Beyond Click-and-View: a Comparative Study of Data Management Approaches for Interactive Visualization

Lorenna Christ'na Nascimento[1], Rodolfo P. Chagas[1], Marcos Lage[1], Daniel de Oliveira[1]

Institute of Computing, Universidade Federal Fluminense, Niterói - RJ, Brazil
{lorennachrist,rodolfochagas}@id.uff.br, {marcos,danielcmo}@ic.uff.br

Abstract. Visual analytics solutions have been growing in popularity in recent years, not only for showing final results but also for assisting in interactive analysis and decision-making. Analysis of a large amount of data requires flexible exploration and visualizations. However, queries that span geographical regions over time slices are expensive to compute, which turns it challenging to accomplish interactive speeds for huge data sets. Such systems require efficient data availability, so that response time does not interfere with the user's ability to observe and analyze. Simultaneously, researches in the database domain have proposed solutions that can be used to support visualization systems. This article presents a comparative study of data management approaches to support interactive visualizations. The chosen data management solutions are (i) Apache Drill (a Polystore system), (ii) Apache Spark (a big data framework), (iii) Elasticsearch (a search engine), (iv) MonetDB (a column-oriented DBMS), and (v) PostgreSQL (a relational DBMS). To evaluate the performance of each solution, we selected a list of spatiotemporal queries among multiple queries submitted by users in a visual analytics system for rainfall data analysis named `TEMPO`. The results of this study show that Apache Spark and MonetDB present the best performance for the selected queries.

Categories and Subject Descriptors: H.2.5 [**Database Management**]: Heterogeneous Databases; H.2.0 [**Database Management**]: General

Keywords: Interactive Vis, Database Evaluation

1. INTRODUCTION

With the emergency and popularity of the Data Science domain over the last years, the use of visualization (henceforth named only as Vis) techniques has gained a lot of attention [Schmidt 2020]. Although data science involves multiple disciplines such as Data Management [Ponchateau 2018], Optimization [Ribeiro 2012], Machine Learning [Bilokon 2022; Ghojogh 2021], and domain-specific areas [Ocaña et al. 2015; de Oliveira et al. 2012; Blanco et al. 2020; de Oliveira et al. 2013], the Vis area plays a key role as it supports understanding and analyzing input data, query results, and machine learning models. In many cases, this process of visual exploration and analysis requires the use of interactive Vis [Lins et al. 2013; Munzner 2014a], *i.e.*, a type of Vis that is capable of reacting "immediately" to a given action performed by the user in the system (*e.g.*, a temporal filter or the selection of an area in space). More precisely, to implement interactive Vis approaches, data must be accessed, filtered, and aggregated in an efficient way to produce a new visual representation with as little latency as possible, ideally below 0.5 seconds [Liu and Heer 2014a].

In this context, one of the challenges of building interactive Vis is that filters and aggregations commonly involve developing programs or queries over space and time. The spatial data involved in these operations consist of collections of points, lines, regions, rectangles, *etc.* [Samet 1990b; 1990a], and can be used to represent more complex structures such as neighborhoods and cities. Thus, in order to perform such queries in an efficient way, many Vis systems use specialized approaches such as Nano Cubes [Lins et al. 2013], and DBMS extensions such as PostGIS [Strobl 2017] to represent and query spatiotemporal data. These approaches have already shown their effectiveness to reduce latency in queries, but they present some disadvantages. In the case of Nano Cubes, data in their finest grain may be unavailable and there may be an exponential storage cost in relation to the number of attributes. On the other hand, many users detect performance issues in PostGIS when working with geospatial data at a large scale [Sevim et al. 2021]. PostGIS is not designed to support interactive Vis mechanisms that consume volumous and detailed geospatial data, and the queries take too long to return results, which is not desired. Along with such approaches, approximate query processing techniques can also be applied to reduce the time-to-query [Zimbrao and de Souza 1998].

Although the aforementioned data management solutions (*i.e.*, PostGIS and Nano Cubes) have been successfully used in many Vis projects, the Database community has been developing a series of domain/problem agnostic solutions for data management in recent years that can be applied in the Vis context, especially in interactive Vis. Such solutions range from NoSQL DBMSs (*e.g.*, MonetDB [Boncz et al. 2006], MongoDB [Makris et al. 2021]) to Polystore systems [Kranas et al. 2021] and large-scale data analysis frameworks such as Apache Spark (with Spark SQL) [Zaharia et al. 2016], in addition to well-known RDBMSs like PostgreSQL. Although the aforementioned solutions are not designed to support interactive Vis systems, they have been shown to be increasingly efficient and can be used in many contexts, replacing domain-specific solutions such as Nano Cubes. However, it is far from trivial to identify the suitable solution for a specific interactive Vis application since the data management approaches present different data models, in-memory processing mechanisms, *etc.*

In this article, we present a comparative study of several data management approaches to support interactive Vis queries. The idea is to analyze in which scenarios a specific solution is more recommended for a Vis system. In addition, we focus on off-the-shelf data management approaches, *i.e.*, domain-specific data management approaches are not evaluated. As benchmark, we chose a series of queries to support interactive Vis submitted by users to a rainfall analysis tool called `TEMPO` (sisTema dE Monitoramento PluviométricO in Portuguese, or Rainfall Monitoring System in English) [Nascimento et al. 2021], during the interactive Vis sessions. `TEMPO` allows for the user to submit spatio-temporal queries to analyze the behavior of rainfall in the city of Niterói. The system can access rainfall data measured at multiple rainfall stations over 13 years (more details in Section 4.1). It is worth mentioning that although the results presented in this article are based on data and queries provided by `TEMPO`, the evaluated data management solutions were not designed for this tool specifically.

This article is an extension of a conference paper published in the Proceedings of the 2021 Brazilian Symposium on Databases (SBBD) [Nascimento et al. 2021]. In this extended version, we enriched the comparative study with more data management approaches (*e.g.*, ElasticSearch [Kononenko et al. 2014]) and added eight more queries to the benchmark. We have also improved the background and the related work. The remainder of this article is structured in five sections besides this introduction. Section 2 presents the interactive Vis and Spatio-Temporal query patterns considered in this article. Section 3 discusses related work. Section 3 discusses the case study. Section 5 explains the technologies used in the comparative study. Section 6 presents the comparative study and discusses the results. Finally, Section 7 concludes this article and points out future work.

