

In Section 5.7 we observe how the parent selection methods can affect the performance of the partitioning approach.

5.5 Initialization

In the initialization step, given a query and data distribution, we design a partitioning scheme that preserves spatial contiguity. We assume that the number of desired partitions are $|P|$. We start by randomly selecting $|P|$ geohashes where the probability of selecting cell g is proportional to $load_g$. The selected geohash cells form our seed units for each partition. Then we iteratively expand each partition by adding non-partitioned neighbor geohashes at each step (Figure 5.3).

Algorithm 11: Initialization process to create new partitioning scheme.

```
function initialize(P, workload)
    seeds ← randomly select P geohash units, where p(geohash) ∝ load(geohash)
     according to the workload
    partitions = ∅
    visited ← Set()
    for seed ∈ seeds do
        partition ← Partition()
        partition ← partition ∪ seed
        visited[seed] = true
    while |visited| < workload.geohashes.size do
        partitions ← sort − ascending(partitions) ▷ grow more sparse partitions first
        for partition ∈ partitions do
            border ← partition.borderUnits
            grow ← ∅
            for g ∈ border do
                grow ← grow ∪ neighbors(g)
            for g ∈ grow do
                if g ∉ visited then
                    partition ← partition ∪ g
                    visited[g] = true
    return (partition)
```

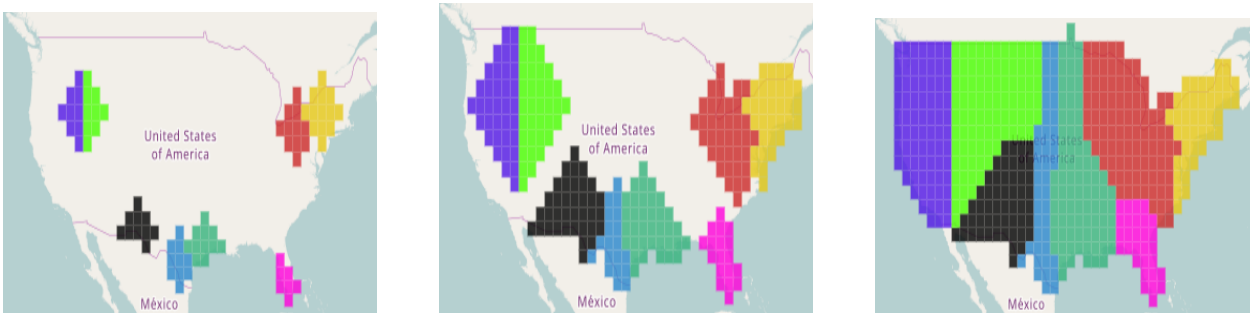


Figure 5.3: Partitions initialization procedure for 8 partitions.

The process is continued till all the geohashes are partitioned (Algorithm 11).

However the initialization approach may lead to form holes in the partitions (Figure 5.4). There are two strategies to resolve this issue:

1. Select seeds on the boundary units of the study region. In this case, a partition cannot be completely surrounded by another partition.

2. Detect if initialization algorithm result in hole (Refer to Algorithm 12 for hole detection) and re-run the algorithm till we achieve a hole-free partitioning.

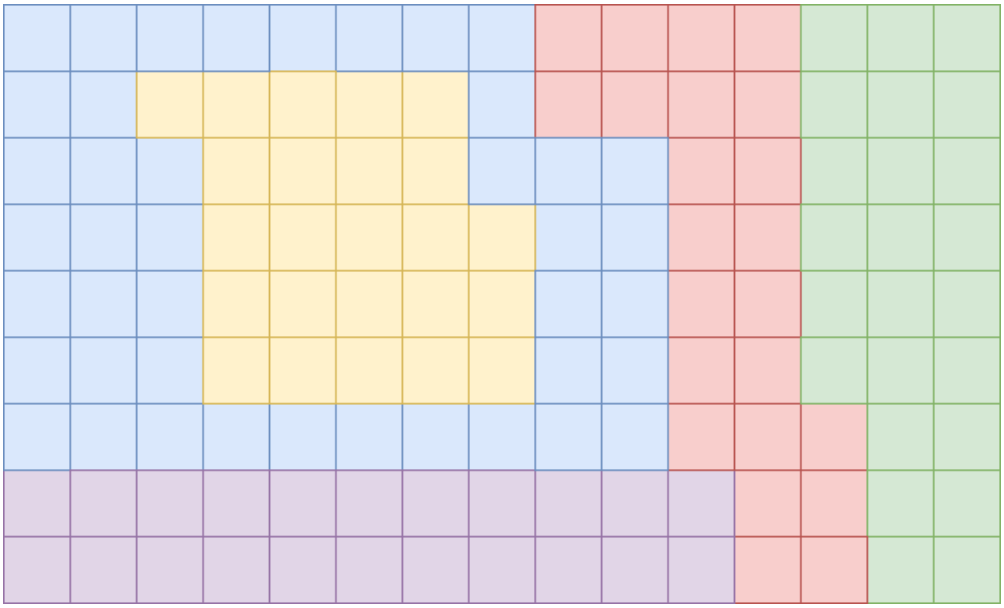


Figure 5.4: The initialization algorithm may form holes in the partitions.

5.6 Spatial Migration Procedure

The goal of spatial migration is to move spatial units (geohashes) among partitions to get more balanced partitions. Contrary to previous work that use static partitioning algorithms such as k-d tree and grid partitioning [Eldawy et al., 2015a, Aji et al., 2013b, Fox et al., 2013], our algorithm does not produce the partitions from scratch and incrementally balance the partitions by moving small set of spatial units among partitions.

The spatial migration procedure (Algorithm 13) is based on the spatial mutation procedure defined in [Liu et al., 2016]. In our fine-grained approach, a set of spatial units in partition A will be selected as the candidates for migration in each iteration and they will move from partition A to B if the spatial contiguity constraint holds.

We avoid expensive contiguity check by designing the migration procedure that preserves spatial contiguity at all time. This is achieved by a) starting from a spatially contiguous

Algorithm 12: Hole detection algorithm.

```
function isHoleFree(partitions)
    hasHole ← true
    for partition ∈ partitions do
        if hasHole(partition) then
            return true
    return false
function hasHole(partition)
    neighborGeohashes ← ∅
    for g ∈ partition.borderUnits do
        neigbors ← neighbor(g)
        if size(neighbors) < 5 then
            return false                    ▷ geohash is in the region boundaries
        neighborGeohashes ← neighborGeohashes ∪ neighbors ;
    neighborPartitions ← ∅
    for g ∈ neighborGeohashes do
        neighborPartition ← getPartition(g)
        if neighborPartition = partition then
            neighborPartitions ← neighborPartitions ∪ neighborPartition
        if size(neighborPartitions) > 1 then
            return false                    ▷ surrounded by more than one regions
    return true
```

initial solution (explained in 5.5) and b) performing spatial operations that preserve contiguity. To guarantee the second condition, we change partition of unit g from partition p_1 to p_2 if g is located in the border of p_1 and p_2 (Figure 5.5). This process will provide spatially contiguous solutions.

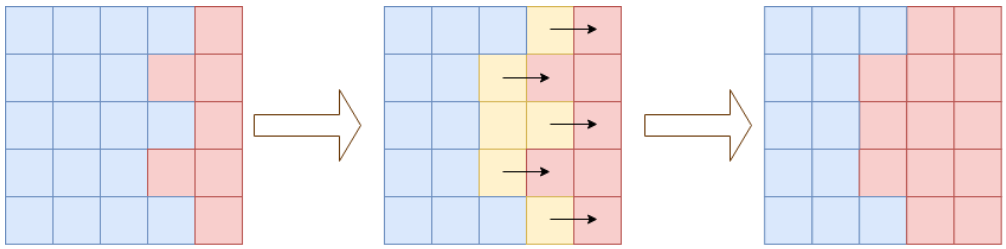


Figure 5.5: Example of load migration to decrease load imbalance ratio.

While we guarantee spatial contiguity, there are situations where using this approach we may break a contiguous partition into two disjoint partitions which is not allowed (Figure

Algorithm 13: Spatial Migration Procedure

```
function migrate(partitions)
    src ← random partition from partitions (weighted by partitions' loads) ;
    dst ← {p|L_p < L_q ∀q ∈ neighbors(src)} ;
    if load(src) >= load(dst) then
        shift(srcPartition, dstPartition) ;
function shift(srcPartition, dstPartition)
    seeds ← randomly select k units in srcPartition bordering dstPartition ;
    candidates ← seeds ;
    updateUnits ← ∅ ;
    for iteration ∈ {1, .., mutationsPerMigration} do
        if {src \ candidates} is underloaded OR {dst ∪ candidates} is overloaded
         then
            break ;
        updateUnits ← updateUnits ∪ candidates ;
        cNeighbors ← neighbors(candidates) ;
        candidates ← ∅ ;
        for g ∈ cNeighbors do
            if partition(g) ≠ src AND g ∉ updateUnits then
                candidates ← candidates ∪ g ;
    dstPartition ← dstPartition ∪ updateUnits ;
    srcPartition ← srcPartition \ updateUnits ;
```

5.6). We avoid this situation by making sure that every partition shift will keep the boundary units of affected partitions within a closed path (Algorithm 14).

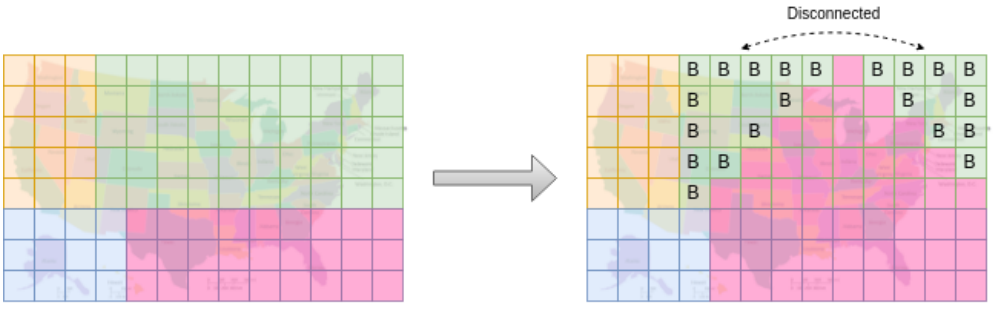


Figure 5.6: An example of a partition that is split as result of Algorithm 13.

Algorithm 14: Check if the migration procedure has caused the partition to be split.

```
function split(partition)
    borders ← partition.getBorders() ;
    current ← Stack() ;
    current.add(borders.iterator().next()) ;
    while current ≠ ∅ do
        geohash ← current.pop() ;
        borders.remove(geohash) ;
        for c ∈ neighbors(geohash) do
            if c ∈ borders then
                current.push(c) ;
    if borders = ∅ then
        return false ;
    else
        return true ;
```

5.7 Experiments

5.7.1 Experiments Setup

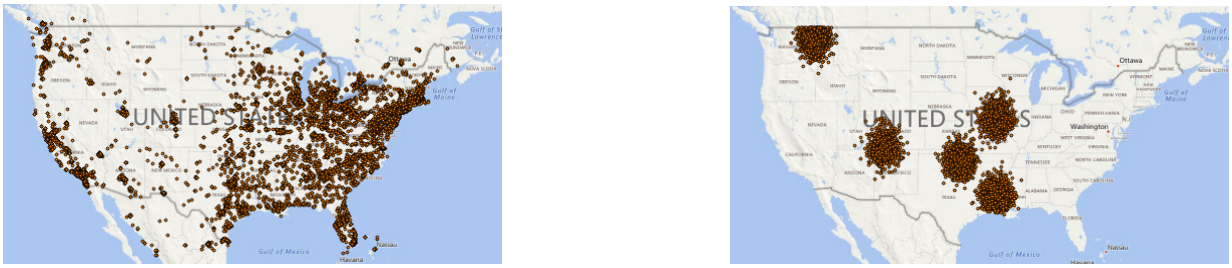
We evaluate our evolutionary partitioning approach by comparing it to the static partitioning approach and adaptive k-d tree partitioning algorithm. To evaluate our approach we form a hybrid workload that includes real geo-tagged Twitter data for the year of 2014 (Figure 5.7a) and a series of randomly formed hotspots (Figure 5.7b) to simulate sudden shifts in the distribution of data. The seeds to generate hotspots are randomly selected from Twitter data where regions with more data intensity are more likely to have a point represented in the seeds. Therefore, the final dataset is a combination of actual Twitter data with some intense regions getting more presentations in each batch of the data. Since spatially focused and recent queries are more common, the random selection gives exponentially increasing probability to smaller distances and shorter time periods. The procedure to generate the test workloads is explained in Procedure 15.

