University of Cyprus

DEPARTMENT OF COMPUTER SCIENCE

ALGORITHMS AND INDEXING STRUCTURES FOR SPATIAL BIG DATA

Constantinos Costa

A Dissertation Submitted to the University of Cyprus in Partial

Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

JULY, 2018

VALIDATION PAGE

Doctoral Candidate: Constantinos Costa

Doctoral Dissertation Title: ALGORITHMS AND INDEXING STRUCTURES FOR SPATIAL BIG DATA

The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Department of Computer Science and was approved on the ***JULY 02, 2018*** *by the members of the Examination Committee.*

Examination Committee:

Research Supervisor _____
Dr. Demetrios Zeinalipour-Yazti

Committee Member _____
Prof. Marios D. Dikaiakos

Committee Member _____
Prof. George S. Samaras

Committee Member _____
Prof. Panos K. Chrysanthis

Committee Member _____
Dr. Herodotos Herodotou

DECLARATION OF DOCTORAL CANDIDATE

The present Doctoral Dissertation was submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy of the University of Cyprus. It is a product of original work of my own, unless otherwise mentioned through references, notes, or any other statements.

Constantinos Costa

Περίληψη

Ζούμε στην εποχή των μεγάλων δεδομένων, όπου παράγονται ετερογενή δεδομένα τα οποία αποθηκεύονται με εκθετικό ρυθμό. Τα παραδοσιακά συστήματα διαχείρισης δεδομένων δεν είναι ικανά να συλλέγουν, να διαχειρίζονται και να επεξεργάζονται τα μεγάλα δεδομένα μέσα σε ανεκτά χρονικά πλαίσια. Σημαντικό μέρος των μεγάλων δεδομένων είναι τα *χωρικά μεγάλα δεδομένα (Spatial Big Data - SBD)*, τα οποία αντιπροσωπεύουν γεωγραφικές δομές δεδομένων που υπερβαίνουν και πάλι την ικανότητα των παραδοσιακών χωρικών υπολογιστικών συστημάτων λόγω του όγκου, της ποικιλίας και της ταχύτητας των δεδομένων. Τα SBD φέρνουν πολλές νέες προκλήσεις για νέες αρχιτεκτονικές διαχείρισης SBD. Συγκεκριμένα, οι πιο πρόσφατες δουλειές χρησιμοποιούν κατανεμημένα περιβάλλοντα για την υλοποίηση χωρικών τελεστών όπως, kNN, συνένωσης, αθροίσματος και επιλογής πάνω από τις αρχιτεκτονικές μεγάλων δεδομένων. Επιπρόσθετα, αρκετές από τις προηγούμενες λύσεις χρησιμοποιούν τεχνικές κατανεμημένης ευρετηρίασης για την ενίσχυση της διαδικασίας αποθήκευσης και της εκτέλεσης των χωρικών τελεστών. Δυστυχώς, οι προτεινόμενες αρχιτεκτονικές είναι αγνωστικές ως προς τις δυνατότητες επεξεργασίας και αποθήκευσης του συστήματος και δεν καταφέρνουν να διατηρήσουν τη βελτιστοποιημένη απόδοση αποθήκευσης και να ικανοποιήσουν τις απαιτήσεις για αναλυτικά και επιχειρησιακά ερωτήματα σε πραγματικό χρόνο.

Σε αυτή τη διδακτορική διατριβή παρουσιάζουμε αλγόριθμους και δομές δεδομένων που επιλύουν σημαντικές προκλήσεις των ΣΒΔ, ειδικότερα σε ότι αφορά τον χρόνο απόκρισης σε επερωτήσεις και την αποδοτικότητα αποθήκευσης δεδομένων. Οι ερευνητικές συνεισφορές της διατριβής πλαισιώνονται μέσα σε μια νέα αρχιτεκτονική η οποία βασίζεται στην απόδοση, με ονομασία *SPATE*$^{+}$, όπου η αποθήκευση, η ευρετηρίαση, η επεξεργασία

επερωτήσεων και τα συστατικά της εφαρμογής μπορούν να επιτύχουν καλύτερη απόδοση από τις εξελίξεις αιχμής. Η αρχιτεκτονική *SPATE*$^+$ χρησιμοποιεί συμπιεσμένα δεδομένα χωρίς απώλεια της αρχικής πληροφορίας για να αποθηκεύσει τα πρόσφατα ρεύματα SBD με τον πιο συμπαγή τρόπο, διατηρώντας τη δυνατότητα για πλήρη ανάλυση των δεδομένων. Το *SPATE*$^+$ εξασφαλίζει την προοδευτική απώλεια της λεπτομέρειας των δεδομένων, που ονομάζεται φθορά δεδομένων, καθώς τα δεδομένα γερνούν με το χρόνο. Συγκεκριμένα, παρουσιάζουμε ένα νέο καινοτόμο τελεστή φθοράς για Μεγάλα Τηλεπικοινωνιακά Δεδομένα (TBD), με ονομασία *TBD-DP (Επίρρηση Δεδομένων)*. Ο τελεστής *TBD-DP* βασίζεται σε γνωστούς αλγόριθμους μηχανικής μάθησης για να γενικεύσει τα TBD σε συμπαγή μοντέλα τα οποία μπορούν να αποθηκευτούν και να επερωτηθούν όταν είναι απαραίτητο.

Η αρχιτεκτονική *SPATE*$^+$ είναι εννοιολογικά χωρισμένη σε τρία επίπεδα: (ι) *αποθήκευση και ευρετηρίαση*, όπου επιτυγχάνεται υψηλή απόδοση αποθήκευσης χρησιμοποιώντας τα συστατικά στοιχεία συμπίεσης και αποσύνθεσης. Αυτή η κατηγορία περιλαμβάνει επίσης ένα χωροχρονικό ευρετήριο με τέσσερα επίπεδα χρονικών αναλύσεων που ελαχιστοποιούν το χρόνο απόκρισης για ερωτήματα αναζήτησης δεδομένων. (ιι) *τελεστές*, όπου ένας τελεστής ερωτημάτων AkNN υψηλής απόδοσης υπερβαίνει τις τελευταίες τεχνικές από την άποψη της αποτελεσματικής κατανομής, αντιγραφής και διύλισης δεδομένων. (ιιι) *εφαρμογές*, όπου ένα αποτελεσματικό πλαίσιο εξερεύνησης ερωτημάτων, με ονομασία SPATE, παρέχει νέες λειτουργίες εξερεύνησης δεδομένων στο χρήστη. Επιπλέον, μια ανώνυμη εφαρμογή ανταλλαγής μηνυμάτων, με ονομασία Rayzit, χρησιμοποιεί SBD για να συνδέσει άμεσα τους χρήστες με τους πλησιέστερους γείτονες (kNN) καθώς κινούνται στον χώρο για να εκθέσει τις συνεισφορές των συστατικών που είναι μέρος της αρχιτεκτονικής *SPATE*$^+$.

Abstract

We live in the big data era where heterogeneous data is produced and stored at an exponential rate. Recently, many solutions have emerged to handle big data for which traditional data management systems are not capable to capture, manage, and process the data within a tolerable elapsed time. A significant portion of big data is *Spatial Big Data (SBD)*, which represents massive geographic objects that again exceed the capability of traditional spatial computing systems due to volume, variety and velocity characteristics. SBD brings many new challenges for novel SBD management architectures. Particularly, recent works are utilizing distributed environments to implement spatial operators like kNN, Joins, Aggregations and Selections on top of big data architectures. Additionally, several of the previous solutions are using distributed indexing techniques to enhance the storing process and the spatial operators. Unfortunately, the proposed architectures are agnostic of underline processing and storage capabilities failing to maintain optimized storage efficiency and satisfy the requirements for online analytical and operational queries.

In this PhD thesis we present algorithms and indexing structures that tackle critical challenges brought forward by SBD, namely query response time and storage efficiency. We frame our research contributions in the context of a novel performance-driven architecture, named *SPATE*$^{+}$, where the storage, indexing, query processing and application components of the architecture can achieve better utilization and efficiency than the state-of-the-art. *SPATE*$^{+}$ uses lossless data compression to ingest recent streams of SBD in the most compact manner retaining full resolution for data exploration tasks. More importantly, it takes care of the progressive loss of detail in information, called

decaying, as data ages with time. Particularly, we present a novel decaying operator for Telco Big Data (TBD), coined *TBD-DP (Data Postdiction)*. *TBD-DP* relies on existing machine learning algorithms to abstract TBD into compact models that can be stored and queried when necessary.

SPATE$^+$ is conceptually divided into three layers: (i) *storage and indexing*, where high storage efficiency is achieved using the compression and decay components. This category also includes a spatio-temporal index with four levels of temporal resolutions minimizing the query response time for data exploration queries; (ii) *operators*, where a high performance AkNN query operator outperforms state-of-the-art techniques in terms of efficient partitioning, replication and refinement; and (iii) *applications*, where an efficient query exploration framework, called SPATE, provides novel data exploration functionalities to the user. We also present an anonymous crowd messaging application, called Rayzit, which utilizes SBD to connect the users instantly to their k Nearest Neighbors (kNN) as they move in space, to expose the contributions of the components that are part of *SPATE*$^+$.

Acknowledgments

First of all, I would like to thank my thesis advisor, Associate Professor Demetrios Zeinalipour-Yazti for his constant guidance and encouragement throughout my PhD studies. I would like to thank him for his infinite patience and his perfectionism that allowed me to learn how to do high quality research. Particularly, I would like to thank him for the ample time he devoted for teaching me how to review, write and present with a high level of scientific professionalism.

Secondly, I would like to thank the fellow researchers at the Data Management Systems Laboratory (DMSL). I would like to thank Dr. George Chatzimilioudis and Dr. Andreas Konstantinidis for allowing me to have a close collaboration with them and expand my research horizon.

I would like to thank my parents, Despoina and Ermogenis for their unconditional love and support, through this long process. I sincerely thank my brother, Dr. Mario Costa, who devoted a lot of time helping with personal and work stuff during my studies.

I truly thank my best friend, Diomidis Papadiomidous for being supportive throughout my studies and the exceptional help for proof reading my work. I would like to thank Dr. Andreas Diavastos for the intriguing discussions about work and life during lunches, coffee breaks and dinners.

I also thank my friends for providing support and opportunities to escape from everything. Thank you Elena Michael, Giorgos Hadjizorzis, Ioanna Herakleous, Kyriakos Herakleous, Onisiforos Onoufriou, Penny Toungoulou, Salwmi Demou, Theodoros demetriou, Claire Papadopoulou, Demetris Trihinas, George Matheou, Marios Mintzis,

George Larkou and George Nikolaides.

A big thank you goes to all those of you who have been on my side during my graduate studies. Your love and support is definitely appreciated.

Dedication

I dedicate this dissertation to my wonderful deeply missed mom, who passed away during my PhD studies.

She was a constant source of motivation and inspiration for my life. She taught me how to face the upcoming challenges with faith and humility.

Although she is not here, she will always provide me with strength to carry on and achieve my goals.

Thank you!

Contributions of this Thesis

Journal publications:

1. Georgios Chatzimilioudis, **Constantinos Costa**, Demetrios Zeinalipour-Yazti, Wang-Chien Lee. *"Crowdsourcing Emergency Data in Non-Operational Cellular Networks"*, Information Systems (InfoSys '17), Elsevier Science Ltd., Volume 64, Pages: 292 - 302, Oxford, UK, 2017.

2. Georgios Chatzimilioudis, **Constantinos Costa**, Demetrios Zeinalipour-Yazti, Wang-Chien Lee and Evaggelia Pitoura. *"Distributed In-Memory Processing of All k Nearest Neighbor Queries"*, IEEE Transactions on Knowledge and Data Engineering (TKDE '16), Volume 28, Pages: 925-938, 2016

Conference and workshop proceedings:

3. **Constantinos Costa**, Andreas Charalampous, Andreas Konstantinidis, Demetrios Zeinalipour-Yazti, and Mohamed F. Mokbel. *"Decaying Telco Big Data with Data Postdiction"*, Proceedings of the 19th IEEE International Conference on Mobile Data Management (MDM '18), IEEE Computer Society, ISBN: 978-1-5386-4133-0, pp. 106–115, June 25 - June 28, 2018, AAU, Aalborg, Denmark, 2018.

4. **Constantinos Costa**, Andreas Charalampous, Andreas Konstantinidis, Demetrios Zeinalipour-Yazti, and Mohamed F. Mokbel. *"TBD-DP: Telco Big Data Visual Analytics with Data Postdiction"*, Proceedings of the 19th IEEE International Conference on Mobile Data Management (MDM '18), IEEE Computer Society, ISBN: 978-1-5386-4133-0, pp. 280–281, June 25 - June 28, 2018, AAU, Aalborg, Denmark, 2018.

5. **Constantinos Costa**, Georgios Chatzimilioudis, Demetrios Zeinalipour-Yazti, Mohamed F. Mokbel. *"Towards Real-Time Road Traffic Analytics using Telco Big Data"*, Proceedings of the 11th Intl. Workshop on Real-Time Business Intelligence and Analytics, collocated with VLDB 2017 (BIRTE '17), ACM International Conference Proceedings Series, pp. 5:1–5:5, August 28, 2017, Munich, Germany, ISBN: 978-1-4503-5425-7/17/08, 2017.

6. **Constantinos Costa**, Georgios Chatzimilioudis, Demetrios Zeinalipour-Yazti, Mohamed F. Mokbel. *"Efficient Exploration of Telco Big Data with Compression and Decaying"*, Proceedings of the IEEE 33rd International Conference on Data Engineering (ICDE '17), IEEE Computer Society, pp. 1332-1343, April 19-22, 2017, San Diego, CA, USA, ISBN: 978-1-5090-6543-1, 2017.

7. **Constantinos Costa**, Georgios Chatzimilioudis, Demetrios Zeinalipour-Yazti, Mohamed F. Mokbel. *"SPATE: Compacting and Exploring Telco Big Data"*, Proceedings of the IEEE 33rd International Conference on Data Engineering (ICDE '17), IEEE Computer Society, pp. 1419-1420, April 19-22, 2017, San Diego, CA, USA, ISBN: 978-1-5090-6543-1, 2017.

8. Georgios Chatzimilioudis, **Constantinos Costa**, Demetrios Zeinalipour-Yazti, Wang-Chien Lee and Evaggelia Pitoura. *"Distributed In-Memory Processing of All k Nearest Neighbor Queries (Extended Abstract)"*, Proceedings of the IEEE 32nd International Conference on Data Engineering (ICDE '16), IEEE Computer Society, pp. 1490–1491, Helsinki, Finland, ISBN: 978-1-5090-2020-1, 2016.

9. **Constantinos Costa**, Chrysovalantis Anastasiou, Georgios Chatzimilioudis and Demetrios Zeinalipour-Yazti. *"Rayzit: An Anonymous and Dynamic Crowd Messaging Architecture"*, Proceedings of the 3rd IEEE International Workshop on Mobile Data Management, Mining, and Computing on Social Networks, collocated with IEEE MDM '15 (Mobisocial '15), Vol. 2, pp. 98-103, Pittsburgh, PA, USA, 2015.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, heterogeneous data is generated and collected at an exponential rate. We live in the big data era and we need to be able to process, analyze and extract information from data in order to exploit all the opportunities of the future. Particularly, social networks, like Facebook, Twitter, Google+, Instagram, are producing tons of terabytes each day and this data can be used for useful prediction and speculation in multiple areas. Furthermore, data is continuously populated through scientific experiments and sensor measurements. For example the CERN[1] experiments are generating one petabyte of data every second [2]. In addition, the economy is focusing more on big data as an imperative source with critical information in terms of wealth. Consequently, big data bring many new challenges for technology, innovation and computer society.

Data derived from mobile devices have spatial attributes due to the location primitive that is omnipresent in the modern world. These attributes change the way we process and analyze such big data, introducing new requirements considering the spatial aspect. *Geographic Information Systems (GISs)* were the first spatial software artifacts utilizing spatial databases as a processing engine and storage layer. The growth of geo-tagged data from social networks redirect the focus on spatial systems, which have the capabilities to geo-tag images, geo-tag videos and geo-tag text. The rapid and dynamic growth of the data has shifted the focus of big data architectures to *Spatial Big Data (SBD)* architectures.

SBD can also be generated in batches by the infrastructure of a *telecommunication company (telco)*, also named *Telco Big Data (TBD)*. While social networks are expanding, the amount of broadband mobile data is increasing exponentially and the telecom

[1]CERN: https://home.cern/

operators have an essential need for novel big data processing infrastructures in order to accommodate and analyze the vast amount of spatio-temporal network-level data. For example, a telco collects 5TBs per day, i.e., almost 2PBs per year, in a single city of 10M cell phone customers [3].

Numerous recent big data architectures have been proposed to the above challenges, thus a spatial big data architecture can be abstracted. It is important to mention that spatial big data architectures share similarities with typical big data architectures, due to the similar techniques that are required for storing, processing and analyzing the incoming data. We can divide the overall architecture into four layers as shown in Figure 1.1: (i) the application layer, which is responsible for presenting the end result to the user; (ii) the spatial layer, where the spatial query processing is executed based on spatial partitions and indexes over a big data infrastructure; (iii) the big data processing layer, where the massive incoming data is processed; and finally (iv) the storage layer, where the data can be stored in a scalable manner. Traditional systems do however lack designated internal structures and components to deal with volume and velocity. In order to comprehend the challenges and the contributions of this work, we present the rest of the thesis based on the following three categories.

- **Storage and Indexing:** Spatial management systems present different characteristics and require different architectures. In a wider perspective, spatial big data architectures need to have different techniques than the ones designed for big data architectures. Distributed processing and big data storage solutions, such as Hadoop [4] and HDFS [5], were proposed in order to scale these systems in terms of storage and processing. However, these solutions can not always be applied and match the requirements of spatial data analysis. Particularly, spatial indexes can be used in order to boost the performance of spatial query execution. Common indexes include Grid (spatial index), Z-order (curve), Quadtree, Octree, UB-tree, R-tree, etc. For example, a nearest neighbor query can easily be answered with an R-tree index [6], but using an R-tree in a distributed environment is nearly impossible due to the high computation cost of constructing the index.

- **Operators:** All the spatial query operators are inspired from traditional database management systems and are categorized into three sets: aggregation, selection and join.

  - Spatial Aggregation: The purpose of a spatial aggregation query is to com-

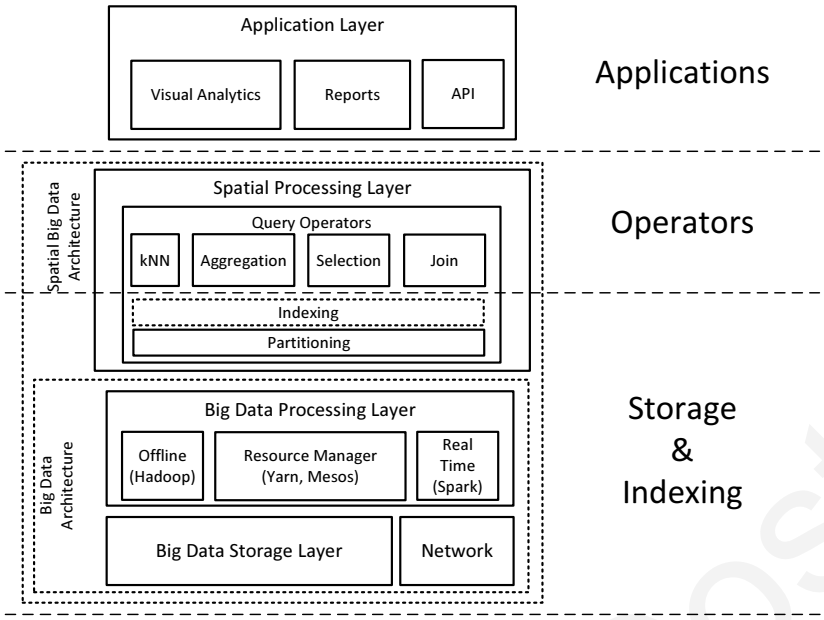

Figure 1.1: SPATE⁺ architecture components

bine several geometric characteristics in order to produce the result. For example a union query may be *"find the average area of all countries"*.

- Spatial Selection: The spatial selection query finds all the containing points within a query range or a query region, where the query region includes query rectangle, circle, polygon and point. An example range query might include *"Find all continents that are contained by a supplied query region"*.
- Spatial Join: A Spatial join is a traditional Cartesian product between two relations with the main condition based upon the spatial property of the object. To give a better explanation of what the spatial join is, we provide an example with geometric shape object in Figure 1.2. The figure illustrates how a spatial join will find the intersecting objects of relations R and S. The spatial join will return the intersection of objects $r_1 \cap s_2, r_2 \cap s_2, r_2 \cap s_3$, and $r_3 \cap s_2$ [7]. An example of a spatial join query would be *"Find roads that cross rivers"* [8].

- **Applications:** Visualizing spatial big data improves the understanding on unexplained behaviors, situations and incidents. The resulted data of an online spatial big data exploration task can be visualized in form of visual analytics, reports or served through an Application Programming Interface (API).

  During the last years, applications based on geospatial social networks have transformed the way individuals express and publish their opinions, feelings or thoug-

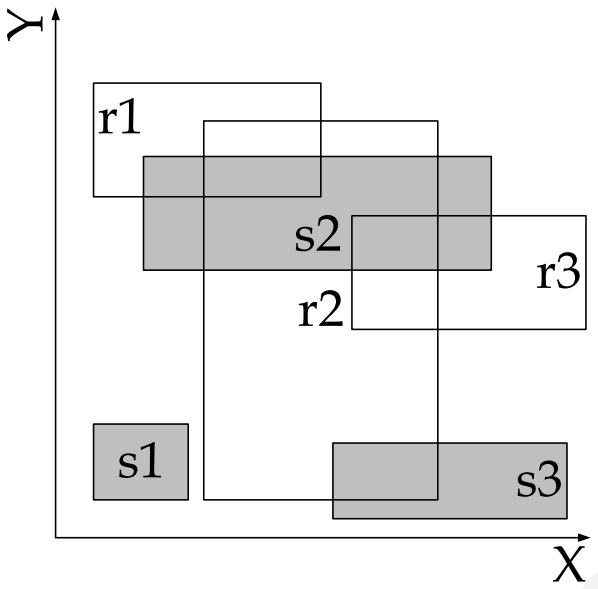

Figure 1.2: A spatial join of objects between relation R and S, where $R = r_1, r_2, r_3$ and $S = s_1, s_2, s_3$.

hts on various topics [9]. Combining the above with the increased amount of sensors on portable devices [10], gives the opportunity to extend the functionality and the intelligence of social networks. This is already illustrated by real world examples like *EARS*, a real time decision support system for earthquakes based on social networks [11].

A SBD architecture is continuously manipulating highly sensitive data and should comply with the privacy and confidentiality rules. The EU General Data Protection Regulation (GDPR), which went into force on May 2018, requires all data sharing layers of the big data architectures to provide the necessary data protection operations and tools to the users. In Chapter 2, we include the related work about privacy of spatial big data. Our proposed architecture is designed to cope with data privacy as a complementary task, but technical privacy contributions are not the focus of this PhD thesis.

1.1 Motivation

This chapter provides the motivation behind this Thesis. Particularly, we describe how important the storage and indexing of SBD over a distributed architecture can be without the degradation for response time. The execution of all the query operators over those data have an imperative role to mobile applications, ranging from social

networking, gaming and entertainment to emergency response and crisis management.

SBD can elevate the intelligence of a social network due to the fact that the location defines user interests and inquiries. Geospatial social networks provide a new way for human interaction and communication. On the other hand, Telco SBD, i.e, TBD, holds concealed value that can be exploited by telecommunication companies and the public opening new opportunities for novel solutions (e.g., real-time road traffic prediction).

Geospatial Social Networks: Classical social networks, like Facebook.com, Twitter.com, Foursquare.com, SnapChat.com and WeChat.com, create a static social graph based on the friendships between users. Friendship links need to be requested and consecutively accepted/rejected in order to be formed, causing the social graph to be built very slowly.

In contrast, applications like Secret.ly, YikYakapp.com, and Whisper.sh create dynamic social graphs based on the user's location [12]. YikYak connects you with the users within a 10 mile range and preserves anonymity. Secret and Whisper have no distance limit, but they store information about the user's profile (i.e., user id) decreasing user's anonymity. WeChat also has a feature that allows you to greet other WeChat users that are around you, which compromises the anonymity of somebody aiming to stay anonymous. *Rayzit* is an anonymous social network that enables anonymous interactions with the k nearest neighbors using state-of-the-art technology. Particularly, a dynamic social graph based on kNN is created where all users will always be connected and the anonymity and privacy of the user will be preserved.

Telco Spatial Big Data Processing: In recent years there has been considerable interest from telcos to extract concealed value from their network data. Consider for example a telco in the city of Shenzhen, China, which serves 10 million users. Such a telco is shown to produce 5TB per day [13] (i.e., thousands to millions of records every second). Huang et al. [14] break their 2.26TB telco big data per day down as follows: (i) *Business Supporting Systems (BSS)* data, which is generated by the internal workflows of a telco (e.g., billing, support), accounting to a moderate of 24GB per day and; (ii) *Operation Supporting Systems (OSS)* data, which is generated by the Radio and Core equipment of a telco, accounting to 2.2TB per day and occupying over 97% of the total volume.

Effectively processing telco big data workflows can unlock a wide spectrum of challenges, ranging from network plan optimization and user experience assessment [14]

to city planning and urban engineering [15] [16]. Zhang et al. [13] have developed *OceanRT* that improves the visualization capabilities of telco big data by the usage of standard spatio-temporal indexes. Iyer et al. [15] present *CellIQ* to optimize queries, such as "traffic hotspots" and "hand-off sequences with performance problems", using graph processing. Huang et al. [14] empirically demonstrate that churn prediction performance can be significantly improved with telco big data by integrating both BSS and OSS data. Luo et al. [16] propose a framework to predict user behavior involving more than one million telco users. Prior work is mainly concerned with the analytic exploration of telco big data, while this work optimizes the operational ability of such tasks. *SPATE* is an innovative telco big data exploration framework that includes visual and declarative interfaces for a variety of telco-specific data exploration tasks [17]. Particularly, the *SPATE UI (User Interface)* allows the user to execute a variety of template queries. The query bar includes snapshot queries and recurring queries (in the form of a time-machine) for drop calls and downflux/upflux, heatmap statistics and settings.

1.2 Problem Statement and Hypothesis

This chapter defines the problem domains which are being researched in the context of this Thesis. Additionally, research questions are provided based on the motivation described in Chapter 1 in order to comprehend the contribution of the ongoing work.

1.2.1 Research Questions

The primary research questions studied in this thesis are the enhancement of queries and analytics over large amounts and a variety of spatial big data. The research topic has a broad spectrum so it is necessary to analyze it in more specific questions. The questions researched in this thesis are provided below.

- **Storage and Indexing:** One key challenge in this new era of telco big data is to *minimize the storage costs* associated with the data exploration tasks, as big data traces and computed indexes can have a tremendous storage and I/O footprint on the data centers of telcos. Although the volume of electronically stored data doubles every year, storage capacity costs decline only at a rate of less than 1/5 per year [18]. Storing big data locally, due to the sensitive nature of data that

cannot reside on public cloud storage, adds great challenges and costs that reach beyond the simplistic capacity cost calculated per GB [19]. From an operator's perspective, the requirement is to: (i) *incrementally store big data in the most compact manner*, and (ii) *improve the response time for data exploration queries.* These two objectives are naturally conflicting, as conjectured in [20].

RQ 1 ***How can we store incremental spatial big data in the most compact manner, and at the same time improve the response time for data exploration queries?***

Hypothesis: The storage space needed to accumulate big data and the response time for data exploration tasks can be improved using innovative indexing structures.

- **Storage and Indexing:** Effectively storing and processing TBD workflows can unlock a wide spectrum of challenges, ranging from churn prediction of subscribers [14], city localization [21], 5G network optimization, user-experience assessment [15–17] and road traffic mapping [22]. Even though the acquisition of TBD is instrumental in the success of the above scenarios, telcos are reaching a point where they are collecting more data than they could possibly exploit. This has the following two implications: (i) it introduces a significant financial burden on the operator to store the collected data locally. Notice that the deep storage of data in public clouds, where economies-of-scale are available (e.g., AWS Glacier), is not an option due to privacy considerations; and (ii) it imposes a high computational cost for accessing and processing the collected data. For example, a petabyte Hadoop cluster, using between 125 and 250 nodes, costs ∼1M USD [23] and a linear scan of 1PB would require almost 15 hours. For completion, we provide a summary of the most recent costs for cloud data storage in July 2018, based on the wasabi[2] storage cost comparison tool with 20% percentage download API calls per month in Table 1.1.

RQ 2 ***How can we achieve a pre-specified reduction of Telco Big Data (TBD) without additional storage space capacity and being able to recover the decayed data accurately and efficiently?***

[2]Wasabi Technologies, Inc: `https://wasabi.com/pricing/`

Table 1.1: Popular Data Storage Solutions

Vendor	**Price per year**
Amazon (AWS)	∼540K USD
Microsoft (Azure)	∼257K USD
Google (Google Cloud Storage)	∼529K USD
Wasabi	∼62K USD

Hypothesis: A pre-specified reduction of TBD can be achieved using state-of-the-art machine learning techniques representing accurately the initial data that will be removed from the system minimizing the storage requirements. The decayed data can be recovered using the stored machine learning models.

- **Operators:** A cellular network is deemed *non-operational* when there is no (sufficient) network connectivity. This might happen due to damage caused by a disaster or due to overloading caused by an unexpectedly large crowd trying to access telecommunication services simultaneously. Each cellular tower has a limited capacity of users it can service simultaneously. Specifically, each cellular tower has a limitation on its communication bandwidth to the carrier (backhaul bandwidth).

  When the cellular network is non-operational, users can not rely on online services for their whereabouts, well-being and communication. Meanwhile, the authorities may not be able to receive all the information needed for intelligent synthesis in order to enable advanced services. In other words, the crowd is not able to generate possibly valuable information (e.g., sensors, tweets) and the authorities are not able to collect this information. Consider the example of a disaster response, the authorities need to operate in four phases: (i) decision making; (ii) implementation in the field; (iii) evaluation of the results; and (iv) making decisions. The right situational awareness is the key for decision making in such cases. The crisis management operator needs to have the right tools to communicate with the affected citizens. The operator has to evaluate and monitor the situation in order to learn and optimize operations in real time. It would be an omission that could lead to the loss of human lives, if technology could not support the

crowd to generate data (e.g., reporting locations, victims using communication and social services to spread their situation) and the crisis management operator to collect and process this data during these phases.

RQ 3 ***How can we create an overlay network that could connect cellular users by exploiting any available device-to-device short-range communication technology?***

Hypothesis: Centralized techniques can be developed to allow the fast computation of the All k Nearest Neighbors (AkNN) graph on a mobile device for non-operational cellular network scenarios. Then, an overlay network can be created to connect all the devices using the AkNN graph of each device.

- **Operators:** In the age of smart urban and mobile environments, the mobile crowd generates and consumes massive amounts of heterogeneous data [24]. Such streaming data may offer a wide spectrum of enhanced science and services, ranging from mobile gaming and entertainment, social networking, to emergency and crisis management services [1]. However, such data present new challenges in cloud-based query processing.

  One useful query for the aforementioned services is the *All kNN (AkNN)* query: *finding the k nearest neighbors for all moving objects every few seconds (batch mode).* Formally, the kNN of an object o from some dataset O, denoted as $kNN(o, O)$, are the k objects that have the most similar attributes to o [25]. Specifically, given objects $o_a \neq o_b \neq o_c$, $\forall o_b \in kNN(o_a, O)$ and $\forall o_c \in O - kNN(o_a, O)$ it always holds that $dist(o_a, o_b) \leq dist(o_a, o_c)$. An *All kNN (AkNN)* query generates a kNN graph. It computes the $kNN(o, O)$ result for every $o \in O$ and has a quadratic worst-case bound. An AkNN query can alternatively be viewed as a *kNN Self-Join*: *Given a dataset O and an integer k, the kNN Self-Join of O combines each object $o_a \in O$ with its k nearest neighbors from O, i.e.,* $O \bowtie_{kNN} O = \{(o_a, o_b) | o_a, o_b \in O \text{ and } o_b \in kNN(o_a, O)\}$.

  A real-world application based on such a query is *Rayzit* [1][3], our award-winning crowd messaging architecture, that connects users instantly to their *k Nearest Neighbors (kNN)* as they move in space. Similar to other social network applications (e.g., Twitter, Facebook), scalability is key in making *Rayzit* functional

[3]Rayzit: `https://rayzit.cs.ucy.ac.cy/`

and operational. Therefore we are challenged with the necessity to perform a fast computation of an AkNN query every few seconds in a scalable architecture. The wide availability of off-the-shelf, shared-nothing, cloud infrastructures brings a natural framework to cope with scalability, fault-tolerance and performance issues faced in processing AkNN queries. Only recently researchers have proposed algorithms for optimizing AkNN queries in such infrastructures [26, 27].

Solving the AkNN problem efficiently in a distributed fashion requires the object set O be partitioned into disjoint subsets O_i corresponding to m servers (i.e., $O = \bigcup_{1 \leq i \leq m} O_i$). To facilitate local computations on each server and ensure correctness of the global AkNN result, servers need to compute distances across borders for the objects that lie on opposite sides of the border and are close enough to each other. In any performance-driven distributed algorithm, the efficiency is determined predominantly by the network messaging cost (i.e., network I/O). Therefore, the goal is to minimize the number of objects transferred (replicated) between servers during the computation of the AkNN query.

Another factor in a distributed system is balancing the workload assigned to each computing node s_i, such that each s_i will require approximately the same time to compute the distances among objects.

RQ 4 ***How can we develop a scalable and high-performance distributed algorithm that solves the AkNN problem in a* fast batch *mode using a shared-nothing cloud infrastructure of* m *servers?***

Hypothesis: The points can be partitioned geographically in order to compute common kNN candidates per partition. This will allow the utilization of the distributed environment minimizing the network I/O and improving the overall performance.

1.3 Thesis Statement and Contributions

In this section, we provide the thesis statement and describe the main contributions of this thesis that provide answers to the research questions posed in Section 1.2.

1.3.1 Thesis Statement

To meet the response time and storage needs of SBD exploration tasks while efficiently utilizing system resources, a complete SBD analytic stack must have storage capabilities with compression and decaying, indexing with multiple levels of temporal and spatial resolutions and efficient spatial query operators.

1.3.2 Contributions

- We propose an innovative telco big data exploration stack, coined *SPATE*, whose objectives are two-fold: (i) minimizing the storage space needed to incrementally retain data *over time*; and (ii) minimizing the response time for spatiotemporal data exploration queries over recent data. We have measured the efficiency of our proposition using a real telco trace and a variety of telco-specific tasks, such as OLAP and OLTP querying, clustering, regression and privacy sanitizing, and showed that we can achieve comparable response times to the state-of-the-art with an order of magnitude less storage space. In the future, we aim to investigate a variety of advanced smart city application scenarios on top of *SPATE*, namely an automated car traffic mapping system and an emergency recovery system after natural disasters.

  Our architecture answers RQ 1 as it is described in Chapter 3.

- We present a novel decaying operator for Telco Big Data (TBD), coined *TBD-DP (Data Postdiction)*. *TBD-DP* relies on existing ML algorithms to abstract TBD into compact models that can be stored and queried when necessary. Our proposed *TBD-DP* operator has the following two conceptual phases: (i) in an offline phase, it utilizes a Long Short Term Memory (LSTM)-based hierarchical ML algorithm to learn a tree of models (coined *TBD-DP* tree) over time and space; (ii) in an online phase, it uses the *TBD-DP* tree to recover data within a certain accuracy. In our experimental setup, we measure the efficiency of the proposed operator using a ∼10GB anonymized real telco network trace. Our experimental results in Tensorflow over HDFS are extremely encouraging as they show that *TBD-DP* saves an order of magnitude storage space while maintaining a high accuracy on the recovered data.

  Our decay operator answers RQ 2 as it is described in Chapter 4.

- We develop techniques that generate the kNN graph of an arbitrary crowd of smartphone users that interconnect through short-range communication technologies, such as, Wi-Fi Direct, 3G/LTE direct and Bluetooth v4.0 (BLE). We present two efficient algorithms, namely *Akin+* and *Prox+*, optimized to work on a resource-limited mobile device. These algorithms partition the user space and compute shared candidate sets per partition. *Prox+* uses a custom heap data structure to update the candidate set as new users are inserted, whereas *Akin+* uses a bulk bottom-up construction of a simple heap to compute the candidate set once all users have been inserted. Our experiments verify the theoretical efficiency of the algorithms and shows that *Prox+* and *Akin+* are very well suited for large scale and skewed data scenarios.

  Our techniques answer RQ 3 and is described in Chapter 5.

- We present *Spitfire*, a scalable and high-performance distributed algorithm that solves the AkNN problem using a shared-nothing cloud infrastructure. Our algorithm offers several advantages over the state-of-the-art algorithms in terms of efficient partitioning, replication and refinement. Theoretical analysis and experimental evaluation show that *Spitfire* outperforms existing algorithms reported in recent literature, achieving scalability both on the number of users and on the number of k nearest neighbors.

  Spitfire algorithm answers to RQ 4 as it is described in and Chapter 6.

- We implement a prototype of SPATE using a modern SPARK-based processing architecture with HDFS and an RDBMS for catalog management (see Figure 7.1). The *SPATE UI (User Interface)* is implemented in HTML5/CSS3 along with extensive AngularJS. An illustrative network exploration interface is shown in Figure 4.5. We have implemented a query sidebar that allows the user to execute a variety of template queries. The query bar includes snapshot queries and recurring queries (in the form of a time-machine) for drop calls and downflux/upflux, heatmap statistics and settings. Furthermore, quick access buttons are provided so that users are able to choose between the available network modalities (2G, 3G, 4G).

- We present an innovative architecture for anonymous dynamic social networks, coined Rayzit, which enables anonymous interactions with the k nearest neig-

hbors. An industrial quality application was implemented that utilizes the aforementioned architecture. We presented and analyzed the data collected from this application and draw interesting conclusions about this new type of dynamic social networking. Our experimental results also confirm our initial hypothesis that the number of the replies is related to the location of the original rayz (i.e., the rayz with the most replies has a maximum range of 8 meters).

Our applications were build on top of our proposed architecture and are presented in Chapter 7.

1.4 Dissertation Outline

This thesis is organized into 9 chapters. A detailed dissertation outline of the thesis is presented below.

Chapter 1 presents a brief introduction, the motivation, the problem domains and the research questions, which are being researched in the context of this PhD thesis.

Chapter 2 describes the background and related work about the Big Data and Spatial Big Data architectures.

Chapter 3 introduces an innovative Telco Big Data exploration framework that aims to minimize the storage space needed to incrementally retain data over time and minimize the response time for spatiotemporal data exploration queries over recent data.

Chapter 4 presents a novel decaying operator for Telco Big Data (TBD) that aims to make a statement about the past value of some data, which does not exist anymore as it had to be deleted to free up disk space.

Chapter 5 proposes techniques that generate a *k-Nearest-Neighbor (kNN)* overlay graph of an arbitrary crowd that interconnects over some short-range communication technology.

Chapter 6 presents a distributed algorithm that provides a scalable and high-performance AkNN processing framework.

Chapter 7 describes an application for Telco Big Data analysis over the architecture described in Chapter 3 and a novel anonymous crowd messaging application, which utilizes the location of each user to connect them instantly to their k Nearest Neighbors (kNN) as they move in space.

Chapter 8 summarizes the contributions of our research and possible directions for future research.

Chapter 2

Related Work

2.1 Big Data Architectures

In this section we present several state of the art systems that cope with general big data, covering Query Processing, Resource Management and Real-time Systems (also referred to as interactive or online analytics).

2.1.1 Query Processing

In the recent years, the amount of generated data from a variety of sources is increasing with tremendous speed. The term big data is referring to data that grows exponentially. The processing and storing layer of the systems that can absorb this kind of data should be able to answer specific queries or predict the future of a business or even to extract important correlations between unrelated data. Nowadays the most common technique for processing and analyzing big data is to develop solutions in parallel and distributed processing frameworks, such as Hadoop or Spark.

Hadoop [28]: Hadoop is an open source cluster computing framework that allows the distributed processing of large data sets across clusters of computers using simple programming models. Hadoop has one or more master nodes that are running a name node service. In particularly, the master node maintains an index that describes where each chunk is placed in a Hadoop Distributed File System (HDFS). The remaining nodes in the cluster are data nodes that store the data in the cluster. MapReduce is the programming model of Hadoop. The jobs consist of two phases: the MAP and the REDUCE phase. During the MAP phase, the input is parsed and processed producing key-value pairs that will be forwarded to a specific reducer based on the key. During

the REDUCE phase an aggregation is applied on key-value incoming data and the output is produced.

Spark [29]: Spark is an open source cluster computing framework that has in-memory primitives, in contrast to Hadoop's disk-based MapReduce model, allowing to load data into a cluster's memory and query it repeatedly. Spark requires a cluster manager and a distributed storage system. Spark provides distributed task dispatching, scheduling, and basic I/O functionalities. The programming abstraction is called Resilient Distributed Datasets (RDDs), which is a logical collection of data partitioned across the cluster. RDDs can be created by referencing datasets in external storage systems, or by applying coarse-grained transformations (e.g., map, filter, reduce, join) on existing RDDs. Instead of specialized programming models for one type of application, Spark provides a first-class control of distributed datasets to the user.

MISO [30]: As the amount of data is increasing, new architectures are evolving in order to accommodate the huge amount of workload. Due to the variety of data, organizations have to maintain various storage systems to retain functionality. As a result, extracting and combining the data from different stores quickly becomes important in order to conclude meaningful results. The system that combines several storage systems is called a multistore system. Specifically, Jeff LeFerve et al. present their method, $MISO$, which describes how a multi-store processing query engine is constructed. Specifically, $MISO$ contains a HIVE[1] database for the heavy processing and a commercial warehouse management system (DBMS) for the lightweight filtering and querying.

epiC [31]: In this work the researchers developed a new system to achieve a high utilization of processing power over a distributed enviroment, coined *epic*, which is an extensible system that introduces a general actor-like concurrent programming model, independent of the data processing model in order to process the data in parallel. The authors state that their system outperforms Hadoop in terms of flexibility, performance and extensibility. The execution of the system is following the Pagerank algorithm [32] and the units are isolated and can process data independently. Each unit can send messages to other units through the master network.

Flink [33]: In data analysis, iterative algorithms are used in many occasions such as machine learning or graph analysis. As a result, in order to run those algorithms over very large data, it was necessary to utilize the distributed and parallel nature of a

[1]Hive: https://hive.apache.org/

cluster. A recent work of Ewen et al. presented Flink system that can achieve a high throughput and utilize all the components efficiently. Recently, Stratosphere became an Apache Incubator project under the name Apache Flink[2]. The purpose of the system is to provide the necessary programming interfaces and the programming model for writing programs in MapReduce fashion. It allows the developement of iterative and incremental algorithms instead of the traditional Hadoop MapReduce paradigm, using a unique data flow.

Spark SQL (Shark) [34]: At this point we can easily understand that there are a variety of big data processing systems and that these systems can be used to satisfy possible requirements according to the organization needs. One obstacle is that most of them provide complex query languages and require high programming skills to achieve a high performance response to each query. Armbrust et al. present a declarative SQL language in order to provide a simple way to construct jobs that can run in parallel, utilizing the infrastructure to answer a query over huge amounts of data. *Spark SQL* is an ancestor of Shark [35] over the Spark framework. Additionally, Spark code and Spark SQL can be combined in the same Spark job utilizing the *Catalyst Optimizer* [34] and switch from the well known *Resilient Distributed Datasets (RDDs)* to *DataFrames* utilizing the full potential of Spark cluster. As described by the authors, the performance results of the DataFrames are better than the traditional Spark template.

Initially, DBMS could not rely on main memory for all operations and transactions of a big data systems. As the price of main memory decreased and its memory size increased, new main memory architectures were established [36]. This allows higher performance by avoiding the costly network and disk I/O. RDDs provide a new more efficient way to utilize the main memory of a cluster. Particularly, most of the distributed systems including Hadoop are based on an acyclic data flow model. The records are loaded from the file system and forwarded to a *Directed Acyclic Graph* (DAG) of deterministic operators and then stored again into the file system. Zaharia et al. [29] support that RDDs are suitable for iterative algorithms and several optional features like caching the aforementioned RDDs into the worker memory retaining the scalability and the fault tolerance of the system. The authors present how the RDDs work on Spark and how they can avoid any possible insufficient memory problem. Specifically, if memory is not enough to cache the RDDs on each machine then the machine is forced to use less space and consequently the final performance is degraded.

[2]Flink: https://flink.apache.org/

It is important to understand the challenges in improving performance of big data analytics. As many researchers consider that the most costly factor is I/O, a recent work of Ousterhout et al. showed that the bottlenecks of big data processing systems are actually CPU and network performance [37]. In addition, the data layout of the query is considered to affect the network latency and CPU cost as the data could be either column or row oriented [38]. Sarma et al. showed how the upper and lower bounds can be computed in order to minimize the cost of a MapReduce job and how the trade-offs can be used [39]. The authors illustrated some bounding methods for several examples (e.g., Hamming distance, Triangle finding, Finding instances of other graphs) and calculated the upper and lower bounds for them.

MapReduce is a state-of-the-art framework as we previously mentioned but it is only used for offline analytics. In order to achieve high performance processing required for answering queries in an online manner time, MapReduce should pipeline the output instead of writing the intermediate parts to the disk as described in [40].

Big data processing is very important for organizations, especially for the telecommunication industry. Churn prediction is the biggest challenge for a telco organization due to the high dependencies on customer decisions [14]. Huang et al. presented telco churn prediction with big data providing a new big data platform for telco processing. In fact, the authors described a new architecture with storage and processing technologies such as Spark SQL, Hive and Hadoop. The telco dataset mainly consists of *Call Detailed Records* (CDRs) that contain users information like their calling number, data usage history, package type, cell tower ID, account balance, etc. and mobile broadband (MBB) data that keeps all the information from the network, specifically from the radio network controller (RNC).

Finally, it is necessary to present how the big data architectures can be evaluated and how the big data system can be benchmarked. BigBench was proposed from Ahmad Ghazal et al. in order to provide a data model and data generator that addresses velocity, variety and volume of structured, semi-structure and unstructured big data [41].

2.1.2 Resource Management and Storage Systems

As previously presented, several big data processing systems need various kind of resources (e.g., memory, disk storage, processing power, network throughput, etc.) in

order to operate at high performance rates on a parallel or distributed cluster environment. In this section, we present several mechanisms that allow the efficient utilization of underlying components of the infrastructure. Those mechanisms are called resource managers; a well known example is the Hadoop's resource manager coined *YARN*[3].

Mesos [42]: Hindman et al. present a resource manager, named *Mesos*, that can support several distributed processing frameworks at the same time. Actually, Mesos shares resources in order to allow frameworks to achieve data locality on each node. The authors illustrate how the Mesos architecture incorporates Hadoop and MPI (Message Passing Interface [43]) schedulers. Furthermore, Mesos achieves fault tolerance using ZooKeeper[4].

Mbal [44]: Adaptive load balancing and in-memory caching is the subject of a work presented by IBM Research [44]. Cheng et al. presented MBal, which is a high performance in-memory object caching framework with adaptive load balancing mechanisms. The article illustrates how Memcached[5] is used in order to cache objects across the cluster and how MBal overcomes the weaknesses of objects that have skew distribution. In addition, the authors provide a performance analysis of showing an important increase of throughput. As a result, the cost of developing applications on Amazon EC2 cluster is minimized in contrast with the Memchached technique. In addition, innovative techniques are trying to eliminate the bottleneck of the storage latency by constructing a memory layer that can be accessed like a distribute in-memory file system such as Tachyon [45].

2.1.3 Real-time Processing Systems

Nowadays, big data real-time processing is very critical for any business decisions and future plans for any organization [46]. We have already discussed various works based on Hadoop and MapReduce paradigm but these solutions are not capable to handle fast changing business environments. In this section, we will describe several real-time and near real-time platforms that can process and extract conclusions in a small amount of time.

Hadoop Online [40]: Hadoop was not able to cope with real-time processing due to the blocking behavior between mappers and reducers. Condie et al. proposed a

[3]YARN: https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

[4]ZooKeeper: https://zookeeper.apache.org/

[5]Memcached: https://memcached.org/

new solution coined Hadoop Online Prototype (HOP), which aims to pipeline the data between internal operators. The authors present how this modified MapReduce version can support continuous queries and adapt the existing MapReduce program without any changes.

OceanRT [13]: Zhang et al. proposed OceanRT, which is a real-time analytics system that can be used over large temporal data. The user can submit a HiveQL query at any node in the cluster managed by Zookeeper. The system has a Parsing Engine that is responsible to deliver the query to the Access Query Engine (AQE). It consists of a parser, an optimizer and a dispatcher. Furthermore, the AQE will decide how the query plan will be executed using Hive, HBase or a traditional RDBMS.