2. SPATIO-TEMPORAL QUERY PATTERNS FOR INTERACTIVE VIS

Visual data exploration and analysis systems have been successfully used to study ill-posed problems (*i.e.*, problems for which one cannot find solutions using purely computational techniques) or to investigate issues that require judgment and/or expertise of a domain expert [Munzner 2014b]. There are Vis systems successfully used in several areas such as healthcare [Caban and Gotz 2015; Verma 2022], safety [Hochstetler et al. 2016], mobility [Wang et al. 2021], education [Ciesielska et al. 2021], and urban management [Zheng et al. 2016]. One of the fundamental requirements of these systems is that they are able to achieve interactive response times. Liu and Heer [2014b] state that latencies greater than 0.5 seconds (500ms) already interfere with the ability of the user to observe, generalize and construct hypotheses during the visual exploration of data. Therefore, one of the common requirements for all Interactive Vis systems is to achieve a *time-to-query* (the time elapsed from the time the user submits the query to the time he receives the response) of less than 0.5 seconds whenever possible. The big challenge is that such queries in Vis systems commonly involve multiple joins (including spatial joins), aggregations, and searches in space and time, which makes queries complex.

Doraiswamy and Freire [2020] describe a set of common spatiotemporal analytical queries that can be used to define a series of patterns. Their analysis is built on top of the classification proposed by Eldawy and Mokbel [2017]. In this article, the classification is inspired in the work of Doraiswamy and Freire [2020], and the query patterns are categorized into four classes: (C1) *Selection*, (C2) *Join*, (C3) *Aggregate*, and (C4) *Nearest Neighbor*. Following we detail each pattern. It is worth noticing that Doraiswamy and Freire [2020] consider only points and polygonal datasets, which is the same scenario found in the case study presented in Section 6. *Selection* class queries are those in which points and polygons are selected according to a specific selection predicate. This type of query can also return a time series based on a spatial selection (*e.g.*, return the rainfall time series in a period of time of selected rainfall stations) and is presented in Fig. 1(b). Let us consider D_p a set of data points (in the context of this article, a set of rainfall stations in a city). Let us also consider $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ the coordinates of each data point (crosses in Fig. 1(a)). A *Selection* query could be expressed as the following SQL-like syntax: `SELECT * FROM` D_P `WHERE` A_i θ `value;` where A_i is an attribute of each data point, $\theta \in \{=, <, \leq, >, \geq, \neq\}$, *value* $\in D_i$, and D_i is the domain A_i. *Join* queries are those in which joins are needed, whether between polygons or between polygons and points, *e.g.*, to find out which rainfall stations (points) are at a given area on the map (polygon). This type of query is presented in Fig. 1 (c). Let us consider D_p a set of data points and D_Y as a polygon dataset. A *Join* query could be expressed as the following SQL-like syntax: `SELECT * FROM` D_P`,`D_Y `WHERE` D_P`.Location INSIDE` D_Y`.Geometry;`

Aggregate queries (Fig. 1(d)) are the ones that commonly require a "group by" over a *Selection* or a *Join*, *e.g.*, to discover the average rainfall over the last two months of a given area of the map. An *Aggregate* over a *Selection* can be represented as the following SQL-like syntax: `SELECT` $aggreg_f$ `FROM` D_P `WHERE` A_i θ `value;` where $aggreg_f \in$ `{COUNT,MIN,MAX,AVG,SUM}`, A_i is an atribute of each data point, $\theta \in \{=, <, \leq, >, \geq, \neq\}$, *value* $\in D_i$, and D_i is the domain A_i. An *Aggregate* over a *Join* can be represented as the following SQL-like syntax: `SELECT` $aggreg_f$ `FROM` D_P`,`D_Y `WHERE` D_P`.Location INSIDE` D_Y`.Geometry;` where $aggreg_f \in$ `{COUNT,MIN,MAX,AVG,SUM}`, D_p a set of data points and D_Y as a polygon dataset. Finally, *Nearest Neighbor* queries returns a set of the k points in space closer to another specified point according to some criterion, *e.g.*, given a specific coordinate, find the k closest rainfall stations. An *Nearest Neighbor* query can be represented as the following SQL-like syntax: `SELECT * FROM` D_P `WHERE` D_P`.Location` $\in$ `KNN(X,K);` where X is the query point and K defines the number of neighbors that are returned.

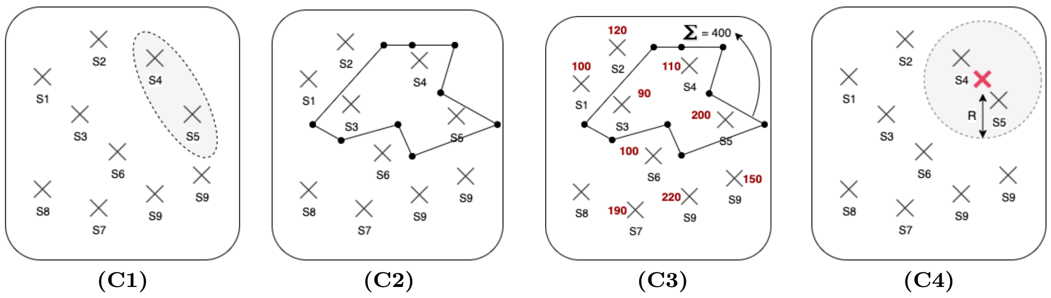

Fig. 1: **(C1)** Selection Query **(C2)** Join Query **(C3)** Aggregate Query **(C4)** K Nearest Neighbor Query

3. RELATED WORK

Comparing the performance of DBMSs and data management solutions with specific goals is a well-known task in the database community. In fact, there are many benchmarks that can be used to evaluate existing and new DBMSs, *e.g.*, TPC [Poess and Nambiar 2019] (which is specialized in TPC-C for OLTP applications, TPC-H [Poess 2019b] for OLAP applications, TPCx-HS [Magdon-Ismail 2019] for Big Data applications, TPC-DS [Poess 2019a] for decision support applications, TPC-E for OLTP applications, TPC-DI for Data Integration, TPCx-V for virtualized databases) SPEC [Bays and Lange 2012], *etc.* However, benchmarks such as SPEC and TPC are not designed for evaluating data management solutions for interactive Vis as they do not consider database queries generated through user interactions. Recently, some efforts are being made to generate benchmarks for interactive Vis [Battle et al. 2020]. Although the benchmark proposed by Battle *et al.* [2020] represents a step forward, they focus on comparing relational databases. The work of Makris *et al.* [2021] also compares two different DBMSs for spatiotemporal applications, but they considered only two types of DBMSs (relational and document-oriented) and does not consider queries generated through user interactions. Eichmann *et al.* [2020] propose a benchmark called IDEBench to evaluate interactive Vis systems. The authors propose a series of synthetic queries to be executed on the column-oriented DBMS MonetDB.

