UNIL | Université de Lausanne

FACULTÉ DES HAUTES ÉTUDES COMMERCIALES

DÉPARTEMENT DE SYSTÈMES D'INFORMATION

THREE TEMPORAL PERSPECTIVES ON DECENTRALIZED LOCATION-AWARE COMPUTING: PAST, PRESENT, FUTURE

THÈSE DE DOCTORAT

présentée à la

Faculté des Hautes Études Commerciales
de l'Université de Lausanne

pour l'obtention du grade de
Docteur ès Sciences en systèmes d'information

par

Bertil CHAPUIS

Directeur de thèse
Prof. Benoît Garbinato

Jury

Prof. Olivier Cadot, Président
Prof. Yves Pigneur, expert interne
Prof. Philippe Cudré-Mauroux, expert externe
Prof. Patrick Thomas Eugster, expert externe

LAUSANNE
2018

UNIL | Université de Lausanne
HEC Lausanne
Le Décanat
Bâtiment Internef
CH-1015 Lausanne

IMPRIMATUR

Sans se prononcer sur les opinions de l'auteur, la Faculté des Hautes Etudes Commerciales de l'Université de Lausanne autorise l'impression de la thèse de Monsieur Bertil CHAPUIS, titulaire d'un bachelor en ingénierie des médias de la Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud et d'un master en systèmes d'information de l'Université de Lausanne, en vue de l'obtention du grade de docteur ès Sciences en systèmes d'information.

La thèse est intitulée :

THREE TEMPORAL PERSPECTIVES ON DECENTRALIZED LOCATION-AWARE COMPUTING: PAST, PRESENT, FUTURE

Lausanne, le 27 août 2018

Le doyen

Jean-Philippe Bonardi

Members of the thesis committee

Prof. Benoît Garbinato
Université de Lausanne
Thesis supervisor

Prof. Yves Pigneur
Université de Lausanne
Internal member of the thesis committee

Prof. Philippe Cudré-Mauroux
Université de Fribourg
External member of the thesis committee

Prof. Patrick Thomas Eugster
Université de la Suisse italienne
External member of the thesis committee

University of Lausanne
Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Bertil Chapuis

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my entire satisfaction.

Signature: _____ Date: August 30, 2018

Prof. Benoît Garbinato
Thesis supervisor

University of Lausanne
Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Bertil Chapuis

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my entire satisfaction.

Signature: Yves Pigneur Date: August 30, 2018

Prof. Yves Pigneur
Internal member of the doctoral committee

University of Lausanne

University of Lausanne
Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Bertil Chapuis

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my entire satisfaction.

Signature:  Date: August 30, 2018

Prof. Philippe Cudré-Mauroux
External member of the doctoral committee

University of Lausanne
Faculty of Business and Economics

Doctorate in Information Systems

I hereby certify that I have examined the doctoral thesis of

Bertil Chapuis

and have found it to meet the requirements for a doctoral thesis.
All revisions that I or committee members
made during the doctoral colloquium
have been addressed to my entire satisfaction.

Signature:  Date: August 30, 2018

Prof. Patrick Thomas Eugster
External member of the doctoral committee

Remerciements

Je tiens tout d'abord à remercier mon épouse, Jessica, pour son amour, son amitié et son soutien durant les douze dernières années. Fonder une famille et accueillir nos deux enfants, Margaux et Robin, restera toujours le plus beau de nos projets.

Je remercie tout particulièrement Monsieur Benoît Garbinato, Professeur à l'Université de Lausanne, de m'avoir accordé sa confiance durant les cinq dernières années. Sous sa direction, j'ai mieux compris ce qui faisait l'originalité d'une question de recherche et, grâce à ses qualités humaines, j'ai apprécié sereinement l'incertitude inhérente à l'innovation.

Je tiens à remercier Monsieur Thibault Estier, Maître d'enseignement et de recherche à l'Université de Lausanne, et Madame Christine Legner, Professeure à l'Université de Lausanne, pour la confiance témoignée lors de mon engagement en tant qu'assistant de recherche et d'enseignement au sein de l'Institut des Systèmes d'Information de l'Université de Lausanne.

J'exprime également ma gratitude à Monsieur Gabor Maksay, Enseignant à la Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud, et Monsieur Yves Pigneur, Professeur à l'Université de Lausanne, pour leur passion de l'enseignement et leur intérêt sincère pour les étudiants. Sans leur soutien et leurs démarches, je n'aurais probablement jamais rejoint le Master en Système d'Information de l'Université de Lausanne.

Je tiens enfin à remercier les membres externes de mon jury de thèse, Monsieur Philippe Cudré-Mauroux, Professeur à l'Université de Fribourg, et Monsieur Patrick Thomas Eugster, Professeur à l'Université de la Suisse italienne, ainsi que tous les chercheurs anonymes qui ont relu mes travaux. Leurs commentaires et leurs corrections m'ont permis de progresser et de grandement améliorer la qualité de mes travaux.

Je remercie finalement tous ceux avec qui j'ai eu l'occasion de collaborer durant ces années dont: Arielle Moro, Vaibhav Kulkarni, Kévin Huguenin, Periklis Andritsos, Lucas Mourot, Holly Cogliati-Bauereis, Vincent Bozzo, Rafal Kowalski, Boris Fritscher, Alexandre Métrailler, Jean-Sébastien Monzani, Nico Hillah, Mauro Cherubini, Mathias Humbert, Igor Bilogrevic et Alexandre Meylan. Pour ceux que j'aurais oubliés, faites-moi signe, je vous dois un café.

Résumé

Durant les quatre dernières décennies, la miniaturisation a permis la diffusion à large échelle des ordinateurs, les rendant omniprésents. Aujourd'hui, le nombre d'objets connectés à Internet ne cesse de croître et cette tendance n'a pas l'air de ralentir. Ces objets, qui peuvent être des téléphones mobiles, des véhicules ou des senseurs, génèrent de très grands volumes de données qui sont presque toujours associés à un contexte spatiotemporel. Le volume de ces données est souvent si grand que leur traitement requiert la création de système distribués qui impliquent la coopération de plusieurs ordinateurs. La capacité de traiter ces données revêt une importance sociétale. Par exemple: les données collectées lors de trajets en voiture permettent aujourd'hui d'éviter les embouteillages ou de partager son véhicule. Un autre exemple: dans un avenir proche, les données collectées à l'aide de gyroscopes capables de détecter les trous dans la chaussée permettront de mieux planifier les interventions de maintenance à effectuer sur le réseau routier. Les domaines d'applications sont par conséquent nombreux, de même que les problèmes qui y sont associés. Les articles qui composent cette thèse traitent de systèmes qui partagent deux caractéristiques clés: un contexte spatiotemporel et une architecture décentralisée. De plus, les systèmes décrits dans ces articles s'articulent autours de trois axes temporels: le présent, le passé, et le futur. Les systèmes axés sur le présent permettent à un très grand nombre d'objets connectés de communiquer en fonction d'un contexte spatial avec des temps de réponses proche du temps réel. Nos contributions dans ce domaine permettent à ce type de système décentralisé de s'adapter au volume de donnée à traiter en s'étendant sur du matériel bon marché. Les systèmes axés sur le passé ont pour but de faciliter l'accès a de très grands volumes données spatiotemporelles collectées par des objets connectés. En d'autres termes, il s'agit d'indexer des trajectoires et d'exploiter ces indexes. Nos contributions dans ce domaine permettent de traiter des jeux de trajectoires particulièrement denses, ce qui n'avait pas été fait auparavant. Enfin, les systèmes axés sur le futur utilisent les trajectoires passées pour prédire les trajectoires que des objets connectés suivront dans l'avenir. Nos contributions permettent de prédire les trajectoires suivies par des objets connectés avec une granularité jusque là inégalée. Bien qu'impliquant des domaines différents, ces contributions s'articulent autour de dénominateurs communs des systèmes sous-jacents, ouvrant la possibilité de pouvoir traiter ces problèmes avec plus de généricité dans un avenir proche.

Abstract

During the past four decades, due to miniaturization computing devices have become ubiquitous and pervasive. Today, the number of objects connected to the Internet is increasing at a rapid pace and this trend does not seem to be slowing down. These objects, which can be smartphones, vehicles, or any kind of sensors, generate large amounts of data that are almost always associated with a spatio-temporal context. The amount of this data is often so large that their processing requires the creation of a distributed system, which involves the cooperation of several computers. The ability to process these data is important for society. For example: the data collected during car journeys already makes it possible to avoid traffic jams or to know about the need to organize a carpool. Another example: in the near future, the maintenance interventions to be carried out on the road network will be planned with data collected using gyroscopes that detect potholes. The application domains are therefore numerous, as are the problems associated with them. The articles that make up this thesis deal with systems that share two key characteristics: a spatio-temporal context and a decentralized architecture. In addition, the systems described in these articles revolve around three temporal perspectives: the present, the past, and the future. Systems associated with the present perspective enable a very large number of connected objects to communicate in near real-time, according to a spatial context. Our contributions in this area enable this type of decentralized system to be scaled-out on commodity hardware, i.e., to adapt as the volume of data that arrives in the system increases. Systems associated with the past perspective, often referred to as trajectory indexes, are intended for the access to the large volume of spatio-temporal data collected by connected objects. Our contributions in this area makes it possible to handle particularly dense trajectory datasets, a problem that has not been addressed previously. Finally, systems associated with the future perspective rely on past trajectories to predict the trajectories that the connected objects will follow. Our contributions predict the trajectories followed by connected objects with a previously unmet granularity. Although involving different domains, these contributions are structured around the common denominators of the underlying systems, which opens the possibility of being able to deal with these problems more generically in the near future.

Contents

List of Figures

Chapter 1

Introduction

During the past two decades, computers became ubiquitous and pervasive due to miniaturization. People first became connected to the Internet with the adoption of broadband cellular networks. Then, connected objects followed a similar path with the emergence of low-cost wireless-sensor networks for the purpose of building the Internet of Things (IoT). As a result, the number of objects communicating on the Internet today is increasing at a fast pace and this trend does not seem to slow down. These objects, which can be smartphones, vehicles, or any kind of sensors, are almost always associated with environmental and situational information, including a *spatio-temporal context* [7]. This contextual information can be used to proactively enrich the experience associated with these devices. Therefore, the ability to process this data is of societal importance. For example, the data collected during car journeys already makes it possible to avoid traffic jams or to share one's car. In the near future, the maintenance interventions to be carried out on the road network will be planned with data collected using gyroscopes. The application domains are therefore numerous, as are the problems associated with them. The amount of these data is often so large that their processing requires the creation of *decentralized* system that involve the cooperation of several computers. Therefore, a new class of mobile systems began to emerge; it is now often referred to as *location-aware computing*, or more broadly as *context-aware computing*.

This thesis is structured as a collection of articles, all sharing two key characteristics: *location-awareness* and *decentralization*. To better understand our motivations, we first discuss the big-data challenges associated with location-aware computing in Section 1.1. We then take a system-architecture point of view and show some of the opportunities associated with decentralization in Section 1.2. In Section 1.3, we further detail some of the key benefits that a decentralized perspective on location-aware computing can bring. Finally, we describe our problem statement in Section 1.4, the organization of the collection of articles in Section 1.5, and our methodology in Section 1.6.

1.1 From Location Awareness to Big Data

Location-awareness broadly refers to systems and services that include a spatio-temporal context. As the number of connected devices, vehicles and objects increases, so does the amount of location data they generate. When dealing with large amount of data, *volume*, *velocity*, and *variety* are often referred to as the three V's of big data [54]. Furthermore, large companies, such as IBM, often introduce *veracity* as a fourth V[1]. Whereas these dimensions are sometimes associated with marketing jargon, we put them in perspective with location-aware computing to better understand the underlying challenges.

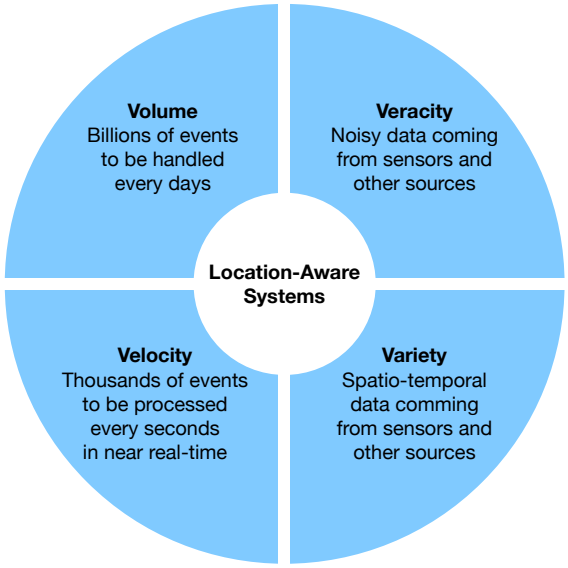


Figure 1.1: Location-Aware Systems are at the Intersection of the Three V's of Big Data

1.1.1 Volume

The *Volume* of data obviously relates to its size. In terms of size, the location data produced by connected devices and objects can be relatively large. For example, it is common for mobile network operators to collect several billions of location-based events per day. In this context, a wide range of value-added applications, such as traffic monitoring or urban planning, benefit from being able to process a very large amount of data. Such a large amount of data is typically greater than what a single networked computer can process, and making sense of a large volume of location-aware data over a long period of time is a real challenge. In this thesis, we devise scalable solutions, i.e., solutions that can adapt to a very large volume of data.

[1] https://www.ibmbigdatahub.com/infographic/four-vs-big-data

1.1.2 Velocity

The *Velocity* of data refers to the speed at which it is produced, ingested and transmitted. Location-aware data are usually associated with a very high velocity. Billions of events per days correspond to throughputs of thousands of events per seconds. If these events have to be continuously processed then such velocity is more than what the usual networked computer can sustain. More importantly, many use-cases require a very low end-to-end latency. In other words, the time needed for an event to propagate from one end of the system to the other should remain very short, even if some costly processing is involved. Therefore, location-aware systems come with many difficult challenges in terms of velocity. In addition to being able to scale, we reduce end-to-end latency by moving computation closer to the data.

1.1.3 Variety

The *Variety* of data refers to its diversity in terms of source, format, and structure. In this regard, sensor data and spatio-temporal data are not an exception as their variety is rich. For instance, a spatial context can be represented with a wide range of geometries that can be expressed in different spatial reference systems. As of today, there is no overall consensus on the protocols and formats used to transport such data. As a result, variety is clearly a critical issue associated with location-aware systems. In this thesis, we do not consider location-aware events as point in times, but as geometries, hence we address one challenging aspect associated with these dimensions.

1.1.4 Veracity

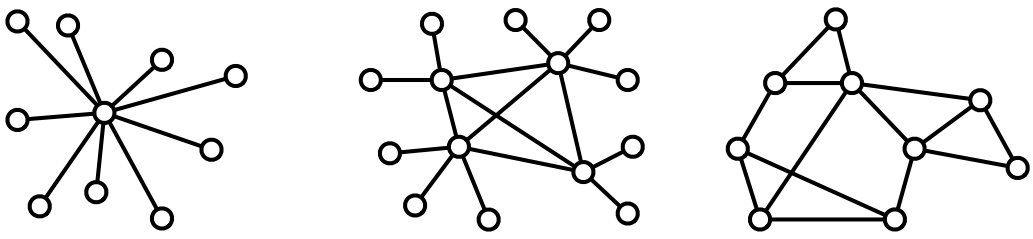
The *Veracity* of data deals with the overall quality of the data, which might contain abnormalities. In the context of this thesis, a poor GPS signal will obviously impact the quality of the data recorded by a sensor. Nevertheless, the quality of spatio-temporal data is rarely questioned and trajectory datasets are almost never associated with a ground truth that would allow to assess the effectiveness of an index. As a result, researcher often favor techniques that do not require such ground truth to operate, making it difficult to trade effectiveness for efficiency. In this thesis, we address this issue and introduce techniques to qualitatively assess trajectory indexes.

1.1.5 Research Challenges

It could be tempting to rely on traditional data-management systems to build location-aware systems. However, as highlighted in Figure 1.1, location-aware systems are at the crossroads of the three V's of big data. Traditional data-management systems usually deal well with one of these dimensions, but to the detriment of the others. For example, many relational database management systems, such as MySQL or PostgreSQL, integrate features that are very useful for dealing with any kind of spatial data (*Variety*). However, none of these systems are intended to deal with the billions of spatio-temporal events (*Volume*) that need to be ingested and indexed in near real-time (*Velocity*). As a result, many challenges arise when two or three of these dimensions overlap and our work addresses some of them.

1.2 From Fully Distributed to Decentralized Systems

In his seminal paper, Paul Baran depicted three kinds of communication networks: the *centralized*, the *decentralized*, and the *distributed* [6]. As highlighted in Figure 1.2a, the nodes of a centralized network are organized hierarchically as a star. Here, the central node is a single point of failure, i.e., its destruction affects every leaf node of the network and results in an impossibility to communicate. Ideally, as depicted in Figure 1.2c, the nodes of a distributed network should be organized as a mesh. Here, every node participates in routing and relaying the information and, as a result, the destruction of a single node does not affect the ability of the network to operate. However, as highlighted in Baran's paper and in Figure 1.2b, the nodes of a networks are often organized, for pragmatic reasons, in a *decentralized* fashion, i.e, as a mixture of star and mesh components. In such a network, the destruction of a central node only affects a small number of leaf nodes.



(a) Centralized (*Client-Server*) (b) Decentralized (*Cloud/Edge*) (c) Distributed (*P2P/MANET*)

Figure 1.2: Baran's Communication Networks (*and the Corresponding System Architectures*)

Today, the terms distributed and decentralized are often used interchangeably and Baran's classification might seem a bit outdated. Since 1964, networking technologies evolved a lot and physical networks are now hidden by one or more overlay networks that abstract the underlying infrastructure. However, the pragmatic observation regarding the organization of communication networks still holds. For instance, the bandwidth associated with connected devices is usually not uniformly distributed. Therefore, it is common to serve connected devices with centralized computers, typically located in a data center and associated with a large bandwidth. As the number of connected devices increases, several computers localized in a centralized datacenter might cooperate to provide a service, hence the (de)centralization. In this regard, this terminology allows to make subtle distinction between communication systems.

In this section, we use Baran's classification to categorize some of the research previously conducted by our research group. We show that this work either falls in the centralized category or in the distributed category. Although being better than its counterparts from a fault tolerance viewpoint, the distributed perspective is also associated with some drawbacks. Therefore, we highlight the opportunities that arise from adopting a pragmatic decentralized perspective, laying out the foundation of the present collection of articles. We do not include additional references in this introduction because each chapter of the thesis is self-contained and discusses the related work individually.

1.2.1 Centralized Systems

In Baran's view, the leaf nodes of a centralized network contact a central node to communicate with each other. In today's terminology, the clients of a centralized system contact a central server that is responsible for providing an application. That is the typical client-server model depicted in Figure 1.2a. It comes with many advantages. For example, from the point of view of a client, contacting a central entity is easier than discovering peers. Furthermore, from an application standpoint, maintaining a consistent state in a centralized setup is easier than in a distributed setup. However, centralized systems also come with limitations. For instance, a central server is a single point of failure that makes the system more vulnerable. In addition, the resources of a server are limited and might become a scalability bottleneck. Many popular relational database management systems, such as PostgreSQL and MySQL, were initially characterized by a centralized architecture. Before we began our research on decentralized location-aware systems, some of the work of our research group used a centralized perspective to devise and evaluate location-aware abstractions.

1.2.2 Distributed Systems

Ideally, in Baran's view, all the nodes of a distributed network should be organized as an interconnected mesh. As depicted in Figure 1.2c, in today's peer-to-peer (P2P) systems, the tasks associated with an application are partitioned between clients, also called peers. Such fully distributed systems are usually considered as being more fault tolerant than their centralized counterparts. However, from an application viewpoint, they also come with tradeoffs that are difficult to make in the context of a wide area network. For instance, the CAP Theorem states that given *consistency*, *availability*, and *partition tolerance*, any distributed system can guarantee at most two of those properties [60]. Many popular P2P applications, such as BitTorrent, Kademlia or Pastry, adopt a fully distributed architecture that typically relaxes consistency in favor of availability and partition tolerance [35, 99, 113]. The features provided by these applications however remain relatively simple (e.g., data dissemination); and the clients accept to live with weak forms of consistency, as long as they are eventually able to retrieve the data. A mobile ad hoc network (MANET) is another form of a P2P network that is said to be infrastructure-less, because the nodes are made of mobile devices that move and interconnect wirelessly. Each node knows its direct neighbourhood and the network is reconfigured continuously. As a result, the network is very unstable and network partitions are extremely frequent. Location-awareness in MANETs have been an active research topic in distributed systems, because it involves complex data-dissemination algorithms. However, as of today, MANETs have not been massively adopted by mobile application developers, hence not by end-users. This is probably due to the mismatch between the level of service required by end-users and the loose guarantees offered by MANETs. This mismatch is probably a direct consequence of the aforementioned CAP tradeoffs. MANETs typically do not function when devices equipped with different networking technologies are supposed to connect with each others. Before we began our research on decentralized location-aware systems, most of the work of our research group focused on the challenging algorithmic problems associated with the distributed perspective.

1.2.3 Decentralized Systems

In Baran's view, communication networks are often decentralized for pragmatic reasons. As highlighted in Figure 1.2c, in today's terminology, it is common for an application to rely on a set of servers that cooperate in a *decentralized* fashion. In such a setup, the servers are usually deployed in controlled environments, such as a data center or a cloud. In contrast to MANETs, the network of a data center is very stable and network partitions are relatively rare. Therefore, consistency is mainly traded to guarantee a very low latency [17, 3]. As a result, though the CAP tradeoffs still have to be made in a controlled environment, it is easier to provide a uniform level of service to

a majority of end-users. In this regard, popular NoSQL databases and large-scale data processing engines, such as Apache Cassandra, Apache Spark or Apache Kafka, can be seen as decentralized systems [88, 132, 83]. In these systems, the networked computers responsible for the application are deployed in a controlled environment, whereas the clients can be located in a wide-area network. It is worth noting that, the current *edge computing* trend pushes the bounderies of the controlled environment toward the end-users and, in this regard, can also be seen as a decentralized approach [75]. In other words, as the overall network infrastructure improves, some services can be moved at the edge, closer to the end-user. Before this thesis, none of the work which involved our research group approached location-aware systems from the decentralized perspective. As none of the work of our research group previously adopted this perspective, we explored the effects of the decentralized perspective on location-aware systems.

1.2.4 Research Opportunities

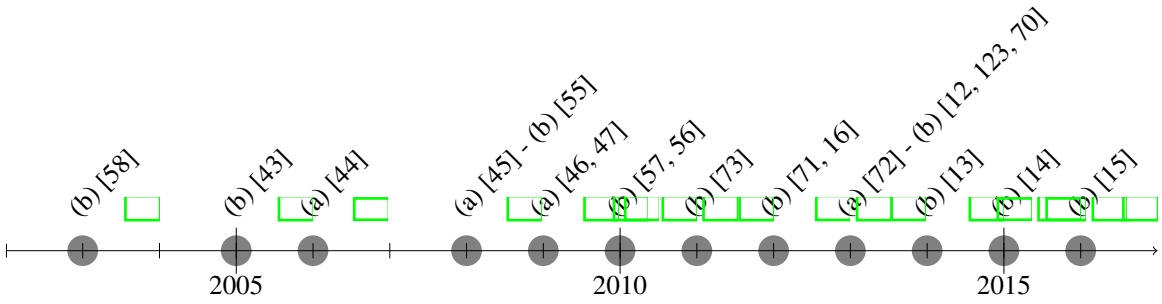


Figure 1.3: Previous Research: (a) Centralized Perspective and (b) Distributed Perspective

In order to better understand the research opportunities associated with the decentralized perspective, we recall some of the previous work people from our research group were involved in. Location-awareness was first approached from an ad-hoc-network viewpoint, with the definition of ad hoc applications and the definition of the location-based publish and subscribe abstraction [58, 43]. Then, the Pervaho middleware, which was initially aimed at being fully distributed, introduced centralized components for developing, testing, and evaluating mobile context-aware applications [44, 45, 46, 47]. From a distributed point of view, progress was made on broadcasting and neighbour-detection algorithms for MANETs [55, 57, 56, 73, 71, 16, 12, 123, 13, 14, 15, 70]. However, in the context of real-world location-aware applications, centralized approaches were still preferred for pragmatic reasons [72], i.e., MANET applications are very hard to deploy and maintain. As depicted in Figure 1.3, the previous work either falls in the centralized or in the distributed perspectives. As highlighted in Section 1.2.3, the centralized and the distributed perspectives both

have drawbacks that can be addressed with the decentralized perspective. Making a step backward from a distributed to a decentralized perspective opens new research avenues and enables pragmatic solutions to difficult problems that we explored in this thesis.

1.3 Benefits of Decentralization for Location-Aware Systems

We adopt a decentralized perspective to solve some of the issues associated with location-aware computing. None of the previous research conducted in our research group approached location-aware systems from this perspective. Therefore, we give here a bird's-eye view of some of the benefits that come with and the opportunities that arise from decentralization.

1.3.1 Horizontal Scalability

The capacity of a centralized system is bound by the maximal capacity of a single networked computer that can therefore constitute a bottleneck. Such a system is said to be *vertically scalable*, because as the demand for more computing resources grows, the capacity of the central computer needs to be adjusted, but this is only possible up to a certain point. In a distributed system, the capacity of the system should ideally be proportional to the number of computers involved. In this regard, such a system is said to be *horizontally scalable*, i.e., its capacity grows as more computers join the system. As its distributed counterpart, a decentralized system can be designed to *scale horizontally* or *scale-out*. The ability to *scale-out* or *scale-in* as more or less computing resources are needed is a significant opportunity for location-aware systems. Our contributions fullfill this horizontal scalability requirement.

1.3.2 Service-Level Agreement

A peer-to-peer system is considered as being more robust than a decentralized system, in the sense that a larger proportion of the networks is allowed to fail. However, as highlighted in Section 1.2.2, tradeoffs that directly affect the quality of service in terms of availability, consistency and partition tolerance have to be made. In a wide-area network, the computing-power, bandwidth and latency associated with networked computers varies greatly. Therefore, making such tradeoffs is particularly hard and establishing a service-level agreement, for example in terms of low latency, is nearly impossible. In addition, failures that occur outside of a controlled environment are very difficult to detect and to correct. In contrast, a controlled environment, such as a data center, alleviates these

issues. It is therefore possible to devise the system accordingly and to make stronger assumptions regarding the level of service provided to end-users. Our contributions give the ability to control and assess the level of service more precisely.

1.3.3 Uniform Data Access

In contrast to a distributed system deployed in a wide-area network, a decentralized system deployed in a controlled environment enables fast and uniform access to the data. For instance, even if reliably accessing all past location data seems difficult in a MANET, it is achievable in a controlled environment such as a data center. Being able to access the networked computers responsible for the data in a reliable and uniform way is crucial for performing data analysis. Therefore, the decentralized perspective enables new opportunities, such as the possibility to index and to analyze past location data in order to improve the end-user experience. Our contributions exploit such a uniform data access, thus enabling trajectory indexing and trajectory prediction.

1.3.4 Decentralized Middleware in the Cloud

A decentralized middleware in the cloud can be used to let objects, which rely on different networking technologies, communicate with each other according to a spatio-temporal context. Cell broadcast typically addresses a similar issue, but it is restricted to the scope of cellular networks. GeoCast also addresses the same problem, but there is not yet a consensus that would allow for the use of it at the level of the Internet [103]. Pragmatically, a decentralized middleware for location-aware computing deployed is the best alternative, provided that such a middleware scales horizontally in the cloud. As illustrated in Figure 1.4, at a physical level, connected objects often rely on different networking technologies to communicate. For instance, mobile users typically make use of a cellular network, whereas vehicles regularly connect to cheaper networking technologies, such as the LoRa network [117]. At a network level, gateways can be used to let devices equipped with different networking technologies communicate with each other, thus enabling the Internet of Things. Unfortunately, the network protocols available at this level do not enable connected objects that share a spatio-temporal context to communicate anonymously with each other. Therefore, a decentralized middleware can be deployed at the level of the cloud to provide this kind of service.

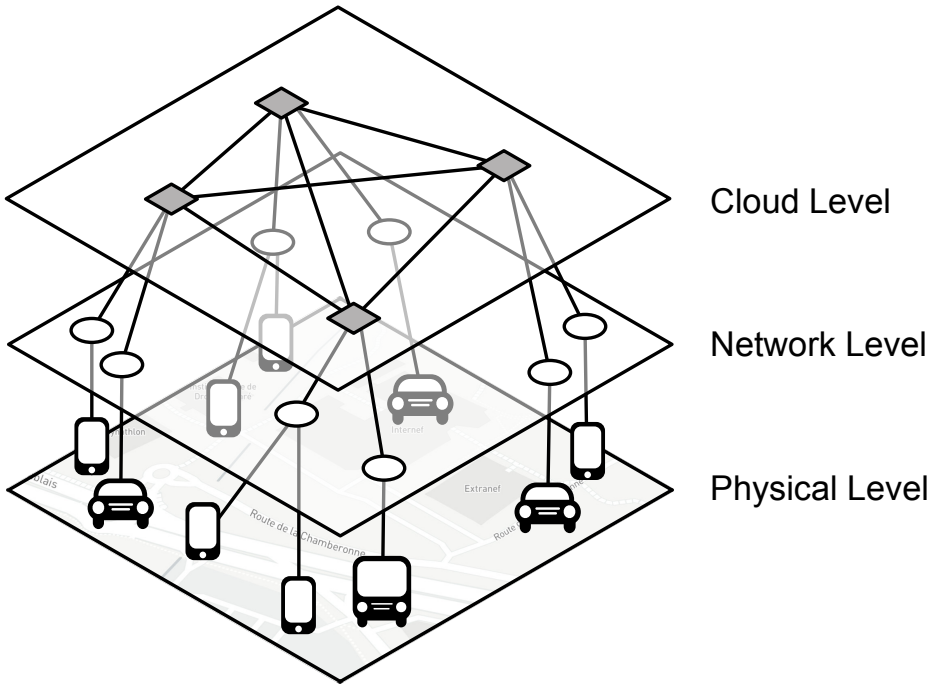


Figure 1.4: Decentralized Location-aware Middleware in the Cloud

1.4 Problem Statement

In this collection of articles, we address the following overall question: *What are the effects of a decentralized perspective on location-aware computing?* As this question is very broad, we first restrict its scope to a set of location-aware systems, which are discussed hereafter. Then, we further divide our main research question into sub-questions.

1.4.1 Scope of the Research

We focus on three main types of location-aware systems: *location-based publish and subscribe*, *trajectory indexing*, and *trajectory prediction*. To better understand the scope of our research, it is worth noting that each of these systems is associated with a spatio-temporal context. Furthermore, each of them take a different perspective on the temporal dimension, which is reflected in the title of the thesis.

Location-Based Publish and Subscribe (Present) Location-based publish and subscribe systems deals with near real-time data and are clearly anchored in the *present*. This communication paradigm enables connected moving objects to communicate with each other according to

a spatio-temporal context. When the spatio-temporal contexts of a publication and a subscription overlap, the publication is transmitted to the subscriber. Location-based publish and subscribe systems are said to be anonymous, because publishers and subscribers do not need to know each other, and asynchronous, because matching publications are pushed to the subscriber.

Trajectory Indexing (Past) Trajectory indexing systems deals with a massive volume of *past* data. The location histories of moving objects, called trajectories, can be used in various applications, such as car-sharing, traffic analysis, or urban planning. The utility of a large trajectory dataset is tightly coupled to the efficiency and the effectiveness of the access methods, referred to as trajectory indexing. In our context, we consider two use cases formulated with a query trajectory: the retrieval of similar trajectories, and the retrieval of similar sub-trajectories.

Trajectory Prediction (Future) Trajectory prediction systems aims at foresee *future* data by making sense of past data. Given the past trajectories and the current location of a moving object, the problem consists in predicting the most probable paths that the moving object will follow. Trajectory prediction systems can typically be used for various applications, such as location-based marketing. Also, good trajectory predictions can influence the way moving entities behave.

1.4.2 Research Questions

Keeping the scope of our research in mind, we now formulate the following research questions.

Q1 *How can decentralized location-based publish and subscribe systems be tested?*
One of the main issues associated with location-based publish and subscribe systems is the lack of testbeds and realistic test data. Some trajectory generation systems, such as Berlin-MOD, have been created to generate synthetic trajectories. However, this kind of test data typically target batch-processing use cases and do not provide the tooling required to generate synthetic data in near real-time. Therefore, our goal is to devise and implement such a testbed for location-based publish and subscribe systems.

Q2 *How can a decentralized location-based publish and subscribe system scale horizontally?*
Decentralized key-value stores, such as Apache Cassandra, use partitioning strategies, such as consistent-hashing, to evenly spread a key space across a set of networked computers, thus enabling horizontal scalability. Unfortunately, in the context of location-based publish and subscribe systems, a spatio-temporal context can span across several dimensions. As a result,

in contrast to a key that matches with one partition, a spatio-temporal context can overlap with several partitions. It is therefore not evident if such partitioning strategy can be used to partition a location-based publish and subscribe systems. Our aim is therefore to investigate and address the potential issues associated to this question.

Q3 *How can a decentralized location-based publish and subscribe system be made reliable?*
In a decentralized key-value store, reliability is usually attained by replicating the data on several networked computers called replicas. The location-based publish and subscribe paradigm is slightly more complicated than a key-value store. When a match occurs between a publication and a subscription, a notification has to be triggered and pushed to the end user. As several failures might occur along the way from one end of the system to the other, our goal is to address reliability with a replication strategy. In addition, we need to understand the impact of such a replication mechanism on the overall scalability of the system.

Q4 *What are the best access methods for large volumes of trajectories?*
Many challenges are associated with very large sets of trajectories. First, in terms of queries, we could search for similar trajectories, but also for trajectories that share similar sub-trajectories. Second, a large trajectory dataset could contain many relevant results for a given query because of its density. As a similarity measure is used to sort relevant results, the performance of the similarity measure can become a bottleneck when a trajectory dataset is very dense. Third, when performing queries, the traditional methods for trajectory indexing assume exact answers. Hence, the obvious tradeoff is to trade exactness for shorter execution time. Here, we devise new algorithms for trajectory indexing that address these challenges.

Q5 *How can we qualitatively assess an index of trajectories?*
As mentioned, given a query, traditional methods for trajectory indexing assume *exact* answers. In such a context, performance is the sole measure of interest. As our aim is to trade exactness for better performance, we need to understand and assess what makes the quality of a result. Therefore, we devise a testbed for trajectory indexing that can be used to prove that a method is good, both in terms of performance and in terms of quality.

Q6 *How can a very large index of trajectories be decentralized?*
In terms of volume, a trajectory index can be too large to fit on a single networked computer. A partitioning strategy is therefore necessary to decentralize the index. The existing data structures for trajectory indexing are relatively complex, difficult to reason about, and sometimes require being reorganized after an insertion. Therefore, we include the scalability issue into our reasoning when devising algorithms for trajectory indexing.

Q7 *How can we predict future trajectories on the basis of past trajectories?*
Mobility prediction has been an active field of research during recent years. Interestingly, as of today, most mobility prediction techniques either focus on a short horizon, with the estimation of motion functions, or on a much longer horizon, with the prediction of the next points of interest. We fill this gap with the prediction of the trajectories that a user will follow to visit its next point of interest. It is worth noting that mobility prediction is of great interest for many applications such as targeted advertisement.

Q8 *How can we qualitatively assess the accuracy of a prediction system for future trajectories?*
When predicting the next point of interest of a user, the prediction is either correct or wrong. Therefore, the quality of such a prediction system can simply be assessed with a binary classifier. In the context of a trajectory, the problem is slightly different because the trajectory of a user between two point of interests can differ depending on external factors, such as the conditions of the road network. A prediction can therefore be partially correct or partially wrong. Given a training dataset and a validation dataset, we devise a more nuanced model for assessing the quality of the trajectories foreseen by a prediction system.

Q9 *How can a prediction system for future trajectories be decentralized?*
Predicting future trajectories requires many computational resources. The past data of a user have to be regularly processed in order to build a prediction model. Furthermore, new predictions have to be performed every time the user moves from one point of interest to the other. Therefore, our goal is to understand how such prediction systems could be decentralized and scaled.

1.5 Organization and Structure

This thesis is composed of five articles that were published in conference and workshop proceedings in computer science. These articles highlight some of the results we obtained during five years of research at the University of Lausanne. We structure the collection of articles according to the three temporal perspectives previously identified: *Present*, *Past*, and *Future*. The articles are self contained, i.e., each of them introduces its own terminology and can be read independently from the others. As a consequence, the terminology might differ slightly from one chapter to the other and the content of the chapters sometimes overlap. Hereafter, we highlight some of our contributions.

1.5.1 Part I: Location-Based Publish and Subscribe (Present)

The load-testing tool presented in Chapter 2 is our answer to question **Q1** *How can a decentralized location-based publish and subscribe system be tested?* In this demonstration paper, we present and implement a testbed that generates trajectory data in real time and that can be plugged into an existing location-based publish and subscribe system. The trajectories correspond either to randomly moving entities or to entities moving on an existing road network. In addition, we present an preliminary version of a decentralized architecture for location-based publish and subscribe. We also introduce monitoring tools that enable us to evaluate our architecture in terms of load, memory consumption, and throughput.

In Chapter 3, we present a more mature version of our decentralized location-based publish and subscribe system. We show that a min-wise hashing agreement can be used to keep the overhead associated with partitioning very low, hence answering question **Q2** *How can a decentralized location-based publish and subscribe system scale horizontally?* We evaluate our system and demonstrate that it scales horizontally in a data center made of up to 200 virtual machines. We discuss several aspects associated with the reliability of the system in order to answer the question **Q3** *How can a decentralized location-based publish and subscribe system be reliable?* We show that a routing mechanism that involves several steps in the decentralized system can be subject to non-independence of failure. We address this issue by grouping the networked computer of the decentralized system in a way that guarantees independence of failure. We also show that our reliability mechanism introduces an overhead that remains proportional to the replication factor.

1.5.2 Part II: Trajectory Indexing (Past)

In Chapter 4, we explore the idea of using data deduplication to index data in a type-agnostic manner, i.e., regardless of its type. We show that, even if data deduplication is originally intended at identifying exact duplicates in binary data, normalization can be used in conjunction with deduplication to detect similarities in any kind of sequential data. We demonstrate the feasibility of our approach with textual data and introduce the idea of using it to index trajectory data. We highlight some promising results, giving a preliminary answer to the question **Q4** *What are the best access methods for large volumes of trajectories?*

In Chapter 5, we focus solely on trajectory data. We replace data deduplication with a more specialized method inspired by data winnowing, which is proven to be more efficient at detecting similarities. We also exemplify the density issue associated with large volumes of trajectory data

and demonstrate that our method efficiently addresses this problem. As previously mentioned, our approach trades accuracy for performance. Therefore, we characterize this tradeoff more precisely to answer question **Q5** *How can we qualitatively assess an index of trajectories?* Using the synthetic trajectories generated with the testbed introduced in Chapter 2, which is necessary to establish a ground truth, we put forth a method to qualitatively evaluate a trajectory index. Furthermore, we show that our method comes with interesting properties which can be used to partition the index, hence addressing question **Q6** *How can a very large index of trajectories data be decentralized?* We show with a large worldwide dataset that a trajectory index can be evenly balanced across several networked computers.

1.5.3 Part III: Trajectory Prediction (Future)

In Chapter 6, we introduce the problem of predicting trajectories. As for mobility prediction, we highlight the literature gap that exists between the short horizon and the distant horizon. We show that a combination of techniques, which include the prediction of the next point of interest of a user and the discretization of its trajectories, can be used to infer the trajectory that a user will follow, hence answering question **Q7** *How can we predict future trajectories on the basis of past trajectories?* As previously stated, the quality of a trajectory prediction system cannot be assessed with a binary classifier. Therefore, we put forth a model for assessing such a prediction system in order to answer question **Q8** *How can we qualitatively assess the accuracy of a prediction system for future trajectories?* Given that mobility prediction is associated with the scope of a user, we give some preliminary insights on how question **Q9** *How can a prediction system for future trajectories be decentralized?* could be addressed.

1.6 Research Methodology

Although we did not formally and systematically apply a design science research methodology, our research process can be put in perspective with the three cycles of design science research, illustrated in Figure 1.5 and introduced by Alan Hevner [69]. For instance, the overall reflexion about taking a step backward from distributed to decentralized systems has been heavily influenced by workshops in design thinking and design science. Therefore, from the *relevance cycle* perspective, this reflexion can be seen as an intent to align the research in location-aware computing with the needs of end users in a better way [106]; this alignment resulted in the identification of new research problems and opportunities. From the *rigor cycle* perspective, we continuously explored the existing knowledge in our field of research to lay out a foundation for our contributions and to

validate their novelty. For example, though trajectory indexing is well covered in the literature, the density problem addressed by our research had not been identified previously, because of the lack of trajectory datasets showcasing this issue. Finally, from the *design cycle* perspective, we iteratively built and evaluated our artifacts, continuously integrating and producing results from and for the relevance and rigor cycles.

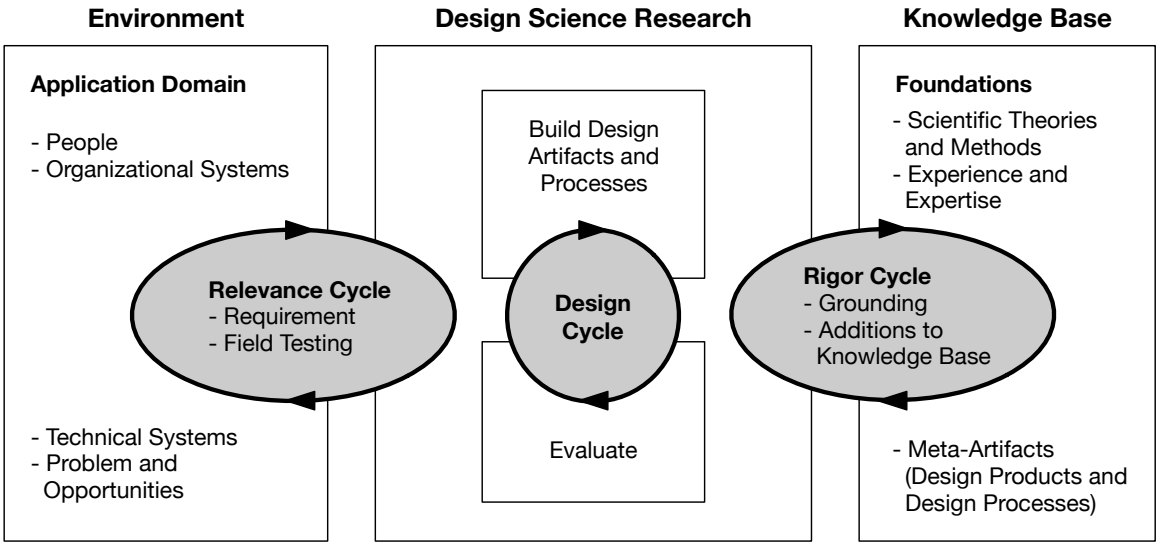


Figure 1.5: Hevner's Three Cycle View of Design Science Research [69]

As our contributions have been published mainly in computer science, we used formal and empirical methods to build and evaluate our artifacts. We usually began our articles by identifying a problem and by defining a model (often formulated in mathematical terms) for reasoning about that problem. On this basis, we set evaluation criterions and we formulated original algorithmic solutions that specifically address the problem. From a formal perspective, these solutions are sometimes evaluated theoretically, i.e., in terms of their computational complexity. From an empirical perspective, these solutions are implemented and compared with existing solutions on real or synthetic datasets by using the evaluation criterions. This process is iterative and continuous in the sense that the implementation and evaluation of an algorithm, which might come from the rigor cycle, often enable us to better understand the problem and to successively refine the model and the solution accordingly.

