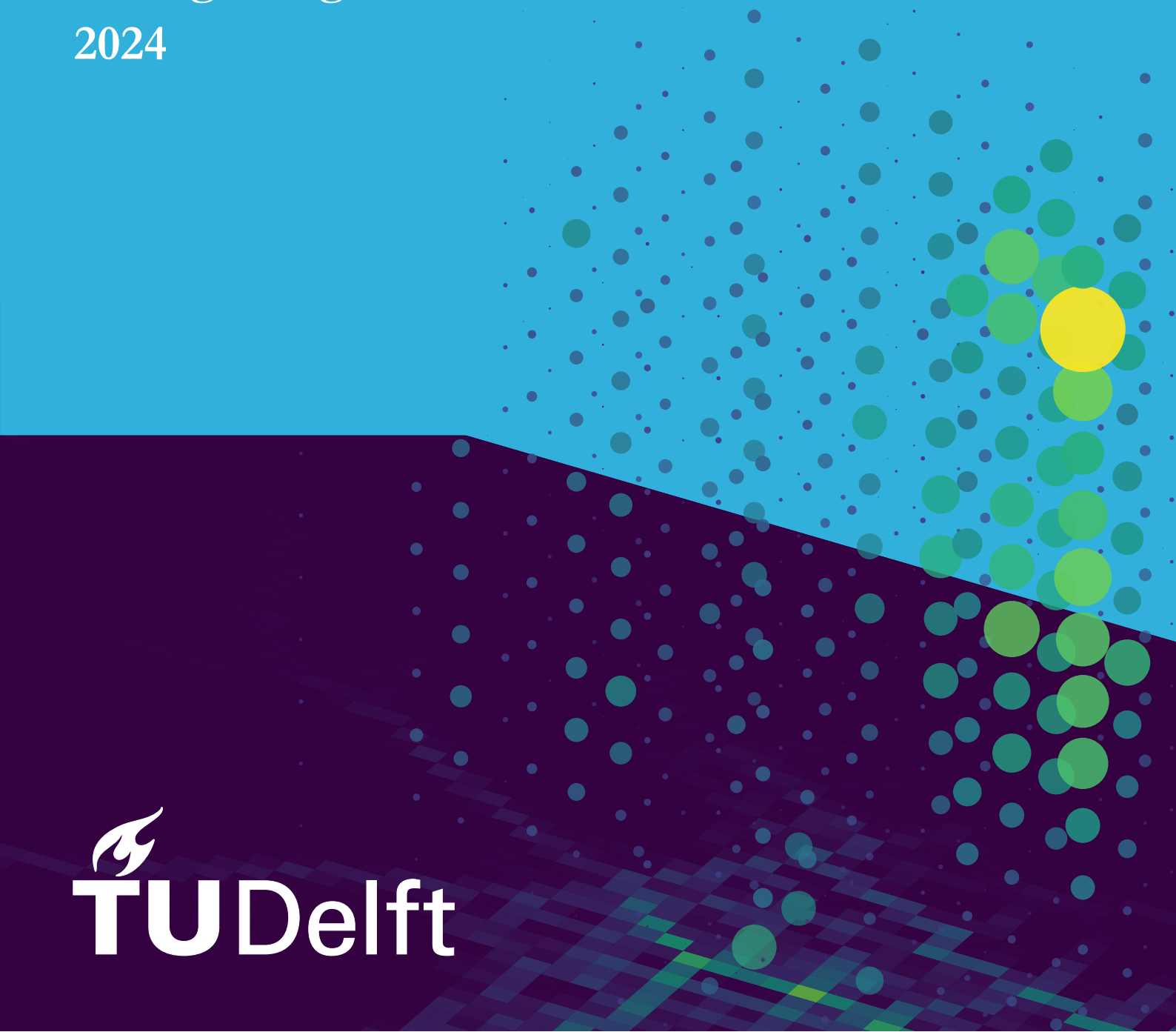
Msc Thesis In Geomatics

Towards Adaptive Trajectory Data Management: Modeling, Accessing, Distributing, and Query Optimization in Distributed Database

Sicong Gong

2024



TU Delft

MSc Thesis in Geomatics

Towards Adaptive Trajectory Data Management: Modelling, Accessing, Distributing, and Query Optimization in Distributed Database

Sicong Gong

June 2024

A thesis submitted to the Delft University of Technology in partial fulfilment of the requirements for the degree of Master of Science in Geomatics

Sicong Gong: *Towards Adaptive Trajectory Data Management: Modelling, Accessing, Distributing, and Query Optimization in Distributed Database* (2024)

The work in this thesis was carried out in the:

Geo-Database Management Center
Delft University of Technology

Supervisors:	Dr.Ir. Martijn Meijers
	Drs. Wilko Quak
Co-reader:	Dr. Ken Arroyo Ohori

Abstract

With the proliferation of the Internet of Things (IoT) infrastructure, the trajectory data is dynamically emerging. This data originates from a variety of moving objects, containing big volumes of multi-dimensional information such as space, time, semantics etc. The underlying information can be potentially applied to create added value through scientific research, decision-making, emergency management etc.

However, due to the special properties of the trajectory data, namely high frequency, cardinality, dimensionality, heterogeneity etc., traditional data management systems face difficulties in handling such data. Even though some distributed solutions or big data solutions exist in other fields, they are not designed considering the modelling, accessing, distributing and querying characteristics of this special spatio-temporal data.

Given the spatial data management problems, a clustering/indexing solution for high dimensional Point Cloud (PC) by Space-filling Curve (SFC) considering the heterogeneous data spatial distribution has been developed, advocated and validated by a series of research finished at the Geo-Database Management Centre (GDMC), Delft University of Technology (TU Delft).

However, it is uncertain whether the framework can be extended to other types of space-related phenomena. Furthermore, whether distributed database techniques can be utilized remains to be explored and what adjustments should be made is still unclear. To some extent, this thesis is an expanded study based on the PC research mentioned above.

To address these data management challenges, this thesis focuses on trajectory data modelling and compression, indexing and clustering, partitioning and distributing. Also, the querying strategies are studied. More specifically, the three main results of this thesis are:

(1) **Model the trajectory as the sequence split by semantic attributes and spatio-temporal cube**. This modelling takes the proximity (locality) preservation and trajectory preservation into account at the same time, resulting in a balanced level of flexibility and aggregation, mitigating the storage burden by row-wise compression. For different subdivision resolutions (depth of Octree for space partitioning), the compression ratio can be up to 10.

(2) **Access the trajectory data by SFC**. The SFC indexing method maps the 3D (high dimensional) indices to 1D (low dimensional) indices, overcoming the contradiction between high dimensionality and high cardinality. Adaptive Octree is used to mitigate the heterogeneity of the trajectory data. Based on the experiment results, the optimal tree depth is 4 or 5. The query optimization (specifically, the range merging technique) is also preliminarily explored.

(3) **Distribute the data on the distributed machines**. This distributed deployment results in higher (nearly linearly in the experiment) scalability (horizontal expansion of disk, memory and Central Processing Unit (CPU) resources) and speed-up. The SFC-based distributing strategy results in a better load-balancing. However, due to the lack of flexibility of the distributed database platform used (specifically, Greenplum), the localization of data and computation (such as local aggregation) is limited.

Acknowledgements

The nearly one-year graduation project was like a marathon. In the middle (even for the last few weeks), I often felt that the remaining distance was endless. Thanks to everyone that helped me, so that I could hold on until the end.

I would like to express my endless gratitude to two of my supervisors, Dr.Ir. Martijn Meijers and Drs. Wilko Quak, for "adopting" me. Without their kind help, the framework of this research would not have been formed. They were tolerant of many of my preliminary and unrealistic (at first glance) ideas and transformed them into interesting and effective solutions through their extensive experience. The whimsical nature of these ideas added joy to our meetings and the resulting hearty laughter was a great relief to my nervousness.

I would also give special thanks to my co-reader Dr. Ken Arroyo Ohori and the delegate of examiners Dr. Liangliang Nan especially for their sacrifice for coordinating P5 time. Besides, their careful examination and enlightening suggestions effectively enhanced the integrity and seriousness of the thesis.

Two years of studying abroad were like an illusory dream. Homesickness, loneliness and self-doubt were "incurable diseases" that often kept me awake during the whole night. I was also thankful for my family and friends, the video calls with them empowered me, bound us together despite the distance and provided much-needed refreshment.

Finally, I would like to acknowledge my personal growth throughout my study in the Geomatics program and this graduation project.

Sicong Gong
June, 2024, Delft

Contents

List of Figures

List of Tables

Acronyms

1. Introduction

Chapter Contents

This chapter gives an introduction to this thesis.

Section 1.1 introduces the background: more and more trajectory data are collected, different kinds of applications are emerging and new data management technologies are evolving. This section proves the significance of this thesis.

Section 1.2 introduces the previous researches done at GDMC, TU Delft: SFC is used to manage multi-dimensional data, histogram tree is used to deal with the uneven data spatial distribution and convex polytope is used to do shape querying. This section gives the contexts (management of PC, a similar spatial data to trajectory) for this thesis.

Section 1.3 presents the motivation: the difficulties in managing trajectory data because of its properties, the objectives of the solutions and the scope of this thesis while less-relevant contents are omitted.

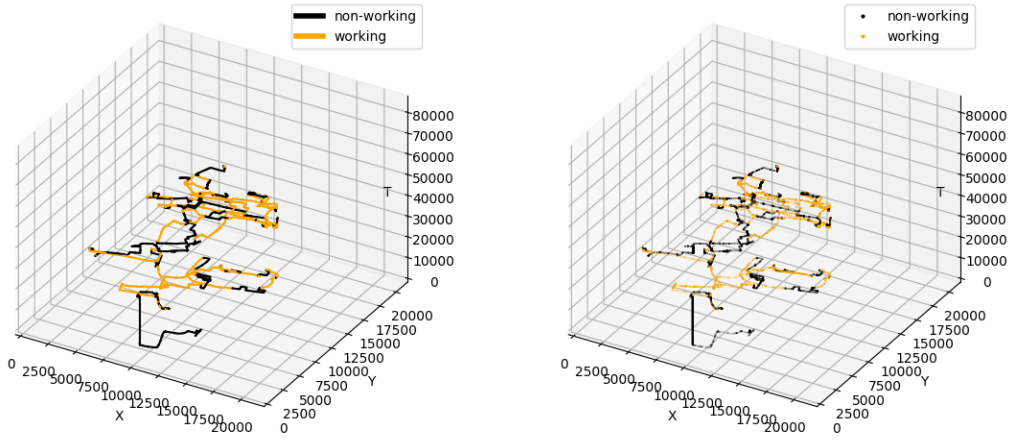
Section 1.4 comes up with the main research question and sub-questions based on the previous discussions: how to model trajectory data, how to access trajectory data and how to distribute (deploy) trajectory data.

At the end of this chapter, Section 1.5 gives a comprehensive outline of the thesis and a reading guide for the readers.

1.1. Research Background

1.1.1. Emergence of Trajectory Data

With the advancements in mobile IoT infrastructure and the popularization of corresponding devices, massive amounts of various spatio-temporal data are emerging, collected and stored, indicating the advent of the big data era in the field of Geographical Information System (GIS) (Li, 2019). Among them, trajectory data (shown and defined in Figure 1.1) of moving objects (such as vehicles, humans (Ellegård, 2019), animals etc.) collected by the Global Navigation Satellite System (GNSS), Global System for Mobile Communications (GSM) etc., is uniquely valuable because of the large amount of information it contains (Li et al., 2020b).



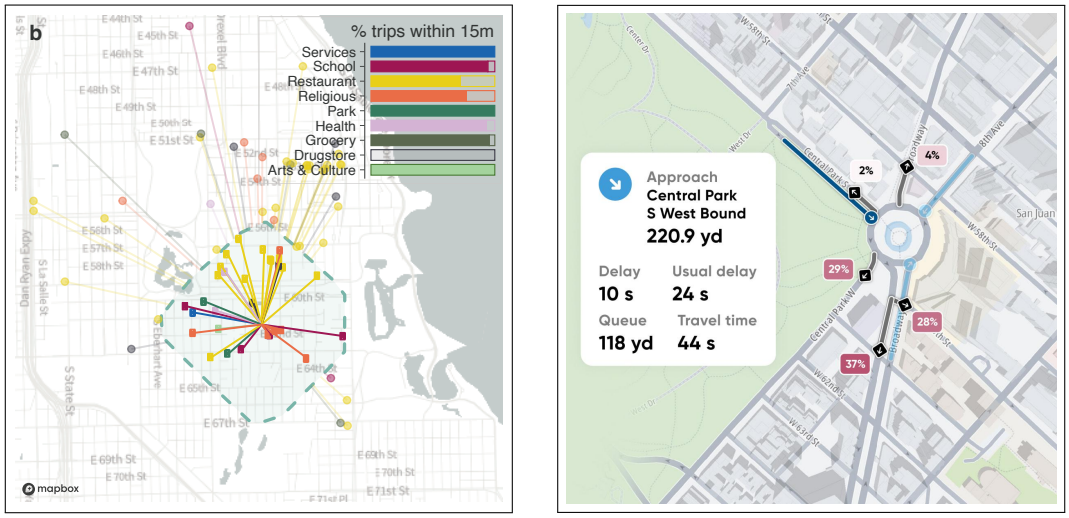
(a) Continuous Sequences (b) Discrete Points

Figure 1.1. Trajectory Data Definition: a trajectory is a sequence of 3D or nD points in 3D(X, Y, T) or nD(X, Y, Z, ..., T) spatio-temporal domain, each point may have some attributes as semantics such as the work state. For consecutive points, these state attributed may be the same, resulting data redundancy.

Generally, a trajectory is a continuous line in the space (shown in Figure 1.1(a)). However, due to the discrete nature of the sensing and storing processes (Alsahfi et al., 2020), the trajectory is always recorded as a series of discrete points (shown in Figure 1.1(b)). Different understandings of the trajectory may lead to different models.

1.1.2. Abundance of Trajectory Applications

The underlying information of the trajectory data has potential applications in scientific research (e.g. migratory patterns of animals), decision-making (e.g. road network optimization), emergency management (e.g. emergency responses for traffic accidents), and particularly real-time applications (e.g. autonomous vehicles), for added-value extraction (Zheng, 2015; Ribeiro de Almeida et al., 2020; Mokbel et al., 2023).



(a) 15-minute City Quantification, adopted from Abbiasov et al. (2024)

(b) Traffic Condition Identification, adopted from TomTom

Figure 1.2. Trajectory Data Applications: in Sub-figure(a), the 15-minute city quantification is done by analysing the time cost of travelling from a district to other spaces by mobile phone data. In Sub-figure (b), the traffic conditions are identified using real-time GPS positioning of vehicles.

For example, mobile phone data can be used to quantify crowd activity behaviour (shown in Figure 1.2(a)) to support the specification of urban planning policies and designs (Abbiasov et al., 2024). Or, the real-time trajectory data of vehicles can be used to identify traffic junctions (shown in Figure 1.2(b)) to remind relevant departments or drivers to respond in time.

1.1.3. Limitations of Centralized Solutions

However, due to the special properties of the trajectory data (detailed in Sub-section 1.3.1), there are difficulties in trajectory data management. Specifically, it faces limitations related to query types, access speeds, storage, scalability and more (Roddick et al., 2004). This seriously hinders the added-value extraction through data analysis or mining technologies.

Although the storage burden can be solved by horizontally extending inexpensive storage media (the data is stored in blocks of files), the query side: the retrieval speed and

reproduction of trajectory data is still limited by the large size and complex structure of the trajectory data, especially in the real-time environment.

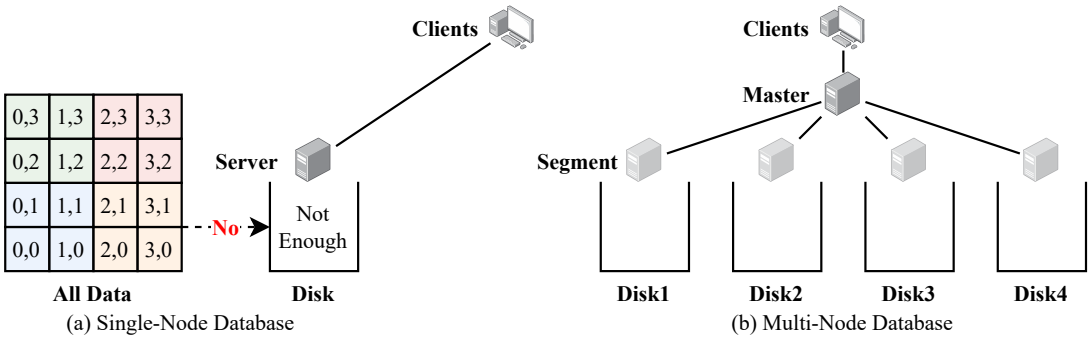


Figure 1.3. Data Management Solutions: traditional single-node Database Management System (DBMS) is unable to fit with the huge amount of data. The new multi-node DBMS could deal with a bigger size of data.

To cope with the query, computation and analysis of massive data, distributed data storage and computation, iconic products represented by GFS (Ghemawat et al., 2003), Bigtable (Chang et al., 2008) and MapReduce (Dean and Ghemawat, 2008), shown in Figure 1.3(b) are gradually replacing traditional centralized strategies, with Apache Hadoop and Massively Parallel Processing (MPP) ecosystems as the dominant ones. While Hadoop is suitable for high-throughput applications, MPP is more suitable for low-latency applications (Wang, 2022).

Although these technologies can be well applied in industries such as the Internet and Finance, actions to expand them to the field of GIS are still progressing. Considering the characteristics of trajectory data (non-structural, dynamically evolving, multi-dimensional, strict order for temporal dimensions etc.), it is still necessary to reconsider multiple aspects adapted to trajectory data, such as modelling, accessing, distributing etc. These functionalities are not directly embedded in distributed database products.

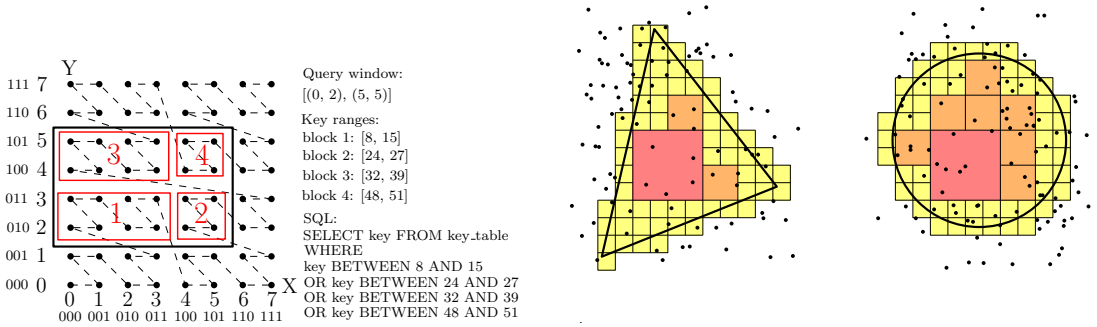
1.2. Research Context

Starting from (external) file-based methods (van Oosterom et al., 2017; Deng, 2020), a series of DBMS researches and solutions (Liu, 2022) for high dimensional PC have been developed at the GDMC, TU Delft (Liu, 2022). Some applications such as visualization (Liu et al., 2018), flood risk analysis (Liu et al., 2021b) and web-based data transmission (Meijers, 2022) have also been validated.

The trajectory data shares certain similar properties with the PC data, and some initial attempts to extend the PC solutions to the trajectory data have also been explored (De Vreede, 2016; Meijers et al., 2016; Meijers and van Oosterom, 2018; Li, 2020). PC solutions may be a useful starting point and foundation for trajectory management research. The latest relevant results are introduced below.

1.2.1. PlainSFC Solution for Point Cloud

As for the PC solution, the initial idea is to encode (integrate) the organizing dimensions (usually the spatial dimensions (X, Y, Z) and temporal dimension T) to SFC codes (usually the Morton or Hilbert curves are used) after offsetting and scaling. The codes themselves represent the exact dimension values (decoding is needed) and also function as the indexing keys with 1D indexing methods such as Balanced Tree (B-Tree) or Block Range Index (BRIN) for data organization.



(a) Executing a window query on a uniformly distributed 2D point set based on Morton encoding

(b) Querying with a triangle and a circle: false positive points in boundary cells will be filtered out by a second filter

Figure 1.4. PlainSFC Solution for Point Cloud: recursively partitioning the extent of data according to SFC regions to match different query geometries, for selecting data in the index-organized table, adopted from Liu (2022).

The typical application is to query the data by a shape: given the implicit relationship between the SFC code and the n-Dimensional Partitioning Tree (Quad-tree in 2D while Octree in 3D), the shape querying process is a traversal of the tree and testing the overlapping relations of the shape and cells (nodes), approximating the shape, resulting in a list of ranges (a range represents the left and right extremes of the code). This whole methodology is called the PlainSFC solution.

1.2.2. HistSFC Solution for Point Cloud

Despite the good performance for evenly distributed data, PlainSFC encounters performance bottlenecks when dealing with in-homogeneously distributed points, particularly in higher dimensions (when also integrating other dimensions in the SFC code such as R, G and B of the colour information). It is unnecessary to refine the hyper-cells where there is no data or nearly no data as the CPU costs of overlapping tests in higher dimensions are high and numerous empty ranges are generated.

To address these challenges, the HistSFC solution is introduced which involves an nD-histogram tree (it could be called adaptive Quadtree in 2D and adaptive Octree in 3D) that optimizes space decomposing according to point density (Liu et al., 2020b). When overlapping tests are performed on sparse data areas, early pruning of the tree can be performed, reducing the generation of vacant ranges, and thereby enhancing querying performance (Liu et al., 2020a).

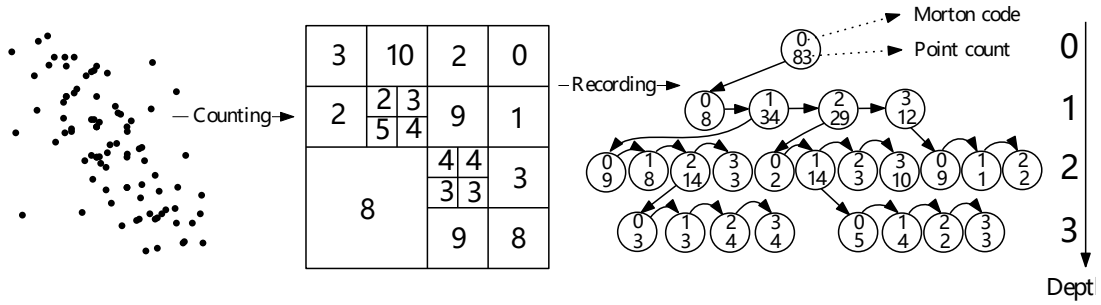


Figure 1.5. HistSFC Solution for Point Cloud: it records the count of points inside an nD node. If the count exceeds a threshold, i.e., the capacity of a leaf node, then the node is decomposed, adopted from Liu (2022).

1.2.3. Convex Polytope Query Approach

When it comes to the details of the overlapping tests, several spatial relation-checking methods between convex polytopes (represented by a set of half-spaces) with the nD boxes are also proposed.