Other approaches focus on providing structures for performing queries for interactive Vis in an efficient way. Many of these structures are specialized for supporting this type of Vis, as Nano Cubes [Lins et al. 2013]. Nano Cubes are in-memory structures that can be used to generate known visualizations such as heat maps, histograms, and parallel coordinate plots. The advantage of Nano Cubes compared to traditional Data Cubes is that they can store up to billions of data points without using disk, while Data Cubes [Battle et al. 2016] can manage a much smaller dataset without using the disk. However, such structures must be reconstructed for each collection of attributes and aggregation types required by the application (*e.g.*, SUM, AVG, *etc.*), which can be laborious. Furthermore, as the data are already pre-aggregated in memory, any query that requires access to the finer grain data can not be performed. Finally, the size of the structure grows exponentially according to the number of attributes considered. Differently from Lins *et al.* [2013] and Battle *et al.* [2016] which opted for specialized in-memory structures to support interactive Vis systems, the work of Jiang *et al.* [2018] proposes that such systems use RDBMS as a data management approach. In the Jiang *et al.* paper, the authors analyze the response time of spatiotemporal queries in PostgreSQL RDBMS. In the current article, in turn, we compare the performance of five data management approaches to support interactive Vis. Each data management approach represents one type of data management (*e.g.*, relational, columnar, *etc.*) in order to analyze the pros and cons of each type. This comparison with five different data management solutions is not found in previous works.

4. BENCHMARK: QUERIES TO SUPPORT INTERACTIVE VIS

In order to evaluate the several data management solutions for interactive Vis, we selected as benchmark several queries submitted during an interactive Vis session. These queries are submitted to data management approaches by the tool named `TEMPO` [Nascimento et al. 2021], whose goal is to capture multi-source rainfall data with different granularities and integrate them into a single database to allow for interactive visual analysis. `TEMPO` was designed and developed in the context of the project *Niterói Organizada e Segura: o Estudo do Impacto das Chuvas*", carried out in partnership with Civil Defense Bureau of the City of Niterói in the context of the Applied Projects Development Program (PDPA). Thus, `TEMPO` provides the rainfall data and queries that are used in the analysis to compare the performance of each approach selected in this article. Following, we provide more details regarding `TEMPO` and the interactive Vis queries.

Studies related to climate have become increasingly important in the last decade, mainly due to the increase in the number of extreme climate events (*e.g.*, tsunamis, floods, droughts, *etc.*) in multiple regions of the Earth, including Brazil [Mizutori and Guha-Sapir 2020]. In particular, in large urban centers, the impact of climate events can be severe. Due to the unordered growth, urban centers do not have many areas covered by vegetation, which makes the flow of rainwater go quickly to places such as rivers, which may not have the adequate flow capacity for the volume of water received, thus causing overflow and floods [Thorndahl and Willems 2008]. This way, monitoring the rainfall (that can cause such floods) is a top priority.

To allow for such monitoring, a set of rainfall stations (sensor-based data acquisition systems that provide the rainfall value in a given area) are installed in many cities throughout the country. TEMPO accesses data provided by such stations to provide interactive analysis features. Besides gathering and storing the raw data (*e.g.*, CSV, JSON, *etc.*), TEMPO performs the necessary data transformations (*e.g.*, granularity adjustment, addresses and location standardization, aggregations, *etc*) and generates a single staging table with the formatted data (in a staging area). From this table, the data are loaded into a Data Warehouse (DW) [Kimball and Ross 2002] named DW-TEMPO. In its original version, the DW-TEMPO was implemented following the ROLAP model (Relational OLAP) using PostgreSQL RDBMS.

The schema of DW-TEMPO is presented in Fig. 2. This schema follows the star schema and has one fact table and four dimension tables. The fact table *Fact_Rainfall* contains the attributes *rainfall_index* (quantity of interest for analysis), *idDim_Source* (PK,FK), *idDim_Time* (PK,FK), *idDim_Locality* (PK,FK) and *idDim_Station* (PK,FK), where all foreign keys reference the associated dimension tables. The *Dim_Locality* dimension represents the locations where the rainfall stations are located (where measurements were taken), *e.g.*, cities, specific addresses, *etc.* The dimension *Dim_Source* represents the data provenance [Freire et al. 2008], *i.e.*, from which source these data were obtained. The *Dim_Time* dimension represents the possible time intervals for querying. Finally, the dimension *Dim_Station* represents the rainfall stations that were considered for analysis. Due to space constraints we do not present the complete schema of DW-TEMPO, but additional information can be obtained from Nascimento *et al.* [2021]. The dump file containing the data used in this article can be obtained at `https://osf.io/pa2zn/`.

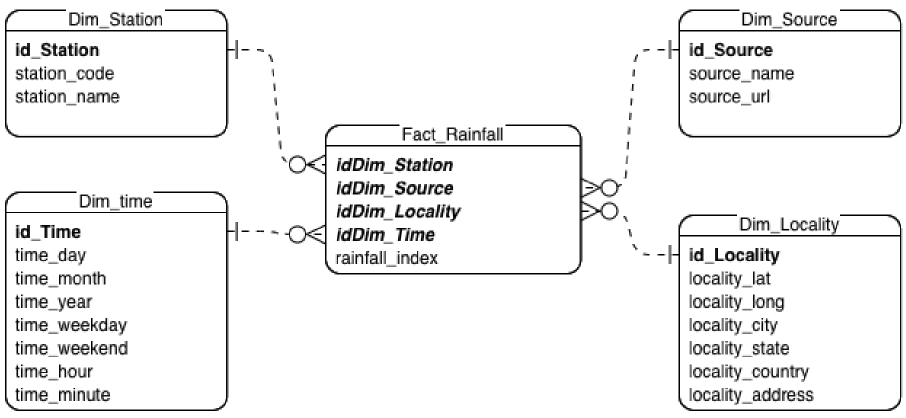

Fig. 2: The Schema of DW TEMPO - adapted from [Nascimento et al. 2021]

Following, some statistics of the DW are presented. Table I presents the total number of tuples per DW table. The DW contains rainfall data collected from 2013 to 2019. The time resolution depends on the source of data. The data downloaded from CEMADEN is captured every hour (when it is not raining) and every 10 minutes (if it is raining) while in Alerta Rio data are captured every

15 minutes. Since TEMPO gathers data from CEMADEN (which is a federal organization), the DW stores rainfall data from several Brazilian cities. However, the visual analyses performed by users in TEMPO are focused on the city of Niterói. Fig. 3 presents the number of tuples in the fact table per rainfall station in the city of Niterói (×1,000). By collecting metadata from user interactions in TEMPO, we identified a series of frequent database queries. In Table II we present 14 frequent queries that are used in the comparative study presented in this article. Each query presented in Table II is classified following the classes defined and discussed in Section 2. It is worth noticing that although there are many other queries submitted during users' analyses, we selected these 14 because they are heterogeneous and cover most of the aforementioned classes.

