Knowledge and Information Systems (2024) 66:2063–2088
https://doi.org/10.1007/s10115-023-02009-y

REGULAR PAPER

How to manage massive spatiotemporal dataset from stationary and non-stationary sensors in commercial DBMS?

Vincenzo Norman Vitale[1] · Sergio Di Martino[1] · Adriano Peron[2] · Massimiliano Russo[1] · Ermanno Battista[3]

Received: 23 December 2022 / Revised: 21 September 2023 / Accepted: 18 October 2023 /
Published online: 20 November 2023

Abstract
The growing diffusion of the latest information and communication technologies in different contexts allowed the constitution of enormous sensing networks that form the underlying texture of smart environments. The amount and the speed at which these environments produce and consume data are starting to challenge current spatial data management technologies. In this work, we report on our experience handling real-world spatiotemporal datasets: a stationary dataset referring to the parking monitoring system and a non-stationary dataset referring to a train-mounted railway monitoring system. In particular, we present the results of an empirical comparison of the retrieval performances achieved by three different off-the-shelf settings to manage spatiotemporal data, namely the well-established combination of *PostgreSQL* + *PostGIS* with standard indexing, a clustered version of the same setup, and then a combination of the basic setup with *Timescale*, a storage extension specialized in handling temporal data. Since the non-stationary dataset has put much pressure on the configurations above, we furtherly investigated the advantages achievable by combining the TSMS setup with state-of-the-art indexing techniques. Results showed that the standard indexing is by far outperformed by the other solutions, which have different trade-offs. This experience may help researchers and practitioners facing similar problems managing these types of data.

✉ Vincenzo Norman Vitale
vincenzonorman.vitale@unina.it

✉ Sergio Di Martino
sergio.dimartino@unina.it

Adriano Peron
adriano.peron@units.it

Massimiliano Russo
massimilia.russo@studenti.unina.it

Ermanno Battista
ermanno.battista@ivmtech.it

1 Department of Information Technology and Electrical Engineering (DIETI), University of Naples ‘Federico II’, Corso Umberto I 40, 80138 Naples, Italy

2 University of Trieste, Piazzale Europa, 1, 34127 Trieste, Italy

3 IVM srl, Via Benedetto Brin, 59, 80142 Naples, Italy

Keywords Mobility datasets · Spatiotemporal databases · Database indexing · Temporal databases · Geographical information systems (GIS) · RDBMS

1 Introduction

In the last decades, we experienced an unprecedented data deluge whose cause is the rapid spread of the latest ICT technologies. In particular, the Internet of things (IoT) computing paradigm, which enables the seamless integration of everyday objects through the network, has been identified as one of the principal sources of massive data production. As of now, the IoT has also been recognized as the key enabler for the establishment of the so-called intelligent environments in various operational contexts [1] like smart cities [2], in which it is fundamental for parking monitoring systems [3] or energy management [4]. On the other hand, the IoT is the foundational technology for Smart Factories [5], where it is essential for production chain assets integration or predictive maintenance on critical assets [6].

Despite smart environments being built for different purposes, it is possible to identify some common elements in their architectures. On the one hand, we recognize smart-environment's backbone in vast sensor networks spread over differently sized geographical areas [7, 8]. On the other hand, these systems are mainly thought to collect continuous and heterogeneous amounts of data, constantly flowing over the system, which must be managed for future usage.

Storing and retrieving such massive data streams is mainly entrusted to specialized database management systems (DBMSs), which usually treat such data as time-series [9, 10]. However, given the intrinsically geographical nature of sensor networks and produced data, they need to provide further attention to the spatial component, too [11]. Indeed, according to some recent studies, spatial data will increase in the next years and location-aware information will constitute more than 20% of the data produced every day [12–14]. Therefore, the ability to efficiently store and search vast spatial and temporal data collections is a fundamental requirement of the present-day data management architectures [15–17]. It follows that the choice of the most appropriate DBMS will be guided only in part by the form and nature of the data. Indeed, in a typical analytical setting, we need proper support for both structures and operators in order to comply with the demanding performance constraints imposed by context-specific applications [18, 19].