Algorithm 15: The procedure to generate experimental workloads.

Procedure `GenerateWorkload(`*tweets*`)`

for $record_i \in tweets$ **do**
  With probability p:
    Add $(record_i)$ to workload
  With probability $1 - p$:
    Add $formQuery(record_i)$ to workload
end
for $i \in \{1, ..., k\}$ **do**
  $\mu \leftarrow$ random point in United States bounds
  $\mathcal{N}_i \leftarrow Gaussian2D(\mu, \Sigma)$
  Add H random records from $\mathcal{N}_i$ to *hotspots*
end
for $record_i \in hotspots$ **do**
  With probability p:
    Add $(record_i)$ to workload
  With probability $1 - p$:
    Add $formQuery(record_i)$ to workload
end

The $formQuery(record_i)$ forms a query around the input point $(latitude_i, longitude_i)$ as $record_i = (latitude_i, longitude_i, time_i, value_i)$ where $latitude \in (latitude_i \pm d)$,$longitude \in (longitude_i \pm d)$ and $time \in (time_i \pm t)$. Here d is selected randomly from $D = \{0.008333, 0.016666, 0.033332, 0.066664, 0.133328\}$ in degrees (roughly translates to 1km,2km,...,16km). In addition, t is randomly selected from $T = \{1, 2, 4, 8, 16\}$ in days. Since spatially focused and recent queries are more common, the random selection gives exponentially increasing probability to smaller distances and shorter time periods.



(a) Real geo-tagged Twitter

(b) Random hotspot with 5 seeds

Figure 5.7: Workload spatial distribution for the experiments.

The workload is formed in 12 batches, one batch for each month, where each batch includes 50 millions requests. The requests are divided equally between ingestion and query requests (50% write ratio) which is considered a significantly write-heavy workload. We set k as the number of hotspots to 10, H as the size of random workload equal to the *tweets* data.

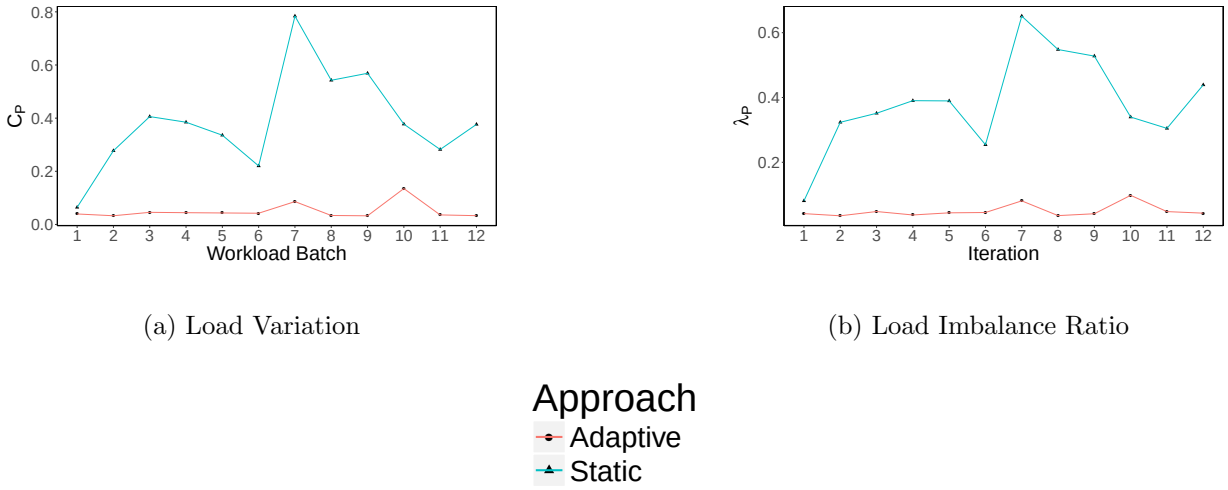
For each batch, once approximately 10% of the workload is observed, we trigger the partitioning change process to evaluate the existing partitioning scheme for the observed workload. The statistics in each node is gathered based on overlaid geohash grid with the precision of 20 bits which is roughly equals to 156km $\times$ 156km spatial units.

We evaluate the partitioning algorithm on three aspects. First, we demonstrate the effect of using an adaptive partitioning versus an static fixed partitioning. Second, we study how number of iterations, mutations per migration, the parent selection method and the weights arrangement of the objective function would affect our evolutionary partitioning algorithm. Finally, we compare the effectiveness of our evolutionary partitioning algorithm with k-d tree partitioning. In all the experiments we set $k = population - size = 50$. All the experiments were conducted five times and the median values are reported as the final metric. For the experiments that include measurements in the level of iteration, we report the median of the targeted metric among all batches. In addition, we define a new metric which is used in multiple experiments. The partition change measures the weighted percentage of spatial units (geohashes) that change their partition from arrangement P_1 to P_2. Specifically, $\Delta(P_1, P_2) = \Sigma_{g \in P_1} \delta_{P_1(g) P_2(g)} load_g / \Sigma_{g \in P_1} load_g$.

5.7.2 Adaptive vs. Static Partitioning

The goal of the first experiment is demonstrating the value of adaptive partitioning versus static partitioning. For the static partitioning we use a sample of the data to generate a partitioning arrangement which will stay constant during the course of the experiment. On the other hand, the adaptive partitioning gets updated on every workload batch based on

the evolutionary algorithm. Figure 5.8 shows how using adaptive partitioning substantially decrease load variation and the maximum load imbalance ratio.



(a) Load Variation

(b) Load Imbalance Ratio

Figure 5.8: Static vs. adaptive partitioning.

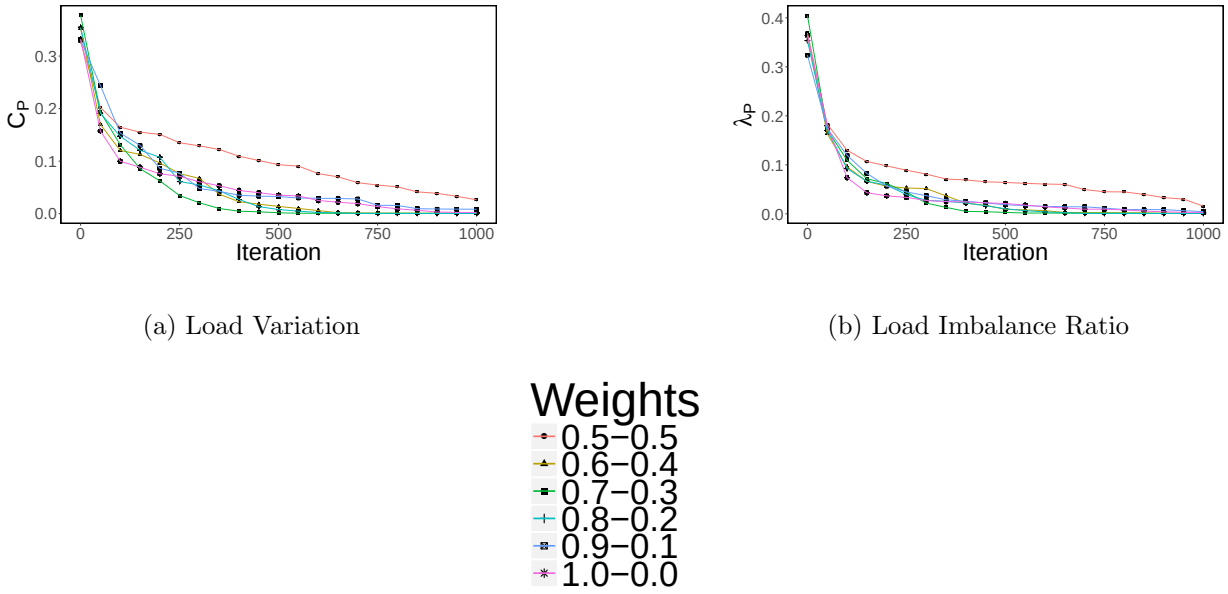
5.7.3 Evolutionary Algorithm Tuning

Fitness Function

The first tuning experiment examines the effect of choosing w_1 and w_2 in defining the fitness function for a partitioning (Equation 5.6). We examined six different combinations of weights (Table 5.1). While most of the weight combinations demonstrated a reasonable convergence of optimization process to a low final load imbalance (Figure 5.9), $\{0.6, 0.4\}$ shows the lowest overall norm for all batches of data (Table 5.1). This shows that while load imbalance is still the dominant factor in defining our fitness function, it should also have a significant trace from compactness.

W_1	W_2	$\|C_P\|$	$\|\lambda_P\|$
0.5	0.5	0.3354575	0.3379257
0.6	0.4	0.04267425	0.06839813
0.7	0.3	0.1170813	0.09567112
0.8	0.2	0.106588	0.0979354
0.9	0.1	0.1825416	0.2175683
1.0	0.0	0.1385816	0.1029528

Table 5.1: The performance of our Geohashing algorithms on 3 different datasets.



(a) Load Variation

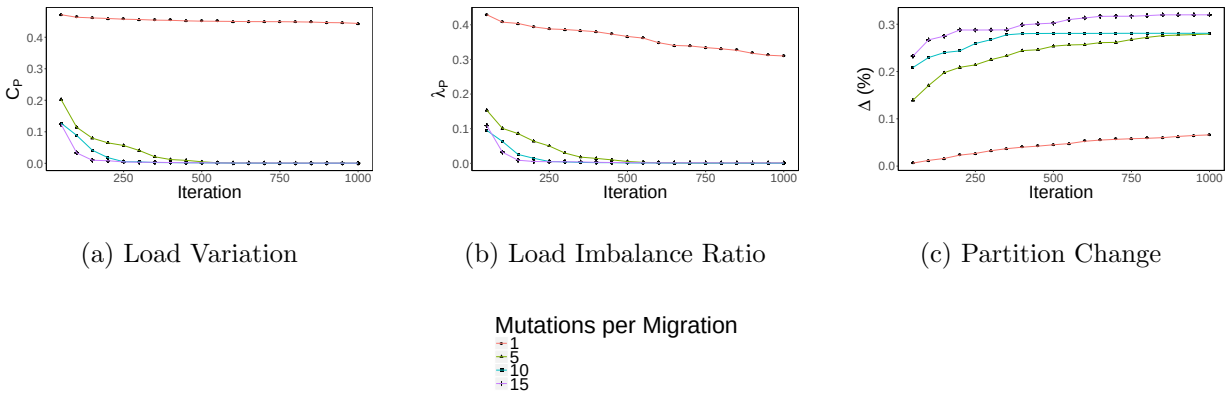
(b) Load Imbalance Ratio

Figure 5.9: The effect of weights for the fitness function on the performance of the partitioning approach for each iteration.

Mutations Per Migration

Our evolutionary partitioning algorithm incrementally modify the partitions arrangement by migrating candidate spatial units between partitions in each iteration. The purpose of our next experiment is determining the effect of number of iterations and number of mutations per migration (that reflects the size of migration candidate) in balancing loads among different partitions. Figure 5.10 illustrates load variation and load imbalance ratio for different iterations and multiple mutations per migration setting. The figure shows that small choice for mutations per migration (1) is not effective since it is migrating only one

spatial unit in each iteration. On the other hand, we observed that using larger values for mutations per migration (e.g. 10 and 15) may lead to slight increase in Δ since there will be more changes made in each iteration. Overall, all the values larger than 1 does not show any significant improvements over each other.