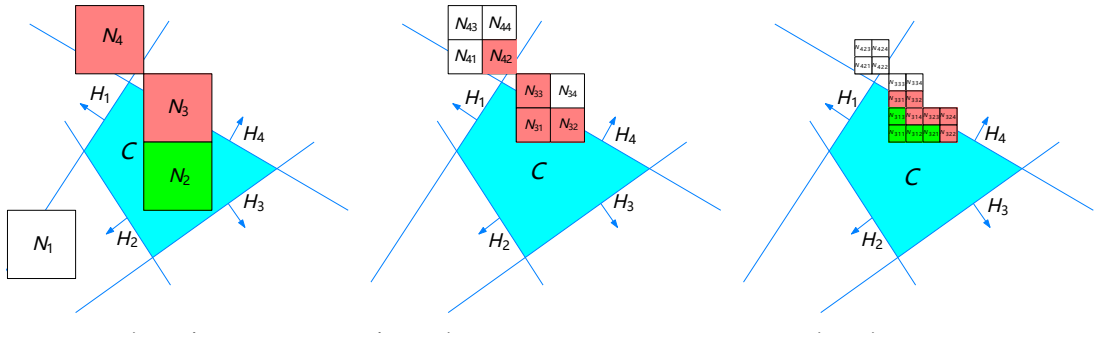


Figure 1.6. Convex Polytope Query Approach: The white nodes are outside, green nodes are inside and red nodes are on the boundary, adopted from Liu (2022). The process is to refine the shape one level by one level.

The intuitive approach is to test the relationship over all the vertices of the nD box and the half-spaces. However, this would lead to a high CPU cost in higher dimensions as the number of the half-spaces and the number of vertices in higher dimensions are all high. Another approach is to check the circumscribed circle of the box with all the half-spaces, therefore, there is no need to traverse all the vertices. However, this would lead to a large amount of unneeded data retrieval in higher dimensions.

Other indeterministic methods are also proposed to promise a significant reduction in false positives and enhanced query performance through adaptive strategies tailored to the dataset's distribution (Liu et al., 2021a).

1.3. Research Motivation

To some extent, this thesis is a continuation of many previous works mentioned above. Based on a similar direction, different models are used for different geographical phenomena and implemented on different platforms (distributed databases).

1.3.1. Research Difficulties

Although the technology to deal with non-spatial big data (such as financing, marketing or multimedia) is quite mature. The main difficulties arise from some properties of trajectory data (spatio-temporal data).

High Frequency: Trajectories are generated with high frequency and dynamism. Taking taxi data as an example, for 1 taxi, there may be 1 record per second. Assuming there are 3000 taxis in a city, on the one hand, every second, there would be 3000 (insertions) transactions into the database, which would be a big burden due to the potential performance degradation due to contention between high-frequency data insertion and concurrent data queries.

High Cardinality: On the other hand, The high frequency also directly leads to $60 \times 60 \times 8 \times 3000 = 8.64 \times 10^7$, eighty million records per one day (Assume the driver would work 8 hours a day) which is a huge size. The high cardinality does not only represent the size of the data. Assuming that the accuracy of the data is $1m$ in space and $1s$ in time, and all data are within a range of $10km$ by $10km$ by $24h$ space, then the possible choices of spatio-temporal coordinate might be $10000 \times 10000 \times 60 \times 60 \times 24 = 8.64 \times 10^{12}$, eight trillion choices. This makes some traditional indexing methods (such as Bitmap) less efficient or even impractical due to the vast number of potential index entries.

High Dimensionality: Even only considering the 2D space and the 1D time together (in a 3D) would lead to such a sparse data distribution ($(8.64 \times 10^7)/(8.64 \times 10^{12}) = 10^{-5}$) while the vast majority of the potential data space is empty. When more dimensions such as (speed, direction, state etc.) are added, the curse of dimensionality occurs reducing the selectivity. The selectivity is especially degenerated when querying by the condition involving multiple dimensions, nearly all irrelevant data would be loaded to be checked.

High Heterogeneity: This is mainly the properties of the space-related phenomenon which means the data may be unevenly distributed in the space. Usually, the spatial homogeneity (spatial auto-correlation) of space-related data is used for data clustering (storing similar data that are more likely to be queried simultaneously in adjacent data storage media) to improve data selectivity, however, this is not the case for distributed database as data must be partitioned and distributed. In addition, trajectory data is different from PC data since there may be strong connections between points on the same trajectory, the locality preservation used to block neighbour points for PC data is not enough as there is also trajectory preservation.

Other properties of the data such as uncertainty, dirty (noise data), real-time, multi-sourced, multi-structured, and multi-scaled are not the main concerns in this thesis, although they may be differently important in different application scenarios, they are excluded from the scope of this research.

1.3.2. Research Objectives

This research is aimed to explore the possibilities to extend the distributed technologies for trajectory data management. Specifically, this research will be motivated to reduce the pressure of data storage, accelerate the speed of data queries, and balance the space and time complexity, as well as the complexity of developing and programming (for the ease of database administrators).

Time Aspect: The main focus is on enhancing the query (especially spatio-temporal related ones such as selection by polytope) speed, meeting the client's demands.

Storage Aspect: Though not the primary focus, the research aims to minimize the redundancy in the original data, exploring compression strategies for efficient storage.

Complexity Aspect: Based on Occam's Razor principle [1], the resultant systems should be simple and easily understandable. For instance, intricate algorithms may benefit in a small domain but sacrifice the generality and make the solutions complex and hard to maintain, increasing the development cost which is not optimal from the software engineering perspective (Brooks, 1974).

1.3.3. Research Scope

This research is mainly divided into two parts: scientific research and engineering implementation. The engineering implementation as the basis of everything includes database deployment, Python-DBMS interface implementation, and basic applications such as visualization. From the scientific research perspective, a certain degree of discussions of the theories of spatio-temporal data, distributed database systems and SFC are given, leading to the main concepts and the engineering decisions.

Detailed system parameters like tuning disk allocation for swap, and cache size for instance etc. in the production environment with extremely large volumes of data is not included due to time and device limitations. Since experiments are held in the virtual environment with a subset of data, this research would not focus on the absolute speed due to the different hardware and software environments (The interfaces implemented in compiling language and Python must not be comparable). However, the relative scale-up or speed-up due to the different virtual resources, different task sizes and different data-distributing strategies will be tested.

[1] Occam's razor is the problem-solving principle that recommends searching for explanations constructed with the smallest possible set of elements.

The geometrical and semantic information of the trajectory is the first concern while the topology is less concerned in this study. In some other studies, the trajectory data could be organized in graph form, but are not relevant here. Cleaning and refining the data and correcting for outliers are also outside the scope of this study. This study only focuses on data management and retrieval-related queries while the complex (computationally intensive) operations for data mining such as iterations (take the k-means algorithm as an example) and recursions are also not included. In addition, only the trajectory data is taken into consideration, in the multi-source data case: spatial joins done between several tables are not included.

1.4. Research Question

Based on the above discussions, the main question and sub-questions are proposed.

Main Question: *What is the potential of integrating the SFC accessing methods and distributed databases for the efficient management of trajectory data with a huge volume?*

1.4.1. Sub-question 1: Trajectory Modelling

How to perform the trajectory modelling? Is it better to model it as a sequence instead of a point cloud or grid?

1.4.2. Sub-question 2: Trajectory Accessing

How to perform the spatial accessing? Is it possible to use the SFC to do indexing and clustering with data awareness for a better querying performance?

1.4.3. Sub-question 3: Trajectory Distributing

How to perform data distributing in distributed DBMS? Is it possible to use the SFC to do so and what are the potential benefits? Will the speed-up and scale-up of distributed databases be guaranteed?

1.5. Thesis Outline

1.5.1. Thesis Organization

Chapter 1 introduces this thesis, including research background, context, motivation and the posed research questions.

Chapter 2 gives the theoretical background of the thesis and the general concepts that should follow. These discussions explain the reasons for many engineering decisions in the later parts.

Chapter 3 gives the literature review, including the modelling, accessing and distributing, some existing counterparts, and the discussions based on them.

Chapter 4 gives a quick description of the methodology for an immediate impression including how to model, access and distribute the trajectory data.

Chapter 5 gives the implementation details which are one-to-one mapping to the Methodology. It describes the exact details focusing more on the pseudo-codes of specific algorithms and many remarks during the implementation.

Chapter 6 gives the designs and results of experiments to confirm the superiority of the methodologies presented in the previous chapters.

Chapter 7 summarizes, concludes and discusses the full thesis in terms of the methodology, implementation, experiments, findings etc. from a higher perspective.

1.5.2. Reading Guide

For readers who want to grasp the main ideas and results, please directly refer to Chapter 4, Chapter 6 and Chapter 7.

For readers who want to understand the conceptual framework and engineering decisions, Chapter 2 and Chapter 3 provide more details.

For readers who want to dive into the implementation, please refer to the Chapter 5 and Appendix C. For a specific algorithm, it is better to have a look at the source code at Github.

Appendix A gives the values of the experiment results rather than the diagrams while Appendix D gives a discussion about the range merging optimization problem which is not tightly relevant but interesting and potentially useful for future work.

2. Theories and Concepts

Chapter Contents

This chapter gives the theoretical background of the thesis and the general concepts that should follow. These discussions explain the reasons for many engineering decisions in the later parts.

Section 2.1 gives a more detailed description of the spatial data properties and the deriving difficulties of using traditional data management solutions. Possible solutions are also presented.

Section 2.2 gives an overview of the modern (distributed) database system architectures, presenting key features and the corresponding design principles. Typical classifications of products are also presented.

Section 2.3 gives an introduction to the SFC, a strong mathematical "weapon" used for dimensional reduction, proximity preservation and distributing (in this case, the pseudo-random sampling strategy).

At the end of this chapter, Section 2.4 concludes the above three sections and integrates them into an abstract framework that helps solve the research questions.

2.1. Real World: Trajectory Phenomenon

The trajectory is a spatio-temporal phenomenon, in addition to inheriting some characteristics from geographical phenomena (e.g. elevation), it also inherits some characteristics from social phenomena (e.g. population).

2.1.1. High Cardinality

As discussed in Sub-section 1.3.1, the high cardinality characteristic of the trajectory data makes the query performance highly inefficient, as the selectivity is low (if a full scan of the table is done). There are some typical solutions and the corresponding limitations.

1. Solution 1: The filtering methods could be used to reduce cardinality, however, this would lead to information loss which may not meet the requirements. Some (down) sampling methods keep the entire information which means the original data could be recovered.
2. Solution 2: The pyramid, vario-scale and data-cube (in Online Analytical Processing (OLAP)) methods (Jensen et al., 2010; Han et al., 2022; Vaisman and Zimányi, 2009; Leonardi et al., 2010; Gómez et al., 2012; Leonardi et al., 2014; Alsahfi et al., 2020; Gao et al., 2022) could be used, however, there may also be problems such as data redundancy or data type limitation (only numerical data supported), these methods are usually used for visualization (Orlando et al., 2007; Gómez et al., 2009).
3. Solution 3: The aggregation method, on the one hand, could aggregate multiple items into one item (aggregation with information loss), but on the other hand, aggregate multiple items into one set of items (lossless aggregation) [1]. By using the latter alternative, the cardinality of the index entries could be reduced.

The above three solutions seek the cardinality reduction from the modelling aspect. In addition, for the query execution, the indexing method is still needed to quickly find the pointers to the location of the requested data. The core idea is the dividing and conquering principle which explicitly or implicitly maintains a tree structure or lookup table.

Several classical indexing methods for 1D data are shown and compared in Table 2.1. Traditionally, the hashing method does not care about the implicit order of the spatial data and fails to preserve spatial locality, which can lead to inefficient query performance for range searches [2].

The B-Tree and BRIN indexing methods are the traditional optimal solutions for the 1D data while BRIN needs data to be sorted and is more suitable for queries where data size is huge and batch processing happens (Liu and Özsu, 2009; Elmasri et al., 2015; Silberschatz et al., 2019).

[1] A typical case of this kind aggregation is to group multiple points into a MultiPoint record.

[2] Locality-preserved hashing does exist that can preserve the order and locality of the data. However, it is not supported in many database systems.

Table 2.1. 1D Indexing Methods

Method	Cardinality	Dimensionality	Application
Hashing	Low	Low	Key-Value
B-Tree	High	Low	Small Items
BRIN	High	Low	Large Range

2.1.2. High Dimensionality

Solving high-cardinality and high-dimensional problems is often at odds with each other. For example, the Bitmap method is suitable for high dimensionality but is only suitable for data with very low cardinality in column-wise tables (Elmasri et al., 2015). It struggles with high cardinality due to the sheer volume of the unique Bitmaps required.

Other indexing methods for multi-dimensional data are shown and compared in Table 2.2. Another example of the contradiction between high dimensionality and cardinality is B-Tree which can not easily deal with high dimensionality data as for each dimension on tree structure is needed, traversal of these threes would be costly in time.

Table 2.2. nD Indexing Methods

Method	Cardinality	Dimensionality	Application
Bitmap	Low	High	Column-wise
R-tree	Medium	Medium	Spatial Data
KD-tree	Medium	Medium	Spatial (Point) Data
Quad-tree	Medium	Medium	Spatial Data

The Rectangle Tree (R-Tree), KD-tree, and Quad-tree are typical indexing methods for spatial data (Li et al., 2023; Zhang et al., 2019; Shekhar and Xiong, 2007). R-tree efficiently manages spatial data using bounding rectangles but faces overlapping problems in high dimensions or with concave shapes (for example, several parallel long line strings may share nearly equivalent big bonding boxes.). KD-tree excels in point-based searches using hyper-planes to construct the tree and is more suited for k-nearest neighbour queries. Quad-tree (and its variants) evenly subdivide the spaces into quadrants, however, they are sensitive to the data distribution. These methods can deal with spatial data indexing in different scenarios.

In addition to the indexing methods, there exist other solutions to deal with the high dimensionality of the data, however, these methods are more suitable in the data analysis use cases rather than the data management use cases.

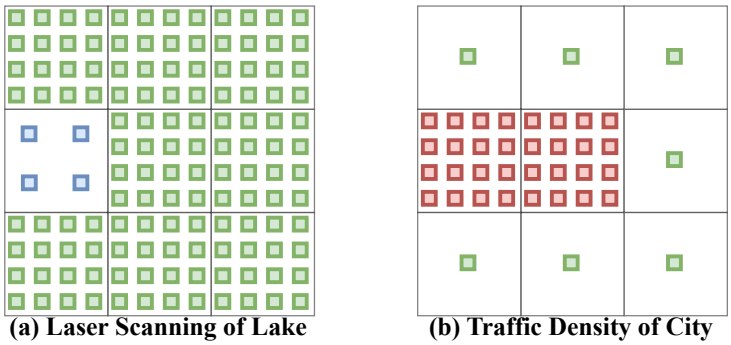
1. Solution 1: Principle component analysis (PCA) or t-distributed stochastic neighbour embedding (T-SNE) dimension reductions could reduce the dimensionality, but they also cause the loss of exact information which does not meet the requirements (e.g. selecting the geometries of trajectories that intersected with a face).

2. Solution 2: OLAP Data Cubes facilitate multidimensional analysis and aggregation but struggle with geometrical dimensions (like line strings) that lack explicit hierarchies.
3. Solution 3: Vector Databases excel in facilitating similarity and nearest neighbour searches through efficient indexing of vector spaces, but they are less effective for handling range queries that involve attributes beyond similarity.

2.1.3. High Heterogeneity

The spatial homogeneity (uniformity within a space) and the heterogeneity (diversity within a space) are unique properties of the geospatial phenomena. There are two generally recognized geographical laws about these properties.

The first one is the spatial homogeneity law proposed by Tobler (2004): "Everything is related to everything else, but near things are more related than distant things." and the second one is the spatial heterogeneity law proposed by Goodchild (2004, 2022): "Spatial heterogeneity could be called the second law of geography".



(a) Laser Scanning of Lake and Grass

(b) Traffic Density of City and Rural Area

Figure 2.1. Scale Effects of Heterogeneity: the Sub-figure(a) shows the point cloud (blue dots for lake while green dots for grassland) retrieved from laser scanning and the Sub-figure(b) shows the positioning points (red dots for data in the urban area while green dots for data in the rural area).

When observing spatial phenomena, especially social-related ones, we can find that the complexity and heterogeneity of geographic spaces increase with granularity. For example, for the laser scanning data shown in Figure 2.1(a). When considered globally, the spatial distribution of the data is uneven because it is almost impossible to obtain laser reflection from the water surface. However, when considered locally (only observing the interior of the water surface or the interior of the grassland), it can be found that the data is relatively evenly distributed in space. The same is true for trajectory data, shown in Figure 2.1(b).

This special property (global heterogeneous but locally homogeneous) could be used for data clustering which means storing similar (local) data together. This also suggests the need to care about spatial distribution (not storing distant data together)when doing indexing (data awareness) and the data distributing strategy for load-balancing in distributed databases (which are discussed in the later sections).

2.2. Digital World: Database Systems

As we transition from the specific data storage problem within the GIS domain to a broader perspective, it becomes essential to understand the macro classification of DBMS architectures. This understanding is crucial not only for grasping the fundamental structures of data storage solutions but also for the scalability, performance, and easy-to-use considerations that influence the choice of architecture for applications.

2.2.1. Database System Architectures

The database system architectures could generally be classified by whether the resources are tightly coupled or locally shared (Elmasri et al., 2015; Wang, 2022; Guan, 2020; Özsu et al., 2020).

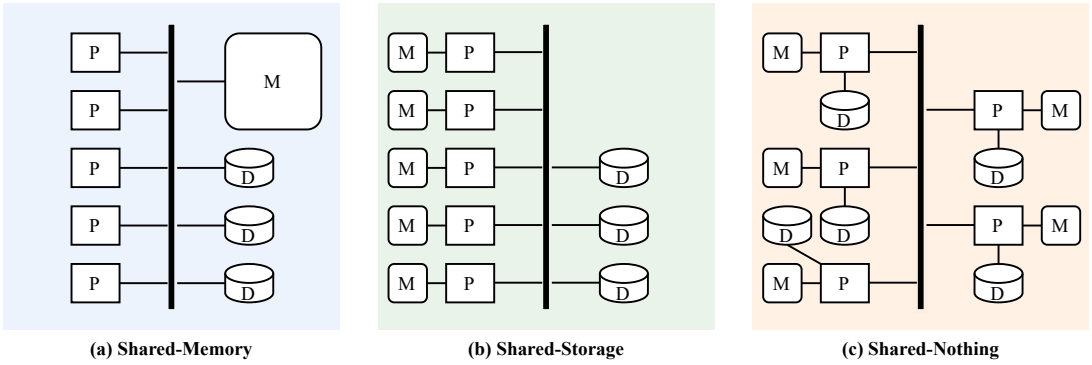


Figure 2.2. Typical DBMS Architectures: the P represents the processor (CPU), the M represents the memory while the D represents the Disk.

Shared-Memory: In this architecture, CPU, Memory, and Disk resources are locally shared and tightly coupled (shown in Figure 2.2 (a)). Depending on the application, the bottlenecks may occur at the CPU cache or IO, causing a decline in performance.

Shared-Storage: In this architecture, disk resources are locally shared (shown in Figure 2.2 (b)). Vertical scaling is possible (though expensive), and the bottleneck may still be associated with IO operations.

Shared-Nothing: In this architecture, CPU, Memory, and Disk are all distributed (shown in Figure 2.2 (c)). Horizontal scaling helps to alleviate the bottlenecks mentioned in the previous two models, but it may introduce additional communication and data exchange costs and marginal effects such as complexity. There are three typical products, Haddop, Spark and MPP.

The shared-nothing architecture and products are what will be used and experimented with within this project due to their high scalability, remember that one main focus is to deal with a huge amount of data and possibly big queries (a typical example of big query is to select all the data and do visualization).

2.2.2. Distributed Database Features

The distributed database is built upon a hierarchical organization which facilitates efficient data management and retrieval by separating the database system into distinct levels, each with its specific role and functionality. This hierarchy can present at almost any item in the computer system. For example, multiple shared-memory machines could be composed to a bigger shared-nothing product (shown in Figure 2.3) [3].

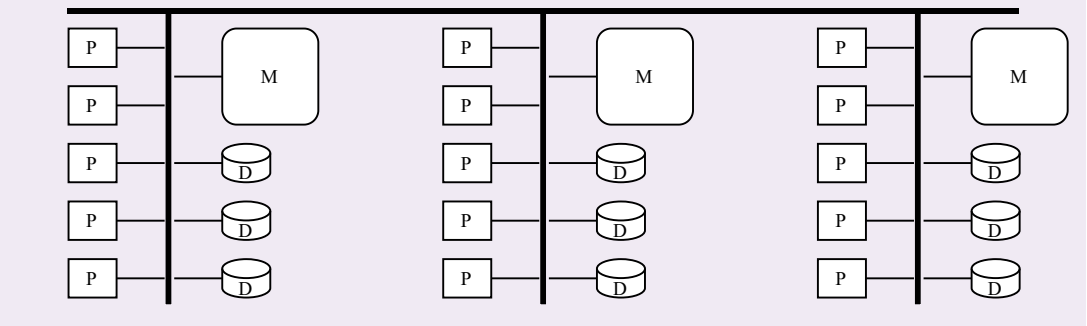


Figure 2.3. Database Hierarchical Organization: the P, M and D have the same meaning in Figure 2.2, the thin line represents the BUS (high speed) while the thick line represents the external network (low speed).

Parallelism: Parallelism in database system architecture leverages the power of multiple processors or machines to perform database operations concurrently, significantly improving system performance and throughput.

1. Coarse granular parallelism: There are multiple nodes (machines) that work together, and each node solves a smaller part of the bigger problem.
2. Fine granular parallelism: For each node, multiple processors work together, and each processor solves a piece of a small problem.
3. Data parallelism: The big data is partitioned across nodes/processors to enable simultaneous operations on different segments/cores.
4. Task parallelism: The big task or query is partitioned across nodes/processors and these sub-tasks are executed in parallel.

Localization: Localization within database system architecture focuses on optimizing data storage and access by considering the spatial or logical proximity of data. This concept aims to reduce latency and improve performance by ensuring that data is stored closer to other data that would be most frequently accessed or processed at the same time, minimizing network traffic and data transfer times. For multiple dimensions (such as space and time), their dominance may vary depending on the nature of the data or the needs of the application. The local powerful CPUs could be fully utilized to deal with compression and decompression.

This concept also takes into account geographically distributed data centres and the need to ensure that local data storage or processing is done in local data centres as

[3] Even the internal structure of modern CPUs is hierarchical to a certain extent, featuring multiple levels of cores and caches.

much as possible to reduce network communications. At the same time, the distribution of data centres needs to consider data fault tolerance (for example, data in earthquake-prone areas may need to be backed up in other data centres). However, fault tolerance and recovery are not included in this thesis.

2.2.3. Distributed Database Products

There are several matured distributed computation and database products such as Hadoop, Spark and MPP databases (e.g. Greenplum) (Wang, 2022).

Hadoop: The Hadoop [4] ecosystem, anchored by the Apache Hadoop core, embraces distributed computing for large-scale data processing. With a shared-nothing architecture, Hadoop facilitates parallel computation and fault tolerance, making it a versatile and scalable solution for diverse big data needs. However, the IO cost is high as for each computation, the data would be read and written from/into the disk. Some variants such as Spark convert disk-based calculations into memory-based calculations, reducing latency.

Spark: Spark [5] optimizes Hadoop's framework by shifting from disk-based to in-memory processing, significantly accelerating data analysis and supporting advanced analytics and real-time applications more effectively. The above two are all "computing" frameworks while a lot of database functions such as ACID (Atomicity, Consistency, Isolation and Durability) are weak (Elmasri et al., 2015).