Table I: Total Number of Tuples per Table

Table	Number of Tuples
Fact_Rainfall	18,291,812
Dim_Time	201,576
Dim_Locality	675
Dim_Station	472
Dim_Source	2

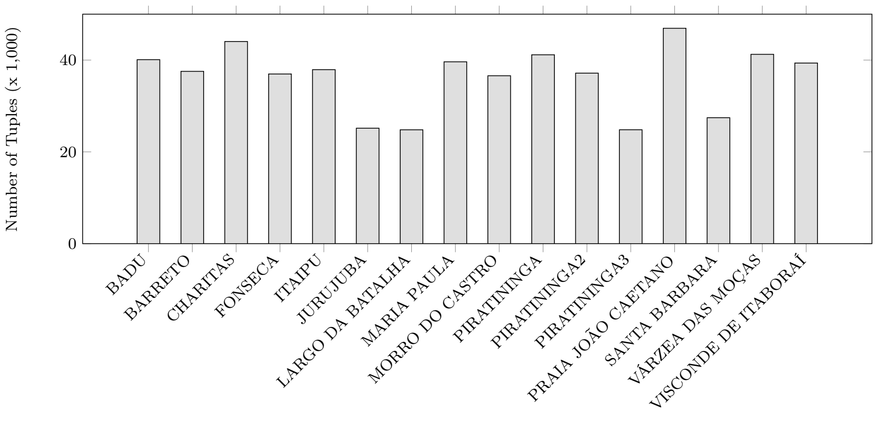

Fig. 3: Number of Tuples per Rainfall Station

Queries Q1, Q2, and Q3 are C1-type queries. Q1 aims at exploring *Selection* based on a date. Queries Q2 and Q3 explore *Selection* based on the rainfall index. The difference between queries Q2 and Q3 is their selectivity. Q2 returns fewer rows than Q3, thereby it requires less CPU and I/O. Queries Q4, Q5, and Q6 are C4-type queries. Q4 returns data from all stations within a 10 km radius of a specific point in space. Q5 and Q6 also return data from all stations within a 10 km radius of a point in space, but they filter data based on the rainfall index and date, respectively. Query Q7 also returns data from all stations within a 10 km radius, but filters based on a time interval. Q8 selects the most recent rainfall index stored for the nearest station of a specific point in space. Q9 and Q10 are also nearest neighbor queries, and the difference between these queries is their K value. While Q9 returns data associated with the 10 nearest stations, Q10 returns data associated with 20 nearest stations. Q11 combines the characteristics of a *Nearest Neighbor* query and a *Aggregation* query since it considers the 10 nearest stations but also calculates the average rainfall index. Q12, Q13, and Q14 are *Join* queries. In these three queries, we consider the Inga region in the city of Niterói (Fig.4). Q12 returns all rainfall time series of rainfall stations within the polygon. Q13 and Q14 also return all rainfall time series of rainfall stations within the polygon but they filter data based on a date and rainfall index, respectively. All the 14 queries were executed in each data management approach that is explained in the next section.

Table II: Database queries generated through user interactions in TEMPO

Query	Description	Class	# Tuples
Q1	Returns the complete rainfall time series for all the rainfall stations in the year of 2018	C1	2.211.196
Q2	Returns the complete rainfall time series for all rainfall stations where the rainfall index is higher than 25.00 (*i.e.*, outliers)	C1	5.457
Q3	Returns the complete rainfall time series where the rainfall index is higher than 1.50	C1	281.924
Q4	Returns the rainfall time series for the nearest stations within a 10 km radius of a point in space	C4	688.653
Q5	Returns the rainfall time series for the nearest stations within a 10 km radius of a point in space where the rainfall index is higher than 5.00	C1,C4	2.646
Q6	Returns the rainfall time series for the nearest stations within a 10 km radius of a point in space in 04/08/2017	C1,C4	408
Q7	Returns the time series of precipitation for the nearest stations within a 10 km radius of a point in space where date is between 11/15/2018 and 11/30/2018	C1, C4	5.105
Q8	Returns the most recent stored rainfall index for the nearest rainfall station of a specific point in space	C1, C3, C4	1
Q9	Returns the rainfall index for the 10 nearest stations from a specific point in space	C1, C4	10
Q10	Returns the rainfall index for the 20 nearest stations from a specific point in space	C1, C4	20
Q11	Returns the average of the rainfall index for the 10 nearest stations from a specific point in space	C3, C4	10
Q12	Returns the rainfall time series for all rainfall stations inside the polygon associated to the Ingá region in Niterói	C1, C2	46.928
Q13	Returns the rainfall time series for the stations which are inside the polygon associated to the Ingá region in Niterói and the date is between 01/01/2019 and 06/30/2019	C1, C2	4.303
Q14	Returns the rainfall time series for all stations which are inside the polygon associated to the Ingá region in Niterói where the rainfall index is equal or higher than 2.00	C1, C2	536

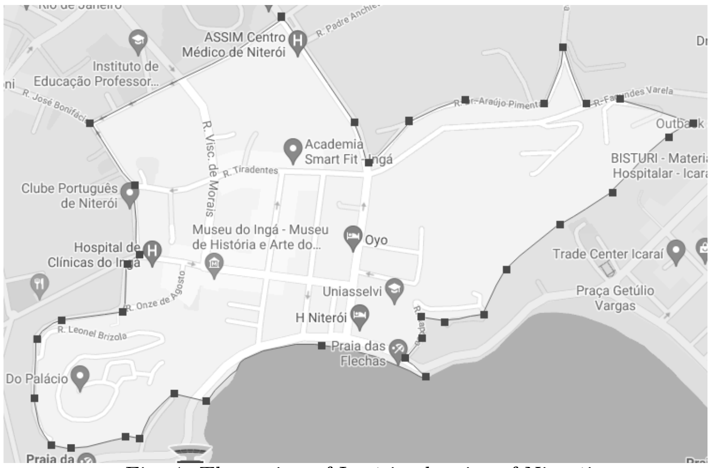
ASSIM Centro Médico de Niterói
Instituto de Educação Professor...
R. José Bonifácio
Academia Smart Fit - Ingá
R. Tiradentes
R. Visc. de Morais
Clube Português de Niterói
Museu do Ingá - Museu de História e Arte do...
Oyo
Hospital de Clínicas do Ingá
BISTURI - Materi Hospitalar - Icara
Trade Center Icaraí
Praça Getúlio Vargas
Uniasselvi
H Niterói
R. Onze de Agosto
R. Leonel Brizola
Praia das Flechas
Do Palácio
Outback
R. Fagundes Varela