CEP [47]: CEP was proposed by Stojanovic et al. in order to solve real-time big data processing based on existing efficient infrastructures like Storm[6]. The authors describe that due to the high mobile sensing, data create the opportunity to extract important information that may be used in real-time applications. They describe a mobile system coined mCEP, which processes the real-time events on the mobile device. Mobile sensors are responsible to forward the event to the event stream manager, which sends an event to CEP engine or to the action handler or sends the events to the server pub/sub mechanism. It is important to notice that there is a real-time processing component, which can process the data in real-time and consists of two sub-components, the CEP engine and the situation analyzer. CEP engine is able to utilize the distributed enviroment to process the event in real-time. In addition, authors provide a detailed analysis of their system illustrating the semantics of the incoming data and how their CEP engine can process the data in a distributed manner and store them into a scalable storage, specifically RIAK[7] and 4store[8].

It is important to monitor these systems especially in scientific big data batch clusters due the high complexity of several jobs. Specifically, Kuehn at al. proposed a new tool that can achieve low performance overheads and does not affect the running jobs [48]. Monitoring the jobs running in the system is one of the most important factors to keep the cluster available and in an efficient mode.

AIM [49]: Big data often can satisfy one type of analytics either online transaction processing (OLTP) or online analytical processing (OLAP) analytics. Bruan et al.

[6]Storm: https://storm.apache.org/

[7]RIAK: https://basho.com/products/#riak

[8]4store: http://4store.org/

explain how a big data DBMS can satisfy both types of analytics. Particularly, the authors present AIM that was developed in order to be able to process mixed workloads and answer event stream processing and analytic queries in a short response time. The system can provide distributed execution like Storm and Hadoop in order to provide high scalability but it can also run as a single machine system. AIM consists of three layers, an event stream processing layer (ESP), a storage layer develop as a key-value store and a SQL query processing layer (RTA). In addition, the authors state that when more CDRs are imported into the system the administrator can simply add more nodes to the ESP layer. The storage subsystem adapts several in-memory techniques in order to provide a high performance and a small response time without affecting the consistency. Additionally, the storage system has a columnar storage layout based on partition attributes across (PAX) [50] in order to minimize the response time for analytic queries.

Bouillet et al. from IBM describe how 6 billion CDRs can be processed per day in real-time, as they are generated by the network machines [51]. The solution was developed on top of IBM's InfoSphere Streams middleware. The specific application is a mediation and analytics solution that is capable to consume all the CDRs stream producing real-time analytics. The system can scale up due to region wise parallelism and intra-Region parallelism. In addition, optimized in-memory lookups for database queries and bloom filter for duplication elimination boost the overall performance. The authors state that they used an optimistic fault tolerance and occasionally create checkpoints.

2.2 Spatial Big Data Architectures

In this section we will present the most recent SBD architectures. Moreover, we will present all the spatial operators that we described in the introduction including the implementation of our own k Nearest Neighbors (kNN) operator.

2.2.1 Spatial Big Data Query Processing

As we have already mentioned, geospatial application is at the center of big data research and industry due to the high sensing capabilities of mobile devices. In the previous chapter we have discussed about the big data processing challenges such as

network and disk I/O costs. After the partitioning, the objects can be placed into the minimum boundary regions (MBR) using Hilbert curve-ordering to preserve locality.

The same challenges are applied to the SBD processing with several extra requirements like a global spatial index that is needed to retrieve answers for a spatial query. Zhong et al. described how the spatio-temporal query processing is achieved efficiently and effectively over Hadoop [52]. Particularly, the authors have designed a geospatial data system that utilizes the locality of the data in the HDFS using a partitioning data model. Furthermore, they have provided a distributed indexing framework, which is used for pruning the search space and consequently minimizing the I/O cost. Additionally, they also provided a MapReduce algorithm that can improve the query computation efficiency along with a caching mechanism.

Asterix [53]: As we have already presented in the introduction, we are living in the "Big Data" era because data generation rate is extremely high especially in social networks, blogs, online communities, etc. As a consequence, new big data challenges derived based on the requirements against data management platforms. Particularly, the system should be scalable, flexible, manageable and have analytic capabilities as mentioned in [53]. The authors of this specific work, presented ASTERIX, which is a scalable warehouse with a web data integration style. The authors present their own data model and query language for describing, querying and analyzing data. According to the authors, ASTERIX is partitioning the data using LSM-based B+trees. In this specific system the authors adapt the Fuzzy matching in order to answer fuzzy selection queries and fuzzy join queries. In addition, the system provides an engine for executing spatial aggregation using a spatial grid technique.

OceanST [3]: OceanST was developed by Huawei [3] to manage huge volumes of telco big data. OceasST is a distributed solution based on Spark and MapReduce. It consists of 3 layers: (i) the storage layer where all the records and indexed and stored over HDFS; (ii) the functional layer where all the exact and approximate queries are provided through an API; and the application layer that incorporates the graphical user interface (GUI).

AQWA [54]: Aly at al. presented a system with an adaptive partitioning technique of SBD, called AQWA, and evaluated the performance using k nearest neighbor queries workloads. AQWA is incrementally updating the partitioning according to the workload. Particularly, AQWA has a k-d tree index on which the leaf node indicates the partition in the distributed file system along with a grid that keeps track of the number

of points existing in a region. As a result, the selected partitions can be passed into a MapReduce job minimizing the I/O cost. Finally, the partitioning can be changed based on the workload data and their distribution.

SBD processing has many challenges in terms of collecting, storing and analyzing the input data that may have a sparse distribution with spatial, temporal and network features [55]. Kelly et. al faced the aforementioned computational issue while analyzing call detail records (CDRs), which are data collected during mobile phone sessions. Their system currently consists of Hadoop, Hive and Spark clusters utilizing the HDFS and using R[9] for statistical analysis and graphic representation.

At this point, it is important to mention that user privacy should be maintained through the processing cycle especially in telco big data analytics. The information that are stored through the telecommunication organizations should be anonymized in order to keep the sensitive data private. A well-known technique is the k-anonymity model that guarantees a high level of anonymization and consequently a high level of privacy [56]. In contrast, Hu at al. presented Differential Privacy (DP) for the telco big data in order to avoid several attacks and weaknesses of the k-anonymity and several k-anonymity adaptations [57]. Particularly, the authors provided a detailed experimental and analytical evaluation of churn prediction accuracy over telco big data. By analyzing the results the authors concluded that the DP solution can be very restrictive for real data mining tasks, unless the privacy budget parameter ϵ is relaxed.

CellIQ [15]: CellIQ is a system that can provide real-time network data analytics and answer to spatial queries based on the collected data from the cellular network. Particularly, CellIQ is capable to detect and track spatio-temporal traffic hotspots and handoff sequences with abnormal failure sequences. This system takes advantage of the spatial and temporal locality to optimize the overall performance. The authors showed that their system outperforms the existing cellular systems by using the optimizations we have described. Iyer et al. have divided CellIQ into three operations, the sliding window operation, the time window operation and the spatial operation. Their solution is based on the Berkeley Data Analysis Stack (BDAS). BDAS includes several of the technologies that we previously mentioned like Spark, Tachyon, Mesos, Yarn, etc. The solution was written by using the RDDs and the GraphX API to incorporate specific optimizations like data placement, radius based message broadcast, spatial aggregation, differential graph update and incremental graph updates.

[9]https://www.r-project.org

SpatialHadoop [58]: Eldawy et al. created SpatialHadoop, which is an extension to Hadoop that adds spatial data awareness in each Hadoop phase. Particularly, SpatialHadoop adds a simple and expressive high level language for spatial data types and operations. In addition, SpatialHadoop adapts traditional spatial index structures such as Grid, R-tree and R+-tree, to form a two-level spatial index [59]. Furthermore, SpatialHadoop has several spatial operations, including range query, kNN and spatial join. The experimental evaluation shows that SpatialHadoop outperforms Hadoop for spatial data processing orders of magnitude. Particularly, the input file, which is a heap file, is passed to FileSplitter. The FileSplitter divides the file into n splits, where n is the number of the slave nodes in the cluster and then the parts are read from a RecordReader that extracts the key-value pairs to the mapper function. On the other hand, in SpatialHadoop the FileSpitter is replaced with a SpatialFileSplitter, which is an extended splitter that exploits the global indexes on spatially indexed input files to early prune file blocks. In addition, the record reader is replaced with a SpatialRecordReader, which reads the splits from the previous step and process them based on local indexes for a better performance during the map function.

The increasing rate of the data generated due to the continuous expanding of sensing capabilities affects the energy consumption of a mobile/sensor network. Data preservation in sensor networks with spatial correlation is one of the problems that Crary et al. addressed. They proposed a solution to minimize the energy and storage needed to preserve the big data inside the sensor network [60]. In addition, continuous queries over a sensor network drain energy, causing problems and failures to the sensor network. Andreou et al. proposed a solution to solve this problem, which involved the development of an energy-driven tree construction algorithm that can minimize the energy consumption and maximize the efficiency of the system [61].

2.3 Spatial Big Data Visualization

In this section we present how SBD systems can represent the processed result graphically. Specifically, we present systems that can show analytics over a map layer and a system that uses a GIS for medical imaging.

2.3.1 Spatial Visualization Systems

CyberGIS [62]: As we have already discussed, the spatio-temporal data are very important in order to extract results and critical information from the collected data [63]. In addition, when data contain geographical attributes it's necessary to provide a visualization of the data to the end users. Geographic Information Systems (GIS) have been providing this functionality over the years but with the increasing rate of the populated data, the existing solutions can't absorb the massive workload. Wang et al. proposed a new generation of GIS for the SBD to face the problem, called CyberGIS. CyberGIS can efficiently absorb and process the huge amount of data. Its noticeable that the CyberGIS is using the Hadoop Distributed File System (HDFS) to store the data and several of the parallel and distributed systems we have previously mentioned. In fact, the processing system is utilizing Hadoop, MPI, Spark, Shark and several in-situ algorithms in order to provide the end result to the users.

Hadoop-GIS [64]: Aji et al. state that SBD has common techniques with the systems previously mentioned and the proposed solution, which is based on Hadoop, is called $Hadoop - GIS$. Hadoop-GIS takes advantage of the global partition indexing and on-demand spatial indexes to answer spatial queries efficiently. The query engine of Hadoop-GIS is based on RESQUE, which we have described in the previous section. Hadoop-GIS spatially partitions the data into buckets (or tiles) that are forwarded to MapReduce in order to be processed in parallel. Then the result tiles are merged and stored in HDFS. Moreover, spatial global and local indexes are created in order to extract the requested features and then aggregate them to the final results stored in HDFS.

In Table 2.1 we compare the SPATE$^+$ and the systems from the literature based on their ability to process big data, answer spatial queries, respond in real-time, provide visualization methods, using compression and decay techniques. These parameters were chosen in order to show the historical evolution of the systems.

System	Big Data	Spatial	Real-time	Visualization	Compression	Decay
MISO [30]	YES	NO	NO	NO	NO	NO
epiC [31]	YES	NO	NO	NO	NO	NO
Stratosphere [33]	YES	NO	NO	NO	NO	NO
Spark SQL [34]	YES	NO	YES*	NO	NO	NO
MapReduce Online [40]	YES	NO	YES	NO	NO	NO
OceanRT [13]	YES	NO	YES	NO	NO	NO
CEP [47]	YES	NO	YES	NO	NO	NO
AIM [49]	YES	NO	YES	NO	NO	NO
ASTERIX [53]	YES	YES	NO	YES	NO	NO
AQWA [54]	YES	YES	NO	NO	NO	NO
CellIQ [15]	YES	YES	YES	NO	NO	NO
SpatialHadoop [58]	YES	YES	NO	YES	NO	NO
CyberGIS [62]	YES	YES	NO	YES	NO	NO
Hadoop-GIS [64]	YES	YES	NO	YES	NO	NO
SPATE^{+} [17, 65]	YES	YES	YES	YES	YES	YES

Table 2.1: Big data and SBD architectures

Chapter 3

Efficient Exploration of Telco Big Data with Compression and Decaying

In the realm of smart cities, telecommunication companies (telcos) are expected to play a protagonistic role as they can capture a variety of natural phenomena on an ongoing basis, e.g., traffic in a city, mobility patterns for emergency response or city planning. The key challenges for telcos in this era is to ingest in the most compact manner huge amounts of network logs, perform big data exploration and analytics on the generated data within a tolerable elapsed time. This chapter introduces *SPATE*, an innovative telco big data exploration framework whose objectives are two-fold: (i) minimize the storage space needed to incrementally retain data over time; and (ii) minimize the response time for spatiotemporal data exploration queries over recent data. The storage layer of our framework uses lossless data *compression* to ingest recent streams of telco big data in the most compact manner retaining full resolution for data exploration tasks. The indexing layer of our system then takes care of the progressive loss of detail in information, coined *decaying*, as data ages with time. The exploration layer provides visual means to explore the generated spatio-temporal information space. We measure the efficiency of the proposed framework using a 5GB anonymized real telco network trace and a variety of telco-specific tasks, such as OLAP and OLTP querying, privacy-aware data sharing, multivariate statistics, clustering and regression. We show that out framework can achieve comparable response times to the state-of-the-art using an order of magnitude less storage space.

3.1 Introduction

Unprecedented amounts and variety of spatiotemporal *big* data are generated every few minutes by the infrastructure of a *telecommunication company (telco)*. The rapid expansion of broadband mobile networks, the pervasiveness of smartphones, and the introduction of dedicated Narrow Band connections for smart devices and Internet of Things (NB-IoT) [66] have contributed to this explosion. An early example of the data volume and velocity of telco big data is described in [3], where a telco collects 5TBs per day, i.e., almost 2PBs per year, in a single city of 10M cell phone customers.

Data exploration queries over big telco data are of great interest to both the telco operators and the smart city enablers (e.g., municipalities, public services, startups, authorities, and companies), as these allow for interactive analysis at various granularities, narrowing it down for a variety of tasks including: network plan optimization and user experience evaluation, precise marketing, emergency response, urban planning and new urban services [3, 13–15, 67]. Data exploration and visualization might be the most important tools in the big data era [68–70], where decision support makers, ranging from CEOs to front-line support engineers, aim to draw valuable insights and conclusions visually.

One key challenge in this new era of telco big data is to *minimize the storage costs* associated with the data exploration tasks, as big data traces and computed indexes can have a tremendous storage and I/O footprint on the data centers of telcos. Although the volume of electronically stored data doubles every year, storage capacity costs decline only at a rate of less than 1/5 per year [18]. Storing big data locally, due to the sensitive nature of data that cannot reside on public cloud storage, adds great challenges and costs that reach beyond the simplistic capacity cost calculated per GB [19]. From a telco's perspective, the requirement is to: (i) *incrementally store big data in the most compact manner*, and (ii) *improve the response time for data exploration queries*. These two objectives are naturally conflicting, as conjectured in [20].

In this chapter we present *SPATE*[1], a SPAtio-TEmporal framework that uses both lossless data *compression* and lossy data *decaying* to ingest large quantities of telco big data in the most compact manner. *Compression* refers to the encoding of data using fewer bits than the original representation and is important as it shifts the resource

[1]SPATE: "*a large number of things that appear or happen in a short period of time" (Merriam-Webster dictionary)*

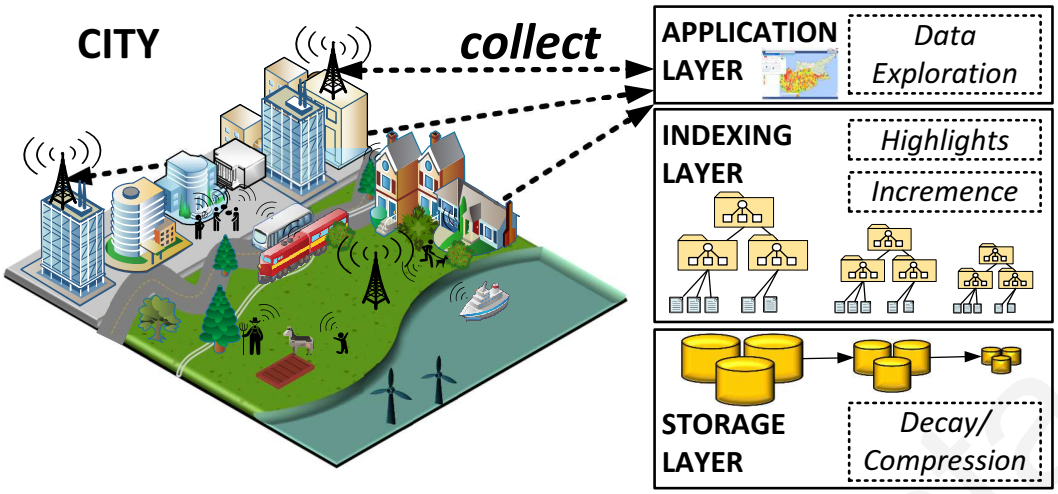

Figure 3.1: SPATE *is an efficient telco big data exploration framework that deploys compression, decaying and exploration of the collected data.*

bottlenecks from storage- and network-I/O to CPU, whose cycles are increasing at a much faster pace [71–73]. It also enables data exploration tasks to retain full resolution over the most important collected data. *Decaying* on the other hand, as suggested in [74], refers to the progressive loss of detail in information as data ages with time until it has completely disappeared (the schema of the database does not decay [75]). This enables data exploration tasks to retain high-level data exploration capabilities for predefined aggregations, without consuming enormous amounts of storage.

SPATE can be regarded as a domain-specific streaming data warehouse, which is divided into the following layers: (i) the *Storage layer*, which passes newly arrived network streams (coined *snapshots*), arriving with a periodic clock, through a lossless compression process that stores the results on a replicated big data file system for availability and performance; (ii) the *Indexing layer*, which provides the structures for efficient data exploration but also invokes the decaying process. Particularly, it is responsible for storing the upcoming snapshots on disk with the incremence module, for identifying interesting event summaries with the highlights module and for decaying the oldest leaf nodes of the index, i.e., by pruning-off parts of the tree index using a provided decaying function (i.e., *data fungus*). *Indexing* is the standard mechanism to speed up queries in incremental spatio-temporal querying and visualization systems [3,13,15,70]; and (iii) the *Application Layer*, which holds components responsible for processing user queries and presenting results through a spatio-temporal visual user interface and a declarative SQL user interface.

We evaluate the performance of the *SPATE* framework using a 5GB anonymized

real telco big data trace, whose structure is explained in the next section. To show the utility of *SPATE*, we carry out a variety of telco-specific querying tasks, such as OLAP and OLTP querying, clustering, regression and privacy sanitization. Our results indicate that *SPATE* requires sub-linear storage space with respect to the amount of data ingested, an update time of only a few seconds without affecting the online data, and a data exploration response time that is independent of the queried temporal window. *SPATE* achieves similar query response times to state-of-the-art solutions [70], but using only a fraction of the data storage space.

There is no prior work that studies *data decay* and efficient *data exploration* for telco big data in combination. In previous work, custom data management systems have been designed with the objectives to save storage space using compression, and speed up temporal range queries using indices [76–79]. None of these considers the notion of "decay" as expressed in [74], which suggests sacrificing either accuracy or read efficiency for less frequently accessed data to save space. Furthermore, these solutions are tailored specifically for managing scientific (floating point) data.

In contrast, we focus on (i) compressing and decaying incremental telco big data, which mostly contains string and integer values; (ii) on spatiotemporal data exploration queries in a telco setting, and (iii) we also develop our solution on top of off-the-shelf open source systems (e.g., Hadoop, Spark) that are widely used in industry, with low installation, administration and maintenance costs.

Our contributions can be summarized as follows:

- We apply efficient compression algorithms in order to reduce the storage space and minimally affect query response time to exploratory search queries in a telco big data setting.
- We introduce a multi-resolution spatio-temporal index that supports the notion of data decaying.
- We provide an extensive experimental evaluation using a variety of telco-specific tasks to show the benefits of our approach.

3.2 Preliminaries

In this section we describe the special characteristics of telco big data and the telco network that produces them.

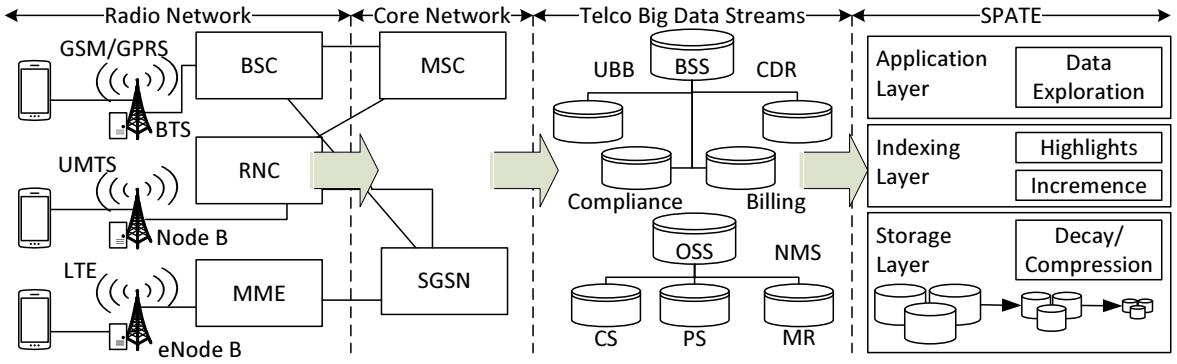

Figure 3.2: The anatomy of a typical telco network architecture that generates telco big data streams consumed by SPATE.

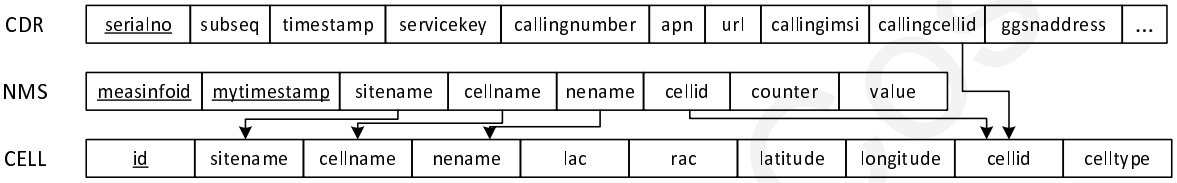

Figure 3.3: The relational schema of the telco's data shows the first 10 out of ∼200 attributes of the CDR data and all the attributes of the NMS and CELL data.

3.2.1 The Anatomy of a Telco Network

A telco network consists of the Radio and the Core Network, as shown in Figure 3.2. A Radio Network operates in three different modes, namely GSM/GPRS, UMTS and LTE. The Global System for Mobile Communications (GSM) is the standard developed for voice communications in a digital circuit-switched cellular network (2G). To allow for data communication, the GSM standard expanded over time to become a packet-switched network via General Packet Radio Service (GPRS). The Universal Mobile Telecommunications System (UMTS) extends GPRS to deal with increased data needs and constitutes the third generation (3G) mobile cellular system that moves toward an all-IP network. Finally, the Long-Term Evolution (LTE) standard was developed to increase the capacity and speed using a different radio interface together with Core Network improvements. It is sometimes called the fourth generation (4G) mobile cellular system. Overall, all three modes interleave one another to offer the best possible coverage to the users. In the future, fifth generation (5G) networks are expected to improve the radio coverage, capacity, data rate and latency with technologies like femtocells, millimeter waves, massive MIMO, beamforming and full duplex [80], i.e., mainly advancing the Radio Network.

The GSM operation is supported by base stations called Base Transceiver Stations (BTS), which are controlled by Base Station Controllers (BSC). Each BTS is equipped with transceivers for transmitting and receiving radio signals, antennas and components for encrypting and decrypting the messages using the BSC. The BSC controls hundreds of BTSs and is responsible for the allocation of radio channels, receives measurements from the mobile phones, and controls hand-overs from BTS to BTS. It ultimately combines the multiple low-capacity connections of its BTSs into combined "virtual" connections that are sent over to the Mobile Switching Center (MSC) in the Core Network. Finally, it provides data to the *Operation Support Subsystem (OSS)*, whose data will be described extensively in the next subsection.

The UMTS operation is supported by base stations, called Node B, which are controlled by Radio Network Controllers (RNC). The RNC provides similar functions to that of BSC, only for the UMTS network. RNC connects to the circuit-switched Core Network through the Serving GPRS Support Node (SGSN). The RNC also provides data to the OSS. The LTE operation is supported by base stations, called eNode B, which can directly connect to the core network. The Mobility Management Entity (MME) authenticates the wireless devices connected to eNode Bs and is involved in hand-offs between LTE and previous standards.

The data generated by the network can be considered as *data streams*, which are aggregated continuously on the telco's data center for operational purposes but also replicated to an exploration and visualization system like SPATE, for efficient indexing, querying, visual exploration and analysis.

3.2.2 The Structure of Telco Big Data

Typical *telco big data* streams [14] consist of: (i) *Business Supporting Systems (BSS)* data, which are used to run the telco business operations related to customers (e.g., orders, payment issues, revenues). BSS are associated with the following specific types of data and stored in conventional SQL databases: User Base Behavior (UBB) records, compliance records, billing records, as well as voice/message *Call Detailed Records (CDR). BSS data has only a limited volume of around 24GB per day (for a 2M+ clientele of a China telco) [14]* and were widespread within telco operational and analytical IT infrastructure even before the big data era; and (ii) *Operation Supporting Systems (OSS)* data, which is generated by the telco's computer systems used to

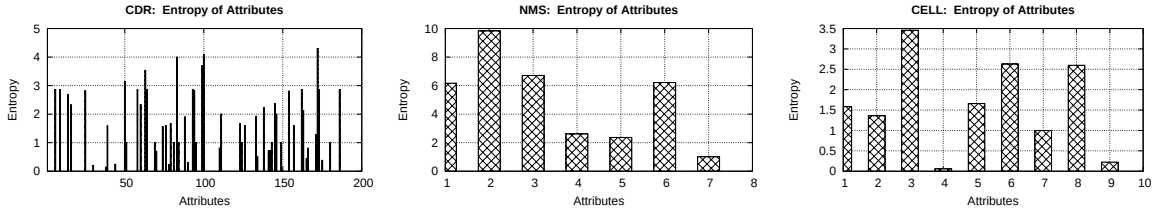

Figure 3.4: The entropy of each attribute in (Left) CDR data, (Center) NMS data, and (Right) CELL data.

manage its networks and stored in NoSQL big data stores. OSS comprises of data in the following parts: Circuit Switch (CS), Packet Switch (PS) - often referred as *Mobile BroadBand (MBB)* data - and Measurement Report (MR). CS data describe call connection quality, PS data describe users' web behavior (e.g., web speed, connection success rate) and MR includes a variety of measurement reports (e.g., for estimating user location [21]). By analyzing OSS one can observe important call connection quality statistics and user experience indicators. As a representative source of OSS, we use *Network Measurement System (NMS)* reports that contain counters for call drop rates, call duration measurements, antenna throughput. *OSS data has a volume of around 2.2TB per day, occupying over 97% data volume of the entire dataset [14].*

A CDR contains only metadata with fields that describe a specific telecommunication session (i.e., transaction), but does not contain the actual content of that transaction. For example, a CDR describing a phone call may contain the phone numbers of the calling and receiving party, the start time, end time, call type, and duration of that call. CDR is the primitive data source for customer billing purposes. On the other hand, the OSS logs contain aggregated performance counters for the three different network types. In general, the data is highly structured and comprises of relational records based on a predetermined schema with a large number (~ 200) of attributes that take mostly nominal text and interval-scaled discrete numerical values (see Figure 3.3).

We analyzed a real anonymized dataset in an effort to understand what compression ratios can be achieved. The time duration of the dataset was 1 week and it consisted of about 1.7M CDR and 21M NMS records coming from approximately 300K users. The total size of the dataset was 5GB. Based on Shannon's source coding theorem [81], the minimum number of bits needed to express a symbol given a set of possible symbols S and their probabilities P is $-\sum_{p_i \in P} \log_2 p_i$, and the maximum compression ratio possible is inversely proportional to the entropy $H = -\sum_{p_i \in P} p_i \log_2 p_i$ of the data. In

Figure 3.4, we plot the entropy of the attributes included in three different file types that arrive at a telco data center. Looking at the first plot that corresponds to CDR data files, it immediately stands out that most attributes have an entropy smaller than 1 and some even have an entropy of 0. This confirms that high compression ratios can be achieved. The attributes that have 0 entropy are optional attributes that usually are left blank.

The data arrives at the data center in batches, called henceforth *snapshot* data noted by d_i, in the form of horizontally segmented files every 30 minutes, a period we call *ingestion cycle* or *epoch*. A snapshot d_i contains records of user activity (e.g., phone call completion) or network activity that took place at some timepoint within the corresponding ingestion cycle, and records that express aggregate values over the same ingestion cycle (e.g., call drops, throughput). Each snapshot d_i can be seen as a table of records (rows) with a predefined set of attributes (columns).

Regarding the spatial information inherent within the telco network data, every record is linked to a specific cell ID c. Each cell ID corresponds to a cell that covers a specific area a_c and is attached to a base station that has a known location. Therefore, we can not talk about spatial data in the traditional sense, as each record is only associated with a specific geographical cellular area a_c that usually spans hundreds of meters and estimating user location, as in [21], is outside the scope of this work.

3.3 SPATE: Overview

In this section we provide an overview of the *SPATE* architecture, which consists of three layers (see Figure 3.1): the storage, the indexing and the application layer. We start out with a problem formulation and then outline our solution.

3.3.1 Problem Formulation

Given a telco setting, where telco big data arrives periodically in batches, we want to: (i) minimize the space needed to store/archive data; and at the same time (ii) minimize response time for spatiotemporal data exploration queries and tasks.

Given storage space S needed for storing data before any compression is performed, storage space S_c needed for the data after compression and storage space S_i needed for storing the access method information (e.g., index), we can measure the ratio of

contribution towards the first objective as $O_1 = S/S'$, where $S' = S_c + S_i$. Similarly, given the query response time T over the uncompressed data and the query response time T_{ci} over compressed and indexed data, we can measure the ratio of contribution towards the second objective of this work as $O_2 = T/T_{ci}$.

3.3.2 Our Solution

We express our solution in three layers overviewed next and described in detail in the following sections:

Storage Layer implements the compression logic in *SPATE*. The *Storage layer* passes newly arrived network snapshots through a lossless compression process storing the results on a replicated big data file system for availability and performance. This component is responsible for minimizing the required storage space with minimal overhead on the query response time. The intuition is to use compression techniques that yield high compression ratios but at the same time guarantee small decompression times. We particularly use GZIP compression that offers high compression/decompression speeds, with a high compression ratio and has maximum compatibility with I/O stream libraries in the big data ecosystem we use. The storage layer is basically only responsible for the leaf pages of the *SPATE* index described in the next layer.

Indexing Layer is responsible for minimizing the query response time for data exploration queries. It maintains and uses a multi-resolution spatiotemporal index that consists of the *Incremence* module and the *Highlights* module. The Incremence module receives the newly arriving snapshot d_i and incorporates it into the index by incrementing it on the right-most path. The Highlights module combines data from the stored snapshots to create efficient representations of interesting events, called "highlights". These highlights are constructed for each day, month and year and the end of each such period, respectively. The highlights of a month are constructed from the daily highlights, and the highlights of a year are constructed from the monthly highlights. Finally, this layer is also responsible for the gradual decay of the data. It does so by pruning-off parts of the tree index by using the notion of data fungus we will explain.

Application Layer implements the querying module and the user interface. In our case, we present the *Data Exploration* module, which receives a data exploration query $Q(a, b, w)$ and based on a, b, and w uses the index to combine the needed highlights to answer the query. Finally, *SPATE* is equipped with an easy-to-use map-based web interface that hides the complexity of the system and accesses all *SPATE* functionality. Details of the web interface are described in Section 3.6.

3.4 SPATE: Storage (Compression) Layer

This section describes the lowest layer of the *SPATE* framework that relates to storage. The *SPATE* storage layer takes care of compressing snapshots of telco streams as they arrive periodically. In our setting, exploratory queries need to be able to perform exact queries over recent data, therefore our compression mechanisms have to be *lossless*. In the following subsections we provide basic compression terminology and the desiderata of our approach. We then provide a qualitative description of various lossless compression libraries that are compared against in a microbenchmark that follows.

3.4.1 Terminology and Desiderata

We start out with some basic terminology and then provide our objectives. Given a lossless compression codec c and a dataset d that occupies space S, the codec can compress d into S_c space in *compression time* T_{c1}. The *compression ratio*, which quantifies the reduction in data size produced by a data compression algorithm, is defined by $r_c = S/S_c$ and depends on both c and d. Finally, c can decompress d back to its original state in *decompression time* T_{c2}.

The first objective of *SPATE* relates to saving space. Particularly, the compression mechanism needs to achieve a high *compression ratio*. On the other hand, the query response time needs to be kept low (second objective), therefore the compression mechanism needs to have a small *decompression time*, since this overhead will be paid for every query. It is important to notice that our approach is not particularly concerned with the *compression time*, as the compression cost is only paid for a single snapshot in each round. One final argument of concern is compatibility with existing stream readers. For example, GZIP [82] is widely supported by various environments and its usage provides maximum portability.

3.4.2 Lossless Compression Libraries

In this section we overview some traditional and some emerging big data lossless compression *libraries*, which can be linked directly to existing big data processing software like *SPATE*. These libraries also have respective standalone command-line tools.

- **GZIP [82]:** is a traditional file format and a software library used for file compression and decompression. It is based on the DEFLATE algorithm that uses a combination of Lempel-Ziv coding (LZ77) [83] and Huffman coding. In LZ77, repeated occurrences (in a look-ahead buffer) are replaced with pointers to a recently encoded sequence (sliding window buffer). In this sense, it is a sequential data compression technique with a dictionary that is constructed during the encoding process. On the contrary, in Huffman coding, a one-to-one symbol-to-code map is constructed based on occurrence probabilities of symbols in a learning corpus (i.e., entropy-based). The DEFLATE algorithm, defined through RFC1951, is then a hybrid algorithm that consists of a series of blocks encoded in either LZ77 or Huffman and preceded by a respective header.

- **7z [84]:** is another traditional dictionary-based compression tool and library based on the LZMA and LZMA2 algorithms. Like GZIP, it aims to build a good statistical model or "dictionary" for the input data upon which bit sequences of frequently encountered data can be compacted more densely.

- **SNAPPY [85]:** is a modern open-source compression and decompression library by Google that aims for maximum compression speed as opposed to maximum compression ratios. In SNAPPY, the compressed files are reported to be 20% to 100% bigger than other compression tools but compression and decompression speeds are reported to be faster due to an optimized implementation. The library has been used extensively by Google in its BigTable, MapReduce and internal RPC systems.

- **ZSTD [86]:** is another modern open-source lossless compression/decompression library developed by Facebook, which targets real-time compression scenarios. ZSTD (aka Zstandard) uses new generation entropy coders HUFF0 (Huffman) and FSE (Fine State Entropy), which are designed to perform well on modern CPUs and belong to the Asymmetric Numeral Systems (ANS) family of entropy

Table 3.1: Lossless Compression with Different Libraries in SPATE (Average results per 30-min snapshot)

Objectives \ Libraries	GZIP	7z	SNAPPY	ZSTD
Compress. Ratio (r_c)	9.06	11.75	4.94	9.72
Compress. Time (T_{c1}) in sec	21.37	20.99	21.39	21.07
Decompress. Time (T_{c2}) in sec	0.11	0.12	0.13	0.11

algorithms. Compared to prior tools, ZSTD allows building domain-specific training dictionaries.

3.4.3 Microbenchmark

Our objective in this subsection is to empirically assess the presented compression libraries as part of the *SPATE* storage layer, which stores snapshots in a directory hierarchy. Our dataset includes 200 snapshots from the 5GB anonymized and uncompressed telco dataset that comprises of 1.7M CDR and 21M NMS records. Our microbenchmark is performed on top of an HDFS v2.5.2 filesystem (more details about the testbed are provided in Section 3.7). We particularly focus on the three common metrics: *compression ratio* r_c, *compression time* T_{c1} and *decompression time* T_{c2}.

Table 3.1 shows the results of our evaluation. Our first observation is that r_c is similarly satisfactory for GZIP, 7z and ZSTD. On the other hand, SNAPPY shows that its r_c is only half as good as the rest of the libraries so it might not be a good alternative for *SPATE*. The T_{c1} and T_{c2} results relate to compression and decompression time, respectively, for a single snapshot and are measured in seconds. Looking at these numbers we clearly observe that T_{c1} takes always more time than T_{c2}, which is very typical for compression algorithms. Looking at the costs more carefully, we observe that the T_{c1}/T_{c2} ratio is about 200 instead of the more typical 2. This is attributed to the fact that *SPATE* performs many additional CPU-bound functions in each compression round, such as parsing. Finally, SNAPPY does not expose any speed benefits overall, as the slow I/O is hiding its benefits.

As a conclusion, we denote that the *SPATE* storage layer can operate with a variety of libraries, each of them coming with different performance trade-offs. In our

implementation and evaluation, we chose the GZIP library, which was readily available from within the `java.util.zip` core libraries and was also supported readily by certain parts of the application layer described in Section 3.6.

3.5 SPATE: Indexing (Decaying) Layer

The *Indexing layer* provides the structures for efficient data exploration but also invokes the decaying process. Particularly, it is responsible for augmenting the upcoming snapshots on disk with the incremence module, identifying interesting event summaries with the highlights module and decaying the oldest leaf nodes of the index, i.e., by pruning-off parts of the tree index using the so called data fungus. In the remainder of this section, we outline the three modules of the *SPATE* indexing layer.

3.5.1 Incremence Module

This module is responsible for the incremental construction of a multi-resolution spatio-temporal index as data snapshots are ingested by *SPATE*. Our index has 4 levels of temporal resolutions (i.e., epoch (30 minutes), day, month, year), with each leaf level containing 2 spatial dimensions (x,y) and N additional domain-specific dimensions (e.g., CDR and NMS).

Figure 3.5 shows an example index in SPATE. As we can see, the root node points to year-nodes, each representing a single year. Each year node points to 12 month-nodes, each representing a single month. Similarly, the month nodes point to their corresponding day-nodes, and each day node points to its corresponding 48 snapshot leaves.

Every time a new snapshot arrives, it is compressed by the storage layer and then the temporal index is incremented on its right-most path. If the new snapshot belongs to an incomplete day, it is just added as a leaf under the existing right-most day-node. Else, we first need to add a new dummy day-node. If this new day is the first day of a new month, we also need to add a new dummy month-node. Similarly, if the new month is the first month of a new year, we first need to add a new dummy year-node.

Each leaf node could store an additional spatial index (e.g., R-tree or quad-tree variant) to speed up data exploration queries within a snapshot. For example, a query like: *"find the aggregates regarding some object of interest in a given spatial bounding*

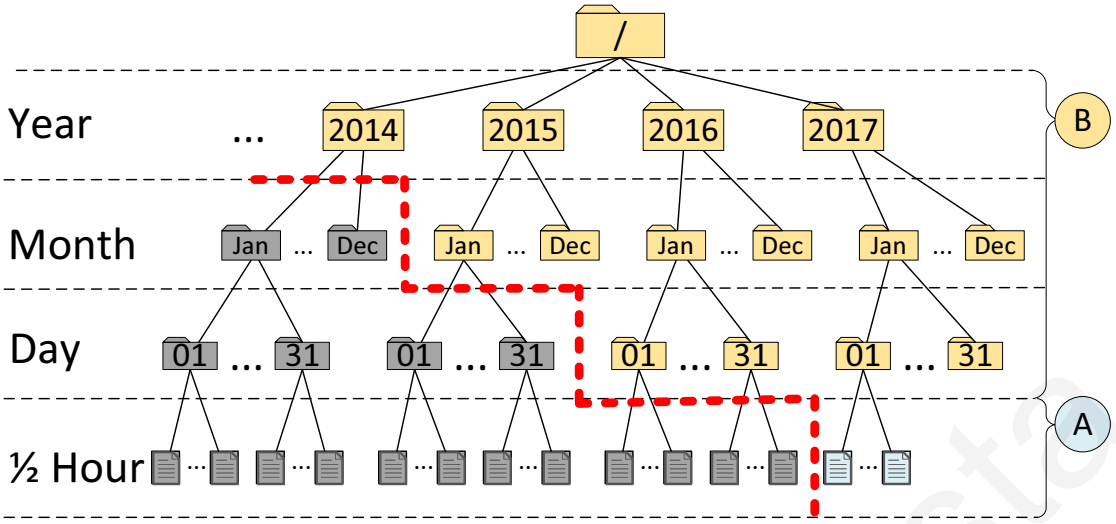

Figure 3.5: The multi-resolution spatio-temporal index in SPATE. *Our index has 4 levels of temporal resolutions with each leaf level containing 2 spatial dimensions (x,y) and N additional domain-specific dimensions (e.g., related to CDR and NMS). The red line denotes the decaying data fungus that evicts progressively the oldest leaf and non-leaf nodes of the tree.*

rectangle for a specific time range" could benefit from such an embedded spatial index after reaching the leaf level of the index. Such an index would allow to quickly scan the attributes stored per snapshot. However, snapshots are usually not very large (i.e., have a 30 min timespan), thus an additional index would only provide modest additional query response time benefits at the price of additional storage space that we aim to minimize.

3.5.2 Highlights Module

To enable interactive data exploration we compute "highlights" from the underlying raw data for each internal node of the temporal index structure. Such highlights are effectively materialized views to long-standing queries of users (e.g., the drop-call counters, bandwidth statistics), which are executed in a periodic manner as the snapshots stream to *SPATE*. In this sense, the highlights can be perceived as an OLAP cube whose construction cost is amortized over time. Building the highlights cube enables the application layer, we will describe next, to swiftly go over the generated statistics for visualization purposes. Consequently, users can drill down or roll up to the desired aggregates without additional delays.

Below we outline the operation of the highlights module: At the end of each day, when all the snapshots of that day have been added as leaves, the highlights of that

day are calculated from the compressed data of its snapshots, and are stored in the day-node. They are also forwarded to the parent month-node, which increments its own monthly highlights. Similarly, at the end of each month/year, the highlights of that month/year are calculated from the highlights of its days/months and are stored in the month/year-node and forwarded to its parent node, which in-turn increments its own highlights. This way the root will store the highlights of all the completed years.

The computation is based on the frequency of occurrence of a value in the data. Frequent values with an occurrence frequency above threshold θ are treated as no-highlights, whereas values with an occurrence frequency below threshold θ are considered highlights. A highlight is described by its type (in case of categorical data) or its peaking point (in case of continuous numerical values) and its duration. It is important to observe that for each level of resolution (day, month, year) a separate frequency threshold θ_i can be used, e.g., lower thresholds for higher levels resolution.

3.5.3 Decaying Module

The last module of the indexing layer deals with decaying of compressed snapshot data and aggregated highlights. *Decaying* refers to the progressive loss of detail in information as data ages with time until it has completely disappeared. Kersten refers to the existence of data fungus in [74], e.g., *"Evict Grouped Individuals"*, which helps in the decaying processing. In our work, we chose a data fungus, we coin *"Evict Oldest Individuals"*, as it helps us to deal more pragmatically with telco network signals, where more recent signals contain more important operational value that needs to be retained fully.

We particularly devise a scheme where operators chose the rate at which the temporal decaying policy becomes effective. The red line in Figure 3.5, denotes one hypothetical such policy that aims to retain up to one year of data exploration with full resolution along with yearly progressive decay. This policy is translated into a continuous decaying process where leaf and non-leaf entries of the spatio-temporal index are purged from replicated storage in a sliding window manner.

The result of the decaying process is that the data exploration warehouse can retain the highest possible data exploration resolution for predefined aggregate queries over extremely long time windows without consuming enormous amounts of storage. Otherwise, the storage overheads would soon enforce administrators to delete large

quantities of telco big data traces, purging at the same time the hope to learn valuable insights from big data at the macroscopic scale.

3.5.4 Indexing Schemes Comparison

OceanST [3] uses a 3-level partitioning scheme. The first level, named bucket level, partitions tuples according to the hash values of user-id as well as provides coarse ranges of time. The second level, named region layer, creates a spatial index on location (long, lat). The third level, named block layer, partitions the previous level in finer ranges of time and actual data in 64MB blocks (i.e., HDFS v1 block sizes). There are two additional data structures to support advanced and approximate queries. The in-block index aims to refine the spatiotemporal granularity of the index structure. The inverted index indicates the index leaf nodes, which are 3-dimensional quadtree associated with the attribute value of interest. On the other hand, SHAHED [70] employs a multi-resolution spatio-temporal index for efficient data indexing and retrieval. The indexing process is taking place at the end of the uncertainty module, where the new cleaned data is added to the current spatio-temporal index structure. Periodic monthly and yearly execution compacts the index structures for efficiency. The index consists of two modes: (i) temporal mode, where the data is organized in three different temporal resolutions; and (ii) spatial mode, where the data is stored in the form of an aggregate quadtrees. The leaf level of the spatial index partition is divided spatially using a uniform grid.

SPATE indexing scheme deploys a multi-resolution spatio-temporal index with compression (described in Section 3.5.1) and decaying (described in Section 3.5.3) at the leaf levels that none of the competing approaches provide. Additionally, SPATE provides a multi-granular highlight component that allows interactive data exploration queries (described in Section 3.5.2).

3.6 SPATE: Application Layer

In this section we present the Application layer that holds components responsible for receiving user queries and presenting the results. These are implemented in the query exploration interfaces, according to a query evaluation process we outline.

3.6.1 Query Evaluation and Processing

This module receives a data exploration query and accesses the index maintained by the Indexing layer of *SPATE*.

In our setup, a data exploration query $Q(a, b, w)$ consists of an attribute selection a, a spatial bounding box b, and a temporal window of interest w. A b can cover an area a few hundreds of square meters up to several hundreds of square kilometers. Similarly, a temporal window can span a few hours to several months or even years. A query $Q(a, b, w)$ can be expressed as *"Explore the values of a within the spatial box b and temporal window w"*. Such queries have direct applicability to interactive visualization tools used for data exploration that present data overlaying a geographical map. The users zoom into the map (thus defining the b) and select the attributes (a) and the time period (w), for which they would like to observe the query results as snapshots or as a video (i.e., "playback highlights in fast-forward").

Given a data exploration query $Q(a, b, w)$ the index is accessed to find the temporal node whose period completely covers w. For example, if the temporal window is from 15 September 2016 until 15 October 2016, the index is accessed up to the year level and the highlights of year-node 2016 are retrieved. Once the correct temporal node is found, all highlight summaries or actual available data whose spatial bounds completely cover b are then retrieved.

SPATE might retrieve records for a larger period than the one requested. However, this is not a problem given that users very often like to have a quick glance to the period before and after the chosen window. In this case, our decision to retrieve a larger period serves as an implicit prefetching mechanism. When users decide to focus on a smaller window within w, it is considered as a data exploration query $Q(a, b, w')$ with $|w'| < |w|$, which can be served directly from the cache of the user interface that is explained in the Chapter 7.

3.7 Experimental Testbed and Methodology

To validate our proposed ideas and evaluate *SPATE*, we have implemented a trace-driven experimental testbed on which we conducted a comprehensive set of experiments. Particularly, we compare *SPATE* against two competing approaches for three different metrics and eight different usage scenarios.

3.7.1 Compared Frameworks

Our aim in this experimental evaluation is to compare the following three frameworks:

- **RAW:** This is the default solution that stores the telco snapshots as data files on the HDFS file system without any compression, indexing or decaying.
- **SHAHED:** This is a MapReduce-based system for querying and visualizing spatio-temporal satellite data proposed in [70]. *SHAHED* is appropriate for online querying and visualization, but does not deploy compression or decaying in its internal structures. To allow fair comparison, we isolated the spatio-temporal aggregate index of SHAHED, part of SpatialHadoop 2.4 [58], and executed it along with the other frameworks in Spark [87].
- **SPATE:** This is the framework proposed in this work. *SPATE* aims to minimize the usage of disk space by retaining fast spatio-temporal querying means.

3.7.2 Experimental Testbed

Our experimental testbed is implemented on top of a *Hadoop Distributed File System (HDFS)* [88] along with Apache Hive [88] for *online* querying and Apache Spark [87] for *offline* (i.e., data-intensive distributed in-memory) data processing. Our testbed stores data in either text format (for *RAW* and *SHAHED*) or compressed (for *SPATE*). We use an HDFS file system with 64MB block size and default replication 3. In order to streamline the trace-driven experimental evaluation process, we have formulated the evaluation tasks as individual Scala programs submitted directly to the Spark computation master. In this way, we managed to circumvent additional latencies and overheads introduced by the query exploration interfaces introduced in Section 3.6.

Our evaluation is carried out on the DMSL-UCY laboratory VCenter IaaS datacenter, a private cloud, which encompasses 5 IBM System x3550 M3 and HP Proliant DL 360 G7 rackables featuring single socket (8 cores) or dual socket (16 cores) Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, respectively. The datacenter is managed through a VMWare vCenter Server 5.1 that connects to the respective VMWare ESXi 5.0.0 hosts. The computing cluster, deployed over our VCenter IaaS, comprises of 4 Ubuntu 14.04 server images, each featuring 8GB of RAM with 2 virtual CPUs (@ 2.40GHz). The images utilize slow 7.2K RPM RAID-5 SAS 6 Gbps disks, available through a

IBM storage system DS3512, formatted with VMFS 5.54 (1MB block size). Each node features the following frameworks, i.e., Hadoop v2.5.2, Spark 1.6.0 and Hive 2.0.