Table 2.3. Shared-Nothing Products

Feature	Traditional Database	Hadoop	Spark	MPP Database
Volume	GB-TB	PB-EB	TB-PB	TB-PB
Robustness	High	High	High	Medium
Scalability	Low	High	High	High
Latency	Medium	High	Low	Very Low
Throughput	Low	High	High	Medium
Data Type	Structured	All	All	Structured

MPP (Massively Parallel Processing) Ecosystem: MPP ecosystems excel in high-performance data management through a shared-nothing DBMS-based architecture, distributing CRUD (Create, Read, Update and Delete) tasks across multiple nodes for efficient handling of large datasets. Horizontal scalability enhances its suitability for complex analytics and data warehousing applications (Wang, 2022; Presser, 2017). A general data retrieval process is shown below. These products are highly encapsulated (for example, many data types are already available) which means they are easy to use as the requirements of development are low compared with Hadoop (the MapReduce

[4] https://hadoop.apache.org/
[5] https://spark.apache.org/

functions may be difficult for junior developers). Remember that reducing development costs is also a motivation (Brooks, 1974). However, it may also pose flexibility issues as the customization is seldom supported.

$$\text{Client} \xrightarrow{\text{SQL}} \text{Master} \xrightarrow{\text{Logical Query Plans}} \text{Segments} \xrightarrow{\text{Result}} \text{Master} \xrightarrow{\text{Result}} \text{Client}$$

2.3. Math World: Space-Filling Curve

As mentioned in the Section 1.2, the SFC serializes multi-dimensional data for efficient querying in databases while keeping the proximity and evenly partitioning and distributing data for balanced computation.

2.3.1. Dimension Reduction

The SFC could be used to do dimension reduction, mapping the nD tuple to a 1D value (Bader, 2012; Lawder, 2000). Take the simplest Row curve (shown in Figure 2.4, it may also be called Sweep curve (Shekhar and Xiong, 2007)) as an example, the mapping rule is: the index of the x coordinate is multiplied by the base (which is 2 raised to the power of the depth), and then the index of the y coordinate is added to this product.

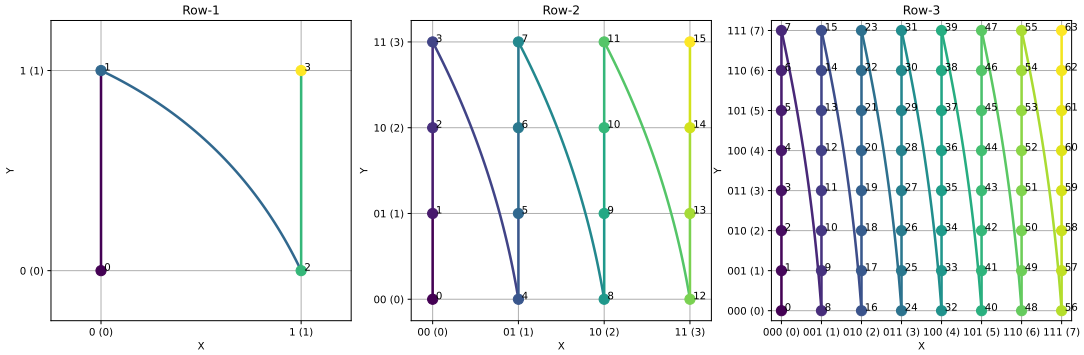


Figure 2.4. Space Filling Curve Example: Row Curve: It shows the Row curves in the 2D case with depth 1 to 3.

Take the two tuples of a row curve with depth 3 as an example, the mapping calculation is shown below.

$$x = 6, y = 5 \rightarrow row\_code = 6 \times 2^3 + 5 = 53$$
$$x = 7, y = 4 \rightarrow row\_code = 7 \times 2^3 + 4 = 60$$

2.3.2. Proximity Preservation

However, the Row curve is not considered a good SFC that preserves the proximity (locality). Take the Morton curve (shown in Figure 2.5, it may also be called Peano

curve or Z-order curve (Shekhar and Xiong, 2007)), with a more complex mapping rule but good proximity preservation as an example. The mapping rule is: interleaving the binary representations of the components of tuples in a multidimensional space to one value in a linear one-dimensional space.

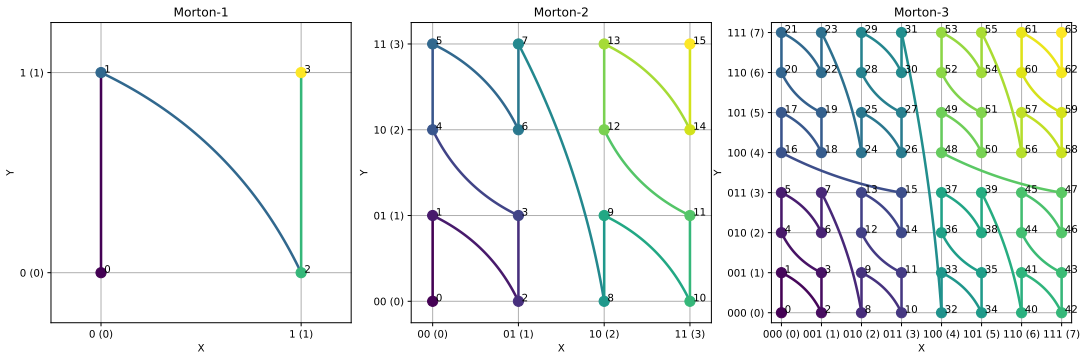


Figure 2.5. Space Filling Curve Example: Morton Curve: It shows the Morton curves in the 2D case with depth 1 to 3.

Take the same two tuples shown above, the mapping calculation is shown below.

$$x = 110\,(6),\ y = 101\,(5) \rightarrow morton\_code = 111001\,(57)$$
$$x = 111\,(7),\ y = 100\,(4) \rightarrow morton\_code = 111010\,(58)$$

For the same two tuples that are closer in the 2D space, the Row codes are 53 and 60 while the Morton codes are 57 and 58. If the two tuples are stored in the 1D array disk based on the codes, it is obvious that the Morton method helps more to store closer tuples also closer in the disk.

In addition to the Row curve and Morton curve, there are many other curves. It is argued that the Hilbert curve may be better at proximity preservation (Lawder, 2000). However, the mapping rule of the Hilbert curve is far more complicated and the calculation is more time-consuming. Considering this, the Morton curve is used in this thesis.

2.3.3. Pseudo-random Sampling

The space-filling curve could also be used for the data partitioning which may be useful in the distributed database (Bader, 2012; Lawder, 2000; Özsu et al., 2020; Eldawy et al., 2015). Take the 2D Morton code as an example. There exist two partitioning methods, block-based partitioning and sample-based partitioning.

Assume, there is an image data shown in the Figure 2.6, each pixel is indexed by the Morton code, the target is to evenly partition it into 4 parts. The first intuition is to partition the ordered list of codes into 4 continuous parts (blocks) and the results are shown in Figure 2.7. This is very easy to explain visually, just fold this image in half two times and then tear it apart.

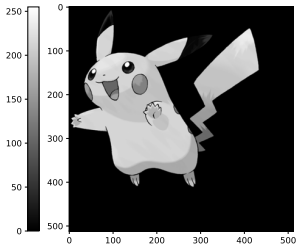


Figure 2.6. Distributing Strategy: Original Data: An gray 512 × 512 image of Pikachu from Wikipedia.

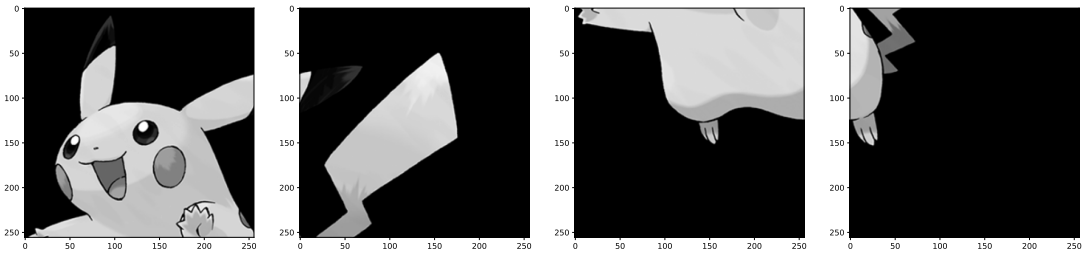


Figure 2.7. Distributing Strategy: Blocking by Morton Curve: 4 256 × 256 images correspond to four quadrants of the original image.

However, this is not an even data partition. Considering that different partitions have different areas of black areas (no data areas) if quadtree compression is used, the data size of the last partition is very small. An alternative is to use the random, however, the locality will be broken and it is hard to reassemble the four parts of the data to the original one.

Thus, a pseudo-random sample method is designed. Assume four neighbouring pixels are one unit, the idea is to allocate the same quadrants of all the units as partitions, assume the 2d data is serialized by a Morton cure in a 1D array, which could be easily implemented by Python list method $series[i :: 4]$.

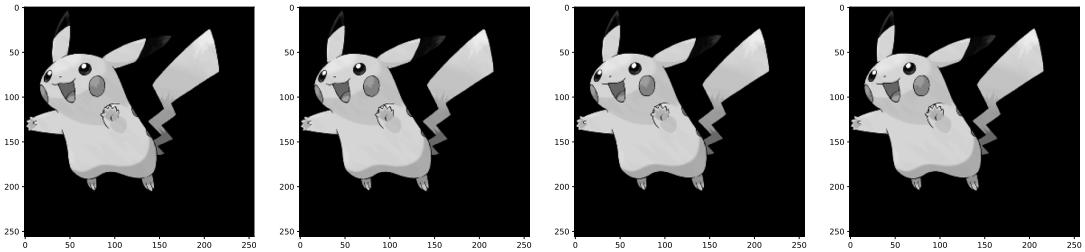


Figure 2.8. Distributing Strategy: Sampling by Morton Curve: 4 256 × 256 images correspond to four samples of the original image, it may not be obvious, but if zoom in, the Pikachu's thumbs are different in these four Sub-figures. The thumb in the first Sub-figure is sharp while round in the second Sub=figure.

It can be found that the four new Sub-figures obtained are almost the same (which means the data partitioning is even leading to load-balancing). Even though each one is the result of down-sampling, they can be restored to the complete original image

since the mapping of the SFC is deterministic.

2.4. Chapter Conclusion

To summarize the above discussion, to solve the research questions raised in Chapter 1, there is a need to design a distributed multidimensional database (the concept is shown in Figure 2.9). Among them, multi-dimensionality corresponds to the high dimensionality of trajectory data, and SFC is used to reduce dimensionality and maintain locality. Distribution is to solve the problem of high cardinality of trajectory data. When distributing data, it is also necessary to consider the spatial heterogeneity of data.

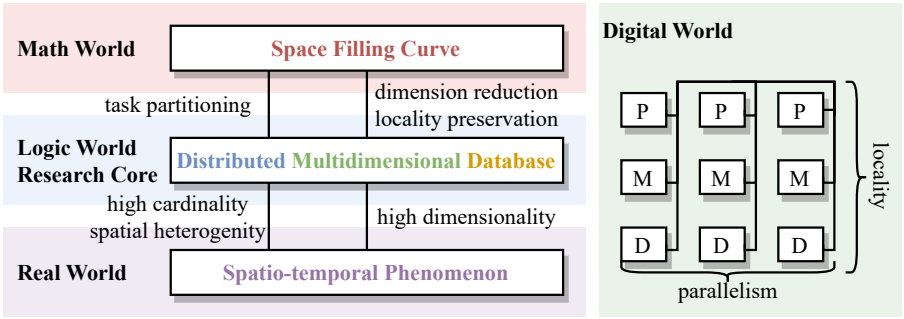


Figure 2.9. Conceptual Framework: The conceptual model is broken into three main components: the Math World, the Logic World, and the Digital World. Each of these components is related to a specific aspect of the database's operation and functionality, and they interact with each other to process and manage spatio-temporal data.

These conceptual designs will be mapped to a specific distributed database utilizing the parallelism and localization features (the framework is shown in Figure 2.10). Considering that the data size may not exceed the EB scale and data retrieval is the main application rather than complex computations, and the latency of data retrieval is an important consideration, the MPP architecture is used (specifically, the Greenplum).

Under the above framework, a typical data flow (a user use case shown in Figure 2.11) is that the client sends a data request (usually querying the data inside a shape) to the master node. The master node analyzes the shape and sends the corresponding indices to each segment node. The segment nodes fetch the data based on the indices (possibly there is post-processing) and return the data to the master node, which sends the returned data to the client.

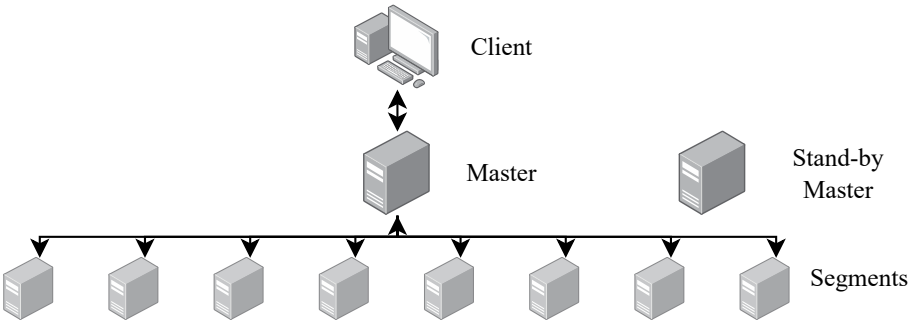


Figure 2.10. Implementation Framework: it is a client-server distributed database architecture with a central master server, a stand-by master for redundancy (in case of malfunction of master), and multiple segment servers for distributed data processing.

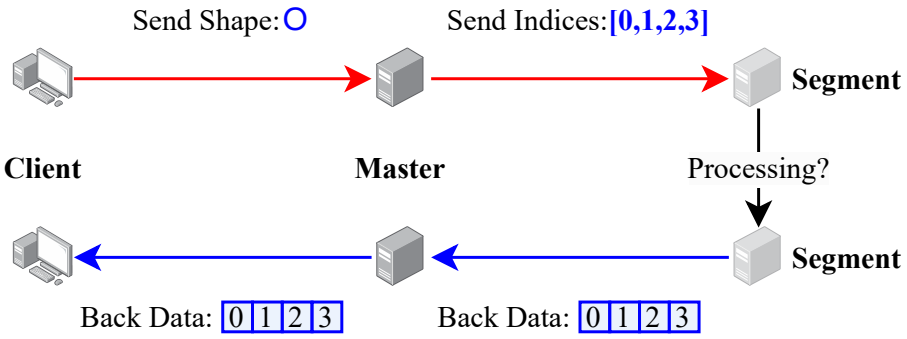


Figure 2.11. Typical Use-case Data-flow: The red arrows show the forward data flow, the blue arrows show the backward data flow while the black arrows show the internal data flow.

3. Related Work

Chapter Contents

This chapter gives the literature review and the discussions based on them.

Section 3.1 presents four types of trajectory modelling methods considering different levels of flexibility and aggregation.

Section 3.2 presents the clustering, indexing and partitioning methods to speed up the access to data and improve the load-balancing and localization in database systems. Everything could be unified by the SFC.

Section 3.3 presents different kinds of queries that should be supported in a trajectory database system.

Section 3.4 presents four exsisting products of trajectory data management systems and gives intuitions derived.

At the end of this chapter, Section 3.5 concludes the lessons learnt from all the related work.

3.1. Trajectory Modelling

After the real-world phenomenon is abstracted into a conceptual model, a data model (data structure) and corresponding storage structure in the computer world must be

determined as well.

Depending on the aggregation level and whether structured, the trajectories are typically modelled as points, segments, sequences (they may also be called sub-trajectories) and grids (Ribeiro de Almeida et al., 2020).

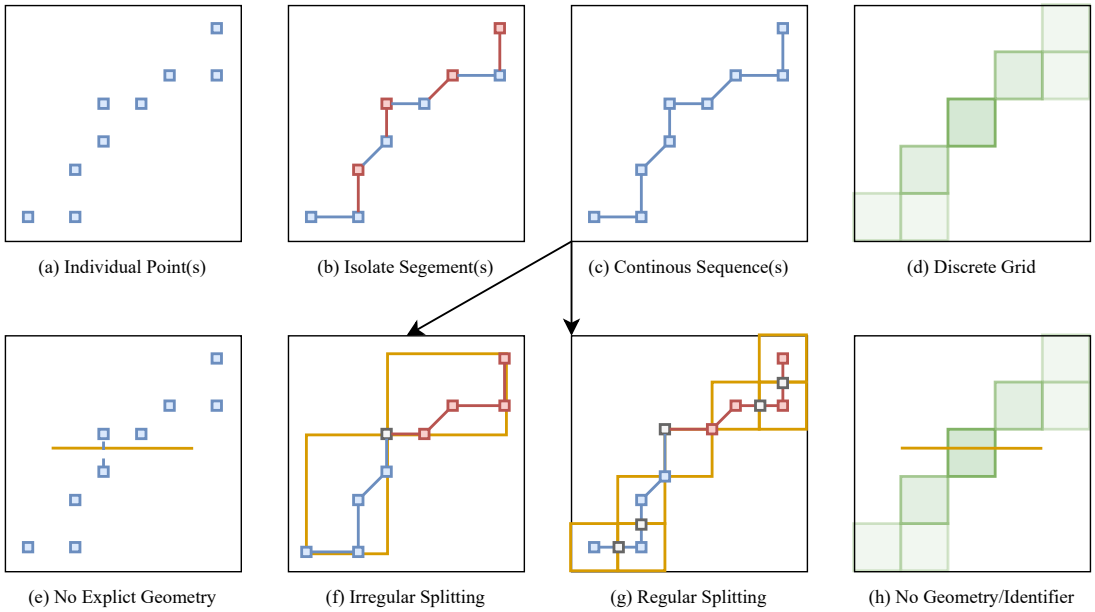


Figure 3.1. Typical Trajectory Models: The sub-figures (a) to (d) follow an increasing aggregation level (form lossless aggregation to aggregation with information loss). The sub-figures (e) and (h) demonstrate the query limitations of point-based and grid-based modellings. The sub-figures (f) and (g) give two directions of line string splitting.

3.1.1. Point-Based Modelling

1. Modelling (shown in Figure 3.1 (a)): It takes the individual points as the basic storage element. Each dimension is encoded as individual columns in the table or integrated as the geometrical point or space-filling code (De Vreede, 2016; Meijers et al., 2016; Meijers and van Oosterom, 2018; Li, 2020; Liu, 2022). An example table schema is shown in Table 3.1.
2. Advantage: Simple for clustering and indexing by space, flexible for all kinds of queries.
3. Disadvantage: No explicit geometries, no trajectory preservation and queries require a lot of pre-reconstruction costs (shown in Figure 3.1 (e)). Generally, the state is an attribute whose value is the same for consecutive points such as whether the car windows are open or whether there are passengers. This brings data redundancy.

Table 3.1. Point-Based Modelling

Field	Id	State	Geometry	...
Example	i_0	s_0	Point(x_0 y_0 t_0)	...

3.1.2. Segment-Based Modelling

1. Modelling (shown in Figure 3.1 (b)): It takes the successive two points as the basic storage element (Pelekis et al., 2015; Pfoser et al., 2000). The lineString geometrical data structure may be used and an example table schema is shown in Table 3.2.
2. Advantage: Explicit geometries are kept, good for queries.
3. Disadvantage: Still too scattered, there is no trajectory preservation. All points are stored twice, worsening the storage burden.

Table 3.2. Segment-Based Modelling

Field	Id	State	Geometry	...
Example	i_0	s_0	LINESTRING(p_0, p_1)	...

3.1.3. Sequence-Based Modelling

1. Modelling (shown in Figure 3.1 (c)): It takes the vector-based lineString as the basic storage element after reconstructing meaningful sequences of points (those points are geometrically or semantically correlated). The semantics are assigned to each sequence as attributes. Depending on the application, sequences are often modelled in 3D or 4D space (Baars, 2004; Zimányi et al., 2020; Biljecki et al., 2013). An example table schema is shown in Table 3.3.
2. Advantage: Based on the trajectory preservation (Pfoser et al., 2000; Li et al., 2020a), there is a potential for row-wise compression to reduce redundant information. While in a distributed environment, the cost of compression and decompression could be alleviated by localizing the computations.
3. Disadvantage: Complexity and difficulty of pre-processing.

Table 3.3. Sequence-Based Modelling

Field	Id	State	Geometry	...
Example	i_0	s_0	LINESTRING(p_0, p_1, ..., p_n)	...

3.1.4. Grid-Based Modelling

1. Modelling (shown in Figure 3.1 (d)): It takes the regular cell/cube as the basic storage element. Pre-calculation/aggregation of certain semantics at certain cells

in the regular spatial-temporal cubes is needed (Leonardi et al., 2010, 2014). An example schema is shown in Table 3.4, and another example is shown by the Python numpy array: `grid = numpy.zeros((height, width, depth))`.
2. Advantage: This kind of modelling is especially good for OLAP applications due to its regular multi-dimensional nature.
3. Disadvantage: Geometries are lost, and querying types are limited (shown in Figure 3.1 (h)).

Table 3.4. Grid-Based Modelling

Field	Semantic	Value	...
Example	"Speed"	{{{v_0, v_1}, {v_2, v_3}}, {{v_4, v_5}, {v_6, v_7}}}	...

Table 3.5. Comparison of Different Modellings

Model	**Flexibility**	**Cardinality**	**Volume**	**Representation**
Point	High	High	High	Multi-col, SFC, SFS [1]
Segment	High	High	Extreme High	SFS
Sequence	Medium	Medium	Medium	SFS
Grid	Low	Low	Low	Data Cube

The comparison of the above 4 modelligns is shown in Table 3.5. The sequence-based modelling is a compromise/balance between point-based modelling and grid-based modelling, aggregating at certain levels (reducing redundancy) while keeping the geometries. The main disadvantage is the unstructured nature such as different sampling rates or lengths for each trajectory. Thus, the core when doing sequence-based modelling is how to transform the unstructured trajectories into structured ones.

$$\text{Point} \underset{\text{deconstruct}}{\overset{\text{aggregate}}{\rightleftarrows}} \text{Segment} \underset{\text{deconstruct}}{\overset{\text{aggregate}}{\rightleftarrows}} \text{Sequence} \overset{\text{aggregate}}{\rightarrow} \text{Grid}$$

After the reconstruction of the whole trajectory, further splitting is needed which is a preparation for spatial accessing methods. Typically, there are irregular splitting and regular splitting (shown in Figure 3.1 (f) and (g)). The irregular method splits the trajectory geometrically (such as by turning points) or semantically (such as by driving state). The regular method splits the trajectory by a regular spatio-temporal cube.

In combination with two splittings, the trajectory is better to be first irregularly split by state (such as whether the taxis are working) and then regularly split. In this way, the length of each sequence can be controlled, and the sequence can be contained in the cells to facilitate subsequent clustering and indexing.

3.2. Trajectory Organizing

It is better to organize the data by clustering and indexing to speed up the CRUD operations in DBMS. There are many existing spatial accessing methods and they could be mainly classified into two groups which are discussed in details below. Since the distributed database is used, partitioning and distributing are other important issues to be considered.

3.2.1. Clustering and Indexing

1. Clustering: Clustering keeps closely related objects, often selected together due to spatial proximity, stored together to enhance retrieval efficiency.
2. Indexing: Indexing swiftly locates storage locations for specific objects, optimizing how spatial data is sorted for quick searches.

Spatial clustering is to use spatial locality to store the closer elements in the real world (or user-defined space) and also closer in the storage media which linearly organizes data. The locality is usually application-dependent and specified by clients. There are several ways to define the closeness such as SFC (van Oosterom and Vijlbrief, 1996; Gaede and Günther, 1998), and some machine learning algorithms such as K-Means could be used, too.

Dynamically Balanced Search Tree - Explicit Tree: It is exemplified by R-Tree and its variants (R*tree, R+tree, Hilbert R-tree, etc.), which dynamically consider the distribution of objects during dimensional space partitioning (van Oosterom, 1999; Mahmood et al., 2019). The primary advantage lies in achieving a balanced index structure, flexibly maintaining objects within each node (both leaf and intermediate nodes), thereby contributing to retrieval performance.