1.7 Complete List of Publications

The present collection of articles does not include all the work that was done during this research. For example, decentralization comes at a cost: storing all the data in a controlled environment, such as a data center, can greatly affect the privacy of end-users. According to the general data protection regulation enforced in Europe, location data is considered as being highly sensitive. Therefore, during our research, we addressed some of the issues associated with the privacy of location-based services. The following list of publication integrates the articles of the present collection and the articles published to address such side issues.

1. Bertil Chapuis and Benoît Garbinato. Geodabs: Trajectory indexing meets fingerprinting at scale. In *38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018

2. Vaibhav Kulkarni, Arielle Moro, Bertil Chapuis, and Benoît Garbinato. Capstone: Mobility modeling on smartphones to achieve privacy by design. In *International Conference on Trust, Security And Privacy In Computing And Communications (TrustCom)*. IEEE, 2018

3. Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. Towards usable checksums: Automating the integrity verification of web downloads for the masses. In *25th ACM Conference on Computer and Communications Security*. ACM, 2018

4. Bertil Chapuis and Benoît Garbinato. Scaling and load testing location-based publish and subscribe. In *37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2543–2546. IEEE, 2017

5. Vaibhav Kulkarni, Arielle Moro, Bertil Chapuis, and Benoît Garbinato. Extracting hotspots without a-priori by enabling signal processing over geospatial data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 79. ACM, 2017

6. Bertil Chapuis, Benoît Garbinato, and Lucas Mourot. A horizontally scalable and reliable architecture for location-based publish-subscribe. In *36th Symposium on Reliable Distributed Systems (SRDS)*, pages 74–83. IEEE, 2017

7. Vaibhav Kulkarni, Bertil Chapuis, and Benoît Garbinato. Privacy-preserving location-based services by using intel sgx. In *Proceedings of the First International Workshop on Human-centered Sensing, Networking, and Systems*, pages 13–18. ACM, 2017

8. Bertil Chapuis, Benoît Garbinato, and Periklis Andritsos. An efficient type-agnostic approach for finding sub-sequences in data. In *19th International Conference on High Performance Computing and Communications; 15th International Conference on Smart City; 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 270–277. IEEE, 2017

9. Bertil Chapuis, Arielle Moro, Vaibhav Kulkarni, and Benoît Garbinato. Capturing complex behaviour for predicting distant future trajectories. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, pages 64–73. ACM, 2016

10. Bertil Chapuis, Benoît Garbinato, and Periklis Andritsos. Throughput: A key performance measure of content-defined chunking algorithms. In *36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 7–12. IEEE, 2016

11. Bertil Chapuis and Benoît Garbinato. Knowledgeable chunking. In *International Conference on Networked Systems*, pages 456–460. Springer, Cham, 2015

Part I

Present: Location-based publish and subscribe

Chapter 2

Scaling and Load-testing Location-based publish and subscribe

Bertil Chapuis and Benoît Garbinato. Scaling and load testing location-based publish and subscribe. In *37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2543–2546. IEEE, 2017

Abstract

The rise of the Internet of things (IoT) poses massive scalability issues for location-based services. More particularly, location-aware publish and subscribe services are struggling to scale out the computation of matches between publications and subscriptions that continuously update their location. In this demonstration paper, we propose a novel distributed and horizontally scalable architecture for location-aware publish and subscribe. Our middleware architecture relies on a multi-step routing mechanism based on consistent hashing and range partitioning. To demonstrate its scalability, we present a traffic data generator, which, in contrast to existing generators, can be used to perform real-time load tests. Finally, we show that our architecture can be deployed on a small 10-node cluster and can process up to 80,000 location updates per second producing 25,000 matches per seconds.

2.1 Introduction

Today, connected mobility is no longer reserved to human beings. More and more moving objects are connected to the Internet and, with initiatives such as the LoRa Alliance[1], even insignificant objects may soon become talkative. As highlighted by Gartner[2] and IDC[3], this trend is not likely to stop, and there may be between 25 to 30 billion connected objects by 2020.

In this context, when it comes to developing context-aware applications that want to take advantage of the Internet of things (IoT) ecosystem, the location-based publish and subscribe paradigm is of particular interest. With this communication paradigm, connected objects can issue publications and subscriptions that are geographically scoped and that move with them. A match occurs between a given publication and a given subscription if a *context* criterion and a *content* criterion are both met simultaneously. On one hand, a proximity condition between a publication and a subscription can be expressed by a context criterion. On the other hand, a semantical relationship between a publication and a subscription can be embodied by a content criterion.

Although the location-based publish and subscribe paradigm offers a lot of flexibility and expressiveness, its implementation poses difficulties in terms of horizontal scalability. As of today, the systems described in the literature have addressed this scalability issue by proposing *vertically scalable* solutions (i.e., with one computing unit being responsible for the full workload) [32, 36, 43, 71, 93, 125]. Obviously, the exponential growth of the IoT ecosystem can easily exceed the load that is sustainable for a vertically scalable computing unit. In this demonstration, we showcase a horizontally scalable middleware architecture for the location-based publish and subscribe communication paradigm, which can be deployed on clusters made of commodity hardware. In addition, we present near real-time load testing tools that can be used to load test and benchmark such architecture by mocking traffic data in real time.

2.2 Middleware Architecture

In this section, we describe a horizontally scalable distributed middleware architecture supporting the location-based publish and subscribe communication paradigm. This architecture addresses the problem of distributing the computation of matches among cluster nodes in a way that allows the overall system to *scale out* or *scale horizontally*. In contrast to centralized spatial data structures,

[1] https://www.lora-alliance.org
[2] http://www.gartner.com/newsroom/id/3165317
[3] https://www.idc.com/getdoc.jsp?containerId=US40755816

which aim at taking advantage of geographical proximity, our solution uses *consistent hashing* in conjunction with range partitioning expressed with the notion of *tiles* in order to distribute the computation of matches across a cluster.

2.2.1 Grid and Tiles

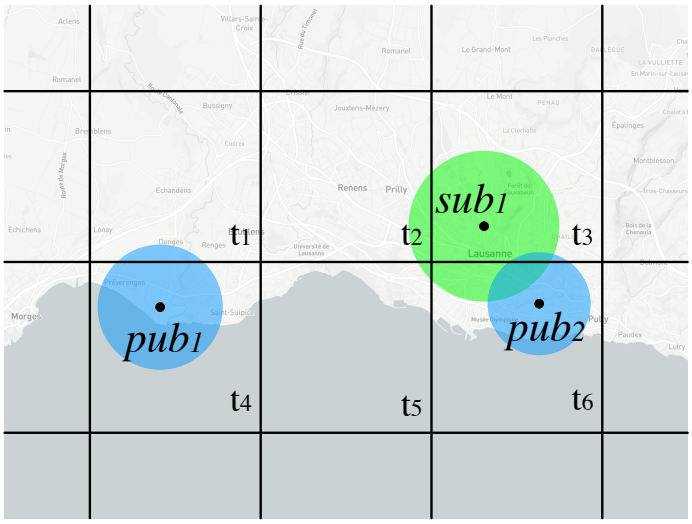


Figure 2.1: Publications, Subscriptions, and Tiles

The notion of grid layout, that divides the world into sets of tiles, is typically used by map services such as Google Maps[4] or Mapbox[5] to serve static data. As illustrated in Figure 2.1, we use a similar approach to distribute the computation of matches between publications and subscriptions among the nodes of a cluster. However, our use case is more complex since publications and subscriptions continuously move from tile to tile and matches must be emitted in real time in reaction to these updates.

2.2.2 Consistent Hashing

In general, in order to scale horizontally, distributed hash tables partition data items across a cluster of nodes with a family of hash functions called *consistent hashing* [78, 79]. It is common to think about the range of hash values produced by consistent hashing as a ring. In such a ring, the largest possible hash value convolutes to the smallest possible hash value [38]. Each node of the cluster

[4] https://maps.google.com
[5] https://www.mapbox.com

is placed on the ring at a fixed position which can be obtained by hashing the unique identifier of that node. To locate the node responsible for storing a given data item, the identifier of that item is first hashed and then the first node with a placement value greater than the resulting hash value is selected.

2.2.3 Message Routing

Our architecture relies on range partitions obtained by tiling of the earth's surface and on a set of consistent hashing functions to partition and distribute the load of computing matches between publications and subscriptions. In other words, consistent hashing functions are responsible for evenly distributing the load on cluster nodes, whereas the subdivision of the earth's surface into tiles is used as a partitioning criteria. Figure 2.2 shows how these notions of tiling and consistent hashing are assembled together to form our middleware architecture.

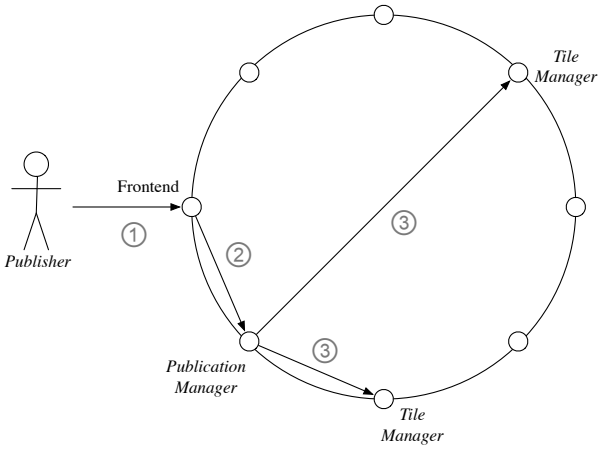
Publication Routing

The routing of publications from a moving entity to a tile is illustrated in Figure 2.2a:

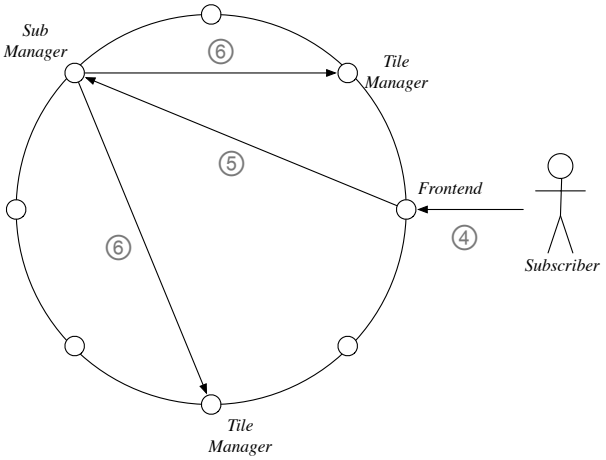
1. When adding a publication to the middleware or updating its location, the moving entity first contacts a frontend service. The frontend service typically runs on every node of the cluster and is placed behind a load balancer, so the moving entity does not know which service is contacted.
2. The frontend service then routes the request to a publication manager service using consistent hashing on the identifier of the publication. This intermediary step is required to manage the changing state of a publication transparently.
3. Finally, publications are routed to a tile manager service using consistent hashing on tile identifiers. Since a publication state changes and can move from tile to tile, some messages are generated to add publications to tiles, while others are generated to remove publications from tiles.

Subscription Routing

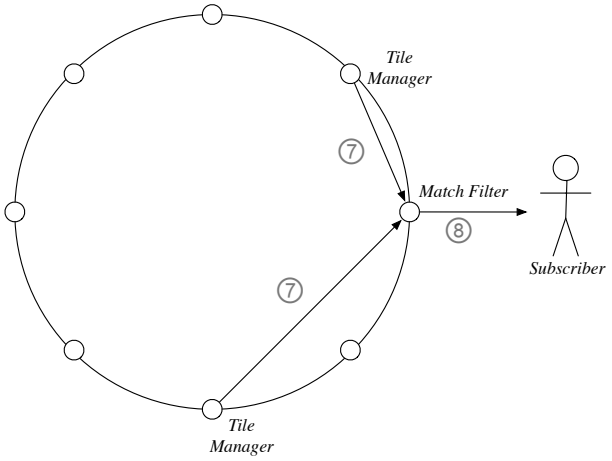
As illustrated in Figure 2.2b, the routing of subscriptions is symmetrical to the routing of publications and achieves the same purpose:



(a) Publication routing



(b) Subscription routing



(c) Match routing

Figure 2.2: Distributed routing based on consistent hashing

4. The addition or update of a subscription first goes through a frontend service.

5. It is then routed to the subscription manager node responsible for it, using consistent hashing on the subscription identifier.

6. Finally, the subscription is routed to the correct tile manager service using consistent hashing on tile identifiers. As a subscription moves, the subscription manager is responsible for generating the correct tile registration and deregistration messages.

Match Routing

The tile manager service is responsible for triggering matches when a publication and subscription overlap in a tile. Figure 2.2c illustrates the routing of a matching publication back to the subscription:

7. Matches between publications and subscriptions are first routed to a match filter service using consistent hashing on the subscription identifier.

8. The match filter may look superfluous but is required since the same match may be computed on several tiles on different nodes. Thus, a filtering mechanism is required and the match filter service is responsible for eliminating duplicates and transmitting matches to subscribers only once.

2.3 Traffic Data and Load Testing

Spatio-temporal and traffic data generators are regularly used to benchmark moving object databases. In this section, we highlight some prior batch data generators and show the need for a new kind of generator that produces data in real time with the intent to load test location-based services.

2.3.1 Batch Generation

The Brinkoff data generator [19] uses a real road network as well as a perturbation model to generate mobility traces. The BerlinMod Traffic Generator [40] relies on the Berlin road network and on the Secondo DBMS to generate data. The Minnesota Traffic Generator [101] provides a web interface to the Brinkoff and BerlinMod traffic data generators. Finally, the Hermoupolis generator [108] uses meaningful semantic data, such as homes and workplaces, to make the generated data more

realistic and similar to real human mobility traces. These data generators were devised for use cases characterized by batch processing requirements. They first generate a large dataset, which is then used to benchmark a moving object database. Our use case requires large volumes of mobility data to load test the scalability of our middleware. In fact, this data is too large to be pre-generated and must be produced in near real-time. Consequently, we devised a data generator that mocks the behavior of a large fleet of moving entities in near real-time.

2.3.2 Real-time Generation

Our real-time data generator produces synthetic mobility traces, which are distinct from each other and stick to an existing road network. To do so, the underlying generation model assumes that each moving entity performs a round trip that passes by several locations randomly picked in a given range on a map. When the round trip of a moving entity ends, the same round trip is simply started again. To infer the location a moving entity at any given time during a round trip, we rely on an open-source routing library called GraphHopper[6]. GraphHopper uses a variant of the Dijkstra algorithm in conjunction with a road network extracted from the OpenStreetMap[7] dataset to calculate routes and durations between locations. In our case, we use the duration of a route to infer the average speed of a moving entity between two locations of a round trip. In addition, the segments composing the route are used to derive the position of the moving entity at any given time. Finally, since the accuracy of GPS trackers varies and mobility traces never match perfectly to an existing road network, we add some Gaussian noise to the generated traces.

2.3.3 Load Testing

To load test our middleware, we must be able to simulate many concurrent moving entities. The Scala[8] programming language and the actor model implemented in the Akka[9] suit this requirement perfectly. In terms of implementation, each moving entity corresponds to an actor whose state contains a roundtrip and a position. The generator can be configured with several parameters, such as the number of moving entities, the road network used to produce the data, or the average rate at which moving entities are reporting their locations. Locations are reported by sending publication updates and subscription updates directly to the middleware using a protocol we built using GRPC[10].

[6] https://www.graphhopper.com/
[7] https://www.openstreetmap.org
[8] http://www.scala-lang.org
[9] http://www.akka.io
[10] http://www.grpc.io

In future releases, we might include additional drivers that could be used to test other systems characterized by the same requirements.

2.4 Demonstration

In this section, we highlight some key characteristics of the interactions proposed to the attendants of this demonstration. Through these interactions, our main goal is to show the scalability of the middleware architecture we described earlier.

2.4.1 Cluster Configuration

The attendant interacts with a small cluster made of 10 virtual machines responsible for computing matches between publications and subscriptions. In this cluster, each virtual machine is located on a different host and equipped with two vCPU and 4 GB of RAM. The middleware is deployed using Kubernetes[11] and Docker[12]. Different traffic data generation scenarios are launched with the same tools in the same cluster to demonstrate the scalability of the middleware. For example, with this cluster configuration, our middleware can process up to 80,000 location updates per second for 100,000 moving publications and 100,000 moving subscriptions, producing up to 25,000 matches per second. Interestingly, despite the number of network hops introduced by the architecture, the average end-to-end latency remains below 50 milliseconds.

[11] `https://kubernetes.io`
[12] `https://www.docker.com`

2.4.2 Cluster Monitoring

Figure 2.3: Cluster Monitoring

As illustrated in Figure 2.3, the middleware is monitored with StatsD[13], Graphite[14], and Graphana[15]. The attendant interacts with a live dashboard and can obtain real-time insight on the status of the middleware. The displayed metrics include the throughput at which publications and subscriptions are updating their locations, the throughput in terms of matches between publications and subscription and the status of each individual node participating in the computation of matches in terms of load average and memory usage.

[13] https://github.com/etsy/statsd
[14] https://graphiteapp.org
[15] http://grafana.org

2.4.3 End-User Interactions

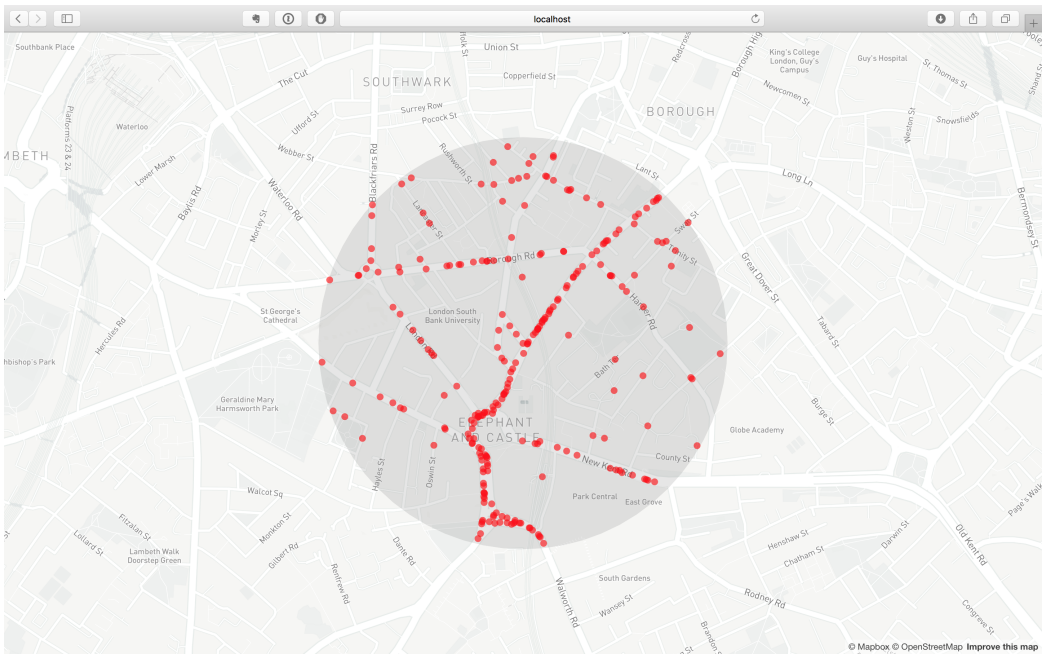


Figure 2.4: Moving Subscription and Synthetic Publications

As illustrated in Figure 2.4, an attendant can interact with the system with a web browser. Here, the large gray circle corresponds to a subscription that moves as the user navigates the map. The small red circles correspond to the set of moving publications located in the range of the subscription. The publication updates that match the subscription are transmitted to the browser in real time due to the websocket protocol. In this demonstration, the publications update their location every second so the map is animated.

2.5 Conclusion & Future Work

The middleware for location-based publish and subscribe presented in this demonstration paper achieves horizontal scalability due to a multi-step routing mechanism that relies on consistent hashing and range partitioning. The new traffic data generator we propose can be used to generate large amounts of real-time data to perform load and scalability tests on location-based services. As of today, it is not clear how such a middleware can recover from failure or scale in and out on demand. Consequently, our future work will focus on investigating these issues.

Chapter 3

A Horizontally Scalable and Reliable Architecture for Location-Based Publish-Subscribe

Bertil Chapuis, Benoît Garbinato, and Lucas Mourot. A horizontally scalable and reliable architecture for location-based publish-subscribe. In *36th Symposium on Reliable Distributed Systems (SRDS)*, pages 74–83. IEEE, 2017

Abstract

With billions of connected users and objects, location-based services face a massive scalability challenge. We propose a horizontally-scalable and reliable location-based publish/subscribe architecture that can be deployed on a cluster made of commodity hardware. As many modern location-based publish/subscribe systems, our architecture supports moving publishers, as well as moving subscribers. When a publication moves in the range of a subscription, the owner of this subscription is instantly notified via a server-initiated event, usually in the form of a push notification. To achieve this, most existing solutions rely on classic indexing data structures, such as R-trees, and they struggle at scaling beyond the scope of a single computing unit. Our architecture introduces a multi-step routing mechanism that, to achieve horizontal scalability, efficiently combines range partitioning, consistent hashing and a min-wise hashing agreement. In case of node failure, an active replication strategy ensures a reliable delivery of publication throughout the multistep routing mechanism. From an algorithmic perspective, we show that the number of messages required to compute a match

is optimal in the execution model we consider and that the number of routing steps is constant. Using experimental results, we show that our method achieves high throughput, low latency and scales horizontally. For example, with a cluster made of 200 nodes, our architecture can process up to 190'000 location updates per second for a fleet of nearly 1'900'000 moving entities, producing more than 130'000 matches per second.

3.1 Introduction

Today, moving objects and moving users produce massive amounts of geo-located data. For example, it is now common for cellular network operators and data analytics companies to collect up to several millions of geographical data points per seconds. Furthermore, this trend is not likely to stop: according to Gartner[1] and IDC,[2] there will be between 25 to 30 billion connected objects by 2020. The emergence of this ecosystem of connected objects, known as the *Internet of Things* (IoT), opens new opportunities, but also comes with new challenges in terms of development and deployment.

3.1.1 Location-Based Publish-Subscribe

Here the location-based publish-subscribe paradigm is of particular interest for developing mobile applications that want to take advantage of the IoT ecosystem. With this communication paradigm, connected objects are able to issue publications and subscriptions that are geographically scoped and that move with them. The scope of a publication or a subscription is known as its *space*. A match occurs between a given publication and a given subscription if both a *content* criterion and a *context* criterion are met simultaneously. The content criterion expresses a semantic relationship between the publication and the subscription, as captured by traditional publish-subscribe systems such as the Java Messaging Service API.[3] The context criterion then expresses some proximity condition between the publication space and subscription space, hence the term *location-based* or sometimes *location-aware publish-subscribe* [43, 93].

The location-based publish-subscribe paradigm is of great interest for any mobile application that requires a precise and up-to-date knowledge of the context of its users. Therefore, it could be used to improve user experience in various domains including social networking, transportation, video game, and augmented reality. Although this communication paradigm offers great expressiveness and flexibility, scaling its implementation to billions of objects is far from trivial. As of

[1] http://www.gartner.com/newsroom/id/3165317
[2] https://www.idc.com/getdoc.jsp?containerId=US40755816
[3] https://jcp.org/en/jsr/detail?id=368

today, location-based publish/subscribe solutions described in the literature have addressed the scalability problem *vertically*, i.e., with a centralized computing unit responsible for the full workload. Obviously, the vertical scalability approach can only work up to a certain load, that the exponential growth of the IoT ecosystem can easily exceed.

3.1.2 Achieving Horizontal Scalability

The traditional centralized spatial indexing approaches mentioned above aim at taking advantage of high stability in geographical locality (because most objects, such as buildings, shops, landmarks, etc., are not moving), via tree-like data structures. Indeed, such approaches are known to be efficient in contexts where the majority of indexed objects fit on a single machine and are static, i.e., when reads on the index greatly outnumber writes. Keeping the underlying tree-based data structures balanced can be very costly in the presence of numerous writes. Therefore, in order to support a large number of moving objects we need to scale out.

Yet to our knowledge, while many efficient spatial data-structures have been proposed to accelerate the computation of matches between moving publishers and moving subscribers, such as variants of R-trees for instance, the problem has not yet been addressed in terms of *horizontal scalability*, i.e, with many distributed computing units, each one being responsible for only a part of the workload. Here, we consider the problem of horizontally scaling the location-based publish/subscribe communication paradigm. Our starting point consists in fragmenting locality using the notions of range partitioning, in conjunction with consistent-hashing, in order to dynamically distribute the computation of the matches.

3.1.3 Contributions & Roadmap

Our key contributions are organized as follows.

1. In Section 3.2, we describe a model for reasoning about the location-based publish/subscribe paradigm in a distributed context and we introduce the problem of scaling out systems supporting this paradigm.

2. In Section 3.3, we introduce a scalable and reliable location-based publish/subscribe architecture. This architecture scales horizontally due to the counter-intuitive idea that selectively fragmenting locality with a combination of range partitioning, consistent hashing and an a

priori min-wise hashing agreement can help us compute matches efficiently. In addition, an active replication strategy makes this distributed architecture tolerant to node failures.

3. In Sections 3.4 and 3.5, we evaluate our solution both theoretically and experimentally. For the latter, we implement and evaluate our protocol on a real cluster setup and highlight its performances in terms of horizontal scalability, throughput and latency.

We then conclude this paper by discussing related work in Section 3.6 and future research opportunities in Section 3.7.

3.2 Scaling location-based publish-subscribe

We consider a distributed system composed of mobile client nodes, that represent computing devices moving in the field, and of fixed server nodes that represent computing resources in some data center.

3.2.1 Client-Side Model

On the client side, mobile nodes can issue long-lived geographically-scoped publications and subscriptions that move with their issuers. Formally, a publication *pub* is defined as tuple $pub = (id, Z, A)$, where $id \in \mathbb{N}$ uniquely identifies *pub*, $Z \in \mathscr{Z}$ denotes the geographical zone[4] where *pub* is active and set $A = \{a_1, a_2, \ldots, a_{|A|}\}$ denotes a collection of attributes of the form $a = (name, value)$. In other words, A defines the *content* of *pub*, whereas Z defines its *context*. Similarly, a subscription *sub* is defined as tuple $sub = (id, Z, A, issuer)$, where *issuer* uniquely identifies the mobile client node that issued *sub*. As publications and subscriptions move with their issuers, the geographical boundaries where they are active are also moving. Therefore, given a moving subscription *sub* and a moving publication *pub*, we say that a match occurs when the following two conditions are met: $sub.Z \cap pub.Z \neq \varnothing$ and $sub.A \subseteq pub.A$. When such a match occurs, the mobile node that issued *sub* is notified by asynchronously receiving a tuple (pub, sub). It is worth noting that every time a publication moves within the range of a matching subscription, the subscription's issuer will receive an update.

[4] Here $\mathscr{Z}$ denotes the set of all zones definable on the earth surface.

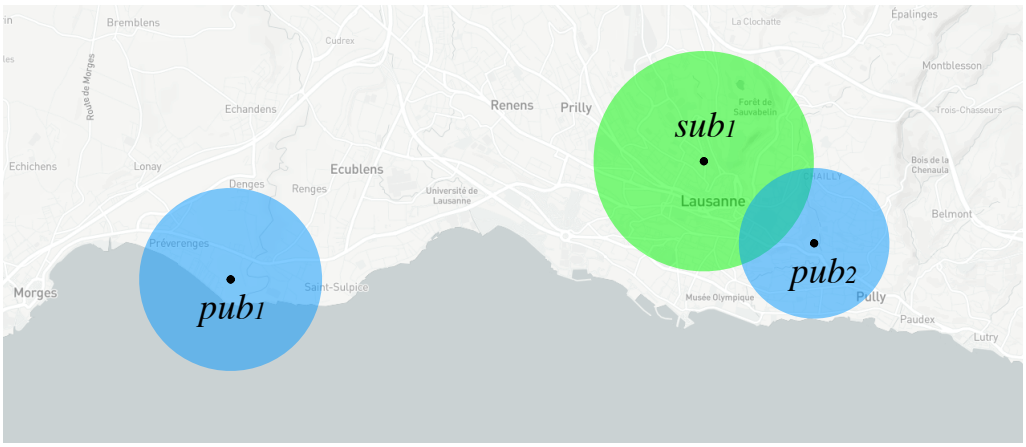


Figure 3.1: Example of location match

Intuitively, the first condition captures the fact that the geographical zones of *pub* and *sub* overlap; this is known as a *context match* or a *location match*. Figure 3.1 depicts two publications pub_1 and pub_2 and a subscription sub_1 that only matches with pub_2 in terms of location. The second condition captures the fact that publication *pub* contains at least all the attributes of subscription *sub*; this is known as a *content match*.

The above client-side model is similar to the one used in [93]. There exists of course many alternative ways to define the notion of content match, but this is out of scope here: in this paper, we focus exclusively on location matches. Similarly, we could define the notion of location match by using an alternative proximity criterion but, as we will see, this would not affect the generality of our approach.

3.2.2 Server-Side Model

On the server side, fixed server nodes consist in inter-connected virtual or physical nodes running on commodity hardware and organised as a cluster in some data centers. In this paper, we consider a set of distributed processes $\Pi = \{p_1, p_2, \cdots, p_n\}$ running on server nodes. Furthermore, we assume that processes in Π know each other and communicate by reliably exchanging uniquely identified messages.

3.2.3 Scaling horizontally

Using the above client-side and server-side models, we can now specify the horizontal scalability problem. We begin by defining how the two models work together, i.e., how nodes on one side communicate with nodes on the other side.

When a client-side mobile node wants to issue a new publication or a new subscription, or when it moves and hence needs to update the geographical zones associated with its existing publications and subscriptions, it sends messages to the cluster. The latter is responsible for computing the matches that are then sent back to the mobiles nodes that issued the subscriptions concerned by those matches. It is worth noting that mobile nodes never communicate directly with each others but always do so via cluster nodes.

Here, we address the problem of distributing the computation of matches among cluster nodes in a way that enables the overall system to *scale horizontally*. That is, we want to answer the following question: How can we organize the work of server-side nodes so that by simply adding new nodes to the cluster, we can manage a growing number of moving publications and subscriptions, while maintaining low latency in term of match computation and delivery?

3.3 A Horizontally Scalable and Reliable Architecture

In this section, we describe a horizontally scalable architecture that supports the location-based publish and subscribe communication paradigm. In contrast to traditional centralized spatial data structures that aim at taking advantage of geographical locality, our solution fragments locality by using the notions of *range partitioning*, in conjunction with *consistent-hashing*, in order to dynamically distribute the computation of matches across sets of processes.

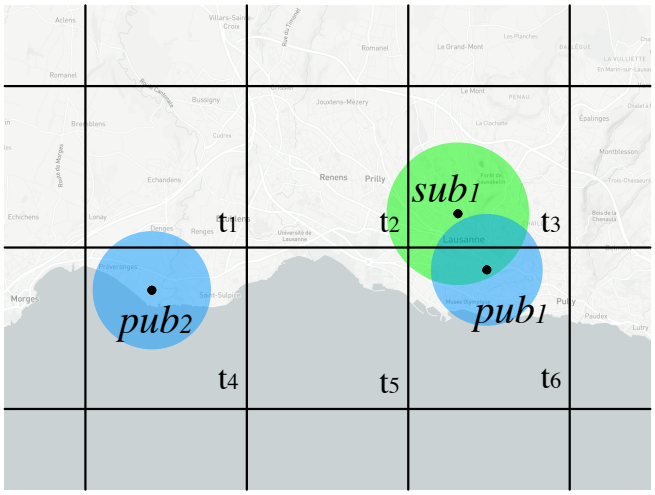
3.3.1 Range Partitioning

Map services such as Google Maps[5] or Mapbox[6] typically rely on a grid layout that divides the world into a set of tiles. In this paper, we use tiles to create range partitions along two dimensions. We then use these partitions to distribute the computation of context matches between publications and subscriptions among processes running in our cluster. For this, we introduce set $G = \{t_1, t_2, \ldots, t_{|G|}\}$: it denotes a grid layout on the earth surface consisting in a set of uniquely

[5] https://maps.google.com
[6] https://www.mapbox.com

identified tiles. In addition, we introduce function $tiles : \mathscr{Z} \rightarrow 2^G$ that maps a geographical zone $Z \in \mathscr{Z}$ to its overlapping set of tiles $T_Z \subseteq G$, i.e., we have $T_Z = \{t \in G \mid t \cap Z \neq \varnothing\}$.

Figure 3.2 illustrates how the *tiles* function is used to determine the sets of tiles overlapping with the publications and subscription depicted in Figure 3.1. Obviously, rectangle-based grids rely on a map projection, which is necessary for mapping coordinates on a sphere to coordinates on a plane. Such map projections are known to introduce spatial distortions that result in the tiles being not uniform in terms of shape and size. However, as in our context tiles are used as units of distribution and parallelism, these distortions have virtually no effect on performance.



$tiles(pub_1.Z) = \{t_3, t_6\}$

$tiles(pub_2.Z) = \{t_1, t_4\}$

$tiles(sub_1.Z) = \{t_2, t_3, t_5, t_6\}$

$tiles(pub_1.Z \cap sub_1.Z) = \{t_3, t_6\}$

$H_{min}(tiles(pub_1.Z \cap sub_1.Z)) = t_3$

Figure 3.2: Example of using the *tiles* function

3.3.2 Consistent Hashing

Distributed hash tables typically rely on a family of functions that offers *consistent hashing* in order to partition data items across a cluster of processes and to scale horizontally. In other words, the unique identifier of each data item is passed to a *consistent hashing function* and the resulting hash value is then used to find the process responsible for handling that particular data item.

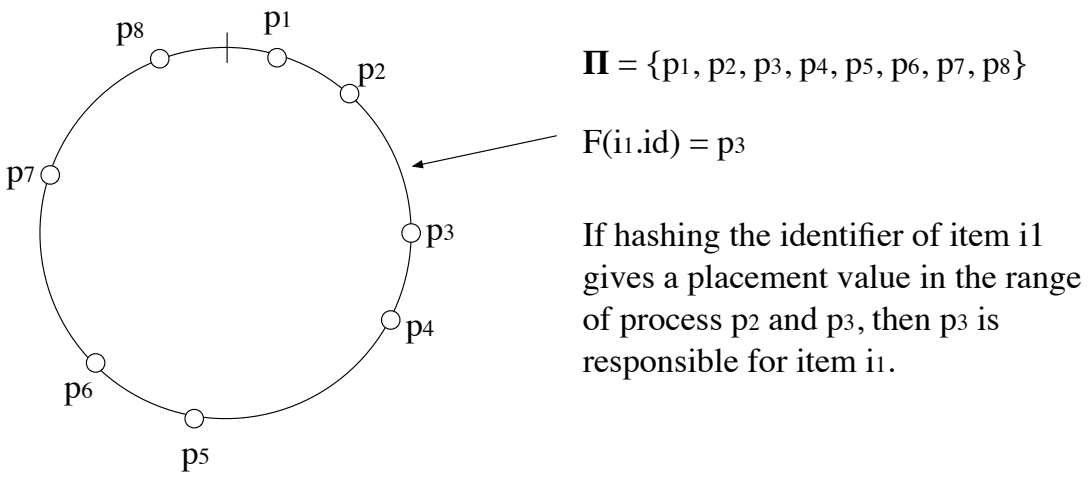


Figure 3.3: Distributing data items across processes in a ring

Formally, consistent hashing can be expressed as function $F : I \rightarrow \Pi$ that distributes a set of data items $I = \{i_1, i_2, \ldots, i_{|I|}\}$ across a set of processes Π. As illustrated in Figure 3.3, it is common to think about consistent hashing as a ring in which the largest possible hash value convolutes to the smallest possible hash value [38]. Each process is assigned a fixed position on the ring, for example by hashing the unique identifier of that process. In order to find the process responsible for handing a given item, the identifier of that item is hashed, then the first process on the ring with a placement value greater than the resulting hash value is selected. Here the monotonicity of F is particularly interesting as it ensures that when processes are added or deleted, the distribution of items across existing processes does not change [78]. However, as illustrated in Figure 3.3, positioning processes by hash values can lead to a non-uniform distribution of the load on the ring. As a consequence, we rely on a notion similar to the "virtual nodes" used by Dynamo [38]. When processes are added to the system, each of them receives multiple positions in the ring, improving the uniformity of the load distribution.

3.3.3 Min-wise Hashing Agreement

Our architecture relies on the tiling of the earth's surface and on a set of consistent hashing functions to partition and distribute the load of computing matches between publications and subscriptions. That is, the subdivision of the earth's surface into tiles is used as a range partitioning criteria, whereas consistent hashing functions are responsible for distributing the load by routing messages to processes. As depicted in Figure 3.2, a problem occurs when the boundaries of a publication or a subscription overlap with several tiles. In such a case, the same match will be computed on

several tiles, here t_3 and t_6, resulting in duplicated messages. A straightforward solution would be to address this issue with an a posteriori agreement, i.e, an centralised process for identifying and eliminating duplicates. However, such a solution would have several disadvantages: First, the detection of duplicates by using a list or a set in an unbounded message stream is not practical due to memory constraints; second, when publications and subscriptions overlap with many tiles, the amount of duplicated messages transmitted in the cluster might result in network congestion.

To avoid these problems completely, we present an efficient a priori min-wise hashing agreement that does not require a centralized coordination. As illustrated in Figure 3.2, both tiles t_3 and t_6 identify the intersection between pub_1 and sub_1 and trigger a match. Hence, by computing the set $tiles(pub_1.Z \cap sub_1.Z)$, each tile is able to infer which other tile will compute the exact same match. Given this fact, a convention can be used to determine which tile is responsible for sending the match to the end user. Let H be a hash function that maps tiles to distinct integers. Given any set of tiles T, we define $H_{min}(T)$ to be the tile $t \in T$ with the minimum hash value. On this basis, the a priori min-wise hashing agreement can be expressed with one condition: given any $tile$ overlapped by both pub_1 and sub_1, a $tile$ transmits the match to the end user only if the condition $tile = H_{min}(tiles(pub_1.Z \cap sub_1.Z))$ is satisfied.

3.3.4 Detailed Architecture and Algorithms

In this section, we provide a more detailed description of the internals of our architecture.

Process roles

When participating in match computation, processes can have one of the three roles described hereafter. When new nodes are added to the cluster to scale out, one or more processes of each role can be started.

1. The *Frontend* role characterizes the set of processes $\Pi_F \subseteq \Pi$ responsible for handling and routing publication and subscription requests in the cluster.
2. The *State Manager* role characterizes the set of processes $\Pi_S \subseteq \Pi$ responsible for tracking and managing the state of publications and subscription in the cluster,
3. The *Tile Manager* role characterizes the set of processes $\Pi_T \subseteq \Pi$ responsible for computing matches between publications and subscriptions that overlap with a specific set of tiles.

Match Triggering Graph

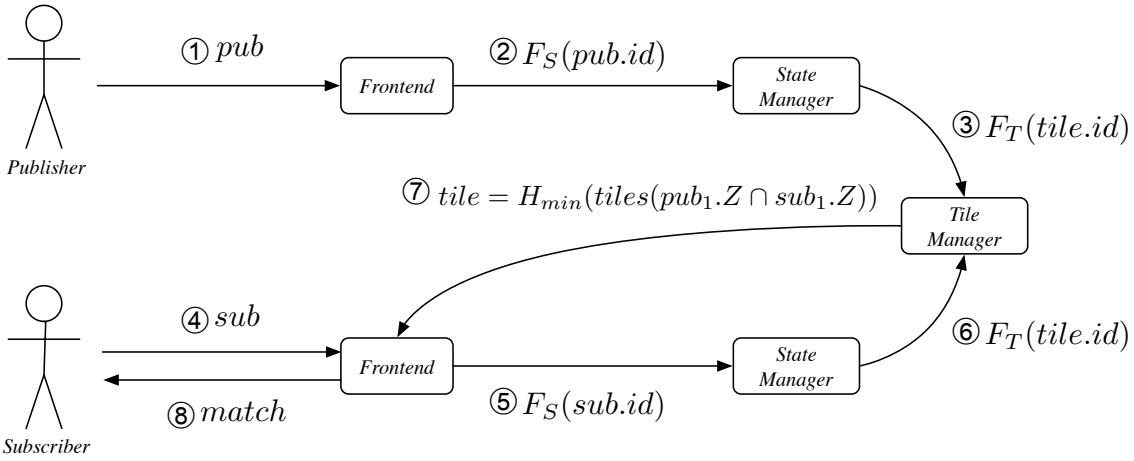


Figure 3.4: Overview of a Match Triggering Graph

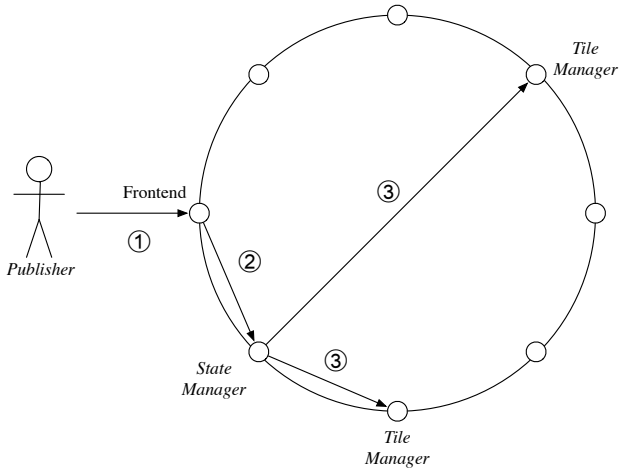
Figure 3.4 gives an overview of how the aforementioned concepts all play together. We use the term *Match Triggering Graph* when referring to the graph that contains all the paths of the messages that lead to a match. In other words, a *Match Triggering Graph* corresponds to the paths that link a *State Manager* process responsible for a particular publication, a *State Manager* process responsible for an overlapping subscription and the *Tile Manager* process that computes a match for this publication/subscription pair.

Message routing

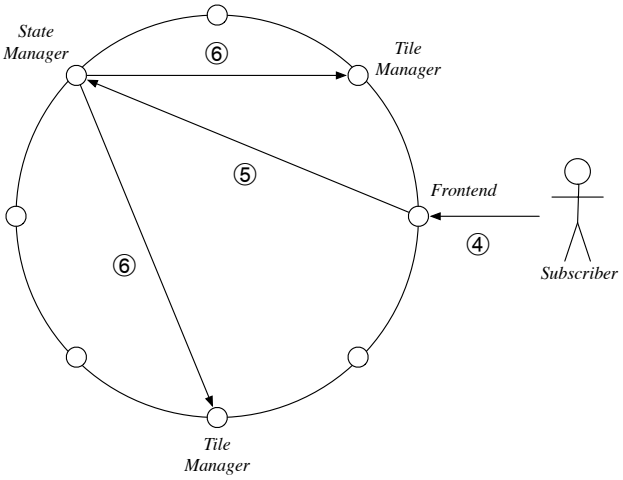
Messages are routed across processes by using the two distinct consistent hashing functions:

1. Function $F_S : \mathbb{N} \rightarrow \Pi_S$ routes messages to *State Manager* processes by hashing publications and subscription identifiers $id \in \mathbb{N}$.
2. Function $F_T : \mathbb{N} \rightarrow \Pi_T$ routes messages to *Tile Manager* processes by hashing tile identifiers $tile.id \in \mathbb{N}$.