3.7.3 Datasets

We utilize an anonymized dataset of telco traces comprising of 1.7M call detail records (CDR), 21M network measurements records (NMS) and 3660 cells (CELL) coming from 1192 2G, 3G and LTE antennas distributed in an area of about 6000 km^2. The data traffic is created from about 300K users and has a total size of ∼5GB. To evaluate the ingestion and querying time of our propositions for various day periods, we have generated the following four large datasets based on the arrival time of the snapshots:

- **Morning Dataset:** was generated by extracting the data that have time arrival between 5 am to 12 pm (noon).
- **Afternoon Dataset:** was generated by extracting the data that have time arrival between 12 pm to 5 pm.
- **Evening Dataset:** was generated by extracting the data that have time arrival between 5 pm to 9 pm.
- **Night Dataset:** was generated by extracting the data that have time arrival between 9 pm to 5 am.

In order to better understand the behavior of our real dataset, we have additionally partitioned the dataset into seven zones at the granularity of week days (i.e., Monday to Sunday).

3.7.4 Metrics

We utilize three metrics, the first two targeting the storage and indexing process and the third one the data querying and exploration process, as follows:

- **Ingestion Time:** this measures the cost incurred for storing each arriving snapshot and incrementing the index. Given a compression library c and a data snapshot d, the ingestion time includes the compression time T_{c1} needed to compress d and the time needed to run the Incremence module that adds the data into our spatiotemporal index.

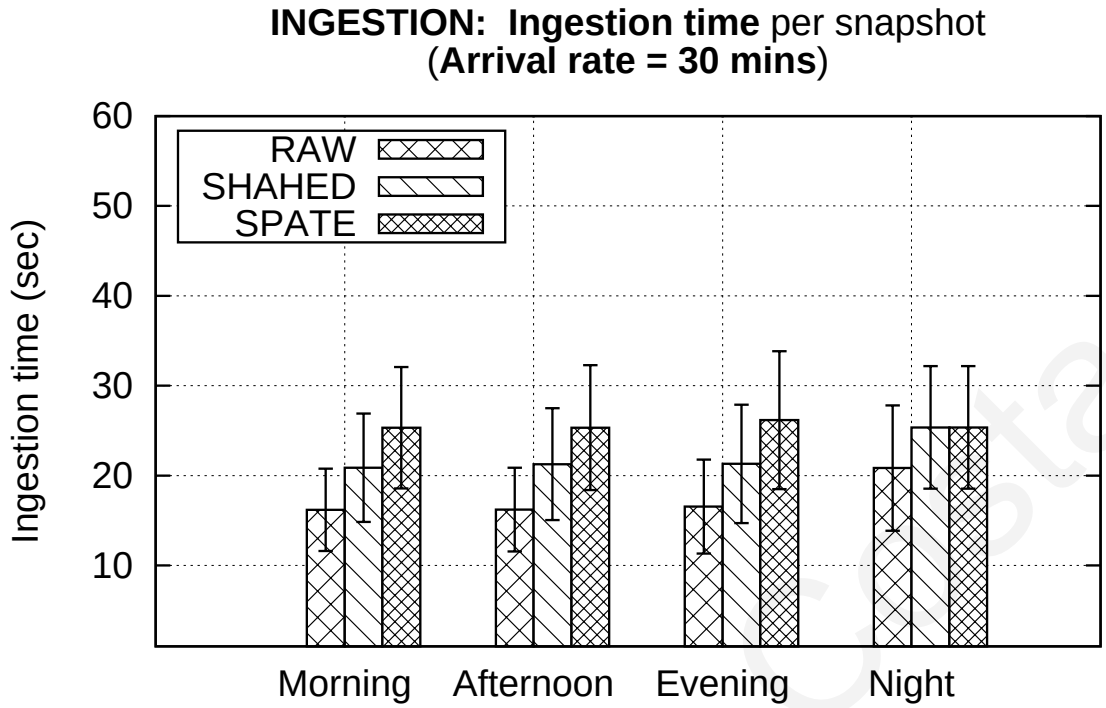

Figure 3.6: **Ingestion time:** *We compare* SPATE *against* RAW *and* SHAHED *on real data partitioned by day period.*

- **Space:** this measures the total space S' that data and index occupy throughout the whole distributed system in Megabytes (MB). Space S' is calculated after all incoming snapshots have been compressed and ingested.
- **Response Time:** this measures the response time for answering a query in seconds (sec). The time is being calculated from the query submission until the answer is calculated. This includes the accessing of the spatiotemporal index, the decompression of the retrieved data and the presentation of the results. Times are averaged over five iterations measured in seconds.

3.7.5 Data Exploration Tasks

Our experimental evaluation has been conducted based on an a diverse mix of OLTP, OLAP, privacy sanitization, statistics, data mining, and Machine Learning (ML) workloads. All aforementioned workloads are driven by a telco-specific domain task. We particularly formulated the following eight tasks segmented into two groups: (i) T1-T5 represent basic operational and analytical queries as well as privacy sanitization tasks that were executed without Spark parallelization; and (ii) T6-T8 represent heavier

computational tasks that were executed with Spark parallelization.

T1. Equality: This task aims to retrieve the download and upload bytes for a requested snapshot, e.g., `SELECT upflux, downflux FROM CDR WHERE ts=‘‘201601221530’’;`

T2. Range: this aims to retrieve the download and upload bytes for a requested time window, e.g., `SELECT upflux, downflux FROM CDR WHERE ts>=‘‘2015’’ AND ts<=‘‘2016’’;`

T3. Aggregate: this aims to retrieve the NMS counters for the drop calls of each cell tower and calculate the drop call rate for each cluster of cells, e.g., `SELECT cellid, SUM(val) FROM NMS WHERE ... GROUP BY cellid;`

T4. Join: this query involves a self-join among two CDR tables. Particularly, it aims to identify the products that have changed their location (as identified by the cell towers).

T5. Privacy: This task retrieves and anonymizes the result set based on the k-anonymity model proposed in [56] through the ARX [89] Java library. Particularly, it generates a k-anonymized dataset by generalizing, substituting, inserting, and removing information as appropriate in order to make the quasi-identifiers indistinguishable among k rows.

T6. Statistics: This task aims to generate a variety of multivariate statistics using Spark's Machine Learning (ML) library (i.e., `Statistics.colStats(observations)`). The goal is to calculate the column-wise max, min, mean, variance, number of non-zeros and the total count.

T7. Clustering: This task aims to cluster a specific range of snapshots using the k-means algorithm in Spark based on the CDR and NMS data.

T8. Regression: This task estimates relationships among the attributes in our dataset using linear regression over a specific temporal window. Spark's Machine Learning (ML) library is used (i.e., `regression.LinearRegression`).

3.8 Experimental Evaluation

In this section we carry out an extensive experimental evaluation that aims to uncover the performance properties of our propositions.

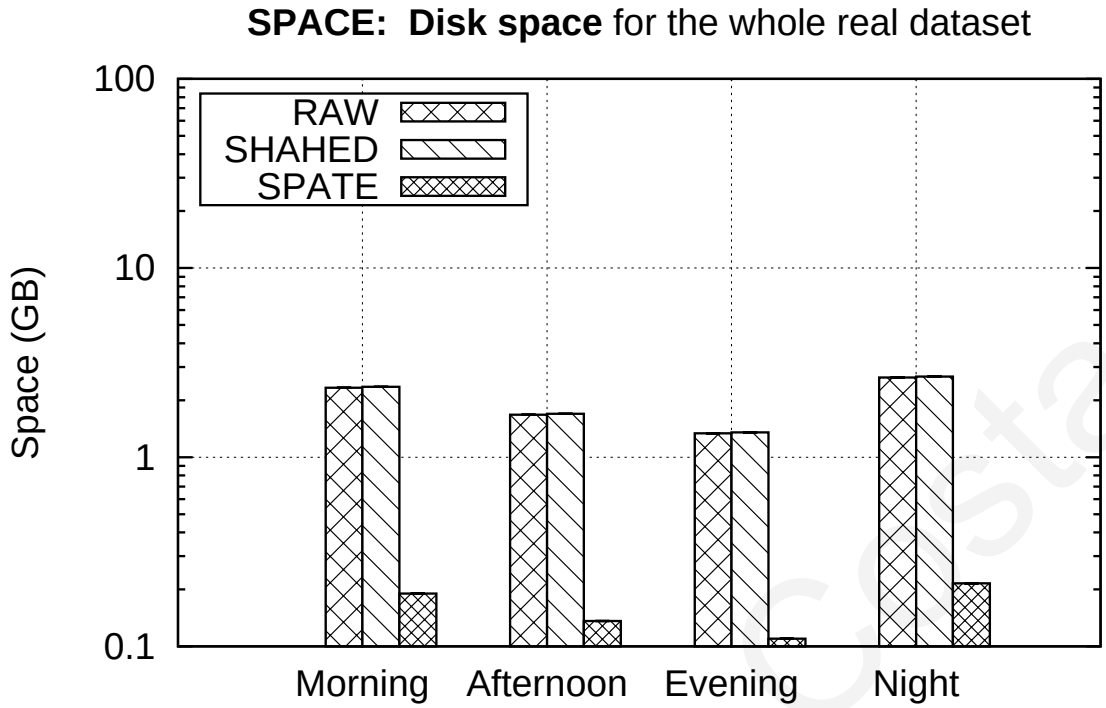

Figure 3.7: **Disk space:** *We compare* SPATE *against* RAW *and* SHAHED *on real data partitioned by day period.*

3.8.1 Performance over varying day-periods

This experimental series studies the effect that the various day periods (i.e., morning, afternoon, evening, night) have on the ingestion time and disk space needed to index and store the incoming snapshots of both NMS and CDR data.

In Figure 3.6 we see that *SPATE* has the slowest ingestion time, although still comparable (i.e., a maximum of 1.25x slower). We also observe that the data load per snapshot affects the ingestion time only negligibly, assuming that the data load varies among day periods. Even though *SPATE* is somewhat slower during ingestion, its benefits are manifested in Figure 3.7, where we observe that *SPATE* requires an order of magnitude less disk space compared to the other techniques. This performance is also steady with respect to the varying data loads associated with each different day period.

We conclude that *SPATE* achieves a large improvement in expensive disk space requirements trading-off a negligible overhead on the ingestion time. Since each snapshot arrives only every 30 minutes (i.e., 1800 seconds) and the ingestion completes within two orders of magnitude less time, we can claim that the small delay in ingestion time is acceptable.

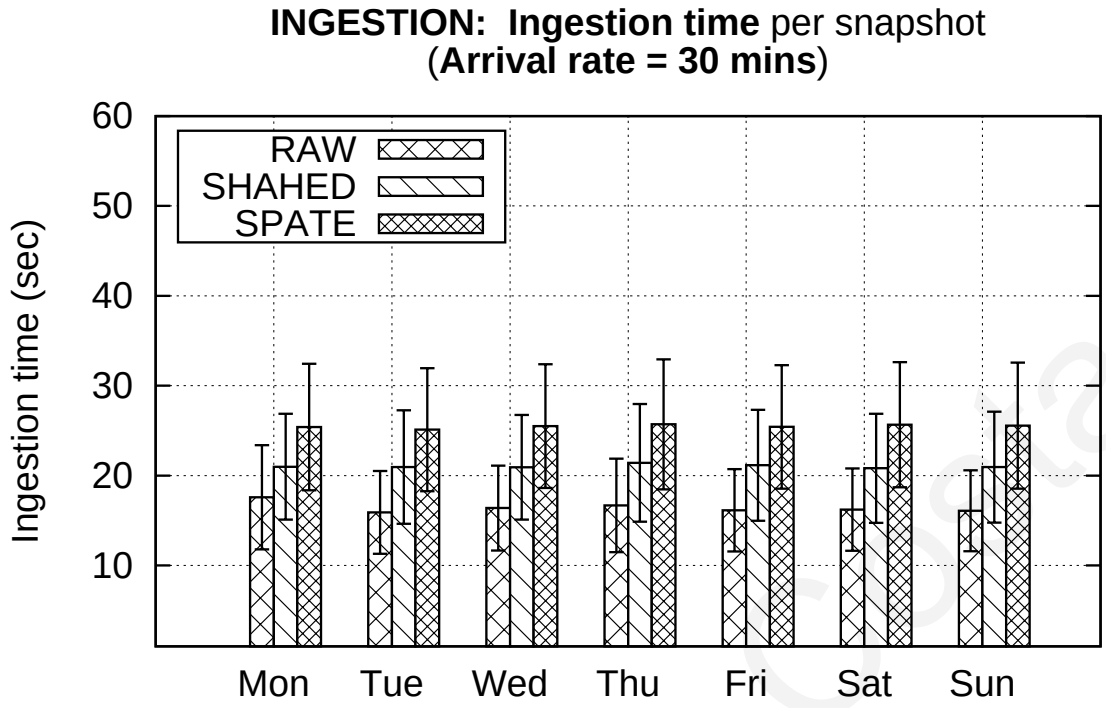

Figure 3.8: **Ingestion time:** *We compare* SPATE *against* RAW *and* SHAHED *on real data partitioned by day of week.*

3.8.2 Performance over days of the week

This experimental series studies the effect that the various week days (i.e., Monday through Sunday) have on the ingestion time and disk space needed to index and store the incoming snapshots of both NMS and CDR data. In Figure 3.8, we see again that *SPATE* has the slowest ingestion time, although still comparable (i.e., a maximum of 1.2x slower this time). We also observe that the data load per snapshot affects the ingestion time only negligibly, assuming that the data load varies between days. In Figure 3.9, we observe that *SPATE* requires again an order of magnitude less disk space. This performance is also steady with respect to the varying data loads among week days. The conclusion here is similar, *SPATE* achieves a large improvement in disk space, trading-off a negligible overhead on the ingestion time.

3.8.3 Response time

This experimental series studies the response time each ingestion framework achieves for the tasks described in Section 3.7.5 using the CDR dataset.

In Figure 3.10, we present the query response time for the simpler tasks T1-T5, all

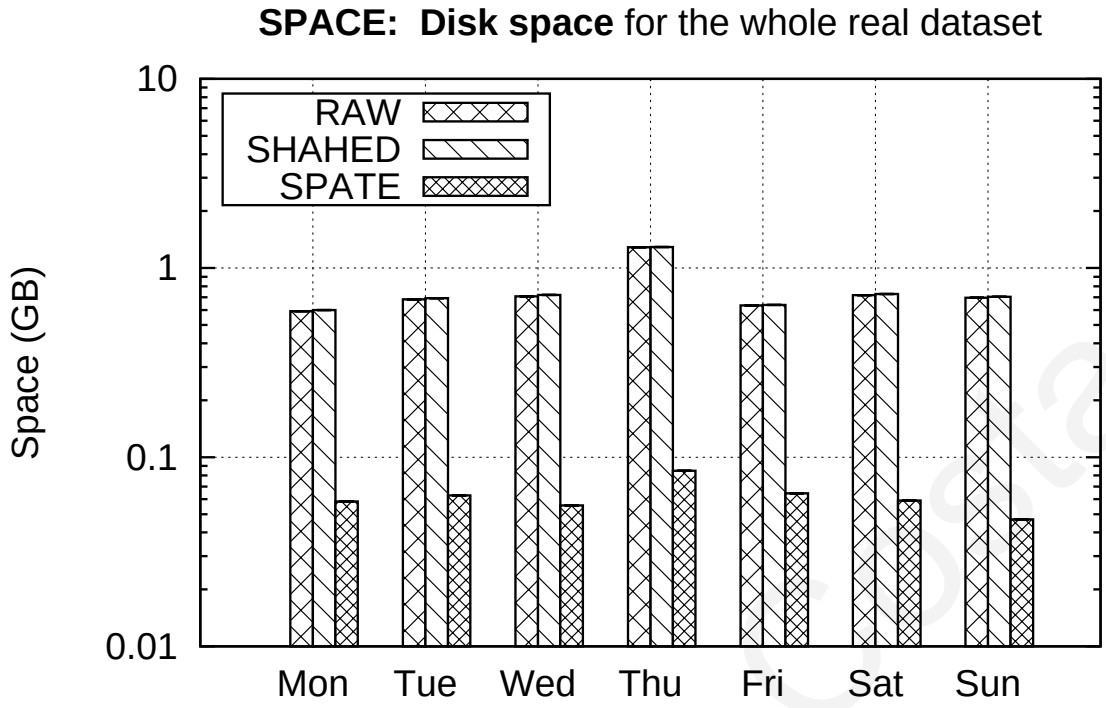

Figure 3.9: **Disk space:** *We compare* SPATE *against* RAW *and* SHAHED *on real data partitioned by day of week.*

of which either involve a single scan or a nested loop on the data stored on HDFS. We observe that *SPATE* is only slightly slower than *SHAHED* for T1-T3 and T5 (i.e., from 0.1s to 3s), which is due to the fact that *SPATE* requires time to decompress the files stored on the HDFS before returning the results. On the other hand, for the join task T4, *SPATE* achieves a 4-5x speed up compared to *SHAHED*, mainly due to the fact that T4 involves a nested loop and that such a loop is much faster in *SPATE* where the HDFS input streams are already compressed in GZIP.

In Figure 3.11, we present the query response time for the heavier tasks T6-T8 in log-scale, which are executed with Spark parallelization on. Our first observation is that the query response time of all three tasks are now in the order of many thousand seconds, but *SPATE* remains close the running time of *SHAHED* in all cases. The second observation is that some tasks, such as the k-means computation in T7 and the Linear Regression in T8, the query response time benefits for *SPATE* are not so apparent. This happens as both T7 and T8 are CPU-bound rather than I/O-bound problems and as such, the existence of compressed input streams is not helping the query response time significantly. The significant benefit of *SPATE* over the other two frameworks for T7 and T8 are still of course that it requires significantly less storage

(i.e., 10x reduction). For all eight tasks, *SPATE* requires the least storage space, i.e., 0.49GB vs. 5.37GB and 5.32GB required by *SHAHED* and *RAW*, respectively.

3.9 Related Work

In this section we present existing research on telco big data and on compressing incremental archives. These works are not directly comparable to SPATE, but are presented for completeness.

3.9.1 Telco Big Data Research

Telco big data research falls in the following categories: (i) real-time analytics and detection; (ii) experience, behavior and retention analytics; and (iii) privacy. There is also traditional telco research not related to big data, but it comprises of topics related to business support services (BSS) data in relational databases.

Real-time Analytics and Detection: Zhang et al. [13] developed *OceanRT*, which was one of the first real-time telco big data analytic demonstrations. Yuan et al. [3] present *OceanST* which features: (i) an efficient loading mechanism of ever-growing telco MBB data; (ii) new spatiotemporal index structures to process exact and approximate spatiotemporal aggregate queries. Iyer et al. [15] present *CellIQ* to optimize queries such as *"spatiotemporal traffic hotspots"* and *"hand-off sequences with performance problems"*. It represents the snapshots of cellular network data as graphs and leverages on the spatial and temporal locality of cellular network data. Zhu et al. [21] deal with the usage of telco MR data for city-scale localization, which is complementary to the scope of our work.

Braun et al. [49] develop a scalable distributed system that efficiently processes mixed workloads to answer event stream and analytic queries over telco data. Bouillet et al. [51] develop a system on top of IBM's InfoSphere Streams middleware that analyzes 6 billion CDR per day in real-time. Abbasoğlu et al. [90] present a system for maintaining call profiles of customers in a streaming setting by applying scalable distributed stream processing. All aforementioned efforts have a similar scope to SPATE, but don't incorporate concepts of compression or decaying of data.

Experience, Behavior and Retention Analytics: Huang et al. [14] empirically demonstrate that customer churn prediction performance can be significantly impro-

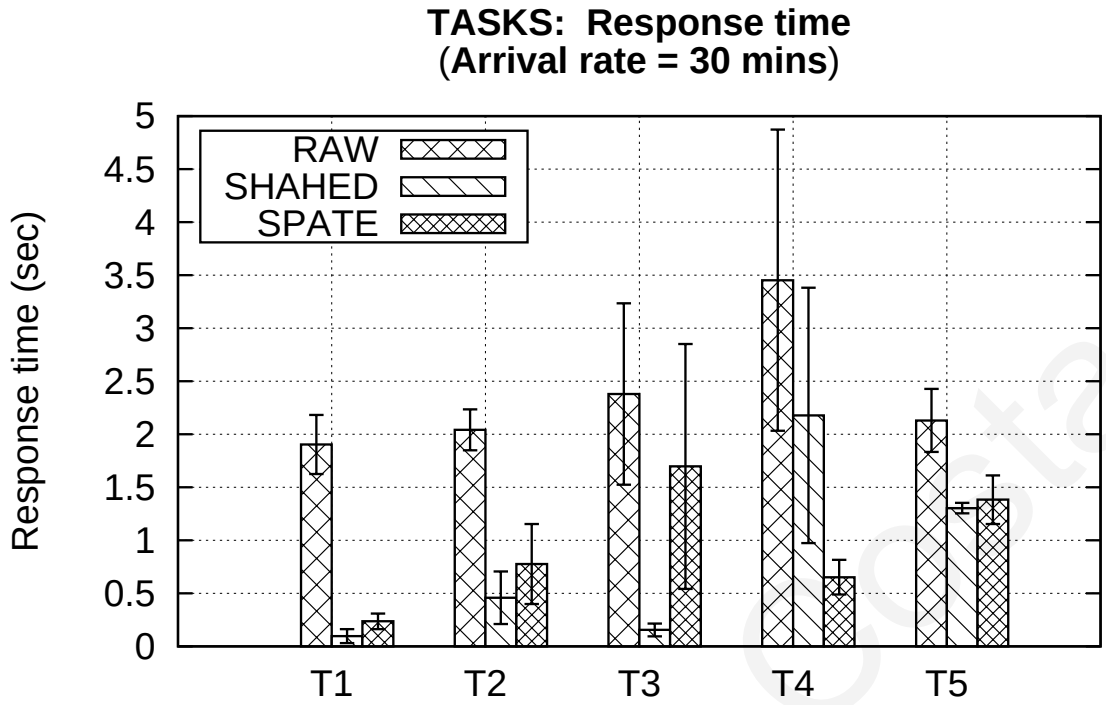

Figure 3.10: **Response time for simpler tasks T1-T4:** *We compare* SPATE *against* RAW *and* SHAHED *on the complete real dataset.*

ved with telco big data. Although BSS data have been utilized in churn prediction very well in the past decade, the authors show how with a primitive Random Forest classifier telco big data can improve churn prediction accuracy from 68% to 95%. Luo et al. [16] propose a framework to predict user behavior involving more than one million telco users. They represent users as documents containing a collection of changing spatiotemporal "words" that express user behavior. By extracting the users' space-time access records from MBB data, they learn user-specific compact topic features that they use for user activity level prediction. Ho et. al. [91] propose a distributed community detection algorithm that aims to discover groups of users that share similar edge properties reflecting customer behavior. Iterative learning algorithms are not the primary target for SPATE, but can be supported at similar costs to RAW data (i.e., decompression occurs in first iteration).

Privacy: Hu et al. [57] study Differential Privacy for data mining applications over telco big data and show that for real-word industrial data mining systems the strong privacy guarantees given by differential privacy are traded with a 15% to 30% loss of accuracy. Privacy and confidentiality are critical for telcos' reliability due to the highly sensitive attributes of user data located in CDR, such as billing records, calling

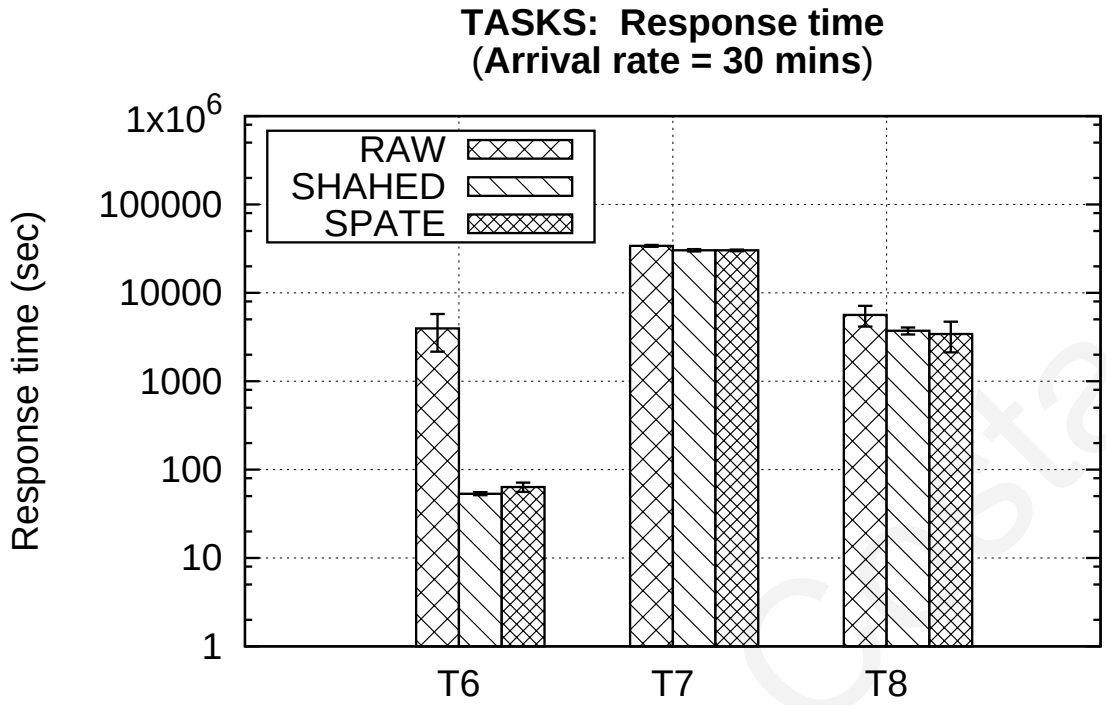

Figure 3.11: **Response time for heavier tasks T5-T8:** *We compare* SPATE *against* RAW *and* SHAHED *on the complete real dataset.*

numbers, call duration, data sessions, and trajectory information. *SPATE* deals with privacy-aware data sharing as a functionality for next generation smart applications.

3.9.2 Compressing Incremental Archives

Domain-specific compression techniques have previously been proposed, e.g., for compressing spatiotemporal climate data [73], text document collections [92], scientific simulation floating point data [76–79], and floating point data streams [93]. None of these prior works has been proposed for distributed systems and can not be directly applied to telco data, which mostly contains generic string and integer values.

Works [94–96] have studied differential compression techniques and the trade-off between compression ratio and decompression times for incremental archival data. Differential compression is a topic we will investigate more carefully in the future as it can reduce the storage layer overheads in each acquisition cycle.

3.10 Summary

In this chapter, we proposed an innovative telco big data exploration stack, coined *SPATE*, whose objectives are two-fold: (i) minimizing the storage space needed to incrementally retain data *over time*; and (ii) minimizing the response time for spatio-temporal data exploration queries over recent data. We have measured the efficiency of our proposition using a real telco trace and a variety of telco-specific tasks, such as OLAP and OLTP querying, clustering, regression and privacy sanitizing, and showed that we can achieve comparable response times to the state-of-the-art with an order of magnitude less storage space.

Chapter 4

Decaying Telco Big Data with Data Postdiction

In this chapter, we present a novel decaying operator for Telco Big Data (TBD), coined *TBD-DP (Data Postdiction)*. Unlike data prediction, which aims to make a statement about the future value of some tuple, our formulated *data postdiction* term, aims to make a statement about the past value of some tuple, which does not exist anymore as it had to be deleted to free up disk space. *TBD-DP* relies on existing Machine Learning (ML) algorithms to abstract TBD into compact models that can be stored and queried when necessary. Our proposed *TBD-DP* operator has the following two conceptual phases: (i) in an offline phase, it utilizes a LSTM-based hierarchical ML algorithm to learn a tree of models (coined *TBD-DP* tree) over time and space; (ii) in an online phase, it uses the *TBD-DP* tree to recover data within a certain accuracy. In our experimental setup, we measure the efficiency of the proposed operator using a ∼10GB anonymized real telco network trace and our experimental results in Tensorflow over HDFS are extremely encouraging as they show that *TBD-DP* saves an order of magnitude storage space while maintaining a high accuracy on the recovered data.

4.1 Introduction

In recent years there has been considerable interest from *telecommunication companies (telcos)* to extract concealed value from their network data. Consider for example a telco in the city of Shenzhen, China, which serves 10 million users. Such a telco is shown to produce 5TB per day [13] (i.e., thousands to millions of records every second). Huang et al. [14] break their 2.26TB per day *Telco Big Data (TBD)* down as follows: (i) *Business Supporting Systems (BSS)* data, which is generated by the internal workflows of a telco (e.g., billing, support), accounting to a moderate of 24GB per day and;

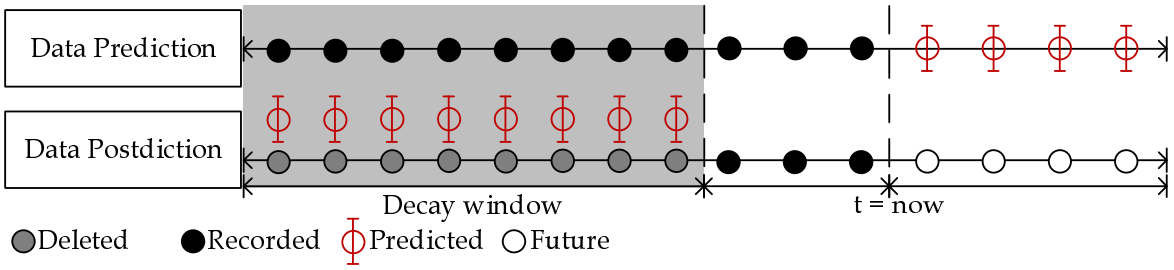

Figure 4.1: **Data Prediction (top)***: aims to find the future value of some tuple.* **Data Postdiction (bottom)***: aims to recover the past value of some tuple, which has been deleted to reduce the storage requirements, using a ML model.*

(ii) *Operation Supporting Systems (OSS)* data, which is generated by the Radio and Core equipment of a telco, accounting to 2.2TB per day and occupying over 97% of the total volume. Additionally, in [18] it is shown that the amount of storage doubles every year and storage media costs decline only at a rate of less than 1/5 per year. Finally, high-availability storage mandates low-level data replication (e.g., in HDFS the default data replication is 3).

Consequently, we claim that the vision of infinitely storing all IoT-generated velocity data on fast high-availability or even deep storage will gradually become too costly and impractical for many analytic-oriented processing scenarios.

To this end, *data decaying* [74, 97] (or data rotting) has recently been suggested as a powerful concept to complement traditional data reduction techniques [98, 99], e.g., sampling, aggregation (OLAP), dimensionality reduction (SVD, DFT), synopsis (sketches) and compression. Data decaying refers to *"the progressive loss of detail in information as data ages with time"*. In data decaying recent data retains complete resolution, which is practical for operational scenarios that can continue to operate at full data resolution, while older data is either compacted or discarded [17, 74, 97]. Additionally, the decaying cost can be amortized over time, matching current trends in micro-batching (e.g., Apache Spark). Unfortunately, data decaying currently relies on rather straightforward methodologies, such as rotational decaying (i.e., FIFO) [97], or decaying based on specific queries [17] rather than the complete dataset itself. Our aim in this work is to expand upon these developments to provide more intelligent and generalized decaying operators.

In this chapter, we present a novel decaying operator for Telco Big Data, coined *TBD-DP (Data Postdiction)* (see Figure 4.1). Unlike data prediction, which aims to make a statement about the future value of some tuple in a TBD store, data postdiction

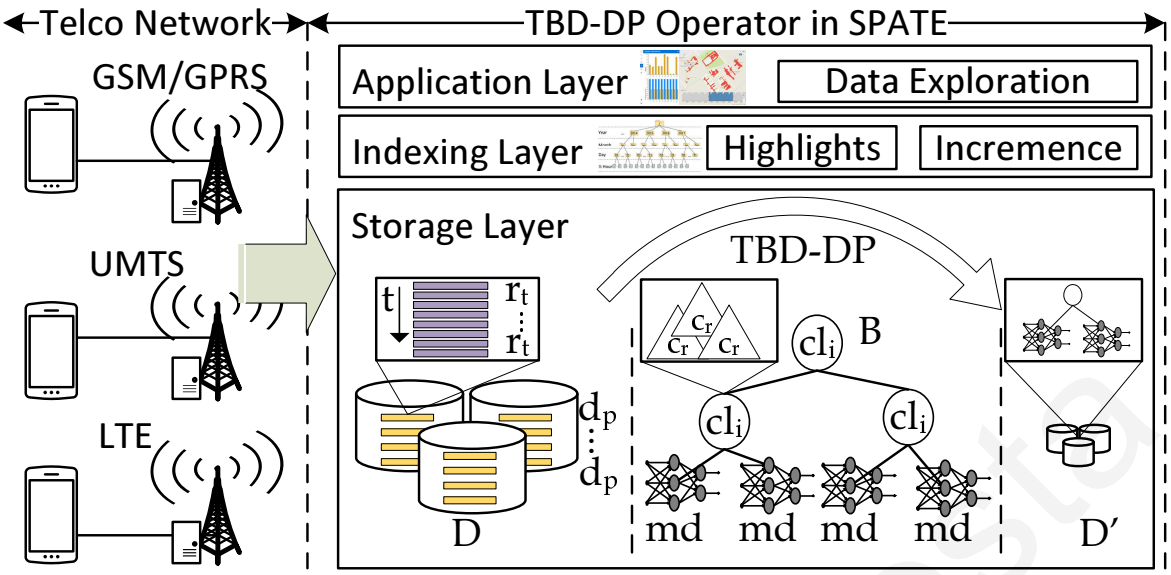

Figure 4.2: System Model: The TBD-DP operator works on the storage layer of a typical TBD stack and abstracts the incoming data signals (D) into abstract models (md) that are organized in a tree data structure (B).

aims to make a statement about the past value of some tuple that does not exist anymore, as it had to be deleted to free up space. *TBD-DP* relies on existing Machine Learning (ML) algorithms to abstract TBD into compact models that can be stored and queried when necessary. Our proposed *TBD-DP* operator has the following two conceptual phases: (i) in an offline phase, it utilizes a LSTM-based hierarchical ML algorithm to learn a tree of models (coined *TBD-DP* tree) over time and space; (ii) in an online phase, it uses the *TBD-DP* tree to recover data with a certain accuracy.

We claim that the LSTM model is capturing the essence of the past through its short and long-term dependencies, similarly to how the brain retains both recent information and important old information at a high resolution.

To understand the operational aspects of our proposed TBD-DP operator, consider Figure 4.2, where we show how incoming telco data signals are absorbed by the TBD architecture and stored on high-availability and fast storage (i.e., D). This helps to carry out operational tasks (e.g., alerting services and visual analytics) with full data resolution. Subsequently, in the first phase of *TBD-DP*, we utilize a specialized *Recurrent Neural Network (RNN)* composed of *Long Short Term Memory (LSTM)* units, which has the ability to detect long-term correlations in activity data and the trained model has a small disk space footprint [100]. This enables *TBD-DP* to utilize minimum storage capacity of the decayed data by representing them with LSTM models on the disk media (D') and provide real-time postdictions with high accuracy in

a subsequent recovery phase, which will be initiated on-demand (i.e., whenever some high-level operator requests the given data blocks).

The contributions of this work are summarized as follows:

- We propose a TBD decay operator that deploys the notion of data postdiction using off-the-shelf LSTM-based prediction models.
- We propose the DP-tree, which is a hierarchical index to organize the generated models in a data structure to enable the efficient recovery of data when necessary.
- We measure the efficiency of the proposed operator using a ∼10GB anonymized telco network trace, showing that *TBD-DP* can be a premise for efficient TBD analytics in the future. We also summarize a prototype architecture and user interface we have developed for the management of TBD.

4.2 System Model and Problem Formulation

This section formalizes our system model, assumptions and problem. The main symbols and their respective definitions are summarized in Table 4.1.

A typical Telco system, illustrated in Figure 4.2, is composed of the Telco network, which is responsible for providing telecommunication services, and a Telco data management system, such as SPATE [17], which is responsible for the efficient analytical exploration of Telco datasets. The data arrives at the data center in batches, called henceforth data snapshots noted by d_p, in the form of horizontally segmented files within an ingestion period p. A snapshot d_p contains multiple records r_t created at a certain timestamp t. Each record r_t consists of a predefined set of attributes including the cell id c_r that represents the spatial information inherent within the Telco network. Particularly, each cell id c_r corresponds to a cell that covers a geographical cellular area that usually spans hundreds of meters or even kilometers. Finally, the cells are spatially grouped into clusters $cl_i, i = i \dots k$ for facilitating the postdiction process by creating a model $md_i, i = i \dots k$ for each cl_i as this will be explained in the next section.

4.2.1 Problem Formulation

Research Goal. *Given a Telco setting, this work aims at achieving a pre-specified decaying of TBD with minimum additional storage space capacity and being able to*

Table 4.1: Summary of Notation

Notation	Description
p, d_p, D	Ingestion period, data snapshot of one p, set of all d_ps
t, r_t	Timestamp within an ingestion cycle, record at t
C, c_r, cl_i	Set of all cell towers, Cell of record r, cluster of records $i = 1, \ldots, k$
md_i, MD	LSTM model of cluster cl_i, set of all models
f	Decaying factor: percentage of data to be removed

recover the decayed data accurately and efficiently.

The efficiency of the proposed techniques to achieve the above goal is measured by the following objectives:

Definition 4.2.1. Storage Capacity (*S*) is the total storage space required for achieving decaying of data based on a pre-specified decaying factor f.

Definition 4.2.2. Normalized Root Mean Square Error (*NRMSE*) is the percentage of the correctly recovered decayed data. It is measured by the normalized root-mean-square error, which is the normalized difference between the actual data ($x_{1,t}$) and the predicted data ($x_{2,t}$), where t is a discrete time point and y_{max}, y_{min} the maximum and minimum observed differences, formally:

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_{t=1}^{n}(x_{1,t} - x_{2,t})^2}}{(y_{max} - y_{min})}$$

4.3 The *TBD-DP* operator

In this section, we introduce the proposed *TBD-DP* operator and discuss its two internal algorithms, namely, the *Construction* (data model creation) and the *Recovery* (data recreation), which capture its core functionality as illustrated in Figure 4.3.

The *Construction* algorithm can be triggered either by the user, or automatically when the total storage capacity reaches a certain level. In both cases, the data are initially clustered based on spatial characteristics and then ordered based on temporal information. Finally, postdiction models based on the LSTM machine learning approach are generated for each cluster and the real data is decayed by $f\%$. The *Recovery*

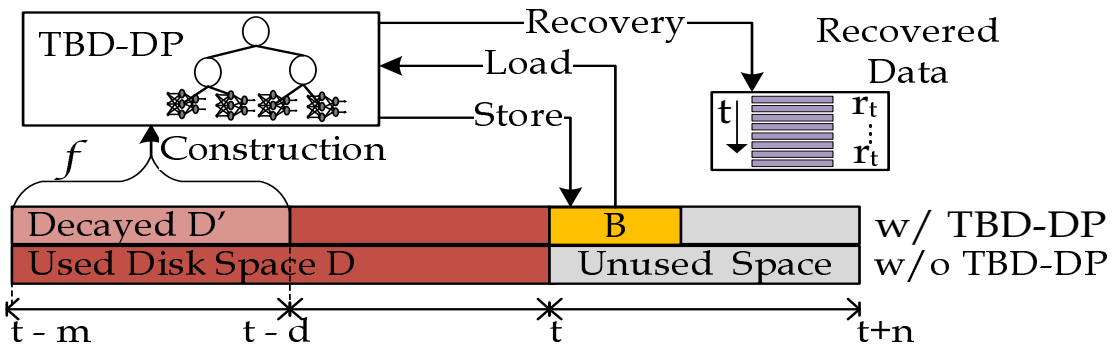

Figure 4.3: TBD-DP Operator Overview.

algorithm utilizes the postdiction models for retrieving the decayed data by adopting a proposed DP-tree based algorithm.

4.3.1 Construction Algorithm

Algorithm 1 outlines the major steps of the construction algorithm. Initially, the decaying factor f specifies the percentage of the whole dataset D that will be decayed, and consequently the decayed subset $D' \subseteq D$ that will be utilized for generating the postdiction models. In the spatial partitioning step (Step 1 - lines 2-5), $k \leq |C|$ clusters are created by using the cell tower locations. Particularly, each cluster $cl_i, i = 1, \ldots, k$ is represented by a cell tower (in cases where $k < |C|$ then the closest cell towers are merged using a kNN approach until we finally generate k clusters). This allows us to construct less models based on the network topology resulting to less computations. k is calculated based on the resources of the system. Then the MAP function associates all records $r_t \in D'$ with the previously created clusters by taking into consideration their cell id c_r attribute. By the end of this function execution, k clusters of cell towers with their associate records will be created. Then all records of each cluster are ordered based on their timestamp (i.e., time originally generated) by using the *ORDER* function of the temporal ordering step (Step 2 - lines 6-8). This allows the neural network to be created correctly based on a continuous time series. Finally, the learning step (Step 3 - lines 9-12) generates k postdiction models md_i for each cluster cl_i by using a specialized Recurrent Neural Network (RNN) known as the Long Short Term Memory (LSTM) model [101].

Specifically, the *LEARNING* function generates, for each cluster at each iteration, an LSTM model that relies on a structure called a memory cell, which is composed of four main elements: an input gate, a neuron with a self-recurrent connection (a

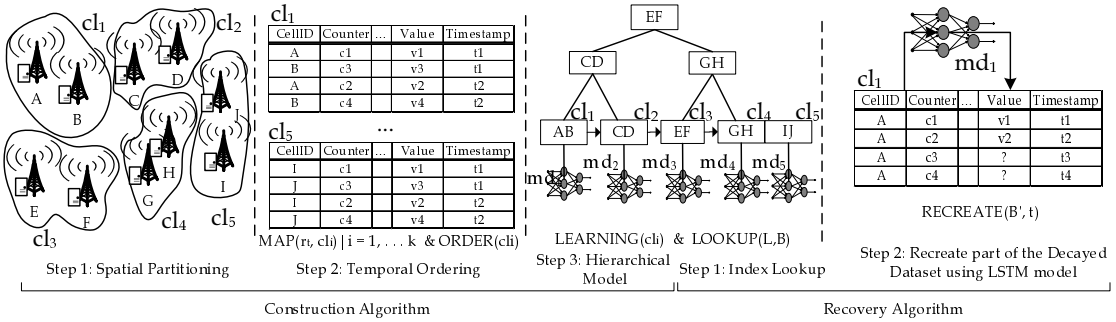

Figure 4.4: The conceptual steps of the proposed TBD-DP construction and recovery algorithm.

connection to itself), a forget gate and an output gate. A memory cell is updated at every time-step by using the following parameters and equations:

- x_t is the input to the memory cell layer at time-step t
- W_i, W_f, W_C and W_o are weight matrices
- b_i, b_f, b_C and b_o are bias vectors

The forget gate layer:

$$f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f),$$

decides what information is going to be thrown away from the memory cells. The input gate layer:

$$i_t = s(W_i[h_{t-1}, x_t] + b_i),$$

decides which values to be updated. The tanh layer decides what new information we are going to store in the memory cells using:

$$\widetilde{C_t} = tanh(W_C[h_{t-1}, x_t] + b_C).$$

Moreover, the update memory cells function:

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C_t},$$

used to forget the things decided to be forgotten earlier and scale the new candidate values by a pre-specified state value.

Finally, the update hidden cells function:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

Algorithm 1 - *TBD-DP* Construction Algorithm

Input: Dataset D, C set of cell towers, Number of clusters k

Output: B: Set of models MD (DP-tree structure)

▷ **Step 0: Decaying Pre-processing**

1: $D' \leftarrow f$ of D ▷ Select $f\%$ of D to be decayed

▷ **Step 1: Spatial Partitioning**

2: Create $k \leq |C|$ clusters cl_i ▷ Use cell towers locations

3: **for all** $r_t \in D'$ **do**

4: $cl_i \leftarrow MAP(r_t, cl_i)|i = 1, \ldots k$ ▷ Associate records to clusters

5: **end for**

▷ **Step 2: Temporal Ordering**

6: **for** $i = 1$ to k **do**

7: $cl_i \leftarrow ORDER(cl_i)$ ▷ Sort records in clusters based on timestamp

8: **end for**

▷ **Step 3: Hierarchical Model**

9: **for** $i = 1$ to k **do**

10: $md_i \leftarrow LEARNING(cl_i)$ ▷ Create an LSTM model for each cl_i

11: Insert md_i in B

12: **end for**

and a sigmoid layer that decide what parts of the cell state to output,

$$h_t = o_t * tanh(C_t).$$

The *Construction* algorithm outputs a set of postdiction models B in a DP-tree for facilitating the recovery algorithm that follows. At the end of the *Construction* algorithm execution, the D' set of data is removed for saving storage space and it is conceptually replaced by the final B set of postdiction models, where $|B| << |D'|$.

Example: Consider the scenario in Figure 4.4 where there are 10 cell towers $\{A, ..., J\}$. First, the *Construction* algorithm creates $k = 5$ clusters $\{cl_1,...,cl_5\}$ denoted with the solid line that surrounds the cell towers in Step 1 of Figure 4.4 (left). The *MAP* function associates the records to a cluster based on the cell id c_r (e.g., all records related to A and B are grouped into cl_1). Then, the *ORDER* function sorts the records of each cluster based on their timestamp t as shown in Step 2 of Figure 4.4 (center). Finally, for each cluster cl_i a model md_i is trained and inserted into a DP-tree index using the cell ids as keys, as shown in Figure 4.4 (right).

Decay Principle of TBD-DP: *Decaying* refers to the progressive loss of detail in information as data ages with time until it has completely disappeared. Kersten refers to the existence of data fungus in [74] with a decaying operator coined *"Evict Grouped Individuals (EGI)"*. The given EGI operator performs biased random decaying, resembling the rotting process in nature (e.g., in fruits with fungus). In our previous work [17], we used the *First-In-First-Out (FIFO)* data fungus, i.e., *"Evict Oldest Individuals"*, which retains full resolution for recent data but abstracts older data into compact aggregation models. Both EGI and FIFO do not retain full resolution for important instances that occurred in the past. Consequently, data would have been rotted and purged either randomly or based on its timestamp. This is called the *long-term dependency* problem [102]. In this work, we chose a radically new decaying technique that could be termed as *LSTM data fungus*, which is explicitly designed to avoid the *long-term dependency* problem. Particularly, the *TBD-DP* operator replaces the data with abstract LSTM models, which capture the essence of the past, i.e., both recent data and important old data is retained at the highest possible resolution.

4.3.2 Recovery Algorithm

Algorithm 2 outlines the *Recovery* algorithm that utilizes the DP-tree structure (B) of postdiction models of Algorithm 1 for retrieving a selected subset from the decayed data, i.e., $pD' \subseteq D'$. For doing this, the *Recovery* algorithm inputs the set of models B as well as some spatiotemporal information L and R that will specify the amount of the decayed data to be retrieved. For example, L can be a cellular tower's location or a user's location associated to a cellular tower and R can be a range of timestamps, within which a number of records were generated and stored in D'. In any case, L and R will be utilized by the DP-tree *LOOKUP* function for deciding a subset of models $B' \subseteq B$ in line 13 that will be used for creating the pD' dataset in line 15.

Example: Consider the scenario of Figure 4.4 (Recovery Algorithm) where the data of cell tower A (part of cl_1) needs to be recovered for timestamps t_1,...,t_4. *LOOKUP* retrieves the LSTM model md_1 for cluster cl_1 created from all records related to cell towers A and B as shown in Step 1 of the given figure. In Step 2, the *Recovery* algorithm recreates the values of cell tower A for each timestamp t recovering in this way a part of the decayed data pD' using the selected LSTM model.

Algorithm 2 - *TBD-DP* Recovery Algorithm

Input: L: spatial input; R: temporal input; B: set of postdiction models in a DP-tree structure

Output: Partial decayed dataset pD'

procedure LOOKUP(k,*node*) ▷ The number of children is b.
  if *node* is a leaf **then**
    return *node*
  end if
  switch k **do**
    case $k < k_0$
      return LOOKUP(k, p_0)
    case $k_i \leq k < k_{i+1}$
      return LOOKUP(k, p_{i+1})
    case $k_d \leq k$ ▷ Each node has at most $d \leq b$
      return LOOKUP(k, p_{d+1})
end procedure

▷ **Step 1: Index Lookup**

$B' \leftarrow LOOKUP(L, B)$ ▷ Select a subset of postdiction models

▷ **Step 2: Recreate part of the Decayed Dataset using LSTM model**

for all $t \in R$ **do**
  $pD' = RECREATE(B', t)$ ▷ Retrieve decayed data of specific time periods.
end for

4.3.3 Performance Analysis

The secondary focus of *TBD-DP* is the efficient decaying of data and consequently the minimization of TBD storage space while maintaining a high accuracy during data recovery.

According to Definition 4.2.1 the total storage space S is equal to the actual data minus the decayed data based on f, plus any additional storage required by the decaying approach to achieve an optimal recreation of the decayed data. When there is no decaying $f = 0\%$ then $S = |D| + |B|$ (B could have been used for predicting future D values), which is the size of the actual (raw) data D and the size of the set of prediction models B. In the case of *TBD-DP*, $S = |D| - |D'| + |B|$, which is the actual data size minus the size of the decayed dataset $|D'| = |D| \times f\%$ plus the size of a set of models B, where $|D| >> |D'| + |B|$. When $f = 100\%$ then all data are decayed and the required storage space of *TBD-DP* is $S = |B|$. In the case of sampling, the storage space is equal

to $S = |D| - |V|$, which is the actual data size minus a sample set $V = sampling(D', s)$ generated by sampling the decayed dataset D' with a pre-specified rate s. Note that $|D| - |D'| + |B| << |D| - |V|$ for a reasonable s that provides an *NRMSE* similar to *TBD-DP*.