Fig. 4: The region of Ingá in the city of Niterói

5. TECHNOLOGY OVERVIEW

In order to evaluate the performance of each query presented in Table II, five different data management approaches are chosen, each one with unique characteristics. The idea is to explore the heterogeneity of the solutions regarding the adopted data model, query processing mechanisms, in-memory processing, *etc.* This way, we have defined five categories of data management approaches that can support interactive Vis queries: (i) Relational databases, (ii) Column-oriented databases, (iii) Big Data frameworks, (iv) Polystore systems and (v) Search engines. For each category we have chosen a system that is representative following the classification provided by Olivera *et al.* [2021].

Following we present the list of selected approaches with a brief explanation about them. We have chosen (i) a relational DBMS - PostgreSQL, (ii) a column-oriented DBMS - MonetDB, (iii) a big data framework - Apache Spark, (iv) a polystore system - Apache Drill and (v) a search engine - Elasticsearch.

Apache Drill is an open-source distributed system for big data querying. Apache Drill is designed to handle up to petabytes of data spread across a large number of servers [Hausenblas and Nadeau 2013]. Some of the key features of Apache Drill are (i) Low latency queries, which means that a simple query returns the result in a few milliseconds; (ii) Ability to access multiple data sources in a single query, such as Hive tables, JSON files and file systems (local or distributed); and (iii) it works with Business Intelligence tools, which allows for direct integration with specialized visualization tools [Drill 2022]. Apache Drill is considered a Polystore system [Kranas et al. 2021] since it can query multiple DBMSs that follow multiple data models.

Apache Spark is a big data framework that can execute tasks on very large datasets and is also able to distribute data processing tasks across multiple computers. Spark improves the performance of applications by performing in-memory data movement (in contrast with Apache Hadoop where operations are disk to disk) and automatically exploiting parallelism. One of the key advantages of Spark in comparison to other big data frameworks is its in-memory structures such as *Resilient Distributed Dataset* (*RDD*) [Zaharia et al. 2012] and *DataFrames* [Armbrust et al. 2016], which essentially are in-memory collections of partitioned data instances that can be processed in parallel. While RDDs are sets of objects representing data, DataFrames are distributed collections of data with named columns, *i.e.*, DataFrames act as tables in relational databases (*e.g.*, PostgreSQL, Oracle, *etc.*). It has an optimization technique called Adaptive Query Execution (AQE), which is based on runtime statistics to estimate the most effective query execution strategy [Spark 2021a]. Spark can read different data formats such as JSON, CSV and ORC files, Hive tables, *etc.* [Spark 2021b].

Elasticsearch is part of the Elastic Stack, or ELK stack. This stack is composed of Elasticsearch, Logstash, and Kibana tools. Elasticsearch is responsible for indexing, searching, and analyzing datasets. Logstash is responsible for collecting and aggregating data to be indexed by Elasticsearch. Finally, Kibana offers ways to perform visualization and exploration of the data [Elastic 2021b]. Elasticsearch supports different types of data and provides an index for each type. These indexes are optimized to ensure efficient and fast searches. An index is a collection of documents in JSON format and a document is a collection of fields represented as key-value pairs. Each field in the document is indexed using a data structure optimized for its specific data type, and this makes turn the queries so efficient [Elastic 2021a]. Elasticsearch supports structured queries, such as those performed using SQL, and full-text queries, in which a search is performed using a string (a set of letters and other characters) in all documents and the result is returned by relevance [Elastic 2021c].

MonetDB is an open-source column-oriented DBMS largely used for workloads that involve a wide collection of data in different industries, such as health care, telecommunications, and others [Idreos et al. 2012]. Initially, it was designed for data warehouse applications, which are widely used to provide decision-making support [Idreos et al. 2012]. MonetDB is built to support distributed processing aiming at reducing the response time for complex queries [MonetDB 2021]. It also supports SQL and provides a modern and scalable solution without requiring considerable hardware investments [MonetDB 2021]. In terms of indexes, although the user can define an index using SQL commands, MonetDB does not create a physical secondary index as defined in the SQL command. It decides which column search accelerator to create, persist and use during SQL query execution. MonetDB supports two types of indexes: Imprints and Ordered Index. Such indexes can be associated with a single column and only numerical attributes can be indexed.

PostgreSQL is a well-known open-source DBMS that follows the relational model and uses SQL combined with features that store and scale complex data workloads [PostgreSQL 2022b]. There are many extensions available to use in PostgreSQL, including PostGIS, which is a spatial database extension for PostgreSQL. PostGIS adds support for geographic objects and enables functions that can be easily interpreted and performed by SQL [PostGIS 2021]. PostgreSQL has proved to be scalable in terms of both the amount of data it can handle and the number of concurrent users [PostgreSQL 2022b]. The index types provided by PostgreSQL are B-tree, hash, GiST, SP-GiST, GIN, and BRIN [PostgreSQL 2022a]. Although PostgreSQL has the GiST index, which is useful in indexing geometric data types [PostgreSQL 2022a], the one used in this work is B-tree, since the other data management approaches do not have indexes similar to GiST.

6. EVALUATING DATA MANAGEMENT SOLUTIONS FOR INTERACTIVE VIS

This section presents the configurations used to execute the experiments and discusses the results achieved. The central idea of this section is to compare the performance of each data management approach for each of the 14 queries presented in Section 3. First, in this section, we briefly describe the environment setup. Then we describe the experiment setup. After that, we present and discuss the results.

6.1 Environment Setup

For the experiments executed in the context of this article, we have deployed each data management approach (*i.e.*, PostgreSQL, MonetDB, Spark, Drill and Elasticsearch) on top of the Amazon AWS environment. Amazon AWS is one of the most popular cloud providers, and many scientific and commercial applications have been deployed on it. Amazon AWS provides several different types of Virtual Machines (VMs) for scientists to deploy and use. Each one of them has unique characteristics (CPU, RAM, and storage). There are several types of VMs, such as a1.medium (1 vCPU, 2 GiB RAM), c6g.2xlarge (8 vCPU, 16 GiB RAM), r4.2xlarge (8 vCPUs, 64 GiB RAM), and d2.4xlarge (16 vCPUs, 122 GiB RAM). In the experiments presented in this article, we have considered just Amazon's m5dn.2xlarge type. We have chosen an M5-type VM since these instances support a balance of computing, memory, and networking resources for different types of workloads. This includes medium and large databases. m5dn.2xlarge has 8 vCPUs, 16 GiB RAM, 110 GB disk, and is powered with Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz (2), and CentOS Linux 7 (Core) Operating System. According to Amazon, all VMs were instantiated in the US East—N. Virginia location and follow the pricing rules of that locality.