Despite the considerable number of DBMSs available on the market, few of them can provide adequate support for spatiotemporal data storage and retrieval. Firstly, time series management systems (TSMS) are specifically designed to manage temporal data [20, 21]. Then, the NoSQL systems are general-purpose databases that proved notable performances in managing extensive temporal data collections, while their support for geospatial data are usually considered limited [14, 22]. Finally, relational DBMSs (RDBMS) have been historically a reference in the management of geospatial data [14, 23], while their capabilities in managing massive (temporal) data were generally considered poor [20, 24]. In the last case, the advent of TSMS allowed RDBMSs to increase their ability to manage massive temporal data. However, to the best of our knowledge, RDBMS performances are poorly studied in the literature.

In this study, we provide experience in handling massive spatial data with a prominent temporal component, with two state-of-the-art Commercial Off The Shelf (COTS) RDBMS: the first one which is a combination of a well-established geospatial system, namely PostgreSQL [25] + PostGIS [25]. The second one is an extension of the former, where we introduced a

TSMS, more specifically Timescale [24]. We evaluated these two solutions with two real-world datasets with different spatial components (i.e., static and dynamic), obtaining this way four different valuable contexts.

Our objective is to evaluate the temporal complexity and disk space footprint of considered system, with different indexing and optimizations, under different real-world constraints.

The remainder of the paper is organized as follows. In Sect. 2 we provide an overview of considered Commercial Off-The-Shelf (COTS) DBMSs and related works. In Sect. 3, we describe the employed real-world datasets, along with our evaluation setup. Finally, in Sect. 4 we report on results achieved. Finally, some remarks conclude the paper.

2 Commercial TSMS and spatial database

Despite the renewed interest in spatial and temporal data analytics optimization [4, 16, 26–28], few studies have been conducted in assessing the performances of commercial DBMS in managing both massive dimensions [15, 29]. However, the management of massive time-series data produced by IoT sensors boosted the interest of academics and enterprises in the study and development of ad hoc DBMSs, namely the time series management systems (TSMS) [20], as can be seen in some recent studies [10, 30, 31]. Even if TSMS provide state-of-the-art performances in managing temporal data streams, they provide limited support (if any) to spatial data. For example, InfluxDB [32] one of the principal representatives for the TSMS category, provides an experimental support only for 2D data points, and S2Geometry indexing [33]. On the other hand, existing relational DBMSs (RDBMSs) solutions improved performances of their geospatial add-on data over the years, and nowadays are considered among the most reliable solutions for this kind of data [14, 15, 22]. Similarly, also traditional, NoSQL DBMSs provide only a limited support for spatial data management, in terms of data types (e.g., 3D points, 2D points, polygons), operators and indexing structures [14, 22, 34, 35]. For example, MongoDB, a widely used NoSQL database belonging to the document-store category, supports various geometry objects, while limiting the spatial indexing support at 2Dsphere [14]

Presently, a mere support for the spatial dimension is not sufficient, as the IoT data presents massive influence dimensions of temporal dimension too. Despite NoSQLs have emerged as a remedy to RDBMSs shortcomings in handling big data, they demonstrated lower performances in managing massive time-series if compared to TSMS [10, 31]. In addition, NoSQL stores proved to be unreliable in retrieving massive spatial data if compared to RDBMS [22]. Conversely, RDBMSs offer higher reliability while handling spatial data [15, 22] and proved to be a suitable option in handling massive temporal data [31]. In addition, consolidated solutions based on RDBMS for spatial data management, namely PostgreSQL [25] and PostGIS [36], have been extended to face challenges deriving from the massive aspect of temporal data from IoT, with capabilities offered by modern TSMS (i.e., Timescale [24]). However, up to our knowledge, available studies do not report on the usage of these systems to manage both dimensions at the same time.

To this end, the COTS DBMS selected for this study are PostgreSQL [25], PostGIS [36] and TimeScale [24] which are described in Sect. 2.1. Then, assuming temporal indexing as imposed by the evaluated TSMS, we investigated the combination with different spatial indexing techniques among those available (see Sect. 2.2).

2.1 The considered COTS DBMSs

In this work, we studied above-mentioned COTS DBMS performances with different settings to manage with two types of typical, massive datasets generated by IoT sensor networks presenting stationary and non-stationary spatial components. In particular, we evaluate the potential advantages and trade-offs derived by introducing a time series management system (TSMS) on top of a well-established COTS database for spatial data management. Thus, we start from the established combination of *PostgreSQL* and *PostGIS*, a widely adopted technological solution for spatial data management. Then we extend it with TSMS [20] capabilities by leveraging the freely available *Timescale* extension for PostgreSQL.