According to Definition 4.2.2 the *NRMSE* measures the similarity of the decayed dataset D' and the recovered dataset pD'. Therefore, in cases where the decaying factor is $f = 0\%$, which corresponds to a low $|D'| = 0$ and no decaying is applied then $NRMSE = 0$. When $f = 100\%$, which corresponds to a high $|D'| = |D|$ and all data are discarded then $NRMSE >> 0$. Moreover, it is reasonable to assume that in sampling, where a sample set V of the decayed data D' is permanently discarded with a sampling rate s then, its $NRMSE\ (V, D')$ will be equal to the normalized difference between the sampled and the actual data. Finally, the *NRMSE* of the proposed *TBD-DP* will be equal to the normalized difference between the predicted data of the LSTM model and the actual data, i.e., $NRMSE\ (pD', D')$.

4.4 Prototype Description

We have developed a complete prototype architecture that integrates *TBD-DP* as part of the TBD Awareness project [1]. Our proposed architecture comprises of three layers (see Figure 4.2), namely Storage Layer, Indexing Layer and Application Layer.

The *Storage layer* passes newly arrived network snapshots through a lossless compression process storing the results on a replicated big data file system for availability and performance. This component is responsible for minimizing the required storage space with minimal overhead on the query response time. The intuition is to use compression techniques that yield high compression ratios but at the same time guarantee small decompression times. We particularly use GZIP compression that offers high compression/decompression speeds, with a high compression ratio and maximum compatibility with I/O stream libraries in a typical big data ecosystem we use. Additionally, this layer uses the *TBD-DP* operator in order to provide the decay methods for the next layer. The storage layer is basically only responsible for the leaf pages of the *SPATE* index described in the next layer.

The Indexing Layer uses a multi-resolution spatio-temporal index, which is incremented on the rightmost path with every new data snapshot that arrives (i.e., every

[1] TBD Awareness, https://tbd.cs.ucy.ac.cy/

Figure 4.5: The TBD-DP operator implemented inside the spatio-temporal SPATE architecture. The interface enables users to carry out high resolution visual analytics, without consuming enormous amounts of storage. The savings are quantified numerically with bar charts and visually with heatmaps of telco and indoor positioning data.

30 minutes). In addition, the component computes interesting event summaries, called "highlights", from data stored in children nodes and stores them at the parent node. For each data exploration query, the internal node that covers the temporal window of the query is accessed, and its highlights are used to answer the query.

The Application Layer implements the querying module and the *data exploration* interfaces, which receive the data exploration queries in visual or declarative mode and use the index to combine the needed highlights and snapshots to answer the query. *SPATE* is equipped with an easy-to-use map-based web interface layer that hides the complexity of the system through a simple and elegant web interface.

4.5 Experimental Methodology and Evaluation

This section presents an experimental evaluation of our proposed *TBD-DP* operator. We start-out with the experimental methodology and setup, followed by two experiments. Particularly, in the first experiment, the performance of *TBD-DP* is compared against two baseline approaches and two decaying-based approaches with respect to various metrics on a set of anonymized datasets. The second experiment examines the influence of several control parameters on the performance of *TBD-DP*.

4.5.1 Methodology

This section provides details regarding the algorithms, metrics and datasets used for evaluating the performance of the proposed approach.

Testbed: Our evaluation is carried out on the DMSL VCenter IaaS datacenter, a private cloud, which encompasses 5 IBM System x3550 M3 and HP Proliant DL 360 G7 rackables featuring single socket (8 cores) or dual socket (16 cores) Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, respectively. These hosts have collectively 300GB of main memory, 16TB of RAID-5 storage on an IBM 3512 and are interconnected through a Gigabit network. The datacenter is managed through a VMWare vCenter Server 5.1 that connects to the respective VMWare ESXi 5.0.0 hosts. Computing Nodes: The computing cluster, deployed over our VCenter IaaS, comprises of 4 Ubuntu 16.04 server images, each featuring 8GB of RAM with 2 virtual CPUs (@ 2.40GHz). The images utilize fast local 10K RPM RAID-5 LSILogic SCSI disks, formatted with VMFS 5.54 (1MB block size). Each node uses Hadoop v2.5.2.

We utilize anonymized measurements from a real Telco operator that comprises of 1192 real cell towers (i.e., 3660 cells of 2G, 3G and LTE networks) distributed in an area of 5,896 km^2. The cells are connected through a Gigabit network to a datacenter. Each cell tower keeps several UMTS/GSM network logs for the performance of the tower and forwards the information through the base station controller (BSC) or the radio network controller (RNC) to be stored. There is a CDR server that generates call detail records (CDRs) for incoming and outgoing calls in the enterprise. When a CDR is generated in the CDR server, the management server and third-party application can use SFTP to obtain the CDR from the CDR server. Then the Telco can query the CDRs for call/data information and check outgoing call/data fees with the carrier.

Algorithms: The proposed *TBD-DP* operator is compared with the following approaches:

- **RAW:** does not apply any decaying on the whole dataset.
- **COMPRESSION:** the decayed dataset is compressed with the *GZIP* library, which has been shown in [17] to offer the best balance between compression/-decompression speeds, compression ratios and compatibility with I/O stream libraries.

- **SAMPLING:** a sampling method that picks every second item in the input stream, yielding a 50% sample size.
- **RANDOM:** uniformly randomly select one record from the decayed dataset.

Note that RAW and RANDOM are the baseline approaches used to demonstrate the trade-off between the storage capacity and the NRMSE objectives.

Datasets: We utilize an anonymized dataset of telco traces comprising of $\sim$ 100M network measurements records (NMS) and 3660 cells (CELL) coming from 2G, 3G and LTE antennas. The data traffic is created from about 300K objects and has a total size of $\sim$10GB. We constructed 6 realistic datasets from real TBD obtained through *SPATE* described in Section 4.5.1 based on the *Key Performance Indicators (KPIs)* [103].

- **Calls (CS):** the number of calls ended normally during snapshot d_t.
- **Call Drops (CSD):** the number of calls dropped during snapshot d_t.
- **Handover Attempts (HA):** the amount of handovers into or from the cells attempted during a snapshot d_t.
- **Handovers (HS):** the number of successful handovers into or from the cells during a snapshot d_t.
- **Call Setup Attempts (CSA):** the amount of call setup processes attempted during snapshot d_t.
- **Call Setups (CE):** the amount of successful call setup processes during snapshot d_t.

Metrics: We evaluate the performance of *TBD-DP* using the metrics defined in Section 4.2.1 in all experiments:

- **Storage Capacity (S):** measures the total space that data and the DP-tree index occupy together, as a percentage of storage required by the RAW method (no decaying, no compression).
- **Normalized Root Mean Square Error (*NRMSE*):** measures the error of the recovered data D' using the NRMSE formula provided at the end of Section 4.2.

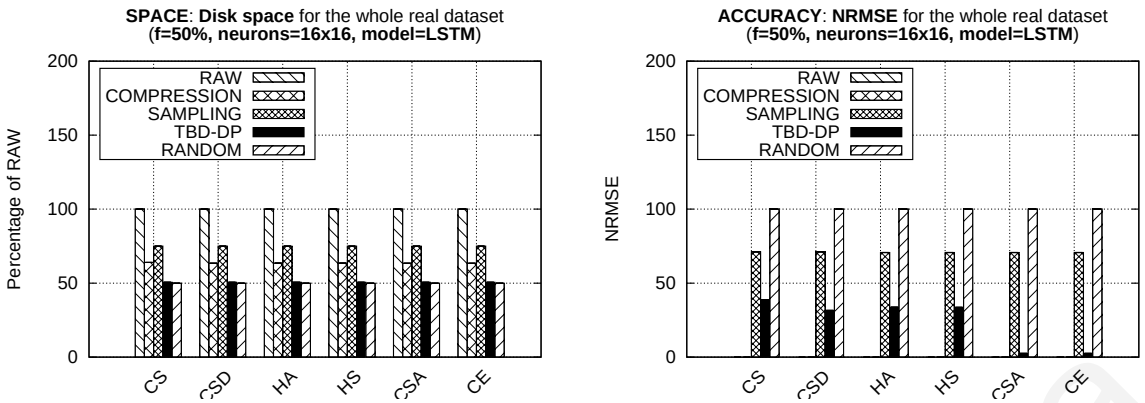

Figure 4.6: Performance Evaluation: TBD-DP evaluation in terms of storage capacity S as a percentage to the RAW data (left) and accuracy in terms of NRMSE *on the decayed set of data (right) in all datasets.*

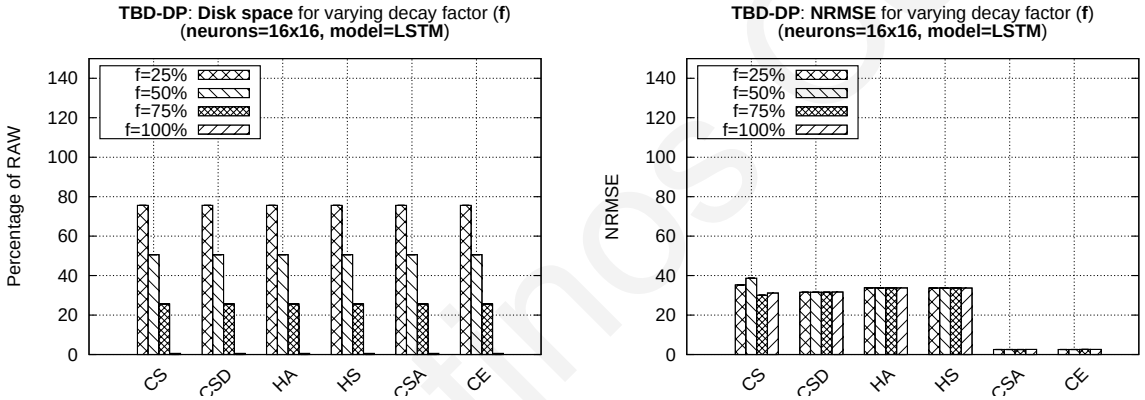

Figure 4.7: Control Experiment - Decaying factor f: examining the storage capacity S and NRMSE *of the proposed TBD-DP approach while varying f.*

A lower *NRMSE* value indicates a higher accuracy in the recovered data. NRMSE is the most appropriate metric due to the facts that the error of the postdiction value is calculated for specific values (i.e., $recall = 1$) and allows the comparison between datasets or models with different scales.

Parameters: In all experiments the simulation parameters were configured as follows: (i) decay factor $f = 50\%$ (indicating the percentage on which we execute the LSTM); (ii) the ML model is LSTM and the number of neurons 16 x 16. The influence of each of those parameters on the proposed approach is investigated individually in Experiment 2 by fixing the rest of the parameters accordingly.

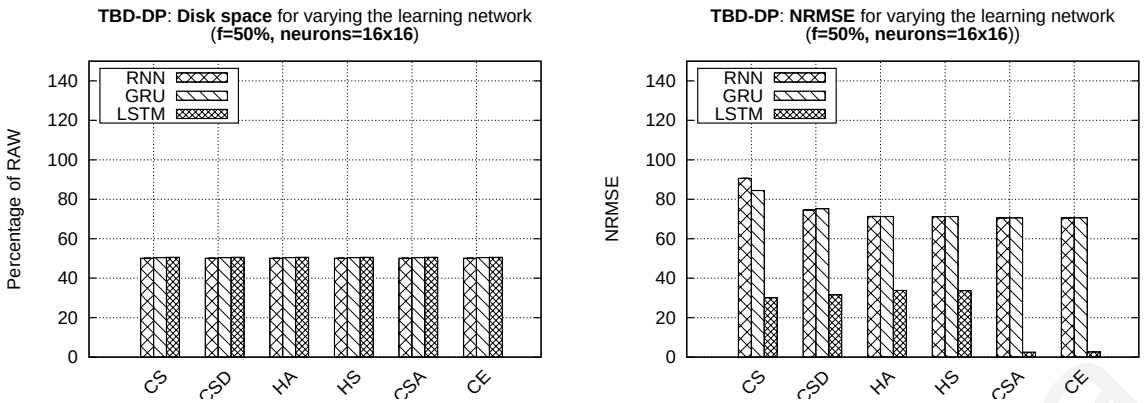

Figure 4.8: Control Experiment - Learning Models: examining the storage capacity S and NRMSE *of the proposed TBD-DP approach while combined with various ML models.*

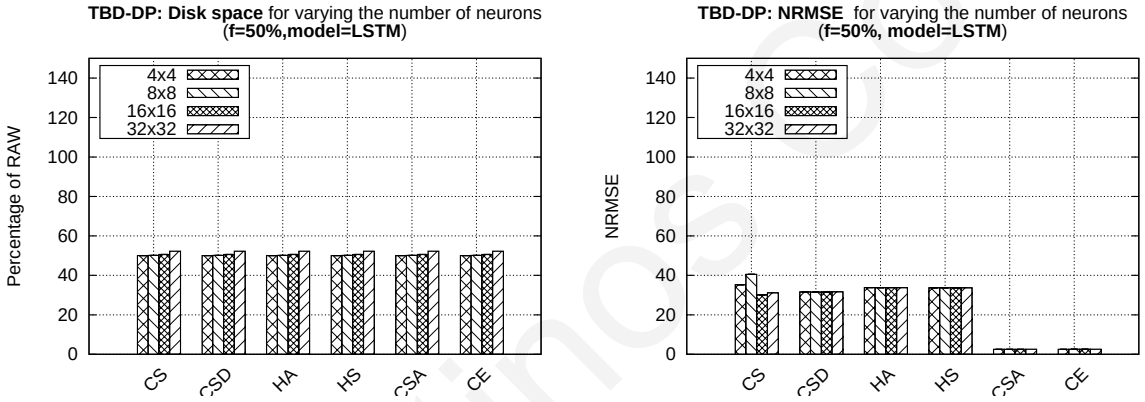

Figure 4.9: Control Experiment - Number of neurons in LSTM: examining the storage capacity S and NRMSE *of the proposed TBD-DP approach while varying the number of neurons.*

4.5.2 Experiment 1: Performance Evaluation

In the first experiment, we evaluate the performance of the proposed *TBD-DP* operator against all four algorithms and over all datasets introduced in Section 4.5.1, with respect to space capacity (as a percentage to the RAW data) and accuracy (in terms of *NRMSE* on the decayed set of data).

Figure 4.6 clearly demonstrates the trade-off between the space capacity S and the *NRMSE* objectives on the results of the baseline approaches, since *RAW* (no decaying) approach obtained the worst possible $S = 100\%$ of the whole dataset, and the lowest error $NRMSE = 0$. In contrast, the *RANDOM* (almost all data are decayed) approach obtained the best possible $S = 50\%$ of the whole dataset and the worst *NRMSE* ≈ 100 on the decayed dataset, for a decaying factor $f = 50\%$. The results of the three other approaches appear in between the results of the two baseline approaches. The proposed *TBD-DP* operator, however, provides around 25% and 50% better space

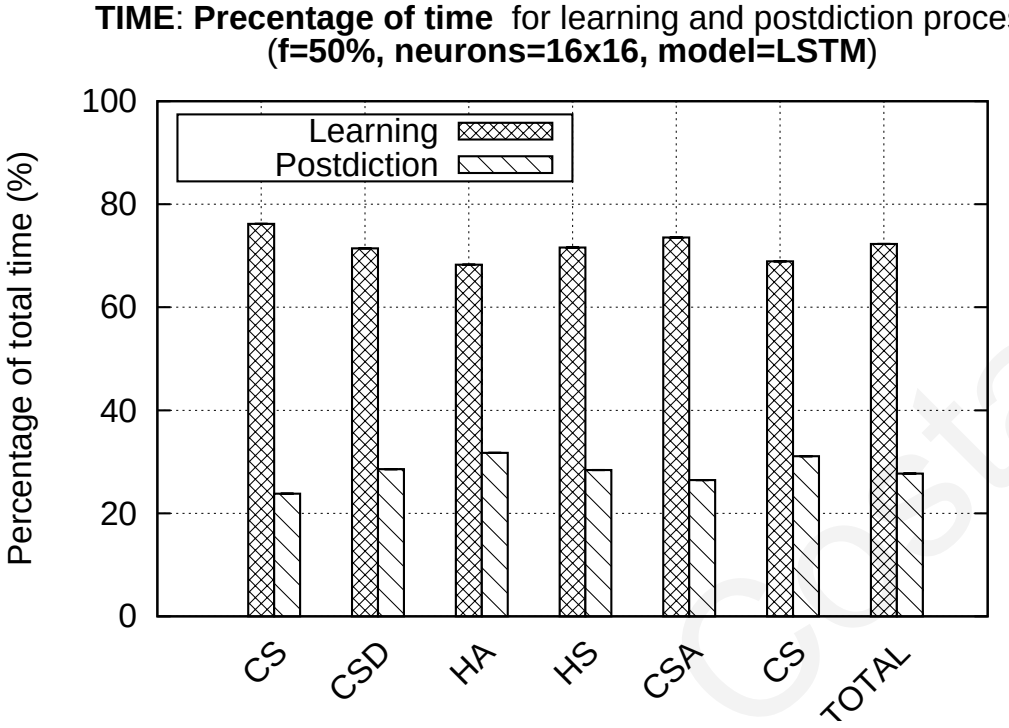

Figure 4.10: Performance Evaluation: TBD-DP evaluation in terms of time percentage for the decayed set of data in all datasets.

capacity S compared to *COMPRESSION* and *SAMPLING* approaches, respectively. This is due to the fact that the additional space required by the set of LSTM models is much less than the sample set of *SAMPLING* and the compressed decayed dataset of *COMPRESSION*.

In terms of *NRMSE*, the *TBD-DP* outperforms the *SAMPLING* approach by 50%, on average, in all datasets. The *COMPRESSION* approach provides an optimal *NRMSE* = 0, since it does not apply any prediction on the decayed data, but recovers them via decompression, when requested. The *COMPRESSION* approach however, can not be customized to achieve an even lower disk space occupancy. On the other hand, the *TBD-DP* can be configured, through its f parameter, to achieve a space occupancy that will fit the space budget of the application. This particular parameter will be investigated in the next experiment.

Figure 4.10 shows the time for the whole process of the *TBD-DP* to finish including the postdiction process. The postdiction process takes much less time in comparison with learning process due to the LSTM network chain. The preprocessing step takes a huge part of the whole process due to network and disk IO.

4.5.3 Experiment 2: Control Experiments

In Experiment 2, we examine the influence of several control parameters on the performance of the proposed *TBD-DP* in terms of S and *NRMSE*. Specifically, we vary the decay factor (f), the ML models and the number of neurons on LSTM.

Figure 4.7 shows how the decaying factor f, and consequently the amount of data that will be decayed and represented by LSTM models, affect the S and *NRMSE* of the proposed *TBD-DP* operator. The results show that the storage capacity required by the *TBD-DP* decreases as the decaying factor increases, which is reasonable due to the fact that the highest f is, the more data need to be decayed and therefore more disk space will be released. The accuracy of the proposed *TBD-DP* however, is not influenced, since *NRMSE* remains almost the same for all decaying factors, in most datasets. This shows the scalability and generalizability of the proposed approach, which is not influenced from the increase on the decaying dataset size. It is also important to note that the variations on the *NRMSE* obtained by *TBD-DP* between the datasets is mainly due to the different characteristics of each dataset.

Figure 4.8 examines the performance of the *TBD-DP* operator in terms of S and *NRMSE* when combined with three different ML models, namely, the traditional *Recurrent Neural Network (RNN)*, the *Gated Recurrent Unit (GRU)* [104] and the *Long Short Term Memory (LSTM)* that is finally adopted by the proposed approach. The results show that *TBD-DP* maintains a similar storage capacity for different learning models, with a slight increase (about 1%) when the LSTM model is used. In terms of *NRMSE*, however, the *TBD-DP*+LSTM combination clearly outperforms the other two combinations providing around 75% less error, on average.

Finally, Figure 4.9 examines how the number of neurons of the LSTM model influences the *TBD-DP*'s performance. The results support our previous observations on the scalability and generalizability of the proposed *TBD-DP* approach. The increase on the number of neurons slightly influences the *TBD-DP* in terms of storage capacity, since the required space slightly increases. This is reasonable since the increase on the number of neurons results in "bigger" models that require more disk space to be stored. The additional required space, however, is almost negligible compared to the disk space needed to store the actual data before decaying. In terms of *NRMSE*, the increase on the number of neurons does not influence the performance of the *TBD-DP* operator, since *NRMSE* remains almost the same while varying this control parameter.

4.6 Related Work

Sampling refers to the process of randomly selecting a subset of data elements from a relatively large dataset. Sophisticated techniques, such as Bernoulli and Poisson sampling, choose data elements using probabilities and statistics. Chaudhuri et al. [105] proposed *stratified sampling* where the probability of the selection is biased. In order to encounter the big data sampling issue, Zeng et al. [106] implemented G-OLA, which is a model that generalizes online aggregation in order to support general OLAP queries utilizing delta maintenance algorithms. Particularly, BlinkDB [107] allows users to choose the error bounds and the response time of query using dynamic sampling algorithms. SciBORQ [108] is a framework that allows the user to choose the quality of the query result based on multiple interesting data samples called impressions.

Several works have adapted the sampling processes to create synopsis of data in order to achieve low response time for ad-hoc queries [108]. Data sketches [98] are compact data structures that enable to efficiently estimate the count of occurences in massive data (contrary to Bloom filters, it encodes a potentially massive number of item types in a small array). Additionally, Wei et al. proposed persistent sketches that can answer queries at any prior time [109] and have the ability to merge in order to answer a generalization query [110].

4.7 Summary

In this chapter, we present a novel decaying operator for Telco Big Data (TBD), coined *TBD-DP (Data Postdiction). TBD-DP* relies on existing ML algorithms to abstract TBD into compact models that can be stored and queried when necessary. Our proposed *TBD-DP* operator has the following two conceptual phases: (i) in an offline phase, it utilizes a LSTM-based hierarchical ML algorithm to learn a tree of models (coined *TBD-DP* tree) over time and space; (ii) in an online phase, it uses the *TBD-DP* tree to recover data within a certain accuracy. In our experimental setup, we measure the efficiency of the proposed operator using a ∼10GB anonymized real telco network trace and our experimental results in Tensorflow over HDFS are extremely encouraging as they show that *TBD-DP* saves an order of magnitude storage space while maintaining a high accuracy on the recovered data.

Chapter 5

Crowdsourcing Emergency Data in Non-Operational Cellular Networks

In overloaded or partially broken (i.e., non-operational) cellular networks, it is imperative to enable communication within the crowd to allow the management of emergency and crisis situations. To this end, a variety of emerging short-range communication technologies available on smartphones, such as, Wi-Fi Direct, 3G/LTE direct or Bluetooth/BLE, are able to enable users nowadays to shape point-to-point communication among them. These technologies, however, do not support the formation of overlay networks that can be used to gather and transmit emergency response state (e.g., transfer the location of trapped people to nearby people or the emergency response guard.) In this chapter, we develop techniques that generate the *k-Nearest-Neighbor (kNN)* overlay graph of an arbitrary crowd that interconnects over some short-range communication technology. Enabling a kNN overlay graph allows the crowd to connect to its geographically closest peers, those that can physically interact with the user and respond to an emergency crowdsourcing task, such as seeing/sensing similar things as the user (e.g., collect videos and photos). It further allows for intelligent synthesis and mining of heterogeneous data based on the computed kNN graph of the crowd to extract valuable real-time information. We particularly present two efficient algorithms, namely *Akin+* and *Prox+*, which are optimized to work on a resource-limited mobile device. We use Rayzit, a real-world crowd messaging framework we develop, as an example that operates on a kNN graph to motivate and evaluate our work. We use mobility traces collected from three sources for evaluation. The results show that *Akin+* and *Prox+* significantly outperform existing algorithms in efficiency, even under a skewed distribution of users.

Figure 5.1: (left) Large crowd protesting in Syria (Reuters 2014), (right) Woman using her mobile while waiting for help in China floods (Reuters 2012).

5.1 Introduction

In the age of smart urban and mobile environments, the mobile crowd generates and consumes massive amounts of heterogeneous data [24, 111]. Such streaming data offer the potential of enhanced science and services, such as emergency and crisis management services, among others. The availability of such services is specifically important in scenarios where a *cellular network becomes non-operational*.

A cellular network is deemed *non-operational* when there is no (sufficient) network connectivity. This might happen due to damage caused by a disaster (e.g., major flooding), or due to overloading caused by an unexpectedly large crowd trying to access telecommunication services simultaneously, e.g., consider the connection problems mobile users have faced during public celebrations of New Year's Eve.

Each cellular tower has a limited capacity of users it can service simultaneously. Specifically, each cellular tower has a limitation on its communication bandwidth to the carrier (backhaul bandwidth), a limitation on the aggregate bandwidth capacity offered by the spectrum and protocol used for wireless communication, and a limitation on the capacity of the network gears [112].

The following are some real-world scenarios making a network non-operational. These are cases where emergency and crisis management services are needed the most.

Ad-Hoc Event Services Large ad-hoc events can be cultural festivals (e.g., Woodstock, Old Car enthusiast gatherings), sporting events, conventions and fairs, ad-hoc demonstrations (e.g., Occupy Wall Street 2011) and ad-hoc protests (e.g., Egypt 2013, Syria 2014, Romania 2014, Hong Kong 2014) as seen in Figure 5.1 (left). *In such*

scenarios, being able to monitor and provide communication within the crowd can aid organizing authorities to better manage the crowd[1] *and prevent lethal crowd disasters*[2,3]. Additional services can also be applied, like new entertainment services[4] and crowd-games[5].

Crowdsourced Emergency Response During a disaster, mobile users can be both victims and rescuers involved not only in receiving but also providing help from/to their neighboring peers (see Figure 5.1 (right)). *In this scenario, providing communication means within the crowd and streaming information to the first response team is vitally important because it can aid in organizing the crowd, allocating peer-to-peer aid and experts, and distributing tools and medicine optimally.* This would enable better management of the disaster monitoring and response cycle, where citizens can get involved in decision making, data acquisition, and advanced planning[6]. In a real world example, it was the citizen's joint efforts to map the 2012 floods in China that materialized faster and more accurately than that government-sanctioned map[7].

When the cellular network is non-operational, users can not rely on online services for their whereabouts, well-being and communication. Meanwhile, the authorities may not be able to receive all the information needed for intelligent synthesis in order to enable advanced services. In other words, the crowd is not able to generate possibly valuable information (e.g., sensors, tweets) and the authorities are not able to collect this information. Consider the example of a disaster response, the authorities need to operate in four phases: (i) rapid and effective actions to avoid making the emergency worse; (ii) implementation in the field; (iii) evaluation of the results; and (iv) personnel management in emergencies. The right situational awareness is the key for decision making in such cases. The crisis management operator needs to have the right tools to communicate with the affected citizens. The operator has to evaluate and monitor the situation in order to learn and optimize operations in real time. It would be an omission that could lead to the loss of human lives, if technology could not support the crowd to generate data (e.g., reporting locations, victims using communication and

[1]WorkingWithCrowds, Online: http://www.workingwithcrowds.com/
[2]Love Parade disaster, Online: http://goo.gl/2FpIbm
[3]Hillsborough disaster, Online: http://goo.gl/xvLRlc
[4]Opphos, Online: https://www.sics.se/projects/opphos
[5]CrowdControlGames, Online: http://crowdcontrolgames.com/
[6]Insight Project EU, Online: http://www.insight-ict.eu/
[7]Eric Blattberg "The Crowd Maps Beijing Floods" 2012, Online: http://goo.gl/0DHZ4v

social services to spread their situation) and the crisis management operator to collect and process this data during these phases.

It is imperative, therefore, to be able to create some overlay network that would connect cellular users by exploiting any available device-to-device short-range communication technology (e.g., Wi-Fi Direct, 3G/LTE direct or Bluetooth/BLE). It is equally important to ensure that this overlay network is *operational*, therefore hot-spots (e.g., too many devices trying to connect to each other), bottlenecks and disconnected components need to be avoided. For this reason we opt for using a kNN overlay graph to configure a communication network, since it guarantees an upper limit of connections per user and according to the crowd size n we can easily set the parameter k to guarantee that the graph is fully connected. Particularly, it leads to connected topologies if the degree $k > log_2 n$ [113]. In the Emergency Response scenario for example, the location of the users in the crowd can be first collected using a breadth-first-search broadcast over Wi-Fi Direct [114, 115] and then the computation of kNN overlay network can take place with the propositions in this work, determining which mobile nodes will connect to each other.

Specifically for mobile environments, valuable information can be mined based on the relations within k geographically nearest neighbors (kNN) [116–120]. Similar to a graph formed by social relationships, a kNN graph is formed by connecting each user to its geographically nearest neighbors. Such a graph can be mined and queried to produce valuable information, e.g., link prediction, shortest hop-distance, large connected communities, socialization suggestions, most central people, most influential people, etc [121].

In this chapter, we develop centralized techniques that allow for the fast computation of the kNN graph in-situ on a mobile device for *non-operational* cellular network scenarios. With the kNN graph at hand, we can then create an overlay network that connects each victim with its k geographically closest users and use this to enable a variety of advanced services. The efficiency of the algorithm allows for frequent recomputations in order to adapt to the movement of the crowd. Our solution however is not designed as a continuous operator, where prior network state is utilized in subsequent network formations, mainly because of the transient network structure of the crowd. On the contrary, we opt for a stateless solution that is re-computed in-situ in an ongoing manner (e.g., every minute). We use Rayzit [1][8], a real-world crowd messaging

[8]Rayzit – Rayz your message, Online: https://rayzit.cs.ucy.ac.cy

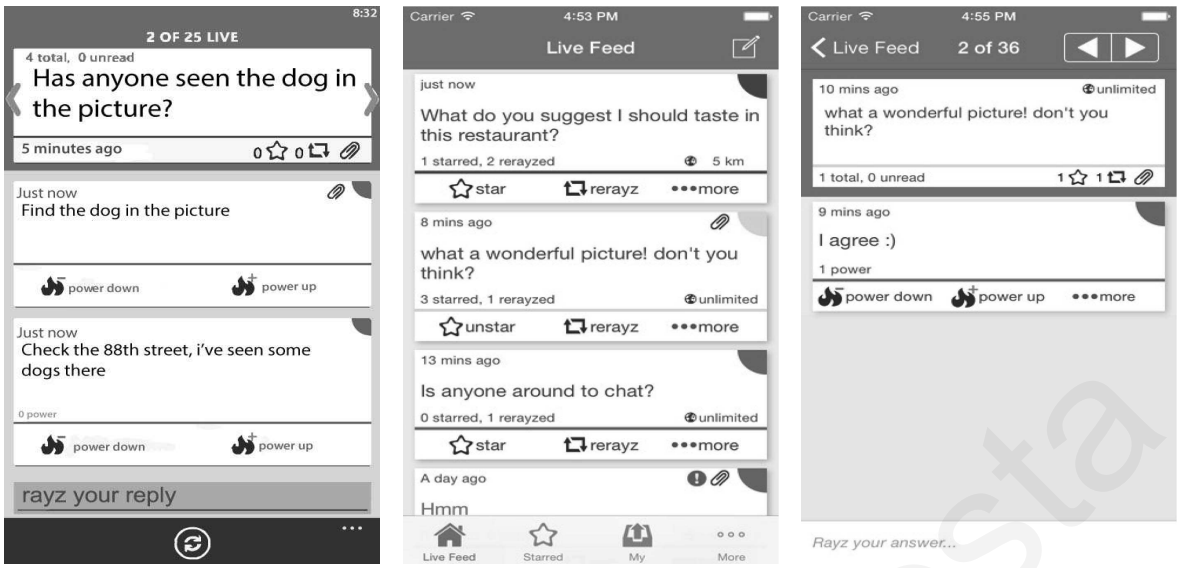

Figure 5.2: Our Rayzit [1] crowd messenger enabling users to interact with their k geographic Nearest Neighbors.

framework we develop, as an example that operates on a kNN graph to motivate and evaluate our work (see Figure 5.2).

In our previous work, we have presented a centralized algorithm, called Proximity [122], which deals with All k Nearest Neighbors (AkNN) queries on a server that can collect the geographic coordinates of users on an ongoing basis. In this work, we re-focus the problem formulation by tackling the AkNN query-processing problem in emerging short-range communication overlay topologies. Particularly, we deal with AkNN computations directly on a smartphone of a crisis management operator that physically resides on the place of an emergency event where an overloaded or partially broken cellular network may exist. Our new contributions are summarized as follows:

- We adapt the previously proposed Proximity algorithm to operate in scenarios where the cellular network base stations are not operational. Particularly, we apply the following improvements: (i) we deploy a pre-processing step that partitions the space using an equi-width grid and implement the *Proximity* algorithm on top of this partition; (ii) we propose an optimization that implements a compact bound for the candidate set and achieving reduced CPU time. This new bound still guarantees correctness of results; (iii) we propose a bulk processing method for the construction of the candidate set that trades slightly higher memory usage for smaller CPU times; and (iv) propose a new pruning heuristic for the final search phase. This heuristic is based on the fact that the candidates

are already ordered. As a result, the candidate bound can be decreased as the nearest neighbors for a specific user are found while the candidate set is scanned.

- We provide an analytical study for the performance, scalability and correctness of our algorithm, showing that it can perform very well independent of the deployment scale and the distribution of input objects.

- We conduct an extensive experimental evaluation that validates our analytical results and shows the superiority of our propositions over competitive algorithms. Particularly, we use three different datasets to test implementations of proposed algorithms and find significant performance improvements.

The remainder of the chapter is organized as follows. Section 5.5 provides the related work on AkNN query processing. Section 5.2 provides our problem definition, system model and desiderata. Section 5.3 presents the phases and algorithms of our AkNN framework, and analyzes correctness and complexity. Section 5.4 presents an extensive experimental evaluation and Section 5.6 concludes the chapter.

5.2 System Model

This section formalizes our system model and design principles. Our main notation is summarized in Table 5.1.

The *k Nearest Neighbors (kNN)* of an object o from some dataset O, denoted as $kNN(o, O)$, are the k objects that have the most similar attributes to o [25]. Formally, given objects $o_a \neq o_b \neq o_c$ and $\forall o_b \in kNN(o_a, O)$ and $\forall o_c \in O - kNN(o_a, O)$, it always holds that $dist(o_a, o_b) \leq dist(o_a, o_c)$[9].

An *All kNN (AkNN)* query, viewed as a generalization of the basic kNN query, computes the $kNN(o, O)$ result for every $o \in O$ and has a quadratic worst-case bound. An AkNN search can alternatively be viewed as a *kNN Self-Join* operation: *Given a dataset O and an integer k, the kNN Self-Join of O combines each object $o_a \in O$ with its k nearest neighbors from O, i.e.,* $O \ltimes_{kNN} O = \{(o_a, o_b) | o_a, o_b \in O \text{ and } o_b \in kNN(o_a, O)\}$.

Research Goal. Given a set of objects O with their locations in a bounding area, a resource-poor query processor QP computes the AkNN result of O by minimizing *CPU time*.

[9] In our discussion, *dist* can be any L_p-norm, such as Manhattan (L_1), Euclidean (L_2) or Chebyshev (L_∞).

Table 5.1: Summary of Notation

Notation	**Description**
o, O, n	Object o, set of all o, $n = \|O\|$
$kNN(o, O)$	k nearest neighbors of o in O
$AkNN(O)$	k nearest neighbors of every o in O
$dist(o_a, o_b)$	L_p-norm distance between o_a and o_b
c, C, O_c	Cell c, set of all c, objects in c
S_c	Candidate Set of c

Let O be a set of n mobile users moving in a planar area, denoted by their current locations, which can either be obtained at a fine granularity (e.g., GPS and RadioMaps) or coarse granularity (e.g., Cell_ID and Wi-Fi_ID databases). Assume that there is a mobile device, denoted as QP (Query Processor), which collects the locations of all users in user set O. Our main desiderata is to minimize CPU time on QP, given that this determines the energy consumption of QP and also determines how frequently the kNN networks can be re-computed.

5.2.1 Emergency Network Model

In a disaster scenario with a non-operational cellular network, the first task is to construct an ad-hoc communication network that would enable information routing within the crowd. This can be done using existing technologies (e.g., Wi-Fi Direct [112]) and algorithms [114, 123], assuming that each user can directly communicate with peers that are within a certain communication radius. With the initial ad-hoc network in place, each user can send out a request to construct a query routing tree, using techniques like [61, 124–126], in order to collect the location of the users in the crowd and compute the kNN graph. We assume that only one request completes, e.g., by letting the request with the oldest initiation timestamp dominate. Therefore, one user within each connected overlay network retrieves the needed locations and computes the kNN graph.

5.3 Centralized AkNN Query Processing

In this section, we introduce *Proximity*, a centralized AkNN query processing framework [122] we developed previously. We then describe the adaptations undertaken to make it functional for non-operational cellular network scenarios using a grid partitioning. Subsequently, we also propose two optimizations, namely *Prox* and *Akin*, which reduce the CPU time for the AkNN computation. We conclude with some further improvements upon *Prox* and *Akin*, coined *Prox+* and *Akin+*, which are founded on more powerful pruning heuristics than their basic counterparts.

5.3.1 Background on Proximity

The *Proximity* framework is designed in such a way that it is: (i) *Memory-resident*, since the dynamic nature of mobile user makes disk resident processing prohibitive; (ii) *Specifically designed* for *highly mobile* and *skewed distribution* environments performing equally well in congested and sparsely-deployed scenarios; and (iii) *Infrastructure-ready*, since it does not require any additional infrastructure or specialized hardware.

The original *Proximity* algorithm exploits the natural geographic partitioning that is provided by the *Network Connectivity Points (NCP)* of a cellular network (e.g., cell towers). This partitioning allows for the derivation of kNN candidate sets, which provide the AkNN answer-set in linear time. Particularly, for every timestep *Proximity* works in two phases: In the first phase one $k+$-heap data structure is constructed per *NCP*, using the location reports of users (the $k+$-heap will be discussed more thoroughly in 5.3.2). In the second phase, the k-nearest neighbors for each user are determined by scanning the respective $k+$-heap and the results are reported back to the users.

5.3.2 Proximity with Grid Partitioning

In this section we adapt the *Proximity* algorithm such that it is effective even under a non-operational cellular network, where there are no NCPs to naturally partition the space. In Algorithm 1 we outline the proposed solution that operates over a grid partitioning of the space into *equi-width* cells C. In Algorithm 2 we discuss in more detail the specifics of the insertion procedure of the basic underlying $k+$-heap data structure.

Algorithm 1 (Outline): Each cell $c \in C$ contains a disjoint subset $O_c \subset O$ of objects.

Algorithm 3 . Proximity with Grid Partitioning

Input: User locations O, set C of all grid cells

Output: *kNN* answer-set for each user in O

```
1: for all c ∈ C do
2:     initialize S_c ▷ Initialize our k+-heap
3: end for
4: for all o ∈ O do ▷ Phase 1: build k+-heap
5:     for all c ∈ C do
6:         insert(o, S_c)
7:     end for
8: end for
9: for all o ∈ O do ▷ Phase 2: scan k+-heap
10:    kNN_0 = ∅ ▷ Conventional k-max heap
11:    c ← o.cell
12:    for all o' ∈ S_c do
13:        if o' is a kNN of o then
14:            update(kNN_o, o')
15:        end if
16:    end for
17: end for
```

Algorithm 3 computes a correct candidate set S_c for each cell $c \in C$ by constructing a $k+$-heap data structure for each cell (lines 1-8). In the second phase, the kNN for each user $o \in O$ are determined by scanning the respective $k+$-heap and computing $kNN(o, S_c)$ (lines 9-17).

The $k+$-heap consists of three separate data structures (see Figure 5.3): (i) an unordered list of internal objects lying inside the boundary of c (coined O_c); (ii) a conventional k-max-heap (coined K_c) capturing the closest external candidates; and (iii) an ordered list of external boundary objects (coined B_c), which are necessary besides K_c for the correct computation of any AkNN query.

The K_c and B_c structures are defined according to definitions 5.3.1 and 5.3.2, respectively. In Definition 5.3.3, we conclude with a formal definition of the $k+$-heap (coined *Candidate Set*, S_c).

Definition 5.3.1 (K_c, k-max-heap of cell c)**.** Given a set of external users $O - O_c$, which is ordered with ascending distance $dist(o, c)$ to the border of cell c, set K_c consists of the top-k elements of this set.

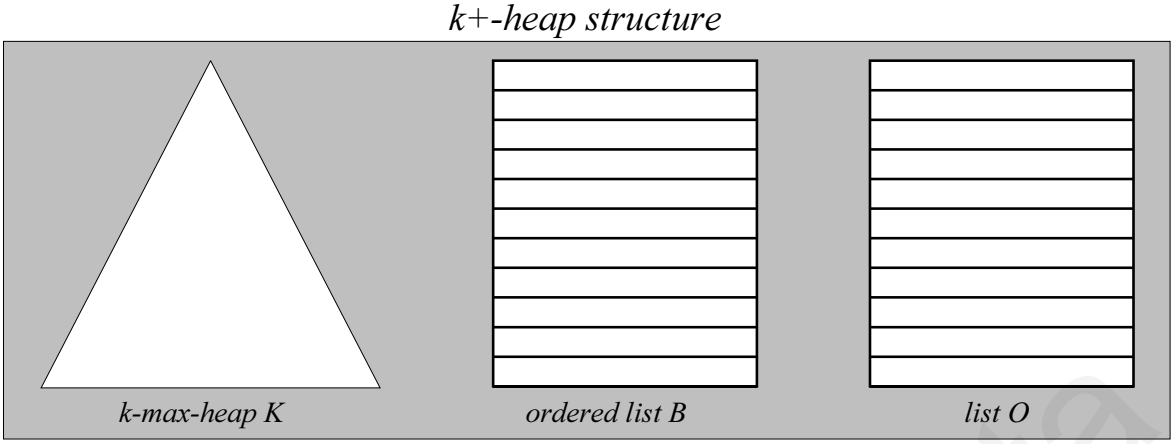

Figure 5.3: A visualization of a k+-heap (denoted as S_c) for a specific cell c, comprises of three structures: O_c, K_c and B_c.

Definition 5.3.2 (B_c, Boundary Set of cell c)**.** Given a cell c and its k^{th} closest external user to the border of c, set B_c consists of all users $o \in O - (O_c \cup K_c)$ with distance $dist(o, c) < dist(kth_c, c) + diag_c$ from the border of c. In other words B_c consists of all users $o \in O$ with distance $dist(kth_c, c) < dist(o, c) < dist(kth_c, c) + diag_c$.

Definition 5.3.3 (S_c, Candidate Set of cell c)**.** Given a cell c and its O_c, K_c and B_c sets, the candidate set S_c of c consists of users $o \in O_c \cup K_c \cup B_c$.

Performance Analysis of Prox: Assuming a grid size of $\sqrt{n}$ cells and uniform data distribution of n objects, Algorithm 3 runs in $O(n^{1.5}\ log\sqrt{n})$ time. Particularly, the loop at line 1 runs in $O(\sqrt{n})$. The loop at line 4 runs in $O(n^{1.5} log\sqrt{n})$, given that the insertion procedure of the heap runs in $O(log\sqrt{n})$. Finally, the last loop in line 9 runs in $O(n^{1.5})$, given that each $k+$-heap has $O(\sqrt{n})$ users under a uniform distribution assumption. In the worst-case distribution, where all users fall in a single cell this algorithm runs in $O(n^2)$ time.

Algorithm 2 (Insertion): We now discuss in more detail the specifics of the insertion procedure of the $k+$-heap. When inserting a new element o_{new} into the $k+$-heap of c, we distinguish among four cases (see Algorithm 4): (i) o_{new} is covered by c and belongs to set O_c (line 2), (ii) o_{new} belongs to set K_c (line 4), (iii) o_{new} belongs to set B_c (line 11), or (iv) o_{new} does not belong to the candidate set $S_c = O_c \cup K_c \cup B_c$ of cell c (line 13). In case (i) the element is inserted into the O_c list. In case (ii) we need to insert o_{new} into heap K (line 5) and move the current head kth_c from K to the boundary list B (lines 7-8). This yields a new head kth'_c in K (line 9). Every time the kth_c changes, the boundary list B needs to be updated, since it might need to evict some elements

Algorithm 4 . $k+$-heap: Insert(o_{new}, c)

Input: Object to be added o_{new}, Cell c of $k+$-heap

Output: S_c updated

```
1: kth_c ← head(K_c)
2: if contained_inside(o_new, c) then
3:     insert(o_new, O_c)
4: else if dist(o_new, c) < dist(kth_c, c) then
5:     insert(o_new, K_c)
6:     if K heap has more than k elements then
7:         kth_c ← pophead(K_c)
8:         insert(kth_c, B_c)
9:         Update_boundary(head(K_c))
10:    end if
11: else if dist(o_new, c) < dist(kth_c, c) + diag_c then
12:    insert(o_new, B_c)
13: else
14:    discard o_new
15: end if
```

according to Definition 5.3.2. In case (iii) we insert o_{new} into the ordered boundary list B (line 12). Note that the sets K_c and B_c are formed as elements are inserted into the $k+$-heap. The first k elements inserted in the empty $k+$-heap define the K_c set. In case (iv) the element is discarded.

5.3.3 Prox: An Optimized Candidate Set Bound

Our initial *Proximity* algorithm has a suboptimal bound determining the candidate set per cell. Here we propose a tighter bound, coined *Prox*, which is founded on the observation that the K'_c set inside a $k+$-heap could have been constructed from objects that reside both inside a cell and outside a cell, as opposed to the original *Proximity* algorithm, where the K_c set emerged from objects residing outside a given cell only. Particularly, we use the following definition to define the optimized candidate set bound:

Definition 5.3.4 (K'_c, smaller K_c)**.** Given a set of any user in O, which is ordered with ascending distance $dist(o, c)$ to the border of cell c, set K'_c consists of the top-k elements of this set.

Note that the difference of K_c to K'_c is that the former definition includes only users

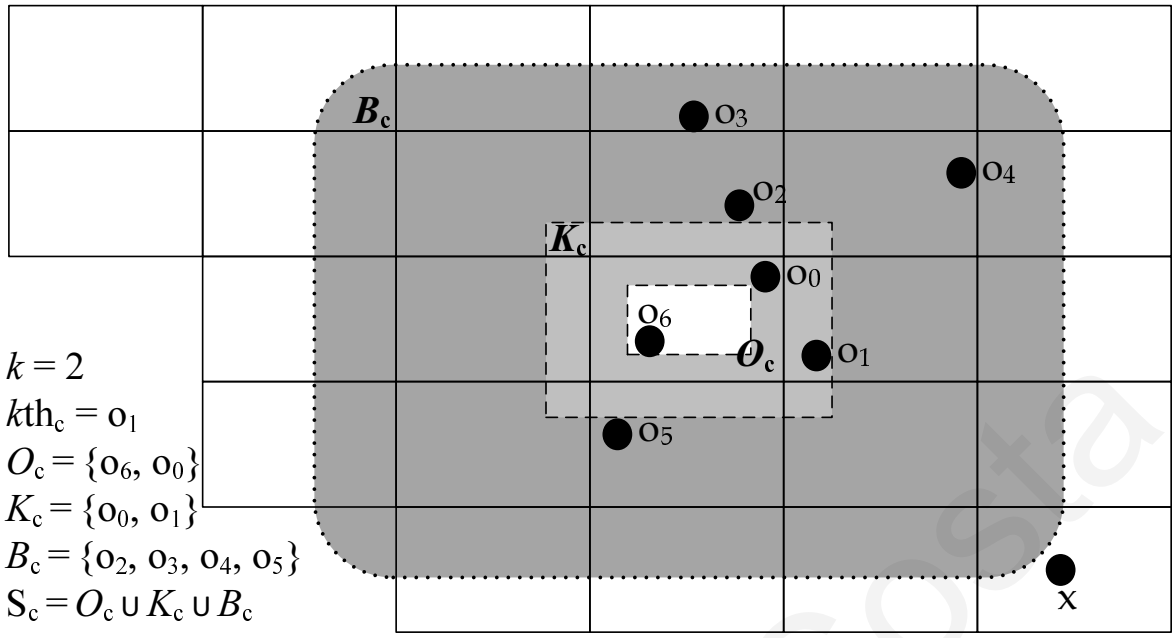

Figure 5.4: (Running Example) The construction of S_c with k=2. The candidate set S_c of c is $\{o_0, o_1, o_2, o_3, o_4, o_5, o_6\}$ and is represented by the area within the dotted line with the rounded corners. Set S_c includes all users O_c inside c (solid line cell), users inside K_c the lighter square ring and the users B_c inside the darker ring. Any node outside S_c (e.g., user x) is guaranteed NOT to be a kNN *of any user inside cell c. The 2-nearest neighbors for the nodes in c are* $\text{kNN}(o_0) = \{o_1, o_2\}$ *and* $\text{kNN}(o_6) = \{o_5, o_0\}$.

outside cell c. This does not allow for a tight bound and a minimum cut-off threshold θ_c. Particularly, by expanding the scope of K'_c, immediately makes B'_c smaller or equal in size to B_c. This effectively allows Prox to process a smaller search space, as the S'_c set now comprises of $B'_c \cup O_c \cup K'_c$, as opposed to $B_c \cup O_c \cup K_c$. Another way to present the difference is to mention that with Prox, it holds that $dist(kth'_c, c) \leq dist(kth_c, c)$.