However, the construction and updating of this balanced tree index are intricate, posing challenges for practical deployment, especially in distributed environments. As the data volume increases, the depth of the index tree grows, resulting in a rapid decline in retrieval efficiency. Dynamically balanced search trees are well-suited for scenarios where the spatial distribution of objects is uneven, but their complexity in construction and updates make them less suitable for distributed environments or applications with rapidly increasing data volumes. Also, the requirement for big memory is needed (Li et al., 2020a).

Regular Dimensional Space Partitioning - Implicit Tree: It is represented by SFC, involving static regular subdivision of dimensional space (Liu, 2022; Guan, 2020). The primary advantage lies in its simplicity of construction and maintenance, eliminating the need for adjustments when adding new data and ensuring ease of use.

However, this approach struggles with the adaptive handling of data distribution. Some units represented by their indices may become too dense, while others may remain devoid of data, resulting in unstable retrieval efficiency. Regular dimensional space partitioning is well-suited for scenarios prioritizing easy construction and main-

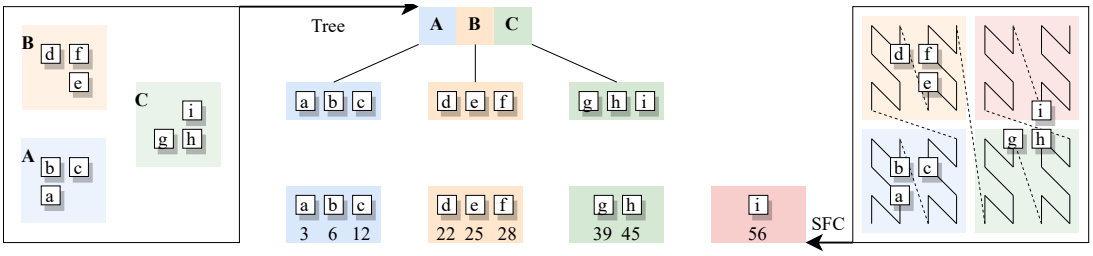


Figure 3.2. Comparison of Explicit and Implicit Tree Clustering: The left case shows the R-Tree method which is adaptive while the right case shows the SFC method which is rigid.

tenance, particularly in environments where data distribution remains relatively stable, and frequent adjustments to the index structure are undesirable.

In seeking a compromise between the two mainstream accessing methods, SFC incorporating spatial distribution, and multi-level indexing hybridizing multiple strategies emerge as potential solutions. It becomes evident that no single method universally suits all scenarios. The optimization for specific refinement and complexity may result in diminished performance in alternative applications.

Notably, while regular methods may excel in distributed environments, each approach presents trade-offs, emphasizing the importance of selection based on the specific demands and characteristics of the given data. In conclusion, the diverse nature of spatial indexing methodologies underscores the need for a understanding and careful consideration of trade-offs to ensure optimal performance in varied scenarios.

3.2.2. Partitioning and Distributing

The first question comes up when storing data in the distributed database is how to partition and distribute the data. This is achieved by partitioning the whole data into fragments and allocating unrelated objects evenly across different machines for load-balancing while maintaining data locality within each node to optimize distributed environments and manage larger datasets efficiently (Özsu et al., 2020).

Assume the regular dimensional space partitioning method is used, similar to what has been discussed in Sub-section 2.3.3 there are two types of partitioning strategies, the block-based method and the sample-based method, their differences can be shown in Figure 3.3.

It can be seen that the sample-based method would give a more balance distributed data over the four nodes which is a representation of good load-balancing. For a shape query, the sample-based method would distribute the task over all the nodes and possibly speed up the retrieval process especially when some post-processing is needed.

A more intuitive illustrations of the two partitioning strategies are shown in Figure 3.4

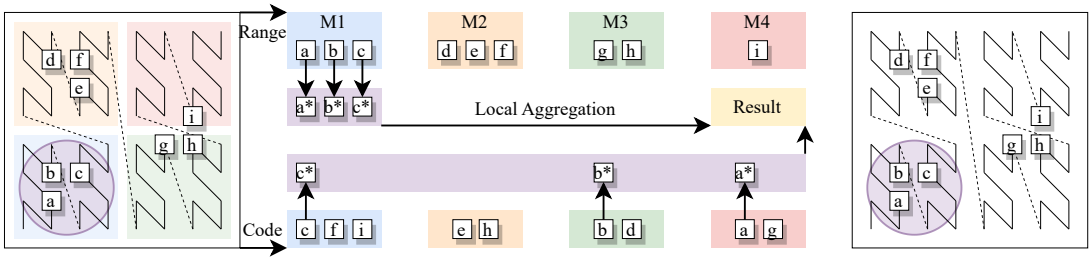


Figure 3.3. Comparison of Partitioning Strategies: The left case shows the block-based partitioning while the right case shows the sample-based partitioning, the purple shape is the query shape used.

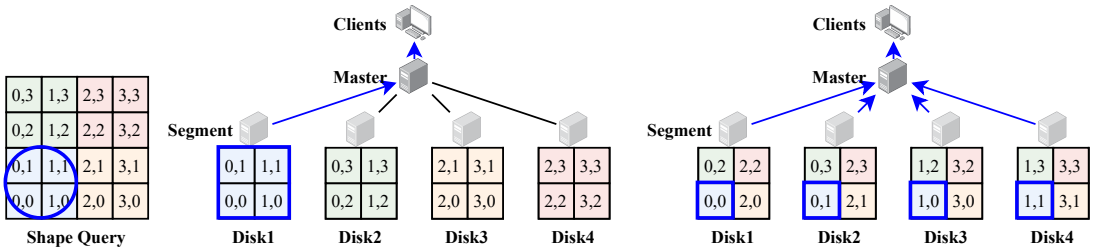


Figure 3.4. More Intuitive Partitioning Strategies: The blue circle represents the shape to be queried and the blue arrows represent the data transmission. The left case shows the data flow of block-based partitioning while the right case shows the data flow of the sample-based partitioning.

3.3. Trajectory Querying

There are a lot of different kinds of queries that are needed for the trajectory data applications. Only the ones that are implemented are shown below.

3.3.1. Querying by Attributes

Querying by Identifier: It selects the features that belong to specific objects, in the taxi trajectory case, it is to select the records by the unique identifier of the vehicle.

Querying by Semantics: It selects the features with certain attributes, in a taxi trajectory example, it is to select the records that the taxi is working.

3.3.2. Querying by Shape (Polytope)

Selection by Containment: It selects the features that are contained inside the querying shapes (such as polyhedron or sphere). It is also called range selection. An in-depth research was done in the PhD thesis of Liu (2022) and the comparisons of them are shown in Table 3.6.

Selection by intersection: It selects the features that intersect with the querying features (such as lineString or polygon), it could be considered as a special case of the containment selection.

Table 3.6. Comparison of Different Shape Querying Solutions

Solution	Operation	Certainty	CPU Cost	IO Cost
Vertex	All corners direction test	High	High	Low
Sweep	Two corners direction test	Medium	Medium	Medium
Sphere	One centre distance test	Low	Low	High

3.3.3. Querying for Visualization

The below three kinds of queries could mainly be used for visualization. Note that for the preparation of the visualized data, some post-processing computations are needed which can be localized.

Aggregation: Aggregation provides a comprehensive overview of the spatial-temporal cube, essential for understanding large datasets holistically.

Projection: Projection enables users to view 2D maps with each cell representing aggregated values over a period, offering a simplified perspective.

Simplification: Simplification allows users to visualize simplified line strings. Simplification is used to reduce transmission costs while keeping the general geometries.

Examples of these queries can be seen in the Figure 6.10.

3.4. Existing Products

Some trajectory management systems do exist. However, some of them are still limited in the centralized database domain and some of them care little about the properties of the trajectory data.

HERMES: A Trajectory DB Engine for Mobility-Centric Applications The HERMES is a centralized database built upon Oracle with OGC-compliant spatial extension (Pelekis et al., 2015). The trajectory is modelled as a sequence, but the number of points of one sequence is limited by the size of the node of the TB-Tree (a specific accessing method adopted from Pfoser et al. (2000)) because they should fit in the page size of the operating system. Although this TB-Tree indexing method considers trajectory preservation, it does not consider spatial distribution.

MobilityDB: A Mobility Database Based on PostgreSQL and PostGIS The MobilityDB is a centralized database built upon PostgreSQL with PostGIS extension (Zimányi et al., 2020). The trajectory is modelled as a sequence, which is indexed by a space partitioning method implemented by the SP-GiST index. However, the spatial distribution is also not taken into consideration.

UlTraMan: A unified platform for big trajectory data management and analytics The UlTraMan is a distributed system built upon Spark with an embedded key-value store (Ding et al., 2018). The trajectory is modelled as sequence, but they are indexed by the STR-Tree which is also proposed by Pfoser et al. (2000). Though the load-balancing is taken into consideration, the spatial distribution of the data is not.

JUST: JD Urban Spatio-Temporal Data Engine The JUST is a distributed database built upon Hbase which means efficient read and write operations are also achieved (Li et al., 2021, 2020a). The trajectory is modelled as a sequence, but with a special space-partitioning method, named Z2T and XZ2T, which consider the different scales of different dimensions for different applications (some dimensions may be less selective). The distributing strategy is random to avoid hot-spot problem, however, the spatial distribution is still missed. It is a matured product for one of the biggest E-commerce companies Jingdong in China.

3.5. Chapter Conclusion

In this chapter, the related work about modelling, accessing, organizing and typical queries are presented and discussed. specifically, sequence-based modelling stands out because of its flexibility and potential for compression. The R-tree indexing methods are not suitable for trajectory indexing because of the data volume and the geometrical properties of the trajectories, thus, the SFC method is used instead. The heterogeneity is a special property of the spatial data, thus, a pseudo-random sampling method is used in the distributing process.

As for the queries, the querying by attributes, querying by shape and querying including post-processing should be implemented. The above three types of queries could well represent the requirements for trajectory data retrieval and analysis. A product analysis has also been done, and it can be concluded that there is still a limited implementation based on the MPP architecture, which is valuable for exploration. In addition, the spatial distribution of the trajectory data is seldom taken into consideration.

4. Methodology

Chapter Contents

This chapter gives a description of the methodology mainly using the visualization language for an immediate impression. The overview of the methodology is shown in Figure 4.1.

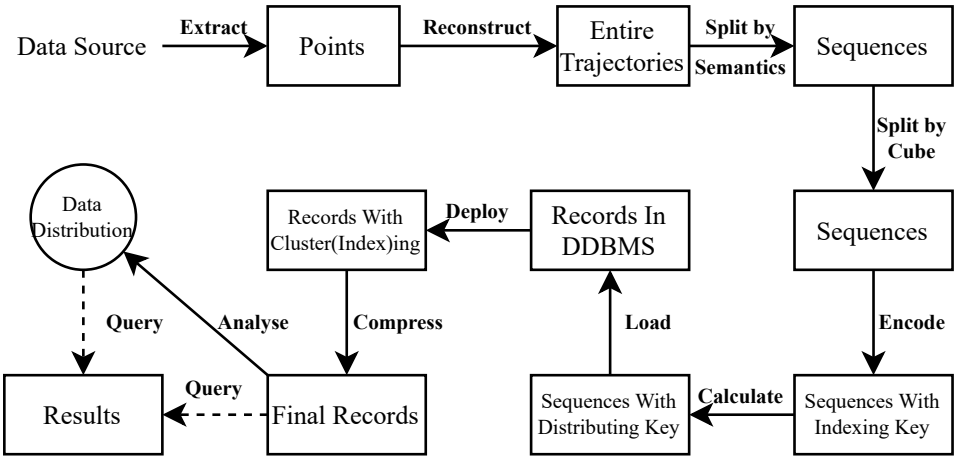


Figure 4.1. Overview of the Methodology: The boxes represent data in Comma-Separated Values (CSV) files, Pandas dataframes or database tables while the circle represents auxiliary data structure. The solid arrows represent processing operations while the dashed arrows represent the query. The workflow starts from the ETL (Extraction, Transformation and Loading) processes, the source data is processed in the form of sequence-based modelling while some other attributes such as indexing_key, distributing_key and state are also calculated. Then, the processed data is loaded into the database and some deployments such as distributing and indexing creation are executed, the analysis of data spatial distribution is also executed here. Finally, the queries could be executed with the help of the data spatial distribution.

Section 4.1 and Section 4.2 present the modelling of the trajectory data which is outside the database.

Section 4.3 presents the data loading and database deploying, the accessing and distributing are finally done here (the calculations are done outside the database).

Section 4.4 presents the data distribution metadata generation and the idea of querying data based on the data-aware SFC-based accessing method.

4.1. Trajectory Splitting

As discussed in the Section 3.1, the sequence-based modelling outperforms other modellings (point-based, segment-based and grid-based) because it preserves the geometries, supports more operations (e.g. intersection test with a face) and gives the potential of compression due to a certain degree of aggregation. The unstructured nature of sequences-based modelling makes the need to split the sequences (as line strings) semantically (e.g. attributes like states) and spatially by a regular spatio-temporal cube.

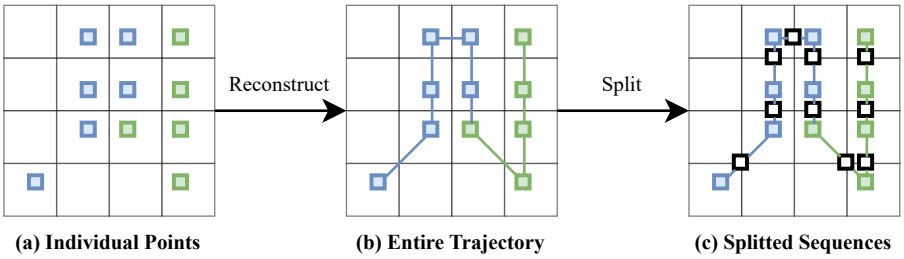


Figure 4.2. Trajectory Splitting: The blue and green dots represent trajectory points with different semantics while the black dots represent the intersected points with the edges. The blue and green lines represent trajectory segments with different semantics.

The general workflow of the trajectory splitting is shown below.

1. First, the point records of a specific trajectory are sorted by the time dimension. This is used to reconstruct the whole trajectory as the order of the points may be a mess in the source data.
2. Then, the reconstructed points should be split by the states, which means there is a need to group the points with the same states. This is achieved by a traversal of all the points, once a new state appears, a new group is created.
3. Finally, the sequences which cross several cells are split by the faces of the spatial-temporal cube. This could be considered as finding the points that intersected with the faces of the spatio-temporal cube.

After the tasks shown above, the original unstructured trajectory becomes more structured, the points of each sequence have the same semantics and the length of sequences is controlled (a sequence must be contained in a cell of the spatio-temporal cube).

4.2. Indexing and Distributing Keys Encoding

As discussed in the Section 3.2, the SFC-based indexing method is suitable for indexing not only because of its simplicity but also because of the nature of the modelling (the cell could approximately represent the space that a sequence occupied). Also, the coordinates of the cells could not be directly used for indexing because they are tuples (they have a certain order but the order is not good at locality preservation), thus, there is a need to encode the coordinate tuples of each cell to Morton code and assign this indexing key to the sequences that are inside the corresponding cell.

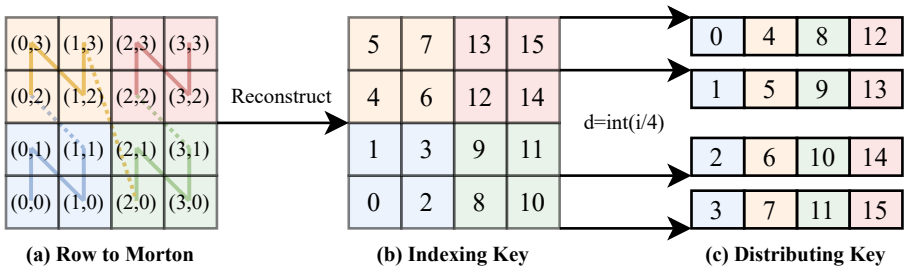


Figure 4.3. Keys Encoding: The color of the boxes represent their quadrants in the 2D space. The tuples in sub-figure (a) represent the coordinates of the cells while the numbers in sub-figure (b) represent the Morton codes. The sub-figure (b) means the whole data is partitioned and distributed into 4 groups by the distributing function.

To partition the whole dataset and distribute them evenly into nodes and enhance the localization (store closer objects also closer in storage media but separate distant objects), a distributing key is calculated based on the indexing key by division and int cast functions, which means distributing neighbouring elements in the same machines. By using this distributing strategy, the 8 neighbours (in our 3D case) could be accessed together without wasting the seeking time.

4.3. Data Loading and Deploying

After all the pre-processing outside the database, we data could be inserted or copied into the database. In addition to inserting (loading) the processed data into the database, there is a need to cluster and index the data and possibly do some compression.

1. As for the ID attribute: B-Tree is used for indexing because of the high cardinality of this attribute.
2. As for the indexing_key: B-Tree and BRIN are all tested for indexing because of the high cardinality of this attribute.
3. As for the state attribute: Bitmap is used for indexing because of the low cardinality.

The distributing_key is not indexed as they are not relevant to queries. Row-wise compression could be done as the points of one sequence are close. The time cost for

compression and decompression could be alleviated by powerful local CPUs in the distributed database.

4.4. Distribution Analysing and Shape Querying

After the data loading, there is a need to analyse the data spatial distribution and create an auxiliary data-aware data structure (adaptive Octree) as it can help end the tree traversal early to save time. To retrieve the data distribution, a full scan of the table is needed. This step could be done in the pre-processing step, however, the data is compressed using the database function, so it is better to use the data (size) distribution after the compression.

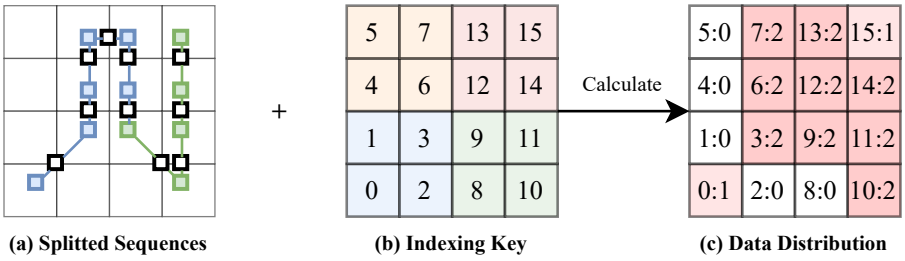


Figure 4.4. Distribution Analysing: The first two sub-figures are the same as the previous figures. The "*:*" in the sub-figure (c) represents the Morton code and the data size (in this example, the number of points) in each cell respectively.

The retrieved data distribution is used for Octree construction. Some parameters such as the tree construction depth and node size threshold should also be specified during the construction process. This process may be complicated so is detailed in the Chapter 5.

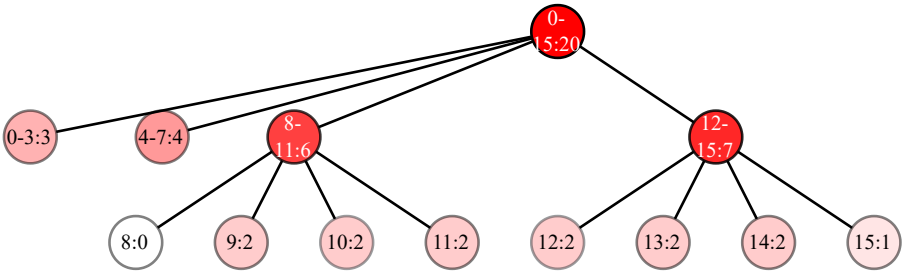


Figure 4.5. Octree Constructing: The opacity of the colour represents the size of the node. In this case, the threshold of the node size is set to 4, therefore, the first two nodes are not further subdivided.

After the deployment and the Octree construction. The spatial-temporal related queries could be done. Due to the implicit relationship between the Octree and the Morton curve, the general idea is to query the Octree and find the corresponding Morton

ranges (a range is composed of two extremes of Morton codes). Some parameters such as the tree search depth and node size threshold could also be specified which are different from the construction settings (the construction depth could be deep but the search depth could be shallower and the construction node size threshold could be fine but the search node size threshold could be coarser).

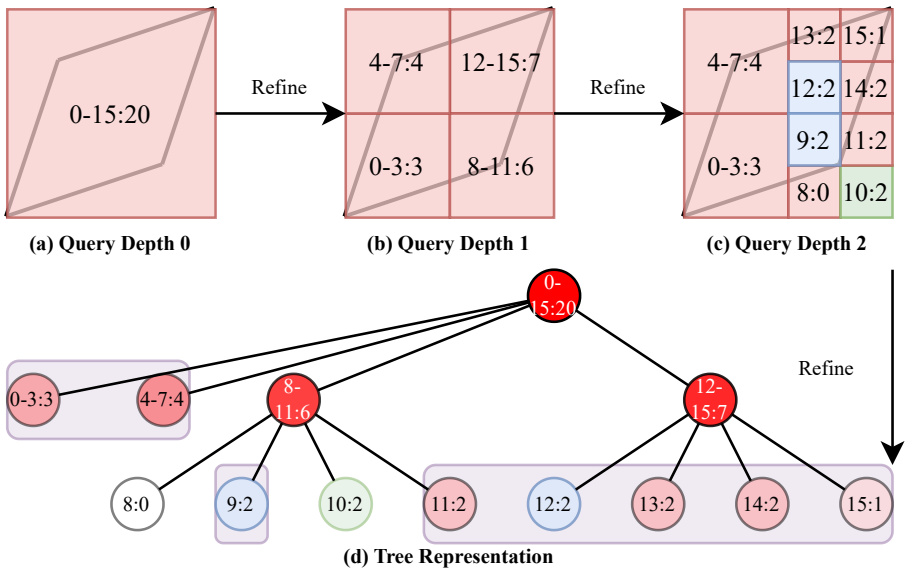


Figure 4.6. Shape Querying: The first three Sub-figures show the process of shape querying, the red boxes represent the nodes that intersect with the shape, the blue boxes represent the nodes that are fully contained in the shape while the green box represents the nodes that are outside the shape. In Sub-figure (d), the purple round boxes mean these nodes are selected. Note that the nodes with Morton code 8 are empty here. The nodes with Morton range 0-3 and 4-7 are small in size, thus, they are not further refined.

5. Implementation

Chapter Contents

This chapter gives the implementation details which are one-to-one mapping to the Chapter 4. Different from using the visualization language to give a glance, this chapter describes the exact details focusing more on the pseudo-codes of specific algorithms and many remarks during the implementation. The overview of the implementation is shown in Figure 5.1, and the entire project can be seen at Source Code.

Section 5.1 describes the necessary pre-processing steps, they can be considered as the data cleaning and encoding steps of the traditional extraction process in data warehousing.

Section 5.2 describes the steps for points interpolating (adding the intersected points with the cube faces) and the sequences splitting (subdividing the whole trajectory into pieces), they are the geometry-related operations.

Section 5.3 describes the steps for indexing keys (keys for doing indexing, the Morton code here) and the distributing keys (keys for partitioning and distributing) calculating, they are the SFC-related operations.