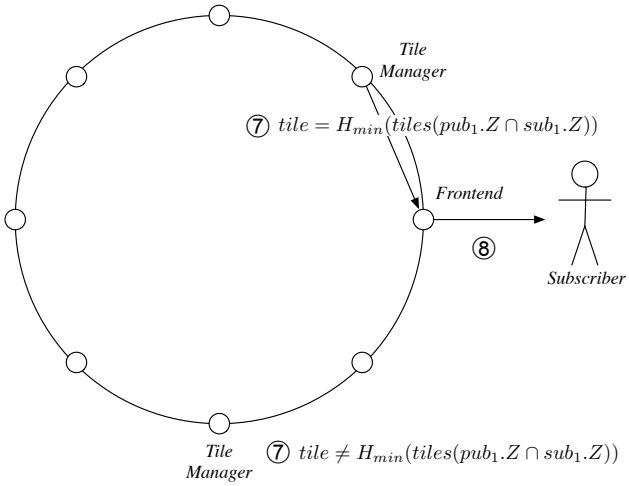
Figure 3.5 illustrates in more details how publications, subscriptions and matches are routed in the cluster. In Figure 3.5a, the publication is first routed from a *Frontend* process to a *State Manager* process. The *State Manager* is responsible for identifying and notifying the tiles that overlap the



(a) Publication routing



(b) Subscription routing



(c) Match routing

Figure 3.5: Distributed routing based on consistent hashing

geographical zone of a publication. Why is this intermediary step between *Frontend* processes and *Tile Manager* processes necessary? When the location of a publication is updated, it might enter some tiles and leave some others. As a consequence, it is necessary to have a process responsible for tracking the state of a publication in the cluster and notifying *Tile Manager* processes in a consistent manner. In Figure 3.5b, a similar routing scenario is depicted for subscriptions. As depicted in Figure 3.5c, the same match can be computed by several *Tile Manager* processes. However, only one of these process satisfies the min-wise hashing condition $tile = H_{min}(tiles(pub_1.Z \cap sub_1.Z))$ and transmits the match to the subscriber. As the computation of this condition is very cheap, it can be verified by every *Tile Manager* process and, as a result, the architecture requires no intermediary step for eliminating duplicated matches.

Frontend algorithm

Algorithm 1 Frontend

upon event $\langle init \rangle$
  $connections \leftarrow \varnothing$

upon event $\langle addPub|pub \rangle$
  send $\langle addPub|pub \rangle$ **to** $F_S(pub.id)$

upon event $\langle deletePub|pub \rangle$
  send $\langle deletePub|pub \rangle$ **to** $F_S(pub.id)$

upon event $\langle addSub|sub \rangle$
  $sub.sender \leftarrow self$
  $connections \leftarrow connections \cup \{(sub.id, connection)\}$
  send $\langle addSub|sub \rangle$ **to** $F_S(sub.id)$

upon event $\langle deleteSub|sub \rangle$
  $connections \leftarrow \{(s,c) \in connections | s \neq sub.id\}$
  send $\langle deleteSub|sub \rangle$ **to** $F_S(sub.id)$

upon event $\langle match|pub, sub \rangle$
  $conn \leftarrow c|(s,c) \in connections|s = sub.id$
  send $\langle match|pub, sub \rangle$ **to** $connection$

A client is not assumed to know which process of the cluster is responsible for managing the state of a particular publication or subscription. Furthermore, in practice, an implementation of the middleware would typically communicate with subscribers through TCP keep-alive connections. As a consequence, the *Frontend* process is typically used behind a load balancer to parse the protocol-specific requests emitted by clients, to route them to the correct server side *State Manager* processes using consistent hashing, and to forward the computed matches back to the subscribers in the protocol specific format. A hypothetic *Frontend* process can receive four kinds of protocol-specific events from the client:

1. $\langle addPub|pub \rangle$ messages are used to add or update the state of a publication *pub* in the cluster;
2. $\langle deletePub|pub \rangle$ messages are used to delete the state of a publication *pub* from the cluster;
3. $\langle addSub|sub \rangle$ messages are used to add or update the state of a subscription *sub* in the cluster;
4. $\langle deleteSub|sub \rangle$ messages are used to delete the state of a subscription *sub* from the cluster.

In Algorithm 1, the $self$ variable corresponds to the $frontend$ process itself and the $connection$ variable corresponds to a client connection, typically a TCP connection.

State Manager algorithm

Algorithm 2 State Manager

upon event $\langle init \rangle$
  $pubs \leftarrow \varnothing$
  $subs \leftarrow \varnothing$

upon event $\langle addPub|pub \rangle$
  $prev \leftarrow p \in pubs$ **such that** $p.id = pub.id$
  for all $tile \in tiles(prev.Z) \setminus tiles(pub.Z)$ **do**
    send $\langle deletePub|tile, pub \rangle$ **to** $F_T(tile.id)$
  for all $tile \in tiles(pub.Z)$ **do**
    send $\langle addPub|tile, pub \rangle$ **to** $F_T(tile.id)$
  $pubs \leftarrow \{p \in pubs | p.id \neq pub.id\} \cup \{pub\}$

upon event $\langle deletePub|pub \rangle$
  $prev \leftarrow p \in pubs$ **such that** $p.id = pub.id$
  for all $tile \in tiles(prev.Z)$ **do**
    send $\langle deletePub|tile, pub \rangle$ **to** $F_T(tile.id)$
  $pubs \leftarrow \{p \in pubs | p.id \neq pub.id\}$

upon event $\langle addSub|sub \rangle$
  $prev \leftarrow s \in subs$ **such that** $s.id = sub.id$
  for all $tile \in tiles(prev.Z) \setminus tiles(sub.Z)$ **do**
    send $\langle deleteSub|tile, sub \rangle$ **to** $F_T(tile.id)$
  for all $tile \in tiles(sub.Z)$ **do**
    send $\langle addSub|tile, sub \rangle$ **to** $F_T(tile.id)$
  $subs \leftarrow \{s \in subs | s.id \neq sub.id\} \cup \{sub\}$

upon event $\langle deleteSub|sub \rangle$
  $prev \leftarrow s \in subs$ **such that** $id = sub.id$
  for all $tile \in tiles(prev.Z)$ **do**
    send $\langle deleteSub|tile, sub \rangle$ **to** $F_T(tile.id)$
  $subs \leftarrow \{s \in subs | s.id \neq sub.id\}$

A *State Manager* process is responsible for managing the state of a subset of publications and subscriptions active in the cluster. It receives four types of messages from *Frontend* processes:

1. $\langle addPub|pub\rangle$ messages are used to add or update the state of a publication *pub* to the state manager;
2. $\langle deletePub|pub\rangle$ messages are used to delete the state of a publication *pub* from the state manager;
3. $\langle addSub|sub\rangle$ messages are used to add or update the state of a subscription *sub* to the state manager;
4. $\langle deleteSub|sub\rangle$ messages are used to delete the state of a subscription *sub* from the state manager.

In addition, a *State Manager* process sends four kinds of message to *Tile Manager* processes:

1. $\langle addTilePub|tile, pub\rangle$ messages are used to add or update the state of a publication *pub* to the *Tile Manager* process responsible for $tile$;
2. $\langle deleteTilePub|tile, pub\rangle$ messages are used to delete the state state of a publication *pub* from the *Tile Manager* process responsible for $tile$;
3. $\langle addTileSub|tile, sub\rangle$ messages are used to add or update the state of a subscription *sub* to the *Tile Manager* process responsible for $tile$;
4. $\langle deleteTileSub|tile, sub\rangle$ messages are used to delete the state state of a subscription *sub* from the *Tile Manager* process responsible for $tile$.

Algorithm 2 describes the internals of a *State Manager* processes. When a user registers a new publication or a new subscription, the *Tile Manager* processes that overlap its geographical zone must be notified. In a similar way, when a user updates the geographical zone of a publication or a subscription, some previously notified *Tile Manager* processes must be left and new ones be notified. As we want these actions to be transparent to the end user, a *State Manager* process records the state of publications and subscriptions and sends the necessary maintenance messages across the cluster. When a *State Manager* process receives a message regarding a publication or a subscription, it uses consistent hashing on tile identifiers $tile.id$ in order to notify the affected *Tile Manager* processes. Each *State Manager* process is responsible for the states of the publications and subscriptions stored in the *pubs* and in the *subs* sets. As these sets only contain the latest publication and subscription

states, their identifiers *pub.id* and *sub.id* are used to identify and eliminate older versions from the sets. We also assume that $tiles(prev.Z)$ returns an empty set when *prev* is null.

Tile Manager algorithm

Algorithm 3 Tile Manager

upon event $\langle init \rangle$
  $pubs \leftarrow \varnothing$
  $subs \leftarrow \varnothing$

upon event $\langle addPub|tile, pub \rangle$
  for all $s \in subs | s.Z \cap pub.Z \neq \varnothing$ **do**
    if $tile = H_{min}(tiles(pub.Z \cap s.Z))$ **then**
      send $\langle match|pub, s \rangle$ **to** $s.sender$
  $pubs \leftarrow \{(tile, pub)\} \cup$
    $\{(t, p) \in pubs | \neg(t = tile \wedge p.id = pub.id)\}$

upon event $\langle deletePub|tile, pub \rangle$
  $pubs \leftarrow \{(t, p) \in pubs | \neg(t = tile \wedge p.id = pub.id)\}$

upon event $\langle addSub|tile, sub \rangle$
  for all $p \in pubs : sub.Z \cap p \neq \varnothing$ **do**
    if $tile = H_{min}(tiles(p.Z \cap sub.Z))$ **then**
      send $\langle match|p, sub \rangle$ **to** $sub.sender$
  $subs \leftarrow \{(tile, sub)\} \cup$
    $\{(t, s) \in subs | \neg(t = tile \wedge s.id = sub.id)\}$

upon event $\langle deleteSub|tile, sub \rangle$
  $subs \leftarrow \{(t, s) \in subs | \neg(t = tile \wedge s.id = sub.id)\}$

A *Tile Manager* process is responsible for computing all the matches between publications and subscriptions whose geographical zones overlap with the zone covered by a specific tile. A *Tile Manager* process receives four kind of messages from *State Manager* processes:

1. $\langle addTilePub|tile, pub \rangle$ messages are used to add or update the state of a publication *pub* in the *Tile Manager* process responsible for $tile$;

2. $\langle deleteTilePub|tile, pub\rangle$ messages are used to delete the state of a publication *pub* from the *Tile Manager* process responsible for $tile$;

3. $\langle addTileSub|tile, sub\rangle$ messages are used to add or update the state of a subscription *sub* in the *Tile Manager* process responsible for $tile$;

4. $\langle deleteTileSub|tile, sub\rangle$ message are used to delete the state of a subscription *sub* from the *Tile Manager* process responsible for $tile$.

In addition, *Tile Manager* processes send match messages in the form of $\langle match|pub, sub\rangle$ to the *Frontend* process attached to the end-user connection of the subscriber. Algorithm 3 shows how collections of publication and subscription states are maintained in *Tile Manager* processes. When a publication state is added or updated, matches are sent to overlapping subscriptions. When a subscription state is added or updated, matches containing the overlapping publications are forwarded to the match filter before reaching the subscription.

3.3.5 Fault Tolerance and Reliability

In order to make our architecture reliable and fault tolerant, we adopt an active replication strategy. In other words, each message is processed by all the replicas and, given a replication factor r, the system should still be able to compute and deliver all the matches when there are at most $r-1$ failing processes. The general idea behind our replication strategy consists into fully replicating the *Match Triggering Graph*. In order to replicate these graphs, we modify our consistent hashing function so that, instead of returning a single process, it returns a list of replicas that contains the r consecutive processes of the hash ring located on distinct physical nodes.

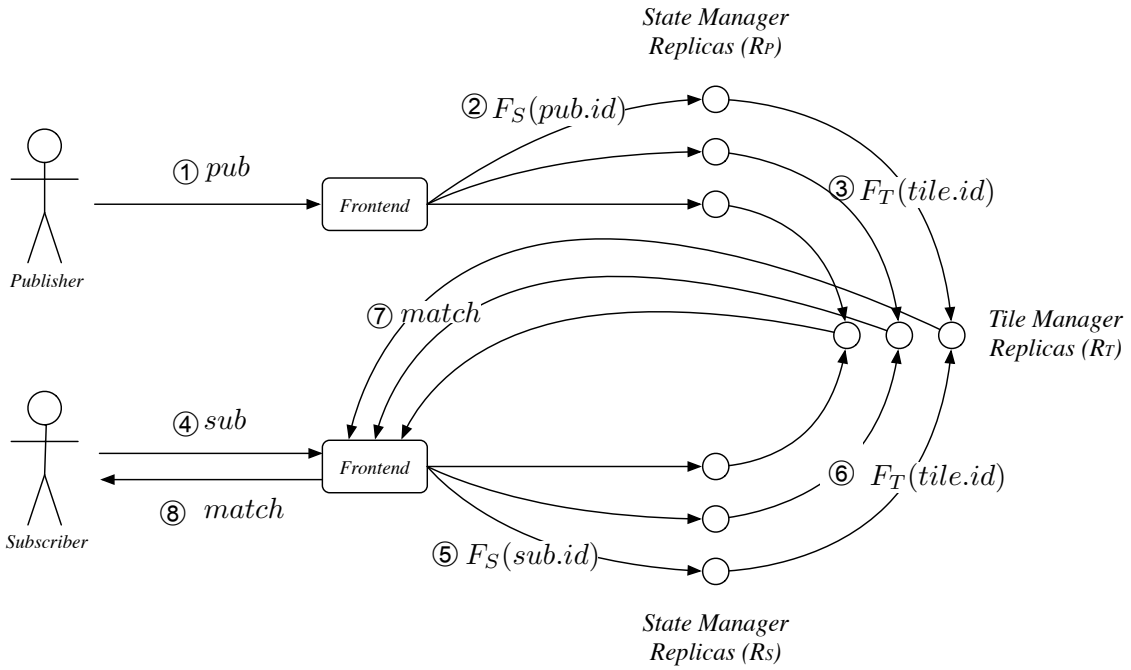


Figure 3.6: Replication of a *Match Triggering Graphs*

Figure 3.6 illustrates in more details this replication mechanism. Here, we first notice a different behaviour in the way *Frontend* processes and *State Manager* processes route messages to replicas. In order to duplicate the routing graph, *Frontend* processes contact all the *State Manager* replicas of the lists R_P and R_S. In order to limit the number of messages, a *State Manager* process from the list R_P only sends one messages to a *Tile Manager* process of the list R_T. The position of *State Manager* process in the list R_S is used to select the corresponding *Tile Manager* process in the list R_T. As a result, the number of messages propagated in the cluster remains proportional to the replication factor.

Independence of failure

The reliability of our architecture relies on the assumption that n node failures do not compromise more than n *Match Triggering Graphs*. However, because of the multiple routing steps involved and the abstraction of virtual nodes typically used in data centers, the lists of nodes associated to the replicated processes can overlap. As a result, several processes along the routing graphs might be located on the same physical node, whose failure might break several graphs and compromise the overall reliability of the system.

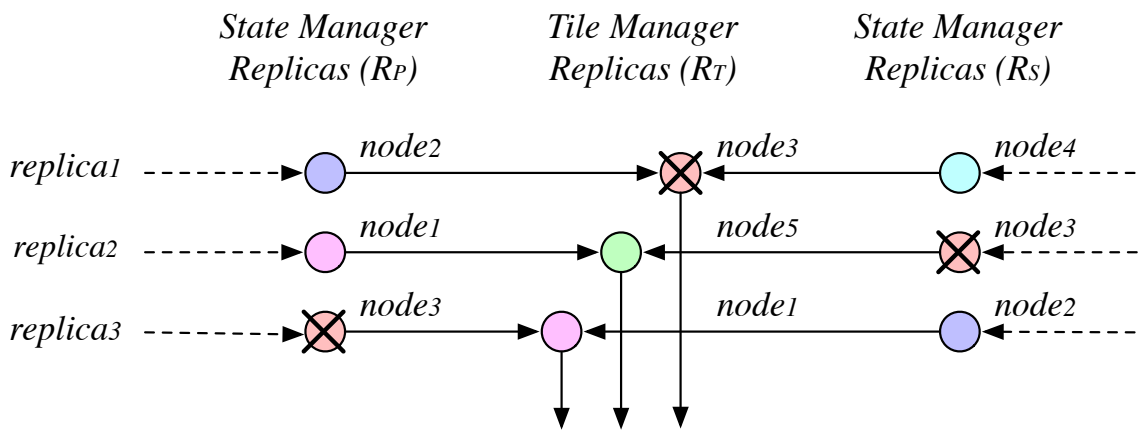


Figure 3.7: Non-Independence of failure in the *Match Triggering Graphs*

Figure 3.7 illustrates such a failure scenario. Here, if $node_3$ fails, all the replicated routing graphs break. A solution to this problem would be to ensure that the lists of physical nodes associated to the lists of replicas R_P, R_T and R_S never overlap. However, enforcing this property requires some sort of agreement.

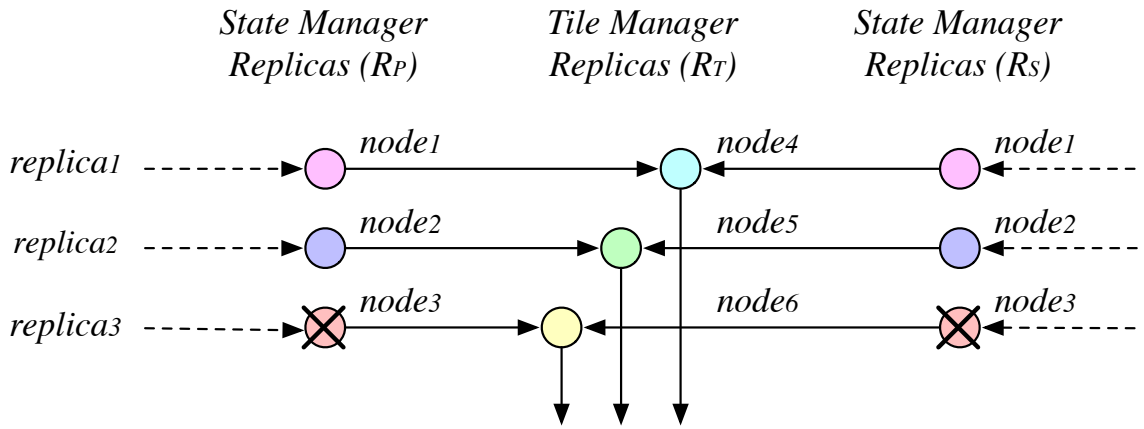


Figure 3.8: Ensuring independence of failure in the *Match Triggering Graphs*

Again, our approach consists in finding an a priori agreement that completely avoid the costs of a distributed agreement or the disadvantages of a centralized solution. At the level of the consistent hashing function, we cannot ensure that the selection of replicas throughout the routing steps will result in non-overlapping lists of physical nodes. However, by creating replication groups in the consistent hash ring, we can ensure that the lists of physical machines associated to the selected

replicas are either fully overlapping or not overlapping at all. In addition, if we order the lists of replicas by the physical addresses of their hosts, we can guarantee that the overlapping lists will always be aligned, i.e., the replicated processes located on the same machine will communicate with each others. Figure 3.8 illustrates such a correct alignement. Here the physical nodes associated to the lists R_P, R_T and R_S are either non-overlapping or fully overlapping. In case of full overlapping, the lists are ordered or aligned by using the physical addresses. As a result, the number of broken routing graphs will never be greater than the number of failures, ensuring the reliability of the system.

3.4 Theoretical Evaluation

In this section, we theoretically evaluate our distributed algorithm. Given a distributed algorithm, two metrics are generally used to measure its performance: the number of messages required for the termination of an operation and the number of communication steps required for its termination [62].

Regarding the first metric, the operation that has the potential to generate the greatest number of messages in the cluster is the insertion or the update of a publication in the cluster. The number of messages generated by the insertion or the update of a subscription will always be lower because only one subscriber is concerned by the operation. In order to calculate the number of messages required to terminate the insertion of a publication, we first introduce the variable $t = |Tiles(pub.Z)|$ that corresponds to the number of tiles affected by the operation. We also introduce the number of matches m generated by the operation such that $m = |\{sub \in subs | sub.Z \cap pub.Z \neq \varnothing\}|$. On this basis, we can easily enumerate the number of messages generated by the insertion of a publication. To inform the *Frontend* process, a first message is required by the client. A second message is sent by the *Frontend* process to the *State Manager* process. Then, t messages are required to inform the *Tile Manager* processes. As the *Tile Manager* processes are able to agree on which process will send the match message to the *Frontent* process without communicating, m messages will be generated. Finally, m messages (i.e., one message per matching subscriber) are required to forward the matches from the *Frontend* processes to the end user. Hence, we end up with a worst case scenario of $1+1+t+m+m$ messages that correspond to a complexity of $O(t+m)$ messages. As a result, knowing that the worst case of a centralized architecture should be characterized by an $O(m)$ complexity, the overhead introduced by our distributed algorithm is optimal in the considered execution model (presented in Section 3.2).

Regarding the second metric, the number of communication steps required for the termination of the distributed algorithm is bounded to five in the worst case. Although at a first glance this

number might seem too high, we demonstrate in the next section that the latency introduced by these steps remains very low, in the context of a data center.

3.5 Experimental Evaluation

In addition to provide a theoretical evaluation of our algorithms, we tested the scalability of our architecture in the context of a large scale cluster setup. In this section, we describe the results we obtained in terms of throughput, reliability, load, memory and latency.

3.5.1 Evaluation setup

Our implementation is written in Scala[7] and relies on a distributed application framework called Akka.[8] This framework relies on the *Actor Model* as an abstraction for distribution, concurrency, parallelism and communication. Therefore, each process described in the architecture corresponds to an actor. We used a Kubernetes[9] cluster hosted in Google Cloud[10] to run our experiments. In Kubernetes, Docker[11] containers are used to accelerate the deployment of applications and a concept named *Daemon Set* can be used to run a copy of an application on a collection of nodes in the cluster. We used StatsD[12] to collect and aggregate statistics across the cluster. We ran our prototype on cluster configurations made of up to 200 virtual machines with two vCPU and 7.5GB of RAM.

3.5.2 Cluster settings

In order to verify that our architecture scales horizontally, we saturated various cluster configurations with move operations. To do so, we created a client application that simulates a fleet of moving entities (either publications or subscriptions) of size n. Moving entities have a circular range with radius of 50 meters, a speed of 20km/h, and send move operations every 10 seconds. Each node in the cluster run an instance of this application and the average CPU load of the node is used to mitigate the amount of move operations generated. In other words, the client applications increase the size of the fleet when the load of the nodes are below a minimal threshold. Inversely, the clients decreases the size of the fleet when a maximal load threshold is reached. As a result, the number of

[7] https://github.com/scala/scala
[8] https://github.com/akka/akka
[9] https://github.com/kubernetes/kubernetes
[10] https://cloud.google.com
[11] https://github.com/docker/docker
[12] https://github.com/etsy/statsd

move operations is dynamically adjusted to the load that the cluster is able to sustain. The members of the fleet respect a uniform density of 100 moving entities per square kilometres. A configuration parameter k corresponds to the ratio of moving entities mapped to publications. Therefore, $p = k * n$ corresponds to the number of publications and $s = (1-k) * n$ corresponds to the number of subscriptions. In our configurations, the size of the tiles is fixed at a width of approximately 1222 meters. Finally, the parameter r corresponds to the replication factor enforced by the cluster.

3.5.3 Horizontal scalability

Figure 3.9 highlights the scalability of our architecture for three distinct scenarios by varying the k parameter. When $k = 0.2$, we have 20% of publications and 80% of subscriptions, which might correspond to the need of a hailing applications. When $k = 0.5$, the balance between publications and subscriptions is perfect, which might correspond to the radar of an autonomous fleet of vehicles. When $k = 0.8$, we hypothetically address the need of an IoT application, where the data of many sensors (publications) are aggregated by moving subscriptions. First, in the three cases, we notice that the number of moves scales almost linearly with the number of nodes. It is interesting to note that, with 200 nodes we were able to sustain the move operations generated by a fleet of 1'900'000 moving entities. Every second, such a load approximately corresponds to 190'000 move operations and 130'000 matches. On a daily basis, this represents a load of more than 15 billion move operations and 11 billion matches. Interestingly, an uneven distribution of publications and subscriptions increases the number of move operations that the system is able to sustain. In fact, as illustrated in Figure 3.9, such distribution tends to reduce the number of matches and releases additional resources for handling move operations.

3.5.4 Reliability overhead

Figure 3.10 highlights the cost associated to reliability. As illustrated here, replicating the *Match Triggering Graph* symmetrically decreases the number of number of move operations that the system is able to sustain and the number of matches it is able to compute. Interestingly, doubling the replication factor does not divide the throughput by two. This is probably due to the fact that the graph is replicated from the frontend process and not from one end to the other. Finally, it is worth noting that the system scales almost linearly for all the tested replication factors.

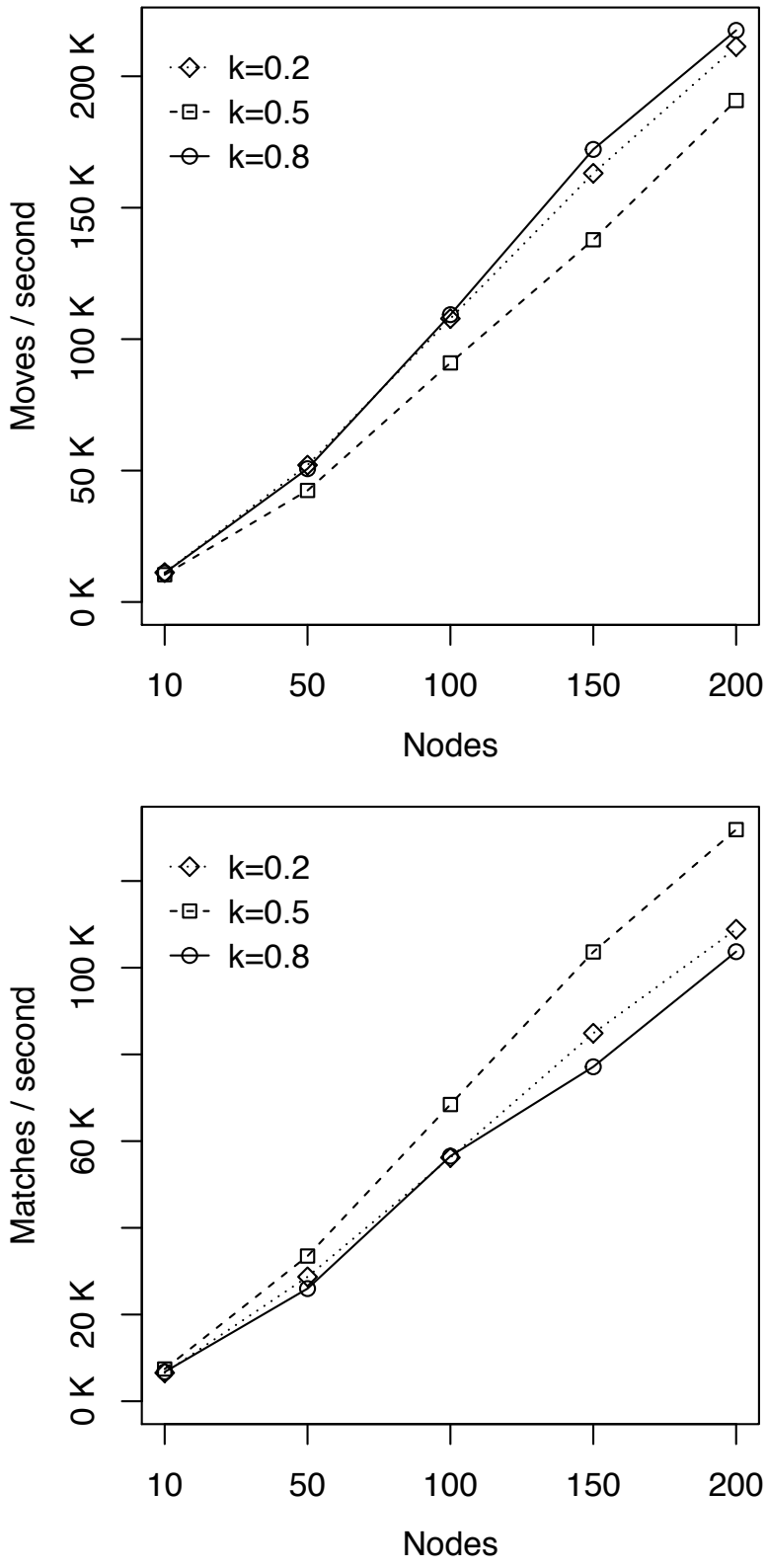


Figure 3.9: Varying the ratio of publications and subscriptions (k)

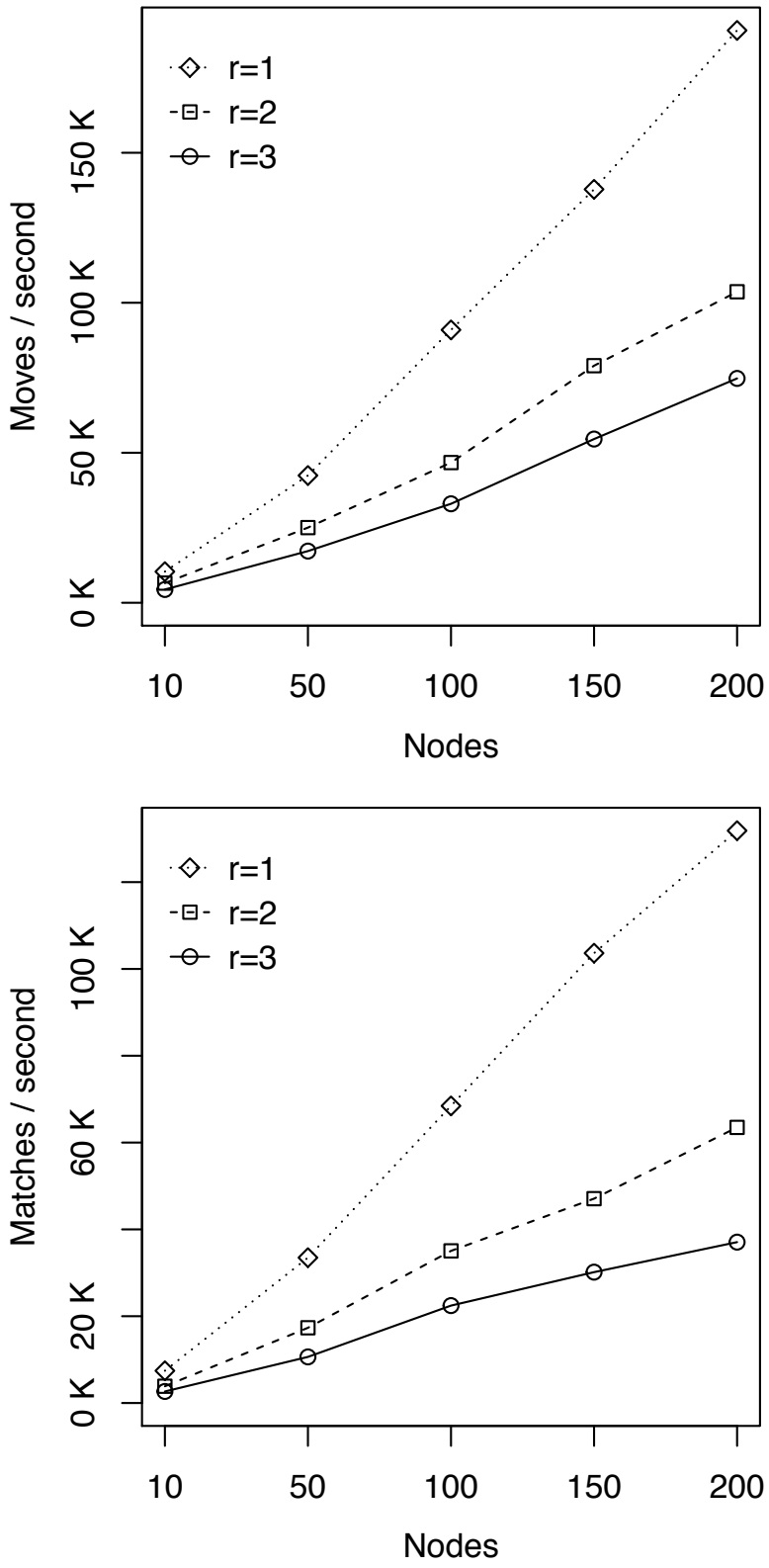


Figure 3.10: Varying the replication factor (r)

3.5.5 Load, memory and latency

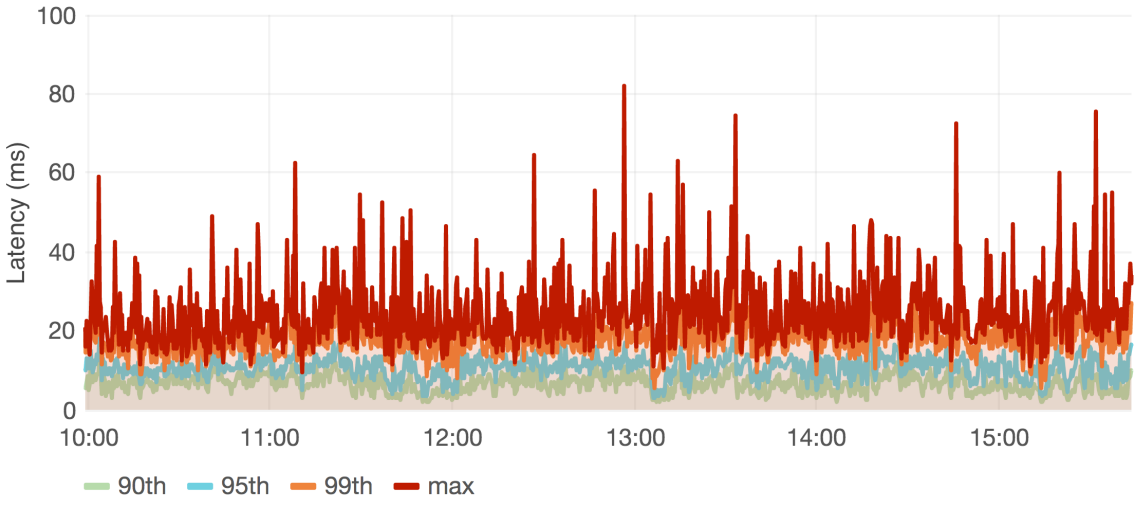


Figure 3.11: Latency (milliseconds) on a 10 nodes cluster

Given a fixed cluster size of 10 nodes, Figure 3.11 shows the latency measured in milliseconds over several hours. Here, latency refers to the time taken by a publication to reach a matching subscription. To measure this kind of end-to-end latency, we collocated a subscription and a matching publications on the same nodes and periodically calculated latency by subtracting the sending time of the publication to its reception time. For this experiment, we started virtual machines in two data centers and guaranteed that the virtual machines were located on different hosts. As highlighted here, 99% of the publications were delivered in less than 50 milliseconds and, in the worst case, latency always remained below 100 milliseconds. Interestingly, we noticed that the latency peaks were often linked to stop-the-world garbage collection tasks performed by Java.

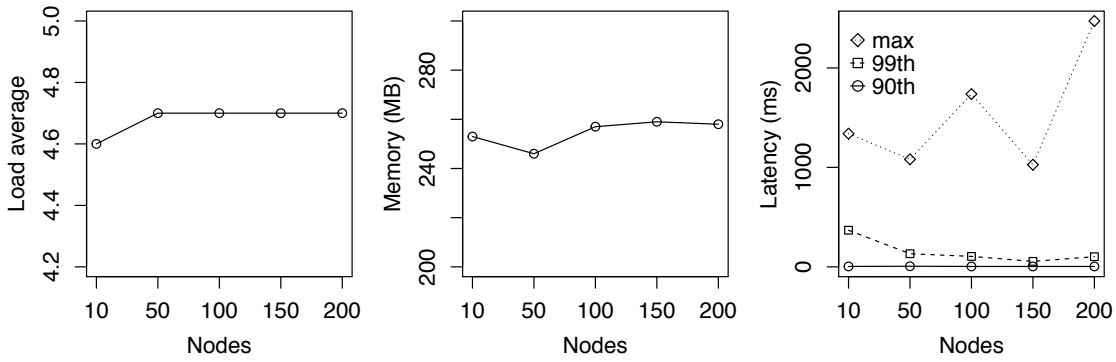


Figure 3.12: Load average, memory and latency

Average CPU load, memory usage and latency are a key metrics that have to remain stable as the size of the cluster grows. As illustrated in Figures 3.12, our solution ensures that these metrics remain stable regardless of the scale. In terms of latency, the max value corresponds to the sole maximal latency measure recorded for a given configuration. Here, the 90th and 99th percentile show that most operations are characterized by a very low latency. Therefore, we conclude that the max value is the result of unpredictable garbage collection tasks and not of a greater cluster size.

3.6 Related Work

3.6.1 Location-Based Publish and Subscribe

To our knowledge, despite the fact that some studies [32, 36, 43, 71] approach location-based publish and subscribe from a distributed-system perspective, none of them focus on horizontal scalability. Interestingly, by addressing scalability in a vertical manner, a huge body of literature recently focused on improving the computation of spatial matches on a single node. For example, Cugola et al. use GPU in order to improve the performance of location-based publish and subscribe [37]. In [127], Yylomenos et al. address the problem of caching data in a publish and subscribe architecture that targets mobile and pervasive devices. In [93] and [131], Guoliang et al. propose an efficient R-Tree based index structure, as well as filtering and pruning algorithms, in order to accelerate the computation of spatial matches. In [74], Hu et al. propose another variant of the R-Tree, the R^I-Tree, to solve the same problem. In [125], Wang et al. propose an alternative index structure, the AP-Tree; it organises records using keywords and spatial data. More recently, in [63], L. Guo et al. described a location-based publish and subscribe system that relies on the concept of safe region in order to reduce communication overhead between a node responsible for the computation of matches and a set of clients. The solution we propose differs from prior work in several ways. For example, in contrast to locality-preserving data structures, such as R-Trees, our solution is somehow counterintuitive because it fragments geographical locality by using consistent hashing in order to improve performance. More importantly, our solution completely departs from approaches that rely on a single node and addresses the problem of horizontally scaling the computation of matches within a cluster.

3.6.2 Continuous KNN Queries

Another field of research closely related to location-based publish and subscribe is the processing of continuous top-k nearest-neighbours queries (KNN). Given a query characterised by a moving

location and some keywords, the problem consists in continuously updating a set of objects that satisfies the specified constraints in terms of context and content. In [4], Bamba et al. define the notions of spatial alarms and safe-region computation: it relies on the assumption that moving queries are performed on static datasets. Under these conditions, it becomes possible to identify areas in which no matching event will occur. Knowing this, the application can simply disconnect when entering a safe-region and reconnect when leaving it, thus greatly reducing network traffic and communication costs. In [66], Hasan et al. present an efficient construction technique for safe-regions, based on range nearest-neighbor queries. In [126], Wu et al. propose a safe-region detection mechanism that include text relevancy. In [76], Huang et al. introduce an alternative representation for safe-region and improve the performance of safe-region construction. The assumption that a part of the data remains static is unfortunately too strong in our case, since our solution specifically supports moving publications and moving subscriptions.

3.6.3 Consistent Hashing

Web services can be subject to what is called the hot spot phenomenon: A web resource can experience a sudden and great popularity, and a single server cannot handle the incoming traffic. In this context, caching and load balancing strategies become vital. In [78], Krager et al. introduced a family of hash functions called consistent hashing: it can be used to relieve hot spots from the web. In [79], the same authors also show how consistent hashing can be used in order to create distributed caches. Among the four original properties described in these papers, two had a huge impact on distributed database communities. Balance and monotone hash functions give good guarantees on the location and the distribution of cached items. These properties are now at the root of most distributed hash tables; and some popular databases such as Dynamo also use them in order to spread and locate records in a distributed system [38]. Another algorithm, called Rendez-vous Hashing [122], developed at the same period achieves the same kind of distributed agreement and can be used as a substitute. Recently, some topic-based publish/subscribe systems use consistent hashing in order to scale horizontally [133, 116, 59]. Contrary to overlapping spatial contexts, topics are strictly isolated from each other. As a result, the architecture we propose for location-based publish/subscribe significantly differs from the usual application of consistent hashing.

3.7 Conclusion and future work

We have highlighted the limitations of existing location-based published and subscribe systems in terms of horizontal scalability and have introduced a model adapted to the context of a dis-

tributed system. On this basis, we have described and implemented a novel architecture based on the assumption that fragmenting locality by using consistent hashing could help compute matches efficiently in a cluster. Furthermore, we have demonstrated that the number of additional communication steps introduced by our protocol, in order to support moving publications and subscriptions, does not compromise horizontal scalability. In addition, we have shown that reliability and fault tolerance can be achieved with a number of messages proportional to the replication factor. Finally, we have showed with a prototype implementation and an experimental evaluation that our architecture can be deployed on commodity hardware and achieve a very high throughput and preserve a low latency.

Although the proposed distributed architecture yields very promising results, it also also raises a certain number of questions. First, the grid we have introduced in this paper is homogenous, whereas human activity is not. Vast areas, such as oceans are mostly empty and a grid that takes this fact into account might produce interesting results. Consequently, we plan to explore different grid topologies that account for the density of cities and the emptiness of remote areas. Second, the related work regarding safe-region detection highlighted the communication cost linked to continuous transmission of matches. Although safe-regions are based on the assumption that a part of the data remains static, Guo et al. demonstrate in [63] that it can be adapted to moving publications and subscriptions in the context of a single node. Therefore, we plan to investigate the possibility of detecting and constructing safe-regions in the context of a distributed system.

Part II

Past: Indexing Trajectories

Chapter 4

An Efficient Type-agnostic Approach for Finding Sub-sequences in Data

Best Paper Award. Bertil Chapuis, Benoît Garbinato, and Periklis Andritsos. An efficient type-agnostic approach for finding sub-sequences in data. In *19th International Conference on High Performance Computing and Communications; 15th International Conference on Smart City; 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 270–277. IEEE, 2017

Abstract

In this paper, we present an efficient type-agnostic approach for finding sub-sequences in data, such as textual documents or GPS trajectories. Our approach relies on data deduplication for creating an inverted index. In contrast with existing data deduplication techniques that split raw sequences of characters arbitrarily, our approach preserves the semantics of the original sequence via the notion of token and can be used to index normalized data. When compared to indexing methods that preserve the semantics and operate on normalized data, our method increases the relevance of the inverted index, reduces its size and improves its performance. As data normalization is generally not used beyond the scope of textual data, we introduce a framework that helps identify the extent to which data should be normalized regardless of its type. On this basis, we demonstrate with a dataset made of GPS trajectories that our method can be used agnostically: it can be used to index and query data of a completely different type. Finally, we show that the resulting spatial-index is characterized by a better discrimination than classic spatial-indexing approaches.