Running Example of Prox: We shall now present a running example of Prox in Figure 5.4. Consider that the following object locations arrive at QP: $O = \{o_0, o_1, o_2, o_3, o_4, o_5, o_6, o_x\}$ Every object is again inserted into every $k+$-heap on the QP (see Algorithm 3, lines 1-5). The order in which the objects are inserted into a $k+$-heap does not affect the correctness of the candidate set. For our example, assume that the objects are inserted in the order seen in the first column of Table 5.2. For every insertion we can see the contents of S'_c in the same Table (i.e., last three columns). For simplicity, we only follow the operation on the S'_c of cell c (that similarly applies to all cells).

When object o_4 is inserted into S'_c it is added to the heap K'_c, which records the closest objects around the cell border of c (both internal and external objects to c). The same logic applies to the next object o_x. At this point, however, the K'_c heap

Table 5.2: Build-up phase of S_c in Prox as object locations are inserted

Object	**Set K'_c**	**Set B'_c**	**Set O_c**
o_4	$\{o_4\}$	$\{\}$	$\{\}$
o_x	$\{o_x, o_4\}$	$\{\}$	$\{\}$
o_2	$\{o_4, o_2\}$	$\{o_x\}$	$\{\}$
o_3	$\{o_3, o_2\}$	$\{o_4, o_x\}$	$\{\}$
o_1	$\{o_2, o_1\}$	$\{o_3, o_4\}$	$\{\}$
o_5	$\{o_2, o_1\}$	$\{o_3, o_4, o_5\}$	$\{\}$
o_6	$\{o_2, o_1\}$	$\{o_3, o_4, o_5\}$	$\{o_6\}$
o_0	$\{o_0, o_1\}$	$\{o_2, o_3, o_4, o_5\}$	$\{o_6, o_0\}$

gets full (assume it is a 2-max-heap). The third insertion of o_2 into K'_c evicts o_x from K'_c (given that in Figure 5.4, o_2 and o_4 are the closer to the boundary c than o_x). It however does not discard o_x, given that it might still be a good candidate for some hypothetical other object o_y in c (e.g., one that resides on the opposite site of the cell).

Before transferring it blindly to B_c, o_x is checked against the new threshold: $\theta'_c = dist(o_4, c) + diag_c$. Given that o_x is below θ'_c, it is inserted into B_c. Alternatively, o_x would have been discarded. A similar procedure is followed for the next three insertions, i.e., o_3, o_1 and o_5. It is important to mention that on each insertion, θ'_c might become smaller. Every time this happens, objects inside B'_c have to be re-evaluated and discarded accordingly (e.g., o_x is discarded when o_1 is inserted). As a concluding remark, notice that any "inside" object (e.g., o_6 and o_0), is automatically added to O_c but these objects are also considered for K'_c (i.e., in our example only o_0 qualifies to be part of K'_c as it is close to the border c).

Phase 1 of Algorithm 3 is completed and the candidate sets are ready after all objects are inserted into the S_c sets. In phase 2 the server scans a single S'_c for each user o_x, according to the cell o_x is mapped to. For users o_0 and o_6, the server QP scans $S_c = o_2, o_1, o_3, o_4, o_5, o_6$ and finds nearest neighbors $\{o_2, o_1\}$ and $\{o_5, o_0\}$, respectively.

5.3.4 Akin: Bulk Candidate Set Construction without a *k+*-heap

In this section, we introduce an alternative technique, called *Akin*, to reduce the CPU time for the candidate set construction. Our proposition is founded on a bulk con-

Algorithm 5 . *Akin*(O, S_c) Algorithm

Input: Candidate set S_c and set of objects O

Output: S_c updated

1: construct Min Heap H_c from O based on $dist(o, c)$
2: kth_c = extract top k objects from H_c
3: $\theta_c = diag_c + dist(kth_c, c)$
4: **for all** $o \in O$ **do**
5: **if** $dist(o, c) < \theta_c$ **then**
6: $S_c = S_c \cup o$
7: **end if**
8: **end for**
9: $S_c = S_c \cup O_c$

struction of the search space without a $k+$-heap. Particularly, we adopt the linear-time heap construction algorithm proposed by Robert Floyd in [127]. The given algorithm, makes the necessity of breaking our initial search space into three sub-structures unnecessary, given that a single heap is constructed for the complete search space of each cell in linear time.

In Algorithm 5, we present our Akin algorithm for the search space construction of cell $c \in C$. Particularly, we scan once each object $o \in O$ to build a k-min heap H_c based on the minimum distance $dist(o, c)$ between o and the cell-border (line 1). The k objects are then scanned from H_c to determine threshold θ_c of c. We subsequently scan objects $o \in O$ once again to determine the set S_c of objects that satisfy the threshold (lines 4-8). At the end of the execution, we carry out the union of O_c and S_c to derive the updated search space.

Performance Analysis of Akin: Assuming a grid size of $\sqrt{n}$ cells and uniform data distribution of n objects, Algorithm 5 runs in $O(n^{1.5})$ time. Particularly, the heap construction in line 1 takes $O(n)$ using the Floyd algorithm. The subsequent loop in line 4 also takes $O(n)$, consequently the above complexity is $O(n)$ for each cell. For all $\sqrt{n}$ cells the AkNN computation cost is $O(n^{1.5})$. In the worst-case distribution, where all users fall in a single cell this algorithm runs in $O(n^2)$ time.

5.3.5 Internal Pruning of Candidate Set: Prox+ and Akin+

In this section we introduce an internal pruning strategy of the candidate set S_c, which is applicable to both the Prox and the Akin algorithms introduced earlier. We denote

the respective algorithms with the extension '+', if they use this further optimization, i.e., Prox+ and Akin+. The particular strategy can be applied once the S_c search space has been constructed, at which point the algorithm is ready to return the AkNN results for each user.

Particularly, the internal pruning heuristic is founded on the observation that the internal structures of the S_c structure (i.e., O_c, K_c and B_c), can be accessed in a particular order to maximize the possibility of converging early-on with the AkNN result-set for each user. Particularly, the heuristic strategy attempts to refine the pruning threshold θ_c as soon as possible, by beginning the exploration of the S_c set from O_c, then if necessary proceed to K_c and finally, if again necessary, proceed to the exploration of B_c. Our experimental evaluation in Section 5.4 reveals that the internal pruning heuristic provides a significant improvement to both Prox and Akin.

5.4 Experimental Evaluation

To evaluate our proposed algorithms we conduct a set of experiments, using three traces of real datasets, in comparison with existing state-of-the-art algorithms. We run all experiments on a virtual octa-core computing node.

The goal of this evaluation is to compare the overall efficiency of our proposed algorithms with existing state-of-the-art algorithms in centralized AkNN query processing. Efficiency is determined by the running time and, in case of running the algorithm on a mobile device, by the energy consumed. Since our algorithms are centralized, both running time and energy consumed are proportional to the CPU time needed for the computation.

5.4.1 Datasets

In our experiments we use the following realistic and real datasets (depicted in Figure 5.5):

Oldenburg (realistic): The initial dataset is generated with the Brinkhoff spatio-temporal generator [128], including 5K vehicle trajectories in a 25km x 25km area of Oldenburg, Germany. The generated spatio-temporal dataset is then decomposed on the temporal dimension, in order to generate realistic spatial datasets of 100, 1000 and 10K users.

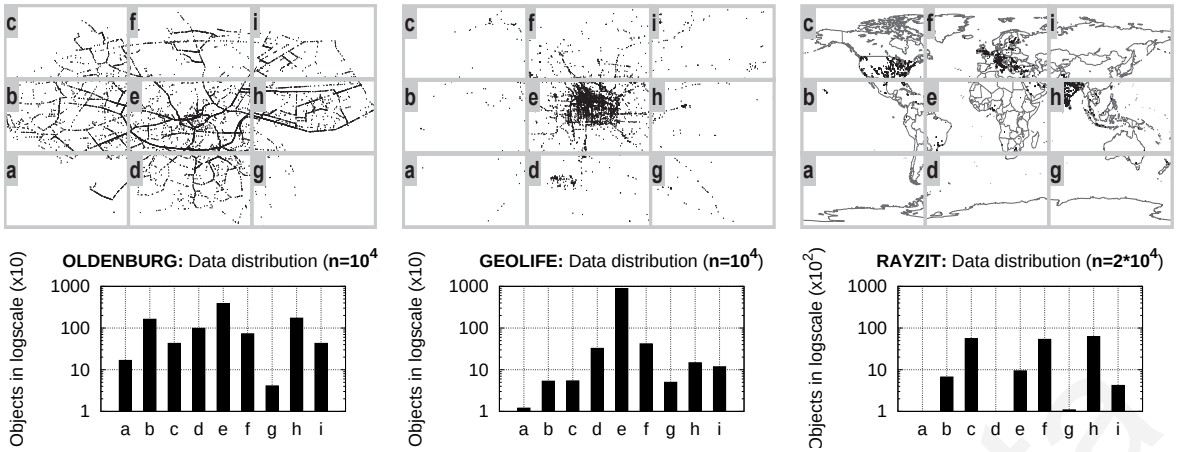

Figure 5.5: Datasets (top row) and population histograms (bottom row) for an indicative 3x3 partitioning.

Geolife (realistic): The initial dataset is obtained from the Geolife project at Microsoft Research Asia [129], including 1.1K trajectories of users moving in the city of Beijing, China over a life span of two years (2007-2009). Similarly to Oldenburg, the generated spatio-temporal dataset is decomposed on the temporal dimension, in order to generate realistic spatial datasets of 100, 1000 and 10K users.

Rayzit (real): This is a real spatial dataset of 20K coordinates captured by our Rayzit service during February 2014. We intentionally do not scale this dataset up to more users, in order to preserve the real user distribution.

Figure 5.5 (second row) shows the population histograms for the three respective datasets, when split into nine equi-width partitions. The standard deviation among the buckets for a total population of 10K objects is: (i) 900 objects in Oldenburg, (ii) 2K objects in Geolife, and (iii) 33 objects for Rayzit.

5.4.2 Evaluated Algorithms

We first evaluate the proposed algorithms in order to experimentally validate the ideas and superiority of our propositions. Our proposed algorithms are:

Proximity: This algorithm is proposed in our previous work [122] for answering AkNN queries in an operational cellular network, which exploits the natural partitioning determined by the Network Connectivity Points (cell towers). We implement *Proximity* on top of the grid partitioning pre-processing step discussed in Section 5.3.2.

Prox: This algorithm implements the optimized candidate set bound, introduced in Section 5.3.3 and founded on the observation that the K'_c set inside a $k+$-heap could

have been constructed from objects that reside both inside a cell and outside a cell.

Akin: This algorithm implements the optimized candidate set construction algorithm, introduced in Section 5.3.4. Note that it does not make use of the k+-heap structure, rather uses a heap constructed in linear time.

Prox+: This is the same algorithm as *Prox* but with the internal pruning strategy described in Section 5.3.5. This optimization allows the final step to terminate earlier.

Akin+: This is the same algorithm as *Akin* but with the internal pruning strategy described in Section 5.3.5. This optimization allows the final step to terminate earlier.

We also take existing state-of-the-art algorithms for answering a kNN query for a single user, including Yu *et al.* [130] and Mouratidis *et al.* [131]. We adapt them to answer an AkNN query. In addition, we compare the adaptation of existing work to our best algorithms.

YPK and ***CPM*:** These methods iteratively enlarge a range search to find the *kNN* for the user (see Figure 5.8b). The search space starts from the cell of the user and iteratively visits neighboring cells until at least k neighbors are found. It is guaranteed that no further neighboring cell can have a user that is closer. For our experiments, we use this adaptation as a baseline for comparison.

In Table 5.3, we summarize the time complexity for each of the above algorithms according to the respective data distribution, where "Best" denotes the uniform distribution and "Worst" the worst distribution that corresponds to each algorithm.

Table 5.3: Algorithm Complexities under Best-case and Worst-case distributions.

Algorithm	Best	Worst
***Proximity*, *Prox*, Prox+**	$O(n^{1.5} log\sqrt{n})$	$O(n^2)$
***Akin*, *Akin*$^+$**	$O(n^{1.5})$	$O(n^2)$
CPM* / *YPK	$O(n^{1.5})$	$O(n^{2.5})$

5.4.3 Evaluation Metrics

CPU time is the metric we use for our evaluation. All algorithms under evaluation are centralized and, thus, do not require any communication. Therefore, CPU time captures both the running time and the energy consumed by the computing node (e.g., mobile device) to run the AkNN algorithm. The CPU times are averaged over five

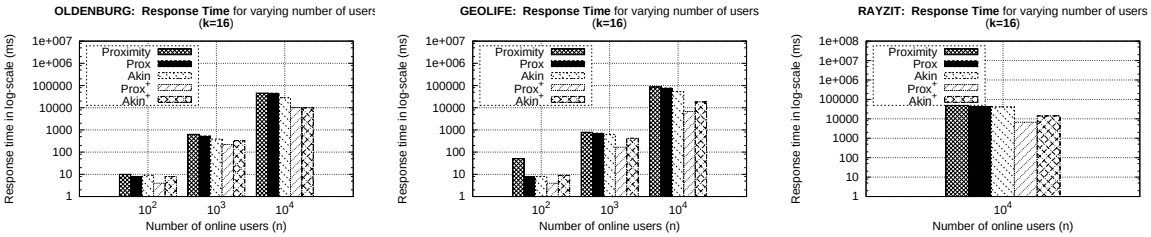

Figure 5.6: CPU time for all algorithms using the datasets: a) Oldenburg; b) Geolife; and c) Rayzit. The plots show that: (i) Proximity without any optimizations has the worst performance; (ii) Internal Pruning (using the "+") has a higher impact on Prox rather than on Akin, making Prox+ the algorithm with the best CPU-time performance; (iii) the more skewed the dataset is (e.g., Geolife) the more improvement the speed-up achieved by our optimizations.

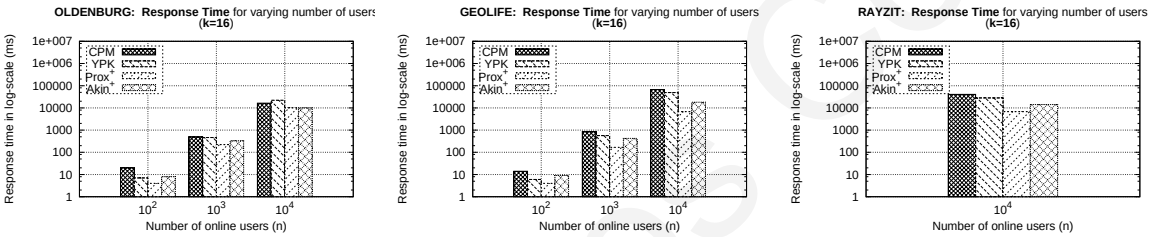

Figure 5.7: CPU time for the best algorithms using the datasets: a) Oldenburg; b) Geolife; and c) Rayzit. The plots show Prox+ and Akin+ outperform adapted state-of-the-art AkNN query processing algorithms YPK and CPM that apply iterative deepening principle rather than bulk computation of the search space.

iterations and are measured in milliseconds. Note that all figures are plotted with the time (y-axis) in *log-scale*, thus the actual difference in efficiency between the algorithms is larger than it visually appears.

5.4.4 Control Experiments

In this experimental series, we evaluate our proposed optimizations. We increase the workload by growing the number of online users (n) exponentially using $n = 10^2, 10^3, 10^4$ for all datasets.

All plots in Figure 5.6 show that Proximity without any optimizations has the worst performance. They also show that using bulk heap construction (i.e., *Akin*) instead of a k+-heap (i.e., *Prox*) we achieve better performance. Looking at the '+' optimization (i.e., the internal pruning strategy), it is evident that $Akin^+$ and $Prox^+$ outperform their counterparts (i.e., *Akin* and Prox) every time.

It is interesting to notice that *Akin*$^+$ does not outperform *Prox*$^+$, even though *Akin* outperforms *Prox*, and *Akin*$^+$ outperforms *Akin*. This is happening because the pruning power inside the candidate set of *Akin*$^+$ is smaller compared to *Prox*$^+$. As described in Section 5.3.4, *Akin* includes also internal objects in the computation of the K_c set resulting in a K_c area that is *narrower and closer* to the cell border. Therefore, when the kNNs of an internal object can not be guaranteed by the objects inside the cell (O_c set), then there is a much higher chance for *Akin*$^+$ to continue the kNN search in the much larger boundary set (B_c) instead of guaranteeing the kNNs by just expanding to the K_c set, as opposed to *Prox*$^+$.

Comparing the various datasets, we conclude that the more skewed the dataset is (e.g., Geolife) the more improvement the speed-up achieved by our optimizations. This stands also when we look at the growing workload. Generally, the proposed optimizations in this chapter always outperform our original algorithm Proximity, specifically for large workloads and skewed datasets where they reach a 10% speed-up (Figure 5.6b).

5.4.5 Comparison Against Existing Work

In this experimental series, we compare our best algorithms (i.e., *Prox+*, *Akin+*) against the state-of-the-art (i.e., *YPK*, *CPM*). Again, we increase the workload of the system by growing the number of online users (n) exponentially using $n = 10^2, 10^3, 10^4$ for all datasets (other than Rayzit, which is kept at 10^4 as explained previously).

Figure 5.7 shows that there are only two instances where a state-of-the-art algorithm (i.e., YPK) outperforms any of our algorithm (i.e., *Akin*$^+$), i.e., for the lightest workloads $n = 10^2$. Looking at the workload, it shows that the our algorithms achieve greater speed-up as the workload increases. Comparing datasets, it is evident that the more skewed the dataset the greater the speed-up achieved by our algorithms.

It can clearly be observed, that the algorithms proposed in this chapter outperform existing approaches for any workload and skewness of dataset. The more skewed the dataset is (i.e., Geolife) the more improvement the speed-up of our algorithms, specifically they reach a 10% speed-up (Figure 5.7b). This is in line with our theoretical comparison in Table 5.3, where the time complexity of the state-of-the-art is greater for the worst-case data distribution.

5.5 Related Work

We provide a summary of existing state-of-the-art research works on neighborhood queries and categorize them according to the characteristics of these queries. Existing works can be classified in spatial data applications and spatio-temporal data applications.

5.5.1 kNN for Spatial Data

kNN search is a classical problem with many centralized algorithms that find applications in computational geometry [132–134] and image processing [135, 136]. An *All kNN (AkNN)* query, which can be viewed as a generalization of the basic kNN query, generates a kNN graph. For large datasets residing on disk (external memory), works like Zhang *et al.* [137], Chen *et al.* [138], and Sankaranarayanan *et al.* [139] exploit possible indices on the datasets and propose algorithms for R-tree based AkNN search. For smaller problems, where data fits inside main memory, early work in the domain of computational geometry has proposed several solutions. Clarkson *et al.* [132] was the first to solve the *ANN* problem followed by Gabow *et al.* [134], Vaidya [140] and Callahan [133]. Given a set of points, a special quad-tree is used in [132–134] and a hierarchy of boxes to divide the data and compute the *ANN* is proposed in [140]. The worst case running time, for both building the needed data structures and searching in these techniques, is *O(nlogn)*, where n is the number of points in the system. For the *AkNN* problem, the algorithms developed in [132, 133] propose an algorithm with *O(kn+nlogn)*, while the algorithm in [140] achieves *O(knlogn)*-time complexity.

5.5.2 kNN for Spatio-Temporal Data

In spatio-temporal data applications the datasets consist of objects and queries that move over time in some Euclidean space. Existing works in this category only tackle the problem of answering a k-nearest neighbor query for a single user over time (*CkNN* query).

For large-scale disk-resident datasets, Tao *et al.* [141], Benetis *et al.* [142], Iwerks *et al.* [143], Raptopoulou *et al.* [144], and Frentzos *et al.* [145] assume that the velocity of the moving objects is fixed and the future position of an object can be estimated. Huan *et al.* [146] assume that there is only some uncertainty in the velocity and direction

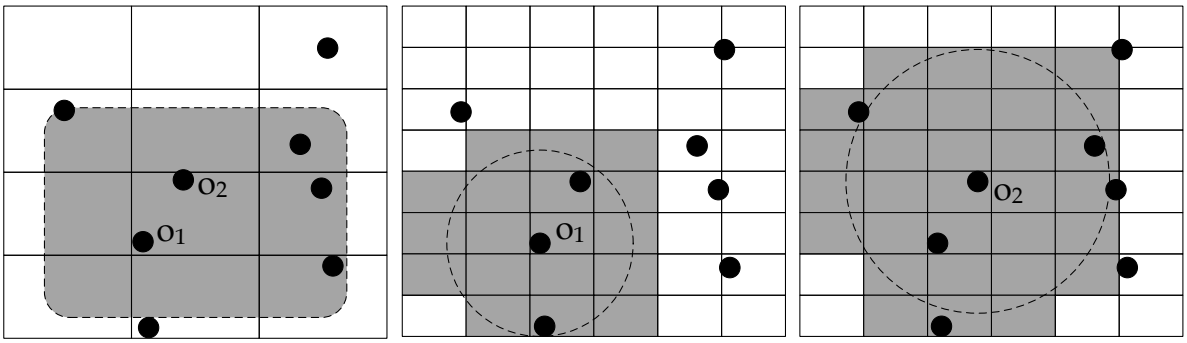

Figure 5.8: a) In Proximity the candidate set is pre-constructed for all users of the same cell (e.g., o_1 and o_2); whereas b) for existing state-of-the-art algorithms the candidate set needs to be iteratively discovered by expanding a ring search for each user separately into neighboring cells.

of the moving objects. Accordingly, they propose algorithms to optimize the case were the future position estimation can also be uncertain. This set of works uses time parameterized R-trees to efficiently search for the nearest neighbors. Kollios *et al.* [115] propose a method to answer *NN* queries for moving objects in 1D space. Their method is based on the dual transformation where a line segment in the native space corresponds to a point in the transformed space, and vice-versa. Xiong *et al.* [147] focus on multiple *kNN* queries and propose an incremental search technique based on hashing objects into a regular grid, keeping CPU time in mind. The main objective of these works on disk-resident data is to minimize disk I/O operations, considering CPU time only as a secondary objective in the best case.

Main-memory processing is usually mandatory for spatio-temporal applications, where objects are highly mobile. The intensity of the location updates is very restrictive for disk-based storage and indexing, and demands optimization in respect to the CPU time. Yu *et al.* [130] (*YPK*) followed by Mouratidis *et al.* [131] (*CPM*) and Hu *et al.* [148] optimize *kNN* queries in a similar fashion as Xiong *et al.* do for disk-resident data. Data objects are indexed by a grid in main-memory (see Figure 5.8) given a system-defined parameter value for the grid size. For each query they both use a form of iteratively enlarging a range search to find the *kNN*. Assuming a grid size of $\sqrt{n}$ cells, their stateless solution has a time complexity of $O(n^{1.5})$ for uniform distributions and $O(n^{2.5})$ for the worst-case distribution, where the search for most of the users needs to be deepened iteratively until it covers most of the space.

5.5.3 Mobile User Community Network

Similar to the motivating examples in the previous section, Konstantinides *et al.* [149] present a distributed search architecture for intelligent search of objects in a mobile social community. Their framework is founded on an in-situ data storage model, where captured objects remain local on smartphones. It searches over a sophisticated structure that is computed dynamically and optimizes several conflicting objectives in a single run. Then a decision-making subsystem is utilized to tune the retrieval preferences of the query user. Their framework yields high query recall rates with minimized CPU time on each mobile device.

5.6 Summary

In this chapter, we develop techniques that generate the kNN graph of an arbitrary crowd of smartphone users that interconnect through a short-range communication technology, such as, Wi-Fi Direct, 3G/LTE direct or Bluetooth v4.0 (BLE). We present two efficient algorithms, namely *Akin+* and *Prox+*, optimized to work on a resource-limited mobile device. These algorithms partition the user space and compute shared candidate sets per partition. *Prox+* uses a custom heap data structure to update the candidate set as new users are inserted, whereas *Akin+* uses a bulk bottom-up construction of a simple heap to compute the candidate set once all users have been inserted. Our experiments verify the theoretical efficiency and shows that *Prox+* and *Akin+* are very well suited for large scale and skewed data scenarios.

Chapter 6

Distributed In-Memory Processing of All k Nearest Neighbor Queries

A wide spectrum of Internet-scale mobile applications, ranging from social networking, gaming and entertainment to emergency response and crisis management, all require efficient and scalable All k Nearest Neighbors (AkNN) computations over millions of moving objects every few seconds to be operational. Most traditional techniques for computing AkNN queries are centralized, lacking both scalability and efficiency. Only recently, distributed techniques for shared-nothing cloud infrastructures (big data architectures) have been proposed to achieve scalability for large datasets. These batch-oriented algorithms are sub-optimal due to inefficient data space partitioning and data replication among processing units. In this chapter we present *Spitfire*, a distributed algorithm that provides a scalable and high-performance AkNN processing framework for both big data architectures but also more traditional High-Performance computing architectures. Our proposed algorithm deploys a fast *load-balanced* partitioning scheme along with an *efficient replication-set* selection algorithm, to provide fast main-memory computations of the exact AkNN results in a batch-oriented manner. We evaluate, both analytically and experimentally, how the pruning efficiency of the *Spitfire* algorithm plays a pivotal role in reducing communication and response time up to an order of magnitude, compared to three other state-of-the-art distributed AkNN algorithms executed in distributed main-memory.

6.1 Introduction

In the age of smart urban and mobile environments, the mobile crowd generates and consumes massive amounts of heterogeneous data [24]. Such streaming data may offer

a wide spectrum of enhanced science and services, ranging from mobile gaming and entertainment, social networking, to emergency and crisis management services [1]. However, such data present new challenges in cloud-based query processing.

One useful query for the aforementioned services is the *All kNN (AkNN)* query: *finding the k nearest neighbors for all moving objects.* Formally, the kNN of an object o from some dataset O, denoted as $kNN(o, O)$, are the k objects that have the most similar attributes to o [25]. Specifically, given objects $o_a \neq o_b \neq o_c$, $\forall o_b \in kNN(o_a, O)$ and $\forall o_c \in O - kNN(o_a, O)$ it always holds that $dist(o_a, o_b) \leq dist(o_a, o_c)$[1]. An *All kNN (AkNN)* query generates a kNN graph. It computes the $kNN(o, O)$ result for every $o \in O$ and has a quadratic worst-case bound. An AkNN query can alternatively be viewed as a *kNN Self-Join*: *Given a dataset O and an integer k, the kNN Self-Join of O combines each object $o_a \in O$ with its k nearest neighbors from O, i.e.,* $O \bowtie_{kNN} O = \{(o_a, o_b) | o_a, o_b \in O \text{ and } o_b \in kNN(o_a, O)\}$.

A real-world application based on such a query is *Rayzit* [1][2], our award-winning crowd messaging architecture, that connects users instantly to their *k Nearest Neighbors (kNN)* as they move in space (Figure 6.1, left). Similar to other social network applications (e.g., Twitter, Facebook), scalability is key in making *Rayzit* functional and operational. Therefore we are challenged with the necessity to perform a fast computation of an AkNN query every few seconds in a scalable architecture. The wide availability of off-the-shelf, shared-nothing, cloud infrastructures brings a natural framework to cope with scalability, fault-tolerance and performance issues faced in processing AkNN queries. Only recently researchers have proposed algorithms for optimizing AkNN queries in such infrastructures.

Specifically, the state-of-the-art solution [26] consists of three phases, namely *partitioning* the geographic area into sub-areas, computing the kNN *candidates* for each sub-area that need to be *replicated* among servers in order to guarantee correctness and finally, computing locally the global AkNN for the objects within each sub-area taking the *candidates* into consideration. The given algorithm has been designed with an offline (i.e., analytic-oriented) AkNN processing scenario in mind, as opposed to an online (i.e., operational-oriented) AkNN processing scenario we aim for in this work. The *performance* of [26] can be greatly improved, by introducing an optimized partiti-

[1]In our discussion, *dist* can be any L_p-norm distance metric, such as Manhattan (L_1), Euclidean (L_2) or Chebyshev (L_∞).

[2]Rayzit: `https://rayzit.cs.ucy.ac.cy/`

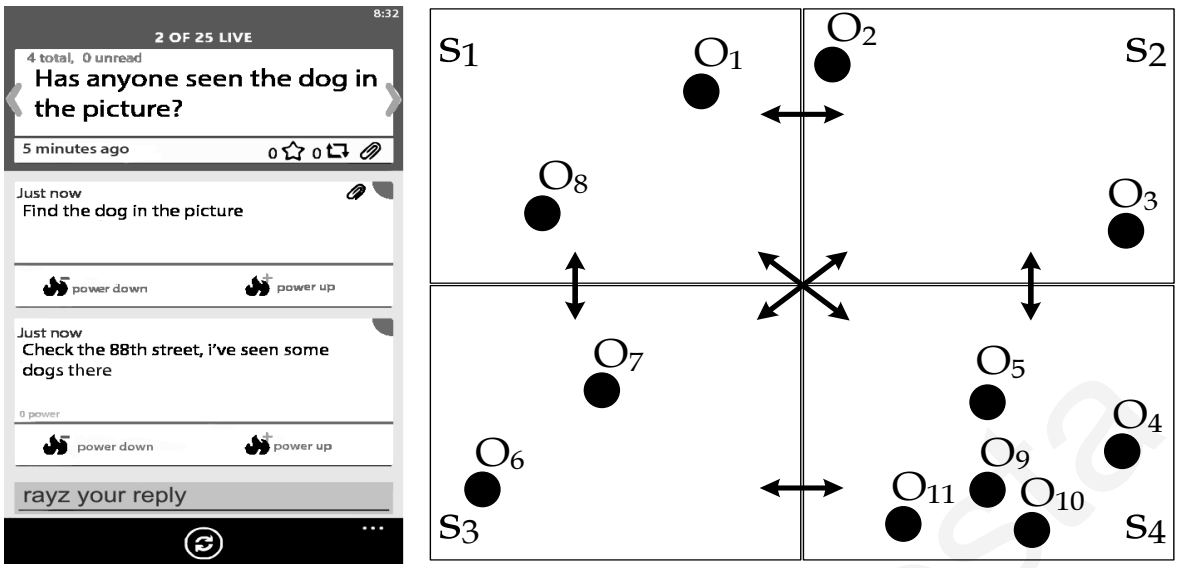

Figure 6.1: (Left) Our Rayzit crowd messenger enabling users to interact with their k geographic Nearest Neighbors. (Right) Distributed main-memory AkNN computation in Rayzit is enabled through the Spitfire algorithm.

oning and replication strategy. These improvements, theoretically and experimentally shown to be superior, are critical in dramatically reducing the AkNN query processing cost yielding results within in a few seconds, as opposed to minutes, for million-scale object scenarios.

Solving the AkNN problem efficiently in a distributed fashion requires the object set O be partitioned into disjoint subsets O_i corresponding to m servers (i.e., $O = \bigcup_{1 \leq i \leq m} O_i$). To facilitate local computations on each server and ensure correctness of the global AkNN result, servers need to compute distances across borders for the objects that lie on opposite sides of the border and are close enough to each other. Consider the example illustrated at the right side of Figure 6.1, where 11 objects are partitioned over 4 spatial quadrants, each being processed by one of four servers $\{s_1 \ldots s_4\}$. Now assume that we are interested in deriving the 2NN for each object $\{o_1 \ldots o_{11}\}$. By visually examining the example, we can identify that the $2NN(o_1, O)$ are $\{o_2, o_8\}$. Although o_8 indeed resides along with o_1 on s_1, the same does not apply to o_2, which resides on s_2. Consequently, in order to calculate $dist(o_1, o_2)$, we will first need to transfer o_2 from s_2 to s_1. The same problem also applies to other objects (e.g., $2NN(o_8, O) = \{o_7, o_1\}$ and $2NN(o_6, O) = \{o_7, o_8\}$).

In any performance-driven distributed algorithm, the efficiency is determined predominantly by the network messaging cost (i.e., network I/O). Therefore, in this work we address the *problem of minimizing the number of objects transferred (replicated)*

between servers during the computation of the AkNN query.

Another factor in a distributed system is balancing the workload assigned to each computing node s_i, such that each s_i will require approximately the same time to compute the distances among objects. By examining Figure 6.1 (right), we can see that s_4 would require to compute 15 distances among 6 objects (i.e., local objects $\{o_4, o_5, o_9, o_{10}, o_{11}\}$ and transferred object $\{o_3\}$), while s_3 would need to compute only 3 distances among 3 objects (i.e., local objects $\{o_6, o_7\}$ and transferred object $\{o_8\}$). This asymmetry, means that s_3 will complete 5 times faster than s_4. In fact, s_4 lies on the critical path of the computation as it has the highest load among all servers.

Consequently, in this work we also address the *problem of quickly deriving a fair partitioning of objects between s_i that would yield a load-balanced execution and thus minimize synchronization time.*

In this chapter we present *Spitfire*, a scalable and high-performance distributed algorithm that solves the AkNN problem in a *fast batch* mode using a shared-nothing cloud infrastructure of m servers. To address the aforementioned load balancing and communication issues, *Spitfire* starts out by partitioning O into disjoint sub-areas of approximately equal population using a fast equi-depth partitioning scheme. It then uses a threshold-based pruning algorithm to determine minimal *replication sets* to be exchanged between servers. Particularly, each server s_i receives from its neighboring servers a set of replicated objects potentially of interest, coined *External Candidates* (EC_i). EC_i supplements server s_i with all the needed external objects to compute the *correct* kNN for every $o \in O_i$, i.e., $kNN(o, EC_i \cup O_i) = kNN(o, O)$.

Particularly, *Spitfire* completes in three discrete phases. First, we devise a simple but fast centralized hash-based adaptation of equi-depth histograms [150] to *partition* the input O into disjoint subsets achieving good load balancing in $O(n+\sqrt{nm})$ time. To do this we first hash the objects based on their locations into a number of sorted equi-width buckets on each axis and then partition each axis sequentially by grouping these buckets in an equi-depth fashion. Subsequently, each s_i computes a subset of O_i, coined *External Candidates* EC_{ji}, which is possibly needed by its neighboring s_j for carrying out a local AkNN computation in the next phase. The given set EC_{ji} is *replicated* from s_i to s_j. Finally, each s_i performs a local $O_i \bowtie_{kNN} (O_i \cup EC_i)$ computation, which is optimized by using a heap structure along with internal geographic grouping and bulk processing.

Spitfire completes in only one communication round, as opposed to two communi-

cation rounds needed by the state-of-the-art [26], and its precise replication scheme has better pruning power, thus minimizing the communication cost/time as it is shown both analytically and experimentally in this work. The CPU time of *Spitfire* is $O(f_{Spitfire}\frac{n^2}{m^2})$ and its communication cost $O(f_{Spitfire}n)$, as this will be shown in Sections 6.2. We show that factor $f_{Spitfire}$ is always smaller than the factor achieved by the state-of-the-art [26]. Finally, *Spitfire* is implemented using the Message Passing Interface (MPI) framework [43]. This makes it particularly useful to large-scale main-memory data processing platforms (e.g., Apache Spark [29]), which have no dedicated AkNN operators.

In our previous work [122], we have presented a centralized algorithm named Proximity, which deals with AkNN queries in continuous query processing scenarios. In this chapter, we completely refocus the problem formulation to tackle the distributed in-memory AkNN query processing problem and propose the *Spitfire* algorithm. Our new contributions are summarized as follows:

- We devise *Spitfire*, a distributed algorithm that solves the AkNN problem in a *fast batch* mode, offering both *scalability* and *efficiency*. It encapsulates a number of innovative internal components, such as: (i) a novel linear-time partitioning algorithm that achieves sufficient load-balancing independent of data skewness, (ii) a new replication algorithm that exploits geometric properties towards minimizing the candidates to be exchanged between servers, and (iii) optimizations added to the local AkNN computation proposed in [122].

- We provide a formal proof of the correctness of our algorithm and a thorough analytical study of its performance.

- We conduct an extensive experimental evaluation that validates our analytical results and shows the superiority of *Spitfire*. Particularly, we use four datasets of various skewness to test real implementations of AkNN algorithms on our 9-node cluster, and report an improvement of at least 50% in the pruning power of replicated objects that have to be communicated among the servers.

The remainder of the chapter is organized as follows. Section 6.5 provides our problem definition, system model and desiderata, as well as an overview of the related work on distributed AkNN query processing. Section 6.2 presents our *Spitfire* algorithm with a particular emphasis on its partitioning and replication strategies, whereas Section 6.3

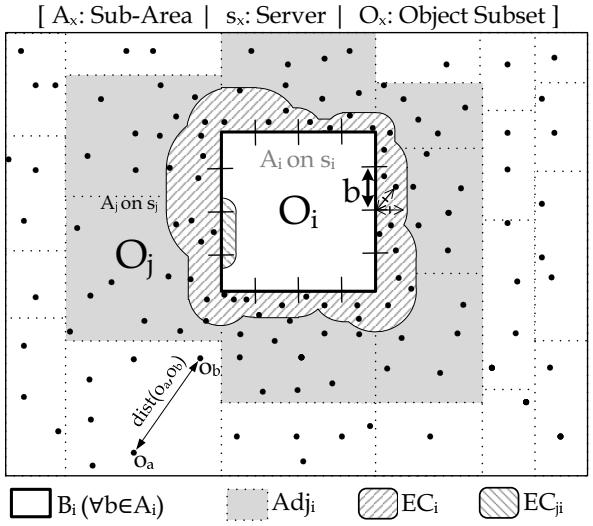

Figure 6.2: Spitfire Overview: (i) Space partitioning to equi-depth quadrants; (ii) Replication between neighboring O_i and O_j using EC_{ji} and EC_{ij}, respectively; and (iii) Local refinement within each $O_i \cup EC_i$.

analyzes its correctness and complexity. Section 6.4 presents an extensive experimental evaluation and Section 6.6 concludes the chapter.

6.2 The Spitfire Algorithm

In this section we propose *Spitfire*, a high-performance distributed main-memory algorithm. We outline its operation and intrinsic characteristics and then detail its three internal steps that capture the core functionality, namely *partition* (Partitioning/Splitting), *computeECB* (Replication) and *localAkNN* (Refinement). The name *Spitfire* is derived by a syllable play using the meaning of these three steps, namely **Split**, Re**fi**ne **re**plicate, which implies good mechanical performance.

6.2.1 Spitfire: Overview and Highlights

As shown in [122], grouping the points geographically and computing common kNN candidates per group, instead of computing the kNN for each point separately, significantly improves performance. Furthermore, partitioning is necessary for distribution, thus such algorithms, which geographically group their points, inherently lend themselves as distributed solutions. For the above reasons, the solution we propose in this work also belongs to the category of "top-down" distributed AkNN solutions.

Overview: In *Spitfire*, the input O is processed in a single communication round,

involving the three discrete steps shown in Algorithm 1 and explained below (please see Figure 6.2 along with the description):

- **Step 1 (Partitioning):** Initially, O is partitioned into (disjoint) sub-areas with an approximately equal number of objects, i.e., $O = \bigcup_{1 \leq i \leq m} O_i$. We use a simple but fast centralized hash-based adaptation of equi-depth histograms [150] to achieve good load balancing in $O(n + \sqrt{nm})$ time. It first hashes the objects based on their locations into a number of sorted equi-width buckets on each axis and then partitions each axis sequentially by grouping these buckets in an equi-depth fashion. Let each O_i belong to area A_i handled by server s_i, B_i be the border segmentations surrounding A_i and Adj_i be the adjacent servers to s_i (handling adjacent areas to A_i).

- **Step 2 (Replication):** Subsequently, each s_i computes a subset of O_i, coined *External Candidates* EC_{ji}, which is possibly needed by its neighboring s_j for carrying out a local AkNN computation in Step 3 (refinement). The given set EC_{ji} is transmitted by s_i to s_j (i.e., left-dashed area within O_i, as depicted in Figure 6.2). Since each s_j applies the above operation as well, we also have the notion of EC_{ij}. The union of EC_{ij} for all neighboring $s_j \in Adj_i$ defines the External Candidates of O_i, i.e., $EC_i = \bigcup_{1 \leq j \leq Adj_i} EC_{ij}$. The cardinality of all EC_i defines the *Spitfire replication factor*, i.e.,

$$f_{Spitfire} = \frac{1}{n} \sum_{i=1}^{m} |EC_i| + 1 \tag{6.1}$$

- **Step 3 (Refinement):** Finally, each s_i performs a local $O_i \bowtie_{kNN} (O_i \cup EC_i)$ computation, which is optimized by using a heap structure along with internal geographic grouping and bulk processing.

The CPU time of *Spitfire* is $T(n) = n + \sqrt{nm} + f_{Spitfire}\frac{n^2}{m^2}$, as this will be shown in Sections 6.2.2 and 6.2.4, respectively. Its communication cost is the total number of objects communicated over the network, i.e., $C(n) = f_{Spitfire} n$.

Highlights: *Spitfire*'s main advantages to prior work follow:

- **Fast Batch Processing:** *Spitfire* is suitable for *online operational* AkNN workloads as opposed to *offline analytic* AkNN workloads. Particularly, it is able to compute the AkNN result-set every few seconds as opposed to minutes required by state-of-the-art AkNN algorithms configured in main-memory.

Algorithm 6 - *Spitfire* Distributed AkNN Algorithm

Input: n Objects O in Area A, m Servers S, Parameter k

Output: AkNN of O

▷ **Step 1: Partitioning (centrally)**

1: $Areas = partition(A, m)$ ▷ **(Algo. 7, Sec. 6.2.2)**
2: **for all** $A_i \in Areas$ **do**
3: *determine* O_i, Adj_i and B_i
4: *transmit* O_i to server $s_i \in S$
5: **end for**

▷ **Step 2: Replication (parallel on each s_i)**

6: **for all** $b \in B_i$ **do** ▷ Find candidates needed by each A_j
7: EC_b =*computeECB(b,*O_i*)* ▷ **(Algo. 8, Sec. 6.2.3)**
8: EC_{ji}=$EC_{ji} \cup EC_b$ ▷ Append to EC_{ji} results.
9: **end for**
10: **for all** $A_j \in Adj_i$ **do** ▷ Exchange External Candidates
11: *Asynchronous send* EC_{ji} to adjacent server s_j
12: *Asynchronous receive* EC_{ij} from adjacent server s_j
13: $EC_i = EC_i \cup EC_{ij}$ ▷ Append to EC_i results.
14: **end for**

▷ **Step 3: Refinement (parallel on each s_i)**

15: $localAkNN(O_i, EC_i)$ ▷ **(Algo. 9, Sec. 6.2.4)**

- **Effective Pruning:** *Spitfire* uses a pruning strategy that achieves replication factor $f_{Spitfire}$, which is shown analytically and experimentally to be always better than that achieved by state-of-the-art AkNN algorithms.
- **Single Round:** *Spitfire* is a single round algorithm as opposed to state-of-the-art AkNN algorithms that require multiple rounds.

6.2.2 Step 1: Partitioning

To fully utilize the processing power of the available servers in the cluster, it is desirable to allocate an evenly balanced workload. This functionality has to be carried out in a fast batch-oriented manner, in order to accommodate the real-time nature of crowd messaging services such as those offered by Rayzit. Particularly, our execution has to be carried out every few seconds. As the partitioning is to be used by each AkNN query, the result will not be useful after a few seconds (i.e., when the next AkNN

Algorithm 7 - *partition*(A, m) Algorithm

Input: object space A, number of partitions m, set of objects O, number of buckets per axis p_{axis}

1: $partition_s = \emptyset$, $1 \leq s \leq m$ ▷ initialize final partitions
2: $xpartition_r = \emptyset$, $1 \leq r \leq \lceil \sqrt{m} \rceil$ ▷ initialize x-axis partitions
3: xbuckets = equi-width hash $\forall o \in O$ into p_x buckets
4: **for all** bucket in xbuckets **do**
5: **if** $|xpartition_r| + \frac{1}{2}|bucket| > \frac{n}{\sqrt{m}}$ and $r < \sqrt{m}$ **then**
6: $r = r + 1$
7: **end if**
8: $xpartition_r \leftarrow bucket$
9: **end for**
10: **for all** *part* in *xpartitions* **do**
11: empty all ybuckets
12: ybuckets=equi-width hash $\forall o \in part$ into p_y buckets
13: **for all** *bucket* in *ybuckets* **do**
14: **if** $|partition_s| + \frac{1}{2}|bucket| > \frac{n}{m}$ and $s < \sqrt{m}$ **then**
15: $s = s + 1$
16: **end if**
17: $partition_s \leftarrow bucket$
18: **end for**
19: **end for**

query is executed). We consequently have not opted for traditional space partitioning index structures (e.g., k-d trees, R-trees, etc.), as these require a wasteful $O(nlogn)$ construction time.

Our *partition* function runs on a master node centrally and uses a hash-based adaptation of equi-depth histograms [150] for speed and simplicity. Instead of ordering the objects on each axis and then partitioning each axis sequentially for a time complexity of $O(nlogn)$, our *partition* function first hashes the objects, based on their location, into $p_{axis} < n$ sorted equi-width buckets on each axis, and then partitions each axis by grouping these buckets for $O(n + \sqrt{mn})$ time.

Particularly, our *partition* function (Algorithm 7) splits the x-axis into p_x equi-width buckets and hashes each object o in O in the corresponding x-axis bucket (Line 3). Then it groups all x-axis buckets into $\lceil \sqrt{m} \rceil$ vertical partitions (*xpartition*) so that no group has more than $\frac{n}{\sqrt{m}} + \frac{1}{2}|bucket|$ objects (Line 4-9). The last x-axis partition gets the remaining buckets. Next, it splits the y-axis into p_y equi-width buckets. For each generated vertical partition $xpartition_i$ it hashes object $o \in xpartition_i$ into the

corresponding bucket (Line 12). Then it groups all y-axis buckets into $\lceil \sqrt{m} \rceil$ *partitions* so that no group has more than $\frac{n}{m}+\frac{1}{2}|bucket|$ objects (Line 13-18). The last y-axis partition gets the remaining buckets.

The result is m partitions of approximately equal population, i.e., $\frac{n}{m}+\frac{1}{2}|bucket|$. The more buckets we hash into, i.e., larger values for p_x and p_y, the more "even" the populations will be. The time complexity of the partition function is determined by the number n of objects to hash into each bucket $(p_x + p_y)$ (Lines 3 and 12) and the nested-loop over all $\sqrt{m}$ xpartitions (Lines 10-19). In our setting, $p_x < \sqrt{n}$ and $p_y < \sqrt{n}$ are used in the internal loop (Lines 13-18). Thus, the total time complexity is $O(n + \sqrt{mn}) = O(n)$, since $n > m$.

6.2.3 Step 2: Replication

The theoretical foundation of our replication algorithm is based on the notion of "hiding", analyzed in detail later in Section 6.3.1. Intuitively, given the kNNs of a line segment or corner b and a set of points O_i on one side of b, it is guaranteed that any point belonging to the opposite side of b, other than the given kNNs of b, is not a kNN of O_i.

Each server s_i computes the *External Candidates* EC_{ji} for each of its adjacent servers $s_j \in Adj_i$ (Algorithm 6, Line 6-9). It runs the *computeECB* algorithm for each border segment or corner $b \in B_i$ (Line 7) and combines the results according to the adjacency between b and Adj_i (Line 8).

computeECB (Algorithm 8) scans all the objects in O_i once to find the $kNN(b, O_i)$, i.e., the k objects with the smallest *mindist* to border b (Line 2), where $mindist(o, b) = min_{p \in b}\{dist(o, p)\}$ and p is any point on b. Note, that the partitioning step guarantees that each server will have at least k objects if $m < \frac{n}{k} - |bucket|$.

A pruning threshold θ_b is determined by $kNN(b, O_i)$ and used to prune objects that should not be part of EC_b. Specifically, threshold θ_b is the worst (i.e., largest) $maxdist(o, b)$ of any object $o \in kNN(b, O_i)$ to border b (Line 3), where $maxdist(o, b)$ is defined as $maxdist(o, b) = max_{p \in b}\{dist(o, p)\}$.

$$\theta_b = argmax_{p \in kNN(b,O_i)}\{maxdist(p, b)\} \tag{6.2}$$

Given θ_b, an object $o \in O_i$ is part of EC_b if and only if its *mindist* to b is smaller than θ_b (Line 4-8) (based on Theorem 1, Section 6.3). Formally,

$$EC_b = \{o | o \in O_i \wedge mindist(o, b) < \theta_b\} \tag{6.3}$$

Algorithm 8 - *computeECB(b,O_i)* Algorithm

Input: border segment (or corner) b, object set O_i

1: construct Min Heap H_b from O_i based on *mindist*(o, b)
2: *kNN*(b, O_i) = extract top k objects from H_b
3: $\theta_b \leftarrow max_{p \in kNN(b,O_i)}\{maxdist(p, b)\}$
4: **for all** $o \in O_i$ **do**
5: **if** *mindist*$(o, b) < \theta_b$ **then**
6: $EC_b = EC_b \cup o$
7: **end if**
8: **end for**
9: **return** EC_b

Algorithm 9 - *localAkNN*(O_i, EC_i) Algorithm

Input: External Candidates EC_i and set of objects O_i

1: partition the area A_i into a set of cells C_i
2: **for all** cells $c \in C_i$ **do**
3: construct Min Heap H_c from O_i on *mindist*(o, c)
4: *kNN*(c, O_i) = extract top k objects from H_c
5: $\theta_c \leftarrow max_{p \in kNN(c,O_i)}\{maxdist(p, c)\}$
6: **for all** $o \in O_i$ **do**
7: **if** *mindist*$(o, c) < \theta_c$ **then**
8: $EC_c = EC_c \cup o$
9: **end if**
10: **end for**
11: compute *kNN*$(o, O_c \cup EC_c), \forall o \in O_c$
12: **end for**

As s_i completes the computation of EC_{ji} for an adjacent server $s_j \in Adj_i$, it sends EC_{ji} to s_j and receives $EC_{ij'}$ from some $s'_j \in Adj_i$ that has completed the respective computation in an asynchronous fashion (Algorithm 6, Line 10-14). When all servers complete the replication step, each s_i have received set EC_i=$\bigcup_{s_j \in Adj_i} EC_{ij}$.