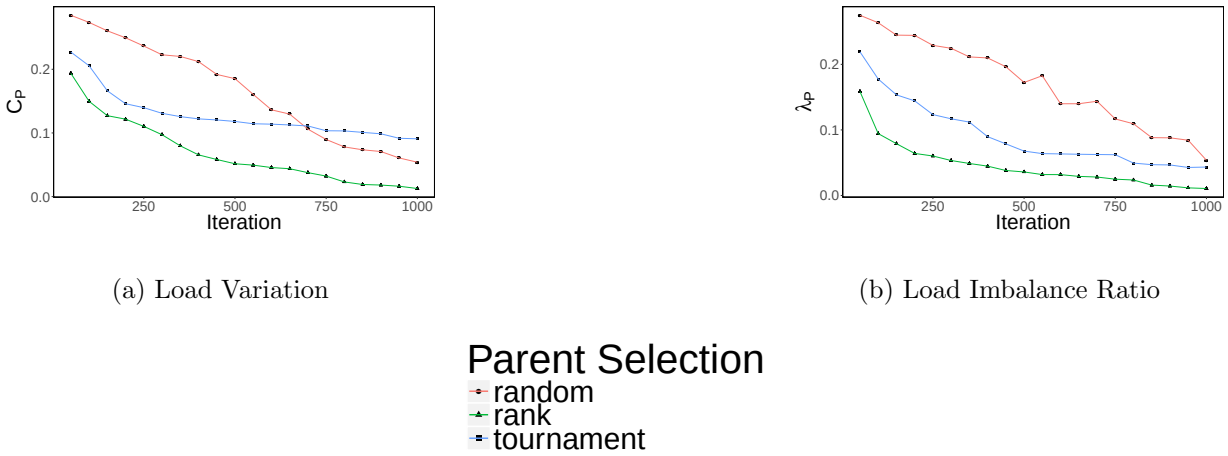
(a) Load Variation (b) Load Imbalance Ratio (c) Partition Change

Figure 5.10: Performance of the evolutionary partitioning based on number of iterations and mutations per migration.

Parent Selection Method

As we explained in the layout of the evolutionary partitioning algorithm, we can use multiple methods to select a partitioning arrangement from the population. This is a crucial step in evolutionary algorithms in general, known as parent selection. In this experiment, we evaluate multiple well-known parent selection methods including a) *random selection* which randomly select a solution from the population; b) *tournament selection* which randomly select a set of parents from the population and select the best solution among those candidates as the result and c) *rank selection* that ranks individual in the population according to their fitness value. Next, it randomly select a solution from the population where higher rank solutions are more likely to be selected than the lower rank solutions. Figure 5.11 shows that rank selection method outperforms both random and tournament selection methods. The gap is particularly evident after some iterations where fitness values of partitions are close

to each other and using pure fitness proportional methods would not give enough advantage to better solutions.



(a) Load Variation

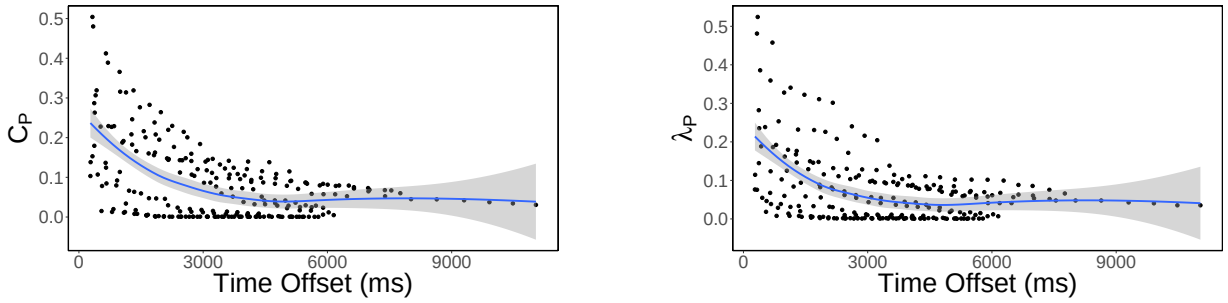
(b) Load Imbalance Ratio

Figure 5.11: Comparison of parent selection methods in evolutionary partitioning performance.

Re-partitioning Time

In the last tuning experiment, we measure the running time of the re-partitioning algorithm (explained in Section 5.6). In Figure 5.12 we plot the offset time (in ms) against C_P and λ_P. The offset time at any moment is calculated as difference of current time and when the re-partitioning was started. As expected in evolutionary optimization methods, the figures illustrate that we can reach a reasonable level of optimization very quickly. However, as we go along the optimization process rate decreases and it would be more expensive to further maximize the fitness function. Therefore, specific applications that require more relaxed constraints on the tolerated load imbalance ratio, may terminate the migration process much sooner than the maximum number of iterations.

In addition, Figure 5.13 shows the running time for each iteration. We observe that offset time has a linear relationship to the number of iteration, which conveys that in each iteration we expect similar computational intensity, determined by $mutationsPerMigrations$.



(a) Load Variation

(b) Load Imbalance Ratio

Figure 5.12: Offset time (in ms) vs. performance of the migration process.

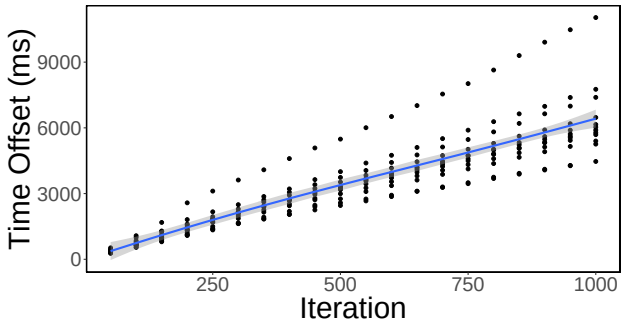
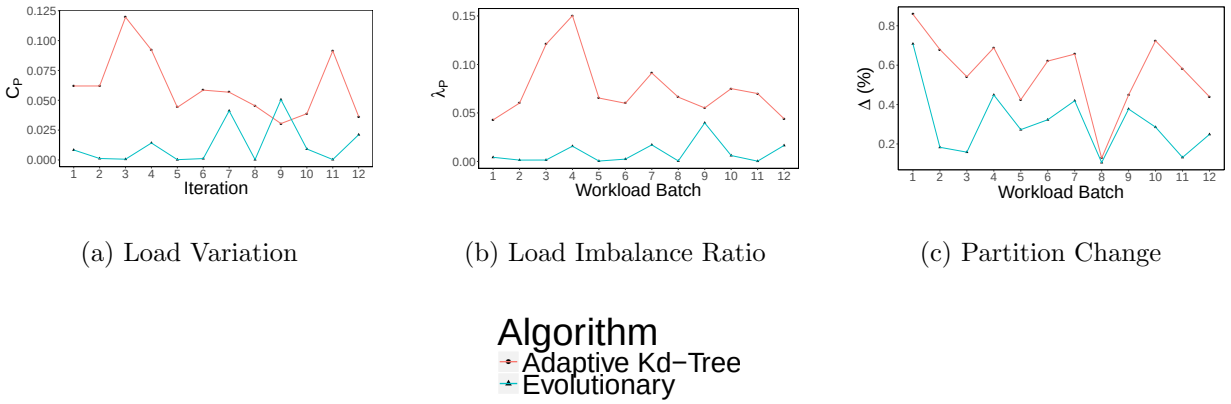


Figure 5.13: Offset time (in ms) for each iteration of the migration process.

5.7.4 Evolutionary Algorithm vs. Adaptive K-d Tree

The this experiment compares our evolutionary partitioning algorithm to adaptive k-d tree partitioning which adapt the partitioning based on the observed sample of data, similar to our evolutionary approach. We compare these approaches in terms of load variation, load imbalance ratio and partition change. Figure 5.14 shows that evolutionary partitioning method outperforms adaptive k-d tree in terms of load variation and load imbalance ratio since it operates on a finer grain by optimizing the assignment of each individual spatial unit. In addition, we observe that evolutionary approach demonstrates lower Δ since it does not generate the partitions from scratch and modify the existing partitions.



(a) Load Variation (b) Load Imbalance Ratio (c) Partition Change

Figure 5.14: Comparison of different partitioning approach in terms of load variation, load imbalance ratio and changes in the partitions.

5.8 Concluding Discussions

In this section, we discuss our novel adaptive partitioning algorithm for geospatial data. Our algorithm addresses the shortcoming of previous static partitioning approaches which were unable to react to changes in the workload. Our approach models the partitioning problem as a fine-grained spatial optimization problem, which we solve using an spatially-tuned evolutionary algorithm. The evolutionary procedure gradually modify the existing partitioning of data to regain the balance among partitions.

We evaluated our approach using a simulated dataset that includes both normal workload and hotspots. The experiments demonstrated that our approach outperforms static partitioning algorithms in terms of final load imbalance. In addition, we successfully showed that our approach outperforms adaptive Kd-Tree approach in terms of both load imbalance and cost of changing the partitions.

In the next chapter we discuss GeoBalance, a framework that implements our adaptive partitioning approach on top of a cloud-based cyberGIS infrastructure.

CHAPTER 6

GEOBALANCE: WORKLOAD-AWARE FRAMEWORK TO MANAGE REAL-TIME SPATIOTEMPORAL DATA

6.1 Background

Over the past two decades cloud-based systems have stabilized themselves as the most popular choice for data-driven applications. In particular, Geospatial community has embraced the cloud architecture to host data-intensive applications [Yang et al., 2011, Wang et al., 2015]. The strength of cloud-based systems comes from three main capabilities [Cooper et al., 2010]:

- **Scale-out** to support large dataset and high request rate.
- **Elasticity** to easily add more resources to improve the performance.
- **High availability** to provide reliable services.

To fully take advantage of cloud computing, the system should also adapt a highly decentralized architecture [Gupta et al., 2016]. This principle strongly influences the optimal design for data-driven frameworks. While some previous research [Eldawy and Mokbel, 2015] has claimed that built-in data management system are more efficient than designing a separate and independent data management layer, they did not take into consideration requirements that arise from running data-driven applications in the cloud space. First, the layered design is crucial to address the growing complexity of data-driven applications [Gupta et al., 2016]. In addition, tightly integrated database can limit the horizontal scalability due to ineffective resource utilization [Hasselbring, 2016]. Therefore, we propose an independent data management middleware that can be easily linked to different data storage technologies.

One phenomena which has been tightly coupled with cloud computing over the past few years, is the design principle of having lightweight and highly focused services, called microservices. Microservices are introduced to increase the cohesion of the systems and decrease the coupling by providing set of small services that communicate through a fast network-protocol. The main advantages of using microservices are [Newman, 2015]:

- Technology heterogeneity: The ability to choose different technologies for different components that best fit the task, as oppose to one-size-fit-all approach.
- Resilience: The system can continue functioning, even if a component fails (no failure cascading).
- Scaling: As oppose to monolithic system, where all the components should scale together, we can choose which part of the system to scale.
- Organization alignment: Provides higher efficiency by allocating tasks to small teams.
- Composability: The opportunity to reuse functionalities for different scenarios.

In this chapter, we introduce GeoBalance, a workload-aware framework which is designed for managing real-time spatiotemporal data sources. This framework takes advantage of the adaptive workload-aware partitioning algorithm that we discuss in Chapter 5 and provide a scalable and reliable environment for write-intensive geospatial applications. In the rest of this chapter, we will discuss the architecture of GeoBalance and discuss strategies for modeling workload, adding/removing nodes on demand (elasticity), rolling partitioning migration and data replication. In addition, we present multiple experiments to demonstrate the scalability of GeoBalance approach, as well as the advantage of using our adaptive partitioning algorithm in the framework.

6.2 Architecture

GeoBalance takes advantage of a highly decentralized architecture to follow the separation of concerns (SoC) design principle, which is crucial to address the growing complexity of data-driven applications [Gupta et al., 2016]. GeoBalance consists of loosely coupled microservies that communicate together via the network and provide high availability and scalability that cannot be achieved using monolithic designs. Since each of the services are operating independently from each other, failure does not cascade in the system. Furthermore, each component can scale separately to address the application's needs. Tightly integrated data management systems can limit the horizontal scalability due to ineffective resource utilization [Hasselbring, 2016].

The architecture of GeoBalance has a specific focus on real-time spatiotemporal data and designed to handle highly skewed spatiotemporal data that can experience significant workload shifts over time. It is important to emphasize that GeoBalance focuses on real-time data and stores data for a limited time window. After the time window has passed, the data is expired and may be moved to another data management system which is designed to answer queries for historical data. The separation of batch(historical) and streaming(real-time) processing of data is known as Lambda architecture [Marz and Warren, 2015] which is a common practice in systems that require handling a massive amount of data. In this architecture, the real-time portion is focused on fast ingestion of data as opposed to the historical portion which provides analytical capabilities for very large data and is designed for read-heavy workloads.