6.2 Experiment Setup

The experiments were performed in the five data management approaches mentioned in the Section 5 using the following versions: Apache Drill version 1.19.0, Apache Spark version 3.2.0, Elasticsearch version 7.15.1, MonetDB version 11.41.11, and PostgreSQL version 9.2.24. Especially in Apache Spark, we transformed the DataFrame into a Resilient Distributed Dataset (RDD), which is a partitioned set of elements that can be processed in parallel by Spark [Spark 2022]. This transformation to an RDD allows for using the method *RDD.ZipWithIndex* to create the index. After creating the index we converted the RDD into a DataFrame again. The indexes are defined on the attributes *time_ day*, *time_ month* and *time_ year* in *Dim_ Time*, on the attributes *locality_ lat*, *locality_ long* and *locality_ city* in Dim_Locality, and on attributes that are foreign keys. Each query presented in Table II was executed 21 times, and the first execution is not considered in the evaluation. The data used in the experiments presented in this article can be downloaded at `https://osf.io/pa2zn/`.

6.3 Results Discussion

Table III presents the average query processing time (*i.e.*, $\overline{x}$), the standard deviation (*i.e.*, σ) and the variance (*i.e.*, $\hat{V}$) of each query depicted in Table II (the first execution time of each query is not considered for calculating the average processing time). One can note that Apache Spark presents the

Table III: Performance of Queries Q1 to Q14 (msec)

Query	Apache Drill	Apache Spark	ElasticSearch	MonetDB	PostgreSQL
	$\overline{x}$	$\overline{x}$	$\overline{x}$	$\overline{x}$	$\overline{x}$
Q1	21664,2	36,1	37244,85	342,15	11158,35
Q2	6371,3	39,1	488,45	21,5	1292,5
Q3	7845,55	40,8	23909,5	66,4	2629,85
Q4	11638,2	49,2	68089,55	2278,7	5347,3
Q5	5989,95	70,75	280,55	37,05	1286,8
Q6	5831,15	43,7	64,0	26,2	1144
Q7	5964,9	71,65	417,05	150,15	1334,6
Q8	6478,5	32,7	489,3	239,15	2002,9
Q9	6228,65	70,4	490,6	239,5	2001,1
Q10	6167,25	40,55	491,15	239,9	2010,25
Q11	7465,0	125,25	250,12	253,8	25529,95
Q12	8766,5	86,85	3923,5	828,85	1540,8
Q13	43625,85	63,25	368,4	7201,85	1381
Q14	7281,2	47,8	54,4	760,55	2410,1

best performance for 11 of 14 queries, followed by MonetDB which presented the best performance for 3 of 14 queries. To understand this behavior of Spark, it is necessary to understand how Spark processes SQL queries. The SQL query is initially translated to an Abstract Syntax Tree (AST). Optimization rules can be applied to these operations using Catalyst, the Spark optimizer. Each expression of the generated (and eventually optimized) AST is transformed into an AST implemented in Scala, and then this code is compiled to Java bytecode, which is sent to the Spark Executors (*i.e.*, the processes that effectively process the query). When this code is available, Spark creates a logical plan of RDDs. Spark transforms this logical plan into a physical execution plan consisting of stages and parallel tasks. The physical plan can be executed in parallel in different Executors, consuming partitions of the RDD. This way, Spark can benefit from parallelism, even in a single virtual machine with multiple vCPUS, while the other approaches do not (although MonetDB has a version with parallel support, this version is not used in the experiments presented in this article). It is also important to highlight that `DW-TEMPO` has approximately 4.8 GB (the fact table contains 12,401,780 tuples) and fits almost entirely in memory. Thus, Spark can perform practically all the processing in memory using *Data Frames*, which makes the execution more efficient compared to other approaches that need to access the disk.

MonetDB presented the best processing times for queries Q2, Q5, and Q6. In addition, despite it does not execute in parallel in our experiments, the processing times of queries Q1 to Q11 are acceptable for interactive Vis (*i.e.*, lower than 0.5 seconds). The advantage of MonetDB is not only the faster reading, but it also benefits from its columnar database technology that only loads into memory the attributes that are needed. The performance of MonetDB for queries Q12 to Q14 presented a non-negligible overhead due to multiple joins with the polygon that represents the Ingá region. ElasticSearch also presented acceptable results for queries Q5 to Q11, Q13, and Q14, but the performance of ElasticSearch for the query Q12 presented an overhead. In addition, in ElasticSearch, large queries can consume too much memory during the parsing phase, in which case the Elasticsearch SQL engine can also add an overhead.

Differently from Spark, MonetDB, and ElasticSearch, both Apache Drill and PostgreSQL presented query processing times superior to 0.5 seconds for most queries. Even creating indexes whenever possible, *e.g.*, on the foreign keys of the fact table, on attributes that are used for selections (*e.g.*, "year = 2018"), the processing times are longer than the expected 0.5 seconds. In the case of Apache Drill, it has to interpret the submitted query into a distributed plan, send it to all the drillbit processes, which then lookup the data sources, access the data using the connectors, execute the query, return the results to the first node for aggregation, and then send the final output. Depending on the data source, Drill may need to load all the data and filter it separately which adds additional overheads. One option to reduce overheads is to use Parquet format instead of loading data from a CSV file.

This way, Spark SQL and MonetDB turned out to be the recommended approaches for processing queries in TEMPO. It is important to highlight that this result is justified since TEMPO typically submits OLAP queries, and the data are not updated or deleted after loading (since TEMPO only loads data captured by sensors). This OLAP scenario is beneficial for both MonetDB and Spark. Another conclusion of the experiments is that Polystore systems such as Apache Drill are not suitable for scenarios where the user has a medium/small dataset stored on a single machine. In this scenario, most existing relational databases will present a better performance than Drill. As next steps, we plan to evaluate other approaches such as BigDAWG [Duggan et al. 2015] and DuckDB [Raasveldt and Mühleisen 2019], in addition to evaluating parallel processing in distributed environments.