4.1 Introduction

We are witnessing an unprecedented rise in the demand for data processing. A large part of data consists of ordered sequences that can be processed in a batch or in a streaming fashion. A common need when processing data is to find common sub-sequences between a query and sequences of items. In this paper, we introduce an efficient type-agnostic approach for finding sub-sequences in sequences of data. We refer to a token as a semantic piece of information that should be preserved as the sequences of data are handled. Tokens can be words in the context of textual data but can also refer to groups of attributes such as the coordinates that compose GPS trajectories. When searching for sub-sequences in data, a common approach consists in segmenting the sequences that compose the dataset. Two very practical segmenting approaches are generally used by indexing solutions for finding sub-sequences in data.

- **N-grams**. The most common technique for finding sub-sequences (mainly for textual data) is called *n-grams* [96]. By computing all the possible overlapping sub-sequences of size n in a sequence of tokens, n-grams give exhaustive results. This approach comes with a high cost in terms of storage requirements that tends to increase with the overlapping factor n. When used in the context of spatial-indexing, the number of possible n-gram combinations explodes and disqualifies this technique.

- **Hash-based**. Another approach for producing segments, called hash-based (HB), identifies non-overlapping contiguous sub-sequences of tokens, called *chunks* [18]. Chunks can be identified by comparing the hash sums of the successive tokens to some constant. If the hash sum is equal to the constant, then a chunk boundary is set, otherwise the next token is hashed and the process is repeated until one is found. This approach requires less storage and computing power than n-grams, but gives non-exhaustive results. In addition, as the number of possible hash sums is limited to the number of possible tokens, it tends to produce many small chunks, which lead to the detection of short and irrelevant sub-sequences.

When studying these two segmentation approaches, we observe that there exists a clear tradeoff between the n-gram approach (that requires a lot of resources) and the hash-based approach (that trades effectiveness for efficiency). Some data deduplication techniques, such as content-defined chunking (CDC), might be considered as optimized variants of the hash-based approach. These techniques greatly improve the aforementioned tradeoff, which is significant in the context of data storage. However, data deduplication operates on raw sequences of bytes and does not preserve the semantic of tokens.

4.1.1 Contributions

In this paper, we introduce token-based chunking (TB), an approach for segmenting sequences of tokens regardless of their type. The chunking mechanism used by this approach is inspired by CDC but preserves the semantics of tokens and overcomes the issues associated with HB. Compared to other segmenting techniques, TB greatly improves the effectiveness of the index and reduces its size. We propose a normalization framework that can be used to evaluate and to determine the best normalization settings. Normalization is relatively intuitive in the context of textual data. However, the extent to which non-textual data, such as GPS trajectories, should be normalized is harder to determine. Finally, we use TB for indexing trajectories and demonstrating that it overcomes the discrimination problem that characterizes traditional spatial-indexing methods.

In the following sections we show that TB can be used as a general purpose technique for building indexes on various kinds of sequential data. In Section 4.2, we give a bird's eye view of our approach. We provide a detailed description of TB in Section 4.4. In in Section 4.5, we describe our normalization framework and show to what extent data should be normalized regardless of its type. In Section 4.6, we present a detailed evaluation of TB with textual data. In Section 4.7, we show how TB can be used to create an index for GPS trajectories. Finally, in Section 4.8, we highlight previous works from the data storage and the data deduplication fields.

4.2 Overall Approach

An inverted index is usually composed of terms from a dictionary, each of which points to a posting list that contains document identifiers. Boolean queries are then used to retrieve documents that contain a set of query terms. Alternatively, phrase queries are used to take the position of the term in the document into account. When building an index for searching for sub-sequences in documents, using the terms from this dictionary usually gives poor results both in terms of efficiency and effectiveness. Therefore, segmentation techniques such as n-gram, CDC or HB are used to create the terms of the inverted index. The approach for creating the index could be summarized in the phases listed hereafter.

1. **Extraction**. This phase depends on the type of the data sequence and extracts only meaningful data. For example, in the case of XML data, the unnecessary tags and attributes are usually removed.

2. **Tokenization**. This phase splits the extracted data into small pieces called tokens. In the case of textual data, punctuation and white spaces can be used to create a sequence of tokens that corresponds to words.

3. **Normalization**. Token normalization comprises all the operations that can be performed on tokens (such as stop-word removal, equivalence classes, case folding, true casting, stemming or lemmatization).

4. **Segmentation**. When looking for sub-sequences, segmentation techniques, such as n-grams or HB, are used to compute overlapping or non-overlapping sequences of tokens that are then used as dictionary terms.

5. **Indexing**. This phase creates the inverted index by adding the segments and the document identifiers to the dictionary and the posting lists.

CDC operates directly on sequences of bytes or characters and Phases 2 and 3 are usually skipped [50, 9]. In contrast, TB operates on tokens and benefits from the tokenization and normalization phases.

4.3 Background and Model

Our approach is built on results from three domains, namely *tokenization*, *normalization* and *data deduplication*. To describe how these techniques operate, we first introduce the notion of an alphabet, consisting of a finite set of symbols. Formally, we define Σ_i, an alphabet containing bit sequences of fixed length i, e.g., $\Sigma_1 = \{0,1\}$ and $\Sigma_2 = \{00,01,10,11\}$; we have $|\Sigma_i| = 2^i$ being the size of the alphabet Σ_i. Next, we assume that the considered alphabet is Σ_8, the set of all possible bytes that can be used to represent ASCII characters. We also define a *byte word* as a sequence of variable size made of symbols from Σ_8. Using Kleene closure, we have Σ_8^*, the set of possible byte words. Assuming Σ_8^n is the set of all byte words of exactly size n, we have:

$$\Sigma_8^* = \bigcup_{n \in \mathbb{N}} \Sigma_8^n \tag{4.1}$$

Hereafter, we introduce the concepts and terminology that are useful for understanding our approach.

4.3.1 Tokenization

In the context of textual data, tokenization splits sequences of characters into words of a given language. In other words, tokenization is a process that takes a sequence of bytes as input and produces a sequence of meaningful tokens as output. Tokens belong to the infinite alphabet $T = \Sigma_8^*$. On this basis, we can define tokenization as a function $f_c : \Sigma_8^* \to T^*$, where T^* is a Kleene closure on T that contains all the possible tokens. For example, assuming the considered language is English and bytes are interpreted as ASCII characters, byte sequence $\langle The\ quick\ brown\ fox \rangle$ is tokenized as $\langle\ \langle The \rangle, \langle quick \rangle, \langle brown \rangle, \langle fox \rangle\ \rangle$, which we simply write $\langle The, quick, brown, fox \rangle$ in the rest of the paper.

4.3.2 Normalization

In many cases, tokens can be different but convey similar semantics. This is the case, for example, when a word starts with a capital letter at the beginning of a sentence. Normalization removes such superficial differences, so that a match can occur on semantically similar tokens. We define it as a function $f_n : T^* \to T_n^*$, where T_n is a set of tokens that depends on the type of normalization being applied. For example, a case-folding normalization function, which simply replaces capital letters by lowercase letters, would produce tokens in a subset of T. In contrast, a stemming normalization function, which defines heuristics for making similar words converge toward the same tokens, would produce tokens that do not necessarily belong to the English language.

4.3.3 Data Deduplication

Data deduplication can be seen as a particular approach to data compression; it operates on large corpora of files, rather than independent files. At the heart of data deduplication techniques, we find chunking algorithms that process multiple files in order to find common data chunks. Formally, a chunking algorithm takes a sequence of data as input, in the form of one (long) byte word $w \in \Sigma_8^*$, and returns a sequence of byte words $r_w = \langle w_1, w_2, \cdots, w_k \rangle$, called *the recipe* of w, such that $\forall w_i \in r_w : w_i \in \Sigma_8^*$ and $w = w_1 \| w_2 \| \cdots \| w_m$. The byte words of recipe r_w are precisely what we call chunks. For example, two possible recipes for byte word $\langle 25763537 \rangle$ are $\langle \langle 25 \rangle, \langle 76 \rangle, \langle 35 \rangle, \langle 37 \rangle \rangle$ and $\langle \langle 257 \rangle, \langle 6353 \rangle, \langle 7 \rangle \rangle$.

Another way to understand recipe r_w consists in introducing a new alphabet $C = \Sigma_8^*$, which contains all the possible byte words (the symbols of that new alphabet), and seeing r_w as a word built using symbols of C. Note that C is an infinite alphabet, contrary to Σ_8. Using Kleene closure

again, we have C^*, the set of all possible recipes. Assuming C^n is the set of all recipes of exactly size n, we have:

$$C^* = \bigcup_{n \in \mathbb{N}} C^n \tag{4.2}$$

On this basis, we define chunking as a function $f_c : \Sigma_8^* \to C^*$ such that $f_c(w) = r_w$, with w and r_w satisfying the constraints mentioned earlier. Note that in practice, each chunk is stored only once and is referenced in one recipe or (hopefully) more (hence the deduplication). That is, the recipe is a type of meta-data containing only references to actual chunks from which the original data sequence can be recreated.

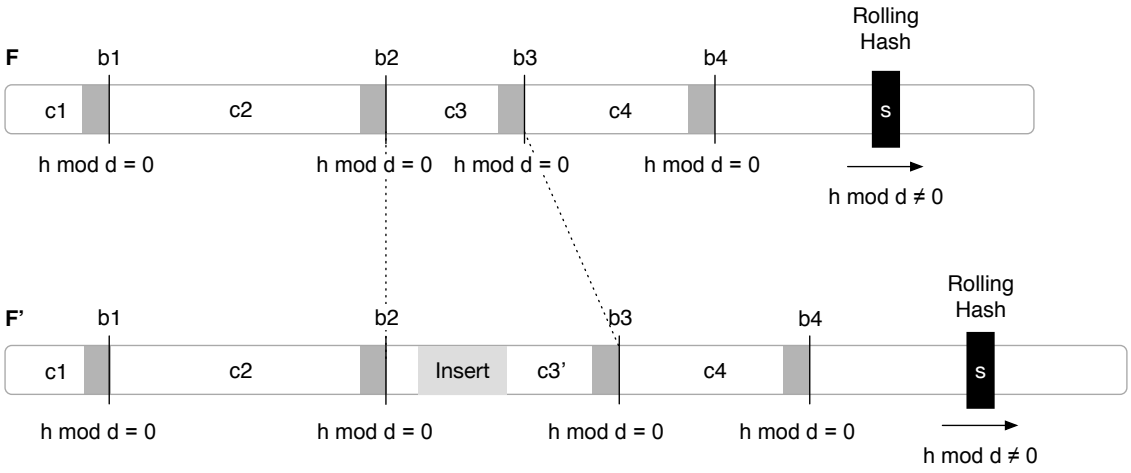


Figure 4.1: Content-defined chunking (CDC)

Fixed-size chunks are common but come with a major drawback: when bytes are added at the beginning of a sequence, all the following chunk boundaries are shifted. CDC solves this issue by detecting addition-resistant chunk boundaries [42]. Figure 4.1 depicts how a simple CDC algorithm operates. A rolling hash function [111] of size s, depicted here by a black box, slides over a file F one byte after the other. After each move, the rolling hash function computes a hash sum h for the bytes that are located in the window. Chunk boundaries are detected by checking the condition $h \ mod \ d = 0$ on the hash sums produced by the sliding window, where d is a user defined divisor. The divisor d is typically used to manage the average chunk size. As the boundaries are based on the content, they remain correct in the face of insertions. If we assume two successive versions of a file, such that F' is a copy of F with an insertion in the middle, the algorithm will be able to find the same boundaries again and to isolate the chunk in which the insertion occurred.

4.4 Token-based chunking

As stated in Section 4.1, CDC breaks the semantics of tokens. This problem comes from the fact that CDC relies on rolling hash functions to identify breakpoints. In order to compute hash sums on sequences of bytes, rolling hash functions usually map the symbols of the alphabets Σ_8 to a set of precomputed random irreducible polynomials. As the alphabet Σ_8 is limited in size, these random polynomials are necessary in order to uniformly and randomly distribute the hash sums produced by the rolling hash function over the hash sum space [111].

Precomputing a set of irreducible random polynomials works well when the size of the alphabet is small and known in advance, such as in the case of Σ_8. In our case, however, we use an alphabet made of tokens T, which is potentially infinite, i.e., $|T| = \infty$. Therefore, it is not possible to precompute one irreducible random polynomial per token. Furthermore, in contrast to CDC, we handle sequences of tokens and produce recipes that contain sequences of tokens. Another way to understand this issue consists in introducing a new alphabet $C_T = T^*$, which contains all the possible sequences of tokens. Again, a Kleene closure can be used to define C_T^*, the set of all possible recipes over the alphabet T, such that:

$$C_T^* = \bigcup_{n \in \mathbb{N}} C_T^n \tag{4.3}$$

Using this definition, we can formally define TB as the function $f_{tb} : T^* \to C_T^*$. The algorithm we introduce does not rely on a precomputed set of irreducible random polynomials in order to detect chunk boundaries and satisfy this definition by producing chunks that consist of sequences of tokens.

The algorithm is decomposed into two parts. The first part is given in Algorithm 4 and is responsible for detecting chunk boundaries. The main configuration parameters are a minimum (*min*) and a maximum (*max*) chunk size, a divisor (d) and a window size (s). As the tokens are consumed, a rolling hash function produces hash sums; new chunks are produced when these hash sums meet a certain criterion, represented here by the expression $h \bmod d = 0$. TB differs from CDC in the sense that it consumes tokens instead of bytes and produces sequences of tokens instead of sequences of bytes.

Algorithm 4 Token-based chunking algorithm

initialize(min, max, d, s):

$chunk \leftarrow \{\varnothing\}$

$hash \leftarrow$ a rolling hash function of size s

read(token):

$chunk \leftarrow chunk :: token$

$h \leftarrow hash.slide(token)$

if $|chunk| \geq min$ **and** ($h \bmod d = 0$ **or** $|chunk| \geq max$) **then**

write$(chunk)$

$chunk \leftarrow \{\varnothing\}$

The second part of the algorithm is the rolling hash function given in Algorithm 5. As it is not possible to precompute random irreducible polynomials for a vocabulary of an unknown size, we assume that the hash sums produced by hashing the tokens are random enough to replace the precomputed irreducible random polynomials. To do so, we use a fast non-cryptographic hash function called Murmur Hash which produces 32 bit integers. To represent the incoming and outgoing tokens, a fixed size sliding window of hash sums is maintained over the sequence of tokens. Our evaluation setup shows that the sums produced by this rolling hash function reach the desired properties and produce balanced hash sums.

Algorithm 5 Token-based rolling hash function

initialize(s):

$a \leftarrow 31$

$b \leftarrow a^s$

$hash \leftarrow$ a murmur hash function

$window \leftarrow$ an array of size s filled with 0

$position \leftarrow 0$

$h \leftarrow 0$

slide(token):

$in \leftarrow hash.digest(token)$

$out \leftarrow window[position]$

$window[position] \leftarrow in$

$position \leftarrow (position + 1) \mod s$

$h \leftarrow a * h + in - b * out$

return h

The segments generated by this algorithm can be used as dictionary terms and overcome the following problems. First, a common pitfall of HB lies in the fact that the hash sum of a single token is used to identify chunk boundaries. Therefore, HB could identify boundaries for very frequent tokens such as *the*. In contrast, our algorithm is not sensitive to token frequency as it computes hash sums over a sliding window. Second, the main problem associated with CDC relies on the fact that it can break the semantics of the sequence of tokens. Unlike CDC, our method preserves the semantics of the tokens. In addition, the introduction of thresholds tends to mitigate the risk of extracting small or large segments that would impact precision and recall negatively. As we will show later, our algorithm is characterized by a much improved effectiveness and efficiency compared to its counterparts.

4.5 Normalization framework

When handling textual data, the extent to which normalization is performed is often based on simple intuitions. However, these intuitions are not valid when dealing with different types of data, such as GPS trajectories. In this section, we introduce a normalization framework for evaluating the extent to which normalization should be performed regardless of the data type. By observing the evolution of precision and recall on a very simple textual dataset normalized at different levels, we show when the normalization of the data should be strengthened or weakened.

In information retrieval, precision and recall are often used to measure the effectiveness of an index, so we begin by recalling these metrics. Precision corresponds to the fraction of retrieved items that are relevant. In other words, $precision = tp/(tp+fp)$, with tp (true positive) the number of relevant items retrieved and fp (false positive) the number of irrelevant items retrieved. Recall corresponds to the fraction of relevant items that are retrieved. More formally, $recall = tp/(tp+fn)$, with tp the number of relevant items retrieved and fn (false negative) the number of relevant items that have not been retrieved.

Another important measure, when looking for similarities, is the Jaccard similarity coefficient. In our context, this coefficient can be used to measure the similarity between two recipes, which corresponds to sets of chunks. Therefore, given two recipes $r_1, r_2 \in C_T^*$, their similarity coefficient is:

$$J(r_1, r_2) = \frac{|r_1 \cap r_2|}{|r_1 \cup r_2|} \tag{4.4}$$

We previously stated that a good normalization function should make highly similar sequences of tokens converge to more similar recipes. So, given two highly similar sequences $s_a, s_b \in T^*$, a good normalization function f_n should have the following property:

$$J(f_{tb}(f_n(s_a)), f_{tb}(f_n(s_b))) > J(f_{tb}(s_a), f_{tb}(s_b)) \tag{4.5}$$

Unfortunately, this metric does not indicate when one should stop making normalization more aggressive in the context of chunks. Given an index and a query that use the same normalization function, precision should remain stable and should be close to 1, as the probability of having two large identical sequences of words in unrelated documents is low. Whereas, recall should increase as more relevant items will be found in the set of relevant items. Therefore, we can say that precision and recall can be used to identify the optimal extent of a normalization function, and we demonstrate it in the following paragraphs.

In the case of textual data, a tokenizer is used to split a given text into a sequence of tokens that belong to the alphabet T. A sequence of tokens can be altered during the normalization phase to make highly similar tokens converge toward the same token. In order to illustrate more practically the effect of data normalization on a small dataset, we consider a set S of four sequences of tokens $s_1, s_2, s_3, s_4 \in \Sigma_8^*$. We assume that the deduplication of these sequences results in four recipes $r_1, r_2, r_3, r_4 \in C^*$ containing a single chunk after deduplication, such that:

$$r_1 = f_{tb}(s_1) = \{\langle A, fox, runs \rangle\} \tag{4.6}$$
$$r_2 = f_{tb}(s_2) = \{\langle The, foxes, run \rangle\} \tag{4.7}$$
$$r_3 = f_{tb}(s_3) = \{\langle Master, fox, is, running \rangle\} \tag{4.8}$$
$$r_4 = f_{tb}(s_4) = \{\langle But, chickens, are, running, faster \rangle\} \tag{4.9}$$

We now consider a query such that $r_q = f_{tb}(s_q) = \{\langle A, fox, runs \rangle\}$ and assume that both s_1 and s_2 are relevant answers. If s_q is used to retrieve relevant items in S by looking for exact duplicates, we expect a single result that corresponds to s_1. In this case, precision would be $1/(1+0) = 1$ and recall would be $1/(1+1) = 1/2$. We now consider a normalization function for textual data f_n^a, which removes common stop-words, applies some case-folding rules and does some stemming on verbs and adjectives. The resulting recipes after normalization and deduplication might look like this:

$$r_1^a = f_{tb}(f_n^a(s_1)) = \{\langle fox, run\rangle\} \tag{4.10}$$

$$r_2^a = f_{tb}(f_n^a(s_2)) = \{\langle fox, run\rangle\} \tag{4.11}$$

$$r_3^a = f_{tb}(f_n^a(s_3)) = \{\langle master, fox, run\rangle\} \tag{4.12}$$

$$r_4^a = f_{tb}(f_n^a(s_4)) = \{\langle chicken, run, fast\rangle\} \tag{4.13}$$

By using the same normalization function on the query s_q, we end up with the recipe $r_q^a = f_{tb}(f_n^a(s_q)) = \{\langle fox, run\rangle\}$ and we expect two results that correspond to the two relevant sequences of token s_1 and s_2. In this case, precision would still be $2/(2+0) = 1$, but recall would improve at $2/(2+0) = 1$. We can now easily show that our assertion regarding Jaccard similarity is true since:

$$J(r_1^a, r_2^a) = \frac{|\{\langle fox, run\rangle\} \cap \{\langle fox, run\rangle\}|}{|\{\langle fox, run\rangle\} \cup \{\langle fox, run\rangle\}|} = \frac{1}{1} = 1 \tag{4.14}$$

is greater than:

$$J(r_1, r_2) = \frac{|\{\langle A, fox, runs\rangle\} \cap \{\langle The, foxes, run\rangle\}|}{|\{\langle A, fox, runs\rangle\} \cup \{\langle The, foxes, run\rangle\}|} = \frac{0}{2} = 0 \tag{4.15}$$

In order to determine when we should stop making a normalization function more aggressive, we consider a second normalization function f_n^b that only retains nouns and drops all the other words. The resulting recipes after normalization and deduplication might look like this:

$$r_1^b = f_{tb}(f_n^b(s_1)) = \{\langle fox\rangle\} \tag{4.16}$$

$$r_2^b = f_{tb}(f_n^b(s_2)) = \{\langle fox\rangle\} \tag{4.17}$$

$$r_3^b = f_{tb}(f_n^b(s_3)) = \{\langle fox\rangle\} \tag{4.18}$$

$$r_4^b = f_{tb}(f_n^b(s_4)) = \{\langle chicken\rangle\} \tag{4.19}$$

In this case, precision would drop to $2/2+1 = 2/3$ and recall would remain stable at $2/(2+0) = 1$. As a consequence, we get an idea of when it is sound or not to normalize in the context of TB. Although recall improves, the data can be normalized more aggressively. On the contrary, a drop in precision indicates that normalization is too aggressive and the tokens do not capture what characterizes the sequence anymore.

4.6 Evaluation

Method	Possible chunks
1-gram	$\{\langle the\rangle, \langle quick\rangle, \langle brown\rangle, \langle fox\rangle\}$
2-gram	$\{\langle the, quick\rangle, \langle quick, brown\rangle, \langle brown, fox\rangle\}$
3-gram	$\{\langle the, quick, brown\rangle, \langle quick, brown, fox\rangle\}$
4-gram	$\{\langle the, quick, brown, fox\rangle\}$
HB	$\{\langle the\rangle, \langle quick, brown, fox\rangle\}$
CDC	$\{\langle the\ quic\rangle, \langle k\ brown\ fox\rangle\}$
TB	$\{\langle the, quick\rangle, \langle brown, fox\rangle\}$

Table 4.1: Indexing methods

In this section, we compare TB with some other state of the art segmentation methods used for building inverted indexes and finding similarities in document corpora. Table 4.1 enumerates these methods and illustrates some possible segments produced by consuming the sequence of four tokens $\langle the, quick, brown, fox\rangle$. As illustrated, the methods based on n-gram compute all the possible contiguous overlapping sequences of tokens. HB produces non-overlapping chunks of variable size [18]. CDC finds chunk boundaries based on bytes, hence a token can be divided arbitrarily [50, 9].

4.6.1 Dataset

For each evaluated method, we indexed a full dump of the English version of Wikipedia (50GB of raw textual data). Except for CDC, all the evaluated methods operate on normalized tokens. The normalization procedure was performed using Apache Lucene [1], a suite of tools that, among others, includes natural language processing methods. We performed the following normalization steps: tokens are normalized to lowercase characters; numeric tokens are filtered out; rare big tokens (larger than 1000 characters) are filtered out; English possessive forms are removed; common stop words are filtered out; and stemming is performed.

4.6.2 Queries

In order to evaluate the effectiveness and efficiency of the different methods, we introduce two search scenarios. The first consists in searching the dataset for *exact sentences*. In this case, the set of queries is built by randomly choosing Wikipedia articles and, within each one, randomly picking

two consecutive sentences. For each consecutive sentence, we then verify their uniqueness in the dataset and eliminate the one that occurs several times. Thus, this set contains 973 queries that are guaranteed to have one relevant result. The second scenario consists in searching the dataset for *cross references*, which typically corresponds to cases of plagiarism detection. We build this set of queries by combining the queries of the first set into 486 queries that are guaranteed to have at most two relevant results. This scenario is important because phrase queries, which accounts for the position of the terms in the document, cannot be used in the case of n-gram indexes, which results in a loss of precision.

4.6.3 Configuration

Method	1-gram	2-gram	3-gram	4-gram	HB	CDC	TB
Unit	Token	Token	Token	Token	Token	Byte	Token
Window size	-	-	-	-	-	23	3
Min size	-	-	-	-	-	41	4
Max size	-	-	-	-	-	248	27
Divisor	-	-	-	-	6	48	3
Backup Divisor	-	-	-	-	-	24	-
Normalization	Yes	Yes	Yes	Yes	Yes	No	Yes
Exact sentence (query type)	Phrase	Phrase	Phrase	Phrase	Boolean	Boolean	Boolean
Cross reference (query type)	Boolean	Boolean	Boolean	Boolean	Boolean	Boolean	Boolean

Table 4.2: Configuration parameters

Table 4.2 lists all of the configuration parameters we used to conduct our experiments. Parameter ”Unit” shows if the method mentioned in the corresponding column operates on tokens or on bytes. The ”Window size”, ”Min size” and ”Max size” parameters depend on the unit mentioned above and specify constraints on the size of the chunks produced. The ”Divisor” and ”Backup divisor” are used in order to detect chunk boundaries, as explained in Section 4.3. In order to compare the segmentation methods in a fair manner, we targeted configuration parameters that would generate segments of approximately 52 bytes on average. To reach this desired average chunk size, we defined the ”Min size”, ”Max size” and ”Divisor” parameters for CDC according to the recommendations described by Eshghi et al. [42].

4.6.4 Environment

We evaluated our indexes with Apache Lucene [1], a state of the art information retrieval library developed in Java. This library provides the necessary components for configuring n-gram indexes.

We implemented some custom components for building HB, CDC and TB indexes. Furthermore, the library contains a good set of benchmarking tools that we used as a basis to measure precision, recall and performances. Regarding our hardware configuration, we ran all our benchmarks using a Dell Power Edge T110 II with an Intel Xeon CPU clocked at 3.50GHz and 16GB of RAM.

4.6.5 Chunk distribution

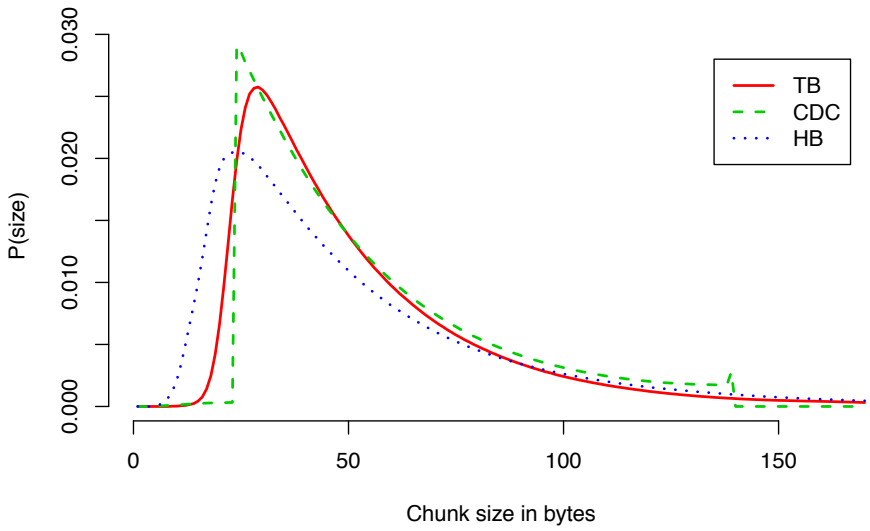


Figure 4.2: Chunk distribution

Figure 4.2 depicts the chunk size distribution for the three non-overlapping segmentation methods we evaluated. As illustrated here, HB produces many small chunks, which results in numerous irrelevant query results (Figure 4.7) and in a relatively poor throughput (Figure 4.5). CDC solves this issue by introducing the *min* and *max* thresholds, but the arbitrary chunk boundaries result in a loss of relevant query results. Therefore, TB reaches the best tradeoff in terms of efficiency and effectiveness by operating on tokens instead of bytes and avoiding arbitrary chunk boundaries.

4.6.6 Index size

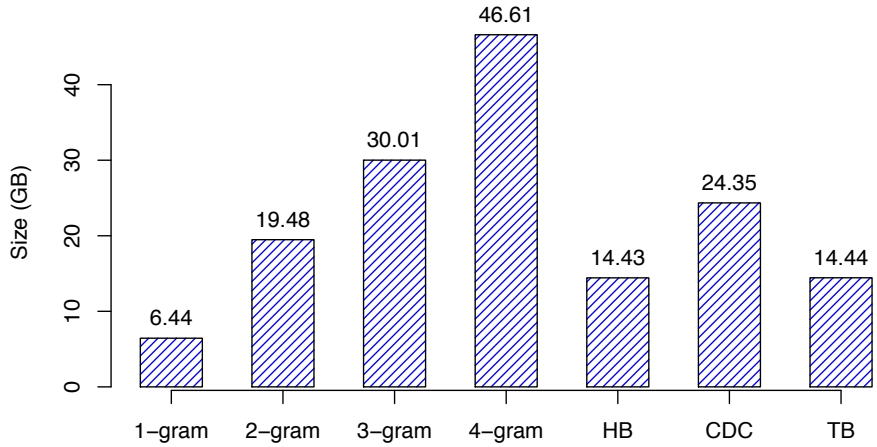


Figure 4.3: Index size

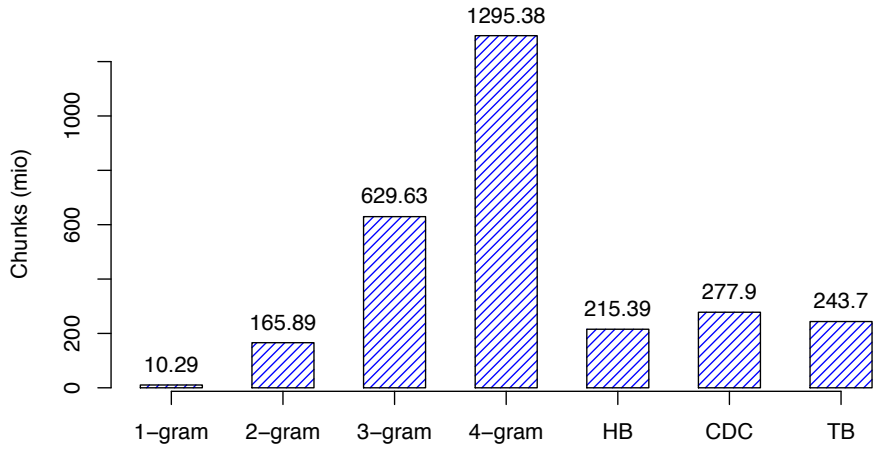


Figure 4.4: Number of dictionary terms

Figure 4.3 and 4.4 depict the index size and the number of dictionary terms for the methods we compared. Here, we notice that, as the size of the n-gram increases, so does the index in terms of size and dictionary terms. From this perspective, HB and TB are more efficient than CDC, which confirms the positive effect of avoiding arbitrary chunk boundaries.

4.6.7 Efficiency

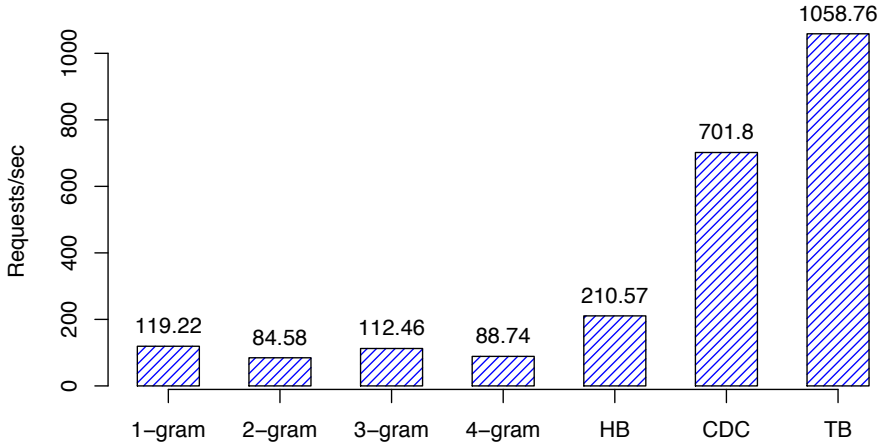


Figure 4.5: Throughput for exact sentence queries

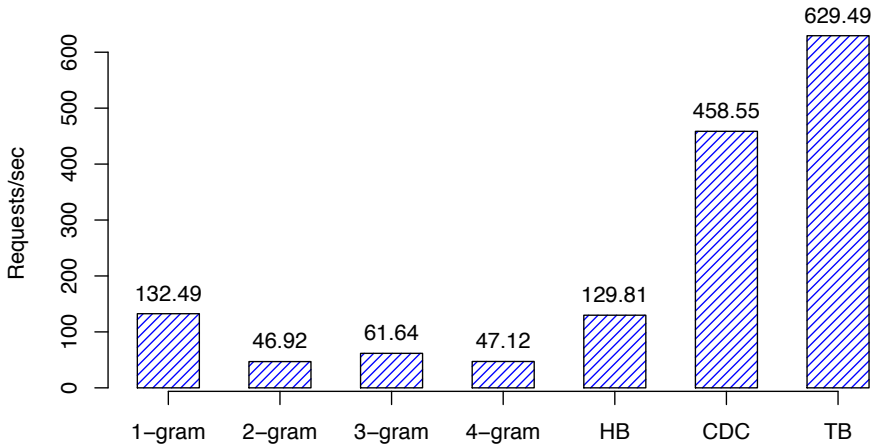


Figure 4.6: Throughput for cross reference queries

Throughput is a measure of efficiency, which corresponds to the number of requests per second that our setup can handle. Figures 4.5 and 4.6 depict the throughput for the methods we compared and for the search scenarios we evaluated, respectively. Regarding this metric, we first notice that n-gram methods display a poor throughput. This mainly comes from the fact that n-gram methods account for the position of the terms in the documents, but also from the fact that the size of the index grows proportionally to the size of the n-gram. In contrast, as the segments extracted by HB, CDC and TB are large and non-overlapping, it is not necessary to account for their positions in the documents.

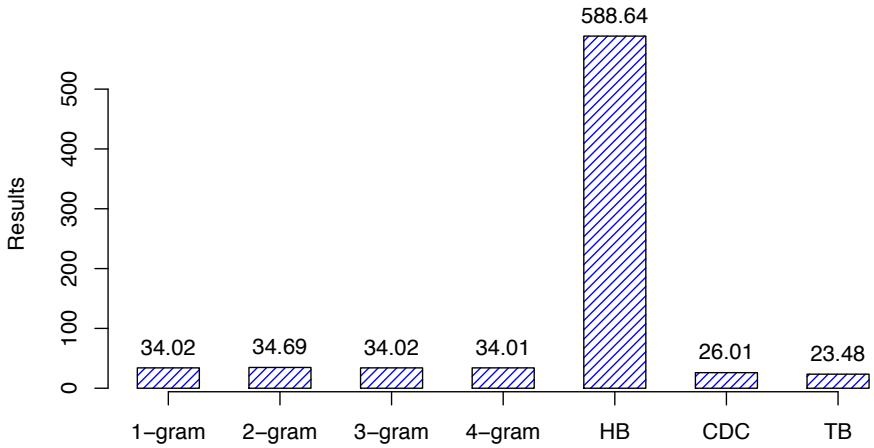


Figure 4.7: Number of results for exact sentence queries

The poor throughput of HB is explained by the great number of irrelevant results (Figure 4.7), whereas the medium throughput of CDC is explained by the index size (Figure 4.3). Therefore, by addressing these two issues, TB is the most efficient method with a maximum throughput of 1058.76 requests per second.

4.6.8 Effectiveness

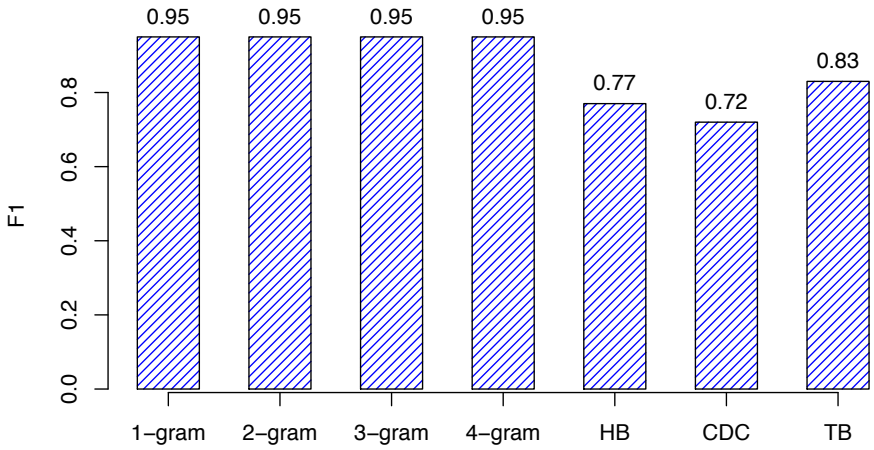


Figure 4.8: F1 score for exact sentence queries

The F_1 score is a measure of effectiveness, which corresponds to the harmonic mean of precision and recall. In Figure 4.8, we notice the cost of adopting HB, CDC or TB over n-grams in terms of effectiveness when searching for exact sentences. On one hand, n-gram methods are the best

in terms of effectiveness but perform poorly in terms of throughput. On the other hand, when comparing TB with n-gram methods, we notice a loss of approximately 10% percent in terms of effectiveness (F_1) for a gain of 800% in term of efficiency (throughput). As highlighted here, in comparison to HB and CDC, TB clearly optimizes this tradeoff.

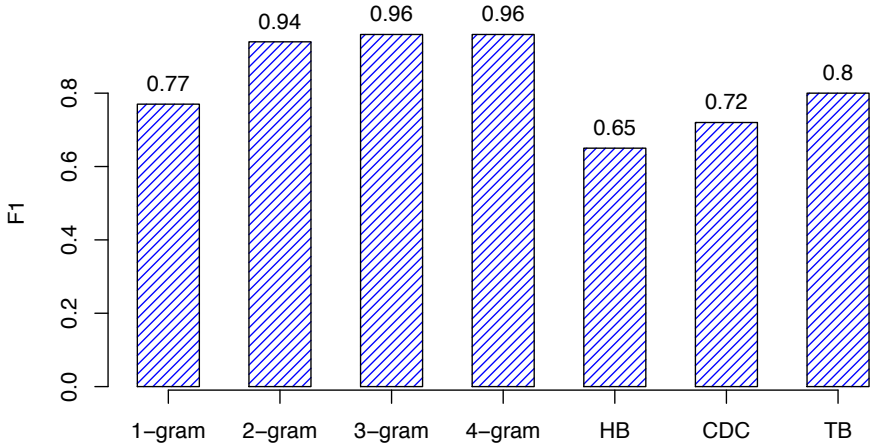


Figure 4.9: F1 score for cross reference queries

Figure 4.9 depicts the F_1 score for the second search scenario that consists in searching for cross references. As mentioned earlier, it is not possible to use phrase queries in this context, and we notice a drop in the F_1 score for the 1-gram method. Consequently, as the other n-gram methods use more storage than TB and have a much lower throughput, the choice regarding this use case is not a matter of tradeoff anymore.

4.7 Toward spatial data

In this section, we empirically show that, by abstracting the notion of token, our approach can be used to create a spatial-index and perform trajectory-based queries. Given a trajectory, the idea is to find all the trajectories in the dataset that share some common sub-sequence with the query. Today, to create search trees, most trajectory indexes use spatio-temporal bounding boxes. For instance, this is the case for the QuadTree, the RTree, and the TBTree [49, 109]. When dealing with trajectories, bounding boxes that span over three dimensions introduce much dead space. The SETI index capitalizes on the special nature of the temporal dimension to address this discrimination issue [22]. This kind of index performs well when the queries involve intervals. Their performances, however, decrease drastically once the queries are based on trajectories. As a result, similarity and distance

measures are used to filter the results and detect which trajectories actually share some common sub-sequence with the query. Computing similarities for a huge result set that includes many outliers introduces an important overhead. As our approach indexes sub-sequences, this filtering step is not necessary.

4.7.1 Dataset

To perform our experiment, we used a set of GPS trajectories gathered by Nokia from 2009 to 2011 [89]. To produce daily trajectories, we grouped the recorded GPS locations per user and per day. The resulting dataset contains 32'144 distinct trajectories located in the area of Lausanne, Switzerland. GPS trajectories are composed of several GPS locations each one having several properties such as longitude, latitude and time. This kind of sequence differs from the kind of data we previously examined, as several dimensions are involved. Hence, using L, we denote the alphabet composed of all the possible longitude/latitude coordinates, and using L^*, we denote the set of all possible sequences of GPS coordinates. In our case, the notion of time is simply expressed by the fact that sequences are ordered.

4.7.2 Normalization

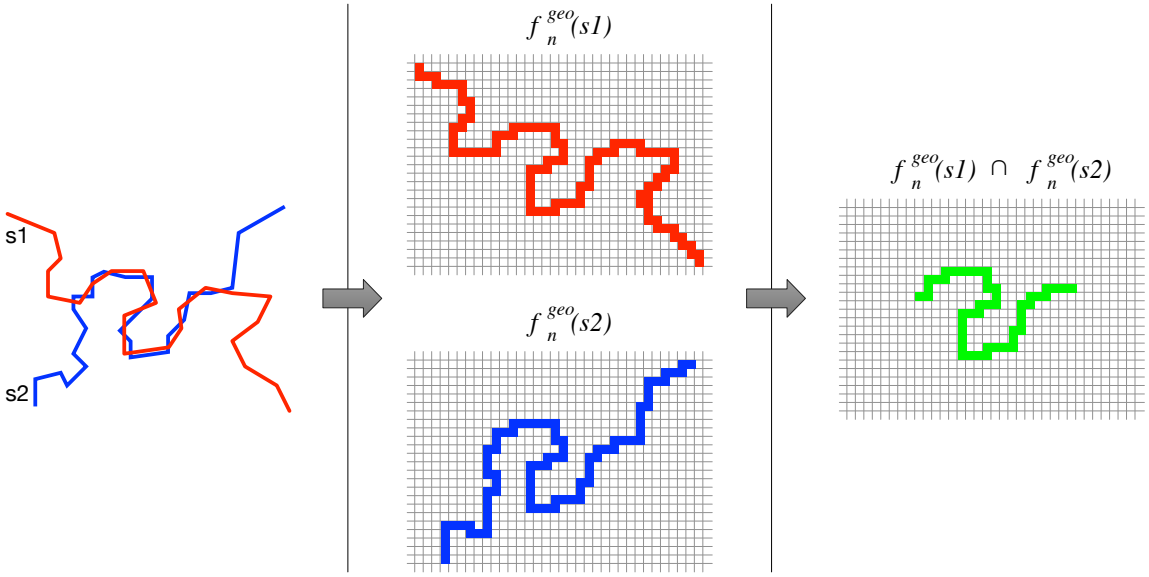


Figure 4.10: Normalization and similarity detection in GPS trajectories

GPS tracking devices are not synchronized and might showcase different sampling rates. Therefore, two persons following the same path will end up with different sequences of GPS locations. This issue can be solved by normalizing the data. In order to find sub-sequences in the trajectories, we normalized the coordinates by using a hash function, called GeoHash, which subdivides the longitude/latitude coordinate system into cells [105]. In our case, GeoHash maps any longitude/latitude coordinates into cells of approximately 150m by 150m. The center of the cell is then used as the normalized coordinate. Such a normalization function could be defined as $f_n^{geo} : L^* \rightarrow L^*_{geo}$ with $L_{geo} \subseteq L$. Figure 4.10 illustrates the fact that, after normalization, trajectories converge toward something more identical. The sequences that result from normalizing the coordinates can directly be consumed by TB. Our configuration produces chunks that have an average length of 10 normalized coordinates that correspond to an average distance of 1,5 kilometers by sub-sequences.