There are four main components in the GeoBalance framework (Figure 6.1): (i) data nodes; (ii) broker nodes; (iii) configuration store; and (iv) coordinator nodes; which are implemented as microservices that can be deployed on multiple nodes in a distributed fashion. The roles and responsibilities of each are detailed below.

1. **Data nodes** are each responsible for indexing and storing a portion of the data. Data

nodes encode spatial objects using geohash-based algorithms and store them using a technique which is described in Section 6.2.1. As we explained earlier, the data nodes will expire the data after time window t and schedule them to be removed. Finally, the data nodes periodically contact the configuration store to announce their observed workload and usage statistics.

2. **Broker nodes** act as routers for incoming users requests. Using up-to-date partitioning information synchronized from the configuration store, the broker nodes determine which data node(s) should be contacted to insert/retrieve the data. In our current implementation of GeoBalance, broker nodes focus on spatial and temporal dimensions of the data to determine the related partitions.

3. **Configuration store** is distributed data storage that can efficiently share configuration data among multiple microservices. The store, implemented using Apache Zookeeper [Hunt et al., 2010], synchronizes the data among multiple services and guarantees eventual consistency.

4. **Coordinator nodes** periodically contact the configuration store to capture usage statistics and workload of each node. They use this information to decide whether a) scale-in/out is required and/or b) partitions should change to maintain load balance tolerance. In the case that a change is required, the coordinator nodes initiate the transition process.

The data nodes in GeoBalance are implemented using Vert.x, a Java-based event-driven framework [1] and store data using RocksDB [2]. RocksDB is an embedded database which was developed by Facebook and optimized for low-latency storage, such as SSDs or memory.

RocksDB uses Log Structured Merge (LSM) tree [O'Neil et al., 1996] (Section 6.2.1) which is optimized for write-heavy workloads where the index is frequently updated [Kim et al.,

[1] `https://vertx.io`
[2] `https://rocksdb.org`

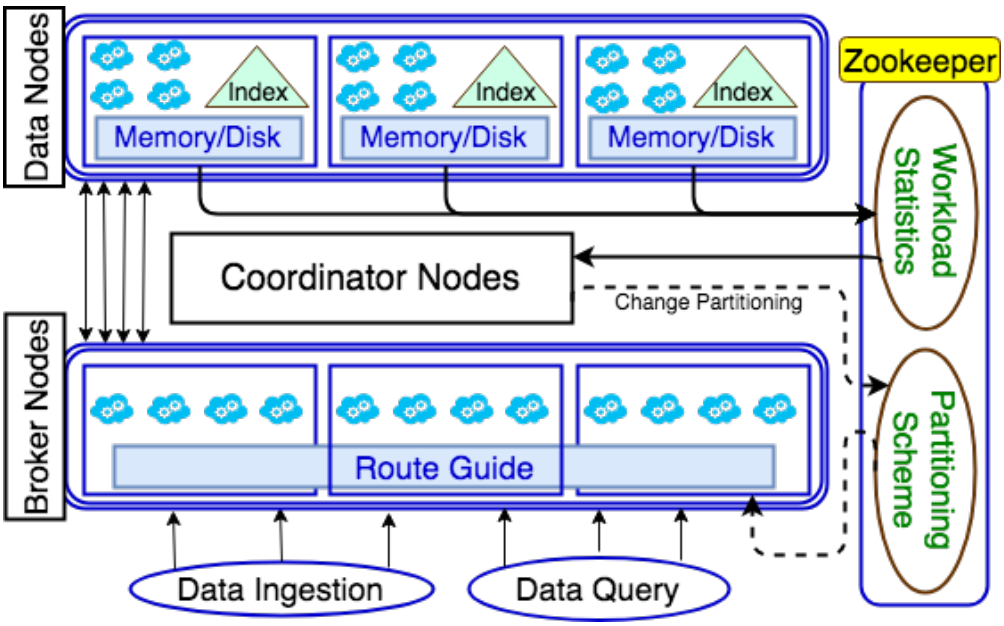


Figure 6.1: GeoBalance micorservice-based architecture.

2017]. In addition, we configure the RocksDB instances to use memory and tune them for in-memory prefix-search workloads which we take advantage of to perform spatial queries.

The core contribution of GeoBalance is the workload-aware spatiotemporal partitioning approach (Chapter 5). In this approach, coordinator nodes gather statistics regarding workload intensity (Section 6.3). Based on these statistics, an objective function is defined to reflect the desirability of current partitions in terms of load imbalance. Based on a pre-defined set of constraints, such as the maximum load-imbalance threshold, coordinator nodes may trigger a partition change in case one or more constraints are violated. In the partition change process, we start from the existing partition and incrementally modify the partitions to achieve more balanced partitions.

In addition, GeoBalance is able to provide elasticity by enabling scale-out and scale-in capabilities for horizontal scaling. If multiple nodes are overloaded or if the partitioning algorithm cannot provide a feasible answer for the existing workload, a node will be added to the cluster and the partition change process is triggered again to find a feasible solution in the modified cluster. Similarly, in case one or more nodes are underloaded, a node will

be removed from the cluster and the partitions will be updated accordingly. The details of elasticity strategies is explained in Section 6.4.

GeoBalance provide reliable service to the users by a) employing a rolling migration approach for changes in the partitioning scheme to continue serving queries while re-partitioning is happening (Section 6.5) and b) provide data replication capabilities to handle possible failures in the data nodes (Section 6.6).

6.2.1 Log structured Merge Trees

Log Structured Merge Trees (LSM-trees) [O'Neil et al., 1996] is a popular approach to store data for write-intensive application, which is used in systems such as LevelDB [3], Apache Cassandra [Lakshman and Malik, 2010] and RocksDB. The power of LSM-tree comes from its multi-level structure to store data which can support fast ingestion of data.

LSM-trees stores the incoming data into an in-memory write buffer, called mem-table. Mem-tables are usually implemented using sorted tree structures such as Red-black tree [Guibas and Sedgewick, 1978]. In addition to mem-table, the data is inserted into a Write-Ahead Log (WAL) which is used for recovery purposes when failure happens in the system. When the size of mem-tables reach a limit (defined based on the application), three operations happen:

1. A new mem-table and WAL is created to service the future write requests.
2. The old mem-table is flushed to a "Sorted Sequence Table" (SST) file.
3. After the old mem-table content is flushed, WAL and mem-table of the flushed data are discarded.

Over the time, the number of SST files may increase, which in turn will lead to performance degradation. This is mainly due to a) possible existence of multiple versions of the same

[3] `http://leveldb.org`

record and b) data for the same key exists in multiple SST files [Kleppmann, 2017]. To address this issue, LSM-tree use a process called *compaction* to merge smaller SST files into larger ones with non-overlapping key ranges. This process also ensures that out-dated values for the keys are discarded. Compaction is the most crucial step in LSM-trees and ensures that the performance of the LSM-tree does not degrade significantly as the new data arrives.

Due to sorted nature of SST files, the compaction process can be done quite efficiently by employing methods such as Merge Sort. In addition, all the writes in the compaction process are performed using sequential I/O and in bulk, which is the main reason for lower write amplification of LSM-tree compared to other approaches such as B-tree [Dong et al., 2017]. Finally, the compaction process can be run in the background to avoid disrupting the ingestion and retrieval of data.

The compaction process is usually implemented in multiple levels (called leveled compaction) to give higher priority to more recent data and avoid having many SST files in a single level. In this scheme, if size of all SSTs in level L surpass a threshold, then one or multiple SSTs of level L are merged with their overlapping SSTs in level $L + 1$.

Despite the fast and low-latency processing of write requests in LSM-tree, the read requests can be quite expensive. For read requests, we first need to check the memtable to see if it contains the data. If not, we move to the most recent segments which is stored on the disk. Subsequently we may need to check multiple levels of SSTs to find the key. To speed up the process, most implementation of LSM-trees store a) in-memory index of multiple offsets for the SSTs to speed up the look-up process and b) maintain a Bloom Filter [Bloom, 1970] of the keys, which enables fast checking for keys which do not exist in the ingested data.

The overall process of LSM-tree for read/write requests is demonstrated in Figure 6.2.

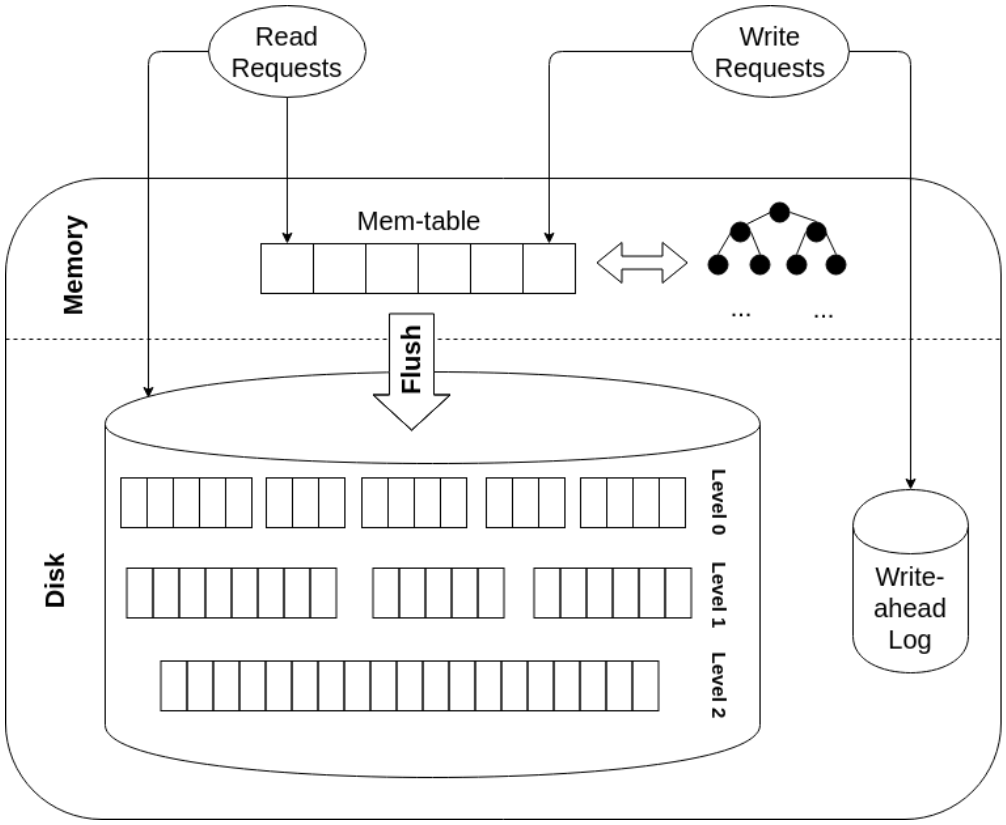


Figure 6.2: Log Structured Merge Tree.

6.3 Modeling Workload

As we explained in Section 6.2, coordinator nodes periodically contact the Zookeeper store to get a record of the most recent workload data, which were collected from data nodes. In addition, the data nodes may call coordinator nodes to trigger a partition check if the resource usage of a node (disk, memory, etc.) surpasses a certain threshold. In each update, two entities are being tracked: data and query distribution as the workload distribution and resource usage of each node in terms of percentage ($resource_{used}/resource_{max}$).

The workload distribution is reported based on geohashes with a fixed precision (grid representation) defined by the user. The algorithms explained in Chapter 5 are used to form the grid-based representation of spatial objects. The grid representation enables efficient workload monitoring and the fast comparison of two partitioning solutions. In addition, the grid representation provides faster spatial operations, such as border detection and finding

neighboring regions. The size of the grid dictates the trade-off between more fine-grained optimization and time and resources spent for the collection of metrics. The grid cells are the most basic unit for the optimization problem, therefore a partitioning scheme is defined by assigning each geohash cell to a partition.