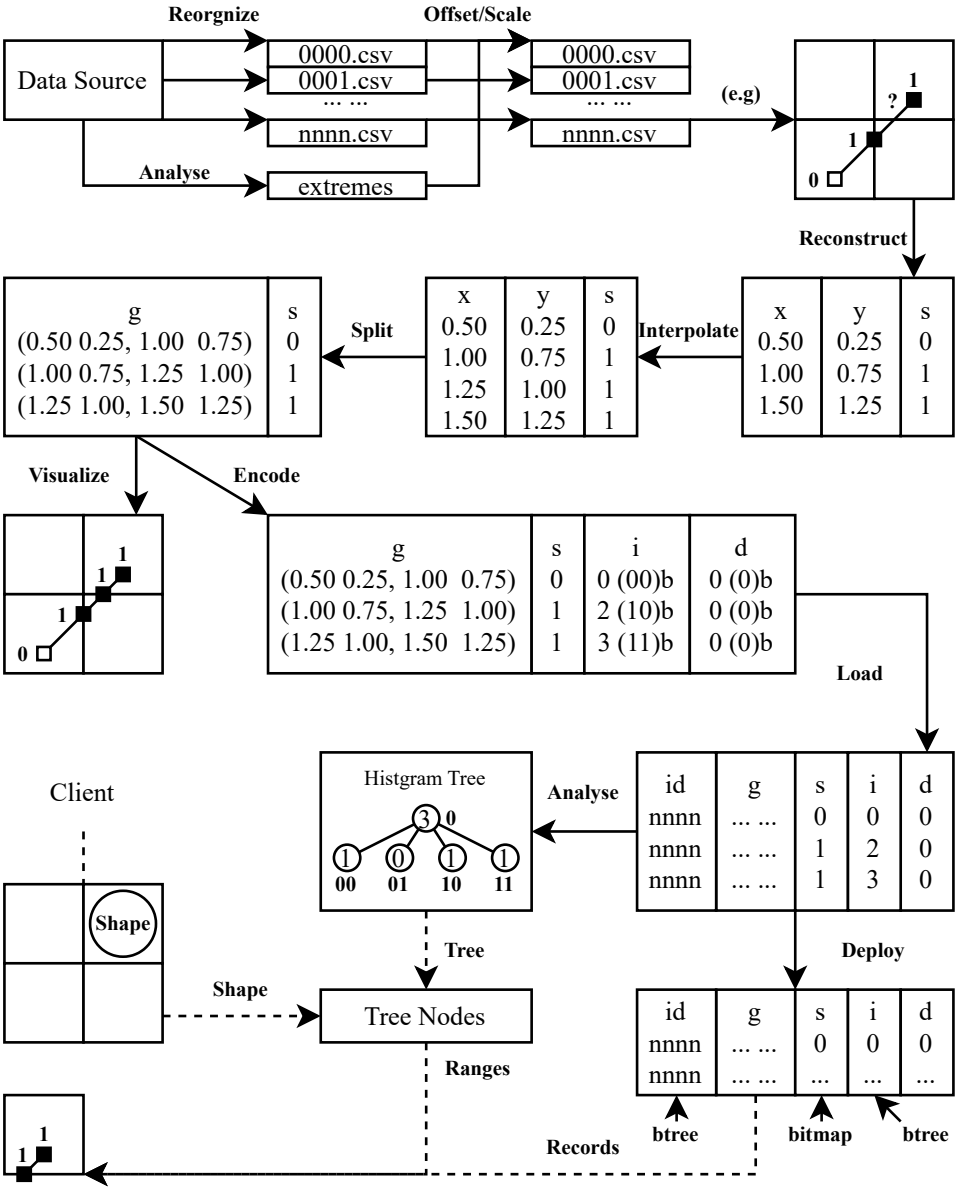


Figure 5.1. Overview of the Implementation: The single boxes represent data in the form of CSV files or specific data objects in Python. The collections of boxes represent data in the form of Pandas dataframes or data tables in DBMS. The 2D four quadrants diagrams represent the original data, processed data, query shape and queried data in a visualization form. The solid arrows represent the operations while the dashed arrows represent the data flow.

Section 5.4 describes the steps for data loading (from the processed file to the database). Also, some deploying manipulations (indexing, clustering, compression etc.) are also presented here.

Section 5.5 describes the steps for data spatial distribution analysis and the following shape querying process. Some other types of queries are also described here.

5.1. Pre-processing

5.1.1. File Reorganizing

Due to the large size of the original data and the limited memory resources, it is necessary to reorganize (split) the original file into several sub-files based on the ID of the moving objects like what is shown in the first line of Figure 5.1. It embodies the principle of divide-and-conquer: each time, independent pieces of data are processed individually, mitigating the out-of-memory issue.

Given a Pandas dataframe containing the original data with the schema shown in Table 6.2. The "reorganizing" can be directly implemented by the method groupby [1]. Note that the **Input** process could be optimized by using the chunk-by-chunk reading mechanism especially when loading big-sized data is slow. However, the **Output** should be taken more carefully as the records belonging to the same moving object may appear in different chunks.

5.1.2. Extremes Analysing

Note that the original file is the CSV file without standardization, it is necessary to do the data cleaning, avoiding the "NaN" values or duplicated records. Besides, all the attributes should be first transformed to decimal encoding: the time in the string should be transformed into a timestamp, and the geographical coordinates should be transformed into projection coordinates. These can be executed file by file without worrying about global dependence.

After the cleaning and re-encoding, for the preparation of the offsetting and scaling, there is a need to calculate the two extremes (minimum and maximum values) of the x, y and time dimensions. These values should be the global ones which means first the local extremes of one file are calculated, and then the global extremes are obtained by comparisons. The reconstructing step can be executed in the meantime: just sort the records in each file based on time.

[1] This algorithm is described in a (virtually) centralized way without parallelism consideration, and the IO process is omitted for simplicity. Many algorithms in this chapter can also be implemented in the distributed environment or with multi-core processing (Guan et al., 2018; Li et al., 2020a; Guan, 2020). However, since the performance is already acceptable for experimenting, they are omitted.

5.1.3. Offsetting and Scaling

To support the indexing method (based on the SFC) and for the convenience of splitting, the x, y and time dimensions should be offset and scaled.

Offsetting: For the offsetting, all the dimensions are offset to the origin of the space. An example for one dimension values is 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0. First, the offset value is the minimum value 1.0. Then, all the values are offset (minus) by 1.0. The resultant values are $0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0$.

Scaling: For the scaling, all the dimensions are scaled based on the depth of the Octree and the current maximum values. Based on the example above, and assuming the depth of the Octree is 2, therefore, the space should be subdivided 2 times and there should be $2^2 = 4$ intervals for that dimension.

Since the current maximum value is 6.0, all the values are scaled (timed) by $4/6.0 = 0.67$. The resultant values are $0.0, 0.67, 1.33, 2.0, 2.67, 3.33, 4.0$. After scaling, it is easy to directly identify which interval one value belongs to. For example, the value 0.0 belongs to the interval 0 and the value 3.33 belongs to the interval 3. To avoid the case 4.0 (interval 4 does not exist), the trick is to further offset the maximum value by a small step.

5.2. Trajectory Splitting

5.2.1. Interpolating

Instead of following the workflow mentioned in the methodology chapter (first do irregular splitting and then do regular splitting), a better method is to first find (interpolate) the points that intersected with the cube faces (called intermediate points) but leave the splitting operations later. The pseudo-code of the interpolating step is shown in Algorithm 1. Note that this process can be executed for each moving object individually without worrying about the dependence.

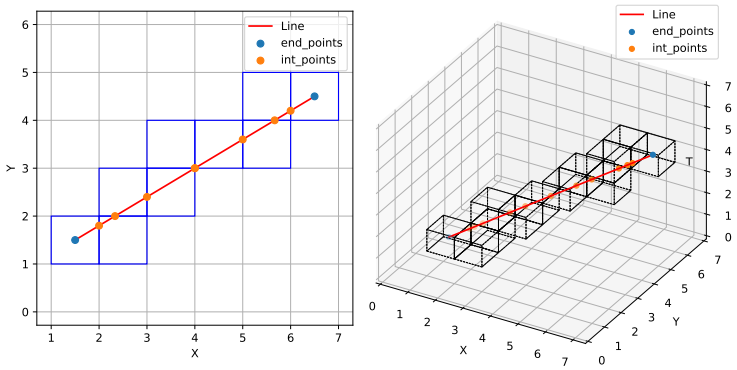
The calculation of the intermediate points is not the main focus and is quite complex. To give an easy understanding, the general idea and the examples of interpolating in 2D and 3D cases are shown in Figure 5.2. For the details, readers could refer to the Source Code: m-2: transform, Core: intermediate_points(), Core: next_point() and Core: get_cell().

5.2.2. Splitting

After the insertion of the intermediate points, the splitting can be executed considering the spatio-temporal related attributes and the semantic attribute (in this case, only a state attributed with boolean value) at the same time. The pseudo-code of the splitting step is shown in Algorithm 2.

Algorithm 1 Interpolating Algorithm

Input: A list of points P_I
Output: A list of points P_O
Begin:
for consecutive point pair p_i, p_{i+1} in P_I **do**
  if p_i and p_{i+1} are in the same cell **then**
    Append p_i into P_O
  else
    Calculate the intermediate points P_T
    Append p_i and P_T into P_O
  end if
end for
Append the last point p_n of P_I into P_O
End



(a) 2d (b) 3d

Figure 5.2. Splitting Results: The calculation of the next point is based on the coordinate of the current point and the direction. First, for each dimension, the predicted next boundary (edge in 2D and face in 3D) could be obtained by the ceil function. Then, based on the predicted next boundaries, the predicted intersected points could be obtained. Based on the predicted intersected points, the predicted distance could be obtained. Among all the possible distances, the minimum one is the correct prediction.

Algorithm 2 Splitting Algorithm

Input: A list of points P
Output: A list of line strings L
Begin:
Initialize an empty line string l
for consecutive point pair p_i, p_{i+1} in P **do**
  if p_i and p_{i+1} are in the same cell and with the same state **then**
    Append p_i into l
  else
    Append p_i and p_{i+1} into l
    Append l into L
    Empty l
  end if
end for
Append l into L
End

The result is a list of line strings and the points inside each line string are guaranteed to be in the same cell and are in the same state. The indices (a tuple of the 3 coordinates) of the cells and states should be assigned to that line string (the result is a table with 3 columns: index, state, geometry).

5.3. Indexing and Distributing Keys Encoding

5.3.1. Indexing Key

Note that the resultant indices above are tuples with 3 components (x, y and time) which are not suitable for the indexing use in the database. Therefore, these tuples should be transformed into 1D values (Morton code) based on the depth of the Octree. The mapping rule of the tuples and Morton code is already explained in Subsection 2.3.2. For the details, readers could refer to the Source Code: m-2: transform and core: encode_morton().

```
def encode_morton(x, y, t, depth):
    x_b = bin(x)[2:].zfill(depth)
    y_b = bin(y)[2:].zfill(depth)
    t_b = bin(t)[2:].zfill(depth)
    m_b = ""
    for i in range(depth):
        m_b += x_b[i] + y_b[i] + t_b[i]
    return int(m_b, 2)
```

5.3.2. Distributing Key

The distributing key is calculated based on the resultant indexing key above and the method is also already explained in Sub-section 2.3.3. For the details, readers could refer to the Source Code: m-2: transform.

5.4. Data Loading and Deploying

5.4.1. Data Loading

Although the performance of insertion one record by one record is already acceptable for experimenting, the data loading process can be further improved by the external (foreign) table technique with parallelism (specifically gpfdist function in Greenplum). The whole workflow starts with moving or copying the resultant CSV file above to the server (it could be the master, segments or a specialized server dealing with Extraction, Transformation and Loading (ETL) issues). Then, run the command shown below in the server where the dataset resides to enable the gpfdist function.

```
gpfdist -d /home/gpadmin/ -p 8081 -m 1048576
```

In the master node, use Structured Query Language (SQL) statement to create an external table with the same Data Definition Language (DDL) as the real table but point to the location of the CSV file.

```
CREATE READABLE EXTERNAL TABLE traj_external (
    taxi_id BIGINT,
    ......
    geometry GEOMETRY(LINESTRINGZ)
    ) LOCATION ('gpfdist://192.168.59.101:8081/test.csv')
    FORMAT 'csv' (HEADER)
    LOG ERRORS SEGMENT REJECT LIMIT 50 ROWS;
```

In the master node, use SQL to insert the data from the external table into the real table, this is different from simple insertion as the process is optimized and parallelized.

```
INSERT INTO traj
SELECT * FROM traj_external;
```

5.4.2. Database Deploying

After the above process, more mechanisms of Greenplum could be used, for example, data compression.

```
CREATE TABLE traj_compressed(
    taxi_id BIGINT,
    ......
    geometry GEOMETRY(LINESTRINGZ)
) WITH (APPENDONLY = TRUE, COMPRESSTYPE = ZSTD, COMPRESSLEVEL = 19);
```

The partitioning and distributing could also be executed in this step.

```
CREATE TABLE traj_distributed(
    taxi_id BIGINT,
    distributing_key INT,
    ......
    geometry GEOMETRY(LINESTRINGZ)
) DISTRIBUTED BY (distributing_key);
PARTITION BY RANGE (taxi_id);
```

All the techniques could be combined in the SQL command of data table creation. And the indexing and clustering can follow.

```
CREATE INDEX traj_index_indexing_key ON traj USING brin (indexing_key);
CREATE INDEX traj_index_taxiid ON traj USING btree (taxiid);
CREATE INDEX traj_index_state ON traj USING bitmap (state);
CLUSTER traj USING traj_index_indexing_key;
```

5.5. Distribution Analysing and Shape Querying

5.5.1. Distribution Analysing

Distribution Querying: After the data loading, a full scan of the table with the SQL function pg_column_size(geometry) could give the data distribution with the finest resolution.

```
SELECT indexing_key, sum(pg_column_size(geometry)) AS size
FROM traj
GROUP BY indexing_key;
```

After querying the result as a dataframe in Python, there is a need to gradually aggregate (each time group 8 neighbour cells in a bigger cell) the data distribution into coarser resolution till there is only one indexing_key (one cell containing all the data). For the details, readers could refer to the Source Code: m-4: analysis.

Octree Constructing: After the retrieval of the data distribution, the data distribution could be used for the adaptive Octree construction. The definition and initialization of the Octree are quite complex, for the details, readers could refer to the Source Code: Core: nDNode and Core: nDTree. This is only the general idea, a lot of SFC-related mappings should be done, note that other parameters such as the tree depth limitation and node size threshold could early stop the subdivision of the tree nodes.

5.5.2. Shape Querying

Shape querying is one of the most important applications, therefore, it is introduced emphatically, the general idea is shown in Algorithm 3.

Algorithm 3 Querying Algorithm

Input: A querying shape S, A queried Octree T
Output: A list of ranges R
Begin:
Traverse T in a depth-first order
if the shape is inside the tree node **then**
  Continue to refine this tree node
else if the tree node is inside the shape **then**
  Append the corresponding range into R
  Omit all the children of the tree node
else if the tree node is outside the shape **then**
  Omit all the children of the tree node
else
  Continue to refine this tree node
end if
End

Note that for the first few iterations, there may be cases that the shape is fully contained in the node which should be taken into consideration. Besides, there are other parameters such as the search tree depth limitation that could early stop the tree traversal process. For the details, readers could refer to the Source Code: Core: nDTree: shapeQuery(), Core: point_box_relation(), Core: check_relation() and Core: tree_shape_relation().

The explanation of how to determine the relationship between the node and shape is omitted here. However, it is the foundation of the implementation, a comprehensive discussion and implementation can be found in the PhD thesis of Liu (2022). After the list of ranges is obtained, the consecutive ranges with these same tags (inside, intersected or outside) can be directly merged. More discussions about the range merging are shown in Appendix D.

```
condition_template = '(%s␣<=␣indexing_key␣AND␣indexing_key␣<=␣%s␣)'
conditions = []

for r in ranges:
    condition_str = condition_template % (r[0], r[1])
    conditions.append(condition_str)

condition = '␣OR␣'.join(conditions)
```

The expression should fit into the WHERE condition of the SQL statement below.

```
SELECT *
FROM traj
WHERE ({}) AND (geometry IS NOT NULL);
```

There are two methods to do the selection based on the ranges, the first one is to encode the WHERE expression inside the SQL which is called the "expression" method shown above, the {} is a placeholder for the condition. The second one is to first create a

range table and use the table join method with BETWEEN expression which is called the "range join" method shown below.

```
SELECT *
FROM traj t JOIN range r
      ON t.indexing_key BETWEEN r.left_key AND r.right_key;
```

5.5.3. Other Queries

Aggregation: The result of the aggregation function is a 3D cube. The idea is to use the group by function in SQL to aggregate the records with the same indexing key. It is also possible to aggregate the 3D cube to a coarser resolution (implemented by */8 :: INTEGER*).

```
CREATE OR REPLACE FUNCTION aggregation(indexing_key INTEGER, depth INTEGER)
RETURNS INTEGER[]
AS
$$
    m_b = bin(indexing_key)[2:].zfill(depth * 3)
    x_b, y_b, t_b = "", "", ""
    for i in range(depth):
        x_b += m_b[3 * i]
        y_b += m_b[3 * i + 1]
        t_b += m_b[3 * i + 2]
    return [int(x_b, 2), int(y_b, 2), int(t_b, 2)]
$$ LANGUAGE plpythonu;
```

The {} in the SQL statement is the place holder for tree depth.

```
SELECT aggregation(indexing_key / 8::INTEGER, {}) AS coordinate, sum(st_leng
FROM traj
WHERE geometry IS NOT NULL
GROUP BY indexing_key / 8::INTEGER;
```

Projection: The result of the projection function is a 2D raster. The idea is to remove one dimension of the 3D cube by accumulating all the values on this axis to a plane.

```
CREATE OR REPLACE FUNCTION projection(indexing_key INTEGER, depth INTEGER)
RETURNS INTEGER[]
AS
$$
    m_b = bin(indexing_key)[2:].zfill(depth * 3)
    x_b, y_b = "", ""
    for i in range(depth):
        x_b += m_b[3 * i]
        y_b += m_b[3 * i + 1]
    return [int(x_b, 2), int(y_b, 2)]
$$ LANGUAGE plpythonu;
```

The {} in the SQL statement is the place holder for tree depth.

```
SELECT projection(indexing_key, {}) AS coordinate, sum(st_length(geometry)) A
FROM traj
WHERE geometry IS NOT NULL
GROUP BY projection(indexing_key, {});
```

Simplification: The result of the simplification function is a set of generalized line strings. The st_simplify function is used. This is usually used for visualization to reduce the network (data transmission) cost and latency.

```
SELECT st_simplify(geometry, 10) AS geometry
FROM traj;
```

6. Experiment and Validation

Chapter Contents

This chapter gives the designs and results of experiments to confirm the validity of the methodologies presented in the previous chapters.

Section 6.1 first shows the hardware and software used for the experiments and then the original and transformed experiment data schemas.

Section 6.2 shows the experiments done in the centralized database. It first gives an analysis of the spatial distribution, then evaluates the compression (storage) and selection (speed) performance of the sequence-based modelling and data-aware SFC-based indexing with varying spatio-temporal cube resolution (the depth of the Octree).

Section 6.3 shows the experiments done in the distributed database. It first evaluates the speed-up (constant problem size with increasing resources) and scale-up (increasing problem size with increasing resources) performance with varying numbers of nodes, then examines the possible benefits of data and computation localization.

At the end of this chapter, Section 6.4 concludes the lessons learnt from all the experiments.

6.1. Experiment Preparation

The hardware and software settings for the experiments are shown below:

Hardware Information: The hardware used is a personal laptop: ThinkBook 14 G5+IRH, CPU: 13th Gen Intel(R) Core(TM) i7-13700H 2.40 GHz, RAM: 32.0 GB, OS: Windows 11 Family.

Virtual Machines Information: The virtual machines are set up on VMware Workstation Pro 17. Their cores are CentOS Linux 7 with 4G memory, 4 CPU and 20G disk. The SSH client WindTerm is used instead of the native Linux command line.

Software Information: The distributed database is deployed on Greenplum 6.25.3. PostGIS 2.5.4 is used for spatial datatype and functions. Python 3.8.8 is used for data pre-processing and interaction with the database.

Table 6.1. Properties of Sample Data

property	value
number of taxis	1000
number of records	6655063
sampling interval	10s
spatial range	[113.4, 22.2, 113.4, 22.4]
temporal range	[2021-10-14 00:00:00, 2021-10-15 00:00:00]
total size	367.85MB

The trajectory data used here is the taxi data from Zhuhai City (China) with a 1,724.32 km^2 urban area and a 2,439,585 urban population. There are more than 3,000 taxis' data, with an average sampling interval of 10 seconds. 1,000 taxis' data in 2021-10-14 within the area of latitude from 22.2 to 22.4 and longitude from 113.4 to 113.4 is chosen as the experiment data. The total size is 367.85MB. The table form of the data properties is shown in Table 6.1. Heat-maps of the trajectory length in 2D and 3D are shown in Figure 6.10.

Table 6.2. Sample of Original Data

taxi_id	longitude	latitude	gps_time	state
7091110149	113.530618	22.22945	2021-10-14 00:00:03	2
7091110149	113.531048	22.228835	2021-10-14 00:00:13	2
7091110149	113.531411	22.228315	2021-10-14 00:00:23	2

For simplicity, only 5 attributes (they can also be considered as 5 dimensions: taxi_id, longitude, latitude, gps_time and state) are filtered out and a sample is shown in Table 6.2. After the transformation as described in the implementation chapter, the data was loaded into the database. A sample of the transformed data is shown in Table 6.3.

Table 6.3. Sample of Transformed Data

taxi_id	distributing_key	indexing_key	geometry	state
7091110149	12	100	LINESTRING Z (...)	0
7091110149	12	101	LINESTRING Z (...)	0
7091110149	12	102	LINESTRING Z (...)	0

6.2. Experiments for Modelling and Accessing

6.2.1. Distribution Test

The first experiment is about the spatial distribution properties of the taxi data with varying Octree depth.

Experiment Design: The independent variable is the depth of the Octree while the dependent variables are:

1. The empty ratio: It is calculated by the number of empty cells divided by the number of all the cells. It represents the sparsity of the data.
2. The global difference: It is calculated by the standard deviation of the cell sizes (including empty cells). It represents the global data skew.
3. The between-group difference: It is calculated by the standard deviation of the group sizes (the group size is measured by the mean of the 8-neighbour cell sizes). It represents the global data skew in a coarser resolution.
4. The within-group difference: It is calculated by the average of the standard deviations within all the groups. It represents the local data skew.

For the comparisons between different Octree depths, the later three metrics are normalized by the overall dataset size [1]. Given that the three metrics assess the differences at different resolutions, normalization should also account for the volumes of the cells and groups.

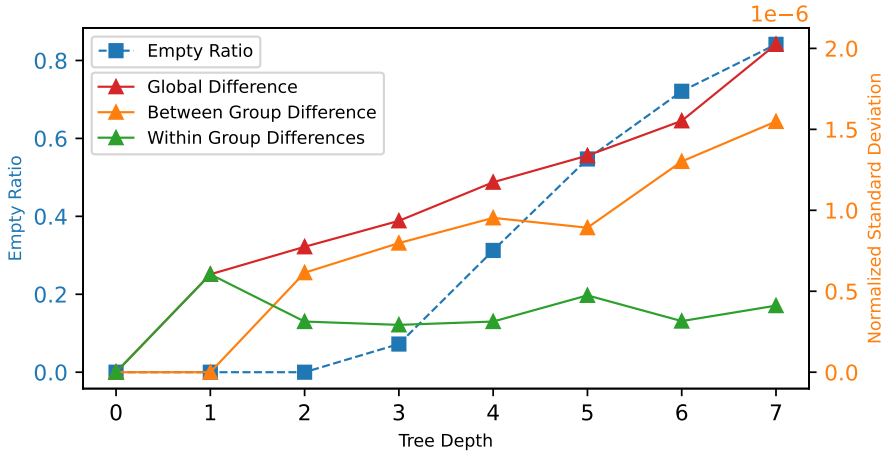


Figure 6.1. Distribution Test Analysis: With the increasing Octree depth, the empty ratio, global difference and between-group difference increase while the within-group difference remains rather stable.

Experiment Result: The experiment result is shown in Figure 6.1, it can be found that with the further subdivision of the space, the cells are becoming more and more sparse.