In the example of Figure 6.3, server s_1 has O_1= $\{o_1, o_2, o_3, o_4\}$ and wants to run *computeECB* for b=$\overline{be}$. The 2 neighbors *2NN*(b, O_1) of border b are $\{o_1, o_2\}$ and therefore θ_b=*maxdist*(o_1, b) (since *maxdist*(o_1, b)>*maxdist*(o_2, b)). Objects o_3 and o_4 do not qualify as part of EC_b, since *mindist*(o_3, b)>θ_b and *mindist*(o_4, b)>θ_b, thus EC_b=$\{o_1, o_2\}$.

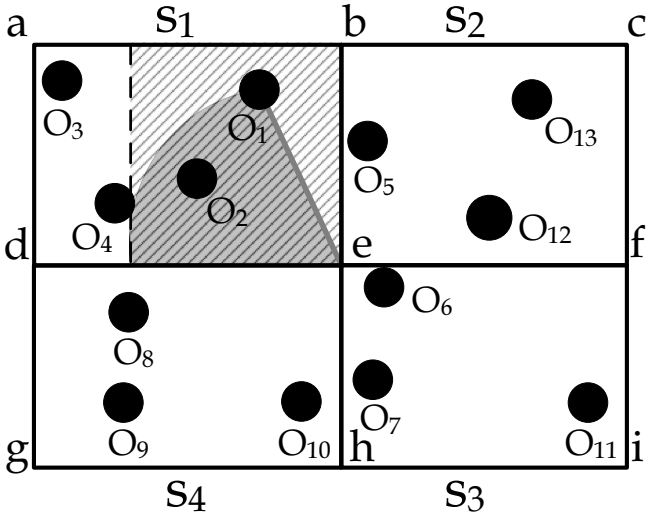

Figure 6.3: Server s_1 sends $\{o_1, o_2\}$ to s_2, $\{o_1, o_2\}$ to s_3, and $\{o_2, o_4\}$ to s_4.

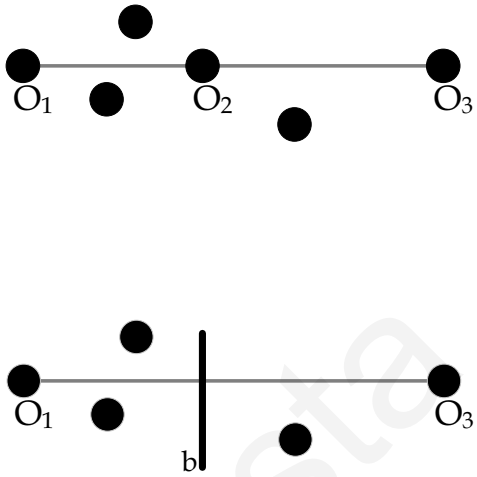

Figure 6.4: (Top) *o_2 hides o_1 from o_3,* (Bottom) *Segment b hides o_1 from o_3.*

6.2.4 Step 3: Refinement

Having received EC_i, each server s_i computes $kNN(o, O_i \cup EC_i), \forall o \in O_i$ (Algorithm 6, Line 15). Any centralized main-memory AkNN algorithm [122, 132] that finds the kNNs from $O_i \cup EC_i$ for each object $o \in O_i$ (a.k.a. kNN-Join between sets O_i and $O_i \cup EC_i$) can be used for this step.

In *Spitfire*, we partition sub-area A_i of server s_i into a grid of *equi-width* cells C_i. Each cell $c \in C_i$ contains a disjoint subset $O_c \subset O_i$ of objects. Next, we compute locally a correct external candidate set EC_c for each cell $c \in C_i$ (similarly to the replication step). Finally, we find the kNNs for each object $o \in O_i$ by computing $kNN(o, O_c \cup EC_c)$.

In Algorithm 9, objects $o \in O_i$ are scanned once to build a k-min heap H_c for each cell $c \in C_i$ based on the minimum distance $mindist(o, c)$ between o and the cell-border (Line 3). The first k objects are then popped from H_c to determine threshold θ_c, based on Equation (6.2) (Lines 4-5). Objects $o \in O_i$ are scanned once again to determine the *External Candidates* EC_c that satisfy the threshold as in Equation (6.3) (Lines 6-10). Finally, s_i computes $kNN(o, O_c \cup EC_c), \forall o \in O_c$ (Line 11).

Given optimal load balancing, the building phase (heap construction and External Candidates) completes in $O(\frac{n}{m})$ time, whereas *finding* the kNN within $O_c \cup EC_c$ completes in $O(f_{Spitfire} \frac{n}{m})$ time, where $\frac{n}{m} = |O_i|$.

6.2.5 Running Example

Given an object set O, assume that a set of servers $\{s_1, s_2, s_3, s_4\}$ have been assigned to sub-areas $\{abde, bcfe, efih, dehg\}$, respectively (see Figure 6.3). In the following, we discuss the processing steps of server s_1. The objects of s_1 are O_1=$\{o_1, o_2, o_3, o_4\}$, its adjacent servers are Adj_i=$\{s_2, s_3, s_4\}$, and its border segments are $B_1 = \{a, \overline{ab}, b, \overline{be}, e, \overline{ed}, d, \overline{da}\}$. For simplicity we have defined the border segments to be a one-to-one mapping to the corresponding adjacent servers. As shown, border segment $\overline{be}$ is adjacent to server s_2, corner e is adjacent to server s_3, and segment $\overline{ed}$ is adjacent to server s_4.

Server s_1 locally computes EC_b for each $b \in B_i$. It does so by scanning all objects $o \in O_1$ and building a heap H_b for each border segment b based on $mindist(o, b)$. The k closest objects to each b are popped from H_b as a result. In our example, $\{o_1, o_2\}$ are the k closest objects to segment $\overline{be}$, $\{o_1, o_2\}$ are the k closest objects to e, and $\{o_2, o_4\}$ are the k closest objects to segment $\overline{ed}$.

For each segment b, its pruning threshold θ_b is determined by the largest *maxdist* of its closest objects computed in the previous step. For instance, for segment $\overline{be}$ this is $\theta_b = maxdist(o_1, \overline{be})$, since $maxdist(o_1, \overline{be}) > maxdist(o_2, \overline{be})$. Given the thresholds θ_b, all objects $o \in O_1$ are scanned again and the condition $\text{mindist}(o, b) < \theta_b$ is checked for each segment b. If this condition holds then object o is part of EC_b. In our example, $EC_{\overline{be}} = \{o_1, o_2\}$, $EC_e = \{o_1, o_2\}$ and $EC_{\overline{ed}} = \{o_2, o_4\}$. Now s_1 sends $EC_{\overline{be}}$ to s_2, EC_e to s_3, and $EC_{\overline{ed}}$ to s_4, based on the adjacency described earlier.

Similarly, the above steps take place in parallel on each server. Therefore, s_1 receives from s_2 the $EC_{\overline{be}}$ of set O_2, from s_3 the $EC_{\overline{e}}$ of set O_3, and from s_4 the $EC_{\overline{ed}}$ of set O_4. Hence, server s_1 will be able to construct its $EC_1 = \bigcup_{b \in B_i} EC_b = \{o_5, o_6, o_7, o_8\}$. The External Candidate computation completes and the local kNN refinement phase initiates computing $kNN(o, O_i \cup EC_i), \forall o \in O_i$ on each server s_i.

6.3 Correctness and Analysis

In this section we first show that our algorithm leads to a correct AkNN result, i.e., $\sum_i^m kNN(o, EC_i \cup O_i) = kNN(o, O)$, based on the External Candidates determined by *computeECB*. Then, we analyze its computational and communication cost.

6.3.1 Correctness of the computeECB function

To prove correctness, we show that it suffices to compute the External Candidates EC_{B_i} to border B_i in order to find the External Candidates EC_i of the whole area A_i, given area A_i, its border B_i, and the necessary objects around B_i. In the following, we first define the notion of *point hiding*.

Definition 6.3.1 (Point Hiding)**.** Given three points o_1, o_2, o_3 on a line, which holds the following relationship $dist(o_1, o_3) = dist(o_1, o_2) + dist(o_2, o_3)$, we say that o_2 *hides* o_1 and o_3 from each other.

In Figure 6.4 *(top)* point o_2 *hides* o_1 and o_3 from each other.

Lemma 1. *Given three points o_1, o_2, o_3 where o_2 hides o_1 from o_3 and the fact that o_1 is not a kNN of o_2, it holds that o_1 is not a kNN of o_3, and vice versa.*

Proof. To prove that o_1 is not a kNN of o_3 it suffices to prove that there are k points closer than o_1 is to o_3. The fact that o_1 is not a kNN of o_2 means that there are k other points, $\{p_1, p_2, ..., p_k\}$, in space that are closer to o_2 than is o_1, $dist(p_i, o_2) \leq dist(o_1, o_2)$. It holds that $dist(p_i, o_3) \leq dist(o_1, o_2) + dist(o_1, o_3) - dist(o_2, o_1)$ based on trigonometry, which gives $dist(p_i, o_3) \leq dist(o_1, o_3)$. Therefore there are k points, namely $\{p_1, p_2, ..., p_k\}$, that are closer to o_3 than is o_1 □

Similarly, we can extend the notion of hiding from a point to a line segment, i.e., border. In Figure 6.4 *(bottom)* segment b *hides* o_1 and o_3 from each other.

Definition 6.3.2 (Segment Hiding)**.** Given two points o_1, o_3, and a segment b, we say that b *hides* o_1 and o_3 from each other, when there is always a point $o \in b$ that *hides* o_1 and o_3 from each other.

Lemma 2. *Given two points o_1 and o_3, a segment b that hides o_1 from o_3, and the fact that o_1 is not a kNN of any point on b, it holds that o_1 is not a kNN of o_3, and vice versa.*

Proof. It suffices if it holds that $dist(k_i, o_3) \leq dist(o_1, o_3)$ for $1 \leq i \leq k$. Given that o_1 is not a kNN of any point $p \in b$, then for each p there are k other points $\{k_1, k_2, ..., k_k\}^p$ in space. It holds that $dist(k_i^{p'}, o_3) \leq dist(o_1, o_3) - dist(o_1, p') + dist(o_1, p')$ based on trigonometry, which gives $dist(k_i^{p'}, o_3) \leq dist(o_1, o_3)$ for $1 \leq i \leq k$. □

Since border B_i of area A_i hides every point that is outside A_i from the points inside A_i, we can easily extend Lemma 1 and Lemma 2 into Lemma 3:

Lemma 3. *Given an area A_i, its objects O_i and its border segments B_i, then any object x outside area A_i, i.e., $x \notin O_i$, that is not a kNN of some point of border B_i is guaranteed not to be a kNN of any object inside A_i.*

6.3.2 Correctness of *Spitfire*

The correctness of computeECB performed on a single server assumes that for a given border b the server has access to all the kNN candidates of b. In a distributed environment this may not be the case as some candidates of b might span over several servers. More specifically, this happens when a server has less than k objects or when there is a θ_b that is greater than the side of the sub-area A_i assigned to the server.

The result of *Spitfire* is always correct since it deals with both cases gracefully. In particular, given a dataset of size n, *Spitfire* does not allow m to be set such that $\frac{n}{m} < k$ and furthermore, at the end of the partitioning step (Algorithm 7) it iterates through the m generated partitions $partition_s$, $1 \leq s \leq m$, to check whether $|partition_s| < k$. If this is the case, *Spitfire* re-instantiates itself using only $m/2$ servers in order to produce partitions with larger population.

To handle the second case, *Spitfire* also computes the side lengths of each partition during the partitioning step (Algorithm 7) and checks on each server during the replication step whether for any $b \in B_i$ it holds that $\theta_b > partition\_side\_length$ in Algorithm 8. If this is the case, *Spitfire* re-instantiates itself using only $m/2$ servers in order to produce partitions with bigger side lengths. The above controls are not shown in the Algorithms for clarity of presentation.

Given that each server s_i receives EC_i computed by function *computeECB* over all adjacent servers $s_j \in Adj_i$ we get:

Theorem 1. *Given an object set O that is geographical partitioned into disjoint subsets $O = \bigcup_{1 \leq i \leq m} O_i$, the bounding border B_i of each O_i, and the segmentation of B_i into segments $b \in B_i$, it holds that $kNN(o, O) = \sum_i^m kNN(o, O_i \cup EC_i)$, $\forall o \in O_i$ if and only if $EC_i = EC_{B_i} = \bigcup_{b \in B_i} EC_b, \forall 1 \leq i \leq m$.*

Proof. Directly from Equation (6.3) and Lemma 3 □

6.3.3 Computational Cost of computeECB

The computational cost is directly affected by the replication factor $f_{Spitfire}$ achieved by *Spitfire*. Assume that the border B_i of area A_i is divided into $|B_i|$ equi-width border segments $b \in B_i$, with width d_b.

Lemma 4. *Given n objects, m servers, $|B|$ number of segments for each area border, and the optimal allocation of objects to the servers $\frac{n}{m}$, the time to compute the candidates EC_i for each sub-area is $O(|B|\cdot(\frac{n}{m}+klog\frac{n}{m}))$.*

Proof. Assuming optimal partitioning and an equal number of segments for each server, it holds that $|O_i| = \frac{n}{m}$ and $|B| = |B_i|$ for each s_i, respectively. In Algorithm 8 *computeECB* is invoked for each border segment $b \in B_i$ in order to compute the candidates EC_i. Determining $kNN(B_i, O_i)$ (Line 1) and θ_i (Line 3) has time complexity $O(\frac{n}{m}+klog\frac{n}{m})$. Scanning set O_i to determine EC_b using θ_b (Lines 4 - 8) has time complexity $O(\frac{n}{m})$. Therefore, each server spends $O(|B|(\frac{n}{m}+klog\frac{n}{m}))$ time to compute the candidates to be transmitted to its neighbors. □

Theorem 2. *Given n objects, m servers, parameter k, the perimeter P_A of area A, the length d_b of each border segments, and the optimal allocation of objects to the servers $\frac{n}{m}$, the time to compute the candidates EC_i for each sub-area is $O(\frac{P_A\sqrt{m}}{d_b}(\frac{n}{m}+klog\frac{n}{m}))$.*

Proof. In Lemma 4 we can replace the number of segments $|B|$ by the total length L over the length of the segments d_b as follows: $|B| = L/d_b$. The total length of all borders based on the *partition* algorithm is $L = \sqrt{m} * A_x + \sqrt{m} * A_y$, as each axis is partitioned $\sqrt{m}$ times. A_{axis} represents the length of area A along the given *axis*. Therefore, $|B|=\frac{P_A\sqrt{m}}{2d_b}$ □

6.3.4 Communication Cost of Replication

The computational cost is directly affected by the replication factor $f_{Spitfire}$, which is the cardinality of the External Candidate set EC_i for each server s_i (see Equation (6.1) in Section 6.2). Each EC_i consists of the k closest objects to its border B_i plus the objects alt_i whose *mindist* is smaller than θ_i, as described Section 6.2.3:

$$|EC_i| = k + |alt_i| \tag{6.4}$$

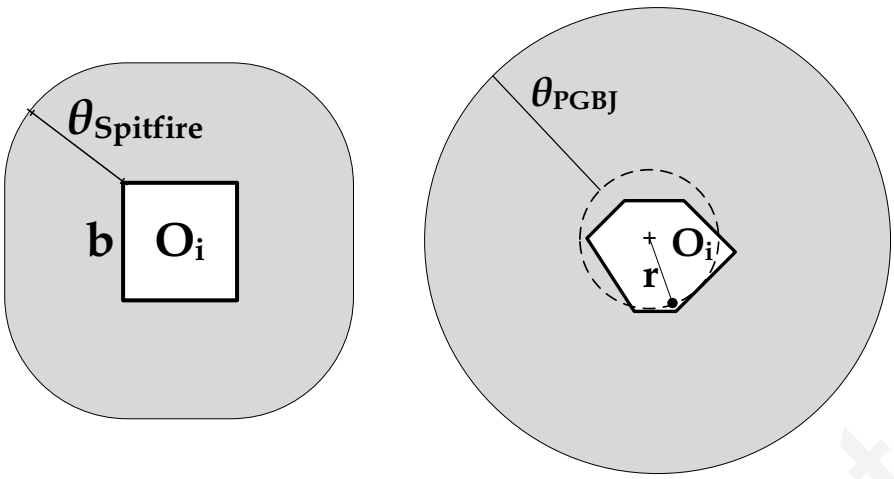

Figure 6.5: Replication factor f in Spitfire (left) and PGBJ (right) shown as shaded areas in both figures.

We can only analyze the replication factor f further if we make an assumption about the distribution of objects. Hereafter, we assume that the distribution is uniform. Further, w.l.o.g. we assume that we use border segments of the same diameter d_b to compose the borders between sub-areas.

Lemma 5. *Given a uniform distribution of n objects over area A, m servers with border segment diameter d_b, and an AkNN query, the alternative external candidate population is $|alt_i| \approx \frac{n}{A} \cdot (d_b + \sqrt{\frac{kA}{n\pi}})^2 - k$.*

Proof. The proof is omitted due to space limitation. □

Theorem 3. *Given a uniform distribution of n objects over area A, m servers with border segment diameter d_b, and an AkNN query, the replication factor is*

$$f_{Spitfire} \approx \frac{m}{A} \cdot (d_b + \sqrt{\frac{kA}{n\pi}})^2 + 1$$

Proof. Follows from Equations (6.1), (6.4) and Lemma 5. □

6.3.5 Optimal border segment size

Given a cluster setup (m), a dataset (A, n), the *CPU* speed of the servers, the *LAN* speed, an AkNN query (k) and the border segment size d_b used in *Spitfire*, we can estimate the total response time as follows:

$$T = \frac{CPU \cdot P_A \cdot n}{d_b \sqrt{m}} + \frac{LAN \cdot n \cdot m}{A} \cdot (d_b + \sqrt{\frac{kA}{n\pi}})^2$$

Given that the only parameter we can fine-tune is the segment length d_b, we find the optimal value for d_b that minimizes the above equation as follows:

$$d_b = argmin_{d_b} T \tag{6.5}$$

6.3.6 Replication Factor: *Spitfire* vs. PGBJ

In this section, we qualitatively explain the difference of the replication factors $f_{Spitfire}$ and f_{PGBJ} achieved by the replication strategies adopted in *Spitfire* and PGBJ, respectively. We use Figure 6.5 to illustrate the discussion.

In *Spitfire*, the cutoff distance for the candidates $\theta_{Spitfire}$ (defining the shaded bound) is determined by the maximum distance of the k closest external objects to the border segment b (let this be of length d). Now assume that all external objects are located directly on the border b. In this case, $\theta_{Spitfire} = d$. On the other extreme, assume that the external objects are exactly d distance from the border b where their worst case maximum distance to a border point would be $\sqrt{2} \cdot d$. In this case, $\theta_{Spitfire} = \sqrt{2} \cdot d$.

In PGBJ, the maximum distance between a pivot (+) and its assigned objects defines the radius r of a circular bound (dashed line), centered around the pivot. θ_{PGBJ} is determined by the maximum distance of the k closest objects to the pivot plus r. Now assume that all objects are located directly on the pivot. In this case, $r = 0$ and $\theta_{\text{PGBJ}} = 0$. On the other extreme, assume that all objects are on the boundary of the given Voronoi cell. In this case, $\theta_{\text{PGBJ}} = 2 \cdot r$. When $d{=}r$, $\theta_{Spitfire}$ has a $\sqrt{2}$ advantage over θ_{PGBJ}.

6.4 Experimental Evaluation

To validate our proposed ideas and evaluate *Spitfire*, we conduct a comprehensive set of experiments using a real testbed on which all presented algorithms have been implemented. We show the evaluation results of *Spitfire* in comparison with the state-of-the-art algorithms.

6.4.1 Experimental Testbed

Hardware: Our evaluation is carried out on the DMSL VCenter[3] IaaS datacenter, a private cloud, which encompasses 5 IBM System x3550 M3 and HP Proliant DL 360 G7

[3]DMSL VCenter @ UCY. `https://goo.gl/dZfTE5`

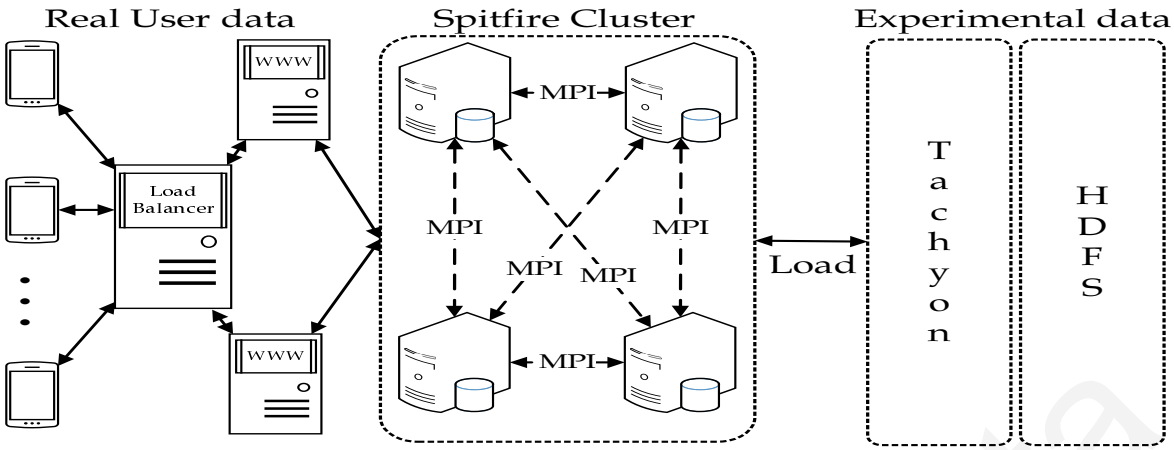

Figure 6.6: Our Rayzit and experimental architecture.

rackables featuring single socket (8 cores) or dual socket (16 cores) Intel(R) Xeon(R) CPU E5620 @ 2.40GHz, respectively. These hosts have collectively 300GB of main memory, 16TB of RAID-5 storage on an IBM 3512 and are interconnected through a Gigabit network. The datacenter is managed through a VMWare vCenter Server 5.1 that connects to the respective VMWare ESXi 5.0.0 hosts.

Computing Nodes: The computing cluster, deployed over our VCenter IaaS, comprises of 9 Ubuntu 12.04 server images (i.e., denoted earlier as s_i), each featuring 8GB of RAM with 2 virtual CPUs (@ 2.40GHz). The images utilize fast local 10K RPM RAID-5 LSILogic SCSI disks, formatted with VMFS 5.54 (1MB block size). Each node features Hadoop v0.20.2 along with memory-centric distributed file system Tachyon v0.5.0 [45]. It also features the Parallel Java Library[4] to accommodate MPI [43] message passing in *Spitfire*.

Rayzit Service [1]: Our service, outlined in Section 6.1, features a HAProxy[5] HTTP load balancer to distribute the load to respective Apache HTTP servers (see Figure 6.6). Each server also features a Couchbase NoSQL document store[6] for storing the messages posted by our users. In Couchbase, data is stored across the servers in JSON format, which is indexed and directly exposed to the Rayzit Web 2.0 API. In the backend, we run the computing node cluster that carries out the AkNN computation as discussed in this work. The results are passed to the servers through main memory (i.e., MemCached) every few seconds.

[4]Parallel Java. `https://goo.gl/uOQsDX`

[5]HAProxy. `http://haproxy.1wt.eu/`

[6]Couchbase. `https://www.couchbase.com/`

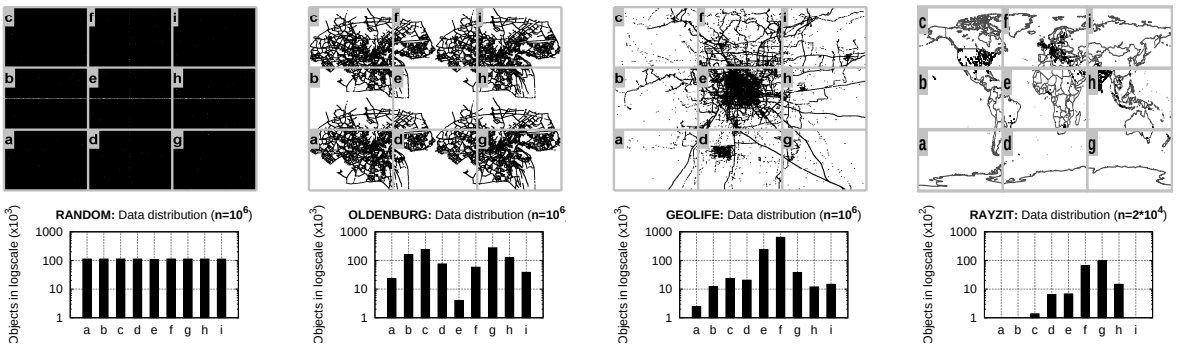

Figure 6.7: Datasets (top row) and population histograms (bottom row) for an indicative 3x3 partitioning.

6.4.2 Datasets

In our experiments we use the following synthetic, realistic and real datasets (depicted in Figure 6.7):

Random (synthetic): This dataset was generated by randomly placing objects in space, in order to generate uniformly distributed datasets of 10K, 100K and 1M users.

Oldenburg (realistic): The initial dataset was generated with the Brinkhoff spatio-temporal generator [128], including 5K vehicle trajectories in a 25km x 25km area of Oldenburg, Germany. The generated spatio-temporal dataset was then decomposed on the temporal dimension, in order to generate realistic spatial datasets of 10K, 100K and 1M users.

Geolife (realistic): The initial dataset was obtained from the Geolife project at Microsoft Research Asia [129], including 1.1K trajectories of users moving in the city of Beijing, China over a life span of two years (2007-2009). Similarly to Oldenburg, the generated spatio-temporal dataset was decomposed on the temporal dimension, in order to generate realistic spatial datasets of 10K, 100K and 1M users.

Rayzit (real): This is a real spatial dataset of 20K coordinates captured by our Rayzit service during February 2014. We intentionally did not scale this dataset up to more users, in order to preserve the real user distribution.

Figure 6.7 (second row) shows the population histograms for the four respective datasets, when split into nine equi-width partitions. The standard deviation among the buckets for a total population of 1M objects is: (i) 2K in Random; (ii) 90K in Oldenburg; and (iii) 200K in Geolife. For Rayzit, which has a population of 20K, the standard deviation is 3.3K.

6.4.3 Evaluated Algorithms

We compare one centralized and four distributed algorithms, which have been confirmed to generate identical correct results to the AkNN query.

Proximity [122]: This centralized algorithm runs on a single server and groups objects using a given space partitioning of cellular towers in a city. It computes the candidates kNNs of each area and scans those for each object within the area. Although this centralized algorithm is not competitive, we use it as a baseline for putting scalability into perspective.

H-BNLJ [151]: This is the two-phase MapReduce algorithm analyzed in Section 6.5.4, which partitions the object set randomly in $\sqrt{m}$ disjoint sets and creates their m possible pairs. Each server performs a kNN-join among each pair. Finally, the local results are gathered and the top-k results are returned as the final k nearest neighbors of each object.

H-BRJ [151]: This is the same algorithm as H-BNLJ, only it exploits an R-tree when performing the kNN-join to reduce the computation time.

PGBJ [26]: This is the two-phase MapReduce algorithm analyzed in Section 6.5.4, which partitions the space based on a set of pivot points generated in a preprocessing step. The candidate set is then computed based on the distance of each point to each pivot. We use the original implementation kindly provided by the authors of PGBJ that comes with the following configurations: (i) the number of pivots used is set to $P = 4000$ (i.e., $\approx \sqrt{n}$, for $n = 1M$ objects).

Spitfire: This is the algorithm proposed in this work. The only configuration parameter we use is the optimal border segment size d_b, which is derived with Equation (6.5), given the provided cluster of m nodes, the preference k, a dataset (A,n), the CPU speed of the servers and the LAN speed.

The traditional Hadoop implementation transfers intermediate results between tasks through a *disk-oriented* Hadoop Distributed File System (HDFS). For fair comparison we port all MapReduce algorithms to UC Berkeley's Tachyon [45] in-memory file system to enable *memory-oriented* data-sharing across MapReduce jobs. As such, the algorithms presented in this section have no Disk I/O operations, i.e., we are thus only concerned with minimizing Network I/Os (NI/Os).

Table 6.1: Values used in our experiments

Section	Dataset	n	k	m
6.4.5	ALL	$[10^4, 10^5, 10^6]$	64	9
6.4.6	Random	10^6	64	9
6.4.7	ALL	10^6 (10^4 Rayzit)	64	9
6.4.8	Random, Rayzit	10^6 (10^4 Rayzit)	$4^i, 1 \leq i \leq 5$	9
6.4.9	Random	10^6	64	$[3, 6, 9]$

6.4.4 Metrics and Configuration Parameters

Response Time: *This represents the actual time required by a distributed AkNN algorithm to compute its result.* We do not include the time required for loading the initial objects to main memory of the m servers or writing the result out. We use this setting to capture the processing scenarios deployed in our real Rayzit system architecture. Times are averaged over five iterations measured in seconds and plotted in log-scale, unless otherwise stated.

Replication Factor (f): *This represents the number of times the n objects are replicated between servers to guarantee correctness of the AkNN computation.* f determines the communication overhead of distributed algorithms, as described in Section 6.5. A good algorithm is expected to have a low replication factor (when f=1 there is no replication of objects).

We also extend our presentation with additional *Network I/O (NI/O)* and Server Load Balancing measurements. Table 6.1 summarizes all parameters used in the experiments.

6.4.5 Varying Number of Users (n)

In this experimental series, we increase the workload of the system by growing the number of online users (n) exponentially and measure the response time and replication factor of the algorithms under evaluation.

Total Computation: In Figure 6.8, we measure the total response time for all algorithms, datasets and workloads. We can clearly see that *Spitfire* outperforms all other algorithms in every case. It is also evident that H-BNLJ and H-BRJ do not scale. H-BRJ achieves the worst time for 10^6 users. Adding up the values shown in Figures

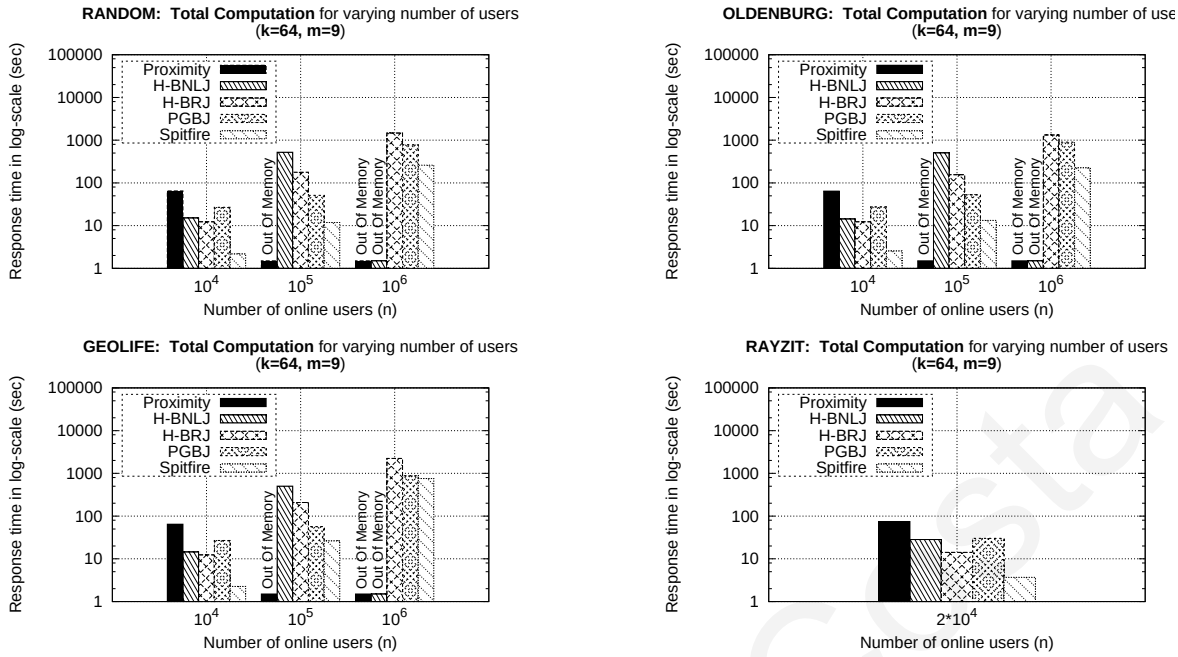

Figure 6.8: AkNN query response time with increasing number of users. We compare the proposed Spitfire algorithm against the three state-of-the-art AkNN algorithms and a centralized algorithm on four datasets.

6.9 and 6.10 and comparing to the total response time in Figure 6.8, it becomes obvious that most of H-BRJ's response time is spent in communication, which is indicated theoretically by its communication complexity of $O(\sqrt{mn})$ shown in Table 6.3. We focus on comparing only *Spitfire* and PGBJ for the rest of our evaluation.

For 10^4 online users, *Spitfire* outperforms all algorithms by at least 85% for all dataset, whereas for 10^5 *Spitfire* outperforms PGBJ, by 75%, 75% and 53% for the Random, Oldenburg and Geolife datasets, respectively.

Spitfire and PGBJ are the only algorithms that scale. For a million online users (n=10^6), *Spitfire* and PGBJ are the fastest algorithms, but *Spitfire* still outperforms PGBJ by 67%, 75%, 14% for the Random, Oldenburg and Geolife datasets, respectively. The small percentage noted for the Geolife dataset is attributed to the fact that this dataset is highly skewed (as observed in Figure 6.7), and that PGBJ achieves better load balancing (as shown later in Section 6.4.7), which in turn leads to a faster refinement step.

Partitioning and Replication: In Figure 6.9 we measure the response time for the partitioning and replication steps in isolation. The theoretical time complexities, as presented in Table 6.3, confirm the outcomes: PGBJ is growing faster with the number of users n, while the other algorithms have only linear growth. These plots also show

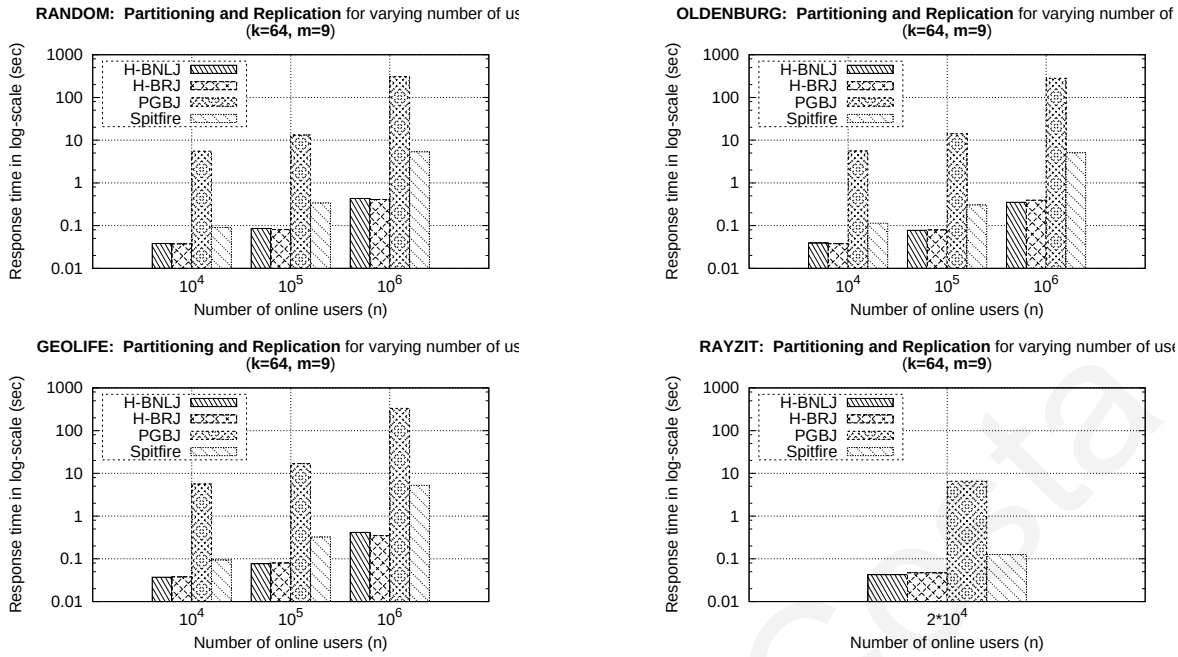

Figure 6.9: Partitioning and Replication step response time with increasing number of users.

that the partitioning step of *Spitfire* features an important advantage: *speed*. *Spitfire* requires only ∼91 milliseconds, as opposed to PGBJ ∼263 milliseconds. In *Spitfire* we have opted for a much faster partitioning algorithm, even if that results in a slightly longer refinement process. Finally, it is also evident that the response time of these steps is independent of the dataset skewness.

Refinement: Figure 6.10 shows that the response time for the refinement step in PGBJ is independent of the dataset skewness, as opposed to *Spitfire*. Specifically, PGBJ achieves a response time of approximately 200 seconds for 10^6 users using any dataset. For the same amount of users *Spitfire* achieves a response time of 90, 100, or 800 seconds depending on the skewness of the dataset. The partitioning step in PGBJ is more sophisticated and produces a more even distribution. This means greater computational cost (Figure 6.9) but reduced response times for refinement (Figure 6.10) due to better load balancing. On the other hand, *Spitfire* strikes a better balance in these two steps, i.e., the much faster partitioning step makes up for the slower refinement step to achieve a much better overall performance.

Replication Factor: In Figure 6.11 we measure the replication factor for the distributed algorithms. It is noteworthy that the replication factor $f_{Spitfire}$ of *Spitfire* is always close to the optimal value 1. *Spitfire* only selects a very small candidate set around the border of each server (Algorithm 8 in Section 6.2.3). As analyzed in Section 6.3.6, in the worst case scenario $f_{Spitfire}$ is only $\sqrt{2}$ times smaller than f_{PGBJ}, but we see that

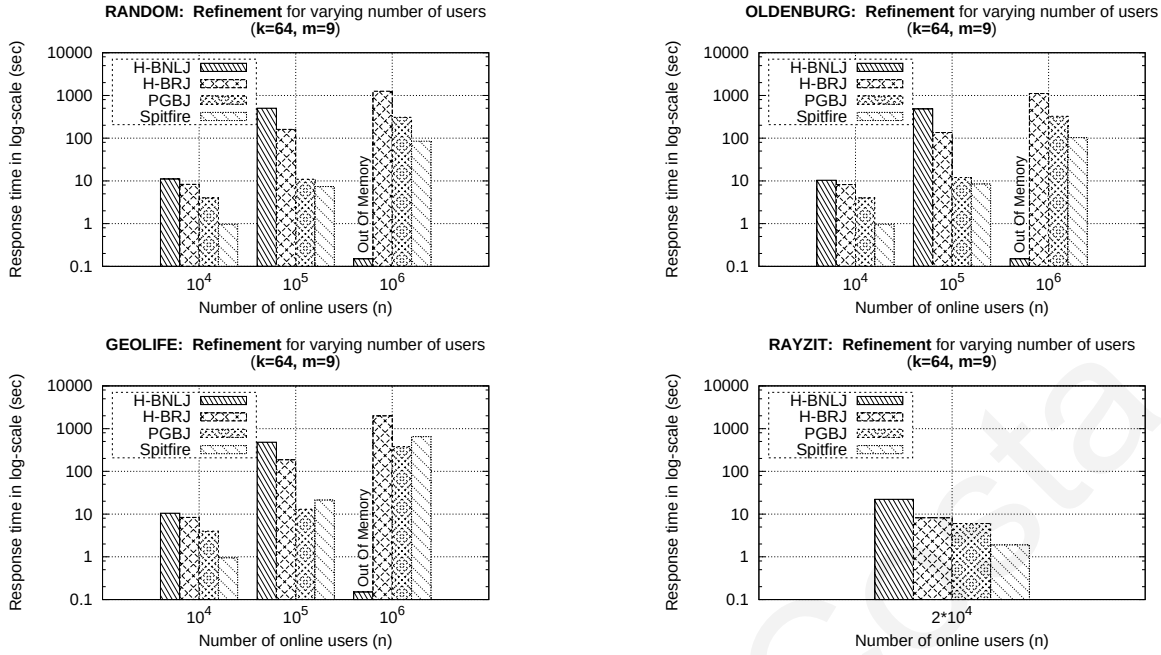

Figure 6.10: Refinement step response time with increasing number of users and for each available dataset.

for real datasets $f_{Spitfire}$ is at least half of f_{PGBJ}. Finally, $f_{\text{H-BNLJ}} = f_{\text{H-BRJ}} = 2\sqrt{m} = 6$ independently of n, as described in Section 6.5.4.

This experimental series demonstrates the algorithmic advantage that Spitfire offers, free from any effect that the implementation framework might add.

6.4.6 Network I/O Performance

We examine the underlying *Network I/O (NI/O)* activity taking place in PGBJ and *Spitfire* in order to better explain the results of Section 6.4.5. For brevity, we only present the Random dataset with n=10^6 online users, using m=9 servers and searching for k=64 NN. The other datasets produce similar results. We measured the Network I/O cost using *nmon*[7].

Figure 6.12 shows that Spitfire features almost no NI/O in its partitioning step, while the respective step for PGBJ is quite intensive and lengthy. In fact, the total network traffic for PGBJ is 215 MB while for *Spitfire* it is only 84 MB. The above observations are compatible with our analysis, where we showed that $f_{Spitfire}$ has a $\sqrt{2}$ advantage over f_{PGBJ} in the worst case. Here the advantage of Spitfire over PGBJ is even greater than $\sqrt{2}$ (i.e., 2.5x).

[7]nmon for Linux. `http://nmon.sourceforge.net/`

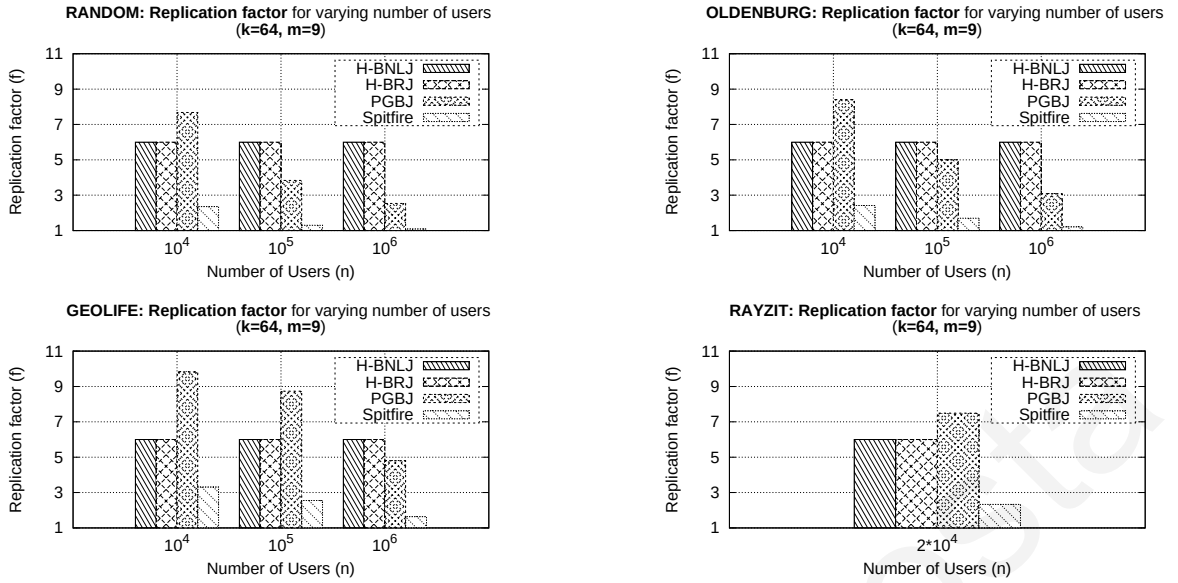

Figure 6.11: Replication factor f with increasing number of users. The optimal value for f is 1, signifying no replication.

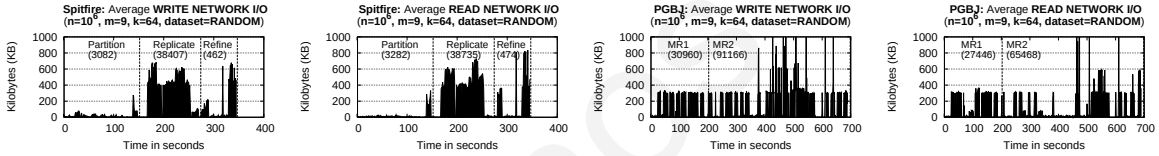

Figure 6.12: Low level Network I/O (NI/O) measurements for Spitfire and PGBJ. Spitfire consumes 2.5x less NI/O.

6.4.7 Partitioning and Load Balancing

In Section 6.4.5, we observe that for certain skewed datasets the competitive advantage of *Spitfire* over PGBJ is relatively small (e.g., in Geolife it is 14%). In this experimental series, we analyze in further depth the performance of the load balancing subroutines deployed in both PGBJ and *Spitfire*, respectively. Going back to our analysis in Section 6.5.4, we recall that PGBJ achieves a close to optimal partitioning using the $\sqrt{n}$ pivots, but at a higher computational cost. Here we experimentally validate these analytical findings.

Figure 6.13 shows that the partitioning technique used by PGBJ achieves almost full load-balancing (i.e., $\pm$ 270 for 10^6 objects), while *Spitfire* achieves a less balanced workload among servers (i.e., $\pm$ 20,315 for 10^6 objects). Clearly, such a workload distribution will force certain servers to perform more distance calculations and will require higher synchronization time. Note, that the load balancing achieved by H-BNLJ and H-BRJ is optimal (standard deviation of object load on servers $\approx$ 0 not

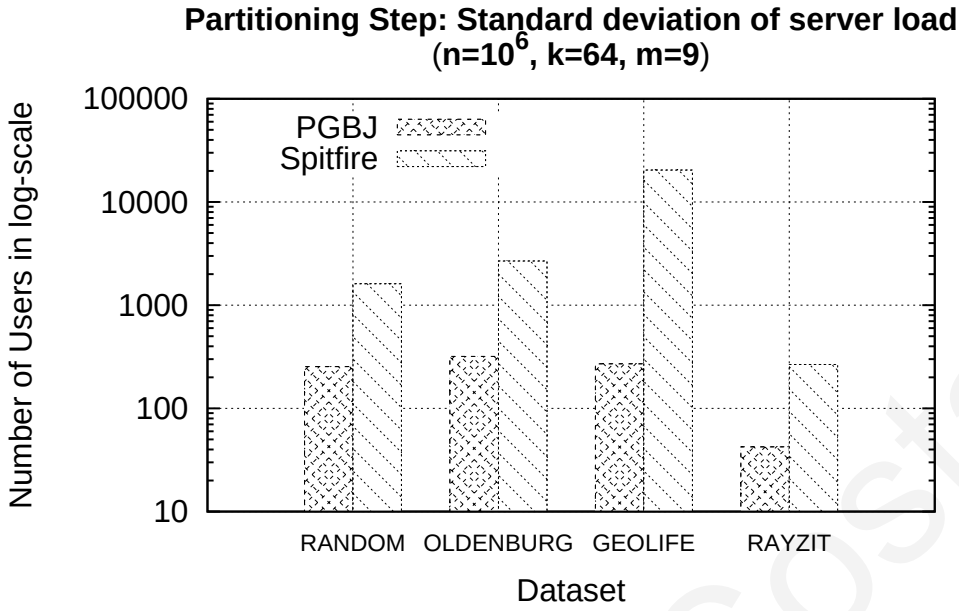

Figure 6.13: Partitioning step: load balancing achieved (less is better). H-BNLJ and H-BRJ achieve optimal load balancing (standard deviation among server load ≈ 0).