It is common for a data node to experience temporary load spikes [Malensek et al., 2016], which are sudden increases in the intensity of data and/or query associated with a particular data node that lasts for a short period of time. To avoid changing the distribution for such short-term changes, we extend the definition of workload distribution to the past k reported metrics. Therefore at time t, the workload of node i is defined as:

$$Workload'_i(t) = \sum_{j=1}^{k} workload_i(t-j) \times w_j \tag{6.1}$$

We use the same formula to calculate the smoothed resource usage over time. Here $w = [w_1, ..., w_k]$ is the window kernel, which is defined based on *exponential decay* to put more emphasis on more recent collected statistics. Using this strategy, given λ as the *exponential decay constant*, the value of w_j is calculated as:

$$w_j = w_0 e^{-\lambda j} \tag{6.2}$$

In our case, we have the additional constraints of $\sum_{j=1}^{k} w_j = 1$. Therefore:

$$\sum_{j=1}^{k} w_0 e^{-\lambda j} = 1 \rightarrow w_0 = \frac{e^{-\lambda} - 1}{e^{-\lambda k} - 1} \times e^{\lambda} \tag{6.3}$$

The λ parameter determines how much relative effect is given to the previous workloads. Larger value of λ results in giving more weight to more recent workloads observations. Figure 6.3 demonstrates the effect of λ in selecting weights for 8 partitions.

Once the smoothed workload is calculated, L_i, the load of partition i, is defined as the sum of geohash cells' loads that exist in that partition.

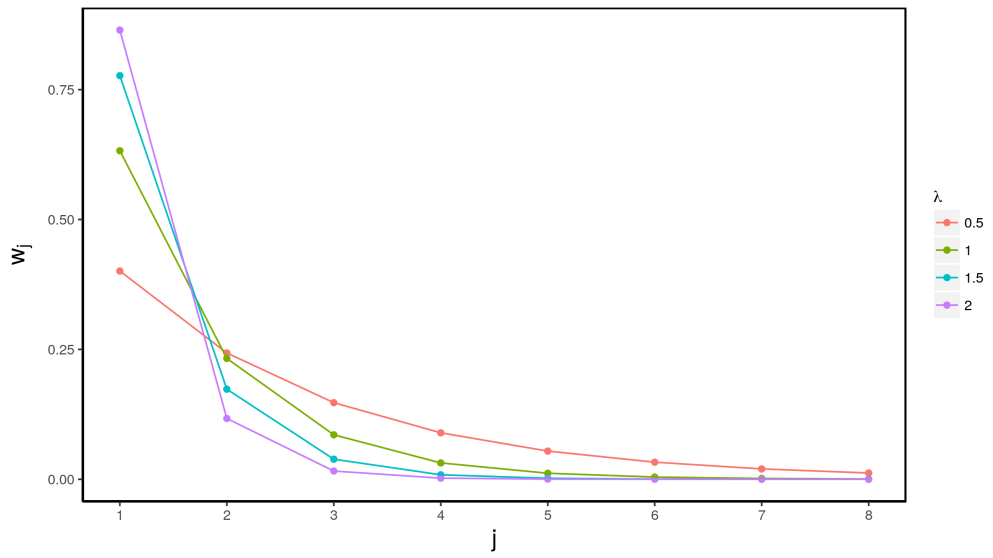


Figure 6.3: The effect of λ in determining workload weights based on exponential decay formula.

6.4 Elasticity Strategies

GeoBalance employs strategies to add/remove nodes on demand to provide better resource utilization according to the observed workload. In this section, we explain the condition and process of adding/removing nodes to the framework.

A node will be added if there exists a node i which is overloaded or if $Resource_i > \gamma \times Max - Resource_i$ (γ is the resource threshold). Similarly, a node will be removed if it is marked as underloaded. Assuming θ as the maximum load imbalance allowed in the framework, we call partition i overloaded if $L_i > (1+\theta)\overline{L}$ and underloaded if $L_i < (1-\theta)\overline{L}$ [Serafini et al., 2016].

If a new node is added to the framework, we should increase the number of partitions to accommodate the new resource. This process is different than the re-partitioning process which is explained in Chapter 5, since the re-partitioning algorithm modify the current existing partitions. When a new node is added to the framework, the partition with the highest

load is selected as the candidate and the two most dense spatial units in that partition are chosen as the seed point. Then an expansion procedure (similar to the algorithm explained in Section 5.5) is run to expand these two seed points into partitions. The only additional constraint is that the expansion procedure is not allowed to go beyond the borders of the original partition.

In addition, when a node is removed from the framework we should assign its partitions to other nodes. However, this process should maintain the spatial contiguity of the partitions. In our approach, for each partition that needs to be reassigned, we merge the partition with the neighbor partitions that has the least load.

It is crucial to note that when the partitioning reassignment is done as the result of adding/removing new nodes, we may need to run the re-partitioning algorithm to balance the loads of partitions.

6.5 Rolling Partitioning Migration

GeoBalance is designed to be a highly available framework; therefore, the framework should continue to serve users requests during the partition transition period. We define the *partition transition period* as the period of time where there are still data in the framework which had been distributed according to the old partitions. This period starts when the new partitioning approach is deployed and ends when all the data from previous partitioning approach are marked as expired. During this period, insert operations will be made according to the most up-to-date partitioning scheme. However, based on the temporal range of the query, we may need to look for the data according to the older partitioning scheme as well. For instance, if new partitions are employed at 2:00PM, a query that requests data for 1:55PM to 2:05PM needs to explore both old and new partitioning schemes to determine nodes where matching data might exist.

To maintain the availability of system during the partition transition period, we use a

"rolling-based" technique, which gradually migrates partitions by maintaining multiple partitioning schemes of the data to hide the partitioning details from users and provide continuous access to the system. Figure 6.4 demonstrates an example of the rolling migration technique for two partitioning schemes. While in theory this approach can host multiple partitioning schemes concurrently, in practice the number of concurrent partitions is limited due to the quick expiration policies of real-time systems. The rolling-based technique provides a temporal element into our partitioning approach by distributing queries on older and recent data separately.

One final consideration related to the transition period is the potential inconsistency of broker nodes regarding the up-to-date partitions. Since Zookeeper provides eventual consistency, for a very brief period of time, nodes might not be in sync about the most up-to-date partitions. To guarantee the correctness of queries which are related to this time window, we take into account a *grace period* for the partitions to get deployed into all nodes. Therefore, if the transition period ends at time t, we will not expire the old partition until $t + \alpha$, where α is the grace period to make sure there is no data loss in the query result.

6.6 Data Replication

GeoBalance provides the optional functionality of data replication as a failure recovery mechanism when one or multiple node become unavailable. In that case, broker nodes will use the replica partitions until the failed node is up and running again. This is a crucial capability to ensure the reliability of the framework, since node failures is quite common in commodity clusters.

In order to assign partitions and their replicas to nodes it is required that: 1) partition i should be assigned to exactly k_i unique nodes and 2) the load imbalance among different nodes should be minimized. We model the replication assignment problem using the $m \times n$ matrix B where:

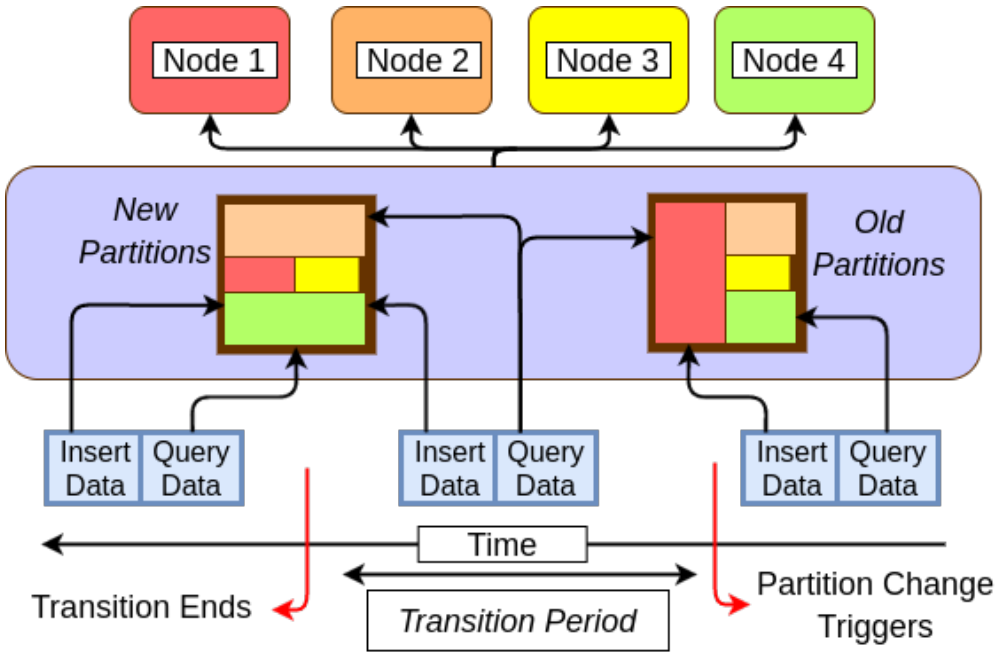


Figure 6.4: Using rolling-based migration to provide continuous access to the framework in the partition transition period.

$$B_{ij} = \begin{cases} 1 & \text{if } partition_i \in node_j \\ 0 & otherwise \end{cases} \tag{6.4}$$

In Section 5.3, we introduced two metrics for measuring load imbalance. For replication, we use Equation 5.2 since we are interested in minimizing the maximum load that a node in the cluster sustain. Therefore, the problem can be formulated as:

$$\begin{aligned} \underset{B}{\text{minimize}} \quad & L_{max} = max(\{\sum_{i=1}^{m} B_{ij} \times L_i : j \in \{1, ..., n\}\} \\ \text{subject to} \quad & \sum_{j=1}^{n} B_{ij} = k_i \quad \forall i \in \{1, ..., m\} \end{aligned} \tag{6.5}$$

We then convert this problem to a mixed integer programming problem by introducing the

dummy variable γ:

$$\begin{aligned} \underset{B}{\text{minimize}} \quad & \gamma \\ \text{subject to} \quad & \sum_{j=1}^{n} B_{ij} = k_i \quad \forall i \in \{1, ..., m\}, \\ & \sum_{i=1}^{m} B_{ij} \times L_i \leq \gamma \quad \forall j \in \{1, ..., n\} \end{aligned} \tag{6.6}$$

This assignment can be easily solved using common mixed integer programming methods.

Figure 6.5 demonstrates an example of assigning 8 partitions to 4 nodes using fixed replication factor of 2 (each partition will be assigned to 2 nodes).

Partitions	1	2	3	4	5	6	7	8
Load	8	12	5	10	18	9	7	10
Replication Factor	2	2	2	2	2	2	2	2

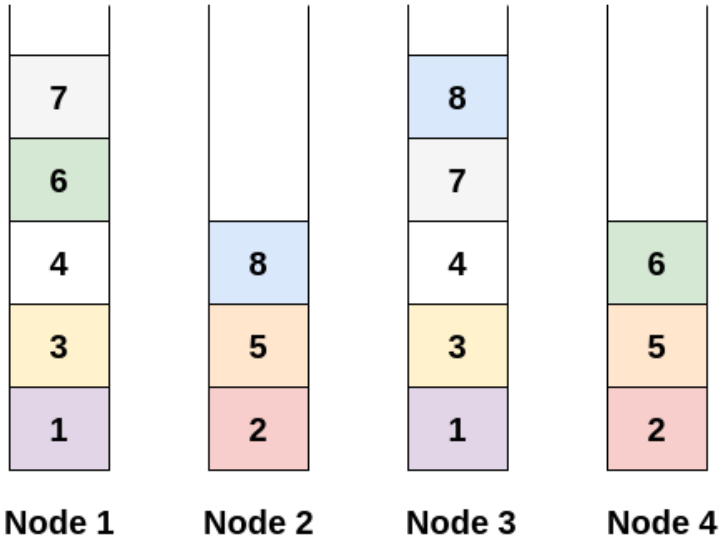


Figure 6.5: Data replication example with fixed replication factor.

However, based on the requirement of the application, we can set different replication

factor. This is particularly useful when we want to balance out query loads of different partitions without re-partitioning the data. Figure 6.6 demonstrates an example where the replication factor varies among different partitions.