[1] Different resolutions lead to different redundant points which causes the differences in the overall dataset sizes, this is not taken into account for normalization (shown in Sub-section 6.2.2).

In the meantime, the spatial distribution is becoming more and more uneven as more and more empty cells are introduced [2]. However, the spatial distribution inside one subdivision can remain relatively stable which also confirms the previously discussed "global heterogeneity and local homogeneity".

Such severe sparsity and global heterogeneity should be carefully considered while the "global heterogeneity and local homogeneity" property of the taxi data could be potentially used.

6.2.2. Compression Test

The second experiment is about the compression performance of the sequence-based modelling with varying Octree depth.

Experiment Design: The independent variable is the depth of the Octree and the size of the original file is constant while the dependent variables are:

1. The table size without compression: It is calculated by the pg_table_size() function.
2. The table size with compression: It is the table size after ZSTD (with level 19) compression.
3. The compression ratio: It is calculated by the size of the original file divided by the size of the table with compression.
4. The B-Tree indexing size: The B-Tree indexing for the to the indexing_key attribute, it is considered as a table and calculated by the same function.

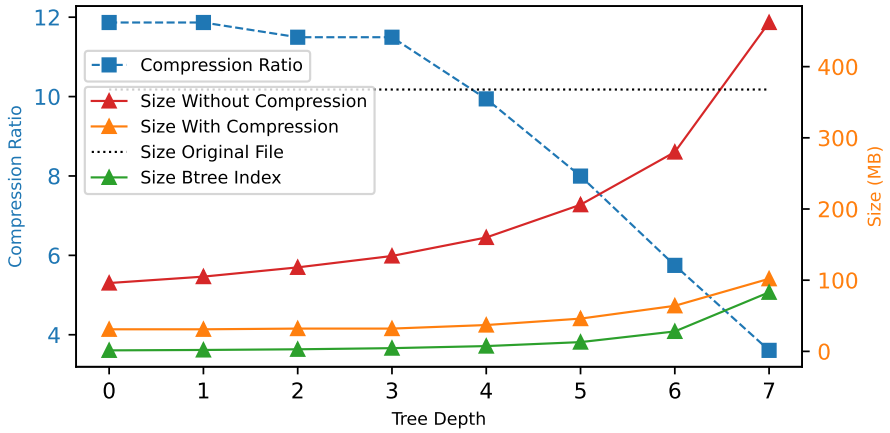


Figure 6.2. Compression Test Analysis: With the increasing Octree depth, the table size (with or without compression) and the indexing size all show a trend of first stabilization and then exponential growth. This leads to the quick drop of the compression ratio after Octree depth 4.

[2]The heterogeneity can be seen in Figure 6.10 (a) and (b) while the data will be sparse in the early morning and data will be concentrated in city centres (especially train stations and customs).

Experiment Result: The experiment result is shown in Figure 6.2, it can be found that with the further subdivision of the space, more and more intermediate points (intersections with the cube faces) are introduced which not only increases the size of the table in the database but also increases the number of the individual records leading to the potential danger of high cardinality (which can be seen from the increasing size of B-Tree). Fewer similar (adjacent) points in each record also lead to the deficiency of compression.

Making transformed data much bigger than the original file is unacceptable, thus, a smaller Octree depth may reduce the data storage burden. However, the selectivity due to the coarse subdivision should also be explored which is shown in Subsection 6.2.3.

6.2.3. Selection Test

The third experiment is about the selection performance with the data-aware SFC-based indexing with varying Octree depth. For simplicity, only the typical spatial selection (box containment selection) is conducted. Given a box (in this case, a box with a size of 20% by 20% by 20% of the whole space shown in Figure 6.5), a set of index ranges is generated to preselect the records by indexing_key.

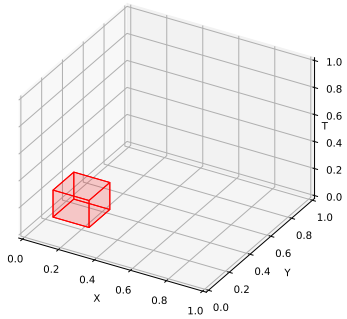


Figure 6.3. Selection Test Case Design: The two extreme corners of the box are (0.1, 0.1, 0.1) and (0.3, 0.3, 0.3).

Experiment Design: The independent variable is the depth of the Octree and the size and position of the box are constant while the dependent variables are:

1. The range number: The number of the resultant index ranges.
2. The record number: The number of the resultant (selected) records.
3. The time without indexing: The median time cost [3] for the selection without B-Tree indexing.
4. The time with indexing: The median time cost for the selection with B-Tree indexing.

[3] All the experiments related to time are done 10 times and the median values are chosen as the representatives.

Note that the time includes the time of execution and fetching but the time for generating index ranges (this process is done outside the database which is not comparable). Also, it is only for experiments, in real applications, the intersected and inside ranges should be further distinguished, the inside ranges could be directly used while the intersected ranges need further filtering (refinement).

In addition, there are two methods to do the selection, the first one is to encode the WHERE expression inside the SQL which is called the "expression" method while the second one is to first create a range table and use the join method with BETWEEN expression which is called the "range join" method. The query depth is set the same as the Octree depth while the node threshold is set to 100 (these parameters are not further explored in this research).

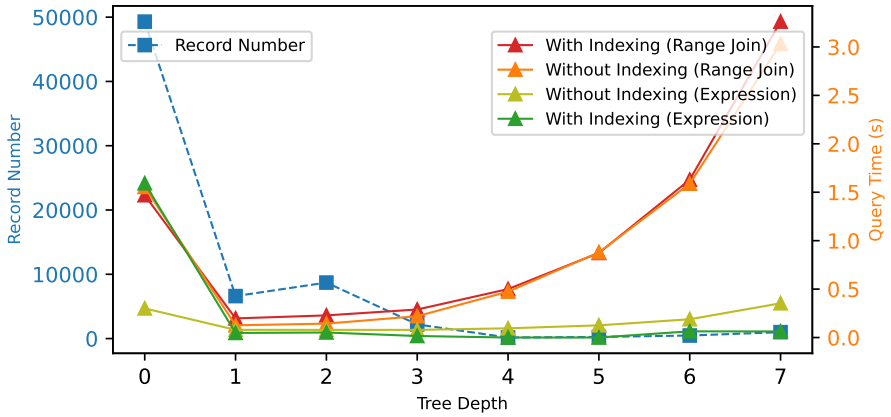
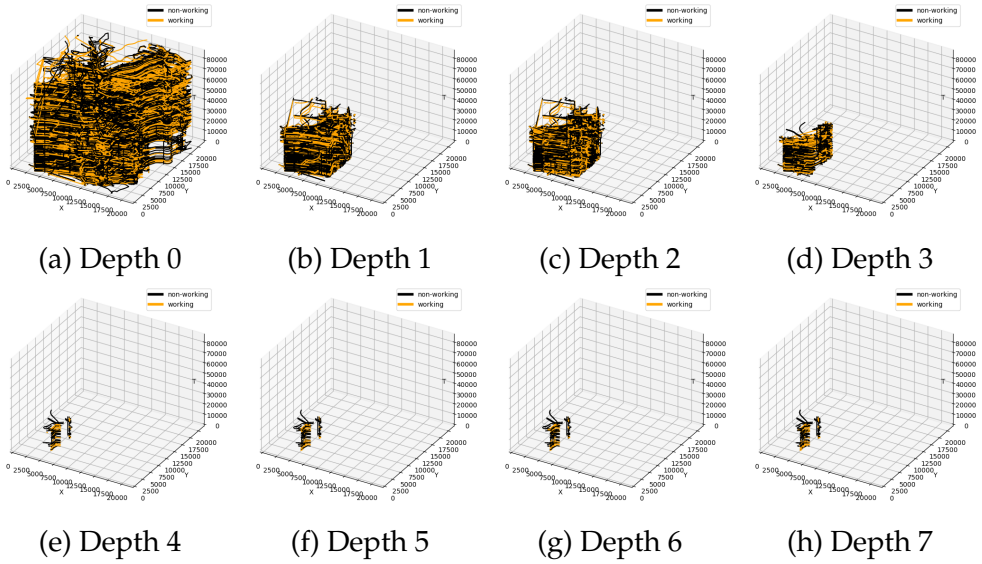


Figure 6.4. Selection Test Analysis: With the increasing Octree depth, the time costs and resultant record number all first decrease and then increase.

Experiment Result: The experiment result is shown in Figure 6.4, it can be found that with the further subdivision of the space, selection performance first increases and then decreases. It can also be found that the B-Tree indexing could reduce the time cost whether the "expression" method or the "range join" method is used.

The first improvement in performance comes from the progressively finer resolution while less redundant data is selected. For example, all the data is selected with Octree depth 0 as there is only one cell (shown in Figure 6.5(a)). The later decline of the performance comes from the increasing number of records (cardinality) and the increasing redundancy of the intermediate points (overall size). For example, from the perspective of visual interpretation, there is almost no difference between Figure 6.5(h) and the previous three Sub-figures, however, there are more records and bigger data size for the Sub-figure (h).

Surprisingly, the "expression" method could give a better result than the "range join" method which is perhaps because the number of ranges is small, with more ranges, the time consumption for the SQL statement transmission may be bigger. After observing the query plans, the "expression" method uses a method that replicates ranges to all the segments and then uses the sequential scans to all the indices while the "range-join" method uses a method that loops the ranges and then uses indexing scans to the



(a) Depth 0 (b) Depth 1 (c) Depth 2 (d) Depth 3

(e) Depth 4 (f) Depth 5 (g) Depth 6 (h) Depth 7

Figure 6.5. Selection Test Result: With the increasing Octree depth, the selectivity first increases (the set of ranges better approximates the shape) then the geometries of the selected data remain relatively the same.

indices. The "range-join" method may be more efficient as it avoids full table scans. However, due to the relatively small size of the data and ranges, the edge cost (such as range table creation or table join operation communications.) might cover the performance gains.

The BRIN indexing may not perform better than the B-Tree perhaps also because of the relatively small number of ranges. It can also be found that the Clustering would not usually introduce better results. At first, the range number is suspected as the reason for the later performance decline. However, in such a low-dimension case (3D), the range number does not increase after Octree depth 5 (depth 0-2: 1 range, depth 3: 10 ranges depth 4-7: 15-16 ranges). However, the number of ranges in higher dimensions would be huge (possibly more than 10,000 (Liu, 2022)) which can surely reduce the efficiency of the range join operation. A preliminary idea for merging ranges is discussed in Appendix D.

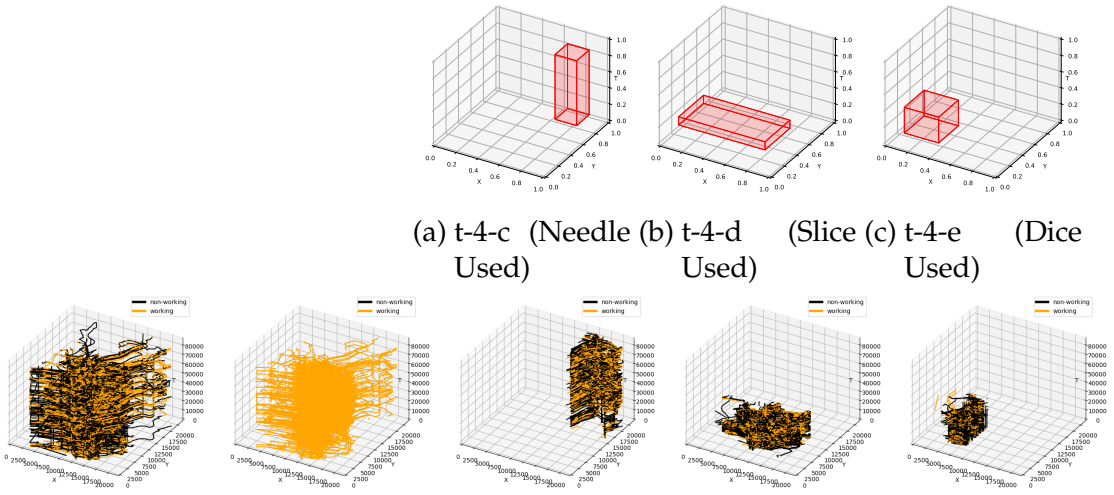
Some of the findings above are not further explored and explained in detail as they are query optimization issues. However, it is obvious that the decomposition of the space should not be too refined nor too coarse due to storage and speed reasons. For later distributed database related experiments, an Octree depth of 5 is a suitable start.

6.3. Experiments for Distributing

6.3.1. Speed-up Test

The fourth experiment is about the speed-up of different queries with the increasing number of nodes (hardware resources) in the distributed database. Five test cases are

designed which are shown in Figure 6.6.



(a) t-4-c (Needle Used) (b) t-4-d (Slice Used) (c) t-4-e (Dice Used)

(d) t-4-a (ID Selection) (e) t-4-b (State Selection) (f) t-4-c (Needle Selection) (g) t-4-d (Slice Selection) (h) t-4-e (Dice Selection)

Figure 6.6. Speed-up Test Cases Design and Result: Sub-figures (d) and (e) are the results of Selection by 100 taxi IDs and further filtered by state=0. Sub-figures (f), (g) and (h) are the Selection results of the box containment Selection sub-figures (a), (b) and (c)

Experiment Design: The independent variable is the number of nodes while the dependent variables are the relative speedup of all the queries.

1. t-4-a (Identifier selection): Select data by a set of taxi IDs (100 in this case).
2. t-4-b (State selection): Select data by a state. There are only two states (0 and 1), this query just adds a conditional statement WHERE state = 0 to the query t-4-a.
3. t-4-c (Spatio-temporal needle selection): Select the data inside a narrow space domain but cross a wide time domain.
4. t-4-d (Spatio-temporal slice selection): Select the data inside a narrow time domain but cross a wide space domain.
5. t-4-e (Spatio-temporal dice selection): It is a query that is between query t-4-c and query t-4-d.

Experiment Result: The experiment result is shown in Figure 6.7, it can be found that all tests show relatively good speed-up, a linear trend. After observing the query plans between ”single-node” and ”multi-node” queries, the main difference is that the ”multi-node” case could use parallelism which leads to the speed up.

Due to the limited hardware resources, only 4 nodes are used, it is expected that the speed-up curves will tend to be horizontal when reaching a threshold of node number. Among them, t-4-b may be negatively affected by the combined indexing search with B-Tree for taxiid and Bitmap for state.

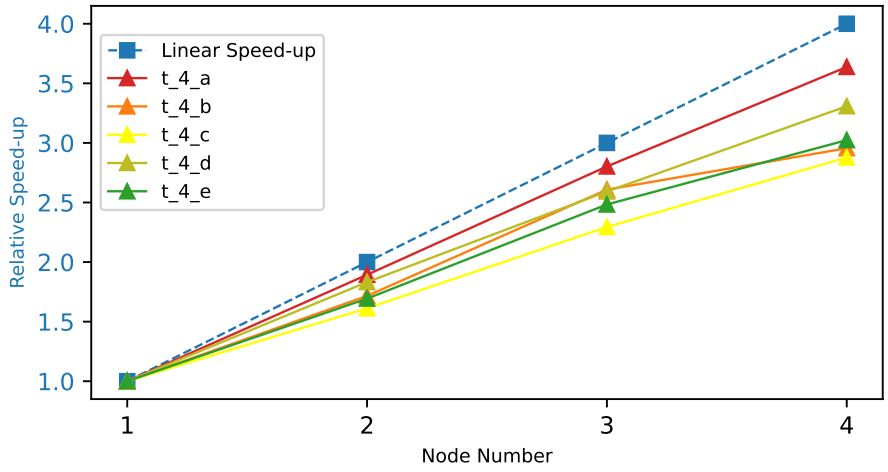


Figure 6.7. Speed-up Test Analysis: The blue dashed line represents the ideal linear relative speed-up, and the other solid lines represent the relative speedup of different queries (all of them are slightly worse than the ideal speed-up).

6.3.2. Scale-up Test

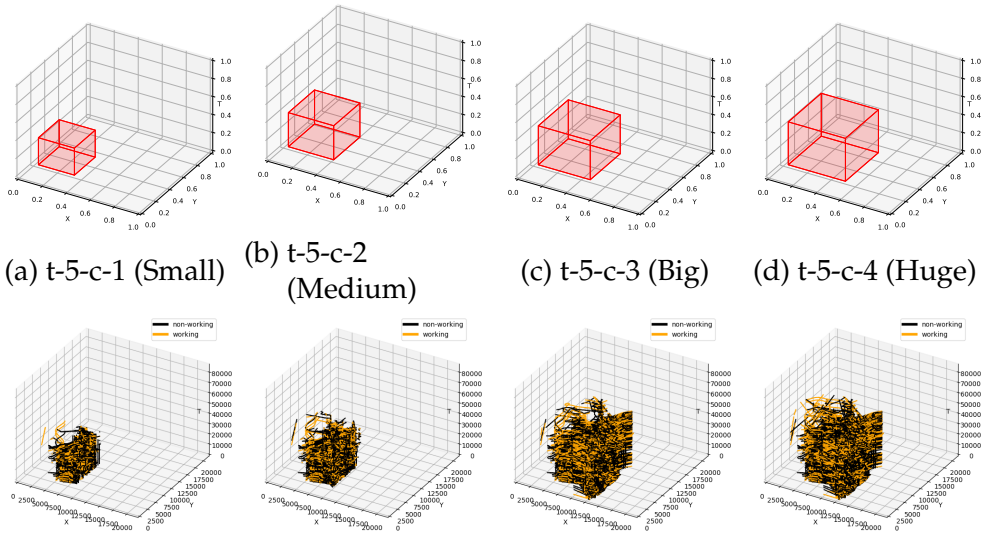
The fifth experiment is about the scale-up of different queries with the increasing number of nodes in the distributed database. It is different compared to the speed-up as not only the number of nodes would be increased, but the "size" of the queries (problems) should be increased.

Experiment Design: The independent variable is the number of nodes and size of the queries while the dependent variables are the relative scale-up of all the queries.

1. t-5-a (Identifier selection): It remains, but the number of IDs is changing from 100 to 400 based on the number of nodes.
2. t-5-b (State selection): Similar as before but with a changing number of IDs.
3. t-5-c (Spatio-temporal dice selection): It is the remained spatio-temporal selection with changing volumes shown in Figure 6.8.

Experiment Result: The experiment result is shown in Figure 6.9, it can be found that the relative scale-ups of all tests oscillate between 0.9 and 1.1 because of randomness, indicating a relatively good scalability.

Among them, the oscillation of t-5-c is relatively strong. After analysis, it is because although the volume scaling of the box is strictly controlled, the actual queried record number of the data (and also the data size represented by the black dashed line) does not show a strictly corresponding trend. For example, even if the box is extremely small, queries will also be slower if the data is very dense. It can be seen from here that spatio-temporal query performance is closely related to the spatial distribution of data which proves once again the need for the data-aware SFC-based indexing method.



(a) t-5-c-1 (Small) (b) t-5-c-2 (Medium) (c) t-5-c-3 (Big) (d) t-5-c-4 (Huge)

(e) t-5-c-1 Result (f) t-5-c-2 Result (g) t-5-c-3 Result (h) t-5-c-4 Result

Figure 6.8. Scale-up Test Cases Design and Result: Sub-figures (e), (f), (g) and (h) are the selection results of the box containment (spatio-temporal dice) selection sub-figures (a), (b), (c) and (d)

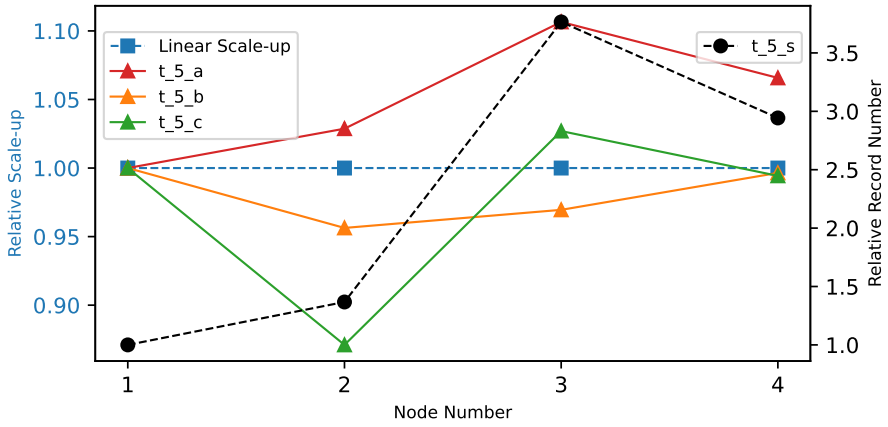
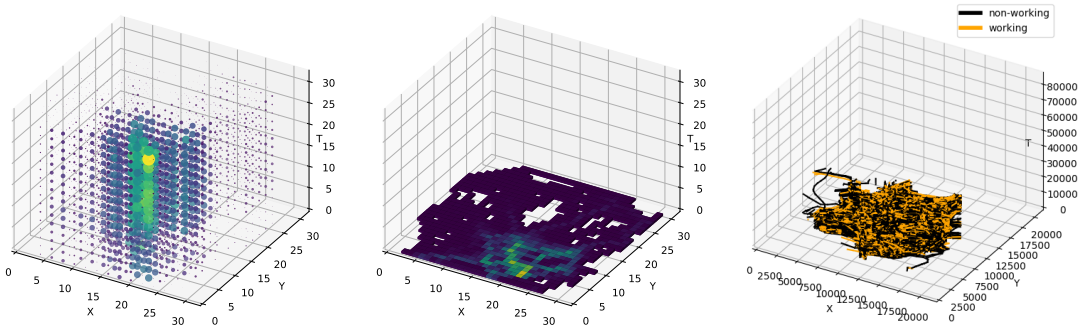


Figure 6.9. Scale-up Test Analysis: The blue dashed line represents the ideal linear relative scale-up, and the other solid lines represent the relative scale-up of different queries. Being lower than the blue dashed line represents better performance, and the black dashed line represents the relative record number resulted from query t-5-c, indicating a non-linear scale-up of the query size. Note that the figure is vertically stretched which exaggerates the fluctuation.

6.3.3. Localization Test

The last experiment is about the localization of data and computation with different distributing strategies (hashing distributing by indexing_key or by distributing_key). Three typical test cases are designed and are shown in Figure 6.10.



(a) t-6-a (Aggregation) (b) t-6-b (Projection) (c) t-6-c (Simplification)

Figure 6.10. Localization Test Result: The three subfigures show the results of aggregation to cube by scatters, projection to raster by flattened bars and the simplification.

Experiment Design: The independent variable is the number of nodes and the distributing strategies while the dependent variables are the time costs of all the queries.

1. t-6-a (Aggregation for visualization): Aggregate the numeric values of 8 neighbour cells to a spatio-temporal cube.
2. t-6-b (Projection for visualization): Project the numeric values (aggregate the number values by one axis, in this case, Z, to a plane) of cells to raster.
3. t-6-c (Simplification for visualization): Simplify the geometries of the result of the query t-4-d. The querying shape is chosen because simplifying all the data would be too time-consuming which will push down the curves of other queries.

Experiment Result: The experiment result is shown in Figure 6.11, it can be found that the query speed increases with the increase of nodes, which is consistent with the speed-up mentioned previously. However, the dashed and solid lines of each query almost overlap, proving that different data-distributing strategies do not significantly influence the performance of these queries.

This gives a negative result, inconsistent with the wanted design achievement. There are three possible reasons:

1. The most possible reason is that the Greenplum ignores the data distributing design, the data reshuffling over different nodes is done as usual even though all the data needed is in the same node which is shown in Figure 6.12. No localization is achieved.
2. The Greenplum does what is expected, however, the edge cost of the localization is too high to cover the benefits.