2.1.1 PostgreSQL

PostgreSQL [25] is an object-relational DBMS (ORDBMS), sparkling from a project of the University of Berkley, currently open source and multi-platform. It is widely used in many different contexts and applications for industrial, government, and academic purposes (e.g., [31, 37, 38]). According to Brewer's *CAP Theorem* [39], PostgreSQL is classified as **AC**, as it is focused on high Availability and strong consistency. Based on the relational data model, PostgreSQL supports a large part of the ISO/IEC SQL standard and offers total compliance with ACID properties. At the time of the experiment, the official documentation stated *"[...] on the version 12 release in October 2019, PostgreSQL conforms to at least 160 of the 179 mandatory features for SQL:2016 Core conformance. As of this writing, no relational database meets full conformance with this standard."* A key feature of PostgreSQL is the possibility of adding plug-ins to increment the capabilities and features offered by the DBMS. To date, several plug-ins are available for domain-specific demands, like spatial (*PostGIS*), temporal (*TimeScale*), Full Text Search engine (*OpenFTS*), etc.

2.1.2 PostGIS

PostGIS [36] is one of the key extensions of PostgreSQL, intended to add support for geographic objects and location queries [23] in a manner that is compliant with the *Simple Feature Access* specifications from ISO/Open Geospatial Consortium (OGC) [40]. PostGIS, like PostgreSQL, is open-source. In particular, PostGIS adds to PostgreSQL a wide range of methods to represent spatial objects; for example, it uses WKT (Well Known Text) and WKB (Well Known Binary) to express different geometry types, spanning from simple 2D coordinates to 3D spatial data objects to support 3DZ, 3DM and 4D coordinates. As for data indexing, PostGIS adds support for *R-tree* [41] and Block Range INdex (BRIN) [42] (see Sect. 2.2.2 spatial indexes. Indeed, let us note that traditional index structures, based on the one-dimensional ordering of key values, like the B-trees, do not work with spatial data, as the search space is multidimensional.

2.1.3 Timescale

Timescale [24, 43] is TSMS [20], intended as an extension of PostgreSQL. Based on the relational model, Timescale transactions comply with ACID properties. Moreover, it offers the advantages of being SQL-compliant to manage Time-Series data and the ability to perform joins among time-series and standard relational tables. Timescale physically orders records by a mandatory temporal field, on which an index (usually a B-tree [44, 45]) is defined by default, like in other TSMSs [20]. The data model stores values in *Hypertables*, namely tables internally partitioned into smaller tables called *chunks*. The maximum period covered by a chunk (called chunk_time_interval) should be established at Hypertable's creation. Then an optional vertical partitioning could be performed based on a meaningful column by specifying the amount or a split function. This way, Timescale breaks extensive tables into an unspecified number of smaller ones, doing the same for indexes. So retrieval operations can be broken up into smaller operations, allowing parallelization and resulting in better performances. Of course, handling such a structure is harder since each operation on a table could involve a Hypertable. To make a chunk behave like a regular table, timescale internally splits the operation according to involved hypertables, resulting in more complex query planning transparent to end-users. It is worth noting that Timescale's partitioning presents some similarities with partitioning techniques, such as sharding [46], adopted by NoSQL storage systems.

2.2 Spatial indexing in the considered COTS DBMS

As analytic solutions based on information collected via IoT sensor networks should provide fast data access over voluminous collections, proper data indexing solutions are a fundamental prerequisite [15, 27]. Choosing the best combination of indexes depends on analytics tasks and data (type and shape). Among the spatial indexing structures supported by the considered COTS DBMS, we decided to compare two different indexes that showed notable performances when dealing with trajectories data [15], namely the R-tree (a dense index) and the spatial BRIN (a sparse index). Although both structures present some advantages when dealing with spatial data, to the best of our knowledge, no studies compare their characteristics in the presence of different types of massive spatial or spatiotemporal data. In the following, we provide a deeper description of considered indexing structures.