Partitions	1	2	3	4	5	6	7	8
Load	8	12	5	10	18	9	7	10
Replication Factor	2	3	2	3	3	2	2	3

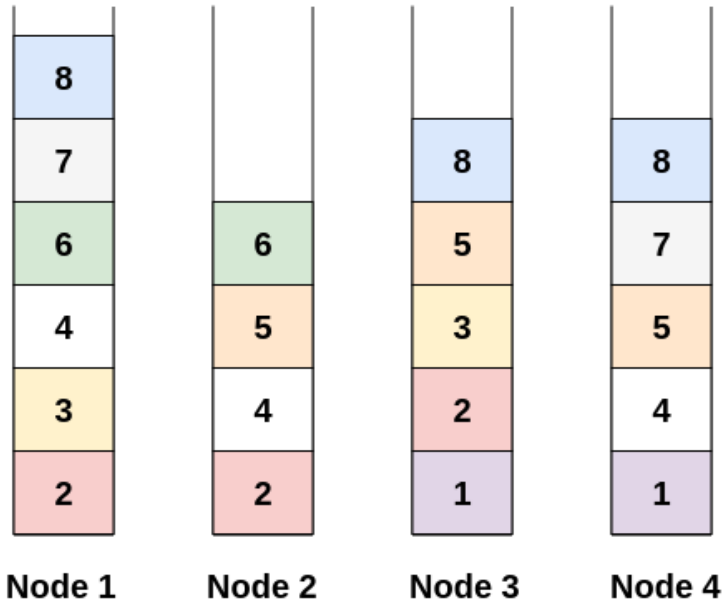


Figure 6.6: Example of data replications where replication factor varies among different partitions.

6.7 Experiments

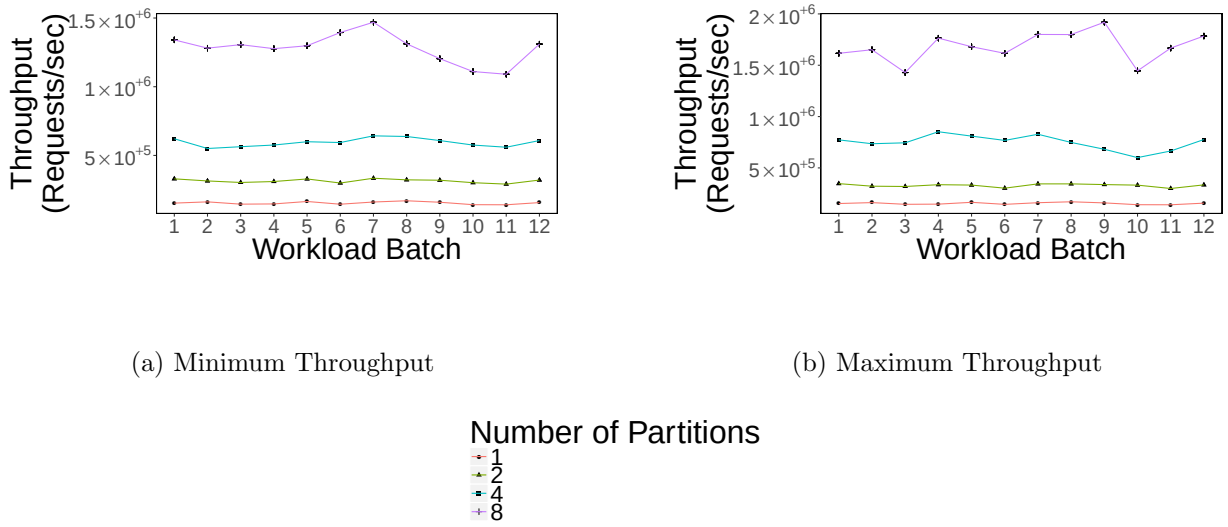
GeoBalance's architecture allows the framework to scale in two dimensions. First, we can add new partitions by adding new nodes and launching data node services on those nodes. Data nodes adapt a "shared nothing" architecture and operate independently of each other.

However, we can also scale GeoBalance by adding more data node microservices to each node to take advantage of multi-core processors. Contrary to how data nodes act across nodes, data node instances are interacting with data through a shared storage space, which is the RocksDB database. Therefore they are competing for resources and may require some level of synchronization in the ingestion process.

For the experiment of this section, we use the same simulated dataset as Chapter 5. The dataset includes 12 batches (one for each month of 2014) and includes hotspots.

In the first experiment, we study how GeoBalance scales when 1) the number of nodes increases by adding partitions (Figure 6.7), and 2) the number of data node microservices increase in a single node (Figure 6.8). We conclude that while by increasing instances in a node we may observe an slightly higher latency due to instances competing for the shared resource and the inevitable context switch, overall we reach a better throughput by interleaving request processing time and database access latency. However, we also observe that having 20 instances per node slightly outperforms having 30 instances per node. This is due to the fact that by launching too many microservices on a single node, the resource contention outweighs the benefits of concurrent request processing.

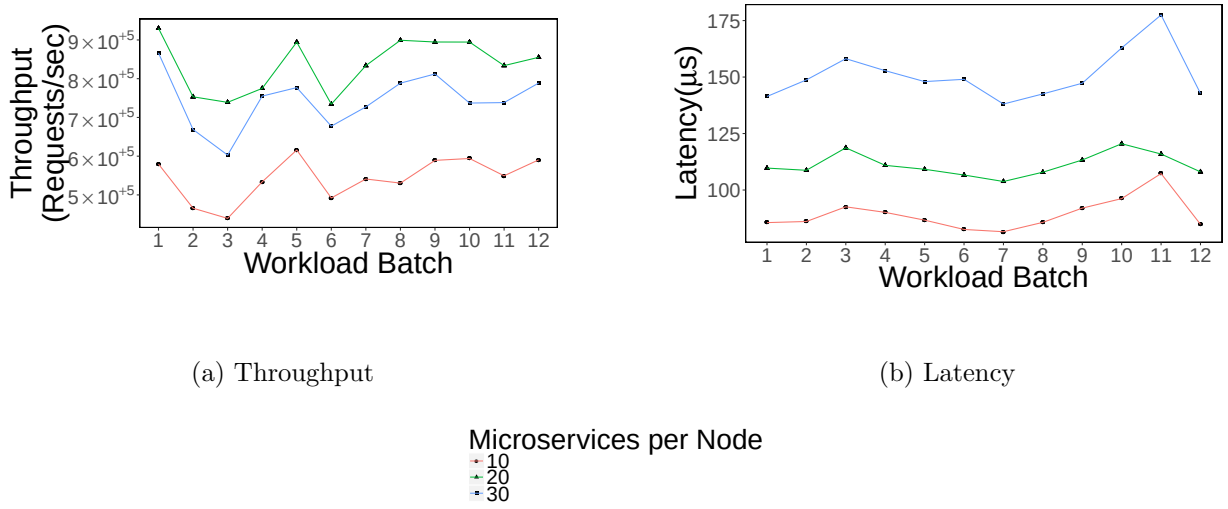
In the last experiment, we compare our evolutionary adaptive partitioning algorithm versus static partitioning, which will stay constant during the course of the experiment. While in Chapter 5 we compared these two approaches in terms of load variation and load imbalance, in this experiment we examine the effect of these algorithms on the throughput. Figure 6.9a demonstrates that our algorithm consistently outperforms static partitioning for all batches of data (although for the first batch, both algorithms practically use the same partitioning). However, the overall throughput is dominated by the effect of long-running queries, which result in thousands of responses. Therefore, in Figure 6.9b we examine the throughput by looking into the distribution of time it took partitions to serve all the requests (ingestion + retrieval). This figure shows that partitions which were generated by the static partitioning algorithm are a) inconsistent in their running time as they may deal with a significantly



(a) Minimum Throughput

(b) Maximum Throughput

Figure 6.7: Partitions' minimum and maximum throughput for increasing number of partitions (number of microservices per node = 20).

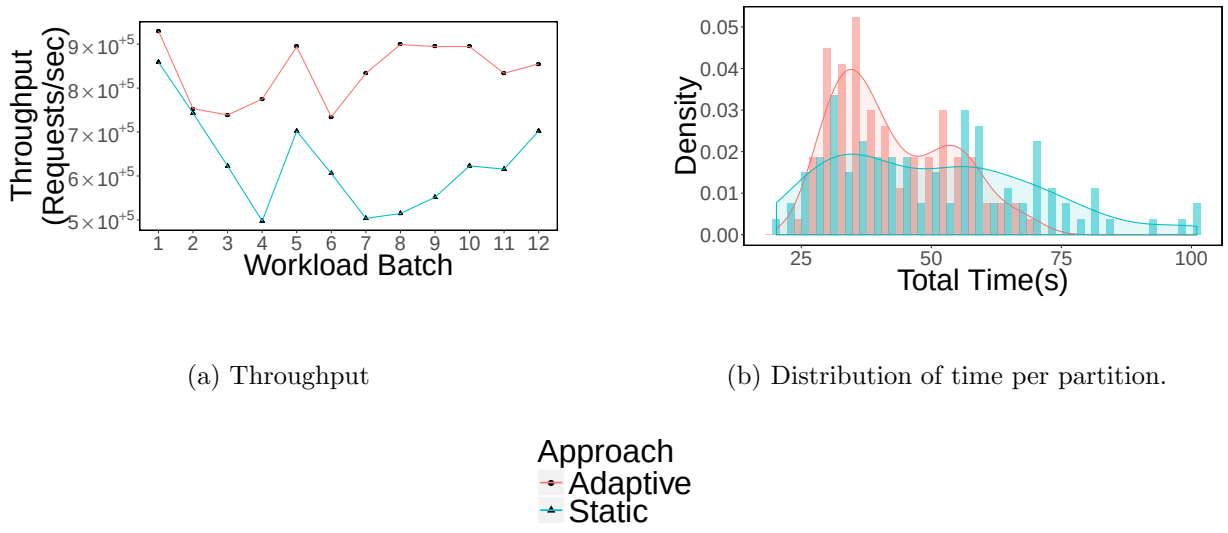


(a) Throughput

(b) Latency

Figure 6.8: GeoBalance throughput and latency when number of microservices per node increase (number of partitions = 8).

different number of requests and b) may experience long running times (the long tail). On the other hand, our approach produces partitions which deal with roughly the same number of requests, which leads to less deviation in their running time and better throughput.



(a) Throughput

(b) Distribution of time per partition.

Figure 6.9: Adaptive vs. static partitioning.

6.8 Concluding Discussions

In this chapter, we presented GeoBalance as a workload-aware framework to manage real-time spatiotemporal data that deploys our adaptive partitioning algorithm (Chapter 5). GeoBalance takes advantage of a highly decentralized microservice-based architecture which provides scalability, elasticity and availability. Particularly, we discussed GeoBalance's approaches to characterize the workload, replicate data, add/remove nodes on demand and migrating partitions. In addition, we used the simulated dataset which was generated in Chapter 5 to measure observed latency and throughput of read/write requests for the GeoBalance framework.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Summary of Contributions

My research explores scalable algorithms and architectures that embrace workload as a critical component to efficiently distribute data in data-intensive geospatial applications. The workload focus introduces significant new challenges in designing both static and dynamic data-intensive applications. In this first phase of my research, I focused on scalability and evolvability of data-intensive geospatial software systems. Throughout my research, I designed algorithms that can scale for high-volume and fast-pace data streams and explored approaches that can efficiently adapt to changes in the intensity of data and related services as the system evolves. In addition, I designed and implemented multiple interactive geospatial applications in the process to a) explore real-life requirements of data-intensive geospatial applications and b) evaluate our proposed algorithms and architectures in practice. In Chapter 2, I demonstrate MovePattern, GeoHashViz, UrbanFlow and CyberGIS-Fusion as four applications based on intensive geospatial data. The requirements of such applications are the main motivations behind the rest of the thesis.

The overarching goal of the research is to use workload-aware spatial data management to achieve three critical goals in designing data-intensive software applications: scalability, maintainability and reliability [Kleppmann, 2017]:

Scalability: In Chapter 3, we introduce novel algorithms to achieve load balancing in data-intensive geospatial applications. Our approach uses a sample of the data or a theoretical analysis to build a *computational plan*, which is used to assign tasks to distributed

workers. The plan can be efficiently updated as the requirements of the applications change. We evaluated our load balancing approach using two well-known spatial operations, multi-resolution aggregation and point in polygon. Our experiments demonstrate that our load balancing approach enables the spatial operations to seamlessly scale as the data grows. In addition, we evaluated the "interactive scalability" [Liu et al., 2013] of the two applications built on top of the spatial operations. The goal of this experiment is to confirm that our approach can sustain its performance under dynamic workload. Our experiments confirmed that increasing access rate and short-term popularity of specific regions (which are quite common scenarios in data-intensive geospatial applications) do not cause significant increase in the latency of interacting with the applications.