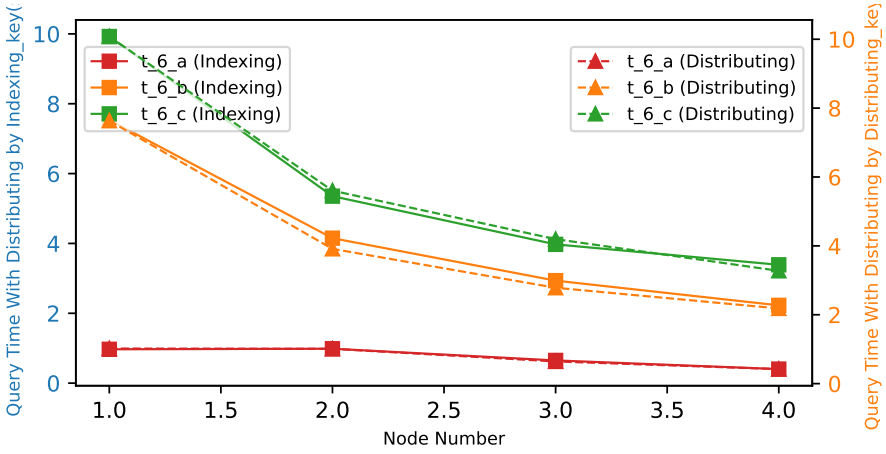


Figure 6.11. Localization Test Analysis: The solid line indicates that indexing_key is used for distribution, and the dashed line indicates that distributing_key is used for distribution.

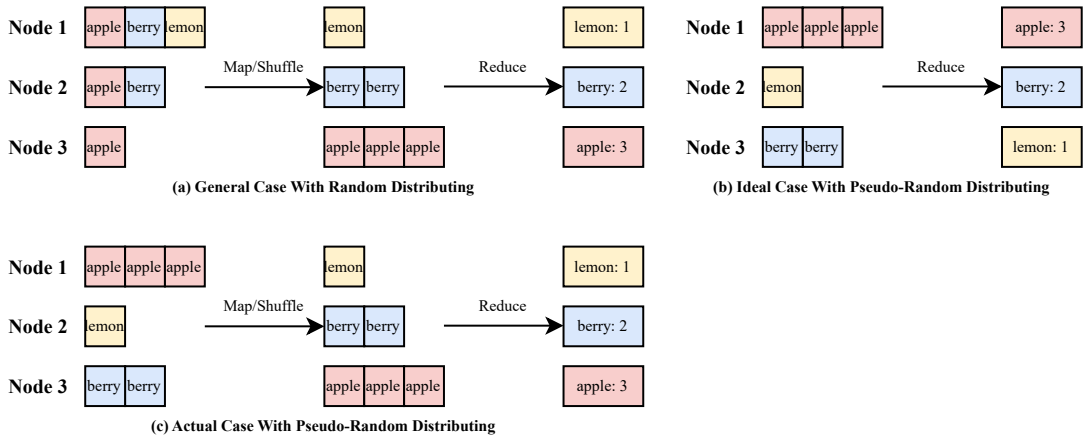


Figure 6.12. Ideal and Actual Localization Processes: Though the systems not implemented in Hadoop, the logic of aggregation in Greenplum must be similar to the MapReduce. An example is given for counting the number of fruits. Generally, shown in sub-figure (a), the data is reshuffled based on the keys (the names of the fruits), for example, all the apples are shuffled to Node 3, and then the counting is done in each node. If the data distribution is specially designed, as shown in sub-figure (b), the ideal case is that the counting could be directly done in each node without reshuffling. However, since the system does not know the designed distributing, the reshuffling is done as usual, negatively influencing performance.

3. The main bottleneck comes from the virtual machines such as data transfer time (network latency and bandwidth) from the server to the client.

Although the localization is not proven to be functioning, the distribution strategy can improve the loading-balancing (evenly distributed data over nodes) while preserving the locality.

Table 6.4. Comparison of Different Distributing

Attribute Used	Max	Min	Difference
Distributing Key	87,796	62,287	29.05%
Indexing Key	81,476	71,649	12.06%
Taxiid	84,699	65,029	23.22%
Geometry	90,150	71,617	20.56%
State	345,125	17,931	94.80%

The number of rows in each node could be calculated. A metric, namely the percentage difference between max & min could be calculated to evaluate the data skew which is shown in Table 6.4, the smaller, the better. Indeixng_key, taxiid or geometry could lead to good load-balancing, however, they can not help to preserve the locality. And it is quite clear that the state attribute is very bad for doing distributing.

6.4. Chapter Conclusion

The first three experiments are done in a centralized environment as they are not relevant to the distributed system features.

1. The distribution experiment is to analyse the spatial distribution properties of the taxi data with varying Octree depth.
2. The compression experiment is to test the compression performance of the sequence-based modelling with varying Octree depth.
3. The selection experiment is to test the selection performance with the data-aware SFC-based indexing with varying Octree depth.

It can be seen that the properties of spatial distribution are where spatial data are very significantly different from other types of data. For example, taxi IDs may be completely randomly distributed (or very regular because some of them belong to the same company) without physical meaning. These properties severely influence the performance of spatial-related queries, thus, a data-aware SFC-based indexing method is needed.

The resolution of spatial subdivisions should not be too coarse (reduce selectivity) nor too fine (increase cardinality and redundant data). This helps the developers and administrators balance the storage and speed based on the needs of certain applications. In addition, how to reduce data redundancy (through merging) is of value for exploration.

The other three experiments are done in a distributed environment to confirm the superiority of the distributed database.

1. The speed-up experiment is to test the speed-up of different queries with the increasing number of nodes (hardware resources).
2. The scale-up experiment is to test the scale-up of different queries with an increasing number of nodes and increasing side of queries.
3. The localization experiment is to test whether the localization of data and computation with different distributing strategies (hashing distributing by indexing_key or by distributing_key) would help.

It can be seen that introducing the distributed database and increasing hardware resources will improve performance. At least, the performance could come from the multi-node or multi-core processing. However, whether localization can help in the distributed system is still only theoretically valid.

7. Conclusion and Discussion

Chapter Contents

This chapter summarizes, concludes and discusses the full thesis in terms of the methodology, implementation, experiments, findings etc. from a higher perspective.

Section 7.1 first presents an overview of the work and contributions of the thesis. It answers the research questions with conclusions and discussions of the experiment results and findings.

Section 7.2 finally finishes by offering discussions about limitations and unresolved issues encountered during the thesis. Corresponding recommendations for future enhancements and directions are also given.

7.1. Summary and Contribution

This thesis aims to design and implement a distributed database solution to enhance the management and querying of trajectory data. The proposed questions in Chapter 1 are answered here.

7.1.1. Answers to Research Questions

As for sub-question 1: *How to perform the trajectory modelling? Is it better to model it as a sequence instead of a point cloud or grid?*

Instead of modelling trajectories as points or grids, it is better to be modelled as sequences mainly due to the application requirements (e.g. trajectory-face intersection test). Sequence-based modelling is more consistent with people's intuitive understanding of trajectory phenomena. Besides, this kind of modelling somewhat reduces the cardinality of the original data, enhancing the compression and later accessing performance.

As for sub-question 2: *How to perform the spatial accessing? Is it possible to use the SFC to do indexing and clustering with data awareness for a better querying performance?*

Drawing on the idea of PC data management, it is also possible to index the spatio-temporal aspect of trajectory sequences by the combination of SFC and B-Tree. This takes into account the dimensional and (linear) shape characteristics of the trajectory.

As for sub-question 3: *How to perform data distributing in distributed DBMS? Is it possible to use the SFC to do so and what are the potential benefits? Will the speed-up and scale-up of distributed databases be guaranteed?*

The distributed database (Greenplum) with the embedded spatial data type is beneficial to the performance with good scalability. It turns out that the SFC could not only be used for indexing but also used for distributing. A suitable data distribution can promote load balancing. However, enhancing the localization of data and computation with the designed distributing strategy is still hard without diving into the source code of Greenplum, although it is theoretically possible. The high degree of integration of the Greenplum system also brings flexibility issues with difficulties in customizing.

Till now, the **main question:** *What is the potential of integrating the SFC accessing methods and distributed databases for the efficient management of trajectory data with a huge volume?* is answered.

The experiments verified the possibility of SFC accessing methods and also proved the benefits of distributed databases.

7.1.2. Conclusion and Reflection

Different from PC solution (each point is directly encoded as the indices), the trajectories are split by the spatio-temporal cubes while each cube becomes the indexing. By doing experiments, it can be found that the subdivision resolution influences the data compression and data selectivity. A coarser resolution gives a better compression. However, the space-temporal queries are negatively influenced by a too-coarse or too-fine resolution.

Similar to the PC solution, the data distribution of the trajectory data is sparse and uneven which is also verified by the experiments, the data awareness of this spatial distribution is crucial for the efficiency of queries. By doing experiments, it can also be found that the data follows a globally heterogeneous but locally homogeneous spatial distribution which is used for the data localization. However, the histogram tree is easy to construct for points (the counts represent the nodes' size), which is not the case for the trajectory line strings. An adaptive modelling method is needed.

Distributed databases (more nodes or more hardware resources) can improve performance including the speed-up and scale-up for the trajectory data management, they are verified by positive experiment results. However, the data-distributing strategy (pseudo-random sampling) does not help for localization.

Finally, the main lesson learnt from this research is the "adaptiveness" concept. If extending a proven effective point cloud data modelling method to the trajectory data,

it is needed to do some adaptive changes. The geometry of the data is no longer a point without a volume but a line, making splitting a requirement.

$$\text{Distribution Awareness} \left(\begin{array}{ll} \text{adaptive modelling (trajectory splitting)} & + \\ \text{adaptive accessing (trajectory indexing)} & + \\ \text{adaptive distributing (trajectory partitioning)} & + \\ \text{adaptive querying (range merging)} & \end{array} \right) \times \text{Distributed Architecture}$$

In terms of performance, distributed databases can improve performance, but spatial data itself has unique characteristics of homogeneity and heterogeneity. Thus, adaptive changes are also needed to be made during the process of data accessing and distributing. This adaptiveness idea (although some are implemented and left for future work) was applied throughout the process of the study.

7.2. Future Work

7.2.1. Completeness Aspect

Due to the time limitation, there is still something to be done to make the research more complete.

Software Test: The current outcome is a simple demo with an experiment held on small test data. Other edge cases may exist because of the precision of floating point operations in splitting algorithms. Therefore, more test cases are needed to validate the robustness of the programs.

Mathematical Proof: Although some of the hypotheses are proved by the experiments, some methodologies are quite intuitive. There are risks in simply ignoring some other factors. Therefore, there is a need to use mathematical tools to make more serious proof. For example, "given a certain amount of data with certain dimensions and distribution, what resolution would be theoretically superior?"

Other Application: Although efforts have been made to cover as many application scenarios as possible. Many other types of queries are not implemented and experimented such as nearest neighbourhood search (Xu et al., 2018; Wang et al., 2019), matching and bundling (Holten and Van Wijk, 2009). Their interaction-intensive or recursion-intensive natures make the management task more complex considering the typical operation is one-turn data retrieval in the database domain. These operations are more suitable to the reals of big data or cloud computing. These more comprehensive test scenarios are needed for practical applications.

7.2.2. Applicability Aspect

As mentioned above, this project is a demo, to make it more applicable in the production environment, it is necessary to do system integration, realistic benchmarking and test the applicability for different sources of data.

System Integration: The current solution is implemented in Python, for a certain application, there is a need to interact with the Python and DBMS several times which does not show the real performance. Therefore, it is needed to integrate the auxiliary data structures (such as the adaptive Octree) and algorithms (such as Shape Querying) into the database by compiling language. Other kinds of supports should also be considered such as multi-user applications (for example, when the query is big, can the system dynamically allocate more resources to that user?), only in this way can it become a usable product.

Realistic Benchmarking: Current tests of solutions only compare themselves (with changing parameters) while the comparisons between other counterparts such as modelling based on point or implemented using different platforms (such as Hadoop) are not done. Therefore, it is needed to do a full benchmarking with other solutions in a real physical distributed environment instead of the virtual machines. Only then can it truly demonstrate the superiority of this solution rather than just pointing out a possibility. Also, there is a need for not only comparing cases with different node numbers but also other database settings such as different thread numbers.

Data Comparison: In this thesis, only taxi data is tested. Other types of data such as ship data and wildlife data may have different spatial distributions and other characteristics. These phenomena and differences in data types may bring certain impacts and influence the applicability of this solution, therefore, comparison with different data is also of certain significance.

7.2.3. Optimization Aspect

To achieve competitive advantages in performance, efficiency and innovation compared to other systems. It is necessary to optimize many of the factors such as data structures, algorithms and the combination of parameters.

Workflow optimization: The workflow (life cycle) of the data may be optimized. For instance, the trajectory splitting and the spatial distribution analysis could be integrated instead of separating to several steps. For example, a new "adaptive splitting" algorithm could avoid splitting some trajectories too much in places where the data is sparse.

Algorithm optimization: Many algorithms such as the splitting are implemented in a straightforward (brute) way. To reduce the time complexity of these algorithms, more advanced data structures and algorithms may be taken into account. There is also an issue with the range mering problem which is detailed in Appendix D.

Learning-based optimization: According to the design, there are many parameters. The combination of these parameters will lead to different performance. Learning-based tuning can solve this cost-based optimization problem for the optimal parameter combinations. At the same time, it is necessary to have a deeper understanding of the database optimizer.

A. Experiment Result

Chapter Contents

This appendix gives the results (in table form) of all the experiments.

A.1. Experiment Results for Modeling and Accessing

A.1.1. Distribution Test Result

Table A.1. Distribution Test Result

tree depth	0	1	2	3	4	5	6	7
gloabl_difference	0.0	6.05e-07	7.74e-07	9.35e-07	1.17e-06	1.34e-06	1.55e-06	2.02e-06
between_group_difference	0.0	0.0	6.15e-07	7.97e-07	9.53e-07	4.75e-07	1.30e-06	1.55e-06
within_group_difference	0.0	6.05e-07	3.13e-07	2.92e-07	3.13e-07	8.92e-07	3.16e-07	4.11e-07
empty_ratio	0.0	0.0	0.0	0.07	0.31	0.55	0.72	0.84

A.1.2. Compression Test Result

Table A.2. Compression Test Result

tree depth	0	1	2	3	4	5	6	7
size_original_file (MB)	367.85	367.85	367.85	367.85	367.85	367.85	367.85	367.85
size_without_compression (MB)	96.0	105.0	118.0	134.0	160.0	206.0	280.0	462.0
size_with_compression (MB)	31.0	31.0	32.0	32.0	37.0	46.0	64.0	102.0
size_btree_index (MB)	1.41	1.98	2.91	4.51	7.42	13.0	28.0	83.0
compression_ratio	11.87	11.87	11.5	11.5	9.94	8.0	5.75	3.61

A.1.3. Selection Test Result

Table A.3. Selectivity Test Result

tree depth	0	1	2	3	4	5	6	7
number_range	1	1	1	10	16	15	15	15
number_record	49303	6608	8710	2204	164	255	473	998
without_index_(expression) (s)	0.3	0.08	0.08	0.08	0.09	0.13	0.19	0.35
without_index_(range join) (s)	1.55	0.13	0.14	0.22	0.47	0.88	1.59	3.03
with_index_(expression) (s)	1.59	0.05	0.05	0.02	0.0	0.0	0.06	0.06
with_index_(range join) (s)	1.47	0.2	0.23	0.29	0.5	0.87	1.63	3.26

A.2. Experiment Results for Distributing

A.2.1. Speed-up Test Result

Table A.4. Speed-up Test Result

node number	1	2	3	4
t-4-a (s)	4.49	2.37	1.6	1.23
t-4-b (s)	1.45	0.85	0.56	0.49
t-4-c (s)	2.02	1.26	0.88	0.7
t-4-d (s)	15.36	8.39	5.93	4.64
t-4-e (s)	8.33	4.92	3.35	2.75

A.2.2. Scale-up Test Result

Table A.5. Scale-up Test Result

node number	1	2	3	4
t-5-a (s)	4.22	4.34	4.67	4.5
t-5-b (s)	1.51	1.45	1.47	1.51
t-5-c (s)	8.85	7.71	9.09	8.8
record number for t-5-c	12640	34567	142781	148807

A.2.3. Localization Test Result

Table A.6. Localization Test Result

node number	1	2	3	4
t-6-a (s)	0.97	0.99	0.65	0.4
t-6-a-d (s)	1.02	1.01	0.64	0.42
t-6-b (s)	7.54	4.15	2.93	2.23
t-6-b-d (s)	7.63	3.91	2.78	2.18
t-6-c (s)	9.92	5.35	3.97	3.39
t-6-c-d (s)	10.08	5.6	4.19	3.28

B. Virtual Machine Set-up Description

This appendix gives the setup steps for the Greenplum virtual machines.

B.1. Create Template Machine

Hardware Setup: The VMware Workstation 17 Pro is used. The configuration of the template machine is:

1. Name: greenplum100
2. Memory: 4G
3. CPU: 2 × 2
4. Disk: 20G (1G + 2G + 17G)
5. CD/DVD(IDE): CentOS-7.5-x86-1804
6. Network: NAT

ID Address Setup: The dynamically distributed IP cab be first checked by ip addr. The network configuration can be modified by vi /etc/sysconfig/network-scripts/ifcfg-ens33, while the changes below should be done.

```
BOOTPROTO="static"
IPADDR=192.168.59.101
GATEWAY=192.168.59.2
DNS1=192.168.59.2
```

After modification, the machine name can be checked by hostname and there is a need to restart the network services by systemctl restart network.

Libraries Setup: It is needed to install some libraries such as epel-release by yum install -y epel-release and other libraries by yum install -y vim net-tools psmisc nc rsync lrzsz ntp libzstd openssl-static tree iotop git.

Firwall Setup: For the convenience of later setup, it is better to stop the firewall and disable restarting by systemctl stop firewalld and systemctl disable firewalld. After the above settings, the firewall status can be checked by systemctl status firewalld.

Hosts Setup: For machine communication, there is a need to modify the host mapping file by vim /etc/hosts. Inside the file, add the mappings below:

```
192.168.59.101 greenplum101
192.168.59.102 greenplum102
192.168.59.103 greenplum103
192.168.59.104 greenplum104
```

```
192.168.59.105 greenplum105
```

After the settings, there is a need to reboot the machine by reboot.

B.2. Clone Other Machines

Based on the template machine created above, four machines can be directly cloned, the only thing that should be changed is their host names by vi /etc/hostname and IP addresses by vi /etc/sysconfig/network-scripts/ifcfg-ens33. Don not forget to reboot these machines by reboot.

B.3. Set System Environments

Libraries Setup: For all the machines, there is a need to install other dependent libraries by yum install -y apr apr-util bash bzip2 curl krb5 libcurl libevent libxml2 libyaml zlib openldap openssh-client openssl openssl-libs perl readline rsync R sed tar zip krb5-devel.

SELinux Setup: For all the machines, there is a need to disable SELinux by vim /etc/selinux/config and modify SELINUX=disabled.

Shared Memory Setup: There is a need to calculate the minimum and maximum shared memory for later usage.

1. Minimum value: echo $(expr $(getconf _PHYS_PAGES) / 2) which should be 482946 in this case.
2. Maximum value: echo $(expr $(getconf _PHYS_PAGES) / 2 $(getconf PAGE_SIZE)) which should be 1978146816 in this case.

After the calculation, the system configurations can be modified by vim /etc/sysctl.conf while the contents below should be added.

```
# Change based on the above calculations
kernel.shmall = 482946
kernel.shmmax = 1978146816
kernel.shmmni = 4096
# See Segment Host Memory
vm.overcommit_memory = 2
# See Segment Host Memory
vm.overcommit_ratio = 95
# See Port Settings
net.ipv4.ip_local_port_range = 10000 65535
kernel.sem = 500 2048000 200 40960
kernel.sysrq = 1
kernel.core_uses_pid = 1
kernel.msgmnb = 65536
kernel.msgmax = 65536
```

```
kernel.msgmni = 2048
net.ipv4.tcp_syncookies = 1
net.ipv4.conf.default.accept_source_route = 0
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.conf.all.arp_filter = 1
net.core.netdev_max_backlog = 10000
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152
vm.swappiness = 10
vm.zone_reclaim_mode = 0
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 100
# See System Memory
vm.dirty_background_ratio = 3
vm.dirty_ratio = 10
```

Assume the modification is done in greenplum101, there is no need to repeat the process in other machines, the best way is to send the file to all the other machines by scp -r /etc/sysctl.conf greenplum102:/etc/.

System Resources Limits Setup: The below settings should be added to two files: vim /etc/security/limits.conf and vim /etc/security/limits.d/20-nproc.conf.

```
*    soft     nofile   65536
*    hard     nofile   65536
*    soft     nproc    131072
*    hard     nproc    131072
```

Do not forget to send the files to all the other machines by scp -r /etc/security/limits.conf greenplum102:/etc/security/ and scp -r /etc/security/limits.d/20-nproc.conf greenplum102:/etc/security/limits.d/.

SSH Threshold Setup: Modify the SSH Threshold by vi /etc/ssh/sshd_config and do the below modification. After modification, do not forget to restart the SSHD by systemctl restart sshd.

```
MaxSessions 200
MaxStartups 100:30:1000
```

Language Set Setup: Change the language set by localectl set-locale LANG=en_US.UTF-8.

Time Unification: There is a need to unify the time over all the machines by ntpdate cn.pool.ntp.org.

B.4. Install the Greenplum

User Creation: First, it is better to create grennplum users for each machines.

```
groupadd gpadmin
useradd gpadmin -r -m -g gpadmin
passwd gpadmin
1234
```

Then, there is a need to grant power to the users by vim /etc/sudoers and do the below modification.

```
root    ALL=(ALL)      ALL
gpadmin ALL=(ALL)      NOPASSWD:ALL
```

SSH Setting: There is a need to set the SSH Login without password by creating key by ssh-keygen -t rsa and send the key to other machines by ssh-copy-id greenplum102.

Configuration Creation: There is a need to create some configurations in the master node (greenplum101) before the real installation by the below commands.

```
mkdir -p /home/gpadmin/conf
touch /home/gpadmin/conf hostlist
touch /home/gpadmin/conf seg_hosts
```

After the creation of these configurations, there is a need to add the contents to them. For the vim /home/gpadmin/conf hostlist, add the below content.

```
greenplum101
greenplum102
greenplum103
greenplum104
greenplum105
```

For the vim /home/gpadmin/conf seg_hosts, add the above content without the first line (master).

Real Installation: There is a need to first create a directory for the software by mkdir -p /home/gpadmin/software and transfer the Greenplum installation package.

For the Greenplum installation and the PostGIS installation, the below commands should be used in the master node.

```
sudo yum -y install ./open-source-greenplum-db-6.25.3-rhel7-x86_64.rpm
sudo chown -R gpadmin:gpadmin /usr/local/greenplum-db*
gppkg -i postgis-2.5.4+pivotal.8.build.1-gp6-rhel7-x86_64.gppkg
```

Connection Setting: T connect all the servers, the below commands can be used in master node.

```
source /usr/local/greenplum-db-6.25.3/greenplum_path.sh
gpssh-exkeys -f /home/gpadmin/conf/hostlist
mkdir -p /home/gpadmin/data/master
```

```
cat <<EOF >> /home/gpadmin/.bashrc
source /usr/local/greenplum-db/greenplum_path.sh
export PGPORT=5432
export PGUSER=gpadmin
export MASTER_DATA_DIRECTORY=/home/gpadmin/data/master/gpseg-1
export PGDATABASE=gp_sydb
export LD_PRELOAD=/lib64/libz.so.1 ps
EOF
```

To configure the environment variable GPHOME, first enter the file by vim /usr/local/greenplum-db/greenplum_path.sh and modify it directly.

```
GPHOME=/usr/local/greenplum-db
```

Then, some other configurations should be done.

```
gpssh -f /home/gpadmin/conf/hostlist
gpcheckperf -f /home/gpadmin/conf/hostlist -r N -d /tmp
mkdir /home/gpadmin/gpconfigs
cp /usr/local/greenplum-db/docs/cli_help/gpconfigs/gpinitsystem_config /home/
```

Real Initialization: Then the initialization can be done by gpinitsystem -c /home/gpadmin/gpconfigs/gpinitsystem_config -h /home/gpadmin/gpconfigs/hostfile_gpinitsystem. If something goes wrong, the system can be deleted by gpdeletesystem -d /home/gpadmin/data/m 1 -f

A database can be created by createdb trajectory and the echo "host all gpadmin 0.0.0.0/0 trust" ¿¿ /home/gpadmin/data/master/gpseg-1/pg_hba.conf and gpstop -u should be run for the remote connection.