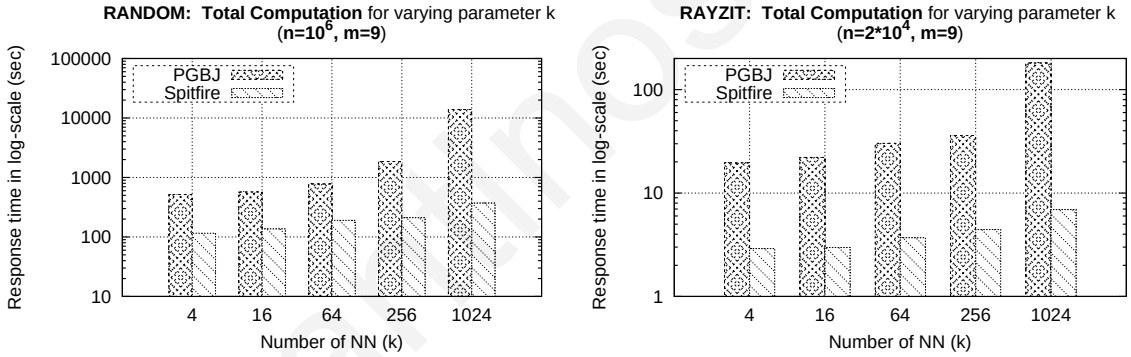

Figure 6.14: The effect of k on response time.

depicted in the figure), because they do not perform spatial partitioning but rather arbitrarily split the original object set into equally sized subsets.

6.4.8 Varying Number of Neighbors (k)

In this experiment, we exponentially increase the query parameter k by a factor of 4 and study its effect on the response time and the replication factor f of both *Spitfire* and PGBJ. We use the Random dataset of $n = 10^6$ online users and the $2{*}10^4$ Rayzit dataset. It is expected that an increasing k increases the workload for the distributed AkNN solutions, as the number of objects exchanged among servers is increased.

In Figure 6.14, we observe that *Spitfire* scales linearly with the increase in k for both datasets. This confirms our analytical result in Section 6.3, which shows *Spitfire*'s

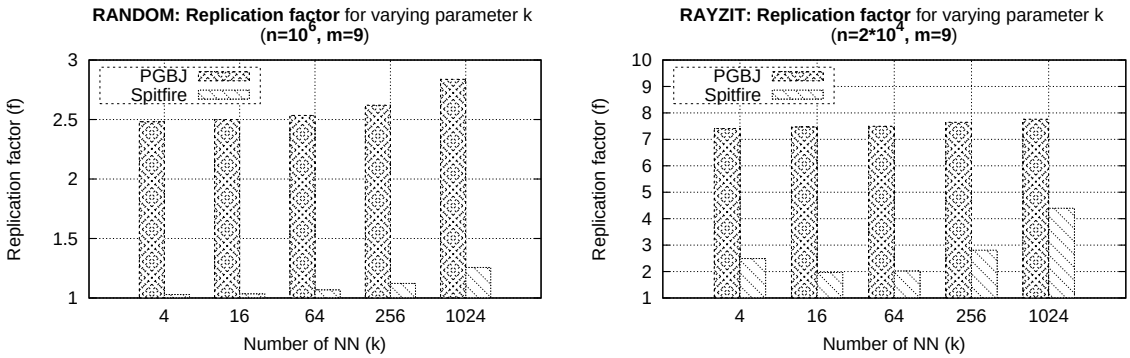

Figure 6.15: The effect of k on the replication factor f.

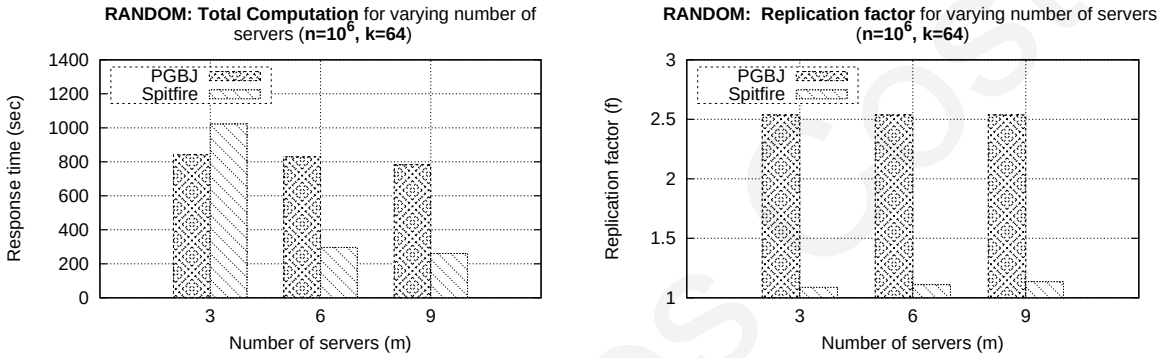

Figure 6.16: The effect of m on response time and the replication factor f.

computational time and replication factor to be sub-linearly proportional to k. *Spitfire* is almost two orders of magnitude faster than PGBJ for $k = 1024$.

Figure 6.15 shows that the replication factor f of *Spitfire*, not only scales well with an increasing k, but also has a very low absolute value. Particularly, $f_{Spitfire}$ is less than 1.07 for $k \leq 64$, and it barely reaches 1.25 for $k = 1024$, showing more than a 95% improvement over f_{PGBJ}. This is one of the main reasons for the better response times exhibited by *Spitfire* in the previous experiments. Therefore, *Spitfire* outperforms PGBJ in scalability when the workload is increased by searching for more nearest neighbors.

6.4.9 Varying Number of Servers (m)

In this experiment we evaluate the effect that the number of servers (m) has on the response time and the replication factor of the distributed algorithms under evaluation.

In Figure 6.16 (left), we observe that with more servers *Spitfire* becomes faster than PGBJ, indicating that *Spitfire* utilizes the computational resources better than PGBJ.

Figure 6.16 (right) shows that the replication factor of *Spitfire* grows slightly faster

than that of PGBJ. This experiment confirms Theorem 3 in Section 6.3, where $f_{Spitfire}$ is shown to increase as the number m of servers increases. Nevertheless, the absolute difference of the replication factor between *Spitfire* and PGBJ remains significantly large, making *Spitfire* the better choice. Comparing the two plots in Figure 6.16 it becomes evident that the replication factor f increases slower than the performance gain with respect to the number of servers, a characteristic that proves the scalability of *Spitfire*.

6.5 Background and Related Work

This section formalizes the problem, describes the general principles needed for efficiency, and overviews existing research on distributed algorithms for computing AkNN queries. Such solutions can be categorized as "bottom-up" or "top-down" approaches. We shall express the AkNN query as a kNN Self-Join introduced earlier. Our main notation is summarized in Table 6.2.

6.5.1 Goal and Design Principles

In this section we outline the desiderata and design principles for efficient distributed AkNN computation.

Research Goal. Given a set of objects O in a bounding area A and a cloud computing infrastructure S, compute the AkNN result of O using S, maximizing *performance*, *scalability* and *load balancing*.

Performance: In a distributed system the main bottleneck for the response time is the communication cost, which is affected by the size of the input dataset for each server. Synchronization, handshake, and/or header data are considered negligible in such environments [39]. Therefore, the lower bound of the communication cost is achieved when the total input of the servers equals to the size of the initial data set O. However, additional communication cost is incurred when some objects need to be transmitted (replicated) to more than one server. Thus, the input is augmented with a number of replicated objects, which is denoted as *replication factor* f.

Scalability: To accommodate the growth of data in volume, an efficient data processing algorithm should exploit the computing power of as many workers as possible. Unfortunately, increasing the number of workers usually comes with an increased

Table 6.2: Summary of Notation

Notation	Description
o, O, n	Object o, set of all o, $n = \|O\|$
s_i, S, m	Server s_i, set of all s_i, $m = \|S\|$
$kNN(o, O)$	k nearest neighbors of o in O
$dist(o_a, o_b)$	L_p-norm distance between o_a and o_b
A, A_i, O_i	Area, Sub-Area i, Objects in sub-area i
b, B_i	A border edge of A_i, set of all $b \in A_i$
Adj_i	Set of all A_j adjacent (sharing b) to A_i
EC_i	External Candidates of A_i

communication cost. A scalable solution would require that the *replication factor f* increases slower than the *performance gain* with respect to the number of servers.

Load Balancing: To fully exploit the computational power of all servers and minimize response time, an efficient algorithm needs to distribute work load equally among servers. In the worst case, a single server may receive the whole load, making the algorithm slower than its centralized counterpart. The work load is determined by the number of objects that are assigned to a server. Therefore, load balancing is achieved when the object set is partitioned equally.

6.5.2 Parallel AkNN Algorithms

There is a significant amount of previous work in the field of computational geometry, where parallel AkNN algorithms for special multi-processor or coarse-gained multi-computer systems are proposed. The algorithm proposed in [133] uses a quad-tree and the well-separated pair decomposition to answer an AkNN query in $O(\log n)$ using $O(n)$ processors on a standard CREW PRAM shared-memory model. Similarly, [152] proposes an algorithm with time complexity $O(n \cdot \log\frac{n}{m} + t(n, m))$, where n is the number of points in the data set, m is the number of processors, and $t(n, m)$ is the time for a global-sort operation. Nevertheless, none of the above algorithms is suitable for a shared-nothing cloud architecture, mainly due to the higher communication cost inherent in the latter architectures.

6.5.3 Distributed AkNN Algorithms: Bottom-Up

The first category of related work on distributed solutions solve the AkNN problem bottom-up by applying existing kNN techniques (e.g., iterative deepening from the query point [137]) to find the kNN for each point separately. The authors in [153] propose a general distributed framework for answering AkNN queries. This framework uses any centralized kNN technique as a black box. It determines how data will be initially distributed and schedules asynchronous communication between servers whenever a kNN search reaches a server border. In [154] the authors build on the same idea, but optimize the initial partitioning of the points onto servers and the number of communication rounds needed between the servers. Nevertheless, it has been shown in [122] that answering a kNN query for each object separately restricts possible optimizations that arise when searching for kNNs for a group of objects that are close.

6.5.4 Distributed AkNN Algorithms: Top-Down

The second category of related work on distributed solutions solve the AkNN problem top-down by first *partitioning* the object set into subsets and then computing kNN *candidates* for each area in a process we call *replication*. These batch-oriented algorithms are directly comparable to our proposed solution, therefore we have summarized their theoretical performance in Table 6.3. All existing algorithms in this category happen to be implemented in the MapReduce framework, therefore we overview basic MapReduce concepts before we describe these algorithms.

Background: *MapReduce [155] (MR)* is a well established programming model for processing large-scale data sets with commodity shared-nothing clusters. Programs written in MapReduce can automatically be parallelized using a reference implementation, such as the open source Hadoop framework[8], while cluster management is taken care of by YARN or Mesos [42]. The Hadoop MapReduce implementation allows programmers to express their query through `map` and `reduce` functions, respectively. For clarity, we refer to the execution of these MapReduce functions as *tasks* and their combination as a *job*. For ease of presentation, we adopt the notation *MR#.map* and *MR#.reduce* to denote the tasks of MapReduce job number #, respectively. Main-memory computations in Hadoop can be enforced using in-memory file systems such as Tachyon [45].

[8] Apache Hadoop. `https://hadoop.apache.org/`

Hadoop Naive kNN Join (*H-NJ* [26]). This algorithm is implemented with 1 MapReduce job. In the map task, O is transferred to all m servers triggering the reduce task that initiates the nested-loop computation $O_i \ltimes_{kNN} O$ (O_i contains n/m objects logically partitioned to the given server). *H-NJ* incurs a heavy $O(\frac{n^2}{m})$ processing cost on each worker during the reduce step, which needs to compute the distances of O_i to O members. It also incurs a heavy $O(mn)$ communication cost, given that each server receives the complete O. The replication factor achieved is $f_{\text{H-NJ}} = m$.

Hadoop Block Nested Loop kNN Join (*H-BNLJ* [151]). This algorithm is implemented with 2 MapReduce jobs, MR1 and MR2, as follows: In MR1.map, O is partitioned into $\sqrt{m}$ disjoint sets, creating m possible pairs of sets in form of (O_i, O_j), where $i, j \leq \sqrt{m}$. Each of the m pairs (O_i, O_j) is sent to one of the m servers. The communication cost for this action is $O(\sqrt{m}n)$, attributed to the replication of m pairs each of size $\frac{n}{\sqrt{m}}$. The objective of the subsequent MR1.reduce task is to allow each of the m servers to derive the "local" kNN results for each of its assigned objects. Particularly, each s_i performs a local block nested loop kNN join $O_i \ltimes_{kNN} O_j$. The results of MR1.reduce have to go through a MR2 job, in order to yield a "global" kNN result per object. Particularly, MR2.map hashes the possible $\sqrt{m}$ kNN results of an object to the same server. Finally, MR2.reduce derives the global kNN for each object using a local top-k filtering. The CPU cost of *H-BNLJ* is $O(\frac{n^2}{m})$, as each server performs a nested loop in MR1.reduce. The replication factor achieved is $f_{\text{H-BNLJ}} = 2\sqrt{m}$.

Hadoop Block R-tree Loop kNN Join (*H-BRJ* [151]). This is similar to *H-BNLJ*, with the difference that an R-tree on the smaller O_i set is built prior to the MR1.reduce task, to alleviate its heavy processing cost shown above. This reduces the join processing cost during MR1.reduce to $O(\frac{n}{\sqrt{m}}\log\frac{n}{\sqrt{m}})$. The communication cost remains $O(\sqrt{m}n)$ and the incurred replication factor is again $f_{\text{H-BRJ}} = 2\sqrt{m}$.

Hadoop Partitioned Grouped Block kNN Join (*PGBJ* [26]): This is the state-of-the-art Hadoop-based AkNN query processing algorithm that is implemented with 2 MapReduce jobs, MR1′ and MR2′, and 1 pre-processing step according to the following logic: in a preprocessing step, a set of approximately $\sqrt{n}$ random pivots in space is generated [26].

During MR1′.map, each object in O is mapped to its closest pivot, thus partitioning O into $\sqrt{n}$ sets (i.e., O=$\bigcup_{1\leq j\leq\sqrt{n}} O_j$). This takes $O(n^{\frac{3}{2}}/m)$ time, since on each server

Table 6.3: Algorithms for Distributed Main-Memory AkNN Queries

[n: objects | m: servers | f: replication factor | $f << m < n$]

Algorithm	**Preproc.**	**Part. & Repl.**	**Refinement**	**Communic.**
H-NJ [26]	-	$O(n)$	$O(\frac{n^2}{m})$	$O(mn)$
H-BNLJ [151]	-	$O(n)$	$O(\frac{n^2}{m})$	$O(\sqrt{m}n)$
H-BRJ [151]	-	$O(n)$	$O(\frac{n}{\sqrt{m}}\log\frac{n}{\sqrt{m}})$	$O(\sqrt{m}n)$
PGBJ [26]	$O(\sqrt{n})$	$O(n^{1.5}/m)$	$O(f_{\text{PGBJ}}\frac{n^2}{m^2})$	$O(f_{\text{PGBJ}}n)$
Spitfire	-	$O(n)$	$O(f_{Spitfire}\frac{n^2}{m^2})$	$O(f_{Spitfire}n)$

the distance of n/m objects is measured against $\sqrt{n}$ pivots. At the same time, the maximum and minimum distances between O_j objects and its corresponding pivot are recorded as lower and upper pruning thresholds for subsequent filtering. In MR2′.map, the given bounds define a set of objects around each partition O_j that must be replicated to O_j (coined F_j). MR1′.reduce in PGBJ is void.

During MR2′.map, the $\sqrt{n}$ subsets defined during MR1′.map are geographically grouped together into m clusters (i.e., $O{=}\bigcup_{1\leq i\leq m} O_i$) using a grouping strategy, which greedily attempts to generate clusters of equal population around some m geometrically dispersed pivots. For each generated cluster O_i, a set F_i is derived based on the union of the respective F_j sets of the cluster defined earlier. Having F_i defined, it allows MR2′.reduce to perform a straightforward $O_i \bowtie_{kNN} (O_i \cup F_i)$ to generate the result set.

The replication factor is $f_{\text{PGBJ}}{=}\frac{1}{n}\sum_{i=1}^{m}|F_i| + 1$. The CPU cost is $O(\sqrt{n})$ for the preprocessing step, $\text{O}(\frac{n}{m}\sqrt{n})$ for MR1′ and $O(f_{\text{PGBJ}}\frac{n^2}{m^2})$ in MR2′. PGBJ only distributes O over m servers and then exchanges $f_{\text{PGBJ}}n$ candidates between servers, therefore its communication cost is $O(f_{\text{PGBJ}}n)$.

6.6 Summary

In this chapter we present *Spitfire*, a scalable and high-performance distributed algorithm that solves the AkNN problem using a shared-nothing cloud infrastructure. Our algorithm offers several advantages over the state-of-the-art algorithms in terms of efficient partitioning, replication and refinement. Theoretical analysis and experimental evaluation show that *Spitfire* outperforms existing algorithms reported in recent literature, achieving scalability both on the number of users and on the number of k nearest neighbors.

Chapter 7

SPATE$^+$ Applications

In this chapter, we present two innovative applications that have been built on top of the *SPATE*$^+$architecture. The applications exploit the component contributions of this thesis.

7.1 The SPATE Application

In this section we present *SPATE* [17], a framework that uses both lossless data *compression* and lossy data *decaying* to ingest large quantities of telco big data in the most compact manner. *Compression* refers to the encoding of data using fewer bits than the original representation and is important as it shifts the resource bottlenecks from storage- and network-I/O to CPU, whose cycles are increasing at a much faster pace. It also enables data exploration tasks to retain full resolution over the most important collected data. *Decaying* on the other hand, as suggested in [74], refers to the progressive loss of detail in information as data ages with time until it has disappeared.

SPATE enables data exploration tasks to retain high-level data exploration capabilities for predefined aggregate queries over extremely long time windows, without consuming enormous amounts of storage. It is shown to offer similar performance to the state-of-the-art for telco-specific tasks [17]. Our objective is to minimize the storage costs associated with telco big data exploration tasks, as storage overheads will inevitably lead to the deletion of valuable data, missing in this way the hope to learn valuable insights at the macroscopic scale.

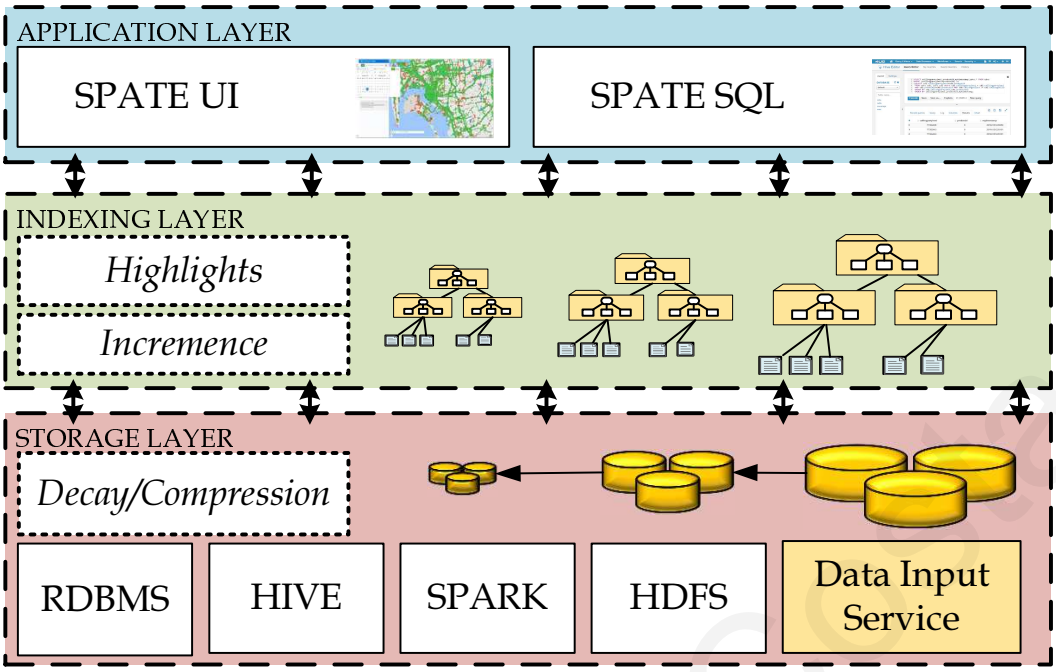

Figure 7.1: SPATE *is an efficient telco big data exploration stack that enables a wide range of smart city applications with a minimal storage cost. It deploys compression, decaying and exploration of the collected data in a unified way.*

7.1.1 Overview of SPATE

We express our solution in three layers (see Figure 7.1), namely Storage Layer, Indexing Layer and Application Layer.

The *Storage layer* passes newly arrived network snapshots through a lossless compression process storing the results on a replicated big data file system for availability and performance. This component is responsible for minimizing the required storage space with minimal overhead on the query response time. The intuition is to use compression techniques that yield high compression ratios but at the same time guarantee small decompression times. We particularly use GZIP compression that offers high compression/decompression speeds, with a high compression ratio and maximum compatibility with I/O stream libraries in the big data ecosystem we use. The storage layer is basically only responsible for the leaf pages of the *SPATE* index described in the next layer.

The Indexing Layer uses a multi-resolution spatio-temporal index, which is incremented on the rightmost path with every new data snapshot that arrives (i.e., every 30 minutes). In addition, the component computes interesting event summaries, called "highlights", from data stored in children nodes and stores them at the parent node.

For each data exploration query, the internal node that covers the temporal window of the query is accessed, and its highlights are used to answer the query. Finally, this layer is also responsible for the gradual decay of the data. It does so by pruning-off parts of the index tree in using the so called data fungus.

The Application Layer implements the querying module and the *data exploration* interfaces, which receive the data exploration queries in visual or declarative mode and use the index to combine the needed highlights and snapshots to answer the query. *SPATE* is equipped with an easy-to-use map-based web interface layer that hides the complexity of the system through a simple and elegant web interface.

7.1.2 SPATE Prototype

We have implemented a prototype of SPATE using a modern SPARK-based processing architecture with HDFS and an RDBMS for catalog management (see Figure 7.1). The *SPATE UI (User Interface)* is implemented in HTML5/CSS3 along with extensive AngularJS. An illustrative network exploration interface is shown in Figure 7.2. We have implemented a query sidebar that allows the user to execute a variety of template queries. The query bar includes snapshot queries and recurring queries (in the form of a time-machine) for drop calls and downflux/upflux, heatmap statistics and settings. Furthermore, quick access buttons are provided so that user is able to choose between the available network modalities (2G, 3G, 4G). The hardware stack of our SPATE installation resides on our laboratory DMSL datacenter and interaction will be achieved over cable or Wi-Fi using a standard laptop, a tablet or smartphones we will bring along at the conference.

7.1.3 Query Exploration Interfaces

We have realized two separate interfaces for the *SPATE* framework: (i) *SPATE-UI*, which is a visual spatio-temporal data exploration interface developed on top of Google Maps; and (ii) *SPATE-SQL*, which is a declarative data exploration interface in Apache Hue (Hadoop User Experience).

The **SPATE-UI** interface allows the user to interactively navigate in space and time (see Figure 7.2). Particularly, the user can set a temporal and spatial predicate and observe the behavior of vital network statistics and how these compare to precomputed network models. For instance, the network coverage is a precomputed heatmap

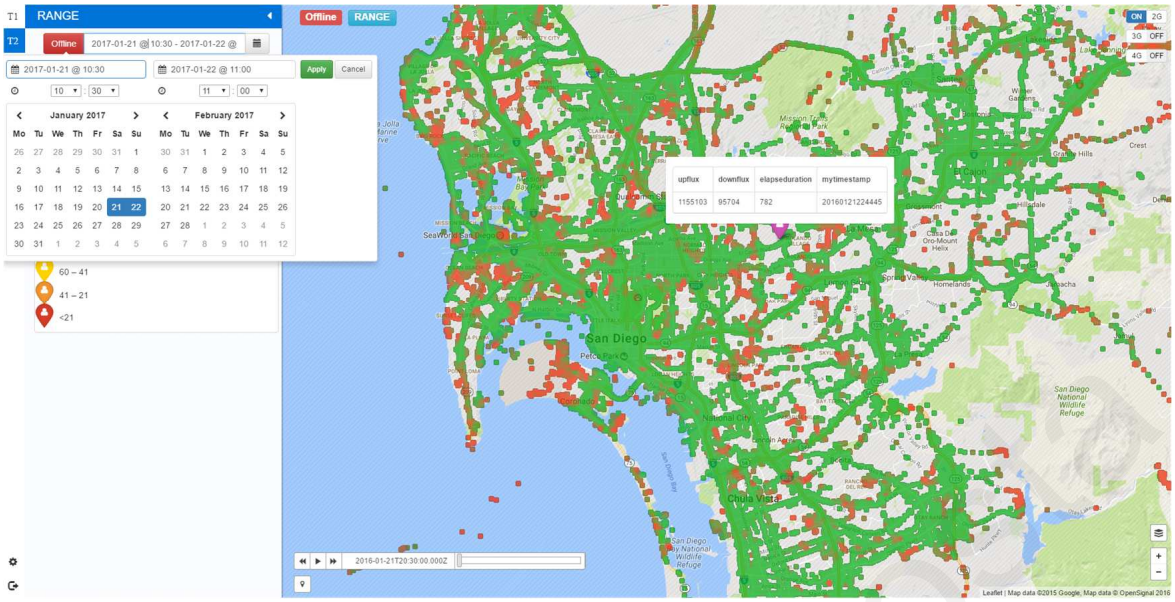

Figure 7.2: SPATE-UI: A spatio-temporal telco data exploration user interface we developed on top of Google Maps, which enables combining network models (e.g., coverage heatmaps) with real network measurements (e.g., CDR, NMS, CELL) encapsulated in the compressed SPATE *structure.*

model that is overlayed and can be visually compared against the real measurements that are loaded from storage to memory through the *SPATE* structure. The above effectively translates to the execution of a variety of spatial range queries on top of the *SPATE* structure. We will show in our experimental section that both range queries as well as other OLTP and OLAP queries retain desirable retrieval properties (response time) at enormous storage savings. The *SPATE-UI* finally also provides a search box that enables a user to narrow the spatial bounding box based on well-known Points-of-Interests (POIs) organized by Google or some other provider (e.g., Openstreetmap). The *SPATE-UI* also contains a query bar that enables the execution of template queries for drop calls and downflux/upflux, heatmap statistics (e.g., showing the RSSi signal intensity around antennas), satellite/terrain map layers (e.g., for Cellular signal propagation faults due to the terrain) and others.

The **SPATE-SQL** interface allows expert users and data scientists to explore the collected data through declarative SQL. The current configuration currently allows all basic SELECT-FROM-WHERE block queries, nested queries, joins, aggregates, etc. directly through the compressed Hadoop Distributed File System (HDFS) representation of the *SPATE* structure, which aims to provide means for ad-hoc query execution with the same storage savings with SPATE-UI.

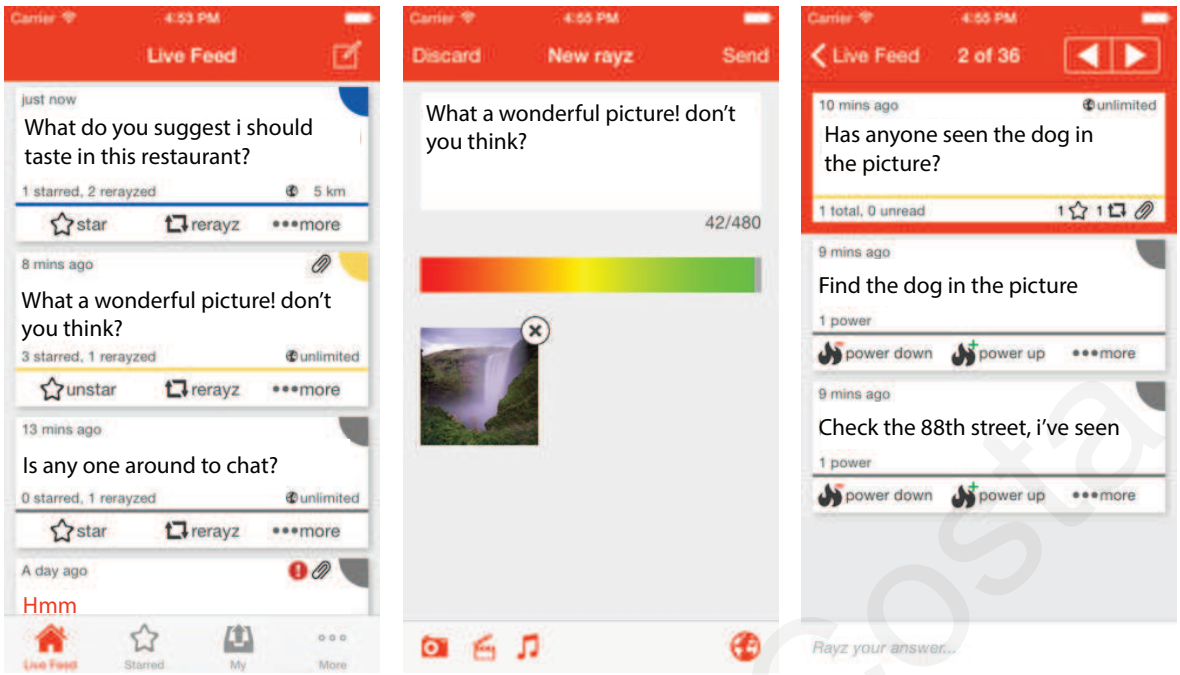

Figure 7.3: **Screenshots of the Rayzit app***: (Left) Live feed of rayz messages; (Center) Sending a new rayz along with the power bar and an attachment (image, audio, video); and (Right) a set of replies to a rayz.*

7.2 The Rayzit Application

The smartphone revolution has introduced a new era of social networks where users communicate over anonymous messaging platforms to exchange opinions, ideas and even carry out commerce. These platforms enable individuals to establish social interactions between strangers based on a common interest or attribute. In this chapter we present *Rayzit*[1], a novel anonymous crowd messaging architecture, which utilizes the location of each user to connect them instantly to their k Nearest Neighbors (kNN) as they move in space. Contrary to the very large body of location-based social networks that suffer from bootstrapping issues, our architecture enables a user to always interact with the geographically closest possible users around. We establish this communication using a fast computation of an All kNN query that generates a dynamic global social graph every few seconds. We present motivating application scenarios and the detailed back-end architecture that allows *Rayzit* to scale. We have collected and analyzed data from the interactions of thousands of active users and confirm our claims.

[1]Rayzit: `https://rayzit.cs.ucy.ac.cy/`

7.2.1 Introduction

During the last years, social networks have transformed the way individuals express and publish their opinions, feelings or thoughts on various topics [9]. A new trend even shows that social networks are starting to become the main commerce platforms, instead of web-apps and online shopping carts[2]. Combining the above with the increased amount of sensors on portable devices [10], gives the opportunity to extend the functionality and the intelligence of social networks. This is already illustrated by real world examples like *EARS*, a real time decision support system for earthquakes based on social networks [11].

The location of a user very often defines their interests and inquiries, therefore, popular location-based social networks with billions of users (i.e., Foursquare.com, YikYakapp.com, goTinder.com) have emerged. Furthermore, anonymous networks have been on the rise, where users do not need to create a profile in order to communicate with others, rather they stay anonymous and have an automatic communication link to a subgroup of users based on some criteria (e.g., location).

In this chapter, we present an anonymous dynamic crowd messaging architecture, coined *Rayzit*, which is based on generating a kNN network among users by connecting each user to their k geographically closest peers at each moment. Traditionally, social networks like Facebook.com, Twitter.com and others are created based on explicit friendship links that yield a static social graph. Such a static graph requires time to become interconnected, changes very slowly and its links require user decision and actions to be created and destroyed (i.e., request or accept friendship, un-friend). In contrast, *Rayzit* uses a kNN graph, which is developed automatically using the current locations of users in the system. This dynamic graph guarantees k links for each user instantly without any user decision or action [119], leading to communication network that is as diverse as one's physical neighborhood.

Our solution utilizes a combination of state-of-the-art computational techniques and crowdsourcing concepts to provide a new compelling social interaction experience through our award-winning app[3], allowing a user to instantly engage in anonymous conversations around them, and using crowdsourcing mechanism to determine the inappropriateness or popularity of posts. Additionally, we have deployed the *Rayzit* app

[2]Chris Messina "Conversational Commerce": `https://goo.gl/xJBolh`

[3]Appcampus: `http://www.appcampus.fi/`

and collected interaction data over a 1 year period from thousands of users, and present our findings through experimental analysis conducted on this data. Such an instant network provides opportunities for advanced location-based services, e.g., location-based marketplace and marketing [156] or disaster relief [157].

7.2.2 Motivating Examples

During a disaster, mobile users can be both victims and rescuers involved in receiving but also providing help from/to their neighboring peers. In such a scenario, constructing a kNN graph can enable communication within the crowd, aid in organizing the crowd, allocating peer-to-peer aid and experts, and distributing tools and medicine optimally. In a real world example, it was the citizen's joint efforts to map the 2012 floods in China that materialized faster and more accurately than the government-sanctioned map[4].

In large ad-hoc events, like cultural festivals (e.g., Woodstock, Old Car enthusiast gatherings, etc.), sporting events, conventions and fairs, demonstrations (e.g., Occupy Wall Street 2011, Egypt Protests 2013, Syria Protests 2014, Rossa Montana Protests Bucharest 2014, Hong Kong Protests 2014, Pakistan 2014, etc.), monitoring the kNN graph centrally and providing communication within the crowd can aid organizing authorities to manage the crowd[5], prevent crowd disasters[6,7], enable new viral marketing strategies, entertainment services[8] and crowd-games[9].

To the best of our knowledge, this is the first anonymous crowd messaging architecture that is utilizing the proximity and the dynamic social environment of a kNN network. With the rise of *conversational commerce*, where commerce is done over messaging services, and crowdsourcing platforms, e.g., TaskRabbit.com, new types of marketplaces will be generated. In this landscape our anonymous crowd messaging platform can offer new possibilities.

The rest of the Section is organized as follows: Subsection 7.2.3 looks at the related work, Subsection 7.2.4 introduces our *Rayzit* architecture thoroughly covering each of its modules. Subsection 7.2.5 presents our prototype application developed for several

[4]The Crowd Maps Beijing Floods: `https://goo.gl/0DHZ4v`

[5]WorkingWithCrowds: `http://www.workingwithcrowds.com/`

[6]Love Parade disaster: `https://goo.gl/2FpIbm`

[7]Hillsborough disaster: `https://goo.gl/xvLRlc`

[8]Opphos: `http://www.sics.se/projects/opphos`

[9]CrowdControlGames: `http://crowdcontrolgames.com/`

platforms (Windows Phone and iOS), Subsection 7.2.6 provides a small evaluation using *Rayzit* data while Subsection 7.2.7 looks into future issues and concludes the section.

7.2.3 Related Work And Background

Here, we highlight background and related work in social networks, Big data infrastructures, crowdsourcing techniques and discuss how a kNN network can be efficiently computed.

Social Networks: Classical social networks, like Facebook.com, Twitter.com, SnapChat.com, Foursquare.com and WeChat.com, create a static social graph based on the friendships between user. Friendship links need to be requested and consecutively accepted/rejected in order to be formed, causing the social graph to be built very slowly.

In contrast, applications like Secret.ly, YikYakapp.com, and Whisper.sh create dynamic social graphs based on the user's location [12]. YikYak connects you with the users within a 10 mile range and preserves anonymity. Secret and Whisper have no distance limit, but they store information about the user's profile (i.e., user id) decreasing user's anonymity. WeChat also has a feature that allows you to greet other WeChat users that are around you, which compromises the anonymity of somebody aiming to stay anonymous.

Rayzit is the only application that creates a dynamic social graph based on kNN and a customizable distance cutoff setting, guaranteeing at the same time that you will always be connected no matter how far your neighbors are. In addition, *Rayzit* preserves the anonymity and privacy of the user, since it does not require an account and discards any information regarding the status of the social kNN graph.

In order to comprehend the difference between the aforementioned social networks we present a small taxonomy in Table 7.1. We choose all the important characteristics that apply to current social networks. In addition, we include the kNN feature that only applies to *Rayzit*.

Crowdsourcing: Crowdsourcing is a process that involves outsourcing tasks to a distributed group of an undefined crowd through an open call for monetary or ethical benefit. There are many examples of crowdsourcing systems, e.g., MTurk.com,

Table 7.1: Taxonomy of anonymous social net applications

Application	Distance	kNN	Anonymous	Location
Facebook	No	No	No	Yes
Twitter	No	No	No	Yes
Foursquare	Yes	No	No	Yes
SnapChat	No	No	No	Yes
WeChat	Yes	No	No	Yes
Secret	Yes	No	Partial	Yes
YikYak	10 miles	No	Yes	Yes
Whisper	Yes	No	Partial	Yes
Rayzit	Custom	Yes	Yes	Yes

Topcoder.com, CrowdFlower.com, and TaskRabbit.com. One of the best crowdsourcing system examples is the ESP game, where the users implicitly collaborate to label images as a side effect while playing the game [158].

It is known that social networks need a mechanism to detect intruders, spammers, attackers [159]. Anonymous services have emerged that can filter out abusive and harmful messages. Most social networks are currently outsourcing this difficult task to external surveillance companies, some of which embrace crowdsourcing solutions. In contrast, *Rayzit* uses an in-situ crowdsourcing mechanism to remove abusive messages.

Conversational Commerce: There is a shift towards using social networks, messaging platforms and even simple SMS to deliver products to customers, who prefer expressing their needs by texting instead of looking for online vendors and following the classical online retail procedure. Magic[10] and Scratch[11], allow users to register to an SMS service and ask for anything they need. The applications personnel figures out a way and a price to deliver, and after confirming with the requestor, the delivery plan is implemented. The application Nativeapp.com is offering advanced travel agency services through messaging. The instant messaging app imQQ.com has been offering shopping services to its users since 2007 in China using virtual money. Path Talk[12]

[10]Magic: `http://getmagicnow.com/`

[11]Scratch: `http://www.tryscratch.com/`

[12]Path Talk: `http://path.com/talk`

allows its users to message businesses directly to order goods.

Online VoIP and messaging applications, such as Skype.com, Viber.com, Tango.me, Kakao.com, LINE.me, WeChat.com, Facebook Messenger, and even SnapChat already sell digital products and are moving towards the direction of general e-commerce[13].

AkNN Computation: The *k Nearest Neighbors (kNN)* of a user u from some dataset U, denoted as $kNN(u, U)$, are the k users that have the most similar attributes to u [25]. kNN search is a classical Computer Science problem with many centralized algorithms that find applications in computational geometry [132–134], image processing [136], spatial databases [137, 138, 160], and social networks [161]. An *All kNN (AkNN)* query, viewed as a generalization of the basic kNN query, computes the $kNN(u, U)$ result for every $u \in U$ and has a quadratic worst-case bound.

There are several AkNN algorithms optimized for offline analytics, aimed at *memory-resident data* [122, 132–134, 140], but also *disk-resident data* [116, 137, 138, 162]. Moreover, there are previous works that were utilizing the kNN properties in order to extend a social network to gain accuracy and predictability [119].

Big Data and Data Mining: Social networks have billions of users generating petabytes of data every day. Additionally, the growth of smartphones, wearables and portable devices add new information such as location, biometrics values and environmental metrics increasing the internet traffic. In most cases the multidimensional data values consist of the user's location and the profile of the user as described by Jensen et al [163].

As a result, Big data infrastructures are necessary in order to accommodate the huge volume of the multidimensional data. Moreover, it is common that data is communicated and stored using semistructured formats such as JSON, XML etc. Consequently, a high performance NoSQL database management system is typically used to store, edit and manipulate the aforementioned data.

7.2.4 The *Rayzit* Architecture

In this subsection, we describe our *Rayzit* crowd messaging architecture. The architecture consists of various components that provide the required functionalities for the end-user.

[13]Forbes: `http://goo.gl/s0gYEM`

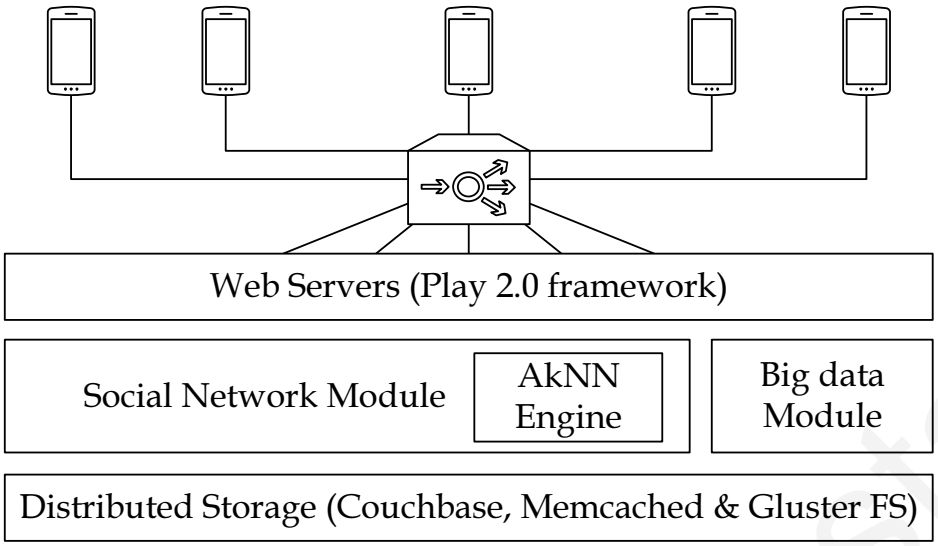

Figure 7.4: Rayzit *distributed architecture*

The *Rayzit* architecture comprises of several modules utilizing state-of-the-art technology. It features a HAProxy[14] HTTP load balancer to distribute the workload to respective WWW servers (see Figure 7.4). Each WWW features a Play 2.0[15] server and a Couchbase NoSQL document store[16] for storing the messages posted by our users. In Couchbase, data is stored across the cluster in JSON format, which is indexed and directly exposed to the *Rayzit* Web 2.0 API. In the backend, we also run the computing cluster that carries out the AkNN computation as we have already discussed in Section 7.2.3. The results are passed to the WWW servers through main memory (i.e., Memcached) every few seconds.

Load balancer and Web servers: *Rayzit* has multiple web servers in order to support a large amount of requests at the same time. In order to achieve this, we use HAproxy, a lightweight load balancer, to distribute the load to each web server. The web servers are deployed using the Play 2.0 framework, which is stateless, scalable, fast and thus can absorb large numbers of requests. In addition, the web servers host the Faye[17] system in order to serve a live feed for the end-user. Faye is a publish-subscribe messaging system based on the Bayeux protocol.

[14]HAProxy: `http://haproxy.1wt.eu/`
[15]Play: `http://www.playframework.com/`
[16]Couchbase: `http://www.couchbase.com/`
[17]Faye: `http://faye.jcoglan.com/`

Social Network Module: The architecture can incorporate any application that wants to use the kNN network. In our case we have implemented and incorporated our *Rayzit* messaging app. The users can broadcast their messages to the k closest users or forward messages they see in order to propagate them to their own k closest neighbors. The proposed architecture maintains a core engine that generates the needed AkNN links in a distributed manner described next.

AkNN Engine: Finding the k geographically nearest neighbors of all users is known as the *All k Nearest Neighbors* (AkNN) query. AkNN query processing algorithms are used in several offline analytic queries. Recent techniques have shown how to scale such queries to shared-nothing cloud architectures using algorithms implemented in MapReduce. Yet, these algorithms are very slow for online operational queries, as needed by *Rayzit*, even when operating in distributed main-memory (e.g., Tachyon as opposed to HDFS).

Rayzit mandates AkNN query executions every few seconds, so that users can exchange microblog messages with their k geographically closest neighbors in an online manner. For this reason, we have developed a fast AkNN algorithm in our previous work [122]. This implementation makes use of a fast partitioning algorithm that achieves good load balancing, and a technique to identify a minimal set of points to exchange between neighboring partitions in order to compute the correct answer to the AkNN query in a distributed and parallel fashion.

Distributed Storage: *Rayzit* architecture was built on a Big data NoSQL document-oriented database that is optimized for intensive applications. Couchbase was chosen in order to maximize the scalability and the performance of our system. We achieve a low latency access to the high volume of documents due to the Memcached layer [164]. Note, that our *Rayzit* app also allows users to attach files in their posts (i.e., photos, audio and videos). Therefore, a distributed file system needs to complement our proposed architecture in order to handle these media files. We use the Gluster[18] file system in order to secure the capability of scaling up to several petabytes and maintain a 24/7 service for thousands of clients.

[18]Gluster: `http://www.gluster.org/`

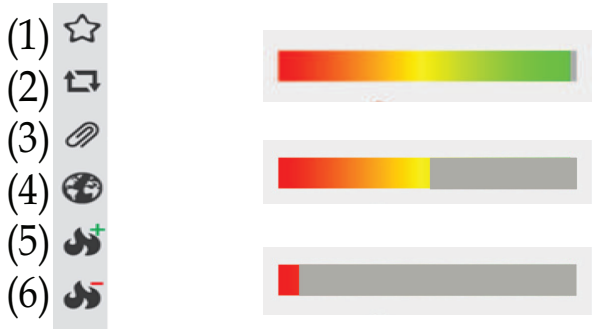

Figure 7.5: (Left) The icons in Rayzit*: A user can "star" a post (1), "re-rayz" a post to their k closest peers (2), attach a file to new post (3), setting a distance cut-off parameter if needed (4), "power up" (5) or "power down" (6) a post and its author. (Right) The anti-spam power bar: the power of each user decreases as they post, and increases every 24 hours or every time one of their posts is "powered up".*

Big Data Management Module: *Rayzit* has thousands of users and it is necessary to monitor the users' behavior considering offensive, bullying and inappropriate messages. Consequently, we construct a Big data module in order to feed the administrator monitor panel with the required information.

Each message can be reported by any user in the crowd and the report is stored in our NoSQL database. We adopt a crowdsourcing majority vote technique to reach a decision. A single query extracts the reported rayz messages from the database, which consequently can be removed.

7.2.5 Rayzit Application

Rayzit was launched in 2014 and has redefined anonymous messaging and social interactions. The main functionality of an anonymous social network is to provide ways for users to publish opinions, thoughts or gossip while preserving anonymity. Connecting each user with the k geographical closest peers (strangers) allows them to communicate anonymously. These connections are updated as users move.

The *Rayzit* mobile application allows users to send messages (rayzes) with media attachments, and reply to rayz posts based on a kNN network. In more details, the messages will be sent to the k nearest users. For each rayz that a user sends, the power of the user is decreasing (see Figure 7.5 (right)). When the power of a user reaches 0 they can not post or reply any more (see Figure 7.5). This functionality was incorporated in order to minimize spamming. The power of the user is fully reset every

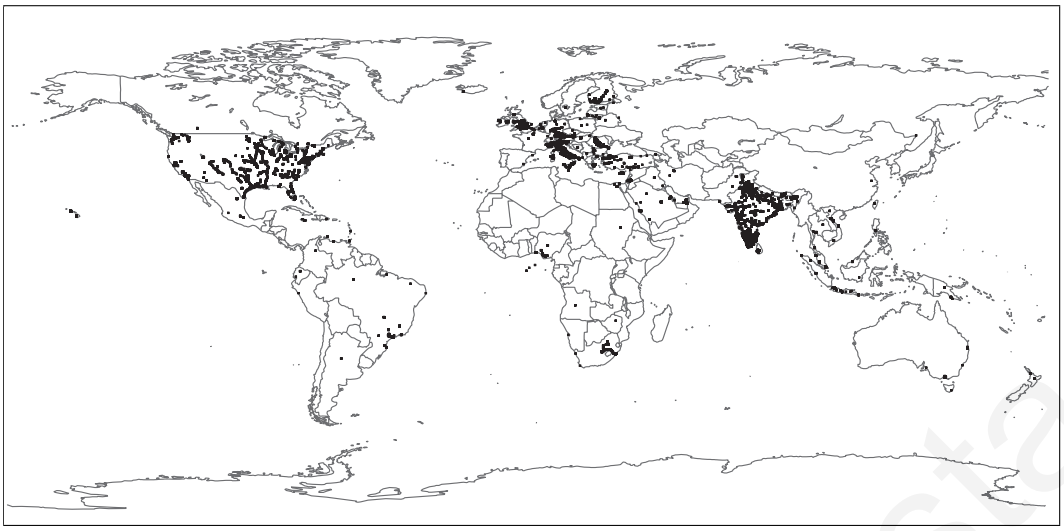

Figure 7.6: Rayzit *global user community.*

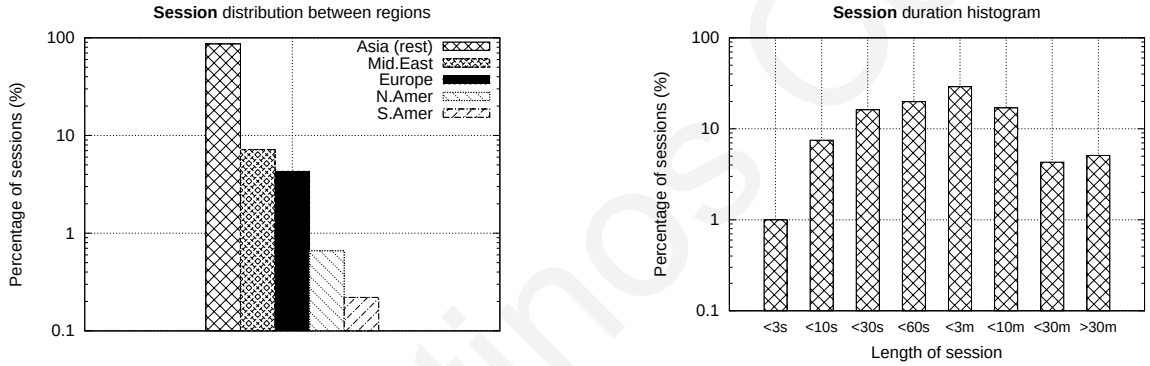

Figure 7.7: (Left) Distribution of Rayzit *users across regions in log scale. (Right) Average duration of each user session (reading and/or posting) in log scale.*

24 hours or if the user's rayz is powered up by another user. In addition, a user can star or re-rayz any rayz which they consider interesting. Re-rayzing is the procedure of re-posting an existing rayz to one's own k nearest peers (similar to the re-tweet functionality in Twitter) and therefore propagating the post to more users.