4.7.3 Preliminary results

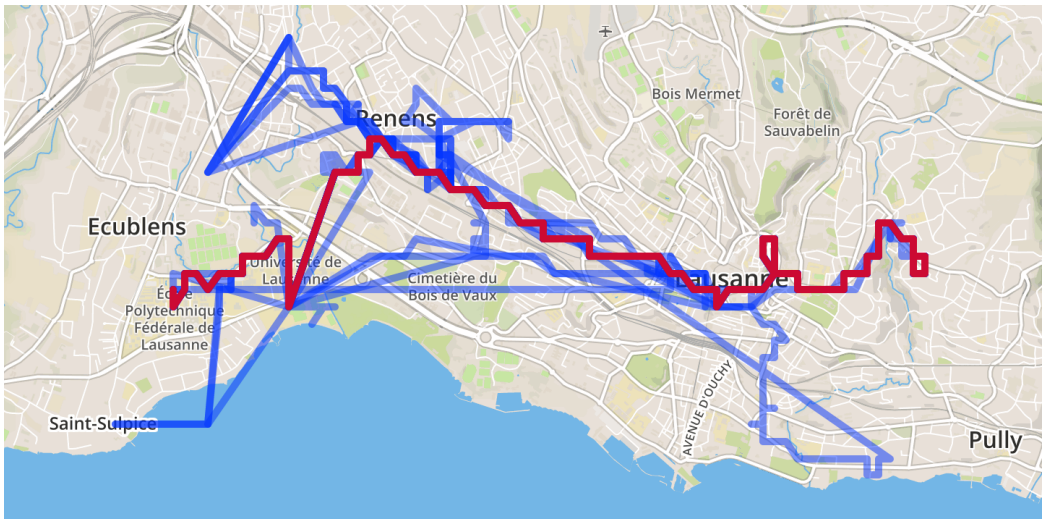


Figure 4.11: A trajectory query (red) and some results that share sub-sequences (blue)

We compared the resulting index with two common spatial-indexes, namely the Quad Tree [49] and the Sort-Tile-Recursive Tree [112] implemented in JTS [2]. To perform our experiment, we selected a trajectory in the dataset and queried the three indexes for similar trajectories. As the dataset is very dense, the Quad Tree returned 22'304 results and the Sort-Tile-Recursive Tree discriminated slightly better with 18'070 results, that, in both cases, represents a great number of outliers that confirms the discrimination problem. As our inverted index has a dictionary made of unique trajectory sub-sequences, it preserves information regarding the similarity of the trajectories. As a result, the query

returned the 36 matches depicted in Figure 4.11; they are guaranteed to share some sub-sequences of more or less 1,5 kilometers with the query.

4.8 Related Work

The idea of computing all the overlapping sub-sequences of terms in a document was first introduced by Mamber et al. in [96]. Brin et al. also describe some methods for copy detection that include n-gram and hash-based segmentation [18]. The term shingle, introduced by Broder et al. [21], is often used as a substitute for n-gram. Data deduplication is mainly used in the context of data storage and data synchronization [102, 110, 100]. It usually relies on CDC [42, 11] for identifying identical sub-sequences in data. In order to avoid redundancies, chunks are identified by their hash sums and stored in content addressable storage. Recipes consist of lists of chunk hash sums that are used to reconstruct the original data. Muthitacharoen et al. [102] improve CDC by introducing maximal and minimal chunk sizes that positively affect the compression ratio. Eshghi and Tang [42] introduce a backup divisor that enables avoiding arbitrary cuts when the maximal threshold is reached. More recently, to achieve even better compression rates, other interesting methods, such as bimodal content-defined chunking [84] and frequency-based chunking [94], were proposed. Two distinct documents whose recipes share common chunks are related to each other with a high probability. This assumption is used by Forman et al. [50] to identify near-duplicates in very large collections of manuals and technical documents. Bhagwat et al. [9] generalize the idea of using recipes to build inverted indexes for near-duplicate search. They create an inverted index where the dictionary terms correspond to the hash sums of chunks and the postings correspond to document identifiers.

Our approach, in contrast, does not operate on raw data and can be used to agnostically index different kinds of data. Similarity detection in textual data has been studied for several decades and many techniques have been investigated. For example, techniques based on local maxima and minima, sometimes referred to as winnowing, are used to filter hash values [115, 10]. In our future work, we will explore how techniques commonly used with textual datasets can be leveraged with different types of data.

4.9 Conclusion & Future Work

In this paper, we have introduced token-based chunking, a generic approach for finding sub-sequences in data. We have studied its characteristics in terms of storage requirements, throughput, precision

and recall and we have demonstrated that it performs better than its traditional counterparts at trading effectiveness for efficiency. We have shown that, by operating on tokens, this technique can be used agnostically on two types of data. We have introduced a framework that helps in identifying the extent to which data should be normalized regardless of its type. In addition, we also have empirically demonstrated that token-based chunking can efficiently index GPS trajectories. Finally, we have shown that the resulting index has discrimination characteristics better than traditional spatial-indexing approaches. To our knowledge, the usage of a segmentation methods inspired by data deduplication in the context of spatial-indexes has not been explored before and our preliminary results are promising. This opens exiting new research avenues for trajectory and sub-trajectory mining and we plan to investigate the properties of this novel kind of spatial-index in the future.

Chapter 5

Geodabs: Trajectory Indexing Meets Fingerprinting at Scale

Bertil Chapuis and Benoît Garbinato. Geodabs: Trajectory indexing meets fingerprinting at scale. In *38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018

Abstract

Finding trajectories and discovering motifs that are similar in large datasets is a central problem for a wide range of applications. Solutions addressing this problem usually rely on spatial indexing and on the computation of a similarity measure in polynomial time. Although effective in the context of sparse trajectory datasets, this approach is too expensive in the context of dense datasets, where many trajectories potentially match with a given query. In this paper, we apply fingerprinting, a copy-detection mechanism used in the context of textual data, to trajectories. To this end, we fingerprint trajectories with *geodabs*, a construction based on geohash aimed at trajectory fingerprinting. We demonstrate that by relying on the properties of a space filling curve geodabs can be used to build sharded inverted indexes. We show how normalization affects precision and recall, two key measures in information retrieval. We then demonstrate that the probabilistic nature of fingerprinting has a marginal effect on the quality of the results. Finally, we evaluate our method in terms of performances and show that, in contrast with existing methods, it is not affected by the density of the trajectory dataset and that it can be efficiently distributed.

5.1 Introduction

The booming trend of ubiquitous computing is massively affecting the volume of data we produce today, in particular via the location traces, or *trajectories*, our smartphones generate. Such trajectories consist of sequences of locations produced by mobile users via their GPS-capable devices. In this paper, we address two key problems associated with dense trajectory datasets: *finding similar trajectories* and *discovering common motifs in trajectories*. These problems are indeed at the heart of many location-based application scenarios, such as car sharing, traffic forecasting, public-transport optimization, etc. By *dense* trajectory dataset, we mean one containing many (partially) overlapping trajectories. Consider, for instance, a city like London, congested with roads and streets: the trajectories associated with people traveling through it every day have a high probability of overlapping, at least partially. This is the type of trajectory data set we consider in this paper.

A common approach to solving these problems consists in splitting the solution into the following two steps:

1. *Select candidate trajectories* by using a *spatial index*
2. *Compare these trajectories* by using a *distance measure*

More precisely, Step 1 consists in querying a spatial index, e.g., a quadtree [49], an r-tree [64], a tb-tree [109], a seti-tree [23] or a k-d tree [8], in order to select candidate trajectories that are similar or that contain common motifs. Such space-partitioning data structures are typically queried with bounding intervals and sometimes a direction. Yet they have a major drawback: as their bounding strategy are coarse grained, their ability to discriminate long trajectories is not very effective. This results in many irrelevant trajectories being selected.

Step 2 then consists in using a distance measure, such as the Discrete Fréchet Distance (DFD) [41] or the Dynamic Time Warping distance (DTW) [129], in order to further discriminate the candidate trajectories selected in Step 1. DFD and DTW give good qualitative results and many systems have adopted them to measure the distance between trajectories. However, computing DFD or DTW for a pair of trajectories of cumulated length n has a complexity of $O(n^2)$. Furthermore, discovering similar motifs in a pair of trajectories requires computing DFD for n^4 pairs of sub-trajectories [119].

In summary, when faced with a dense set of trajectories, traditional spatial-indexing structures tend to select many irrelevant trajectories, upon which a costly distance measure such as DFD or DTW must then be computed. As a consequence, this combination of techniques results in serious performance issues when used on dense datasets.

5.1.1 Fingerprinting to the Rescue

We argue that the similarity between trajectories and textual data has not been fully exploited. A text can be seen as a sequence of words, and a trajectory can be seen as a sequence of points. It is then known that slight variations in word form, e.g., singular vs. plural, conjugation, etc., must be normalized to compare similar but not strictly identical texts. This is necessary, for instance, to detect plagiarism. Similarly, minor variations in the location accuracy and sampling rate, which are known to happen when using GPS devices, can compromise the detection of similar trajectories. Hence, by discretizing trajectories, e.g., by using geohashing [105], the negative effect of such minor variations can be mitigated.

A first attempt to exploit this similarity can be found in some geographical information systems that rely on geohashing to create inverted indexes of landmarks. For example, an open-source search engine called Elastic and adopted by foursquare, relies on this approach.[1] Google even conceived an alternative to geohash called S2 that provides the additional guarantee that the surfaces covered by hashes have uniform areas.[2] More recently, geohashing has been used for sub-sampling and clustering location traces [118, 39]. As of today, however, no research exploits the similarity between trajectories and textual data in terms of their sequentiality, i.e., the fact that sequences of locations are similar to sequence of words.

Extending the analogy between textual data and trajectories is our main contribution in this paper. More precisely, in the context of textual data, word indexing is known to be ineffective in detecting similarities between large portions of a text, even more so between complete documents. This is where fingerprinting comes to the rescue, by computing hashes on groups of contiguous words and by using the number of common fingerprints between two documents as a distance measure. An inverted-index made of fingerprints that point to lists of document identifiers can then be used to efficiently retrieve documents that share some of their content. By analogy, fingerprinting can be used for finding similar trajectories and discovering similar motifs. Intuitively, fingerprinting captures the temporal dimension of trajectories by taking into account the ordering of their points. To our knowledge, the possibilities offered by this observation have not yet been explored.

5.1.2 Contribution and Roadmap

We introduce geodabs, a special kind of fingerprint that can be used for indexing and discovering similarities in dense trajectory datasets. Geodabs are extracted from trajectories with a fingerprint-

[1] https://elastic.co
[2] https://s2geometry.io

ing algorithm called winnowing [115]. Geodabs combine hashing and geohashing to achieve two key properties. First, hashing addresses the discrimination issue associated with regular spatial indexation techniques. Second, geohashing enables us to distribute the index accross several nodes in a balanced fashion. As a result, geodabs can be used to create effective and scalable trajectory indexes.

The remainder of the paper is organized as follows. We formally introduce the problems addressed in this work, together with some basic definitions, in Section 5.2. Then, in Section 5.3, we provide the background required to understand our approach and we discuss related work. In Section 5.4, we describe geodabs, our fingerprinting based-solution. In Section 5.5, we shows how normalization affects two key measures in information retrieval, namely *precision* and *recall*. Finally, in Section 5.6, we evaluates geodabs both in terms of efficiency and effectiveness.

5.2 Trajectory-based querying

In this section, we introduce some basics definitions and formally state the problem addressed in this paper.

5.2.1 Moving Objects, Trajectories and Distances

Every object located on earth has a real position that can be expressed with a latitude-longitude point $p = (\varphi, \lambda)$. The real position p of a moving object at time t can be expressed with a continuous-time function $P(t) = p$. In practice, the points forming a trajectory are obtained with some GPS-tracking device, which reduces the continuous-time function P to a discrete trajectory S with a certain degree of accuracy. Hence, formally, a trajectory is modeled as a sequence of points $S = \langle s_1, ..., s_n \rangle$. The length of a trajectory is denoted $length(S)$. A motif (sub-trajectory) of S is denoted $\bar{S}$.

Several methods enable us to compute the distance between two trajectories. Depending on the method, the distance has different scales and meanings. Here, we generalise this idea with the distance function $\delta(S_i, S_j) = d$, with $d \in \mathbb{R}_0^+$. The smaller the distance between a pair of trajectories, the greater their similarity is.

5.2.2 Finding Similar Trajectories and Motifs

Here, we address the problems of *finding similar trajectories* and of *discovering common motifs in trajectories*. As we target dense trajectory datasets, we express these problems in terms of ranked retrieval, i.e., many trajectories are expected to match a given query. In addition, ranked retrieval also implies sorting the matching trajectories according to some criterion, in order to place those most relevant early in the result list. With this in mind, we can now formally define the two problems addressed in this paper.

Finding similar trajectories

Given a trajectory dataset $D = \{S_1, ..., S_n\}$, a trajectory $S_q \notin D$, a distance function δ and a distance Δ_{max}, the problem consists in returning an ordered set $R \subseteq D$ where $S \in R \Rightarrow \delta(S_q, S) \leq \Delta_{max}$. The ordering in R is then defined as follows: $\forall S_i \in R$ and $\forall S_j \in R$ with $i < j$, we have that $\delta(S_q, S_i) \leq \delta(S_q, S_j)$. That is, trajectories in R are at a maximum distance Δ_{max} from trajectory query S_q and they are ordered by their distance with respect to S_q.

Discovering common motifs in trajectories

Given two trajectories S_i and S_j, a distance function δ and a length l, the problem consists in returning a pair of motifs $(\bar{S}_i, \bar{S}_j)$ such that $length(\bar{S}_i) = length(\bar{S}_j) = l \wedge \nexists(\bar{S'}_i, \bar{S'}_j)$ for which $\delta(\bar{S'}_i, \bar{S'}_j) < \delta(\bar{S}_i, \bar{S}_j)$. That is, among all the pairs of motifs of S_i and S_j of length l, $(\bar{S}_i, \bar{S}_j)$ is the one with the smallest distance between them.

5.3 Background and related work

5.3.1 Information Retrieval

Boolean Retrieval

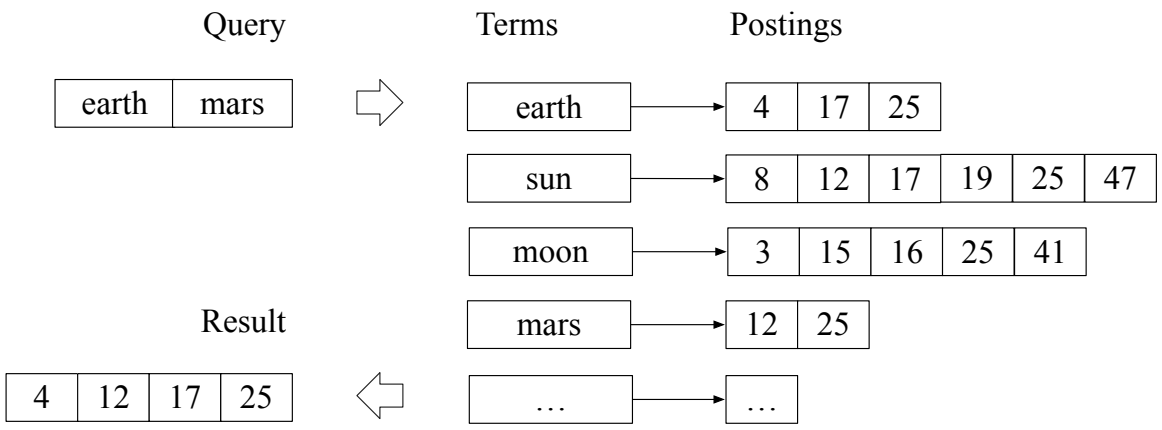


Figure 5.1: Building an Inverted Index

As highlighted in Figure 5.1, in its simplest form, an inverted index is usually composed of terms that point to collections of document identifiers called postings lists [97]. Boolean queries can then be used to retrieve all the documestnts that contain a set of words. Optionally, a posting list can also contain the position of the term in the document. This positional information can then be used to search for sub-sequences in documents. However, when searching for long sub-sequences of terms, this approach showcases poor performances. In the context of trajectory indexing, we replace the terms of the inverted index with features, called geodabs, extracted from trajectories.

Ranked Retrieval

In ranked retrieval, many records match with the query specified by the user, and it is common to rank the results according to a similarity measure. Therefore, the user can begin by considering the most relevant results and then decide to retrieve the remaining ones if necessary. In the context of textutal data, the union and the intersection of two sets of words F and G can be used to derive relevant similarity measures. For example, the Jaccard coefficient $J(F,G)$ is commonly used to gauge the similarity between texts and rank results [97]. Interestingly, the Jaccard distance $d_J(F,G)$ expressed in Equation 5.1 is complementary to the Jaccard coefficient and is proven to obey the

triangular inequality [82]. Therefore, this distance can be used in conjunction with an index of pre-computed distances to efficiently prune candidates. In our context, we use the Jaccard distance to implement function $\delta(S_q, S_j)$ and rank the trajectories retrieved from the inverted index.

$$d_J(F,G) = 1 - J(F,G) = 1 - \frac{|F \cap G|}{|F \cup G|} \tag{5.1}$$

Normalization

It is worth noting that, in many cases, terms can be different but convey similar semantics or meaning. In general, the process of mitigating these differences is referred to as normalization [97]. For example, a common normalization technique consists in using equivalence classes for synonyms. In the context of trajectory indexing, we refer to normalization as a function $N(S) = S'$, where S and S' are sequences of points.

Sharding

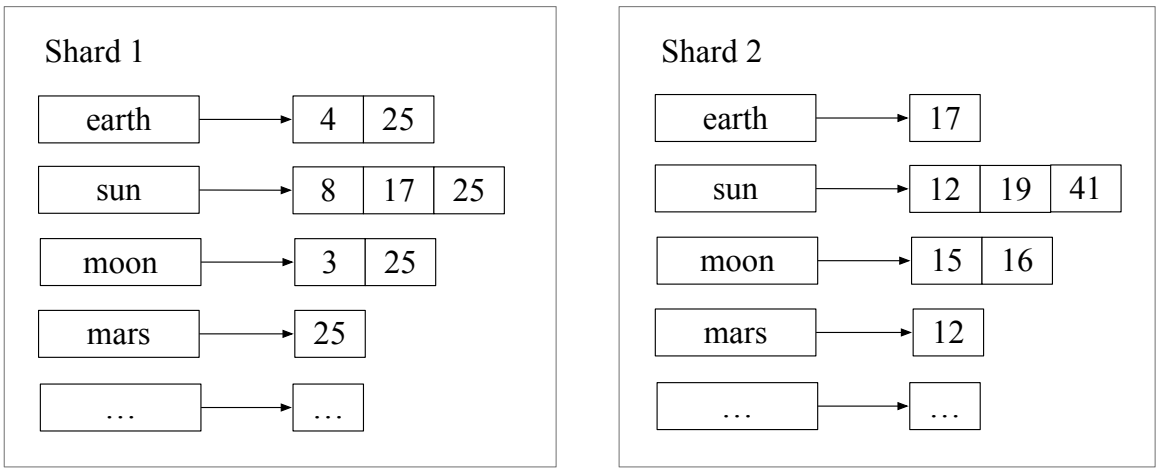


Figure 5.2: Sharding an Inverted Index

When an index becomes very large, it might not fit on a single computer anymore. As illustrated in Figure 5.2, sharding the index across the nodes of a cluster becomes necessary. Here, the idea is to route documents to specific shards in order to spread the load throughout the cluster. At query time, all the shards might need to process a query to compute the result. Therefore, given the terms

specified in a query, a good sharding strategy tries to minimize the number of shards that need to be contacted. Our solution effectively addresses this issue.

5.3.2 Fingerprinting

As mentioned in Section 5.3.1, searching for long sub-sequences in textual data by using words and positional information is not very efficient. In practice, an inverted-index aimed at searching for sub-sequences is usually populated with a different class of terms, referred to as fingerprints [20, 96, 67, 18]. Fingerprints usually correspond to a sub-set of the hash sums obtained by hashing the n-grams of a document [115]. A n-gram is a sequence of n contiguous items, i.e., n words in the context of textual data. A fingerprint usually corresponds to the hash sum $h \in \mathbb{R}$ obtained by hashing a n-gram. As the number of n-grams for a given text can be very large, a common practice consists in retaining the subset of fingerprints that satisfy the condition $h \bmod p = 0$, where p is a fixed sampling constant. The extracted fingerprint can then be used as terms in the inverted index. Furthermore, given two sets of fingerprints, their similarity can easily be derived by the Jaccard coefficient. In our context, we refer to fingerprinting as a function $W(S) = F$, where S is a trajectory and F is an ordered set of fingerprints.

5.3.3 Geohashing

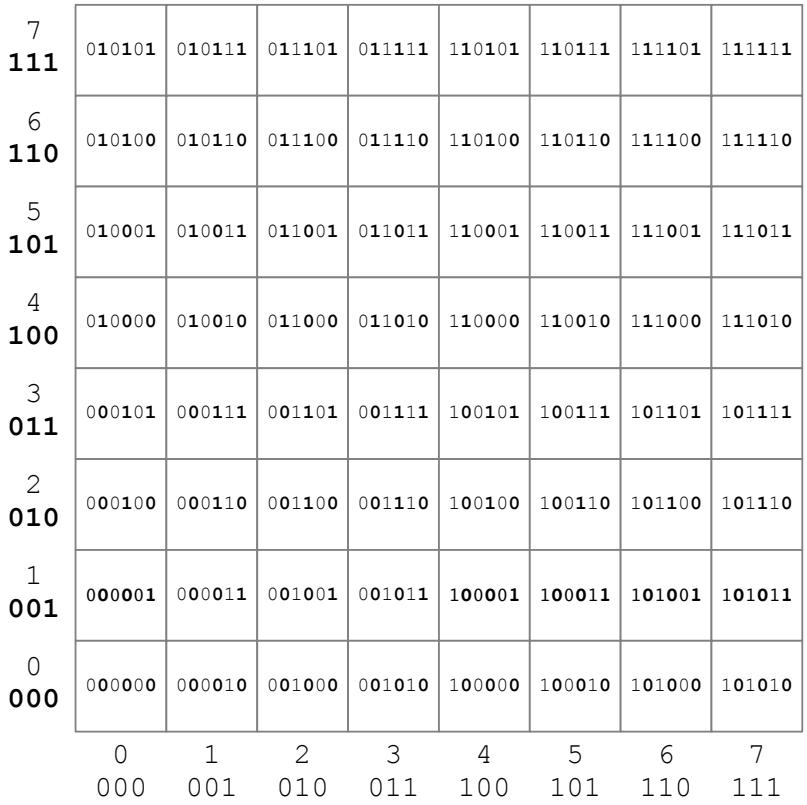


Figure 5.3: Geohash

A function that produces geohashes maps a point p to a sequence of bits b that repeatedly bisect space up to a desired depth d that defines the precision of the geohash [105]. In the case of a latitude/longitude space, the first subdivision usually occurs on the longitude axis, the second on the latitude axis and the process is repeated up to depth d. Figure 5.3 illustrates this subdivision for a depth $d = 6$, where two interleaved sequences of three bits are respectively dedicated to the subdivision of the longitude and latitude axes. Every geohash covers a delimited area on earth and, given a set of points $\{p_1, p_2, ..., p_n\}$, it is relatively easy to find the highest precision geohash that overlaps with the whole set. Hereafter, we formally refer to such an overlapping geohash with the function $geohash(\{p_1, p_2, ..., p_n\}) = b$.

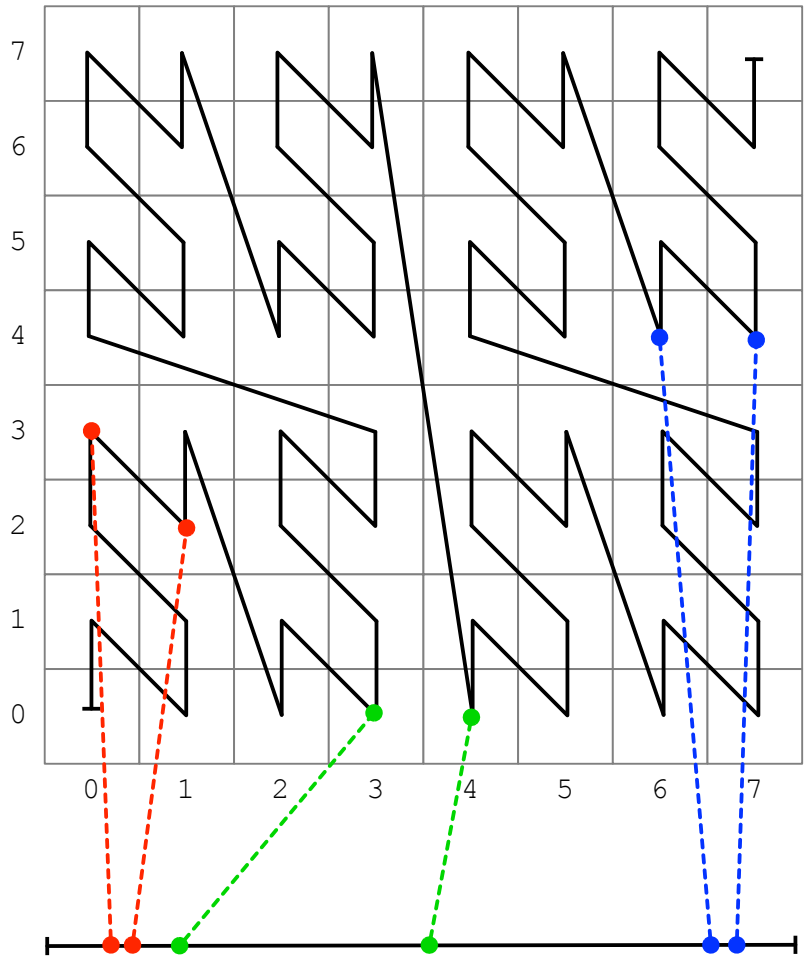


Figure 5.4: Space-filling Curve

As highlighted in Figure 5.4, the ordered list of hashes obtained by subdividing the space with a geohash function can be represented as a z-order space-filling curve. Interestingly, when two points are close to each other on a space-filling curve, then they are close to each other in the latitude/longitude space. However, the reverse is not necessarily true since: two points near each other in the latitude/longitude space might be far from each other on the space-filling curve.

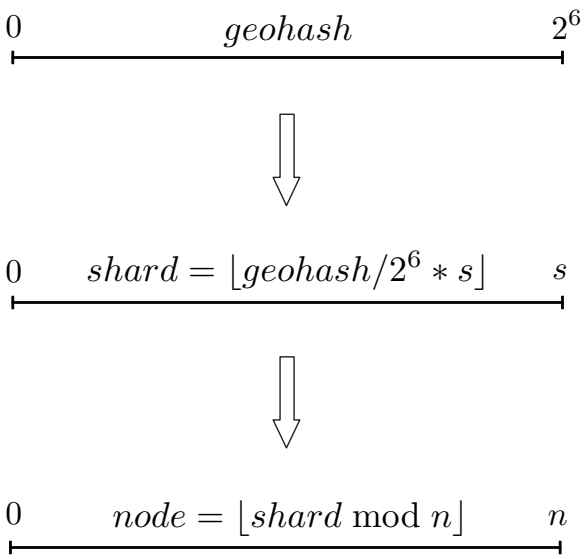


Figure 5.5: Sharding

Then, as illustrated in Figure 5.5, the space-filling curve can be used to shard a spatial index across the nodes of a cluster. Two steps characterize the sharding strategy pictured here. First, geohashes are mapped to shards in a locality preserving way, i.e., geohashes near each other on the space-filling curve are placed on the same shard. Second, the shards are mapped to nodes with a modulo operation that breaks locality to improve the balance of the index across the nodes.

5.4 Fingerprinting with Geodabs

We now introduce geodabs, a construction that combines geohashing and hashing to fingerprint trajectories. Our motivation for introducing geodabs is based on two key reasons. First, as illustrated in Figures 5.4, the correspondence between the distance on the space-filling curve and the latitude/longitude space can be used as a means to shard and distribute the index. Sharding with geohash on the space-filling curve guarantees that the locality of the index will be preserved and that a minimal number of shards will be contacted to answer a query. Second, the fingerprints of a trajectory should capture the notion of order present in trajectories. Given a sequence of points, geohashes only capture the area that overlap with these points. Therefore, we use regular hashing to address this issue.

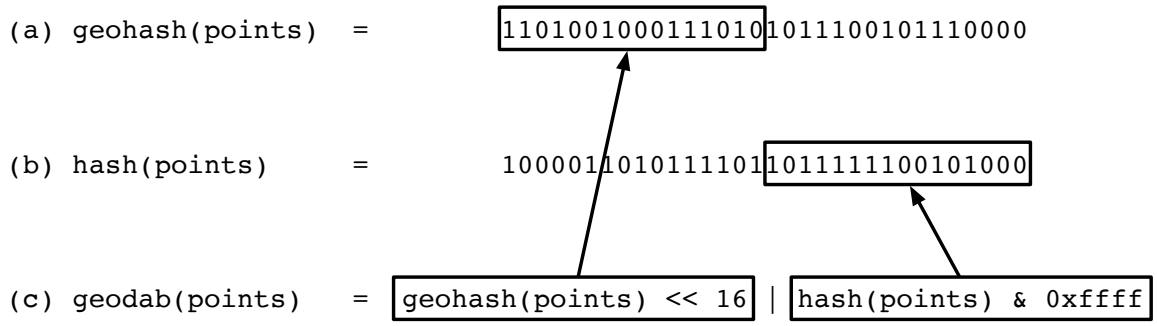


Figure 5.6: Construction of a geodab

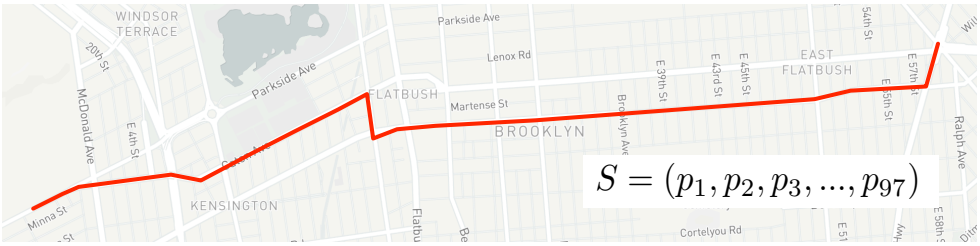
Figure 5.6 illustrates how geodabs are constructed. (a) Given a sequence of points, we first compute the geohash that overlaps with the whole sequence. The geohash acts as a prefix that naturally distributes the geodabs on the space-filling curve according to the location of the points. (b) We compute a hash that is sensitive to the ordering of the points. This hash acts as a suffix for the geodab that discriminates among sequences of points, according to their path and their ordering. (c) We merge the two hashes, here encoded on 32 bits, using bitwise operators. The length of the prefix can easily be adjusted, depending on the desired space partitioning.

5.4.1 Trajectory Fingerprinting and Indexing

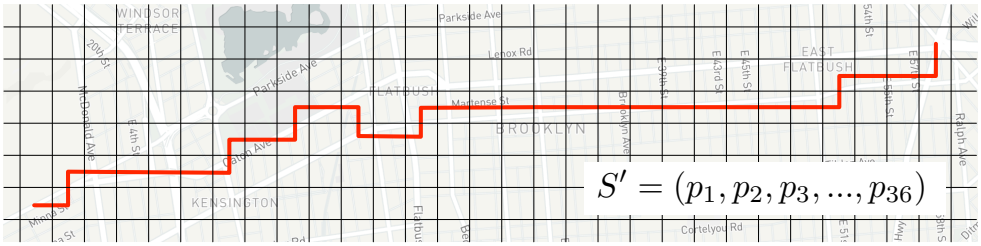
When comparing sequences, finding similarities by sampling fingerprints is probabilistic and does not guarantee the detection of all the similarities. Winnowing is a special fingerprinting algorithm that address this problem and comes with additional guarantees [115]. First, it ensures that matches shorter than a user defined lower-bound k are considered as noise. To satisfy this guarantee, winnowing only considers hashes of k-grams. Second, it ensures that at least one k-gram is detected in any common sequence of size greater or equal to an upper-bound $t \geq k$. To satisfy this guarantee, the algorithm defines a window of size $w = t - k + 1$ that slides over the sequence of k-gram hashes. For each window, the algorithm selects the minimum hash value or the right-most minimum hash value if the same hash appears more than once in the window. As the dataset densifies, the upper threshold can be used to reduce the number of fingerprints extracted from queries in order to set the efficiency/effectiveness tradeoff.

Figure 5.7 shows how the steps described in [115] can be adapted to extract fingerprints from trajectories. (a) The raw trajectories can showcase different sampling rates, (b) it is therefore necessary to normalize them. In Section 5.5, we discuss how normalization can be used to make trajectories

(a) Raw trajectory:



(b) Normalized trajectory:



(c) Sequence of *k*-grams based on the normalized trajectory (k=5):

$$((p_1, p_2, p_3, p_4, p_5), (p_2, p_3, p_4, p_5, p_6), (p_3, p_4, p_5, p_6, p_7), ...$$
$$..., (p_{32}, p_{33}, p_{34}, p_{35}, p_{36}))$$

(d) Sequence of geodabs derived from the *k*-grams:

$$(h_1, h_2, h_3, ..., h_{32})$$

(e) Windows of geodabs (w=*4) with minimal values in bold*:

$$((h_1, h_2, \boldsymbol{h_3}, h_4), (h_2, \boldsymbol{h_3}, h_4, h_5), (h_3, h_4, h_5, \boldsymbol{h_6}), ...$$
$$..., (\boldsymbol{h_{29}}, h_{30}, h_{31}, h_{32}))$$

(f) Hypothetical sequence of geodabs selected by winnowing:

$$(h_3, h_6, h_{10}, h_{12}, h_{16}, h_{19}, h_{22}, h_{26}, h_{29})$$

Figure 5.7: Trajectory Winnowing

Algorithm 6 Geodabs extraction by winnowing

Input: trajectory S, lower-bound t, upper-bound k
Output: Geodabs G
$C \leftarrow (\varnothing)$ ▷ Sequence of candidate hashes
for $i \leftarrow 1$ to $|S| - k$ **do** ▷ Iterate over k-grams
    $g \leftarrow geodab(S_{i,i+k})$ ▷ Compute k-gram geodabs
    $C \leftarrow C \parallel (g)$ ▷ Add geodabs to candidates
$G \leftarrow \{\varnothing\}$ ▷ Set of winnowed geodabs
$w \leftarrow t - k + 1$ ▷ Window size
for $i \leftarrow 1$ to $|C| - w$ **do** ▷ Iterate over windows
    $m \leftarrow i$ ▷ Initialize minimum geodabs index
    for $j \leftarrow i + 1$ to $i + w$ **do** ▷ Iterate over the window
        if $C_j <= C_m$ **then** ▷ Select right most minimum
            $m = j$ ▷ Set new minimum
    $G \leftarrow G \cup \{C_m\}$ ▷ Add minimum to geodabs
return G

converge to similar sequences. (c) The sequence of k-grams can then be computed and (d) the corresponding hashes can be derived. (e) The sliding window of size w can then be used to (f) select the hashes that constitute the fingerprints of the trajectory. The resulting fingerprints can then be used as terms in an inverted index, where posting lists are filled with trajectory identifiers.

Algorithm 6 shows in more detail how winnowing can be implemented for fingerprinting trajectories. An optimised version of this algorithm relies on circular buffers and rolling hash functions for iterating over k-grams of points and windows of hashes. In contrast with normalized documents that often contain thousands of words, normalized trajectories are relatively short sequences of points. As we did not notice a significant performance gain, we dropped this optimization.

In our implementation, we use roaring bitmaps to represent the sets of fingerprints F [92]. Any set made of integers can be represented with bitmaps. Given two bitmaps, their intersection, union, or difference can be computed very efficiently with bitwise operations. Roaring bitmaps are fast memory-efficient bitmaps that outperform most existing techniques. The fingerprints generated by the trajectory-winnowing algorithm are used as terms in the inverted index. Each entry of the postings lists contain a reference to the raw trajectory and a reference to the trajectory bitmap. As a result, when querying the index, the bitmap of the query can be compared to the bitmap of the result in order to gauge their similarity.

5.5 Trajectory Normalization

In this section, we show that a sound normalization procedure can be applied to trajectories. In the context of textual data, normalization relies on semantic rules and linguistic techniques to find equivalence classes for terms. For example, such techniques include case-folding, true-casting, stemming, lemmatization, and the identification of stop-words. These techniques remove superficial differences in textual data. Normalization in the context of trajectories differs, but the notion of equivalence classes remains the same, as highly similar trajectories should converge toward similar sequences of points. In this section, we introduce two normalization methods and show the extent to which trajectory data should be normalized.

5.5.1 Normalizing with Geohash

A simple normalization technique consists in mapping the points of a trajectory to a sequence of geohashes at a constant depth d. The resulting geohashes can then be cleaned from consecutive duplicates and converted back to sequences of points. This approach is very lightweight and removes most irrelevant differences in trajectories. If two trajectories follow the edges of one or several geohashes, the resulting sequences of normalized points could be different. However, as it will be demonstrated in Section 5.6, it does not really affect the quality of the results.

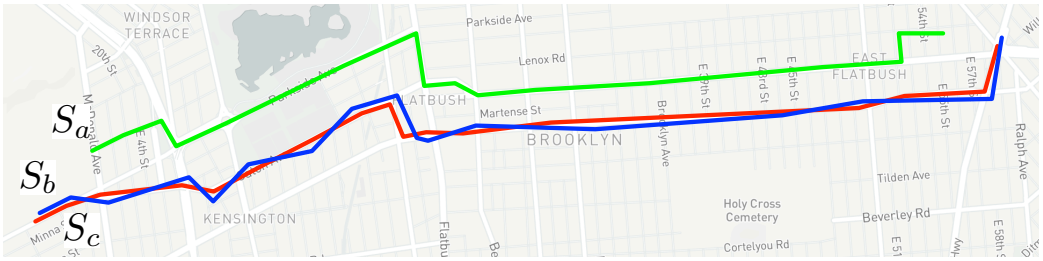
5.5.2 Normalizing with Map Matching

Another normalization technique consists in mapping trajectories to an existing road network [104]. This approach, called map matching, can give very good results, especially if the moving entity at the origin of a trajectory is known to be constrained to a road network. Most recent map-matching algorithms rely on the Viterbi algorithm to compute a match [61]. For each point of a trajectory, the idea is to first retrieve a set of matching nodes on a road network within a certain radius. The Viterbi algorithm then computes the most probable sequence of nodes on the road network [104]. This approach is computationally costly, but this price is paid only at the creation of the index. When searching for similarities, the query trajectory has to be normalized in a similar way.

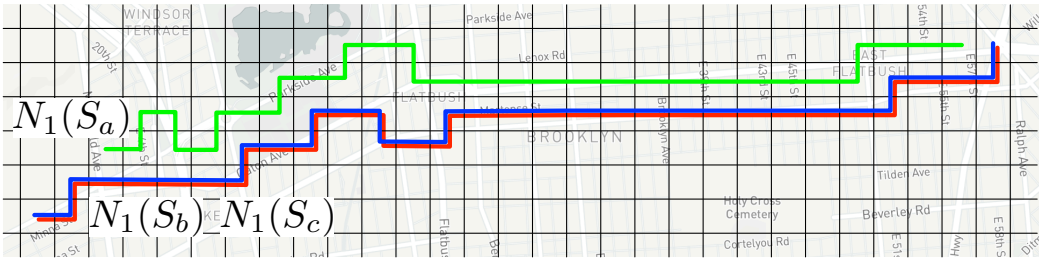
5.5.3 Extent of the Normalization

In the context of textual data, normalization usually relies on simple intuitions. Unfortunately, when dealing with trajectories, it is difficult to rely on the same intuitions. In this section, we show how

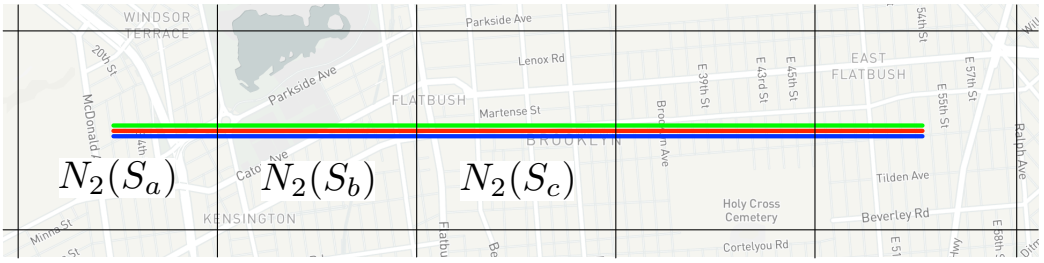
(a) The raw trajectories



(b) Normalization on a grid of geohashes with N_1



(c) Normalization on a grid of geohashes with N_2



(d) Normalization on a road network with N_3

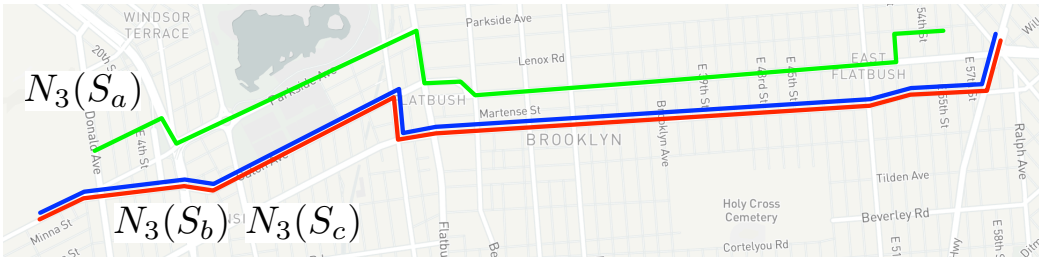


Figure 5.8: Trajectory Normalization

the measures introduced in [28] can be applied to trajectory normalization.

As stated previously, with normalization, trajectories should converge toward similar sequences of points. Therefore, a good normalization function n applied to a pair of similar trajectories S_a and S_b should satisfy the property $J(W(N(S_a)),W(N(S_b))) > J(W(S_a),W(S_b))$. Focusing solely on this property, however, leads to aggressive normalization functions that make every trajectories converge to the same output.

In order to address this issue, we start by recalling two key effectiveness measures in information retrieval. First, precision measures the fraction of selected items that are relevant. More formally, given the number of relevant items retrieved tp (true positive) and the number of irrelevant items retrieved fp (false positive), $precision = tp/(tp+fp)$. Second, recall measures to the fraction of relevant items that are selected. In other words, given the number of relevant items retrieved tp and the number of relevant items that have not been retrieved fn (false negative), $recall = tp/(tp+fn)$.