Maintainability: In a data-intensive application, maintainability refers to the ability of the application to adapt to changes in their lifetime [Kleppmann, 2017]. This is particularly critical for today's data-intensive geospatial applications, as new data sources are revealed in a fast pace. Chapter 5 explores algorithms for adaptive workload-aware partitioning of geospatial data. Our algorithm is specifically designed to address inevitable workload changes, by modeling the partitioning process as an spatial optimization problem that is continuously evaluated against the observed workload. We propose an evolutionary algorithm to re-partition data when load imbalance is observed. Therefore, we do not partition the data from scratch, as opposed to previous methods such as Adaptive K-d Tree. Our experiments confirm the advantage of our approach over static partitioning methods and previous adaptive partitioning methods in terms of maximum load imbalance ratio, load imbalance deviation and cost of changing the partitions.

Reliability: Chapter 6 introduces the GeoBalance framework, which implements the adaptive partitioning algorithm discussed in Chapter 5. The main research contribution of GeoBalance is to achieve reliability by designing cloud-based elastic architecture and address possible failures of services. Particularly, GeoBalance realizes a) Elasticity (Section 6.4) by dynamically allocating resources to different distributed components; b) Replication (Section

6.6) and c) Rolling Migration (Section 6.5), which is used to sustain the availability of the framework when a partition change is in progress. We evaluated GeoBalance framework under different number of nodes and services per node. Our experiments highlight the measured performance of GeoBalance (in terms of throughput and latency) using innovative microservice-based architecture, in addition to taking advantage of the adaptive partitioning algorithm.

7.2 Future Work

The workload-aware management of geospatial big data is a research topic that has gained increasing attention during recent years [Aly et al., 2015, Taft et al., 2014, Serafini et al., 2016, Quamar et al., 2013]. This is particularly critical in cloud-based architectures, since if data cannot be optimally distributed, we have to allocate resources based on the maximum needs of applications which leads to poor performance utilization [Serafini et al., 2016]. This challenge has been tackled by the innovation of *evolutionary architecture* which is designed to automatically adapt to the changes in both resources and workload. Evolutionary architecture has quickly became a critical factor in many data-intensive applications, as simple heuristics cannot adapt to the pace of changes.

While we studied algorithms and architectures that enable workload-aware management of dynamic and massive geospatial data, the evolutionary architecture philosophy can be applied to other important tasks in managing the data as well. For instance, one of the most important and time-consuming operations in data-intensive geospatial applications is multi-dimensional indexing [Malensek et al., 2016]. In many real-world applications, the geospatial data sources are accompanied by other types of data, e.g. time, environmental measures and texts. In this scenario, an effective indexing approach is desirable to index the data in multiple dimensions to take into account the growing complexity of the data. However, according to evolutionary architecture, the data intensity may grow in different

directions over time. Therefore, the indexing approach should be able to address this growth by optimizing indexing over time. One immediate use case of such indexing approach is smart city applications. In such applications data comes from multiple different sources, measuring fundamentally different quantities with varying spatial and temporal resolutions. An adaptive geospatially tuned indexing approach could digest such data without manual intervention that is not practical for dynamic data sources.

In addition, I am planning to extend our workload modeling to a graph-based model where the intensity of a workload is not only defined based on the intensity of each spatial unit, but also with the joint intensity among neighboring spatial units. This graph-based model can be effectively used in data partitioning processes to assign spatial units that are accessed more frequently with each other (in both query and data workload) into the same partition. This strategy promises to dramatically improve the performance of the partitioning approach by eliminating the number of distributed transactions, which is proved to be a major cause of performance degradation in distributed data management [Serafini et al., 2016].

REFERENCES

[esr, 2013] (2013). Esri gis-tools-for-hadoop. https://github.com/Esri/gis-tools-for-hadoop.

[Achakeev et al., 2012] Achakeev, D., Seeger, B., and Widmayer, P. (2012). Sort-based query-adaptive loading of r-trees. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2080–2084. ACM.

[Aji et al., 2013a] Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., and Saltz, J. (2013a). Hadoop gis: a high performance spatial data warehousing system over mapreduce. *Proceedings of the VLDB Endowment*, 6(11):1009–1020.

[Aji et al., 2013b] Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., and Saltz, J. (2013b). Hadoop gis: A high performance spatial data warehousing system over mapreduce. *Proc. VLDB Endow.*, 6(11):1009–1020.

[Aly et al., 2015] Aly, A. M., Mahmood, A. R., Hassan, M. S., Aref, W. G., Ouzzani, M., Elmeleegy, H., and Qadah, T. (2015). Aqwa: adaptive query workload aware partitioning of big spatial data. *Proceedings of the VLDB Endowment*, 8(13):2062–2073.

[Amanatides and Woo, 1987] Amanatides, J. and Woo, A. (1987). A fast voxel traversal algorithm for ray tracing. In *In Eurographics '87*, pages 3–10.

[Arzuaga and Kaeli, 2010] Arzuaga, E. and Kaeli, D. R. (2010). Quantifying load imbalance on virtualized enterprise servers. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, WOSP/SIPEW '10, pages 235–242, New York, NY, USA. ACM.

[Beasley et al., 1993] Beasley, D., Bull, D. R., and Martin, R. R. (1993). An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69.

[Berger and Bokhari, 1987] Berger, M. and Bokhari, S. (1987). A partitioning strategy for nonuniform problems on multiprocessors. *Computers, IEEE Transactions on*, C-36(5):570–580.

[Bloom, 1970] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

[Blum et al., 1973] Blum, M., Floyd, R. W., Pratt, V. R., Rivest, R. L., and Tarjan, R. E. (1973). Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461.

[Bora et al., 2014] Bora, N., Chang, Y.-H., and Maheswaran, R. (2014). *Social Computing, Behavioral-Cultural Modeling and Prediction: 7th International Conference, SBP 2014, Washington, DC, USA, April 1-4, 2014. Proceedings*, chapter Mobility Patterns and User Dynamics in Racially Segregated Geographies of US Cities, pages 11–18. Springer International Publishing, Cham.

[Cao et al., 2015] Cao, G., Wang, S., Hwang, M., Padmanabhan, A., Zhang, Z., and Soltani, K. (2015). A scalable framework for spatiotemporal analysis of location-based social media data. *Computers, Environment and Urban Systems*, 51:70 – 82.

[Cary et al., 2009a] Cary, A., Sun, Z., Hristidis, V., and Rishe, N. (2009a). Experiences on processing spatial data with mapreduce. In *International Conference on Scientific and Statistical Database Management*, pages 302–319. Springer.

[Cary et al., 2009b] Cary, A., Sun, Z., Hristidis, V., and Rishe, N. (2009b). *Experiences on Processing Spatial Data with MapReduce*, pages 302–319. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Chang,] Chang, H.-C. A new perspective on twitter hashtag use: Diffusion of innovation theory. *Proceedings of the American Society for Information Science and Technology*, 47(1):1–4.

[Cooper et al., 2010] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA. ACM.

[Curino et al., 2010] Curino, C., Jones, E., Zhang, Y., and Madden, S. (2010). Schism: A workload-driven approach to database replication and partitioning. *Proc. VLDB Endow.*, 3(1-2):48–57.

[Daae Lampe and Hauser, 2011] Daae Lampe, O. and Hauser, H. (2011). Interactive visualization of streaming data with kernel density estimation. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium*, PACIFICVIS '11, pages 171–178, Washington, DC, USA. IEEE Computer Society.

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

[DeRose et al., 2007] DeRose, L., Homer, B., and Johnson, D. (2007). Detecting application load imbalance on high end massively parallel systems. In *Proceedings of the 13th International Euro-Par Conference on Parallel Processing*, Euro-Par'07, pages 150–159, Berlin, Heidelberg. Springer-Verlag.

[Dong et al., 2017] Dong, S., Callaghan, M., Galanis, L., Borthakur, D., Savor, T., and Strum, M. (2017). Optimizing space amplification in rocksdb. In *CIDR*.

[Eldawy, 2014] Eldawy, A. (2014). Spatialhadoop: Towards flexible and scalable spatial processing using mapreduce. In *Proceedings of the 2014 SIGMOD PhD Symposium*, SIGMOD'14 PhD Symposium, pages 46–50, New York, NY, USA. ACM.

[Eldawy et al., 2015a] Eldawy, A., Alarabi, L., and Mokbel, M. F. (2015a). Spatial partitioning techniques in spatialhadoop. *Proc. VLDB Endow.*, 8(12):1602–1605.

[Eldawy and Mokbel, 2015] Eldawy, A. and Mokbel, M. F. (2015). The era of big spatial data: Challenges and opportunities. In *Proceedings of the 2015 16th IEEE International Conference on Mobile Data Management - Volume 02*, MDM '15, pages 7–10, Washington, DC, USA. IEEE Computer Society.

[Eldawy et al., 2015b] Eldawy, A., Mokbel, M. F., Alharthi, S., Alzaidy, A., Tarek, K., and Ghani, S. (2015b). Shahed: A mapreduce-based system for querying and visualizing spatio-temporal satellite data. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 1585–1596. IEEE.

[Ester et al., 1996] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press.

[Fox et al., 2013] Fox, A., Eichelberger, C., Hughes, J., and Lyon, S. (2013). Spatio-temporal indexing in non-relational distributed databases. In *Big Data, 2013 IEEE International Conference on*, pages 291–299.

[Frias-Martinez and Frias-Martinez, 2014] Frias-Martinez, V. and Frias-Martinez, E. (2014). Spectral clustering for sensing urban land use using twitter activity. *Engineering Applications of Artificial Intelligence*, 35:237–245.

[Fu et al., 2018] Fu, Q., Timkovich, N. P., Riteau, P., and Keahey, K. (2018). A step towards hadoop dynamic scaling. In *Proceedings of the 20th IEEE International Conference on High Performance Computing and Communications (HPCC-2018)*.

[Gansner et al., 2011] Gansner, E., Hu, Y., North, S., and Scheidegger, C. (2011). Multilevel agglomerative edge bundling for visualizing large graphs. In *Pacific Visualization Symposium (PacificVis), 2011 IEEE*, pages 187–194.

[Ghosh et al., 2016] Ghosh, M., Xu, L., Qian, X., Kao, T., Gupta, I., and Gupta, H. (2016). Getafix: Workload-aware distributed interactive analytics. *UIUC Ideals*.

[Graham, 1972] Graham, R. L. (1972). An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133.

[Guibas and Sedgewick, 1978] Guibas, L. J. and Sedgewick, R. (1978). A dichromatic framework for balanced trees. In *Foundations of Computer Science, 1978., 19th Annual Symposium on*, pages 8–21. IEEE.

[Guo, 2009] Guo, D. (2009). Flow mapping and multivariate visualization of large spatial interaction data. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1041–1048.

[Gupta et al., 2016] Gupta, A., Yang, F., Govig, J., Kirsch, A., Chan, K., Lai, K., Wu, S., Dhoot, S., Kumar, A. R., Agiwal, A., Bhansali, S., Hong, M., Cameron, J., Siddiqi, M., Jones, D., Shute, J., Gubarev, A., Venkataraman, S., and Agrawal, D. (2016). Mesa: A geo-replicated online data warehouse for google's advertising system. *Commun. ACM*, 59(7):117–125.

[Hasselbring, 2016] Hasselbring, W. (2016). Microservices for scalability: Keynote talk abstract. In *Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering*, ICPE '16, pages 133–134, New York, NY, USA. ACM.

[Huang et al., 2002] Huang, H.-C., Cressie, N., and Gabrosek, J. (2002). Fast, resolution-consistent spatial prediction of global processes from satellite data. *Journal of Computational and Graphical Statistics*, 11(1):63–88.

[Hunt et al., 2010] Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA. USENIX Association.

[Jain, 1990] Jain, R. (1990). *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons.

[Jia et al., 2011] Jia, Y., Garland, M., and Hart, J. C. (2011). Social network clustering and visualization using hierarchical edge bundles. *Computer Graphics Forum*, 30(8):2314–2327.