Figure 7.6 shows the collection of active users in *Rayzit*. It is evident that most users have their nearest neighbors inside a small area. Now consider the scenario, where somebody has lost their dog and want to start looking in the neighborhood in order to find it and at the same time ask for help from the social network. *Rayzit* offers this functionality out of the box. The user can post a picture of the dog, which the users in the vicinity will see. If someone sees a dog that matches the attached picture (see in Figure 7.3 (right)), they can reply to the post with whereabouts and/or photos of the lost pet.

In order to comprehend how the *Rayzit* architecture works, consider the aforemen-

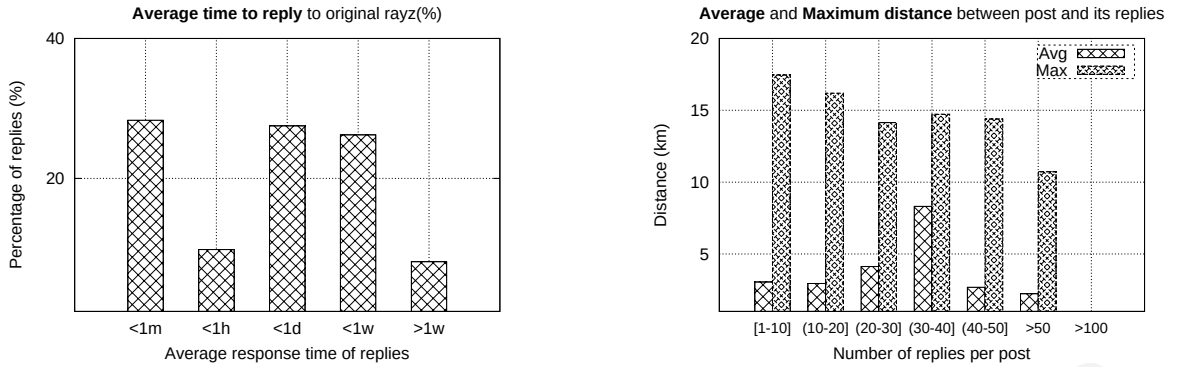

Figure 7.8: (Left) Average time elapse between a post (rayz) and all its replies. (Right) The average and maximum distance between the locations of a post (rayz) and its replies, grouped by number of replies per post.

tioned scenario in terms of communication interactions. First of all, the client sends the rayz attaching a photo of the lost dog. The post, the photo, and their location is sent to the load balancer, which assigns them to a web server. The responsible web server stores the rayz through the Big data module. Then the social network module uses the dynamic social AkNN network computed by the AkNN engine in order to propagate the rayz to the k nearest neighbors. The k nearest neighbors will receive the rayz into their live feed and they can reply if they have any information.

7.2.6 Data Analysis and Evaluation

In this subsection, we have collected data produced by our *Rayzit* application and provide a brief data analysis. The data consists of several elements such as UUID and timestamp. Furthermore, we conduct experiments for evaluating and analyzing the performance of *Rayzit*.

Our evaluation focuses on two aspects: (1) the average percentage of replies per rayz at a specific time; and (2) how the distance affects the number of replies. We used Flurry.com analytics to get usage and distribution statistics of *Rayzit* app.

Session Measurements: In this experiment we study the usage of *Rayzit* app in respect to region and duration of interaction.

The histograms in Figure 7.7 show the distribution of sessions in percentage. The session length is defined simply as the length of time between the start application event and the end application event. The Figure 7.7 (right) plots the distribution of the sessions by displaying the number of sessions for which the session length falls into

predefined duration slots.

Figure 7.7 (left) plots the distribution of sessions in percentage for each region. More than 87.6% of sessions are created in Asia, and more than 7.2% of them in Middle East. 4.3% of all sessions are created in Europe. Only 0.7% of sessions are created in North America. Only 0.2% of the sessions have been created in South America.

Figure 7.7 (right) plots the distribution of *Rayzit*'s sessions across duration time slots. It is noticeable that 1% of the sessions is completed within 3 seconds. 7.50% completed between 3 and 10 seconds and 16.20% observed between 10 and 30 seconds. 19% of the sessions completed between 30 and 60 seconds. The time slot of 1 to 3 minutes has the maximum percentage with 29%. Furthermore, 17% completed within 10 minutes and 4.3% between 10 and 30 minutes. Finally, only the 4.3% completed after 30 minutes. Consequently, the AkNN engine needs to deliver the dynamic kNN graph in less than a minute to satisfy most users.

Interaction Measurements: In this experiment we study how users interact through *Rayzit*, and in particular how fast a user replies to a post and how the distance affects interaction.

We used real data that was collected through the *Rayzit* back-end. The data include the time and location stamp of a post or a reply. The time between a rayz and a reply is presented in time slots: $(< 1m)$ less than a minute, $(< 1h)$ between a minute and an hour, $(< 1d)$ between an hour and a day, $(< 1w)$ between a day and a week and $(> w)$ more than a week. The average and maximum distances of the replies for each rayz were measured in order to depict how the distance affects the number of replies.

Figure 7.8 (left) plots the distribution of the reply arrival time, which is the time between each reply and the original rayz. 28% of replies arrive within the first minute of the original rayz, and more than 28% of replies arrive within a day. Only 10% of replies arrive with in the first hour. 26% of replies arrive between the first day and the first week. Only the 8% of replies arrive one week or more after the rayz creation. This confirms our intuition for the temporal dimension of a rayz: it is very unlikely to get attention after a while.

Figure 7.8 (right) plots the distance that a rayz can reach. It is noticeable that rayz posts with 1 - 50 replies have a maximum distance of almost 15km. Rayz posts with more than 50 replies have 10km maximum distance and 2km average distance. Moreover, rayz posts with more than 100 replies have only 8m maximum and 6m

average distance. This validates our initial claim that the nearest users are more likely to be related to a specific subject than the faraway users.

7.2.7 Summary

This section presents an innovative architecture for anonymous dynamic social networks, coined Rayzit, which enables anonymous interactions with the k nearest neighbors using state-of-the-art technology. An industrial quality application was implemented that utilizes the aforementioned architecture. We presented and analyze the data collected from this application and draw interesting conclusions about this new type of dynamic social networking. Our experimental results also confirm our initial hypothesis that the number of the replies is related to the location of the original rayz (i.e., the rayz with the most replies has a maximum range of 8 meters).

Chapter 8

Conclusions and Future Work

In this chapter, we summarize the contributions of this thesis and present future directions for research.

Storage and Indexing: The completion of the PhD thesis will provide a good starting point to start developing new storage and indexing techniques using decay and compression. This will allow new researchers to accelerate their work for efficient storage and indexing over spatial big data.

Operators: In this PhD thesis, we focused on join operators due to their high complexity and popularity in a real world applications. Particularly, we implemented new distributed algorithms with high performance in order to answer queries like $AkNN$. This will allow new researchers to explore and compare new join algorithms for spatial big data with the ones we have already published.

Applications: Novel applications could be implemented over the $SPATE^{+}$ architecture. Through the PhD thesis, *Rayzit* and $SPATE$ applications were developed providing detailed analysis and evaluation of the underlying architecture.

8.1 *SPATE*$^{+}$: A Performance-driven Architecture for Spatial Big Data Management

$SPATE^{+}$ presents a complete performance-driven architecture, which provides efficient storage, indexing and query processing of spatial big data. $SPATE^{+}$ minimizes the storage space needed to incrementally retain data *over time* and the response time for

spatiotemporal data exploration queries over recent data. Additionally, we developed a novel decaying operator for Telco Big Data (TBD), coined *TBD-DP*. *TBD-DP* relies on existing ML algorithms to abstract TBD into compact models that can be stored and queried when necessary.

SPATE$^+$ integrates techniques that generate the kNN graph of an arbitrary crowd of smartphone users that interconnect through a short-range communication technology, such as, Wi-Fi Direct, 3G/LTE direct or Bluetooth v4.0 (BLE). We present two efficient algorithms, namely *Akin+* and *Prox+*, optimized to work on a resource-limited mobile device. These algorithms partition the user space and compute shared candidate sets per partition. *Prox+* uses a custom heap data structure to update the candidate set as new users are inserted, whereas *Akin+* uses a bulk bottom-up construction of a simple heap to compute the candidate set once all users have been inserted. *Spitfire* is a scalable and high-performance distributed operator that solves the AkNN problem using a shared-nothing cloud infrastructure. Our *SPATE*$^+$ operator offers several advantages over the state-of-the-art algorithms in terms of efficient partitioning, replication and refinement.

We have implemented two real applications on top of *SPATE*$^+$: (i) SPATE is implemented in HTML5/CSS3 along with extensive AngularJS. An illustrative network exploration interface is shown in Figure 4.5. We have implemented a query sidebar that allows the user to execute a variety of template queries. The query bar includes snapshot queries and recurring queries (in the form of a time-machine) for drop calls and downflux/upflux, heatmap statistics and settings. Furthermore, quick access buttons are provided so that user is able to choose between the available network modalities (2G, 3G, 4G); and (ii) Rayzit enables anonymous interactions with the k nearest neighbors using the *Spitfire* algorithm resided in the proposed architecture.

We have measured the efficiency of our proposition using a real telco trace and a variety of telco-specific tasks, such as OLAP and OLTP querying, clustering, regression and privacy sanitizing, and showed that we can achieve comparable response times to the state-of-the-art with an order of magnitude less storage space. Additionally, we measured the efficiency of the proposed operator using a ∼10GB anonymized real telco network trace and our experimental results in Tensorflow over HDFS are extremely encouraging as they show that *TBD-DP* saves an order of magnitude storage space while maintaining a high accuracy on the recovered data. Considering the *operator* category, our experiments verify the theoretical efficiency and show that *Prox+* and

Akin+ are very well suited for large scale and skewed data scenarios. Theoretical analysis and experimental evaluation show that *Spitfire* outperforms existing algorithms reported in recent literature, achieving scalability both on the number of users and on the number of k nearest neighbors.

8.2 Future Work

In this chapter we describe future directions that can be realized after the completion of the PhD thesis.

In the future, we aim to investigate a variety of advanced smart city application scenarios on top of *SPATE*$^{+}$, namely an automated car traffic mapping system and an emergency recovery system after natural disasters.

We will generalize the data decaying operators beyond TBD into new domains (e.g., signals from other type of IoT). This task might give space to new ML algorithms. Additionally, we aim to theoretically derive the accuracy/efficiency bounds of our data postdiction framework. Furthermore, we will provide extensions to allow streaming computation of *TBD-DP* learning with velocity data. We also plan to carry out an extensive experimental study that will focus solely on decaying of big data.

Additionally, we plan to study the temporal extensions to support more gracefully higher-rate AkNN scenarios with streaming data, as well as AkNN queries over high-dimensional data. We also plan to provide an approximate AkNN version of *Spitfire*. Finally, we are interested in developing online geographic hashing techniques at the network load-balancing level and also port our developments to general open-source large-scale data processing architectures (e.g., Apache Spark [162] and Apache Flink [33]).

Finally, we will investigate further the advantages of the dynamic AkNN graph in order to enhance user experience in our social application. In addition, better content filtering mechanisms will be studied in order to provide the best possible quality to the users.

8.3 Broad Impact

This work has enabled the development of a new generation SBD systems that provide new user experiences and new opportunities of growth in the telecommunication sector. Consequently, all kinds of applications based on our research can be further enhanced for telemedicine (e.g., remote patient monitoring, more accurate diagnostics), scientific investment (e.g., environmental startups), security (e.g., cyber security attack recognition), entertainment (e.g., optimized media scheduling, churn prevention), e-commerce (e.g., targeted advertisements, location based marketing) and games (e.g behavioral analysis, personalized modes).

Bibliography

[1] C. Costa, C. Anastasiou, G. Chatzimilioudis, and D. Zeinalipour-Yazti, "Rayzit: An anonymous and dynamic crowd messaging architecture," in *2015 16th IEEE International Conference on Mobile Data Management*, vol. 2, June 2015, pp. 98–103.

[2] A. Barczyk, A. Mughal, H. Newman, I. Legrand, M. Bredel, R. Voicu, V. Lapadatescu, and T. Wildish, "Towards managed terabit/s scientific data flows," in *Proceedings of the Fourth International Workshop on Network-Aware Data Management*, ser. NDM '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 23–27. [Online]. Available: http://dx.doi.org/10.1109/NDM.2014.8

[3] M. Yuan, K. Deng, J. Zeng, Y. Li, B. Ni, X. He, F. Wang, W. Dai, and Q. Yang, "Oceanst: A distributed analytic system for large-scale spatiotemporal mobile broadband data," *Proc. VLDB Endow.*, vol. 7, no. 13, pp. 1561–1564, Aug. 2014. [Online]. Available: http://dx.doi.org/10.14778/2733004.2733030

[4] "Apache Hadoop." [Online]. Available: http://hadoop.apache.org/

[5] "Hadoop Distributed File System (HDFS)." [Online]. Available: http://hadoop.apache.org/

[6] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: ACM, 1984, pp. 47–57. [Online]. Available: http://doi.acm.org/10.1145/602259.602266

[7] E. H. Jacox and H. Samet, "Spatial join techniques," *ACM Trans. Database Syst.*, vol. 32, no. 1, Mar. 2007. [Online]. Available: http://doi.acm.org/10.1145/1206049.1206056

[8] W. Pedrycz and S.-M. Chen, Eds., *Information Granularity, Big Data, and Computational Intelligence*. Springer International Publishing, 2015.

[9] M. Watanabe and T. Suzumura, "How social network is evolving?: A preliminary study on billion-scale twitter network," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13 Companion. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2013, pp. 531–534. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487788.2487988

[10] T. Steiner, "Dc proposal: Enriching unstructured media content about events to enable semi-automated summaries, compilations, and improved search by leveraging social networks," in *Proceedings of the 10th International Conference on The Semantic Web - Volume Part II*, ser. ISWC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 365–372. [Online]. Available: http://dl.acm.org/citation.cfm?id=2063076.2063108

[11] M. Avvenuti, S. Cresci, A. Marchetti, C. Meletti, and M. Tesconi, "Ears (earthquake alert and report system): A real time decision support system for earthquake crisis management," in *Proceedings of the 20th ACM SIGKDD*

International Conference on Knowledge Discovery and Data Mining, ser. KDD '14. New York, NY, USA: ACM, 2014, pp. 1749–1758. [Online]. Available: http://doi.acm.org/10.1145/2623330.2623358

[12] G. Wang, B. Wang, T. Wang, A. Nika, H. Zheng, and B. Y. Zhao, "Whispers in the dark: Analysis of an anonymous social network," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, ser. IMC '14. New York, NY, USA: ACM, 2014, pp. 137–150. [Online]. Available: http://doi.acm.org/10.1145/2663716.2663728

[13] S. Zhang, Y. Yang, W. Fan, L. Lan, and M. Yuan, "Oceanrt: Real-time analytics over large temporal data," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 1099–1102. [Online]. Available: http://doi.acm.org/10.1145/2588555.2594513

[14] Y. Huang, F. Zhu, M. Yuan, K. Deng, Y. Li, B. Ni, W. Dai, Q. Yang, and J. Zeng, "Telco churn prediction with big data," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD. New York, NY, USA: ACM, 2015, pp. 607–618. [Online]. Available: http://doi.acm.org/10.1145/2723372.2742794

[15] A. P. Iyer, L. E. Li, and I. Stoica, "Celliq: Real-time cellular network analytics at scale," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 309–322. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789792

[16] C. Luo, J. Zeng, M. Yuan, W. Dai, and Q. Yang, "Telco user activity level prediction with massive mobile broadband data," *ACM Trans. Intell. Syst. Technol.*, vol. 7, no. 4, pp. 63:1–63:30, May 2016. [Online]. Available: http://doi.acm.org/10.1145/2856057

[17] C. Costa, G. Chatzimilioudis, D. Zeinalipour-Yazti, and M. F. Mokbel, "Efficient exploration of telco big data with compression and decaying," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, ser. ICDE'17. San Diego, CA, USA: IEEE, April 2017, pp. 1332–1343.

[18] C. LaChapelle, "The cost of data storage and management: where is the it headed in 2016?" 2016. [Online]. Available: http://www.datacenterjournal.com/cost-data-storage-management-headed-2016/

[19] Z. Li, A. Mukker, and E. Zadok, "On the importance of evaluating storage systems' $costs," in *6th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage)*, Philadelphia, PA, 2014.

[20] M. Athanassoulis, M. S. Kester, L. M. Maas, R. Stoica, S. Idreos, A. Ailamaki, and M. Callaghan, "Designing access methods: The rum conjecture," in *Intl. Conf. on Ext. Database Technology (EDBT)*, 2016.

[21] F. Zhu, C. Luo, M. Yuan, Y. Zhu, Z. Zhang, T. Gu, K. Deng, W. Rao, and J. Zeng, "City-scale localization with telco big data," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*,

ser. CIKM. New York, NY, USA: ACM, 2016, pp. 439–448. [Online]. Available: http://doi.acm.org/10.1145/2983323.2983345

[22] C. Costa, G. Chatzimilioudis, D. Zeinalipour-Yazti, and M. F. Mokbel, "Towards real-time road traffic analytics using telco big data," in *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics, BIRTE, Munich, Germany, August 28, 2017*, 2017, pp. 5:1–5:5. [Online]. Available: http://doi.acm.org/10.1145/3129292.3129296

[23] E. Savitz, "Forbes magazine," 2012, [Online; April 16, 2012]. [Online]. Available: https://goo.gl/eM1uwV

[24] E. C. H. Ngai, M. B. Srivastava, and J. Liu, "Context-aware sensor data dissemination for mobile users in remote areas," in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 2711–2715.

[25] N. Roussopoulos, S. Kelley, and F. Vincent, "Nearest neighbor queries," in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '95. New York, NY, USA: ACM, 1995, pp. 71–79. [Online]. Available: http://doi.acm.org/10.1145/223784.223794

[26] W. Lu, Y. Shen, S. Chen, and B. C. Ooi, "Efficient processing of k nearest neighbor joins using mapreduce," *Proc. VLDB Endow.*, vol. 5, no. 10, pp. 1016–1027, Jun. 2012. [Online]. Available: http://dx.doi.org/10.14778/2336664.2336674

[27] G. Chatzimilioudis, D. Zeinalipour-Yazti, W. Lee, and M. D. Dikaiakos, "Continuous all k-nearest-neighbor querying in smartphone networks," in *13th IEEE International Conference on Mobile Data Management, MDM 2012, Bengaluru, India, July 23-26, 2012*, 2012, pp. 79–88. [Online]. Available: http://dx.doi.org/10.1109/MDM.2012.19

[28] T. White, *Hadoop: The Definitive Guide*, 1st ed. O'Reilly Media, Inc., 2009.

[29] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 2–2. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228301

[30] J. LeFevre, J. Sankaranarayanan, H. Hacigumus, J. Tatemura, N. Polyzotis, and M. J. Carey, "Miso: Souping up big data query processing with a multistore system," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. New York, NY, USA: ACM, 2014, pp. 1591–1602. [Online]. Available: http://doi.acm.org/10.1145/2588555.2588568

[31] D. Jiang, G. Chen, B. C. Ooi, K.-L. Tan, and S. Wu, "epic: An extensible and scalable system for processing big data," *Proc. VLDB Endow.*, vol. 7, no. 7, pp. 541–552, Mar. 2014. [Online]. Available: http://dx.doi.org/10.14778/2732286.2732291

[32] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Technical Report 1999-66, November 1999, previous number = SIDL-WP-1999-0120. [Online]. Available: http://ilpubs.stanford.edu:8090/422/

[33] S. Ewen, S. Schelter, K. Tzoumas, D. Warneke, and V. Markl, "Iterative parallel data processing with stratosphere: An inside look," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 1053–1056. [Online]. Available: http://doi.acm.org/10.1145/2463676.2463693

[34] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, and M. Zaharia, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 1383–1394. [Online]. Available: http://doi.acm.org/10.1145/2723372.2742797

[35] R. S. Xin, J. Rosen, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Sql and rich analytics at scale," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 13–24. [Online]. Available: http://doi.acm.org/10.1145/2463676.2465288

[36] D. J. DeWitt, R. H. Katz, F. Olken, L. D. Shapiro, M. R. Stonebraker, and D. A. Wood, "Implementation techniques for main memory database systems," in *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '84. New York, NY, USA: ACM, 1984, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/602259.602261

[37] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun, "Making sense of performance in data analytics frameworks," in *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 293–307. [Online]. Available: http://dl.acm.org/citation.cfm?id=2789770.2789791

[38] J. Dittrich and J.-A. Quiané-Ruiz, "Efficient big data processing in hadoop mapreduce," *Proc. VLDB Endow.*, vol. 5, no. 12, pp. 2014–2015, Aug. 2012. [Online]. Available: http://dx.doi.org/10.14778/2367502.2367562

[39] A. D. Sarma, F. N. Afrati, S. Salihoglu, and J. D. Ullman, "Upper and lower bounds on the cost of a map-reduce computation," *Proc. VLDB Endow.*, vol. 6, no. 4, pp. 277–288, Feb. 2013. [Online]. Available: http://dx.doi.org/10.14778/2535570.2488334

[40] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "Mapreduce online," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 21–21. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855732

[41] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen, "Bigbench: Towards an industry standard benchmark for big data analytics,"

in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '13. New York, NY, USA: ACM, 2013, pp. 1197–1208. [Online]. Available: http://doi.acm.org/10.1145/2463676.2463712

[42] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 295–308. [Online]. Available: http://dl.acm.org/citation.cfm?id=1972457.1972488

[43] P. S. Pacheco, *Parallel Programming with MPI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996.

[44] Y. Cheng, A. Gupta, and A. R. Butt, "An in-memory object caching framework with adaptive load balancing," in *Proceedings of the Tenth European Conference on Computer Systems*, ser. EuroSys '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:16. [Online]. Available: http://doi.acm.org/10.1145/2741948.2741967

[45] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica, "Tachyon: Reliable, memory speed storage for cluster computing frameworks," in *Proceedings of the ACM Symposium on Cloud Computing*, ser. SOCC '14. New York, NY, USA: ACM, 2014, pp. 6:1–6:15. [Online]. Available: http://doi.acm.org/10.1145/2670979.2670985

[46] X. Liu, N. Iftikhar, and X. Xie, "Survey of real-time processing systems for big data," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, ser. IDEAS '14. New York, NY, USA: ACM, 2014, pp. 356–361. [Online]. Available: http://doi.acm.org/10.1145/2628194.2628251

[47] N. Stojanovic, L. Stojanovic, Y. Xu, and B. Stajic, "Mobile cep in real-time big data processing: Challenges and opportunities," in *Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '14. New York, NY, USA: ACM, 2014, pp. 256–265. [Online]. Available: http://doi.acm.org/10.1145/2611286.2611311

[48] E. Kuehn, M. Fischer, C. Jung, A. Petzold, and A. Streit, "Monitoring data streams at process level in scientific big data batch clusters," in *Proceedings of the 2014 IEEE/ACM International Symposium on Big Data Computing*, ser. BDC '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 90–95. [Online]. Available: http://dx.doi.org/10.1109/BDC.2014.21

[49] L. Braun, T. Etter, G. Gasparis, M. Kaufmann, D. Kossmann, D. Widmer, A. Avitzur, A. Iliopoulos, E. Levy, and N. Liang, "Analytics in motion: High performance event-processing and real-time analytics in the same database," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD. New York, NY, USA: ACM, 2015, pp. 251–264. [Online]. Available: http://doi.acm.org/10.1145/2723372.2742783

[50] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, "Weaving relations for cache performance," in *Proceedings of the 27th International Conference on Very Large Data Bases*, ser. VLDB. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 169–180. [Online]. Available: http://dl.acm.org/citation.cfm?id=645927.672367

[51] E. Bouillet, R. Kothari, V. Kumar, L. Mignet, S. Nathan, A. Ranganathan, D. S. Turaga, O. Udrea, and O. Verscheure, "Processing 6 billion cdrs/day: From research to production (experience report)," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '12. New York, NY, USA: ACM, 2012, pp. 264–267.

[52] Y. Zhong, X. Zhu, and J. Fang, "Elastic and effective spatio-temporal query processing scheme on hadoop," in *Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*, ser. BigSpatial '12. New York, NY, USA: ACM, 2012, pp. 33–42. [Online]. Available: http://doi.acm.org/10.1145/2447481.2447486

[53] S. Alsubaiee, A. Behm, R. Grover, R. Vernica, V. Borkar, M. J. Carey, and C. Li, "Asterix: Scalable warehouse-style web data integration," in *Proceedings of the Ninth International Workshop on Information Integration on the Web*, ser. IIWeb '12. New York, NY, USA: ACM, 2012, pp. 2:1–2:4. [Online]. Available: http://doi.acm.org/10.1145/2331801.2331803

[54] A. M. Aly, A. R. Mahmood, M. S. Hassan, W. G. Aref, M. Ouzzani, H. Elmeleegy, and T. Qadah, "Aqwa: Adaptive query workload aware partitioning of big spatial data," *Proc. VLDB Endow.*, vol. 8, no. 13, pp. 2062–2073, Sep. 2015. [Online]. Available: http://dx.doi.org/10.14778/2831360.2831361

[55] I. Kelley and J. Blumenstock, "Computational challenges in the analysis of large, sparse, spatiotemporal data," in *Proceedings of the Sixth International Workshop on Data Intensive Distributed Computing*, ser. DIDC '14. New York, NY, USA: ACM, 2014, pp. 41–46. [Online]. Available: http://doi.acm.org/10.1145/2608020.2608025

[56] L. Sweeney, "K-anonymity: A model for protecting privacy," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 10, no. 5, pp. 557–570, Oct. 2002. [Online]. Available: http://dx.doi.org/10.1142/S0218488502001648

[57] X. Hu, M. Yuan, J. Yao, Y. Deng, L. Chen, Q. Yang, H. Guan, and J. Zeng, "Differential privacy in telco big data platform," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1692–1703, Aug. 2015. [Online]. Available: http://dx.doi.org/10.14778/2824032.2824067

[58] A. Eldawy and M. F. Mokbel, "SpatialHadoop: A MapReduce Framework for Spatial Data," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, 2015, pp. 1352–1363.

[59] H. Samet, "Modern database systems," W. Kim, Ed. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995, ch. Spatial Data Structures, pp. 361–385. [Online]. Available: http://dl.acm.org/citation.cfm?id=187362.187464

[60] N. Crary, B. Tang, and S. Taase, "Data preservation in data-intensive sensor networks with spatial correlation," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15. New York, NY, USA: ACM, 2015, pp. 7–12. [Online]. Available: http://doi.acm.org/10.1145/2757384.2757389

[61] P. Andreou, A. Pamboris, D. Zeinalipour-Yazti, P. K. Chrysanthis, and G. Samaras, "Etc: Energy-driven tree construction in wireless sensor networks," in *2009*

Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, May 2009, pp. 513–518.

[62] S. Wang, H. Hu, T. Lin, Y. Liu, A. Padmanabhan, and K. Soltani, "Cybergis for data-intensive knowledge discovery," *SIGSPATIAL Special*, vol. 6, no. 2, pp. 26–33, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2744700.2744704

[63] S. Shekhar, V. Gunturi, M. R. Evans, and K. Yang, "Spatial big-data challenges intersecting mobility and cloud computing," in *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, ser. MobiDE '12. New York, NY, USA: ACM, 2012, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/2258056.2258058

[64] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, "Hadoop gis: A high performance spatial data warehousing system over mapreduce," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1009–1020, Aug. 2013. [Online]. Available: http://dx.doi.org/10.14778/2536222.2536227

[65] C. Costa, G. Chatzimilioudis, D. Zeinalipour-Yazti, and M. F. Mokbel, "Spate: Compacting and exploring telco big data," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, April 2017, pp. 1419–1420.

[66] Ericsson.com, "Cellular Networks For Massive IoT – enabling low power wide area applications," 2016. [Online]. Available: https://goo.gl/Sf2Cj4

[67] J. Reades, F. Calabrese, A. Sevtsuk, and C. Ratti, "Cellular census: Explorations in urban data collection," *IEEE Pervasive Computing*, vol. 6, no. 3, pp. 30–38, 2007.

[68] H. Chen, R. H. Chiang, and V. C. Storey, "Business intelligence and analytics: From big data to big impact." *MIS quarterly*, vol. 36, no. 4, pp. 1165–1188, 2012.

[69] TeraLab, "TeraLab Data Science for Europe," 2016. [Online]. Available: http://www.teralab-datascience.fr/

[70] A. Eldawy, M. F. Mokbel, S. Alharthi, A. Alzaidy, K. Tarek, and S. Ghani, "Shahed: A mapreduce-based system for querying and visualizing spatio-temporal satellite data," in *2015 IEEE 31st International Conference on Data Engineering*, April 2015, pp. 1585–1596.

[71] Y. Chen, A. Ganapathi, and R. H. Katz, "To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency," in *Proceedings of the First ACM SIGCOMM Workshop on Green Networking*, ser. Green Networking '10, 2010, pp. 23–28.

[72] B. Welton, D. Kimpe, J. Cope, C. M. Patrick, K. Iskra, and R. Ross, "Improving i/o forwarding throughput with data compression," in *2011 IEEE Intl. Conference on Cluster Computing*, Sept 2011, pp. 438–445.

[73] T. Bicer, J. Yin, D. Chiu, G. Agrawal, and K. Schuchardt, "Integrating online compression to accelerate large-scale data analytics applications," in *Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. IEEE, May 2013, pp. 1205–1216.

[74] M. L. Kersten, "Big data space fungus," in *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2015, Online Proceedings*, 2015.

[75] M. Stonebraker, R. Castro, F. Dong Deng, and M. Brodie, "Database decay and what to do about it." 2016. [Online]. Available: https://goo.gl/tJNa9m

[76] S. Lakshminarasimhan, N. Shah, S. Ethier, S. Klasky, R. Latham, R. Ross, and N. F. Samatova, "Compressing the incompressible with isabela: In-situ reduction of spatio-temporal data," in *European Conference on Parallel Processing*. Springer, 2011, pp. 366–379.

[77] E. R. Schendel, Y. Jin, N. Shah, J. Chen, C.-S. Chang, S.-H. Ku, S. Ethier, S. Klasky, R. Latham, R. Ross *et al.*, "Isobar preconditioner for effective and high-throughput lossless data compression," in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, 2012, pp. 138–149.

[78] J. Jenkins, I. Arkatkar, S. Lakshminarasimhan, D. A. Boyuka II, E. R. Schendel, N. Shah, S. Ethier, C.-S. Chang, J. Chen, H. Kolla *et al.*, "Alacrity: Analytics-driven lossless data compression for rapid in-situ indexing, storing, and querying," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems X*. Springer, 2013, pp. 95–114.

[79] E. Soroush and M. Balazinska, "Time travel in a scientific array database," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 98–109.

[80] A. Gupta and R. K. Jha, "A survey of 5g network: Architecture and emerging technologies," *IEEE Access*, vol. 3, pp. 1206–1232, 2015.

[81] C. Shannon, "A mathematical theory of communication, bell system technical journal 27: 379-423 and 623–656," *Mathematical Reviews (MathSciNet): MR10, 133e*, 1948.

[82] "Gzip." [Online]. Available: http://gzip.org/

[83] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theor.*, vol. 23, no. 3, pp. 337–343, Sep. 2006.

[84] "7z." [Online]. Available: http://7-zip.org/

[85] "Snappy." [Online]. Available: http://google.github.io/snappy/

[86] "Zstd." [Online]. Available: https://github.com/facebook/zstd

[87] "Apache Spark." [Online]. Available: http://spark.apache.org/

[88] "Apache Hive." [Online]. Available: http://hadoop.apache.org/

[89] "Arx data anonymization tool." [Online]. Available: http://arx.deidentifier.org/

[90] M. A. Abbasoğlu, B. Gedik, and H. Ferhatosmanoğlu, "Aggregate profile clustering for telco analytics," *Proc. VLDB Endow.*, vol. 6, no. 12, pp. 1234–1237, Aug. 2013. [Online]. Available: http://dx.doi.org/10.14778/2536274.2536284

[91] Q. Ho, W. Lin, E. Shaham, S. Krishnaswamy, T. A. Dang, J. Wang, I. C. Zhongyan, and A. She-Nash, "A distributed graph algorithm for discovering unique behavioral groups from large-scale telco data," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, ser. CIKM. New York, NY, USA: ACM, 2016, pp. 1353–1362. [Online]. Available: http://doi.acm.org/10.1145/2983323.2983354

[92] H. Yan, S. Ding, and T. Suel, "Inverted index compression and query processing with optimized document ordering," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 401–410.

[93] M. Burtscher and P. Ratanaworabhan, "Fpc: A high-speed compressor for double-precision floating-point data," *IEEE Transactions on Computers*, vol. 58, no. 1, pp. 18–31, 2009.

[94] F. Douglis and A. Iyengar, "Application-specific delta-encoding via resemblance detection." in *USENIX Annual Technical Conference, General Track*, 2003, pp. 113–126.

[95] L. L. You, K. T. Pollack, D. D. Long, and K. Gopinath, "Presidio: a framework for efficient archival data storage," *ACM Transactions on Storage (TOS)*, vol. 7, no. 2, p. 6, 2011.

[96] S. Bhattacherjee, A. Chavan, S. Huang, A. Deshpande, and A. Parameswaran, "Principles of dataset versioning: Exploring the recreation/storage tradeoff," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1346–1357, 2015.

[97] M. L. Kersten and L. Sidirourgos, "A database system with amnesia." in *CIDR*, 2017.

[98] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Found. Trends databases*, vol. 4, no. 1–3, pp. 1–294, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1561/1900000004

[99] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik, "The new jersey data reduction report," *IEEE Data Eng. Bull.*, vol. 20, no. 4, pp. 3–45, 1997. [Online]. Available: http://sites.computer.org/debull/97DEC-CD.pdf

[100] K. Krishna, D. Jain, S. V. Mehta, and S. Choudhary, "An lstm based system for prediction of human activities with durations," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 147:1–147:31, Jan. 2018. [Online]. Available: http://doi.acm.org/10.1145/3161201

[101] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[102] Y. Bengio, P. Frasconi, and P. Simard, "The problem of learning long-term dependencies in recurrent networks," in *IEEE International Conference on Neural Networks*, 1993, pp. 1183–1188 vol.3.

[103] J. Laiho, A. Wacker, and T. Novosad, *Radio Network Planning and Optimisation for UMTS*. John Wiley & Sons, 2006.

[104] R. Dey and F. M. Salemt, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Aug 2017, pp. 1597–1600.

[105] S. Chaudhuri, G. Das, and V. Narasayya, "Optimized stratified sampling for approximate query processing," *ACM Trans. Database Syst.*, vol. 32, no. 2, Jun. 2007. [Online]. Available: http://doi.acm.org/10.1145/1242524.1242526

[106] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica, "G-ola: Generalized on-line aggregation for interactive analysis on big data," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 913–918. [Online]. Available: http://doi.acm.org/10.1145/2723372.2735381

[107] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica, "Blinkdb: Queries with bounded errors and bounded response times on very large data," in *Proceedings of the 8th ACM European Conference on Computer Systems*, ser. EuroSys '13. New York, NY, USA: ACM, 2013, pp. 29–42. [Online]. Available: http://doi.acm.org/10.1145/2465351.2465355

[108] L. Sidirourgos, Martin, and P. Boncz, "Sciborq: Scientific data management with bounds on runtime and quality," in *In Proc. of the Int'l Conf. on Innovative Data Systems Research (CIDR*, 2011, pp. 296–301.

[109] Z. Wei, G. Luo, K. Yi, X. Du, and J.-R. Wen, "Persistent data sketching," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: ACM, 2015, pp. 795–810. [Online]. Available: http://doi.acm.org/10.1145/2723372.2749443

[110] P. K. Agarwal, G. Cormode, Z. Huang, J. Phillips, Z. Wei, and K. Yi, "Mergeable summaries," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '12. New York, NY, USA: ACM, 2012, pp. 23–34. [Online]. Available: http://doi.acm.org/10.1145/2213556.2213562

[111] C.-J. Wang and W.-S. Ku, "Anonymous sensory data collection approach for mobile participatory sensing," in *Data Engineering Workshops (ICDEW), 2012 IEEE 28th International Conference on*, April 2012, pp. 220–227.

[112] Y.-B. Lin and J. Y.-B. Lin, *Wireless and Mobile Network Architectures*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 2000.

[113] B. Bollobás, *Modern graph theory*. Springer Science & Business Media, 1998.

[114] S. Trifunovic, B. Distl, D. Schatzmann, and F. Legendre, "Wifi-opp: Ad-hoc-less opportunistic networking," in *Proceedings of the 6th ACM Workshop on Challenged Networks*, ser. CHANTS '11. New York, NY, USA: ACM, 2011, pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/2030652.2030664

[115] G. Kollios, D. Gunopulos, and V. J. Tsotras, "Nearest neighbor queries in a mobile environment," in *Proceedings of the International Workshop on Spatio-Temporal Database Management*, ser. STDBM '99. London, UK, UK: Springer-Verlag, 1999, pp. 119–134. [Online]. Available: http://dl.acm.org/citation.cfm?id=646518.756858

[116] C. Xia, H. Lu, B. C. Ooi, and J. Hu, "Gorder: An efficient method for knn join processing," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, ser. VLDB '04. VLDB Endowment, 2004, pp. 756–767. [Online]. Available: http://dl.acm.org/citation.cfm?id=1316689.1316755

[117] V. Hautamaki, I. Karkkainen, and P. Franti, "Outlier detection using k-nearest neighbour graph," in *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 3 - Volume 03*, ser. ICPR '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 430–433. [Online]. Available: http://dx.doi.org/10.1109/ICPR.2004.671

[118] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1–37, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10115-007-0114-2

[119] N. Lathia, S. Hailes, and L. Capra, "knn cf: A temporal social network," in *Proceedings of the 2008 ACM Conference on Recommender Systems*, ser. RecSys '08. New York, NY, USA: ACM, 2008, pp. 227–234. [Online]. Available: http://doi.acm.org/10.1145/1454008.1454044

[120] W. Liu, J. Wang, and S. fu Chang, "Hashing with graphs," in *In ICML*, 2011.

[121] C. Aggarwal, "An introduction to social network data analytics," in *Social Network Data Analytics*, C. C. Aggarwal, Ed. Springer US, 2011, pp. 1–15. [Online]. Available: http://dx.doi.org/10.1007/978-1-4419-8462-3_1

[122] W.-C. Lee, D. Zeinalipour-Yazti, G. Chatzimilioudis, and M. D. Dikaiakos, "Continuous all k-nearest-neighbor querying in smartphone networks," *2012 13th IEEE International Conference on Mobile Data Management (MDM)*, vol. 00, pp. 79–88, 2012.

[123] P. Kolios, A. Pitsillides, and O. Mokryn, "Bilateral routing in emergency response networks," in *ICT 2013*, May 2013, pp. 1–5.

[124] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis, and G. Samaras, "Mint views: Materialized in-network top-k views in sensor networks," in *2007 International Conference on Mobile Data Management*, May 2007, pp. 182–189.

[125] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis, G. Samaras, and A. Pitsillides, "The micropulse framework for adaptive waking windows in sensor networks," in *2007 International Conference on Mobile Data Management*, May 2007, pp. 351–355.

[126] G. Chatzimilioudis, A. Cuzzocrea, and D. Gunopulos, "Optimizing query routing trees in wireless sensor networks," in *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*, vol. 2, Oct 2010, pp. 315–322.

[127] R. W. Floyd, "Algorithm 245: Treesort," *Commun. ACM*, vol. 7, no. 12, pp. 701–, Dec. 1964. [Online]. Available: http://doi.acm.org/10.1145/355588.365103

[128] T. Brinkhoff, "A framework for generating network-based moving objects," *Geoinformatica*, vol. 6, no. 2, pp. 153–180, Jun. 2002. [Online]. Available: http://dx.doi.org/10.1023/A:1015231126594

[129] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 247–256. [Online]. Available: http://doi.acm.org/10.1145/1367497.1367532

[130] X. Yu, K. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, April 2005, pp. 631–642.

[131] K. Mouratidis, D. Papadias, and M. Hadjieleftheriou, "Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '05. New York, NY, USA: ACM, 2005, pp. 634–645. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066230

[132] K. Clarkson, "Fast algorithms for the all nearest neighbors problem," in *Foundations of Computer Science, 1983., 24th Annual Symposium on*, Nov 1983, pp. 226–232.

[133] P. B. Callahan, "Optimal parallel all-nearest-neighbors using the well-separated pair decomposition," in *Proceedings of the 1993 IEEE 34th Annual Foundations of Computer Science*, ser. SFCS '93. Washington, DC, USA: IEEE Computer Society, 1993, pp. 332–340. [Online]. Available: http://dx.doi.org/10.1109/SFCS.1993.366854

[134] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, "Scaling and related techniques for geometry problems," in *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '84. New York, NY, USA: ACM, 1984, pp. 135–143. [Online]. Available: http://doi.acm.org/10.1145/800057.808675

[135] Y.-R. Wang, S.-J. Horng, and C.-H. Wu, "Efficient algorithms for the all nearest neighbor and closest pair problems on the linear array with a reconfigurable pipelined bus system," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 3, pp. 193–206, March 2005.

[136] T. Lai and M.-J. Sheng, "Constructing euclidean minimum spanning trees and all nearest neighbors on reconfigurable meshes," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 7, no. 8, pp. 806–817, Aug 1996.

[137] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-nearest-neighbors queries in spatial databases," in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 297–. [Online]. Available: https://doi.org/10.1109/SSDBM.2004.12

[138] Y. Chen and J. Patel, "Efficient evaluation of all-nearest-neighbor queries," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, April 2007, pp. 1056–1065.

[139] J. Sankaranarayanan, H. Samet, and A. Varshney, "A fast all nearest neighbor algorithm for applications involving large point-clouds," *Comput. Graph.*, vol. 31, no. 2, pp. 157–174, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.cag.2006.11.011

[140] P. M. Vaidya, "An o(n log n) algorithm for the all-nearest-neighbors problem," *Discrete Comput. Geom.*, vol. 4, no. 2, pp. 101–115, Jan. 1989. [Online]. Available: http://dx.doi.org/10.1007/BF02187718

[141] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-nearest-neighbors queries in spatial databases," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*, June 2004, pp. 297–306.

[142] R. Benetis, C. S. Jensen, G. Karciauskas, and S. Saltenis, "Nearest neighbor and reverse nearest neighbor queries for moving objects," in *Database Engineering and Applications Symposium, 2002. Proceedings. International*, 2002, pp. 44–53.

[143] G. S. Iwerks, H. Samet, and K. Smith, "Continuous k-nearest neighbor queries for continuously moving points with updates," in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB '03. VLDB Endowment, 2003, pp. 512–523. [Online]. Available: http://dl.acm.org/citation.cfm?id=1315451.1315496

[144] K. Raptopoulou, A. Papadopoulos, and Y. Manolopoulos, "Fast nearest-neighbor query processing in moving-object databases," *GeoInformatica*, vol. 7, no. 2, pp. 113–137, 2003. [Online]. Available: http://dx.doi.org/10.1023/A%3A1023403908170

[145] E. Frentzos, K. Gratsias, N. Pelekis, and Y. Theodoridis, "Algorithms for nearest neighbor search on moving object trajectories," *Geoinformatica*, vol. 11, no. 2, pp. 159–193, Jun. 2007. [Online]. Available: http://dx.doi.org/10.1007/s10707-006-0007-7

[146] Y.-K. Huang, S.-J. Liao, and C. Lee, "Efficient continuous k-nearest neighbor query processing over moving objects with uncertain speed and direction," in *Proceedings of the 20th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 549–557. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-69497-7_37

[147] X. Xiong, M. F. Mokbel, and W. G. Aref, "Sea-cnn: Scalable processing of continuous k-nearest neighbor queries in spatio-temporal databases," in *Proceedings of the 21st International Conference on Data Engineering*, ser. ICDE '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 643–654. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2005.128

[148] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD

'05. New York, NY, USA: ACM, 2005, pp. 479–490. [Online]. Available: http://doi.acm.org/10.1145/1066157.1066212

[149] A. Konstantinidis, D. Zeinalipour-Yazti, P. Andreou, G. Samaras, and P. K. Chrysanthis, "Intelligent search in social communities of smartphone users," *Distrib. Parallel Databases*, vol. 31, no. 2, pp. 115–149, Jun. 2013. [Online]. Available: http://dx.doi.org/10.1007/s10619-012-7108-0

[150] M. Muralikrishna and D. J. DeWitt, "Equi-depth multidimensional histograms," in *Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '88. New York, NY, USA: ACM, 1988, pp. 28–36. [Online]. Available: http://doi.acm.org/10.1145/50202.50205

[151] C. Zhang, F. Li, and J. Jestes, "Efficient parallel knn joins for large data in mapreduce," in *Proceedings of the 15th International Conference on Extending Database Technology*, ser. EDBT '12. New York, NY, USA: ACM, 2012, pp. 38–49. [Online]. Available: http://doi.acm.org/10.1145/2247596.2247602

[152] F. Dehne, A. Fabri, and A. Rau-Chaplin, "Scalable parallel geometric algorithms for coarse grained multicomputers," in *Proceedings of the Ninth Annual Symposium on Computational Geometry*, ser. SCG '93. New York, NY, USA: ACM, 1993, pp. 298–307. [Online]. Available: http://doi.acm.org/10.1145/160985.161154

[153] E. Plaku and L. E. Kavraki, "Distributed computation of the knn graph for large high-dimensional point sets," *J. Parallel Distrib. Comput.*, vol. 67, no. 3, pp. 346–359, Mar. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2006.10.004

[154] N. Nodarakis, S. Sioutas, D. Tsoumakos, G. Tzimas, and E. Pitoura, "Rapid aknn query processing for fast classification of multidimensional data in the cloud," *CoRR*, vol. abs/1402.7063, 2014. [Online]. Available: http://arxiv.org/abs/1402.7063

[155] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[156] H. Gao, J. Tang, X. Hu, and H. Liu, "Modeling temporal effects of human mobile behavior on location-based social networks," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, ser. CIKM '13. New York, NY, USA: ACM, 2013, pp. 1673–1678. [Online]. Available: http://doi.acm.org/10.1145/2505515.2505616

[157] D. Yang, D. Zhang, K. Frank, P. Robertson, E. Jennings, M. Roddy, and M. Lichtenstern, "Providing real-time assistance in disaster relief by leveraging crowdsourcing power," *Personal Ubiquitous Comput.*, vol. 18, no. 8, pp. 2025–2034, Dec. 2014. [Online]. Available: http://dx.doi.org/10.1007/s00779-014-0758-3

[158] A. Doan, R. Ramakrishnan, and A. Y. Halevy, "Crowdsourcing systems on the world-wide web," *Commun. ACM*, vol. 54, no. 4, pp. 86–96, Apr. 2011. [Online]. Available: http://doi.acm.org/10.1145/1924421.1924442

[159] A. Bozzon, M. Brambilla, and S. Ceri, "Answering search queries with crowdsearcher," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 1009–1018. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187971

[160] M. Renz, N. Mamoulis, T. Emrich, Y. Tang, R. Cheng, A. Zufle, and P. Zhang, "Voronoi-based nearest neighbor search for multi-dimensional uncertain databases," in *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ser. ICDE '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 158–169. [Online]. Available: http://dx.doi.org/10.1109/ICDE.2013.6544822

[161] N. Armenatzoglou, S. Papadopoulos, and D. Papadias, "A general framework for geo-social query processing," *Proc. VLDB Endow.*, vol. 6, no. 10, pp. 913–924, aug 2013. [Online]. Available: http://dx.doi.org/10.14778/2536206.2536218

[162] B. Yao, F. Li, and P. Kumar, "K nearest neighbor queries and knn-joins in large relational databases (almost) for free," in *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, March 2010, pp. 4–15.

[163] C. S. Jensen, A. Kligys, T. B. Pedersen, and I. Timko, "Multidimensional data modeling for location-based services," *The VLDB Journal*, vol. 13, no. 1, pp. 1–21, Jan. 2004. [Online]. Available: http://dx.doi.org/10.1007/s00778-003-0091-3

[164] C. Xu, X. Huang, N. Wu, P. Xu, and G. Yang, "Using memcached to promote read throughput in massive small-file storage system," in *2010 Ninth International Conference on Grid and Cloud Computing*, Nov 2010, pp. 24–29.