In Figure 5.8, we illustrate the effect of two hypothetical normalization functions N_1 and N_2 on precision and recall. We first introduce a dataset that contains the raw trajectories S_a, S_b. We also assume a query trajectory S_q, which has one relevant result S_b in the dataset. Figure 5.8a depicts an inverted index built with the raw trajectories and queried with S_q. Without normalization, this index would return no relevant result Both precision and recall would therefore be equal to 0. Figure 5.8b depicts an inverted index built with the normalization function N_1 and queried with S_q. This index would return one relevant results. Thus, precision would be $1/(1+0)$ and recall $1/(1+0)$. Figure 5.8c depicts an inverted index built with a more aggressive normalization function N_2. In this case, S_q would return two results, precision would drop to $1/(1+1)$ and recall would remain stable at $1/(1+0)$. As long as all the true positives are all included in the result set, recall remains high. However, in the context of geodabs, an aggressive normalization function could over simplify trajectories. In this case, recall would start dropping, especially if the normalized sequence of points is shorter than the noise threshold specified by the winnowing algorithm. Observing the evolution of precision and recall is therefore a good way to determine the extent of the normalization. In ranked retrieval, this is precisely the aim of a precision and recall (PR) curve [97]. In section 5.6.1, we empirically show how a PR curve can be used to find the best parameters for a normalization function.

In conclusion, as long as precision and recall improves, the extent to which a trajectory is normalized can be increased. In contrast, a drop in precision or recall clearly indicates that the fingerprints do not capture what characterizes and differentiates the sequences of points anymore. Hence, to identify the optimal extent of a normalization function, the evolution of precision and recall can be observed on a sample dataset.

5.6 Evaluation

In this section, we highlight the cost of computing distances and discovering motifs with DFD, DTW and Jaccard in dense trajectory datasets. To this aim, we characterize a large synthetic datasets and the configuration parameters we used to perform our evaluation. Our experiments confirm the pragmatic observation made in Section 5.1.1 and enable us to focus on Jaccard based methods. We then compare geodabs with geohashes, both in term of efficiency and effectiveness on a large and dense trajectory dataset. Finally, we evaluate how a geodab index can be sharded across a set of nodes.

5.6.1 Evaluation Setup

Datasets

To perform our evaluation, we extensively rely on a synthetic dataset and on the road network extracted from the OpenStreetMap dataset [65]. In the context of trajectory indexing, we noticed a lack of large and dense trajectory datasets. For example, the Nokia and Geolife datasets [90, 135] are too small and too sparse to validate our contributions. In addition, these datasets lack the trajectory queries and the associated ground truth that is required to qualitatively assess our solution. These kinds of queries and ground truths, also lack in the popular BerlinMod synthetic dataset [40]. Therefore, we used the trajectory generator introduced in [25] to create a synthetic dataset, a set of trajectory queries and the associated ground truth.

Our dataset is based on 5'000 unique routes constrained on a road network and generated with the GraphHopper library [80]. The routes are all located in a dense area of 300 square kilometres located around the center of London. We use these routes to generate 10 similar trajectories in one direction and 10 similar trajectories in the opposite direction. These trajectories are sampled uniformaly at a rate of one point every second. The speed of the moving entities is based on the route duration computed by the GraphHopper library. In addition, we add 20 meters of random Gaussian noise to every sampled points to make the dataset more realistic.

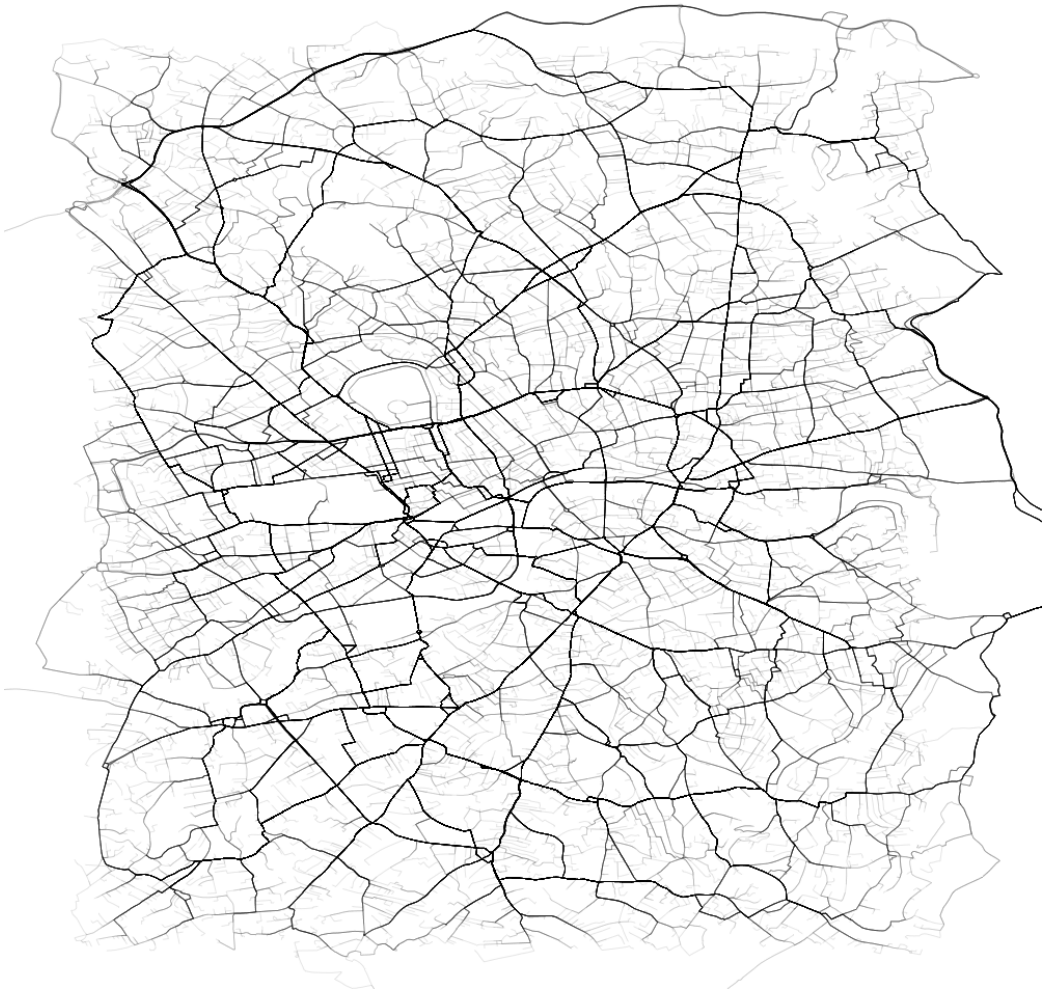


Figure 5.9: Routes used to generate the trajectory dataset

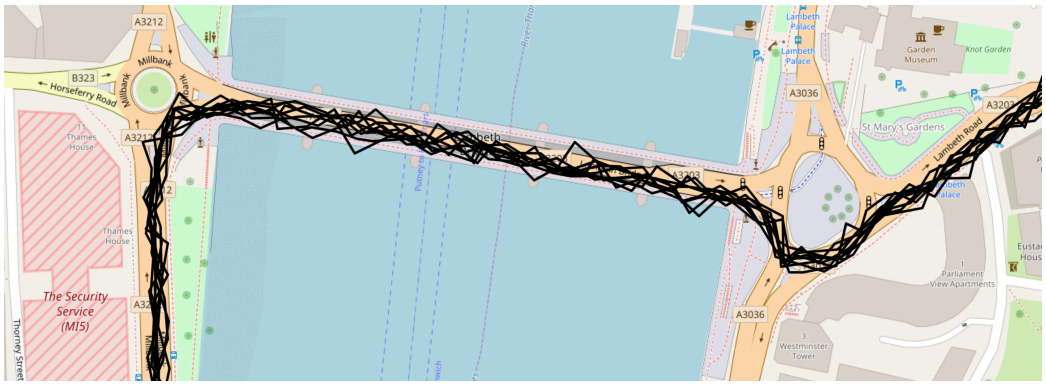


Figure 5.10: Similar trajectories extracted form the dataset

Figure 5.9 visually depicts the 5′000 routes used to generate the trajectory dataset. The 100′000 trajectories derived from these routes form a very dense dataset of 6.3*Gb* that mimics the traces GPS trackers would record. As of today, we found no publicly available counter parts to this synthetic dataset. However, we believe that additional work on the generation of a dense trajectory dataset would greatly benefit the research community. Figure 5.10 illustrates a set of 10 similar trajectories generated on the basis of a route constrained to the road network.

Configuration Parameters

In order to evaluate the effectiveness of our solution, we need to find appropriate configuration parameters. First, we empirically evaluated several configurations and observed the best results with a normalization based on geohashes of 36 bits, a lower bound of $k = 6$ and an upper-bound of $t = 12$. As we get closer to the poles, the width of the geohashes tends to shrink. In London, a geohash of 36 bits has a width of 95 meters and a height of 76 meters. As a trajectory normalized with geohashes rarely follows a diagonal path, we can assume that the average length of a move between two geohashes is approximately 85 meters. Therefore, the lower-bound k translates to a segment threshold of approximately 510 meters. Segments shorter than this threshold are considered as noise. The upper-bound t translates to a segment threshold of approximately 1020 meters. Segments greater than this threshold are guaranteed to be detected. To validate our parameters, we tested several levels of normalization, performed queries on a sample of our dataset and plotted the corresponding precision and recall curves.

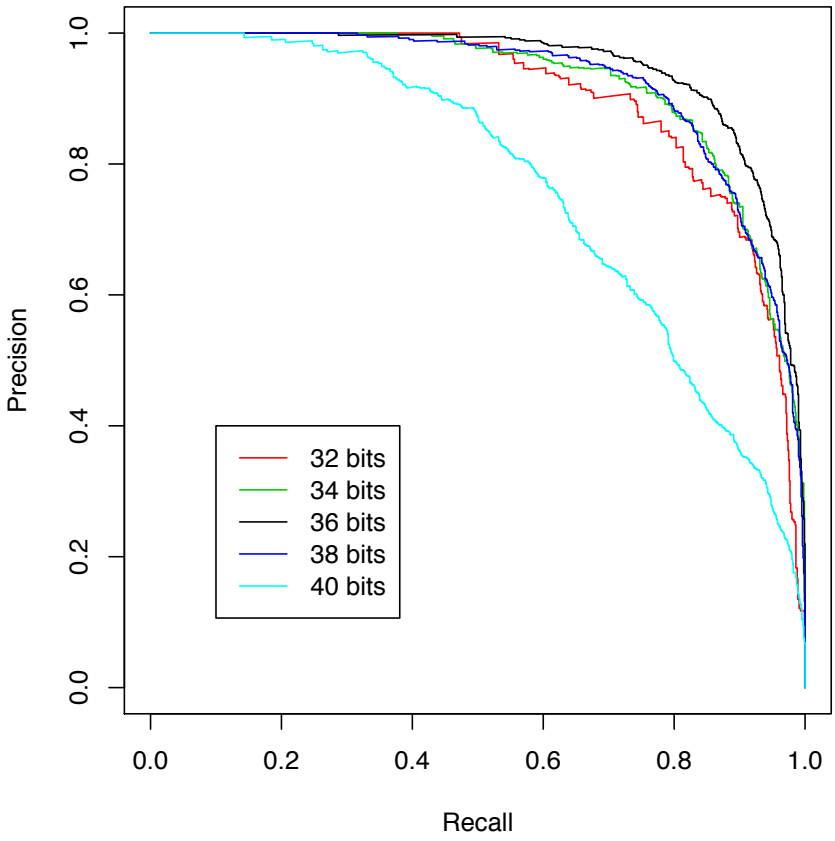


Figure 5.11: Verifying configuration parameters with a PR curve

As highlighted in Figure 5.11, the normalization based on geohashes of 36 bits clearly outperform its upstream and downstream counterparts. Automating the discovery of the appropriate parameters is a difficult task, because the number of possible combinations is very large and each configuration requires building and querying an index. A hill-climbing strategy could probably be used to address this problem, and this might be part of our future work.

5.6.2 The Cost of Computing Distances

In this section, we characterize the cost of computing distances between trajectories and discuss their limits when searching trajectories in dense datasets. We begin by reminding some fundamental distance measures. We then compare them with the Jaccard distance used in the context of our paper.

Ground Distance

Equation 5.2 describes the haversine ground distance formula, where R corresponds to the earth's radius in meters. Given a pair of points $p_l = (\varphi_l, \lambda_l)$ and $p_k = (\varphi_k, \lambda_k)$, the resulting value in $d(p_l, p_k)$ corresponds to the ground distance in meters.

$$2R \arcsin \sqrt{\sin^2\left(\frac{\varphi_l - \varphi_k}{2}\right) + \cos(\varphi_k)\cos(\varphi_l)\sin^2\left(\frac{\lambda_l - \lambda_k}{2}\right)} \tag{5.2}$$

Dynamic Time Warping

Equation 5.3 presents the recursive function used to compute the dynamic time-warping distance (DTW) [129]. Given a trajectory $P = \langle p_1, ..., p_m \rangle$ and a trajectory $Q = \langle q_1, ..., q_n \rangle$, the resulting value in $dtw(|P|, |Q|)$ corresponds to the DTW distance between the trajectories.

$$dtw(i,j) = \begin{cases} \infty & \text{if } i = 0 \text{ or } j = 0 \\ 0 & \text{if } i = j = 0 \\ d(P_i, Q_j) + min \begin{cases} dtw(i-1, j) \\ dtw(i, j-1) \\ dtw(i-1, j-1) \end{cases} & \text{otherwise} \end{cases} \tag{5.3}$$

Discrete Fréchet Distance

Similarly, Equation 5.4 describes the recursive function used to compute the discrete Fréchet distance (DFD) [41]ite. The resulting value in $dfd(|P|, |Q|)$ also corresponds to the DFD distance between the trajectories.

$$dfd(i,j) = \begin{cases} d(P_i, Q_j) & \text{if } i = j = 1 \\ max \begin{cases} d(P_i, Q_j) \\ min \begin{cases} dfd(i-1, j) \\ dfd(i, j-1) \\ dfd(i-1, j-1) \end{cases} \end{cases} & \text{otherwise} \end{cases} \tag{5.4}$$

Performance Evaluation

We compute the distance between a single query trajectory of length t and a set of trajectory candidates of size c, where each candidate has a length of t. In such a scenario, the computational cost associated with the computation of DTW and DFD is characterised by a complexity of $O(c * t^2)$.

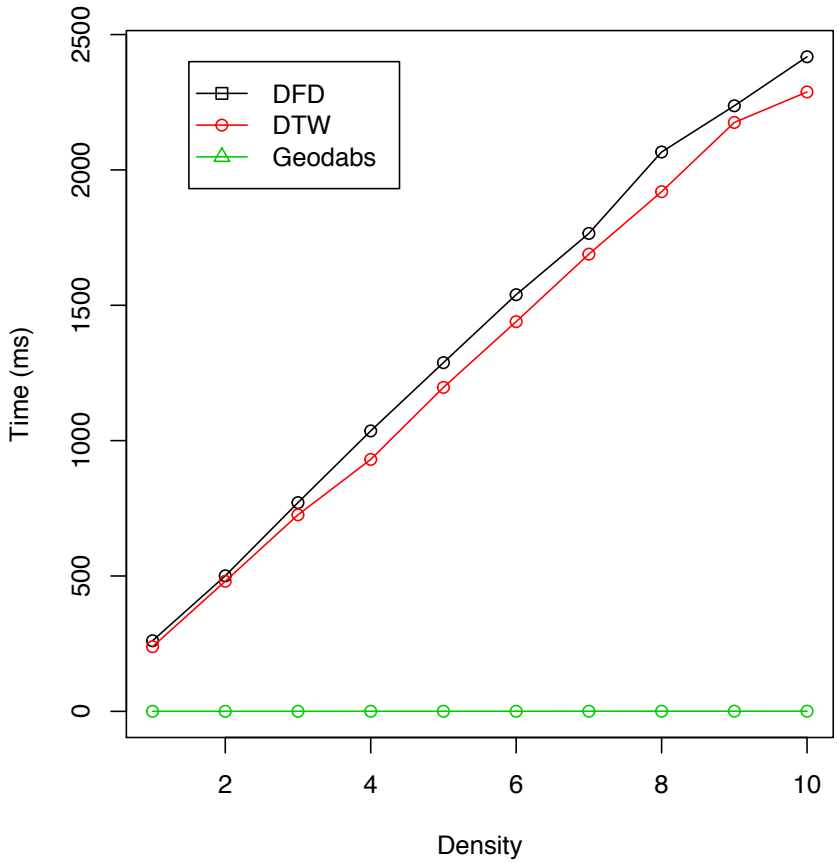


Figure 5.12: Increasing the number of trajectory candidates

In Figure 5.12, the size of the set of trajectory candidates c remains constant, and the length of the query and candidate trajectories increases. As highlighted here, as the length of the trajectories increases, so does the computational time in a polynomial manner. Therefore, when a dataset is primarily made of long trajectories recorded at a high sampling rate, computing DTW or DFD is impractical.

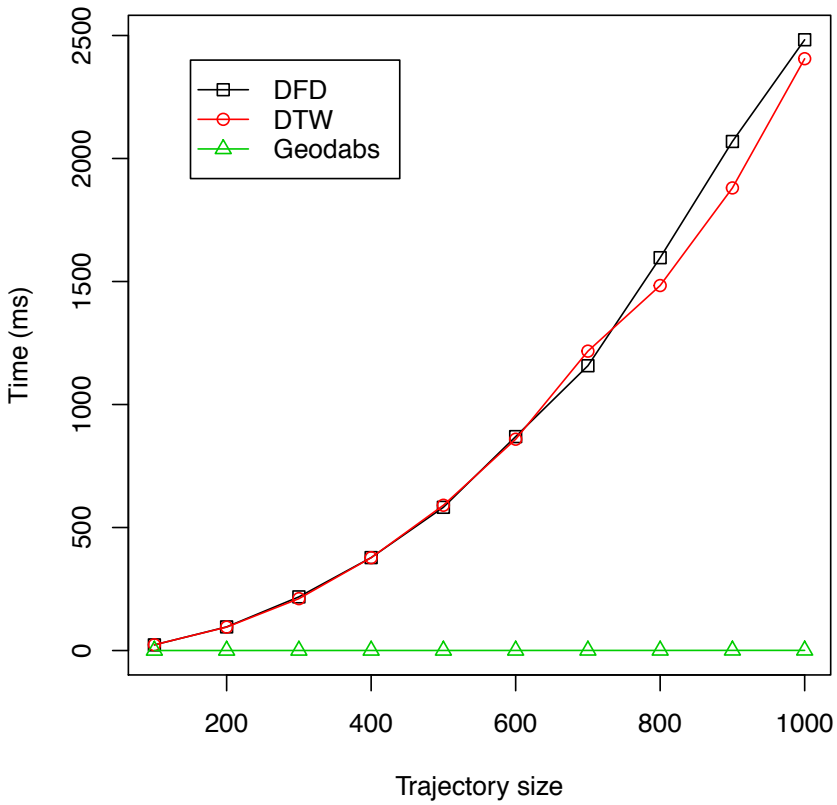


Figure 5.13: Increasing the length of the trajectory candidates

In Figure 5.13, the size of the set of trajectory candidates c densifies and the length of the trajectories t remains constant. As the size of the of the candidate set increases, so does the computational time in a linear manner. In both cases, we notice that computing the scores associated with 10 trajectories of 1000 points takes more than 2500 milliseconds. In the context of a very dense trajectory dataset, a query can return many more relevant trajectory candidates, for which a distance measure still has to be computed. Therefore, it is obvious that relying on these distance measures might be qualitatively sound but clearly unsustainable at scale. In contrast, as highlighted in Figures 5.12 and 5.13, computing the Jaccard distance for ordered sets of geodabs extracted from the trajectories is very inexpensive. This clearly confirms the correctness of the pragmatic observation made in Section 5.1.1.

5.6.3 The Cost of Discovering Motifs

In order to find motifs in pairs of trajectories with geodabs, we have to make some assumptions regarding the normalization step. First, we use our dataset to estimate the average number of fin-

gerprints extracted per meters a from normalized trajectories. As a result, when looking for motifs of length l, we can translate this length to a number of fingerprints $f = l * a$. Therefore, given two ordered sets of geodabs F_i and F_j obtained by fingerprinting the trajectories S_i and S_j, the problem now consists in returning a pair of motifs $(\bar{F}_i, \bar{F}_j)$ such that $length(\bar{F}_i) = length(\bar{F}_j) = f \wedge \nexists(\bar{F'}_i, \bar{F'}_j)$ for which $d_J(\bar{F'}_i, \bar{F'}_j) < d_J(\bar{F}_i, \bar{F}_j)$. As the ordered sets F_i and F_j are usually relatively small, a brute force implementation of this method gives good results. Because of the normalization and the fingerprinting, the motifs discovered with this approach are not strictly equivalent in terms of length and are subject to threshold effects. However, the results we observed in practice are good approximations of the best result.

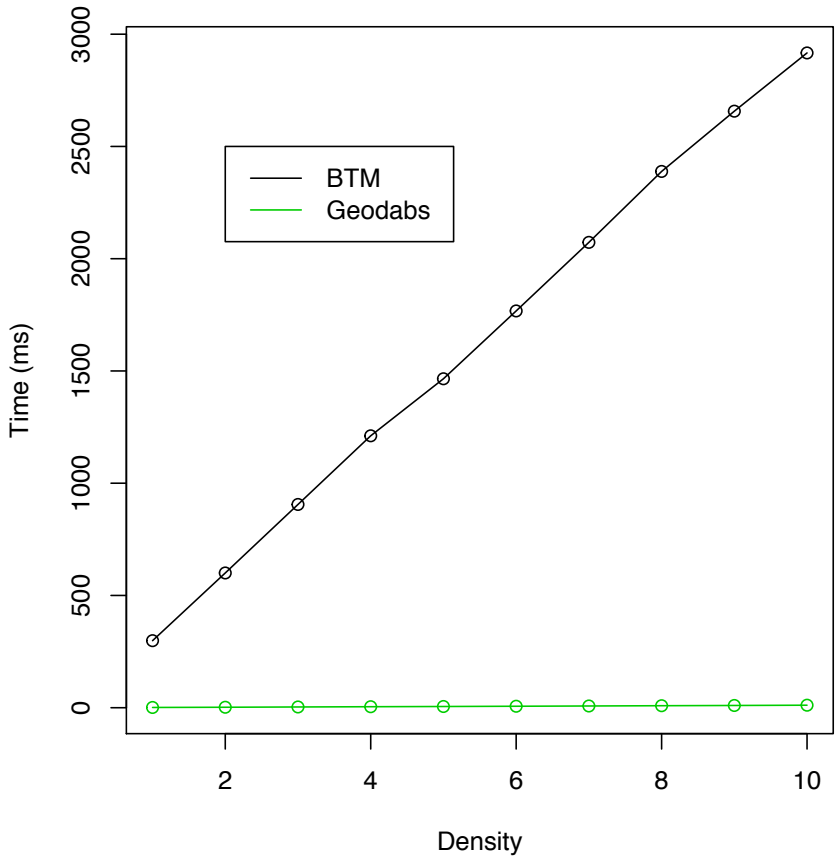


Figure 5.14: Motif discovery with increasing trajectory candidates

In Figure 5.14, we compare our method with an optimized algorithm, called bounding-based trajectory motif (BTM), which gives exact solutions to the motif-discovery problem by computing DFD for every motif pair in the trajectories [119]. As illustrated here, as the number of trajectory candidates densifies, so does the computational time. Again, our method based on geodabs appears to be a necessary tradeoff.

5.6.4 The Cost of Indiscrimination

An inability of the index to discriminate between true and false positive translates to a greater set of trajectories for which the distance has to be computed. In this section, we characterize the effectiveness and the probabilistic nature of the geohash and geodabs indexes. We then show how an inability to discriminate directly affects performances.

Index Effectiveness

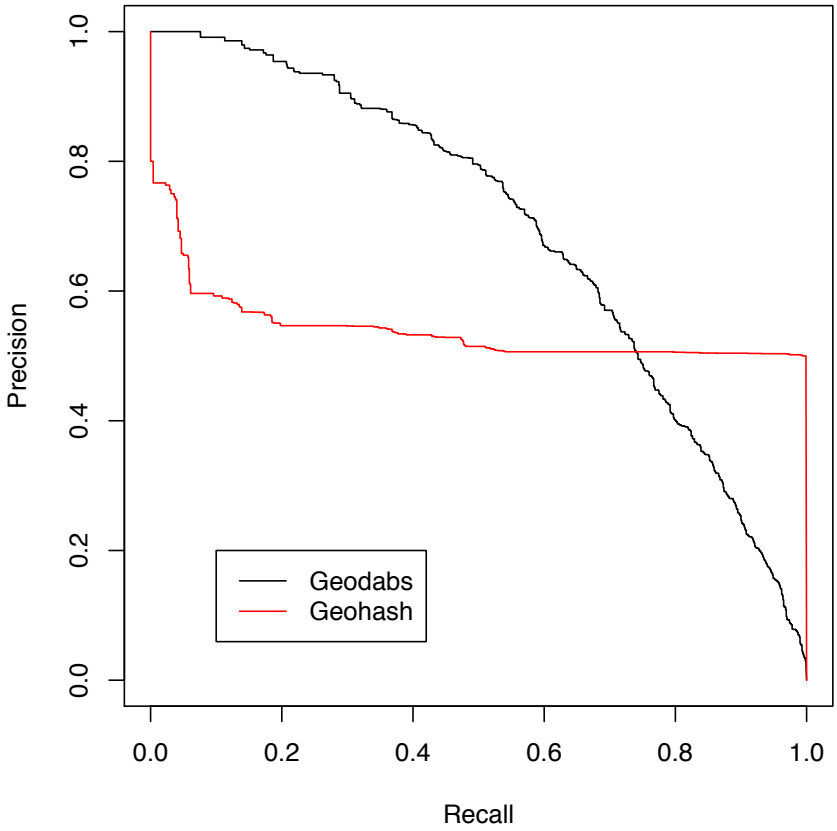


Figure 5.15: PR curve

Figure 5.15 depicts the PR curves obtained by querying the geodabs and geohash indexes [97]. When looking at the geohash curve, we first notice that precision drops rapidly as recall increases. This is due to the inability of the geohash index to discriminate among similar trajectories that go in opposite directions. Because each trajectory of our synthetic dataset is associated with a return path, the geohash curve tends to stabilise at a precision of 0.5, as recall increases. The curve

associated with the geodab index clearly shows that our method addresses this discrimination issue. Furthermore, we also notice that the first results returned by the geodab index are characterized by very high precision. In the context of a very dense dataset, this property is desirable because we can focus on the subset of the most relevant results.

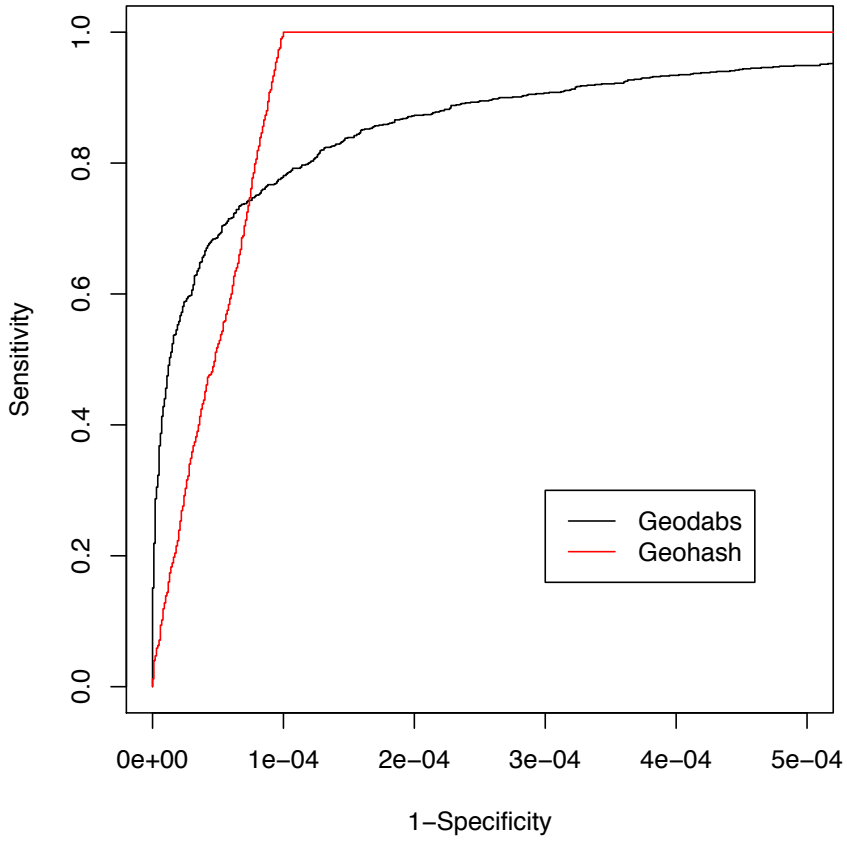


Figure 5.16: ROC curve

Figure 5.16 depicts the receiver-operating-characteristics (ROC) curve obtained by querying the geodab and geohash indexes [48]. In ranked retrieval, *sensitivity* usually corresponds to *recall* and *specificity* is given by $tn/(fp+tn)$. Thus, in contrast with the PR curve, the ROC curve enables us to qualitatively assess the full retrieval spectrum [97]. As we look at the full retrieval spectrum, the quality our results is exacerbated by the size of the dataset. Therefore, it is important to notice that the plot focuses on a very narrow interval of the *specificity*. In fact, qualitatively speaking, both indexes are characterized by a very high sensitivity and a marginally low number of relevant results are lost. This is confirmed by computing the area under the ROC curve (AUC) that is of 0.999889 for geodabs and 0.9999521 for geohashes. Here, the minor difference in terms of AUC comes from the fact that a marginal number of relevant results can be missed with geodabs. the fact that the

curve associated with geodabs climbs more steeply, however, confirms that the first results returned by our method are more relevant.

Index Efficiency

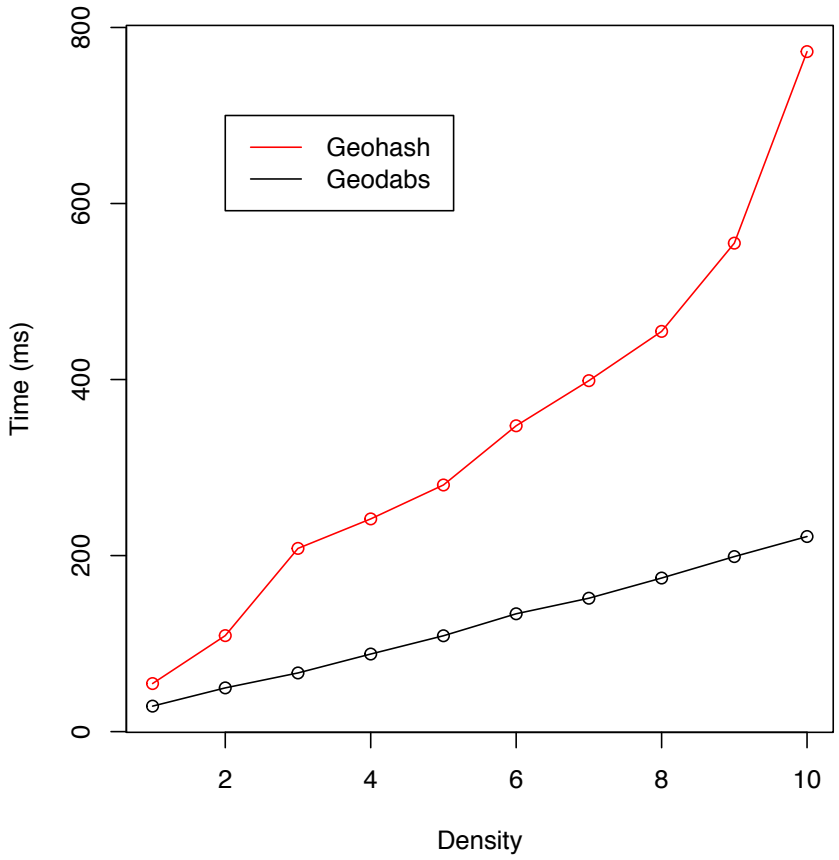


Figure 5.17: Executing 100 queries on a large dataset of increasing density

Figure 5.17 compares the average time needed to process 100 queries on inverted indexes built with a sample of up to 10′000 trajectories. Here, in contrast with the results highlighted in Section 5.6.2, the number of trajectory candidates is not controlled. Therefore, the difference between geohash and geodabs mainly highlights the inability of geohash to discriminate among trajectories. By combining several cells into one hash, a geodab not only discriminates on the direction of the trajectory, but also by all its constituents. Therefore, the number of candidates for which the Jaccard distance has to be computed is significantly reduced. As a result, processing queries is significantly faster but as shown earlier, the quality is not compromised.

5.6.5 The Distribution of the Index

We test the distributed nature of our index with a global road network extracted from the full dump of OpenStreetMap. Here, we make the assumption that the distribution of trajectories recorded across the world should mostly fit on a road network and be characterized by a similar distribution. The geodabs produced by our algorithm are characterized by a geohash prefix of 16 bits that can easily be extracted with a bitwise operation. Geohashes of depth 16 subdivide space into 2^{16} cells characterized by a width of approximately 156 kilometres at the equator.

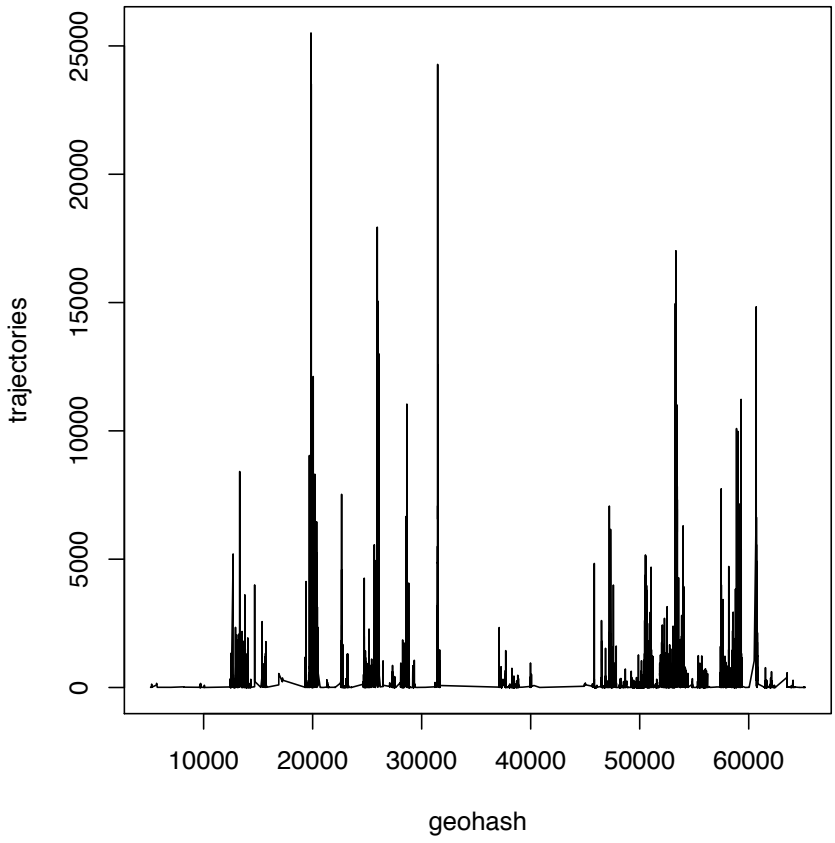


Figure 5.18: Distribution of the trajectories in geohash areas

In Figure 5.18, we plot the number of trajectories per geohash and we notice some very dense areas. For example, the highest peak located on the left of the diagram corresponds to the geohashes located around Mexico city. In contrast, the voids we have between the peaks correspond to areas of low activity, such as oceans.

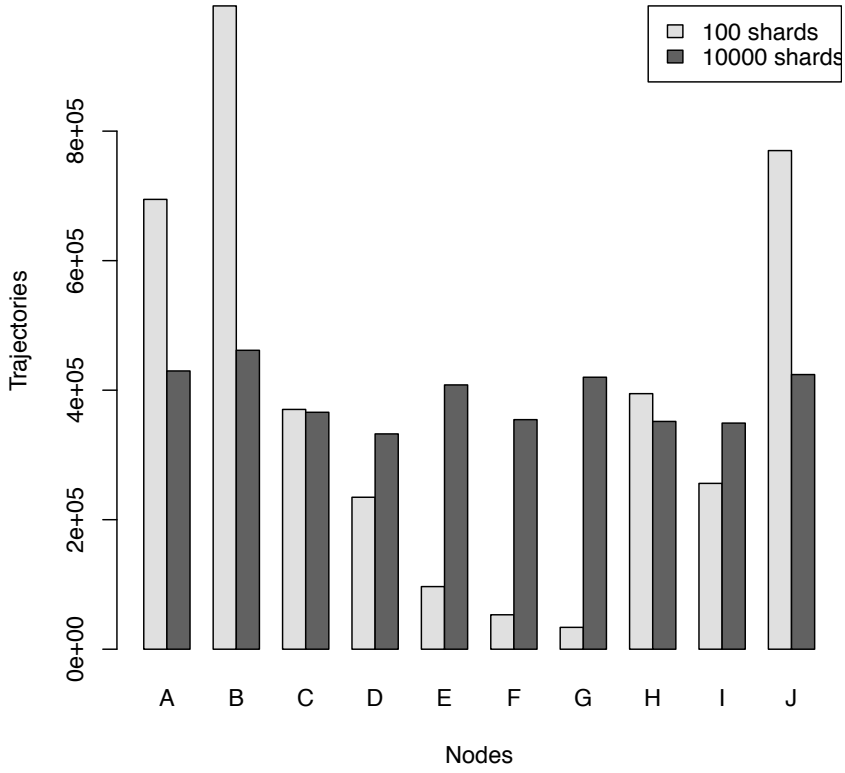


Figure 5.19: Distribution of the trajectories in a 10 nodes cluster

In Figure 5.19, we assume that the index is distributed across 10 nodes. We notice that a small number of shards (100) is not sufficient to distribute the data in a balanced fashion. However, with a greater number of shards (10′000), the data are well balanced on the nodes. Therefore, there is a tradeoff to make between preserving locality (to reduce the number of shards contacted when performing queries) and breaking locality (to spread the data evenly in the the cluster).

5.7 Conclusion

In this paper, we have shown that fingerprinting, and more specifically winnowing, can be used for indexing trajectories. We have introduced geodabs, a construction that combines hashing and geohashing to discriminate on the spatial and on the temporal dimensions. In addition, we have shown that geodabs can be used to scale and distribute an index across several nodes in a cluster. We have demonstrated how trajectory normalization can be used to improve the quality of an index. Finally, we discussed several pragmatic experiments that demonstrated the effectiveness and efficiently of trajectory fingerprinting with geodabs.

Part III

Future: Predicting Trajectories

Chapter 6

Capturing complex behaviour for predicting distant future trajectories

Bertil Chapuis, Arielle Moro, Vaibhav Kulkarni, and Benoît Garbinato. Capturing complex behaviour for predicting distant future trajectories. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, pages 64–73. ACM, 2016

Abstract

We put forth a system, to predict distant-future positions of multiple moving entities and index the forecasted trajectories, in order to answer predictive queries involving long time horizons. Today, the proliferation of mobile devices with GPS functionality and internet connectivity has led to a rapid development of location-based services, accounting for user mobility prediction as a key paradigm. Mobility prediction is already playing a major role in traffic management, urban planning and location-based advertising, which demand accurate and long time horizon forecasting of user movements. Existing prediction methodologies either use motion patterns or techniques based on frequently visited places for predicting the next move. However, when it comes to distant-future, human mobility is too complex to be represented by such statistical functions. Therefore, the existing techniques are not well suited to answer distant-future queries with a satisfactory level of accuracy. To tackle this problem, we introduce a novel spatial object, *'Representative Trajectory'*, which embodies the movements of users amongst their zones of interest. We propose means to empirically evaluate the quality of this object and dynamically adapt its extraction method based on user mobility behaviour. We rely on an inverted index to store the predicted trajectories that scales well with the number of moving entities. Our evaluation results show that the technique achieves more than 70% accurate predictions with the best extraction technique. This shows that

longer query time horizons do not necessarily demand complex spatial indexing schemes, which have to be rebalanced as they grow and which is a constantly experienced problem while answering predictive queries.

6.1 Introduction

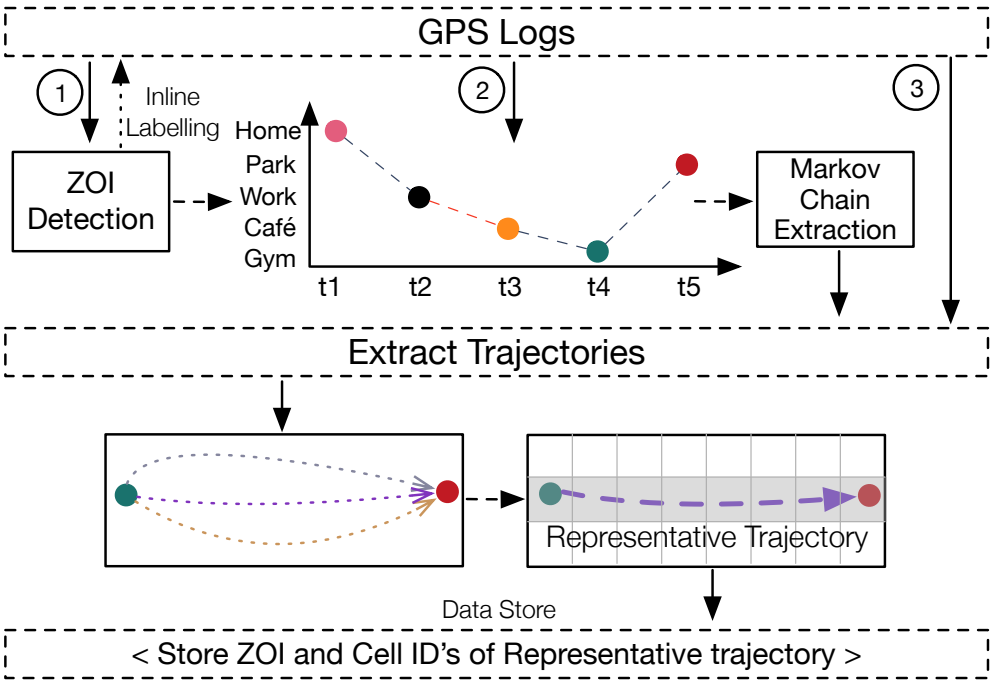


Figure 6.1: Overview of the System Model.

The booming trend of ubiquitous computing, behavioural prediction and ease of availability of internet services, is directly impacting the way we store, retrieve, process and query data. The coming era will witness dramatic advances in the domain of positioning technologies and localisation. Therefore, location tracking and user mobility prediction is becoming increasingly important. Formulating prediction techniques to attain satisfactory accuracies at high granularities puts forth several challenges. Firstly, maintaining high accuracy is crucial for applications such as urban planning, traffic prediction and managing fleets of autonomous vehicles. Secondly, when multiple moving users are involved, it is important to have a scalable indexing technique specially while answering predictive queries.

The existing solutions to the aforementioned problems are not well adapted to solve all the requirements mentioned above. Firstly, the prediction techniques that are built on motion func-

tions, pattern mining or temporal extrapolations, do not truly capture the complex nature of human movement. This factor is especially critical when accounting for distant future predictions, which typically lie in the order of several hours, for which such statistical means fail. The second problem is, the existing techniques, which attempt to model long time horizon predictions, collect a set of frequently visited places by the user and formulate the future place prediction within this set. Such techniques lose the information lying in between these frequently visited places, which is essential to achieve the fine granularity while predicting. An application, where such a level of prediction comes in useful, is to answer predictive queries related to vicinity matching around some locations that are not necessarily included in the set of frequently visited places of a user, but may lie on the trajectory taken to travel from one frequent place to another. The present techniques, which index trajectories, rely on tree structures that require rebalancing when the tree is updated and thus present scaling issues specially when multiple moving points are involved.