[Jindal and Dittrich, 2011] Jindal, A. and Dittrich, J. (2011). Relax and let the database do the partitioning online. In *International Workshop on Business Intelligence for the Real-Time Enterprise*, pages 65–80. Springer.

[Kai and Boa, 2010] Kai, C. and Boa, H. (2010). Comparison of spatial compactness evaluation methods for simple genetic algorithm based land use planning optimization problem. In *Proceedings of the Joint International Conference on Theory, Data Handling and Modelling in GeoSpatial Information Science*, pages 26–28.

[Kamath et al., 2013] Kamath, K. Y., Caverlee, J., Lee, K., and Cheng, Z. (2013). Spatio-temporal dynamics of online memes: a study of geo-tagged tweets. In *Proceedings of the 22nd international conference on World Wide Web*, pages 667–678. ACM.

[Keahey et al., 2017] Keahey, K., Riteau, P., and Timkovich, N. P. (2017). Lambdalink: an operation management platform for multi-cloud environments. In *Proceedings of the10th International Conference on Utility and Cloud Computing*, pages 39–46. ACM.

[Keim, 2005] Keim, D. (2005). Scaling visual analytics to very large data sets. *Workshop on Visual Analytics*.

[Kim et al., 2017] Kim, Y. S., Kim, T., Carey, M. J., and Li, C. (2017). A comparative study of log-structured merge-tree-based spatial indexes for big data. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 147–150.

[Kini and Emanuele, 2014] Kini, A. and Emanuele, R. (2014). Geotrellis: Adding geospatial capabilities to spark. *Spark Summit*.

[Kleppmann, 2017] Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems*. " O'Reilly Media, Inc.".

[Kwon et al., 2012] Kwon, Y., Balazinska, M., Howe, B., and Rolia, J. (2012). Skewtune: Mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 25–36, New York, NY, USA. ACM.

[Lakshman and Malik, 2010] Lakshman, A. and Malik, P. (2010). Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40.

[Liu et al., 2014] Liu, J., Li, H., Gao, Y., Yu, H., and Jiang, D. (2014). A geohash-based index for spatial data management in distributed memory. In *2014 22nd International Conference on Geoinformatics*, pages 1–4.

[Liu et al., 2015] Liu, Y., Padmanabhan, A., and Wang, S. (2015). Cybergis gateway for enabling data-rich geospatial research and education. *Concurrency and Computation: Practice and Experience*, 27(2):395–407.

[Liu et al., 2010] Liu, Y., Wu, K., Wang, S., Zhao, Y., and Huang, Q. (2010). A mapreduce approach to gi*(d) spatial statistic. In *Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems*, HPDGIS '10, pages 11–18, New York, NY, USA. ACM.

[Liu et al., 2016] Liu, Y. Y., Cho, W. K. T., and Wang, S. (2016). Pear: a massively parallel evolutionary computation approach for political redistricting optimization and analysis. *Swarm and Evolutionary Computation*, 30:78 – 92.

[Liu and Wang, 2015] Liu, Y. Y. and Wang, S. (2015). A scalable parallel genetic algorithm for the generalized assignment problem. *Parallel computing*, 46:98–119.

[Liu et al., 2013] Liu, Z., Jiang, B., and Heer, J. (2013). immens: Real-time visual querying of big data. *Computer Graphics Forum (Proc. EuroVis)*, 32.

[Malensek et al., 2013a] Malensek, M., Pallickara, S., and Pallickara, S. (2013a). Polygon-based query evaluation over geospatial data using distributed hash tables. In *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, pages 219–226.

[Malensek et al., 2013b] Malensek, M., Pallickara, S., and Pallickara, S. (2013b). Polygon-based query evaluation over geospatial data using distributed hash tables. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 219–226. IEEE Computer Society.

[Malensek et al., 2016] Malensek, M., Pallickara, S., and Pallickara, S. (2016). Autonomous cloud federation for high-throughput queries over voluminous datasets. *IEEE Cloud Computing*, 3(3):40–49.

[Marz and Warren, 2015] Marz, N. and Warren, J. (2015). *Big Data: Principles and best practices of scalable realtime data systems*. Manning Publications Co.

[Miller and Goodchild, 2015] Miller, H. J. and Goodchild, M. F. (2015). Data-driven geography. *GeoJournal*, 80(4):449–461.

[Moussalli et al., 2015] Moussalli, R., Srivatsa, M., and Asaad, S. (2015). Fast and flexible conversion of geohash codes to and from latitude/longitude coordinates. In *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 179–186.

[Newman, 2015] Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc., 1st edition.

[Niemeyer, 2008] Niemeyer, G. (2008). Geohash.

[Nishimura et al., 2013] Nishimura, S., Das, S., Agrawal, D., and El Abbadi, A. (2013). \mathcal {MD}-hbase: design and implementation of an elastic data infrastructure for cloud-scale location services. *Distributed and Parallel Databases*, 31(2):289–319.

[O'Neil et al., 1996] O'Neil, P., Cheng, E., Gawlick, D., and O'Neil, E. (1996). The log-structured merge-tree (lsm-tree). *Acta Inf.*, 33(4):351–385.

[Padmanabhan et al., 2014] Padmanabhan, A., Wang, S., Cao, G., Hwang, M., Zhang, Z., Gao, Y., Soltani, K., and Liu, Y. (2014). Flumapper: A cybergis application for interactive analysis of massive location-based social media. *Concurr. Comput. : Pract. Exper.*, 26(13):2253–2265.

[Pavlo et al., 2012] Pavlo, A., Curino, C., and Zdonik, S. (2012). Skew-aware automatic database partitioning in shared-nothing, parallel oltp systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 61–72, New York, NY, USA. ACM.

[Quamar et al., 2013] Quamar, A., Kumar, K. A., and Deshpande, A. (2013). Sword: scalable workload-aware data placement for transactional workloads. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 430–441. ACM.

[Romero et al., 2011] Romero, D. M., Meeder, B., and Kleinberg, J. (2011). Differences in the mechanics of information diffusion across topics: Idioms, political hashtags, and complex contagion on twitter. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 695–704, New York, NY, USA. ACM.

[Serafini et al., 2016] Serafini, M., Taft, R., Elmore, A. J., Pavlo, A., Aboulnaga, A., and Stonebraker, M. (2016). Clay: fine-grained adaptive partitioning for general database schemas. *Proceedings of the VLDB Endowment*, 10(4):445–456.

[Sheng et al., 2010] Sheng, C., Zheng, Y., Hsu, W., Lee, M. L., and Xie, X. (2010). Answering top-k similar region queries. In *Proceedings of the 15th International Conference on Database Systems for Advanced Applications - Volume Part I*, DASFAA'10, pages 186–201, Berlin, Heidelberg. Springer-Verlag.

[Shneiderman, 1996] Shneiderman, B. (1996). The eyes have it: a task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 336–343.

[Shvachko et al., 2010a] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010a). The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, pages 1–10.

[Shvachko et al., 2010b] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. (2010b). The hadoop distributed file system. In *Mass storage systems and technologies (MSST), 2010 IEEE 26th symposium on*, pages 1–10. IEEE.

[Soliman et al., 2017] Soliman, A., Soltani, K., Yin, J., Padmanabhan, A., and Wang, S. (2017). Social sensing of urban land use based on analysis of twitter users' mobility patterns. *PloS one*, 12(7):e0181657.

[Soliman et al., 2015] Soliman, A., Yin, J., Soltani, K., Padmanabhan, A., and Wang, S. (2015). Where chicagoans tweet the most: Semantic analysis of preferential return locations of twitter users. In *Proceedings of the First ACM SIGSPATIAL International Workshop on Smart Cities and Urban Analytics*, UrbanGIS '15.

[Soltani et al., 2015a] Soltani, K., Padmanabhan, A., and Wang, S. (2015a). Movepattern: Interactive framework to provide scalable visualization of movement patterns. In *Proceedings of the 8th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, IWCTS '15.

[Soltani et al., 2015b] Soltani, K., Parameswaran, A., and Wang, S. (2015b). Geohashviz: Interactive analytics for mapping spatiotemporal diffusion of twitter hashtags. In *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, XSEDE '15, pages 37:1–37:2, New York, NY, USA. ACM.

[Soltani et al., 2016] Soltani, K., Soliman, A., Padmanabhan, A., and Wang, S. (2016). Urbanflow: Large-scale framework to integrate social media and authoritative landuse maps. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale*, XSEDE16, pages 2:1–2:8, New York, NY, USA. ACM.

[Taft et al., 2014] Taft, R., Mansour, E., Serafini, M., Duggan, J., Elmore, A. J., Aboulnaga, A., Pavlo, A., and Stonebraker, M. (2014). E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proceedings of the VLDB Endowment*, 8(3):245–256.

[Tang et al., 2011] Tang, W., Bennett, D. A., and Wang, S. (2011). A parallel agent-based model of land use opinions. *Journal of Land Use Science*, 6(2-3):121–135.

[Tzoumas et al., 2009] Tzoumas, K., Yiu, M. L., and Jensen, C. S. (2009). Workload-aware indexing of continuously moving objects. *Proceedings of the VLDB Endowment*, 2(1):1186–1197.

[Wakamiya et al., 2011] Wakamiya, S., Lee, R., and Sumiya, K. (2011). Urban area characterization based on semantics of crowd activities in twitter. In *International Conference on GeoSpatial Sematics*, pages 108–123. Springer.

[Wang, 2010] Wang, S. (2010). A cybergis framework for the synthesis of cyberinfrastructure, gis, and spatial analysis. *Annals of the Association of American Geographers*, 100(3):535–557.

[Wang, 2016] Wang, S. (2016). Cybergis and spatial data science. *GeoJournal*, 81(6):965–968.

[Wang and Armstrong, 2005] Wang, S. and Armstrong, M. P. (2005). A theory of the spatial computational domain. In *Proceedings of GeoComputation*, pages 1–3.

[Wang and Armstrong, 2009] Wang, S. and Armstrong, M. P. (2009). A theoretical approach to the use of cyberinfrastructure in geographical analysis. *Int. J. Geogr. Inf. Sci.*, 23(2):169–193.

[Wang et al., 2005] Wang, S., Armstrong, M. P., Ni, J., and Liu, Y. (2005). Gisolve: A grid-based problem solving environment for computationally intensive geographic information analysis. In *challenges of large applications in distributed environments, 2005. CLADE 2005. proceedings*, pages 3–12. IEEE.

[Wang and Goodchild, 2019] Wang, S. and Goodchild, M. F. (2019). *CyberGIS for Transforming Geospatial Discovery and Innovation*, pages 3–10. Springer Netherlands, Dordrecht.

[Wang et al., 2015] Wang, S., Hu, H., Lin, T., Liu, Y., Padmanabhan, A., and Soltani, K. (2015). Cybergis for data-intensive knowledge discovery. *SIGSPATIAL Special*, 6(2):26–33.

[Wang et al., 2016] Wang, S., Liu, Y., and Padmanabhan, A. (2016). Open cybergis software for geospatial research and education in the big data era. *SoftwareX*, 5:1–5.

[Wu and Murray, 2008] Wu, X. and Murray, A. T. (2008). A new approach to quantifying spatial contiguity using graph theory and spatial interaction. *International Journal of Geographical Information Science*, 22(4):387–407.

[Yang et al., 2011] Yang, C., Goodchild, M., Huang, Q., Nebert, D., Raskin, R., Xu, Y., Bambacus, M., and Fay, D. (2011). Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? *International Journal of Digital Earth*, 4(4):305–329.

[Yu et al., 2015] Yu, J., Wu, J., and Sarwat, M. (2015). Geospark: A cluster computing framework for processing large-scale spatial data. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 70. ACM.

[Zhang et al., 2015] Zhang, J., You, S., and Gruenwald, L. (2015). Lightweight distributed execution engine for large-scale spatial join query processing. In *Big Data (BigData Congress), 2015 IEEE International Congress on*, pages 150–157.

[Zinsmaier et al., 2012] Zinsmaier, M., Brandes, U., Deussen, O., and Strobelt, H. (2012). Interactive level-of-detail rendering of large graphs. *IEEE Transaction on Visualization and Computer Graphics*, 18(12):2486–2495.