Sometimes, some nodes may be shut down and the gpstop and gpstart should be used. To change the number of nodes, there is a need to first delete the system, and then modify the hostlist and seg_hosts files, and finally, initialize he system again.

C. Code Structure Description

This appendix gives the structure of all the code modules (the implementation and experiments), many of which are referred by the Chapter 5 as it is better to refer to the source codes rather than pages of complex pseudo-codes of algorithms.

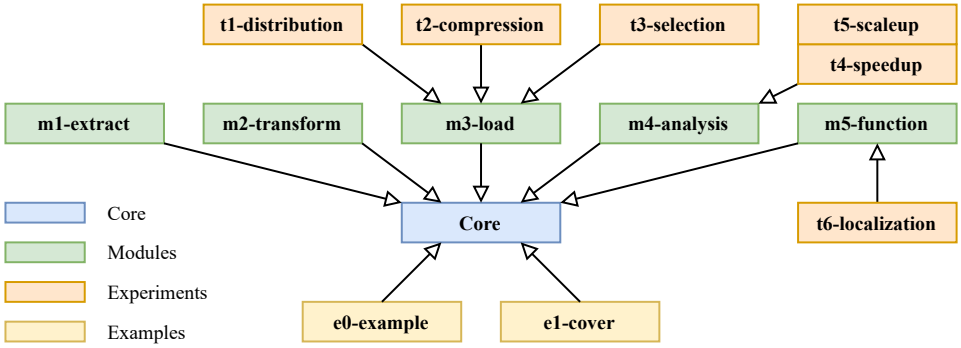


Figure C.1. Code Structure: One box represents one module while the arrows represent the dependence relationship.

The core module is highlighted in blue, which contains all the classes and functions that all the other modules rely on such as the function that is used to split the trajectory and the classes of the adaptive Octree.

The implementation modules are highlighted in green, which contains the whole life cycle of the data. The first three modules deal with the extraction, transformation and loading processes of the data. The fourth module deals with the analysis of the data spatial distribution which is used to create the adaptive Octree data structure. The final module deals with the definitions of all the SQL functions for the applications

The experiment modules are highlighted in orange, which contain the codes for experiments and visualizations shown in Chapter 6.

The example modules are highlighted in yellow, which contain the codes to generate demos for the readers to better understand the concepts and methodologies. The diagrams of these demos are also used in the Chapter 2 and Chapter 4. The last module (e1-cover) is used to generate the cover of this thesis.

D. Range Merging Problem

This appendix discusses the definition of the range merging problem from a theoretical view and gives simple examples.

The range merging problem comes from the huge number of ranges after the shape querying, which causes the deficiency in range joining especially when the dimension is high (more than 10,000 ranges would be generated). It is argued that reducing the number of ranges would have potential benefits. Reducing the number of ranges is the main constraint and then, the question becomes given a list of ranges, which two successive ranges would be the optimal merging pair. By repeatedly merging the optimal merging pair, the number of ranges can be gradually reduced.

Instead of merging the ranges based on the gaps (e.g., the Morton distances between each range), a data-aware range merging method is proposed. The ranges inside the shape are "clean" which could be directly selected without refinement. The ranges intersected or outside the shape are "dirty" which should be further refined. If a "clean" range is merged with a "dirty" range, then the merged range is also "dirty". Then, the question is transformed into an optimization problem that reduces the number of ranges without introducing many "dirty" ranges, in other words, the target is to introduce less redundant data. An example of cost calculation and two iterations of range merging is shown in Figure D.1.

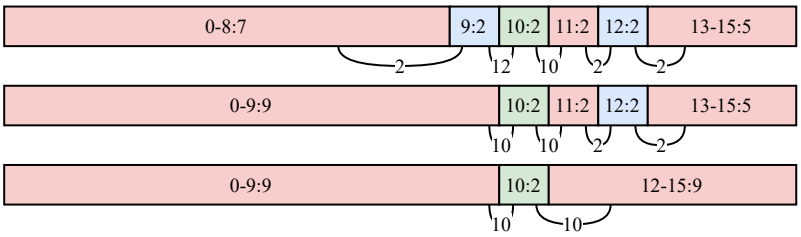
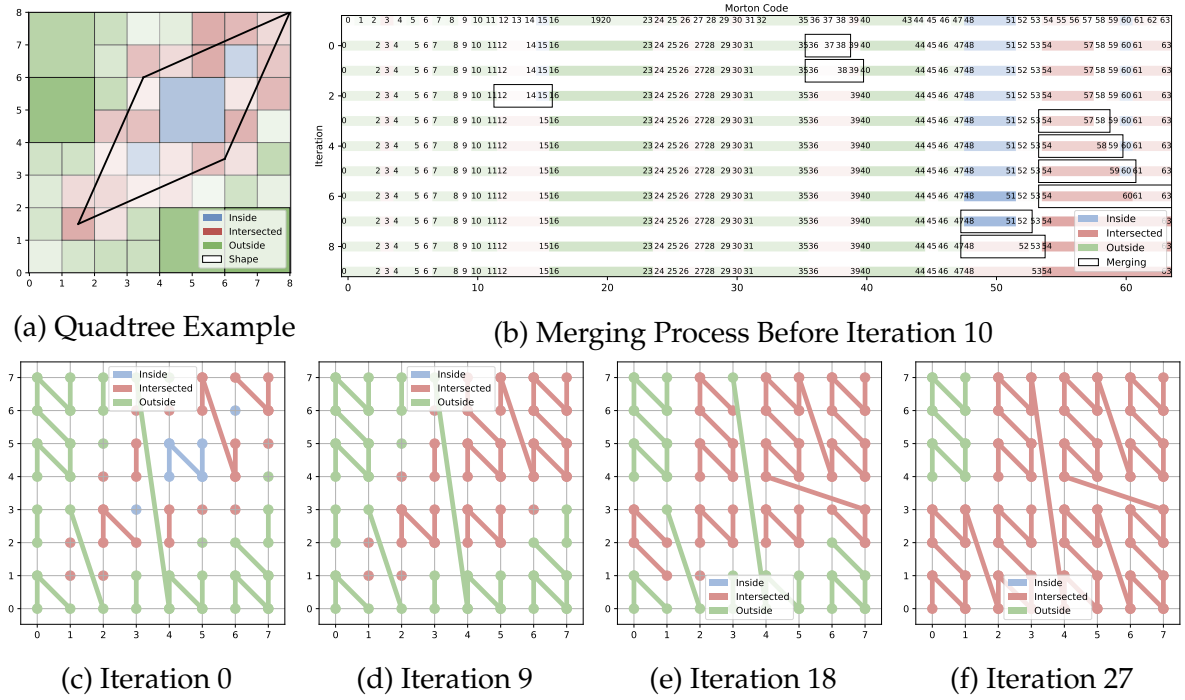


Figure D.1. Range Merging Method: All the spatial distribution information of the quadrant that the shape is fully contained is queried. The cost (redundant data size introduced) for each iteration could be calculated. Specifically, different weights could be assigned to different types of ranges, for example, the ranges that are outside should be harder (in this case 5 times harder) to be included.

A more comprehensive example is shown in the Figure D.2, the data size (from 0 to 10) is randomly generated in the 8 by 8 grid, which means the construction depth of the quadtree is 3. The search depth is also set to 3 and the node size threshold is set to 1. The weight for introducing outside ranges is set to 5 which means, that only when the sizes of the outside ranges are 5 times smaller than the inside ranges, would they be introduced.



(a) Quadtree Example

(b) Merging Process Before Iteration 10

(c) Iteration 0

(d) Iteration 9

(e) Iteration 18

(f) Iteration 27

Figure D.2. Range Merging Example: Sub-figure (a) shows the spatial distribution and the querying result while the opacity represts the size of the data. Sub-figure (b) shows the range merging process, in each iteration (y-axis), two ranges would be merged. Sub-figures (c)-(f) show the results of the range merging with different iterations, note that the number of ranges becomes smaller. It can also be seen that if there is one more iteration after sub-figure (f), all the ranges will be merged as one.

Generally speaking, reducing the number of ranges can improve query performance as the seeking time is reduced. However, performance may decrease again as more and more redundant data is introduced. At the same time, the query will also be closely related to the spatial distribution of the data. Assuming that data only exists inside the shape, even if we merge all ranges, no redundant data will be introduced, and the performance at this time is the best.

The experiment is not conducted in this thesis, first because of the nature of the taxi data, the data is modelled in low(3D) dimension without a severe sparsity problem. The second reason is due to the less control of the hardware resources, making the exploration of the relationship between the parameters and the hardware performance impossible.

However, this interesting question was deeply discussed during the thesis process, the potential directions should be mentioned here in the appendix in case of need especially when the dimensionality is high and the data distribution is skewed. Further investigations, potentially using advanced machine learning techniques for parameter optimization, are recommended to validate the effectiveness of range merging in real-world scenarios.

E. Reflection

This appendix gives reflections about the master thesis, the relevance to the Geomatics program and some personal feelings.

Reflection About Re-productivity

- Marks for each of the criteria
  1. Input data: 2
  2. Pre-processing: 3
  3. Methods: 3
  4. Computational environment: 2
  5. Results: 3
- Re-productivity self-assessment: the codes for pre-processing, methods and results (even the diagrams drawing) are open in Github. However, due to the file size limitation of the Github, only a small piece of data is provided as open. Even though the setup steps are well documented, the setup of the Greenplum database is very time-consuming, even very difficult for the ones that are not familiar with virtual machines and Linux operating systems.

Reflection About Geomatics and Thesis

After 2 years of studying in the Geomatics domain, this master thesis is also a test of how well I have learned from all the courses. This thesis is relevant to what has been taught in the Geomatics program: positioning, DBMS, computational geometry, web data (distributed systems) etc. I am happy that what I have learned is helpful in this thesis as whenever I encountered difficulties I could turn to the slides and materials provided in the courses.

What I have learned most from this thesis is to grasp the main contradiction (problem) using a systematic view as there would be many factors to consider. Guided by the underlying principles from the scientific perspective, the main question is defined and the concept is built. Then, based on the available resources and existing solutions from the engineering perspective, the alternatives are identified and the (sub)optimal solutions are proposed.

The first difficulty I encountered was the "wrong" direction around P2: I was trying to use the OLAP technique as the foundation of my research. However, after a few meetings with my supervisors and nearly two months of experiments, It turned out that OLAP may not be applicable as the "aggregation" idea only holds for numerical data but not for spatial data. In retrospect, perhaps I should have changed direction to

the distributed database after only a month without getting positive results to avoid wasting time.

Another difficulty was my weakness in reading literature as I tended to first find a large collection of papers to build a systematic view. However, this perfectionism slowed down the progress of the research, things were worse as I preferred writing after reading rather than making notes at the same time. It was a lesson that it is best to fill in the content whenever there is something to record such as reading notes, preliminary ideas, remarks etc.

There was one good thing I insisted on during the thesis progress which was I was able to prepare small presentations (in reports or slide forms) for each regular meeting to communicate with my supervisors. These materials were kept which helped in the thesis writing and reflection processes as what I have explored and what we (I and my supervisors) have discussed were quite a lot but are "remembered" during the one-year-long process.

Reflection About Personal Feelings

Recall the reflection in my undergraduate thesis, some things have become blurry in the past two years, including the understanding of education and the planning of the career. Learning Geomatics seems to be just an escape from Urban Planning and the high-intensity courses seemed to make me lose time to think about myself. The future may become more uncertain, and I may (or must) take a good break after this phase. When setting off again, there may not be much opportunity for me to prepare in advance ... Just keep moving forward!

Bibliography

Timur Abbiasov, Cate Heine, Sadegh Sabouri, Arianna Salazar-Miranda, Paolo Santi, Edward Glaeser, and Carlo Ratti. The 15-minute city quantified using human mobility data. *Nature Human Behaviour*, pages 1–11, 2024.

Tariq Alsahfi, Mousa Almotairi, and Ramez Elmasri. A survey on trajectory data warehouse. *Spatial Information Research*, 28(1):53–66, 2020.

Macro Baars. Moving objects in a geo-DBMS: Structuring, indexing, querying and visualizing moving point objects in a geo-DBMS context. Master's thesis, Delft University of Technology, 2004.

Michael Bader. *Space-filling curves: an introduction with applications in scientific computing*, volume 9. Springer Science & Business Media, 2012.

Filip Biljecki, Hugo Ledoux, and Peter Van Oosterom. Transportation mode-based segmentation and classification of movement trajectories. *International Journal of Geographical Information Science*, 27(2):385–407, 2013.

Frederick P Brooks. The mythical man-month. *Datamation*, 20(12):44–52, 1974.

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008.

Irene De Vreede. Managing Historic Automatic Identification System Data by Using a Proper Database Management System Structure. Master's thesis, Delft University of Technology, 2016.

Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Mutian Deng. Using foreign data wrapper in PostgreSQL to expose point clouds on file system. Master's thesis, Delft University of Technology, 2020.

Xin Ding, Lu Chen, Yunjun Gao, Christian S Jensen, and Hujun Bao. UlTraMan: A unified platform for big trajectory data management and analytics. *Proceedings of the VLDB Endowment*, 11(7):787–799, 2018.

Ahmed Eldawy, Louai Alarabi, and Mohamed F Mokbel. Spatial partitioning techniques in SpatialHadoop. *Proceedings of the VLDB Endowment*, 8(12):1602–1605, 2015.

Kajsa Ellegård. *Introduction: The roots and diffusion of time-geography*. Routledge, 2019.

R Elmasri, SB Navathe, R Elmasri, and SB Navathe. *Fundamentals of Database Systems*. Springer, 2015.

Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys (CSUR)*, 30(2):170–231, 1998.

Fan Gao, Peng Yue, Zhipeng Cao, Shuaifeng Zhao, Boyi Shangguan, Liangcun Jiang, Lei Hu, Zhe Fang, and Zheheng Liang. A multi-source spatio-temporal data cube for large-scale geospatial analysis. *International Journal of Geographical Information Science*, 36(9):1853–1884, 2022.

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 29–43, 2003.

Leticia Gómez, Sophie Haesevoets, Bart Kuijpers, and Alejandro A Vaisman. Spatial aggregation: Data model and implementation. *Information Systems*, 34(6):551–576, 2009.

Leticia I Gómez, Silvia A Gómez, and Alejandro A Vaisman. A generic data model and query language for spatiotemporal OLAP cube analysis. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 300–311, 2012.

Michael F Goodchild. The validity and usefulness of laws in geographic information science and geography. *Annals of the Association of American Geographers*, 94(2):300–303, 2004.

Michael F Goodchild. Commentary: General Principles and Analytical Frameworks in Geography and GIScience. *Annals of GIS*, 28(1):85–87, 2022.

Xuefeng Guan. *High-performance spatiotemporal computing and applications (In Chinese)*. China Science Publishing, 2020. ISBN 9787030669575.

Xuefeng Guan, Peter Van Oosterom, and Bo Cheng. A parallel N-dimensional space-filling curve library and its application in massive point cloud management. *ISPRS international journal of geo-information*, 7(8):327, 2018.

Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

Danny Holten and Jarke J Van Wijk. Force-directed edge bundling for graph visualization. In *Computer graphics forum*. Wiley Online Library, 2009.

Christian Jensen, Torben Bach Pedersen, and Christian Thomsen. *Multidimensional databases and data warehousing*. Morgan & Claypool, 2010.

Jonathan K Lawder. *The application of space-filling curves to the storage and retrieval of multi-dimensional data*. PhD thesis, Citeseer, 2000.

Luca Leonardi, Gerasimos Marketos, Elias Frentzos, Nikos Giatrakos, Salvatore Orlando, Nikos Pelekis, Alessandra Raffaetà, Alessandro Roncato, Claudio Silvestri, and Yannis Theodoridis. T-Warehouse: Visual OLAP analysis on trajectory data. In *2010 IEEE 26th international conference on data engineering (ICDE 2010)*, pages 1141–1144. IEEE, 2010.

Luca Leonardi, Salvatore Orlando, Alessandra Raffaetà, Alessandro Roncato, Claudio Silvestri, Gennady Andrienko, and Natalia Andrienko. A general framework for trajectory data warehousing and visual OLAP. *GeoInformatica*, 2014.

Deren Li. The intelligent processing and service of spatiotemporal big data. *Journal of Geo-Information Science*, 2019.

Jinglan Li. Manage 4D historical AIS data by Space Filling Curve. Master's thesis, Delft University of Technology, 2020.

Lin Li, Zhongliang Cai, Zhangcai Ying, and Wan You. *Principle of Spatial Database (In Chinese)*. China Surveying and Mapping Press, 2023. ISBN 9787503044724.

Ruiyuan Li, Huajun He, Rubin Wang, Yuchuan Huang, Junwen Liu, Sijie Ruan, Tianfu He, Jie Bao, and Yu Zheng. Just: Jd urban spatio-temporal data engine. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1558–1569. IEEE, 2020a.

Ruiyuan Li, Huajun He, Rubin Wang, Sijie Ruan, Tianfu He, Jie Bao, Junbo Zhang, Liang Hong, and Yu Zheng. Trajmesa: A distributed NoSQL-based trajectory data management system. *IEEE Transactions on Knowledge and Data Engineering*, 35(1): 1013–1027, 2021.

Wenwen Li, Michael Batty, and Michael F Goodchild. Real-time GIS for smart cities, 2020b.

H Liu, P Van Oosterom, M Meijers, and E Verbree. Management of large indoor point clouds: An initial exploration. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42:365–372, 2018.

H Liu, P Van Oosterom, M Meijers, and E Verbree. An optimized SFC approach for nD window querying on point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 6(4/W1):119–128, 2020a.

Haicheng Liu. *nD-PointCloud Data Management: continuous levels, adaptive histograms, and diverse query geometries*. PhD thesis, Delft University of Technology, 2022.

Haicheng Liu, Peter van Oosterom, Martijn Meijers, Xuefeng Guan, Edward Verbree, and Mike Horhammer. HistSFC: Optimization for nD massive spatial points querying. *International Journal of Database Management Systems (IJDMS)*, 12(3):7–28, 2020b.

Haicheng Liu, Rodney Thompson, Peter van Oosterom, and Martijn Meijers. Executing convex polytope queries on nD point clouds. *International Journal of Applied Earth Observation and Geoinformation*, 105:102625, 2021a.

Haicheng Liu, P Van Oosterom, B Mao, M Meijers, and R Thompson. An efficient nd-point data structure for querying flood risk. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences-ISPRS Archives*, 43(B4-2021):367–374, 2021b.

Ling Liu and M Tamer Özsu. *Encyclopedia of database systems*, volume 6. Springer New York, 2009.

Ahmed R Mahmood, Sri Punni, and Walid G Aref. Spatio-temporal access methods: a survey (2010-2017). *GeoInformatica*, 23:1–36, 2019.

Martijn Meijers. PCSERVE-ND-Pointclouds Retrieval over the Web. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 10(4/W2-2022): 193–200, 2022.

Martijn Meijers and Peter van Oosterom. Clustering and Indexing Historic AIS Data With Space Filling Curves. Technical report, Delft University of Technology, 2018.

Martijn Meijers, Peter van Oosterom, and Wilko Quak. Management of AIS messages in a Geo-DBMS. Technical report, Delft University of Technology, 2016.

Mohamed Mokbel, Mahmoud Sakr, Li Xiong, Andreas Züfle, Jussara Almeida, Walid Aref, Gennady Andrienko, Natalia Andrienko, Yang Cao, Sanjay Chawla, et al. Towards mobility data science (vision paper). *arXiv preprint arXiv:2307.05717*, 2023.

Salvatore Orlando, Renzo Orsini, Alessandra Raffaetà, Alessandro Roncato, and Claudio Silvestri. Spatio-temporal aggregations in trajectory data warehouses. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 66–77. Springer, 2007.

M Tamer Özsu, Patrick Valduriez, et al. *Principles of distributed database systems*, volume 2. Springer, 2020.

Nikos Pelekis, Elias Frentzos, Nikos Giatrakos, and Yannis Theodoridis. HERMES: A trajectory DB engine for mobility-centric applications. *International Journal of Knowledge-Based Organizations (IJKBO)*, 5(2):19–41, 2015.

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

Marshall Presser. *Data Warehousing with Greenplum: Open Source Massively Parallel Data Analytics*. O'Reilly Media, 2017.

Damião Ribeiro de Almeida, Cláudio de Souza Baptista, Fabio Gomes de Andrade, and Amilcar Soares. A survey on big data for trajectory analytics. *ISPRS International Journal of Geo-Information*, 9(2):88, 2020.

John F Roddick, Erik Hoel, Max J Egenhofer, Dimitris Papadias, and Betty Salzberg. Spatial, temporal and spatio-temporal databases-hot issues and directions for PhD research. *ACM SIGMOD Record*, 33(2):126–131, 2004.

Shashi Shekhar and Hui Xiong. *Encyclopedia of GIS*. Springer Science & Business Media, 2007.

Abraham Silberschatz, Henry F Korth, and Shashank Sudarshan. *Database system concepts*. McGraw-Hill Education, 2019.

Waldo Tobler. On the first law of geography: A reply. *Annals of the association of American geographers*, 94(2):304–310, 2004.

Alejandro Vaisman and Esteban Zimányi. What is spatio-temporal data warehousing? In *International Conference on Data Warehousing and Knowledge Discovery*, pages 9–23. Springer, 2009.

Peter van Oosterom. Spatial access methods. *Geographical information systems*, 1:385–400, 1999.

Peter van Oosterom and Tom Vijlbrief. The spatial location code. In *Proceedings of the 7th international symposium on spatial data handling, Delft, The Netherlands*, pages 12–16, 1996.

Peter van Oosterom, Oscar Martinez-Rubi, Theo Tijssen, and Romulo Gonçalves. Realistic benchmarks for point cloud data management systems. *Advances in 3D Geoinformation*, pages 1–30, 2017.

Chunbo Wang. *Efficiently use Greenplum, entry, advanced and data centre (In Chinese)*. China Machine Press, 2022. ISBN 9787111696490.

Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, and Xiaolin Qin. Fast large-scale trajectory clustering. *Proceedings of the VLDB Endowment*, 13(1):29–42, 2019.

Jianqiu Xu, Ralf Hartmut Güting, and Yunjun Gao. Continuous k nearest neighbour queries over large multi-attribute trajectories: a systematic approach. *GeoInformatica*, 22(4):723–766, 2018.

Penglin Zhang, Changhui Yu, and Weiqing Li. *Spatiotemporal database principles and technologies (In Chinese)*. Wuhan University Press, 2019. ISBN 9787307210691.

Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):1–41, 2015.

Esteban Zimányi, Mahmoud Sakr, and Arthur Lesuisse. MobilityDB: A mobility database based on PostgreSQL and PostGIS. *ACM Transactions on Database Systems (TODS)*, 45(4):1–42, 2020.