Figure 6.2: An Example of a Distant Future Query Represented with 3 Mobile Users.

In this paper, we put forth solutions to the above problems and propose a complete system capable to answer the kind queries as expressed below, and depicted in Figure 6.2. The query can be expressed as: *"Select all the users who will travel in the vicinity of a given location during their next move with a probability higher than a given threshold"*. We first extract the frequently visited places of the user, formally called Zones of Interest (ZOIs), and the transitions amongst them that will be used to train a Markov mobility model. We then extract the past trajectories in order to attain all the possible paths the user takes amongst ZOIs. We introduce a novel spatial object, "representative trajectory" that captures the substantive user mobility behaviour and adapts its extraction according

to the dynamically changing user movements. The representative trajectories are then indexed when a user enters in a ZOI to answer predictive queries. The system overview is depicted in Figure 6.1. Our key contributions are listed hereafter.

- We put forth a complete system capable of predicting distant future trajectories of mobile users and thus answering matching queries for multiple moving users. This is made possible by taking the transitions within the individual ZOIs, thus accounting for trajectories lying within and therefore achieving higher granularity.
- Our ZOI and representative trajectory computation scheme considers dynamic user movements and adapts the computation parameters according to the mobility behaviours. We introduce a novel spatial object, called *'Representative Trajectory'*, which captures the practical nature of human mobility, by considering the fact that, users can move between two ZOIs through different paths. We further discuss and derive means to extract the best path amongst the several paths to represent the most significant trajectory of the user.
- We describe an evaluation framework that utilises precision and recall to assess the quality of representative trajectories.
- Lastly, we present a practical indexing technique based on inverted index, in order to store the trajectory predictions. This technique does not demand costly rebalancing actions as opposed to existing tree structures.

6.2 Related work

Indexing past, current and future positions of moving entities in order to answer predictive queries is an actively research topic, due to the ubiquity of location based services. Therefore, it is important to distinguish our contribution from the plethora of existing works. At the top level, the literature can be separated, into queries related to a single point moving in one dimensional space, viz., *"find all café's around me in next hour"* [114, 121, 120] and queries accounting for multiple moving points in space, viz., *"find all users that will be in the vicinity of café X in next hour"*. Secondly, the existing work can also be separated on the basis of the sampling rate of tracking the moving object locations. Low sampling rate based approaches rely on manual checkins by the users that may range in the order of one location log a day. On the other hand, high sampling rate based approaches, track the user locations after every few seconds. Both these approaches demand different prediction and indexing techniques and the low sampling rate based technique as presented in [34] is beyond the scope of this work. The proliferation of mobile devices with GPS functionality and uninterrupted internet services today, foster and ease the process of continually tracking the moving objects with a high sampling rate. Hence our focus lies on location logs collected at high sampling rates. Further, the time horizon of the query window is an important aspect, which can be dissected into near future

and distant future queries. Majority of the published work today focuses on near future queries in the order of next 15 minutes [114, 121, 120, 107, 31]. However, our work focuses on distant future queries in the order of several hours. The work attempting to solve distant future queries relies on motion prediction techniques by modelling the movement in terms of motion patterns, motion functions and temporal extrapolation [120]. However, according to our observation, such prediction methodologies fail to grasp and accurately represent long term user movements. Therefore, we utilise Markov models to perform distant movement predictions. We further discuss how to index such predictions for efficiently answering the queries we described in Section 6.1. To summarise, our work lies in answering vicinity matching queries for multiple moving points, whose location logs are tracked at a high sampling rate. We focus on distant future queries that demand accurate prediction methodology for which we depend on mobility Markov models.

Regarding the prediction techniques, a majority of existing work is focussed on predicting movements between certain points of interests [51, 130, 98, 53, 52, 95]. A domain of research also relies on cell based techniques for making predictions at the granularity of network cells [51, 95]. Such schemes completely ignore the trajectories lying in between the individual places, which is critical to answer distant future queries involving multiple moving points with a high degree of accuracy. Additionally, the size of a typical network cell lies in the range of several kilometres, which is not adequate to answer queries related to fine grained vicinity matching. On the other hand, predictions based on map matching techniques are complex and need additional services such as network availability, which is not always feasible and is computationally expensive [81]. Existing prediction techniques considering user trajectories amongst points of interests do not store these models, which is a critical factor to answer certain queries. Further, Kalman filter based prediction approaches, involve higher complexity and thus results in higher latency as discussed in [91]. Our work consists of estimating the ZOIs in which a user spends considerable amount of time and then attain the representative trajectory in between these zones that assist to answer the queries described above. Several techniques have been demonstrated to extract points of interest of users whose central theme is based on clustering [120]. As compared to these traditional approaches, our clustering technique enables to extract the frequently visited places of a user according to the mean of the number of visits, the time spent and the distance covered in the significant location, which better represents such a place. Additionally, setting the spatiotemporal bounds based on individual mobility behaviour allows to extract places that are not necessarily found with direct clustering. Further, we follow a technique to dynamically adjust the parameters to extract the representative trajectory based on the user behaviour to consistently maintain satisfactory levels of precision and recall. Thus, unlike the methods presented in the literature, we account for the user behaviour to set parameters for both, extracting the ZOIs and representative trajectories lying in between the zones.

In traditional indexing schemes, the content is only altered when users explicitly perform updates. This is as opposed to indexing moving object locations, where the data quickly becomes outdated and continual write operations are necessary to keep the data updated. A common ap-

proach to address this issue and decrease the number of updates is to adopt an alternative model for representing the location of moving objects. In [114], Saltenis et al. present techniques to index positions of continuously moving objects. The position of the objects is modelled as linear/non-linear function of time and velocity. They present efficient techniques to index trajectories and partition R-Tree containing motion functions. However, as previously discussed, such techniques fail to accurately formulate distant future predictions. In [68], Hendawi et al. present a framework to predict answers to queries as well as queries themselves by monitoring high query rate areas. However, this technique is restricted to a single object prediction. In [68], Bao et al. propose an index structure for processing predictive queries, however with the assumption that moving objects follow shortest paths during their travel from source to the destination, which is not necessary true in practice according to our observation on real work mobility traces. In [120], Tao et al. present methods to predict and index unknown motion patterns of moving objects using recursive motion patterns to express complex trajectories. In [128], Yanagisawa et al. model the motions into three distinct categories including staying, moving straight and moving randomly. In [77], Jeung et al. present a hybrid prediction model for near future and distant future predictions. However, all these methods are either based only on frequently visited places and ignore encompassed trajectories or fail to model distant future predictions. The indexing techniques, discussed in [114, 121, 120], rely on tree structures that require rebalancing as the number of moving points increase and thus do not necessarily scale well. As a result, our approach is based on a simple and practical strategy, which is an inverted indexing model.

6.3 System Model and Definitions

In this section, we introduce our system model, with formal definitions and notations.

6.3.1 Users and Locations

We consider a set of users $U = \{u_1, \ldots, u_n\}$ moving on the surface of the earth with mobile devices that have the ability to locate themselves, typically via a Global Positioning System (GPS)[1] or some other positioning means, e.g., WiFi Positioning System (WPS)[2]. The definitions presented below are from the view point of one user. The location history of the user is expressed as a sequence L of n locations, as $L = \langle loc_1, \ldots, loc_n \rangle$. Each location, loc_i contained in L, is represented by a 3-item tuple $loc_i = (\phi, \lambda, t)$. The latitude and longitude of the coordinate are represented by $\phi, \lambda \in \mathbb{R}$ respectively, and its timestamp by $t \in \mathbb{N}$.

[1] http://www.schriever.af.mil/GPS

[2] http://en.wikipedia.org/wiki/Wi-Fipositioningsystem

6.3.2 Clusters and Zones of Interest

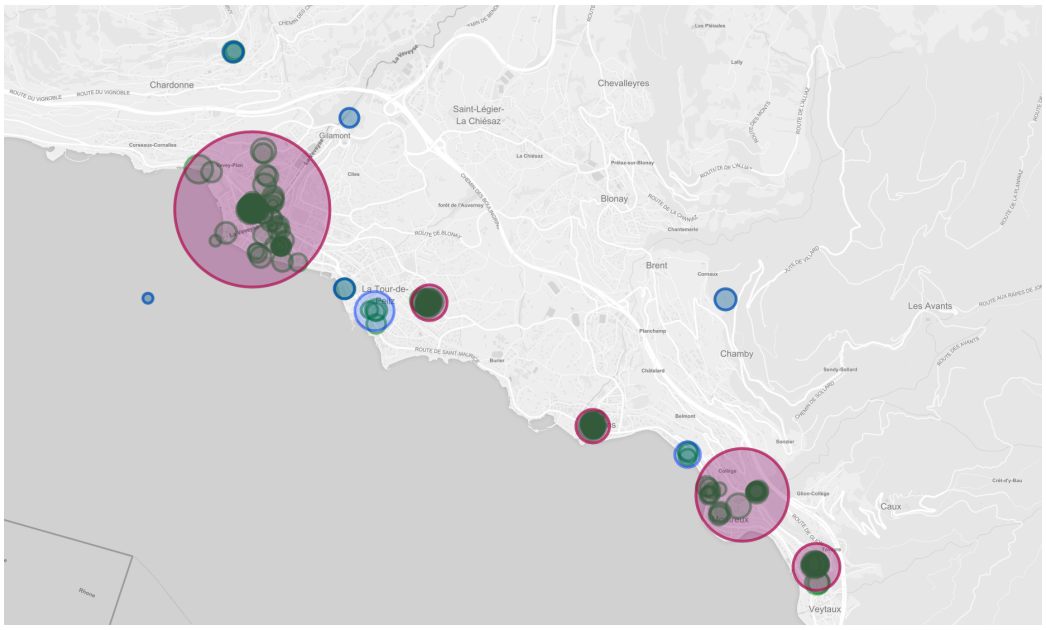


Figure 6.3: Clusters, Cluster Groups and Zones of Interest.

In order to extract the Zones of Interest (ZOI) of a user based on the location history L, we must first introduce the notions of cluster and cluster group.

Cluster

A cluster represents a visit or a stay in a delimited area. It is formed from a subset of locations, sharing the same spatial and temporal characteristics. A cluster is a 4-item tuple $c = (\phi, \lambda, \Delta r, l)$, where ϕ and $\lambda \in \mathbb{R}$ are the latitude and longitude coordinates of a centroid, $\Delta r \in \mathbb{R}$ is its radius in meters and $l \in L$ is the subset of successive locations belonging to c. The centroid of the cluster is the mean of all ϕ and λ of the locations contained in l. Here, the radius corresponds to the maximum distance between the centroid of the cluster and the locations belonging to l. In order to build clusters, we introduce the constraining constants Δd_{max} and $\Delta t_{min} \in \mathbb{R}$, which correspond to the distance expressed in meters and a time duration expressed in seconds respectively. The distance between all the locations lying inside l and the centroid of the cluster must be lower than or equal to Δd_{max}. In addition, the duration between the first location and the last location must be greater than or equal to Δt_{min}. On this basis, we introduce C, the set of clusters extracted from the location history of a user as $C = \{c_1, \ldots, c_n\}$.

Cluster Group

A cluster group is an aggregation of overlapping clusters. Formally, a cluster group is a 4-item tuple $g = (\phi, \lambda, \Delta r, \{c_1, \ldots, c_n\})$. The first three items of a cluster group are the same as the ones present in a cluster. Clusters are grouped whenever they overlap. Consequently, the last item of the tuple corresponds to the set of n overlapping clusters belonging to C. The centroid of the cluster group is the mean of all the centroids of the clusters contained in g, and Δr must be computed in order to enclose all the individual clusters present in g. Finally, we introduce G which contains the n cluster groups belonging to a user as $G = \{g_1, \ldots, g_n\}$.

Zone of Interest (ZOI)

Intuitively, a ZOI is a cluster group that is frequently visited by a user. Let $v_{min} \in \mathbb{N}$ be a constant that represents a minimal number of visits. A cluster group becomes a ZOI if and only if the number of clusters in the group is greater than or equal to the constant v_{min}. However, if we only take into account this constant, it is not possible to find the ZOIs of a user from the beginning of the process, especially if a very high value is set a priori. To resolve this issue, we introduce a variable v_{mean} that is the mean of the number of visits per cluster for a user. This value acts as a reference visit threshold until reaching v_{min}. A ZOI consists of the same items as those of g, further denoted as z to distinguish the two tuples. The centroid and the radius values of z are the same as for g. In addition, we introduce a set Z containing the n ZOIs of a user represented as $Z = \{z_1, \ldots, z_n\}$.

6.3.3 Trajectories

A user can take multiple paths to move from one ZOI to another. Consequently, we introduce the set of trajectories $T_{i,j}$ that can be extracted from the raw set of locations L. Formally, $T_{i,j}$ is a set of n trajectories $T_{i,j} = \{l_1, \ldots, l_n\}$, where each trajectory l_i is a substring of the sequence L in which the first location is contained in z_i and the last location is located in z_j. In addition, the locations recorded between z_i and z_j in l_i do not pass over trajectories going towards any other ZOI.

6.3.4 Mobility Prediction Model

In this work, we consider a mobility prediction model following the structure of a first order Markov chain. Each user has a unique Markov chain, computed on the basis of the elements defined previously. Equation 6.1 depicts a matrix M containing $n \times n$ transitions probabilities, where $n = |Z|$. In other words, each ZOI is a state of the matrix M and a transition probability $p_{i,j}$ represents the prob-

ability to move from a specific z_i to another z_j. As shown in Equation 6.2, the transition probabilities $p_{i,j}$ of the matrix M can be computed using the cardinalities of the sets of trajectories $T_{i,j}$.

$$M = \begin{bmatrix} p_{1,1} & \cdots & p_{1,i} & \cdots \\ \vdots & \ddots & \vdots & \ddots \\ p_{j,1} & \cdots & p_{i,j} & \cdots \\ \vdots & \ddots & \vdots & p_{n,n} \end{bmatrix} \tag{6.1}$$

$$p_{i,j} = \frac{|T_{i,j}|}{\sum_{z_k \in Z} |T_{i,k}|} \tag{6.2}$$

6.4 Predicting Trajectories

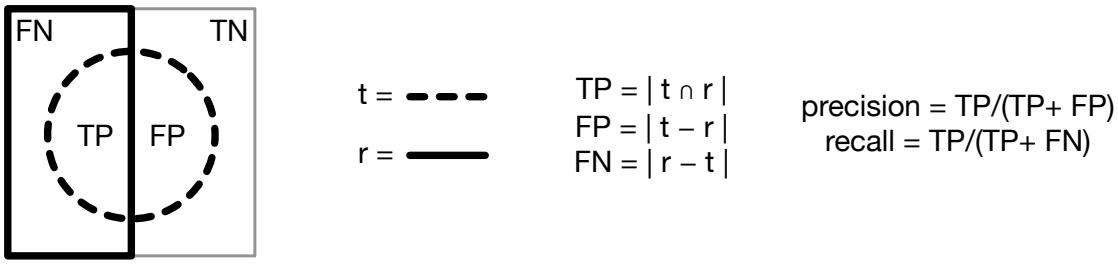


Figure 6.4: Binary Classification in the Context of Trajectories.

Given two ZOIs z_i and z_j, we consider the problem of computing the trajectory that accurately predicts the future moves from one zone to the other. In other words, the idea consists in extracting spatial objects from the actual trajectories that represent the future trajectories of a user between ZOIs. We call these spatial objects *representative trajectories* as they capture the essence of the past movements of a user amongst ZOIs. In this section, in order to evaluate the predictive capacity of a representative trajectory, we first introduce some measures that can be used to assess their accuracy. We then evaluate several strategies for building representative trajectories, some of which take the behaviour of the user into account in order to make the predictions more relevant.

6.4.1 Evaluating Representative Trajectories

The first question to answer is, *"how can the effectiveness of a representative trajectory, be measured in the context of a predictive query?"*. In information retrieval, the performance of a system

is often measured in terms of precision and recall. Given the results of a query, binary classification is used to assess how many of the results are relevant (precision) and how many relevant results were selected (recall). Similarly, given the actual trajectory followed by a user and the corresponding predicted trajectory, it is possible to assess how many subparts of the trajectory are relevant according to the predicted trajectory (precision) and how many subparts of the predicted trajectory were selected (recall). To achieve this goal, we discretise space by introducing a *Grid* that allows us to perform binary classification on all the subparts of a trajectory. More formally, a *Grid* can be described as a set of uniquely identified cells, such that $Grid = \{cell_1, \ldots, cell_n\}$. Figure 6.4 illustrates the calculation of precision and recall in the context of discretised trajectories. Assuming t a set of cells corresponding to a actual trajectory and r a set of cells corresponding to a representative trajectory, the number of true positive cells TP can be expressed by the cardinality $|t \cap r|$, the number of false positive cells FP can be expressed by the cardinality $|t - r|$ and the number of false negative cells FN can be expressed by the cardinality $|r - t|$. Consequently, *precision* can be expressed as $TP/(TP+FP)$ and recall by $TP/(TP+FN)$. On this foundation, we formally introduce *Precision* and *Recall* that are defined in Equations 6.3 and 6.4. Finally, the measure F_β, often denoted as F-score, combines precision and recall in a single metric that can be expressed as the weighted harmonic mean described in Equation 6.5. Depending on the situation, one may decide to give more importance to precision or to recall by adjusting the weight factor β.

$$Precision(t,r) = \frac{|t \cap r|}{|t \cap r| + |t - r|} \tag{6.3}$$

$$Recall(t,r) = \frac{|t \cap r|}{|t \cap r| + |r - t|} \tag{6.4}$$

$$F_\beta(t,r) = (1+\beta^2) \cdot \frac{\text{Precision(t,r)} \cdot \text{Recall(t,r)}}{\beta^2 \cdot \text{Precision(t,r)} + \text{Recall(t,r)}} \tag{6.5}$$

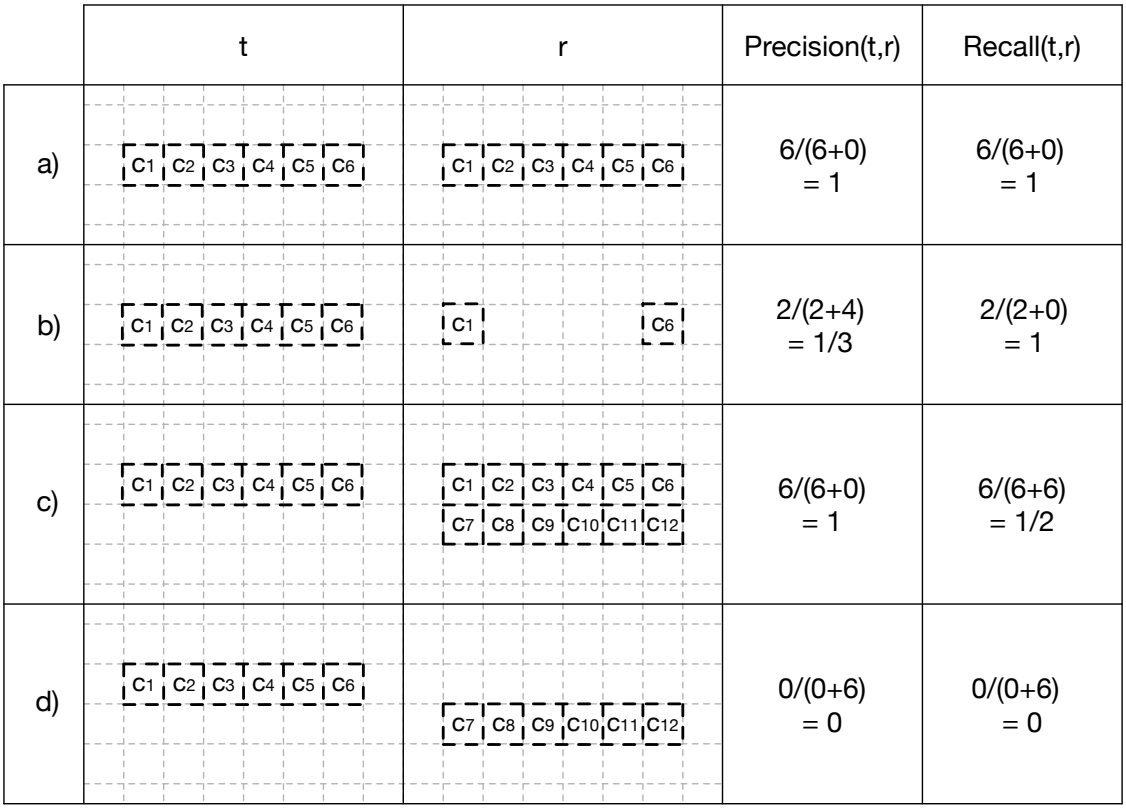


Figure 6.5: Precision and Recall in the Context of Trajectories.

Figure 6.5 shows how precision and recall can be calculated given the cells of an actual trajectory t and the cells of a representative trajectory r. As illustrated here, precision and recall accurately answer the questions how many subparts of the actual trajectory t are relevant according to the representative trajectory r and how many subparts of the predicted trajectory were selected. Figure 6.5a shows that, if t and r perfectly overlap, then precision and recall are high. Figure 6.5b shows that, when r is a subset of t, then precision drops because only few cells of the actual trajectory are relevant according to the representative trajectory. Figure 6.5c shows that if t is a subset of r, then recall drops, because only a few cells of the representative trajectory were selected. Finally, Figure 6.5d shows that, when t and r do not overlap, precision and recall are both low.

6.4.2 Building Representative Trajectories

In the previous section, we have not considered how the cells of a representative trajectory were actually selected. In order to build this representative trajectory, we first introduce the set $\tau_{i,j}$ that is a discretised version of the set of trajectories $T_{i,j}$. More formally, $\tau_{i,j}$ is a set of n discretised trajectories $\tau_{i,j} = \{t_1, \ldots, t_n\}$, where each trajectory t_k is a set of n cells such that $t_k \in \tau_{i,j}$:

$\{cell_1, cell_2, \ldots, cell_n\}$. The importance of a cell in a *representative trajectory* is defined by its number of occurrences in $\tau_{i,j}$. Thus, we introduce the multi set $O_{i,j}$ that counts the number of occurrences of a cell in $\tau_{i,j}$ and can formally be defined as $O_{i,j} \subseteq Grid \times \mathbb{N}^*$. Finally, the cells with a number of occurrences greater than a given threshold are gathered in set $R_{i,j}$ that constitutes the representative trajectory. As a consequence, when building a representative trajectory, the main challenge consists in selecting a threshold $\theta \in \mathbb{N}^*$ that will select the most accurate and relevant cells for predicting the future moves of a user. In a more formal way, given the threshold θ, a representative trajectory $R_{i,j}$ can be obtained with the function described in Equation 6.6.

$$R(O_{i,j}, \theta) = \{c | (c,n) \in O_{i,j} \wedge n \geq \theta\} \tag{6.6}$$

In addition to these general definitions, we introduce some utility functions that can be used to select thresholds that take the behaviour of the user into account. The function $Mean(O_{i,j})$, formally defined in Equation 6.7, returns the mean number of cell occurrences of the multiset $O_{i,j}$. In a similar way, the functions $Min(O_{i,j})$ and $Max(O_{i,j})$, defined in Equations 6.8 and 6.9, return the minimum and maximum number of cell occurrences of $O_{i,j}$ respectively.

$$Mean(O_{i,j}) = \frac{\sum\limits_{(c,n) \in O_{i,j}} n}{|O_{i,j}|} \tag{6.7}$$

$$Min(O_{i,j}) = \min_{(c,n) \in O_{i,j}} n \tag{6.8}$$

$$Max(O_{i,j}) = \max_{(c,n) \in O_{i,j}} n \tag{6.9}$$

In the previous section we introduced *Precision* and *Recall* in the context of a trajectory t and a representative trajectory r. Since $\tau_{i,j}$ contains several discretised trajectories, we introduce the functions $AvgPrecision(\tau_{i,j}, r)$ and $AvgRecall(\tau_{i,j}, r)$ respectively defined in Equations 6.10 and 6.11 that measure the average precision and the average recall for multiple sets of cells.

$$AvgPrecision(\tau_{i,j}, r) = \frac{\sum\limits_{t \in \tau_{i,j}} Precision(t, r)}{|\tau_{i,j}|} \tag{6.10}$$

$$AvgRecall(\tau_{i,j}, r) = \frac{\sum\limits_{t \in \tau_{i,j}} Recall(t, r)}{|\tau_{i,j}|} \tag{6.11}$$

Mean Threshold

An obvious approach to select a threshold consists in using the mean cell occurrences as illustrated in Equation 6.12. While this method is straightforward and computationally efficient, it does not account for the behaviour of the user within the ZOIs. For example, during the week, a user may always take the same route to go from home to work, while during the weekend he would leave home for excursions and come back at the same place. A threshold, based on the mean of the cell occurrences, is not adapted to capture this kind of behaviour.

$$\theta_{mean} = Mean(O_{i,j}) \tag{6.12}$$

F-Score Threshold

In order to build better representative trajectories, we can consider the problem of selecting the threshold as an optimisation of the F_β score introduced in Section 6.4.1. In other words, given the trajectories of a set $\tau_{i,j}$, the idea consists in computing the average F-Score for all the possible thresholds. Then, the threshold that gives the best average score for F_β can be considered as the best possible threshold for the representative trajectory. More formally, assuming a set of candidate representative trajectories $CR_{i,j}$ expressed in Equation 6.13, one can find the representative trajectory that gives the best F-Score, as described in Equations 6.14 and 6.15.

$$CR_{i,j} = \{r_\theta = R(O_{i,j}, \theta) | \theta \in [Min(O_{i,j}), Max(O_{i,j})]\} \tag{6.13}$$

$$\forall r_\theta \in CR_{i,j} : F_\beta^\theta = \frac{\sum\limits_{t \in \tau_{i,j}} F_\beta(t, r_\theta)}{|\tau_{i,j}|} \tag{6.14}$$

$$F_\beta^{max} = \max_{r^\theta \in CR_{i,j}} F_\beta^\theta \tag{6.15}$$

Finally, θ_{F_β} can be defined as the min θ for which $F_\beta^{max} = F_\beta^\theta$. With this approach, one can decide to give more importance to precision or recall by adjusting the β parameter. The main disadvantage of this technique lies in the fact that, F_β scores have to be computed over all the possible thresholds, which is computationally expensive.

Adaptative Threshold

A third approach consists in assuming that precision and recall, both contain meaningful insights, regarding the behaviour of the user, that can be used to adapt an existing threshold. First, as illustrated in Figure 6.5a, if the user always follows the same path, the representative trajectory is easy to build and the precision and recall are both expected to be very high. In that case, no actions are needed. Second, as illustrated in Figure 6.5b, a low precision and a high recall suggests that the user takes several paths to go from one place to the other. Some regular paths are not included in the representative trajectory, which causes the precision to drop. Consequently, the threshold used to select the cells can be lowered in order to include these cells in the representative trajectory. Third, as illustrated in Figure 6.5c, a low recall with a high precision suggests exactly the opposite. The user follows several paths and some insignificant ones are included in the representative trajectory. Consequently, the threshold used to select the cells can be strengthened. Finally, as illustrated in Figure 6.5d, if the user does not have a consistent behaviour between two ZOIs, which usually happens for week-end excursions, the precision and recall both drop and can be used as insights on the poor quality of the representative trajectory. In order to account for these different scenarios, we first compute a representative trajectory r using the mean threshold, such that $r = R(O_{i,j}, \theta_{Mean})$. On this basis, Equation 6.16 introduces a readjusted threshold that accounts for the user behaviours as discussed.

$$\theta_A = Max(O_{i,j}) * \frac{AvgPrecision(\tau_{i,j}, r) + (1 - AvgRecall(\tau_{i,j}, r))}{2} \tag{6.16}$$

6.5 Solution Architecture

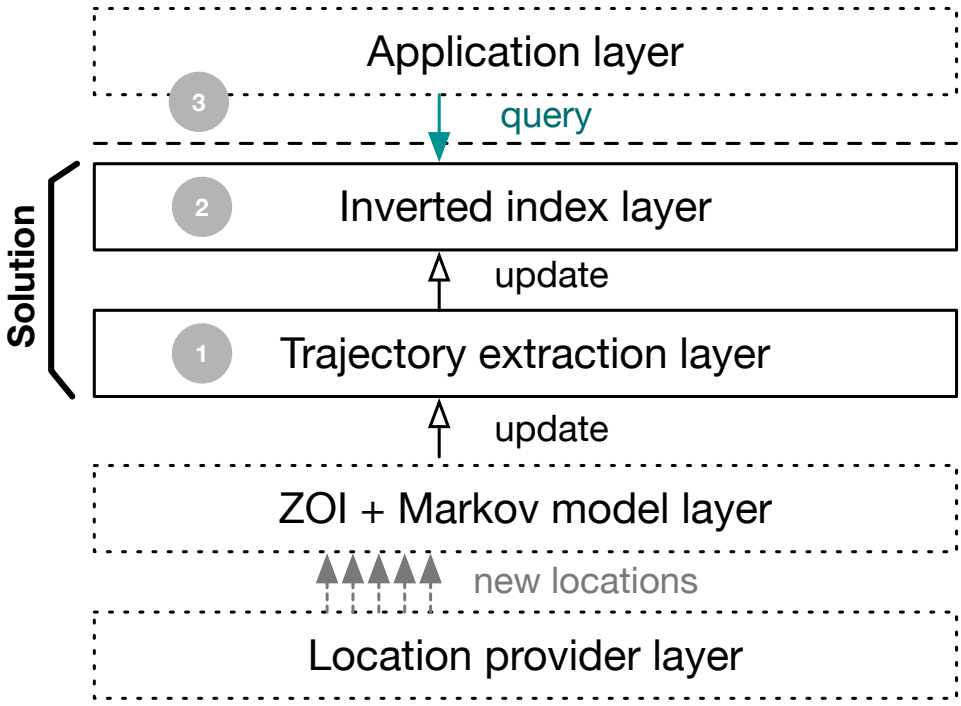


Figure 6.6: Layers of the Solution Architecture.

The previous sections showed how representative trajectories can be extracted from the location data of a single user. In this section, we go back to the initial requirement, which consists in a predictive query involving a set of users. We present a solution architecture based on an inverted index that can be used to answer such queries. In its simplest form, an inverted index is composed of two parts: a *dictionary of terms* where each element points to a *postings list*. In the context of text retrieval, the terms of the dictionary would correspond to words and the postings would correspond to lists of documents. In our context, the terms of the dictionary are the cells of a *Grid* and the postings correspond to tuples $(u, p_{i,j})$ where u is a user identifier belonging to U and $p_{i,j}$ corresponds to the transition probability between two ZOIs, z_i and z_j belonging to M as introduced earlier. On this foundation, the architecture can be divided into three distinct procedures that act on the different layers depicted in Figure 6.6.

6.5.1 Prediction Model Extraction

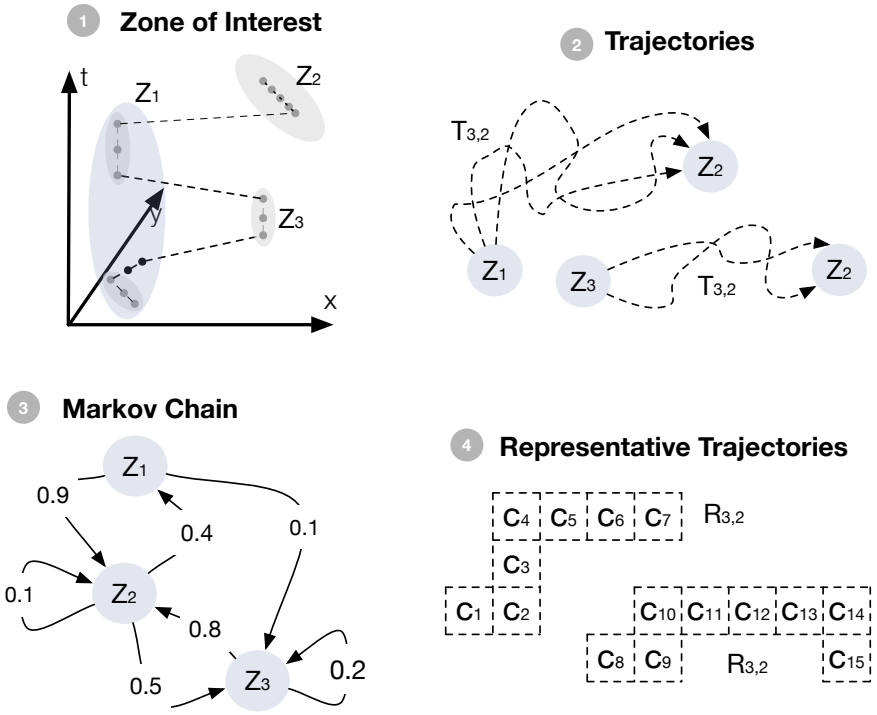


Figure 6.7: Extracting Representative Trajectories.

The first procedure aims at extracting the representative trajectories from the location history of a user. Figure 6.7 recalls the four main successive steps involved in the extraction of the representative trajectories. These steps are described as below and executed for each user u belonging to the set U.

1. **Zones of Interest discovery.** As introduced in Section 6.3, the procedure, first uses a clustering algorithm to extract the set of ZOIs, called Z, from the location history L.

2. **Trajectories extraction.** On the basis of the discovered ZOIs, the sets of trajectories $T_{i,j}$ amongst ZOIs, can be extracted by examining the location history L a second time.

3. **Markov chain computation.** The sets of trajectories $T_{i,j}$ can then be used to compute the transition probabilities $p_{i,j}$ of the Markov chain M. The Markov chain M is then stored for later use.

4. **Representative trajectory extraction.** The set of trajectories $T_{i,j}$ can also be used to compute the representative trajectories $R_{i,j}$. All the representative trajectories $R_{i,j}$ are then persisted for later use.

6.5.2 Inverted Index Update

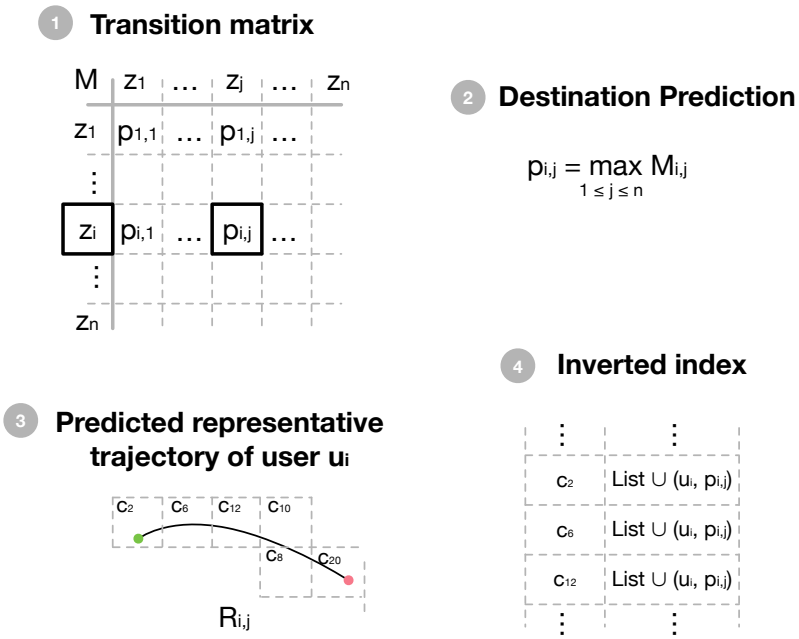


Figure 6.8: Updating Inverted Index.

Using the previously persisted items, the second procedure updates the inverted index every time a user enters in one of the ZOIs. Before presenting the procedure in detail, we must introduce variables z_i and z_j that are assumed to be the current and the predicted ZOIs of a user respectively. Figure 6.8 depicts the four following steps enabling to update the inverted index.

1. **Current ZOI extraction.** The procedure is triggered when a user u enters in one of the ZOIs and the identified ZOI becomes the current ZOI z_i. From that point, it is possible to find all the next possible ZOIs and their respective transition probabilities in the matrix M.

2. **Next ZOI prediction.** The predicted ZOI z_j corresponds to the ZOI with the maximal transition probability $p_{i,j}$ in the Markov chain M amongst the transitions from z_i.

3. **Representative trajectory retrieval.** As soon as the current and predicted ZOIs are identified, the representative trajectory $R_{i,j}$ associated with z_i and z_j can be retrieved amongst all the representative trajectories stored for the user.

4. **Inverted index update.** After the retrieval of the representative trajectory $R_{i,j}$, the inverted index must be updated. To do so, the tuple $(u, p_{i,j})$ is added to all the postings lists pointed by the cells of the representative trajectory.

6.5.3 Answering the query

The third procedure is used to answer queries. We consider queries of the following nature: *"Select all the users who will travel in the vicinity of a given location during their next move with a probability higher than a given threshold"*. We assume that we know the search zone to initiate the query, which can be formally expressed as a tuple $search_{zone} = (loc, \Delta r)$ by the requestor. In addition, the probability threshold stated in the query is denoted p_{th}. In order to answer the query, the $search_{zone}$ is first converted into a set of cells belonging to the *Grid* and called $search_{cells}$.

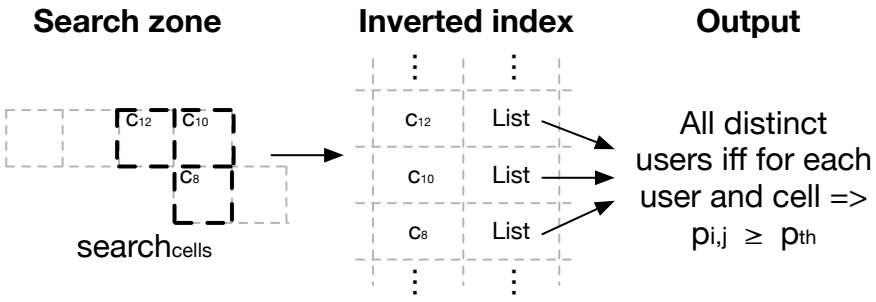


Figure 6.9: Answering the Query.

As illustrated in Figure 6.9, assuming that the predicted representative trajectories $R_{i,j}$ have been added to the inverted index for each user belonging to U, it is now possible to retrieve all the users who will move through the cells specified in the query. In other words, users are selected if the following two conditions are met. First, when at least one cell of their representative trajectory of their next predicted move matches with one of the $search_{cells}$. Second, if their probability $p_{i,j}$ associated with their matching cell is greater than or equal to the probability threshold p_{th}.

6.6 Evaluation and Discussion

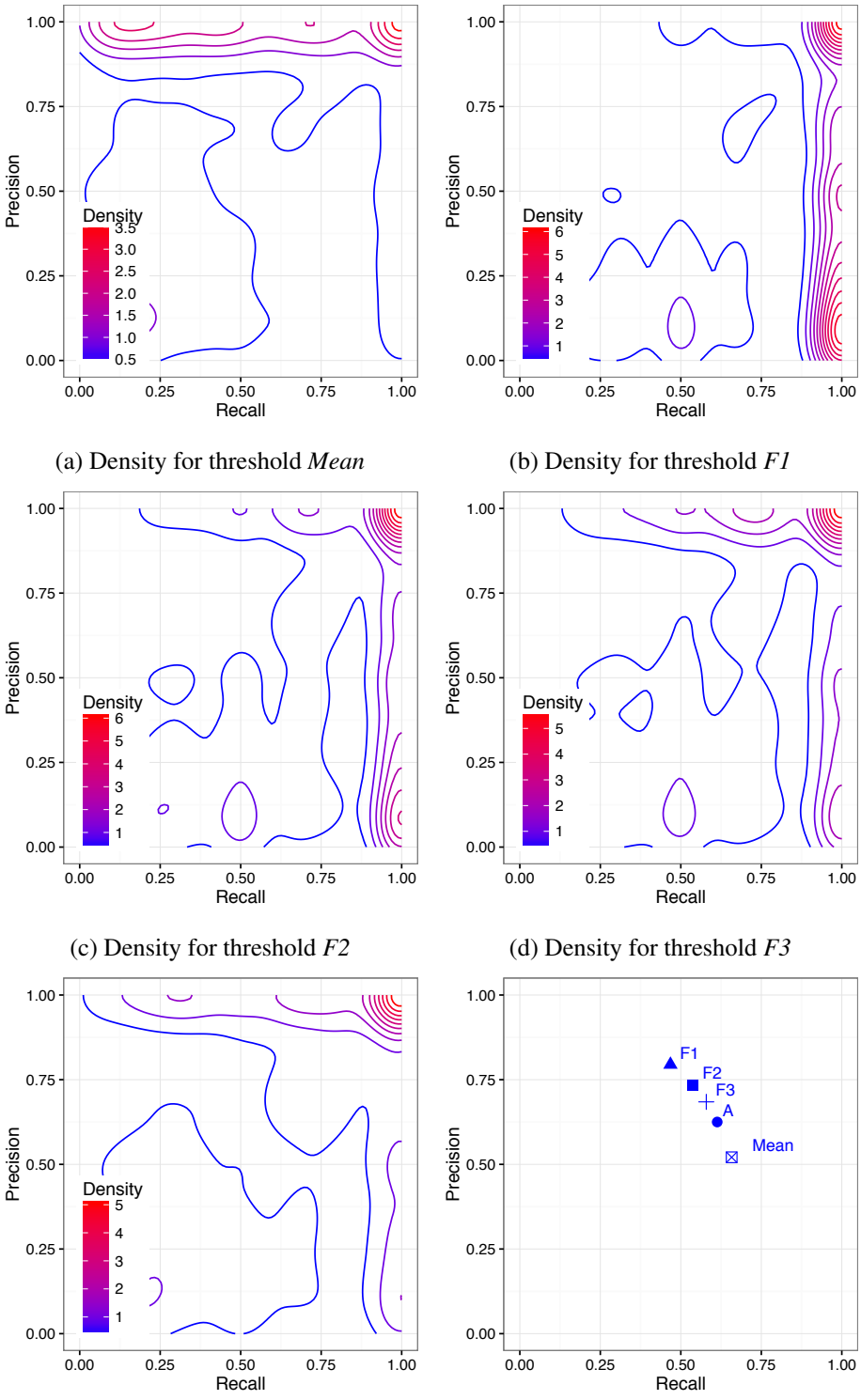
We perform the evaluation of the system based on the Nokia data set [90], which consists of mobility traces collected from 188 users around lake Geneva region in Switzerland from October 2009 to March 2011. The mean duration of the participants, which mainly consisted of university students and professionals, was about 14 months consisting of more than 10 million location points. We implemented the components of our architecture, including the clustering algorithm, the Markov chain and the representative trajectory extraction algorithms in Scala. We used the Java Google S2 Library in order to discretise space[3]. The grid provided by this library comes with the guarantee that

[3] https://github.com/google/s2-geometry-library-java

the cells will have similar areas and we configured it to produce cells that are one square kilometre on an average. In the context of this evaluation, we used a first order Markov chain to predict the movements of a user across ZOIs. This choice was motivated by the fact that, our experiments with second order Markov chains showed a gain of only 2% in terms of accuracy for predictions made with the whole dataset. We believe a much greater gain in terms of accuracy can be obtained by simply cleaning and sanitising the dataset.