7. CONCLUSIONS AND FUTURE WORK

Many applications in the Data Science field are based on Vis techniques. In many cases, these applications require the use of interactive Vis, which traditionally sets a limit of 0.5 seconds as the maximum response time to actions in the interactive application. This response time should encompass data access, filtering, and aggregation. Although there are specific techniques and data structures to support such visualizations, the data management approaches proposed in recent years can be used to support Vis systems without much effort. In this article, we present a comparative study of data management approaches aiming to support interactive Vis. We compared the performance of known solutions such as PostgreSQL, MonetDB, ElasticSearch, and Spark SQL to process multiple spatiotemporal queries of the TEMPO tool to visualize rainfall data. From the 14 queries, Spark SQL and MonetDB are the data management approaches that presented the best performance, being capable of processing most queries in less than 0.5 seconds. One of the advantages of Spark SQL is that it performs in-memory processing, taking advantage of Spark's Data Frames structure. MonetDB is also able to process most of the queries in less than 0.5 seconds but presented a performance loss on queries involving spatial joins. Future work includes evaluating new approaches, including Polystore systems such as BigDAWG and in-memory solutions such as DuckDB.

ACKNOWLEDGEMENTS

This work was funded by CNPq, FAPERJ, and Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

REFERENCES

Armbrust, M., Bateman, D., Xin, R., and Zaharia, M. Introduction to spark 2.0 for database researchers. In *SIGMOD '16*. San Francisco, California, USA, pp. 2193–2194, 2016.

Battle, L., Chang, R., and Stonebraker, M. Dynamic prefetching of data tiles for interactive visualization. In *SIGMOD*. ACM, pp. 1363–1375, 2016.

Battle, L., Eichmann, P., Angelini, M., Catarci, T., Santucci, G., Zheng, Y., Binnig, C., Fekete, J.-D., and Moritz, D. Database benchmarking for supporting real-time interactive querying of large data. In *SIGMOD*. New York, NY, USA, pp. 1571–1587, 2020.

Bays, W. and Lange, K. SPEC: driving better benchmarks. In *Third Joint WOSP/SIPEW International Conference on Performance Engineering, ICPE'12, Boston, MA, USA - April 22 - 25, 2012*, D. R. Kaeli, J. Rolia, L. K. John, and D. Krishnamurthy (Eds.). ACM, pp. 249–250, 2012.

Bilokon, P. A. *Python, Data Science and Machine Learning - From Scratch to Productivity.* WorldScientific, 2022.

Blanco, G., Traina, A. J. M., Jr., C. T., Azevedo-Marques, P. M., Jorge, A. E. S., de Oliveira, D., and Bedo, M. V. N. A superpixel-driven deep learning approach for the analysis of dermatological wounds. *Comput. Methods Programs Biomed.* vol. 183, 2020.

Boncz, P. A., Flokstra, J., Grust, T., van Keulen, M., Manegold, S., Mullender, K. S., Rittinger, J., and Teubner, J. Monetdb/xquery-consistent and efficient updates on the pre/post plane. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, Y. E. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Böhm, A. Kemper, T. Grust, and C. Böhm (Eds.). Lecture Notes in Computer Science, vol. 3896. Springer, pp. 1190–1193, 2006.

Caban, J. J. and Gotz, D. Visual analytics in healthcare–opportunities and research challenges, 2015.

Ciesielska, M., Rizun, N., and Janowski, T. Interdisciplinarity in smart sustainable city education: exploring educational offerings and competencies worldwide. In *54th Hawaii International Conference on System Sciences, HICSS 2021, Kauai, Hawaii, USA, January 5, 2021*. ScholarSpace, pp. 1–10, 2021.

de Oliveira, D., Ocaña, K. A. C. S., Baião, F. A., and Mattoso, M. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J. Grid Comput.* 10 (3): 521–552, 2012.

de Oliveira, D., Ocaña, K. A. C. S., Ogasawara, E. S., Dias, J., de A. R. Gonçalves, J. C., Baião, F. A., and Mattoso, M. Performance evaluation of parallel strategies in public clouds: A study with phylogenomic workflows. *Future Gener. Comput. Syst.* 29 (7): 1816–1825, 2013.

Doraiswamy, H. and Freire, J. A gpu-friendly geometric data model and algebra for spatial queries. In *SIGMOD*. New York, NY, USA, pp. 1875–1885, 2020.

Drill, A. Apache drill - https://drill.apache.org/docs/drill-introduction/, 2022.

Duggan, J., Elmore, A. J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T., and Zdonik, S. B. The bigdawg polystore system. *SIGMOD Rec.* 44 (2): 11–16, 2015.

Eichmann, P., Zgraggen, E., Binnig, C., and Kraska, T. Idebench: A benchmark for interactive data exploration. In *SIGMOD*. New York, NY, USA, pp. 1555–1569, 2020.

Elastic. Elasticsearch docs - https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html, 2021a.

Elastic. Elasticsearch intro - https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html, 2021b.

Elastic. Elasticsearch search and analyze - https://www.elastic.co/guide/en/elasticsearch/reference/current/search-analyze.html, 2021c.

Eldawy, A. and Mokbel, M. F. The era of big spatial data. *Proc. VLDB Endow.* 10 (12): 1992–1995, aug, 2017.

Freire, J., Koop, D., Santos, E., and Silva, C. T. Provenance for computational tasks: A survey. *Comput. Sci. Eng.* 10 (3): 11–21, 2008.

Ghojogh, B. *Data Reduction Algorithms in Machine Learning and Data Science*. Ph.D. thesis, University of Waterloo, Ontario, Canada, 2021.

Hausenblas, M. and Nadeau, J. Apache drill: interactive ad-hoc analysis at scale. *Big data* 1 (2): 100–104, 2013.

Hochstetler, J., Hochstetler, L., and Fu, S. An optimal police patrol planning strategy for smart city safety. In *18th IEEE International Conference on High Performance Computing and Communications; 14th IEEE International Conference on Smart City; 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016, Sydney, Australia, December 12-14, 2016*, J. Chen and L. T. Yang (Eds.). IEEE Computer Society, pp. 1256–1263, 2016.

Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, S., and Kersten, M. Monetdb: Two decades of research in column-oriented database. *IEEE Data Engineering Bulletin*, 2012.

Jiang, L., Rahman, P., and Nandi, A. Evaluating interactive data systems: Workloads, metrics, and guidelines. In *SIGMOD*. New York, NY, USA, pp. 1637–1644, 2018.

Kimball, R. and Ross, M. *The Data Warehouse Toolkit: The complete guide to dimensional modeling*. Wiley, New York, 2002.

Kononenko, O., Baysal, O., Holmes, R., and Godfrey, M. W. Mining modern repositories with elasticsearch. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, P. T. Devanbu, S. Kim, and M. Pinzger (Eds.). ACM, pp. 328–331, 2014.