In order to evaluate our methods for building representative trajectories, we used 70% of the dataset for creating the Markov chains and the representative trajectories. Using the remaining 30% of the dataset, we performed 4727 trajectory predictions and evaluated the quality of the outputs using the actual trajectories followed by the users. Figure 6.10 uses two dimensional Kernel Density Estimate (KDE) to evaluate these predictions in terms of precision and recall. In these plots, a high density corresponds to a large concentration of predictions. The density can be greater than one as the probability is multiplied by an area of the two dimensional space. In these plots, the upper-right corner is the sweet spot. As illustrated in Figure 6.10a we ideally foster predictions characterised by a high precision and a high recall. The lower-right corner would typically contain predictions as the one illustrated in Figure 6.10b. Such predictions are symptomatic of a strong threshold that filters too many cells out of the representative trajectory. On the contrary, the upper-left corner would typically contain predictions as the one illustrated in Figure 6.10c. Such predictions are often synonym of a weak threshold that preserves too many cells in the representative trajectory. Finally, the lower-left corner contains results characterised by a poor precision and a poor recall as the one illustrated in Figure 6.10d. Such predictions can be synonymous with an inconsistent behaviour between the ZOIs or with a completely wrong result at the level of the Markov chain.

The five threshold selection methods we evaluate in order to build representative trajectories are *Mean* (θ_{Mean}), *F1* (θ_{F_1}), *F2* (θ_{F_2}) and *F3* (θ_{F_3}) and *A* (θ_A). In Figure 6.10a, the threshold is set to the *Mean* number of occurrences of the cells in the representative trajectory. This plot highlights a high density on the upper side, i.e, predictions are usually characterised by a high precision but recall varies a lot. As previously stated, this gives us the intuition that the threshold is too weak. When the user takes several distinct paths to go from one ZOI to the other, some irrelevant cells are included in the representative trajectories. Therefore, this method fails at accurately accounting for the behaviour of the user and may return a significant number of cells when used in practice. In Figure 6.10b, the threshold is obtained by searching a value that gives the best F_β score when β is set to 1. In that case, the density plot highlights predictions characterised by a very high recall but a highly varying precision. This is symptomatic of a strong threshold value that filters out most of the cells of the representative trajectory. In other words, the predicted cells will be highly accurate but cover only a subpart of the trajectory followed by the user in practice. In Figure 6.10c and 6.10d, the β score is respectively set to 2 and 3. Here the tradeoff between precision and recall is clearly highlighted by the fact that, the predictions shift towards a better balance between precision and recall. The positive impact of adjusting the β weight shows that the selected thresholds



(a) Density for threshold *Mean*

(b) Density for threshold *F1*

(c) Density for threshold *F2*

(d) Density for threshold *F3*

(e) Density for threshold *A*

(f) Average precision and recall

Figure 6.10: Precision-Recall Kernel Density Estimation (KDE) for Various Thresholds Selection Methods.

produce fewer predictions at the upper-left and lower-right corners in Figure 6.10d. In Figure 6.10e, the threshold is adaptive and set by taking the behaviour of the user into account as shown in Equation 6.10e. As highlighted in this plot, we obtained the best tradeoff between precision and recall with this method. Furthermore, the density shows that only few results are characterised by a low precision or a low recall. Finally, as previously demonstrated, this method is more efficient than the others since it is not required to compute the F_β score for all the possible thresholds.

Figure 6.10f highlights the average prediction and recall obtained with the five techniques. This plot clearly illustrates the tradeoff that occurs between precision and recall and confirms the well balanced results we obtain with the threshold A. Interestingly, if we compare the results in terms of F-score, as it is often the case in Information Retrieval, the combination of precision and recall would give similar values. Adjusting, the β score may help at evaluating the methods with a single measure, but, as it will be done in the next section, we advise to visually check the predictions in order to assess the quality of the tradeoff between precision and recall.

We now give an overview of the predictions extracted from the dataset with a first order Markov chain and an adaptative threshold θ_A. In Figure 6.11, the blue cells correspond to the predicted representative trajectories. The green and red circles correspond to the starting and ending ZOIs respectively. The black line corresponds to the path followed by the user between the two ZOIs. We first notice that most predictions are visually accurate, in particular the one presented in Figure 6.11a, b and e that are characterised by a high precision and a high recall. Some predictions, such as the ones highlighted in Figure 6.11c and f are correct but at some point the user probably decided to take an alternative path for some unknown reasons. Since we are in the context of future movements, we can easily imagine that, in a live system, the result of the prediction could be used to create an incentive that would influence the behaviour of the user and directly impact the quality of the predictions. Some predictions, such as the one depicted in Figure 6.11b, d and f, have the same starting and ending ZOIs. Such results can typically not be obtained with methods that solely rely on ZOIs and shortest paths to make predictions and clearly highlight the benefit of using representative trajectories to predict future moves.

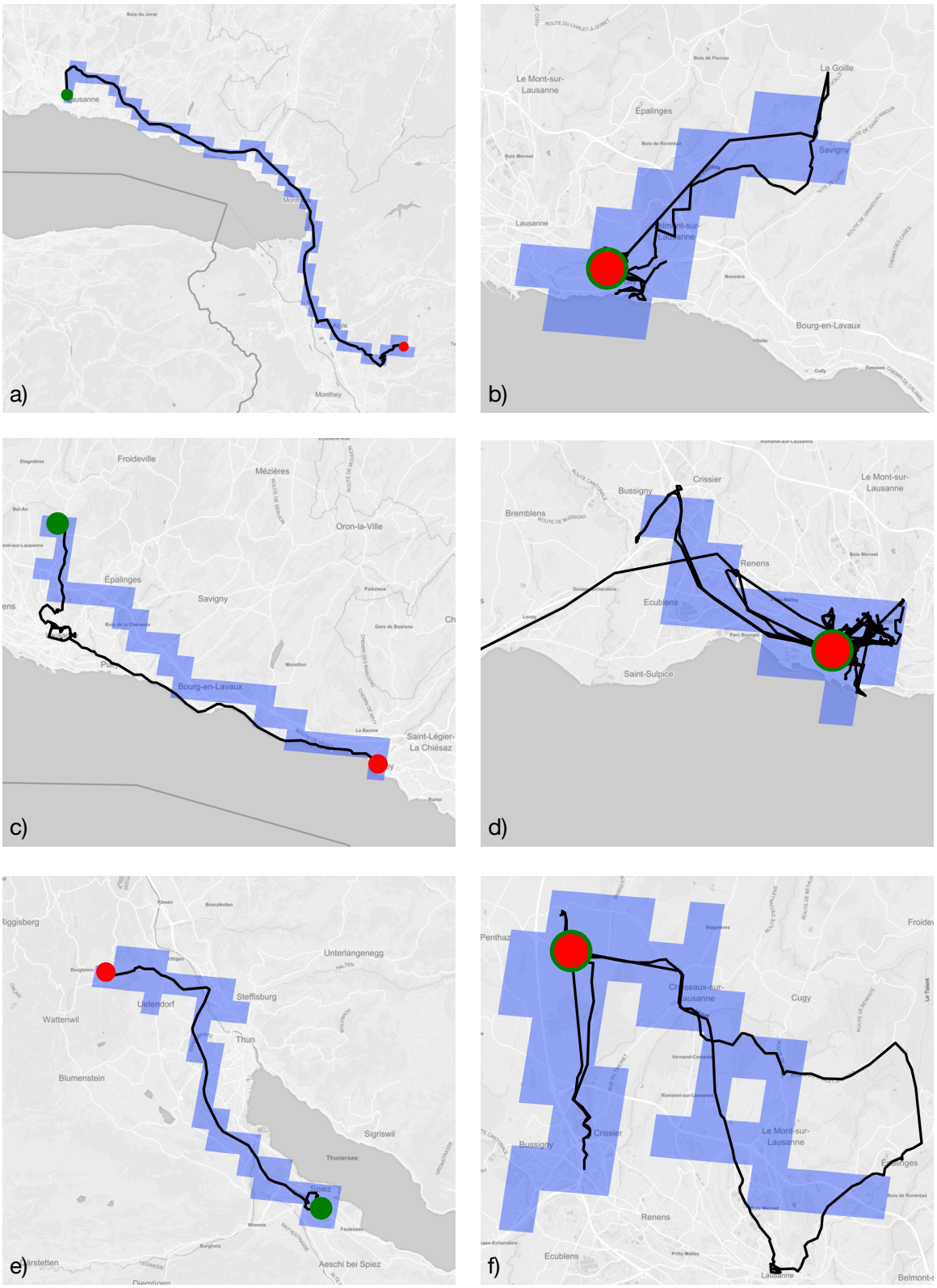


Figure 6.11: Predictions Made with a First Order Markov Chain and an Adaptative Threshold A.

6.7 Conclusion

The growing ubiquity of mobile devices equipped with location aware services is opening the opportunities for novel applications. It is becoming possible to predict human mobility on a large scale today, which can be utilised to answer predictive queries. This has put forth some challenges, which the current prediction and indexing techniques are not well adapted to solve. Through this paper, we introduce an architecture that is capable to predict mobility over long time horizons, index the predicted trajectory and answer the relevant queries. Predicting mobility at distant future allows to devise applications involving high level planning and management. To facilitate this, we present a novel spatial object, 'Representative Trajectory', that accounts for user movements within their ZOIs. Further, we propose means to empirically adapt the extraction of such objects depending on user mobility behaviour. The achieved results over real world mobility traces corroborates our solution, which achieves more than 70 % correct predictions with the best suited extraction method. Our indexing technique, based on inverted indexing, scales with the number of users unlike the tree structures proposed by existing works for predictive indexing. More importantly, we highlight the limits of mobility models, that solely rely on frequently visited places in the context of distant future predictions. Our analysis also shows that the trajectories taken in practice are often complex and as such, the user behaviour has to be taken into account for prediction over distant future. This justifies the requirement for such a spatial object and the indexing technique in order to improve the quality of distant future predictions.

6.8 Future work

Our solution architecture involves several layers, each of which may be enhanced for further developments. For clarity, our initial model considered the problem in its simplest form and our future research will foster improvements. Sets of cells are used to construct representative trajectories and some useful notions may enrich them in order to predict the future movements with a higher accuracy:

1. The model initially relies on a clustering algorithm to find ZOIs. Current techniques often merge overlapping visited places to estimate them, and the resulting areas can be relatively large. Among other possibilities, adding a notion of time at this level may help at discovering ZOIs with fine granularities and thus avoiding some undesirable merges.

2. The model uses Markov chain and its transition probabilities for formulating predictions. In the future, preserving some notion of time, as well as the number of occurrences of a given cell, in the representative trajectories may help at computing these probabilities more accurately and thus making better predictions.

3. The model uses relatively large overlapping cells of one square kilometre to compute representative trajectories. We observed that when reducing the size of the cells, the model starts suffering from the lack of precision, introduced by tracking devices. Since the discretisation of space with a grid is really close from what occurs when a vector image goes through rasterisation, techniques coming from this field, such as anti-aliasing, may help at improving the granularity of the predictions.

Such additions to the model may compromise the system in terms of scalability and is bound to an exhaustive performance analysis. We observed that, the size of our dataset and other publicly available datasets such as GeoLife [134] fits in the memory and are therefore not sufficiently large enough to produce an in-depth quantitative analysis as well as relevant performance measures. A possible solution would be to generate a large synthetic dataset. While such synthetic datasets may be satisfying for performance and scalability measures, they can hardly grasp the complex nature of the human behaviours required for a quantitative analysis. A solution may be to produce synthetic traces based on the mobility models of real users with the aim to reproduce the complex behaviours. Consequently, before investigating these improvements, our priority is to find a suitable way to validate our current findings.

Chapter 7

Conclusion

This conclusion has two purposes. First, we take a step back by recalling our key contributions and by linking them with each others. Second, we highlight some research opportunities that arise from our results, giving some perspective on our on-going and future work.

7.1 Contributions

In this thesis, we have highlighted some of the effects of decentralization on location-aware computing. Our contributions consist in problem definitions, model descriptions, evaluation frameworks, algorithms, and software implementations. Here, we recall some of these contributions by using the three temporal perspectives that structured this thesis: present, past and future. We also highlight how these contributions can interplay and integrate with each other.

7.1.1 Part I: Location-Based Publish and Subscribe (Present)

In Chapter 2, we have answered **Q1** *How can decentralized location-based publish and subscribe systems be tested?* by introducing a testbed that evaluates location-based publish and subscribe systems. This testbed include a trajectory generator that can be used primarily to emulate moving entities in real time, but also to generate large amounts of trajectory data for batch processing. As of this writing, the generator implements two mobility behaviours: the entities move randomly or follow a road network. These tools have been very helpful for verifying the scalability of our location-based publish and subscribe system described in Chapter 3 and to evaluate the trajectory indexing algorithm described in Chapter 5. As progress continues in this field of research, being able to create synthetic datasets and to establish a ground truth becomes critical and our tools address this need. Unfortunately, our testbed cannot yet be used to evaluate trajectory prediction systems. Hu-

man mobility is far more complex than the types of behaviours implemented and, for now, requires real location data. Further research would be necessary to generate very realistic trajectories.

In Chapter 3, we have presented a scalable and reliable architecture for location-based publish and subscribe systems that answers **Q2** *How can a decentralized location-based publish and subscribe system scale horizontally?* This architecture uses a grid in conjunction with consistent hashing to partition the data between several networked computers. As publications and subscriptions can cover an area and hence several partitions, we have introduced a min-wise hashing agreement that selects the partition responsible for computing a match. We have then empirically demonstrated the horizontal scalability of our approach in a cluster made of up to 200 virtual machines. We believe that this agreement mechanism could be of great interest in other areas, such as event stream-processing, especially when events covers intervals on several dimensions. Therefore, we filled a patent application for this mechanism, in order to protect it. However, as the area covered by publications and subscriptions grows, the granularity of the grid determines the number of partitions that need to be contacted. Further research would therefore be needed to reduce the number of messages required to propagate publications and subscriptions that cover very large areas.

Finally, given a routing mechanism that involves multiple network hops in a decentralized system, we have answered **Q3** *How can a decentralized location-based publish and subscribe system be reliable?* by introducing a mechanism that guarantees independence of failure and preserves the scalability of the system. Interestingly, the first step proposed in our architecture routes publications and subscriptions by their identifiers. This scope is ideal for accessing the location histories associated with publications and subscriptions. Therefore, it could constitute an ideal point of entry for integrating trajectory indexing and trajectory prediction.

7.1.2 Part II: Trajectory Indexing (Past)

In Chapter 4, we have used data normalization and data deduplication to answer **Q4** *What are the best access methods for large volumes of trajectories?* Our solution extracts similar segments from sequential data regardless of the type of data. We have refined this indexing method in Chapter 5 by focusing solely on trajectory indexing. This time, instead of using segments obtained by deduplication to construct the inverted index, we have introduced a fingerprinting method inspired by data winnowing. In contrast to the non-overlapping segments extracted by deduplication, the trajectory fingerprints extracted by winnowing overlap and hence come with stronger guarantees regarding the detection of similarities in the data. In addition, the fingerprints obtained with our method combine a geohash and a regular hash, enabling us to further discriminate the data in the context of very dense trajectory datasets.

In Chapter 4 and 5, we have answered **Q5** *How can we qualitatively assess an index of trajectories?* by introducing a framework that uses precision and recall to decide on the ideal extent of the

normalization that should be applied to the data. With this framework, we have demonstrated that our method makes a very good tradeoff between the quality of the results and the gain in terms of performances. Therefore, we also filled a patent application for this method, in order to protect it. Because of the lack of real trajectory datasets, we have evaluated our approach with a large synthetic dataset created with the generator introduced in Chapter 2. Further research would be necessary to validate our results with a real trajectory dataset. However, such datasets are not publicly available today, mainly due to privacy concerns.

Finally, in Chapter 5, we have answered **Q6** *How can a very large index of trajectories be decentralized?* by introducing an inverted index that contains the hash sums of the segments obtained by performing deduplication on normalized data. Interestingly, such an inverted index does not need to be reorganized or rebalanced as it grows and can be easily sharded. In addition, such an index can also be built as the data arrives in the system. These properties make our approach ideal for continuously indexing the data coming from a location-based publish and subscribe system, such as the one presented in Chapter 3. Our method is limited in the sense that true positive results can be missed. However, this limitation is also an opportunity because it is common to trade exactness for performances when dealing with very large datasets.

7.1.3 Part III: Trajectory Prediction (Future)

In Chapter 6, we have introduced the problem of predicting future trajectories. We have answered **Q7** *How can we predict future trajectories on the basis of past trajectories?* by defining a model that relies on the prediction of the next point of interest and a discretized representation of past trajectories to infer the future trajectories that users will probably follow. On the basis of this model, we have introduced a testbed that can be used to answer **Q8** *How can we qualitatively assess the accuracy of a prediction system for future trajectories?* With this testbed, we have evaluated several approaches for constructing representative trajectories. Our solution relies extensively on an inverted index that shares some common properties with the trajectory index presented in Chapter 5. As our model for making trajectory predictions is embarrassingly parallel and it can be used to answer **Q9** *How can a prediction system for future trajectories be decentralized?* The results we have presented in this chapter also correspond to a preliminary solution. A better model would probably associate different probabilities with the discretized locations that form a representative trajectory. As a consequence, further research would be necessary to refine the model and validate its horizontal scalability.

7.2 On-going and Future Work

As highlighted in Section 7.1, the distinct problems addressed by our contributions closely interplay with each other. Here, we discuss some research opportunities that could arise from addressing these problems generically at the level of a stream-processing solution. Furthermore, we highlight some research avenues that could be addressed at a protocols and at a tooling level.

7.2.1 Spatio-Temporal Stream-Processing

In the context of the Internet of Things, decentralized solutions that process large unbounded event streams in near real-time have recently gained much attention. For example, Apache Spark provides a rich map/reduce abstraction that enables developers to create applications that process data streams in a discretized fashion, often referred to as micro-batches. Another alternative, called Apache Kafka, provides a flexible producer-consumer abstraction largely inspired by the publish-subscribe model. As these solutions continue to evolve, they tend to overlap and converge in terms of features, making it difficult for developers to choose between them. It might be tempting to rely on these solutions to solve location-aware problems. Therefore, we highlight some of the key limitations that could motivate further research in spatio-temporal stream-processing.

Location-Aware Abstractions

In Chapter 3, we described abstractions for location-based publish and subscribe systems. Unfortunately, solutions such as Apache Spark or Apache Kafka map each event to exactly one partition because events are considered as points in time. In contrast, location-based publish and subscribe systems deals with spatio-temporal intervals and, as a result, publications and subscriptions can overlap with many partitions. As the partitioning abstraction is at the core of many low-level decisions, it illustrates well one of the main limitations associated with existing stream-processing solutions. Further research is needed to understand the consequence of a one-to-many relationship between events and partitions. It might also be interesting to devise higher level abstractions to generically address spatio-temporal stream-processing problems, such as traffic congestion detection or public transport monitoring.

Moving Processing Closer to the Data

Existing decentralized stream-processing solutions tend to move the data to a computer that is then responsible for processing it. In Chapter 3, we have done exactly the opposite. In order to compute matches with a high throughput and a low latency, we have moved the computation closer to

the route followed by the data. Good location-aware abstractions could help to generically move complex computations closer to the data. For instance, as the path followed by the data involves a geographic context, this context could be used to instantiate location-aware processing functions for transforming or enriching the data stream in near realtime. Further research is needed to identify all the use cases that would benefit from such an approach.

Consistency, Availability and Partition Tolerance Tradeoffs

Solutions, such as Apache Spark or Apache Kafka, assume that network partitions are rare in data centers. These solutions usually relax availability in order to remain consistent. For example, in case of network partition, the only nodes which remain available are those who belong to the partition that attains a quorum. As mentioned in Section 1.2.3, consistency is often traded to guarantee a very low latency [17, 3]. For many IoT use cases, such as vehicle to vehicle communication, low latency is a requirement and is ensured with a mobile or vehicular ad hoc network (VANET). In addition, the emergence of location-aware video games will probably involve similar latency requirements. Further research might therefore be necessary to understand the consistency requirements of future location-aware applications and to verify if such requirements could be satisfied with a decentralized solution.

7.2.2 IoT Protocols

IoT protocols are of great importance, despite the fact that they have not been discussed in this thesis. For instance, in order to make systems interoperable, the Advanced Message Queuing Protocol (AMQP) standardizes several common messaging abstractions, such as queuing or topic-based publish and subscribe messaging [124]. In contrast, the Message Queue Telemetry Transport (MQTT) standardizes a very simple and lightweight topic-based publish and subscribe abstraction and shines in terms of transport efficiency and security [5]. Interestingly, though both of these standards took opposite directions, they are both used to address use cases that largely overlap. This probably indicates that IoT protocols are still in their infancy and that the requirements associated with IoT have to be further characterized. For example, the following questions could be asked: What kind of reliability guarantee should be provided at the level of the protocol? What is the effect of a messaging abstraction on energy consumption? What kind of security mechanism should be provided at the level of the protocol?

7.2.3 Reproducible Research

As we implemented location-aware systems, we faced the difficulty of reproducing prior results. We also acknowledge that reproducing our own results in a large cluster would necessitate an important time investment. Several reasons can be at the origin of this reproducibility issue: The source code is not always publicly available; the descriptions of the algorithms are not sufficient to reproduce the system; the descriptions provided to reproduce the results are incomplete; the datasets used in the experiments are not publicly available; the execution environments are difficult to setup and expensive; and the authors are now working on completely different topics. In Chapter 2, we have introduced a testbed for location-based publish and subscribe systems, which enabled us to verify the scalability of our system. Similar testbeds and standard datasets could probably be used for evaluating and benchmarking different solutions that address the same problems. Such tools and datasets will become more and more critical for addressing the lack of reproducibility that affects research in location-aware systems, and investing in them is probably the way to go.

Bibliography

[1] Apache lucene. http://lucene.apache.org/.

[2] JTS topology suite. http://www.vividsolutions.com/jts/.

[3] Daniel J Abadi. Consistency tradeoffs in modern distributed database system design. *Computer-IEEE Computer Magazine*, 45(2):37, 2012.

[4] Bhuvan Bamba, Ling Liu, Arun Iyengar, and Philip S Yu. Safe region techniques for fast spatial alarm evaluation. 2008.

[5] Andrew Banks and Rahul Gupta. Mqtt version 3.1. 1. *OASIS standard*, 29, 2014.

[6] Paul Baran. On distributed communications networks. *IEEE transactions on Communications Systems*, 12(1):1–9, 1964.

[7] Paolo Bellavista, Antonio Corradi, Mario Fanelli, and Luca Foschini. A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys (CSUR)*, 44(4):24, 2012.

[8] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[9] Deepavali Bhagwat, Kave Eshghi, and Pankaj Mehra. Content-based document routing and index partitioning for scalable similarity-based searches in a large corpus. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 105–112. ACM, 2007.

[10] Nikolaj Bjørner, Andreas Blass, and Yuri Gurevich. Content-dependent chunking for differential compression, the local maximum approach. *Journal of Computer and System Sciences*, 76(3-4):154–203, 2010.

[11] Deepak R Bobbarjung, Suresh Jagannathan, and Cezary Dubnicki. Improving duplicate elimination in storage systems. *ACM Transactions on Storage (TOS)*, 2(4):424–448, 2006.

[12] Behnaz Bostanipour and Benoît Garbinato. Improving neighbor detection for proximity-based mobile applications. In *Network Computing and Applications (NCA), 2013 12th IEEE International Symposium on*, pages 177–182. IEEE, 2013.

[13] Behnaz Bostanipour and Benoît Garbinato. Using virtual mobile nodes for neighbor detection in proximity-based mobile applications. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 9–16. IEEE, 2014.

[14] Behnaz Bostanipour and Benoît Garbinato. Effective and efficient neighbor detection for proximity-based mobile applications. *Computer Networks*, 79:216–235, 2015.

[15] Behnaz Bostanipour and Benoît Garbinato. Neighbor detection based on multiple virtual mobile nodes. In *Parallel, Distributed, and Network-Based Processing (PDP), 2016 24th Euromicro International Conference on*, pages 322–327. IEEE, 2016.

[16] Behnaz Bostanipour, Benoît Garbinato, and Adrian Holzer. Spotcast–a communication abstraction for proximity-based mobile applications. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, pages 121–129. IEEE, 2012.

[17] Eric Brewer. Cap twelve years later: How the" rules" have changed. *Computer*, 45(2):23–29, 2012.

[18] Sergey Brin, James Davis, and Hector Garcia-Molina. Copy detection mechanisms for digital documents. In *ACM SIGMOD Record*, volume 24, pages 398–409. Acm, 1995.

[19] Thomas Brinkhoff. Generating network-based moving objects. In *Scientific and Statistical Database Management, 2000. Proceedings. 12th International Conference on*, pages 253–255. IEEE, 2000.

[20] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[21] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

[22] V. Prasad Chakka, Adam Everspaugh, and Jignesh M. Patel. Indexing large trajectory data sets with seti. In *CIDR*, 2003.

[23] V Prasad Chakka, Adam C Everspaugh, and Jignesh M Patel. Indexing large trajectory data sets with seti. *Ann Arbor*, 1001(48109-2122):12, 2003.

[24] Bertil Chapuis and Benoît Garbinato. Knowledgeable chunking. In *International Conference on Networked Systems*, pages 456–460. Springer, Cham, 2015.

[25] Bertil Chapuis and Benoît Garbinato. Scaling and load testing location-based publish and subscribe. In *37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2543–2546. IEEE, 2017.

[26] Bertil Chapuis and Benoît Garbinato. Geodabs: Trajectory indexing meets fingerprinting at scale. In *38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018.

[27] Bertil Chapuis, Benoît Garbinato, and Periklis Andritsos. Throughput: A key performance measure of content-defined chunking algorithms. In *36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 7–12. IEEE, 2016.

[28] Bertil Chapuis, Benoît Garbinato, and Periklis Andritsos. An efficient type-agnostic approach for finding sub-sequences in data. In *19th International Conference on High Performance Computing and Communications; 15th International Conference on Smart City; 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 270–277. IEEE, 2017.

[29] Bertil Chapuis, Benoît Garbinato, and Lucas Mourot. A horizontally scalable and reliable architecture for location-based publish-subscribe. In *36th Symposium on Reliable Distributed Systems (SRDS)*, pages 74–83. IEEE, 2017.

[30] Bertil Chapuis, Arielle Moro, Vaibhav Kulkarni, and Benoît Garbinato. Capturing complex behaviour for predicting distant future trajectories. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, pages 64–73. ACM, 2016.

[31] Su Chen, Christian S Jensen, and Dan Lin. A benchmark for evaluating moving object indexes. *Proceedings of the VLDB Endowment*, 1(2):1574–1585, 2008.

[32] Xiaoyan Chen, Ying Chen, and Fangyan Rao. An efficient spatial publish/subscribe system for intelligent location-based services. In *Proceedings of the 2nd international workshop on Distributed event-based systems*, pages 1–6. ACM, 2003.

[33] Mauro Cherubini, Alexandre Meylan, Bertil Chapuis, Mathias Humbert, Igor Bilogrevic, and Kévin Huguenin. Towards usable checksums: Automating the integrity verification of web downloads for the masses. In *25th ACM Conference on Computer and Communications Security*. ACM, 2018.

[34] Meng-Fen Chiang, Wen-Yuan Zhu, Wen-Chih Peng, and S Yu Philip. Distant-time location prediction in low-sampling-rate trajectories. In *2013 IEEE 14th International Conference on Mobile Data Management*, volume 1, pages 117–126. IEEE, 2013.

[35] Bram Cohen. The bittorrent protocol specification, 2008.

[36] Gianpaolo Cugola and Jose Enrique Munoz de Cote. On introducing location awareness in publish-subscribe middleware. In *25th IEEE International Conference on Distributed Computing Systems Workshops*, pages 377–382. IEEE, 2005.

[37] Gianpaolo Cugola and Alessandro Margara. High-performance location-aware publish-subscribe on gpus. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 312–331. Springer, 2012.

[38] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6):205–220, 2007.

[39] Pranita Dewan, Raghu Ganti, and Mudhakar Srivatsa. Som-tc: Self-organizing map for hierarchical trajectory clustering. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 1042–1052. IEEE, 2017.

[40] Christian Düntgen, Thomas Behr, and Ralf Hartmut Güting. Berlinmod: a benchmark for moving object databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 18(6):1335–1368, 2009.

[41] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Tech. Report CD-TR 94/64, Information Systems Department, Technical University of Vienna, 1994.

[42] Kave Eshghi and Hsiu Khuern Tang. A framework for analyzing and improving content-based chunking algorithms. *Hewlett-Packard Labs Technical Report TR*, 30:2005, 2005.

[43] P Th Eugster, Benoît Garbinato, and Adrian Holzer. Location-based publish/subscribe. In *Fourth IEEE International Symposium on Network Computing and Applications*, pages 279–282. IEEE, 2005.

[44] Patrick Eugster, Benoit Garbinato, and Adrian Holzer. Pervaho: A development & test platform for mobile ad hoc applications. In *Mobile and Ubiquitous Systems-Workshops, 2006. 3rd Annual International Conference on*, pages 1–5. IEEE, 2006.

[45] Patrick Eugster, Benoît Garbinato, and Adrian Holzer. Design and implementation of the pervaho middleware for mobile context-aware applications. In *e-Technologies, 2008 International MCETECH Conference on*, pages 125–135. IEEE, 2008.

[46] Patrick Eugster, Benoît Garbinato, and Adrian Holzer. Pervaho: A specialized middleware for mobile context-aware applications. *Electronic Commerce Research*, 9(4):245–268, 2009.

[47] Patrick Th Eugster, Benoit Garbinato, and Adrian Holzer. Middleware support for context-aware applications. In *Middleware for Network Eccentric and Mobile Applications*, pages 305–322. Springer, 2009.

[48] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. *Machine learning*, 31(1):1–38, 2004.

[49] Raphael A Finkel and Jon Louis Bentley. Quad trees a data structure for retrieval on composite keys. *Acta informatica*, 4(1):1–9, 1974.

[50] George Forman, Kave Eshghi, and Stephane Chiocchetti. Finding similar files in large document repositories. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 394–400. ACM, 2005.

[51] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Show me how you move and i will tell you who you are. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Security and Privacy in GIS and LBS*, pages 34–41. ACM, 2010.

[52] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núnez del Prado Cortez. Towards temporal mobility markov chains. In *1st International Workshop on Dynamicity Collocated with OPODIS 2011, Toulouse, France*, pages 2–pages, 2011.

[53] Sébastien Gambs, Marc-Olivier Killijian, and Miguel Núñez del Prado Cortez. Next place prediction using mobility markov chains. In *Proceedings of the First Workshop on Measurement, Privacy, and Mobility*, page 3. ACM, 2012.

[54] Amir Gandomi and Murtaza Haider. Beyond the hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, 35(2):137–144, 2015.

[55] Benoît Garbinato, Adrian Holzer, and François Vessaz. Six-shot broadcast: A context-aware algorithm for efficient message diffusion in manets. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 625–638. Springer, 2008.

[56] Benoit Garbinato, Adrian Holzer, and François Vessaz. Context-aware broadcasting approaches in mobile ad hoc networks. *Computer Networks*, 54(7):1210–1228, 2010.

[57] Benoit Garbinato, Adrian Holzer, and Francois Vessaz. Six-shot multicast: A location-aware strategy for efficient message routing in manets. In *Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on*, pages 1–9. IEEE, 2010.

[58] Benoît Garbinato and Philippe Rupp. From ad hoc networks to ad hoc applications. In *Proceedings of the 7th International Conference on Telecommunications*, pages 145–149. Citeseer, 2003.

[59] Julien Gascon-Samson, Franz-Philippe Garcia, Bettina Kemme, and Jörg Kienzle. Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud. In *Distributed Computing Systems (ICDCS), 2015 IEEE 35th International Conference on*, pages 486–496. IEEE, 2015.

[60] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News*, 33(2):51–59, 2002.

[61] Chong Yang Goh, Justin Dauwels, Nikola Mitrovic, Muhammad Tayyab Asif, Ali Oran, and Patrick Jaillet. Online map-matching based on hidden markov model for real-time traffic sensing applications. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 776–781. IEEE, 2012.

[62] Rachid Guerraoui and Luis Rodrigues. *Introduction to reliable distributed programming*. Springer Science & Business Media, 2006.

[63] Long Guo, Dongxiang Zhang, Guoliang Li, Kian-Lee Tan, and Zhifeng Bao. Location-aware pub/sub system: When continuous moving queries meet dynamic event streams. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 843–857. ACM, 2015.

[64] Antonin Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

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

[66] Mahady Hasan, Muhammad Aamir Cheema, Xuemin Lin, and Ying Zhang. Efficient construction of safe regions for moving knn queries over dynamic datasets. In *International Symposium on Spatial and Temporal Databases*, pages 373–379. Springer, 2009.

[67] Nevin Heintze et al. Scalable document fingerprinting. In *1996 USENIX workshop on electronic commerce*, volume 3, 1996.

[68] Abdeltawab M Hendawi, Jie Bao, Mohamed F Mokbel, and Mohamed Ali. Predictive tree: An efficient index for predictive queries on road networks. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1215–1226. IEEE, 2015.

[69] Alan R Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[70] Adrian Holzer, Patrick Eugster, and Benoit Garbinato. Evaluating implementation strategies for location-based multicast addressing. *IEEE Transactions on Mobile Computing*, 12(5):855–867, 2013.

[71] Adrian Holzer, Patrick Eugster, and Benoıˆt Garbinato. Alps–adaptive location-based publish/subscribe. *Computer Networks*, 56(12):2949–2962, 2012.

[72] Adrian Holzer, Sten Govaerts, Jan Ondrus, Andrii Vozniuk, David Rigaud, Benoît Garbinato, and Denis Gillet. Speakup–a mobile app facilitating audience interaction. In *International Conference on Web-Based Learning*, pages 11–20. Springer, 2013.

[73] Adrian Holzer, François Vessaz, Samuel Pierre, and Benoît Garbinato. Plan-b: Proximity-based lightweight adaptive network broadcasting. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 265–270. IEEE, 2011.

[74] Jiafeng Hu, Reynold Cheng, Dingming Wu, and Beihong Jin. Efficient top-k subscription matching for location-aware publish/subscribe. In *International Symposium on Spatial and Temporal Databases*, pages 333–351. Springer, 2015.

[75] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[76] Weihuang Huang, Guoliang Li, Kian-Lee Tan, and Jianhua Feng. Efficient safe-region construction for moving top-k spatial keyword queries. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 932–941. ACM, 2012.

[77] Hoyoung Jeung, Qing Liu, Heng Tao Shen, and Xiaofang Zhou. A hybrid prediction model for moving objects. In *2008 IEEE 24th International Conference on Data Engineering*, pages 70–79. Ieee, 2008.

[78] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997.

[79] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. *Computer Networks*, 31(11):1203–1213, 1999.

[80] Peter Karich and S Schröder. Graphhopper. *http://www.graphhopper.com, last accessed*, 4(2):15, 2014.

[81] Dong-Oh Kim, Kang-Jun Lee, Dong-Suk Hong, and Ki-Joon Han. An efficient indexing technique for location prediction of moving objects. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 1–9. Springer, 2007.

[82] Sven Kosub. A note on the triangle inequality for the jaccard distance. *arXiv preprint arXiv:1612.02696*, 2016.

[83] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, pages 1–7, 2011.

[84] Erik Kruus, Cristian Ungureanu, and Cezary Dubnicki. Bimodal content defined chunking for backup streams. In *FAST*, pages 239–252, 2010.

[85] Vaibhav Kulkarni, Bertil Chapuis, and Benoît Garbinato. Privacy-preserving location-based services by using intel sgx. In *Proceedings of the First International Workshop on Human-centered Sensing, Networking, and Systems*, pages 13–18. ACM, 2017.

[86] Vaibhav Kulkarni, Arielle Moro, Bertil Chapuis, and Benoît Garbinato. Extracting hotspots without a-priori by enabling signal processing over geospatial data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 79. ACM, 2017.

[87] Vaibhav Kulkarni, Arielle Moro, Bertil Chapuis, and Benoît Garbinato. Capstone: Mobility modeling on smartphones to achieve privacy by design. In *International Conference on Trust, Security And Privacy In Computing And Communications (TrustCom)*. IEEE, 2018.

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

[89] J. K. Laurila, Daniel Gatica-Perez, I. Aad, Blom J., Olivier Bornet, Trinh-Minh-Tri Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Pervasive Computing*, 2012.

[90] Juha K Laurila, Daniel Gatica-Perez, Imad Aad, Olivier Bornet, Trinh-Minh-Tri Do, Olivier Dousse, Julien Eberle, Markus Miettinen, et al. The mobile data challenge: Big data for mobile computing research. In *Pervasive Computing*, number EPFL-CONF-192489, 2012.

[91] Joseph J LaViola. Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proceedings of the workshop on Virtual environments 2003*, pages 199–206. ACM, 2003.

[92] Daniel Lemire, Owen Kaser, Nathan Kurz, Luca Deri, Chris O'Hara, François Saint-Jacques, and Gregory Ssi-Yan-Kai. Roaring bitmaps: Implementation of an optimized software library. *arXiv preprint arXiv:1709.07821*, 2017.

[93] Guoliang Li, Yang Wang, Ting Wang, and Jianhua Feng. Location-aware publish/subscribe. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 802–810. ACM, 2013.

[94] Guanlin Lu, Yu Jin, and David HC Du. Frequency based chunking for data de-duplication. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 287–296. IEEE, 2010.

[95] Xin Lu, Erik Wetter, Nita Bharti, Andrew J Tatem, and Linus Bengtsson. Approaching the limit of predictability in human mobility. *Scientific reports*, 3:srep02923, 2013.

[96] Udi Manber et al. Finding similar files in a large file system. In *Usenix Winter*, volume 94, pages 1–10, 1994.

[97] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[98] Wesley Mathew, Ruben Raposo, and Bruno Martins. Predicting future locations with hidden markov models. In *Proceedings of the 2012 ACM conference on ubiquitous computing*, pages 911–918. ACM, 2012.

[99] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[100] Dutch T Meyer and William J Bolosky. A study of practical deduplication. *ACM Transactions on Storage (TOS)*, 7(4):14, 2012.

[101] Mohamed F Mokbel, Louai Alarabi, Jie Bao, Ahmed Eldawy, Amr Magdy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. Mntg: an extensible web-based traffic generator. In *International Symposium on Spatial and Temporal Databases*, pages 38–55. Springer, 2013.

[102] Athicha Muthitacharoen, Benjie Chen, and David Mazieres. A low-bandwidth network file system. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 174–187. ACM, 2001.

[103] Julio C Navas and Tomasz Imielinski. Geocast—geographic addressing and routing. In *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pages 66–76. ACM, 1997.

[104] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 336–343. ACM, 2009.

[105] Gustavo Niemeyer. Geohash, 2008.

[106] Alexander Osterwalder, Yves Pigneur, Greg Bernarda, and Alan Smith. *Value proposition design*. Campus Verlag, 2015.

[107] Jignesh M Patel, Yun Chen, and V Prasad Chakka. Stripes: an efficient index for predicted trajectories. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 635–646. ACM, 2004.

[108] Nikos Pelekis, Stylianos Sideridis, Panagiotis Tampakis, and Yannis Theodoridis. Hermoupolis: a semantic trajectory generator in the data science era. *SIGSPATIAL Special*, 7(1):19–26, 2015.

[109] Dieter Pfoser, Christian S Jensen, Yannis Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *VLDB*, pages 395–406, 2000.

[110] Sean Quinlan and Sean Dorward. Venti: A new approach to archival storage. In *FAST*, volume 2, pages 89–101, 2002.

[111] Michael O Rabin. *Fingerprinting by random polynomials*. Center for Research in Computing Techn., Aiken Computation Laboratory, Univ., 1981.

[112] Philippe Rigaux, Michel Scholl, and Agnes Voisard. *Spatial databases: with application to GIS*. Morgan Kaufmann, 2001.

[113] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[114] Simonas Šaltenis, Christian S Jensen, Scott T Leutenegger, and Mario A Lopez. *Indexing the positions of continuously moving objects*, volume 29. ACM, 2000.

[115] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85. ACM, 2003.

[116] Vinay Setty, Maarten van Steen, Roman Vitenberg, and Spyros Voulgaris. Poldercast: Fast, robust, and scalable architecture for p2p topic-based pub/sub. In *Middleware*, 2012.

[117] N Sornin, M Luis, T Eirich, T Kramp, and O Hersent. Lorawan specification. *LoRa alliance*, 2015.

[118] Mudhakar Srivatsa, Raghu Ganti, and Prasant Mohapatra. On the limits of subsampling of location traces. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pages 1032–1041. IEEE, 2017.

[119] Bo Tang, Man Lung Yiu, Kyriakos Mouratidis, and Kai Wang. Efficient motif discovery in spatial trajectories using discrete fréchet distance. EDBT, 2017.

[120] Yufei Tao, Christos Faloutsos, Dimitris Papadias, and Bin Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 611–622. ACM, 2004.

[121] Yufei Tao, Dimitris Papadias, and Jimeng Sun. The tpr*-tree: an optimized spatio-temporal access method for predictive queries. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 790–801. VLDB Endowment, 2003.

[122] David G Thaler and Chinya V Ravishankar. Using name-based mappings to increase hit rates. *IEEE/ACM Transactions on Networking (TON)*, 6(1):1–14, 1998.

[123] François Vessaz, Benoît Garbinato, Arielle Moro, and Adrian Holzer. Developing, deploying and evaluating protocols with manetlab. In *Networked Systems*, pages 89–104. Springer, 2013.

[124] Steve Vinoski. Advanced message queuing protocol. *IEEE Internet Computing*, 10(6), 2006.

[125] Xiang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Wei Wang. Ap-tree: efficiently support location-aware publish/subscribe. *The VLDB Journal—The International Journal on Very Large Data Bases*, 24(6):823–848, 2015.

[126] Dingming Wu, Man Lung Yiu, Christian S Jensen, and Gao Cong. Efficient continuously moving top-k spatial keyword query processing. In *2011 IEEE 27th International Conference on Data Engineering*, pages 541–552. IEEE, 2011.

[127] George Xylomenos, Xenofon Vasilakos, Christos Tsilopoulos, Vasilios A Siris, and George C Polyzos. Caching and mobility support in a publish-subscribe internet architecture. *IEEE Communications Magazine*, 50(7):52–58, 2012.

[128] Yutaka Yanagisawa. Predictive indexing for position data of moving objects in the real world. In *Transactions on Computational Science VI*, pages 77–94. Springer, 2009.

[129] Byoung-Kee Yi, HV Jagadish, and Christos Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 201–208. IEEE, 1998.

[130] Josh Jia-Ching Ying, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S Tseng. Semantic trajectory mining for location prediction. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 34–43. ACM, 2011.

[131] Minghe Yu, Guoliang Li, Ting Wang, Jianhua Feng, and Zhiguo Gong. Efficient filtering algorithms for location-aware publish/subscribe. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):950–963, 2015.

[132] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[133] Ye Zhao, Kyungbaek Kim, and Nalini Venkatasubramanian. Dynatops: a dynamic topic-based publish/subscribe architecture. In *DEBS*, 2013.

[134] Yu Zheng, Hao Fu, Xing Xie, Wei-Ying Ma, and Quannan Li. Geolife gps trajectory dataset-user guide, 2011.

[135] Yu Zheng, Xing Xie, and Wei-Ying Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.