Kranas, P., Kolev, B., Levchenko, O., Pacitti, E., Valduriez, P., Jiménez-Peris, R., and Patiño-Martínez, M. Parallel query processing in a polystore. *Distributed Parallel Databases* 39 (4): 939–977, 2021.

Lins, L. D., Klosowski, J. T., and Scheidegger, C. E. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE TVCG* 19 (12): 2456–2465, 2013.

Liu, Z. and Heer, J. The effects of interactive latency on exploratory visual analysis. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2014a.

Liu, Z. and Heer, J. The effects of interactive latency on exploratory visual analysis. *IEEE transactions on visualization and computer graphics* 20 (12): 2122–2131, 2014b.

Magdon-Ismail, T. Tpcx-hs. In *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya (Eds.). Springer, 2019.

Makris, A., Tserpes, K., Spiliopoulos, G., Zissis, D., and Anagnostopoulos, D. Mongodb vs postgresql: A comparative study on performance aspects. *GeoInformatica* 25 (2): 243–268, 2021.

Mizutori, M. and Guha-Sapir, D. Human cost of disasters 2000-2019. Tech. rep., United Nations Office for Disaster Risk Reduction, 2020.

MonetDB. Monetdb - https://www.monetdb.org/documentation-jul2021/user-guide/introduction-to-monetdb/key-concepts/, 2021.

Munzner, T. *Visualization Analysis and Design*. A.K. Peters visualization series. A K Peters, 2014a.

Munzner, T. *Visualization analysis and design.* CRC press, 2014b.

Nascimento, L. C., Knust, L., Santos, R., Sá, B., Moreira, G., Freitas, F., Moura, N., Lage, M., and Oliveira, D. Análise de dados pluviométricos multi-fonte baseada em técnicas olap e de visualização: uma abordagem prática. In *Anais do XII Workshop de Computação Aplicada à Gestão do Meio Ambiente e Recursos Naturais.* SBC, Porto Alegre, RS, Brasil, pp. 1–10, 2021.

Nascimento, L. C., Lage, M., and de Oliveira, D. Um estudo sobre o uso de abordagens de gerência de dados em sistemas de análise visual de dados espaço-temporais. In *Anais do XXXVI Simpósio Brasileiro de Bancos de Dados.* SBC, Porto Alegre, RS, Brasil, pp. 361–366, 2021.

Ocaña, K. A. C. S., Silva, V., de Oliveira, D., and Mattoso, M. Data analytics in bioinformatics: Data science in practice for genomics analysis workflows. In *11th IEEE International Conference on e-Science, e-Science 2015, Munich, Germany, August 31 - September 4, 2015.* IEEE Computer Society, pp. 322–331, 2015.

Olivera, H. V., RuiZhe, G., Huacarpuma, R. C., da Silva, A. P. B., Mariano, A. M., and Holanda, M. Data modeling and nosql databases - A systematic mapping review. *ACM Comput. Surv.* 54 (6): 116:1–116:26, 2021.

Poess, M. TPC-DS. In *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya (Eds.). Springer, 2019a.

Poess, M. TPC-H. In *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya (Eds.). Springer, 2019b.

Poess, M. and Nambiar, R. TPC. In *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya (Eds.). Springer, 2019.

Ponchateau, C. *Conception et exploitation d'une base de modèles: application aux data sciences. (Design and Exploitation of a Models Database: Applied to Data Sciences).* Ph.D. thesis, Ecole Nationale Supérieure de Mécanique et d'Aérotechique, Poitiers, France, 2018.

PostGIS. Postgis - https://postgis.net/, 2021.

PostgreSQL. Index types - https://www.postgresql.org/docs/9.5/indexes-types.html, 2022a.

PostgreSQL. Postgresql - https://www.postgresql.org/about/, 2022b.

Raasveldt, M. and Mühleisen, H. Duckdb: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska (Eds.). ACM, pp. 1981–1984, 2019.

Ribeiro, C. C. Sports scheduling: Problems and applications. *Int. Trans. Oper. Res.* 19 (1-2): 201–226, 2012.

Samet, H. *Applications of spatial data structures - computer graphics, image processing, and GIS.* Addison-Wesley, 1990a.

Samet, H. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990b.

Schmidt, J. Usage of visualization techniques in data science workflows. In *Proc. of the VISIGRAPP.* pp. 309–316, 2020.

Sevim, A., Mahin, M. T., Vu, T., Maxon, I., Eldawy, A., Carey, M., and Tsotras, V. A brief introduction to geospatial big data analytics with apache asterixdb. In *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on APIs and Libraries for Geospatial Data Science.* pp. 1–2, 2021.

Spark, A. Apache spark aqe - https://spark.apache.org/docs/latest/sql-performance-tuning.htmladaptive-query-execution, 2021a.

Spark, A. Spark sql data sources - https://spark.apache.org/docs/latest/sql-data-sources.html, 2021b.

Spark, A. Apache spark - rdd - https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.rdd.html, 2022.

Strobl, C. Postgis. In *Encyclopedia of GIS*, S. Shekhar, H. Xiong, and X. Zhou (Eds.). Springer, pp. 1623–1630, 2017.

Thorndahl, S. and Willems, P. Probabilistic modelling of overflow, surcharge and flooding in urban drainage using the first-order reliability method and parameterization of local rain series. *Water Research* 42 (1): 455–466, 2008.

Verma, R. Smart city healthcare cyber physical system: Characteristics, technologies and challenges. *Wirel. Pers. Commun.* 122 (2): 1413–1433, 2022.

Wang, A., Zhang, A., Chan, E. H. W., Shi, W., Zhou, X., and Liu, Z. A review of human mobility research based on big data and its implication for smart city development. *ISPRS Int. J. Geo Inf.* 10 (1): 13, 2021.

Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, S. D. Gribble and D. Katabi (Eds.). USENIX Association, pp. 15–28, 2012.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., Ghodsi, A., Gonzalez, J., Shenker, S., and Stoica, I. Apache spark: a unified engine for big data processing. *Commun. ACM* 59 (11): 56–65, 2016.

Zheng, Y., Wu, W., Chen, Y., Qu, H., and Ni, L. M. Visual analytics in urban computing: An overview. *IEEE Transactions on Big Data* 2 (3): 276–296, 2016.

Zimbrao, G. and de Souza, J. M. A raster approximation for processing of spatial joins. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, A. Gupta, O. Shmueli, and J. Widom (Eds.). Morgan Kaufmann, pp. 558–569, 1998.