2.2.1 R-tree index

The R-tree is a tree-like indexing structure employed to speed-up search operations on multidimensional data like spatial ones. In the case of spatial data, it acts by approximating groups of spatial objects (e.g., points, polygons) by a Minimum Bounding Rectangle (MBR) [47], namely the minimum rectangle capable of containing all the objects belonging to the group. In a further step of approximation a group can contain both spatial objects or MBRs. A group reunites both MBRs or actual spatial objects based on spatial proximity. At the leaves level, every node contains pointers to actual spatial data (red rectangles in Fig. 1). On the other hand, internal nodes recursively point to other nodes represented by an MBR (black and blue rectangles in Fig. 1). So, the root node will be an MBR recursively containing all internal and leaf nodes. To perform a searching operation, we will define a search filter consisting of a generic polygon that will be compared with every MBR encountered along a path. The search stops when it reaches spatial data in leaf nodes or if the intersection between the filter and any MBR is empty.

It is worth noting that a non-empty intersection with the MBR of an internal node causes the filter to compare against the MBRs of all child nodes, which can also produce an empty result. This means that a larger filter implies a greater probability of wasting resources and time.

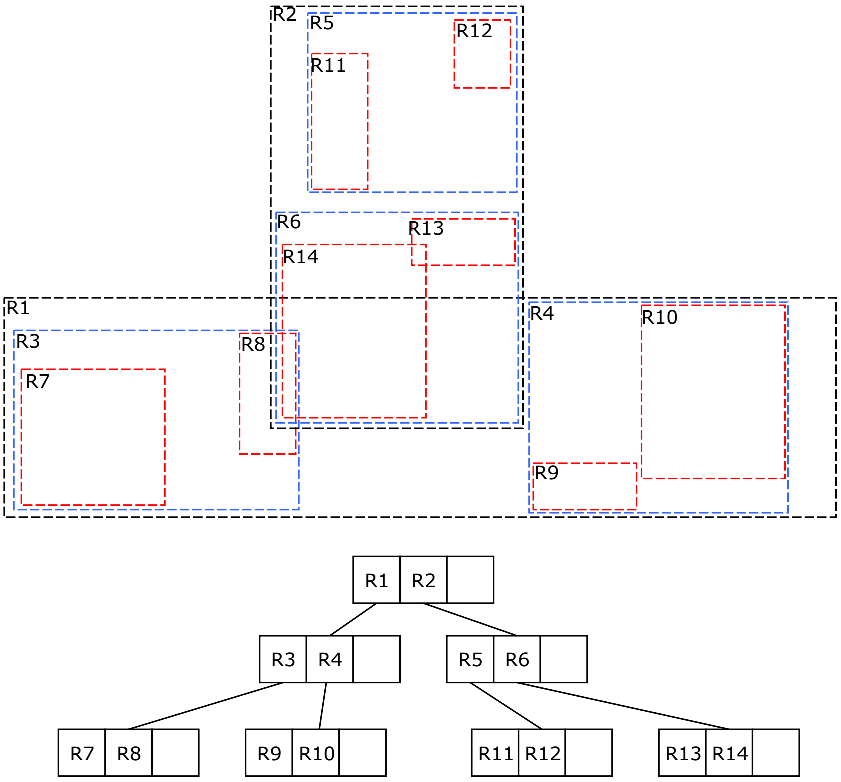


Fig. 1 An example of a bi-dimensional R-tree, reproduced from [47]

Although this index has demonstrated its efficiency in targeted analyses of spatial data [15], its effectiveness in the presence of a massive temporal component has been barely studied.

2.2.2 Spatial BRIN index

A Block Range Index (BRIN) is a sparse indexing structure [15, 48] used to improve performances on extensive collections (e.g., tables). It assumes the existence of a correlation between the indexed dimension and the internal collection ordering. It groups large tables in fixed disk page range units, storing the maximum and the minimum value for each unit and associates an MBR enclosing all tuples belonging to that range. Inserting or updating a tuple inside a page range causes an update to the associated page range when necessary. On the other hand, a cancellation does not cause any updates. As for R-trees, a filter is represented by a generic polygon. The query processor will scan only index entries for which the intersection between the search polygon and MBRs is not empty. This structure tries to keep the overlap between the MBRs as low as possible, allowing for scanning the minimum amount of storage memory necessary to return the result. In contrast with R-tree, the Spatial BRIN performs better as the search polygon gets wider on dense spatial data.

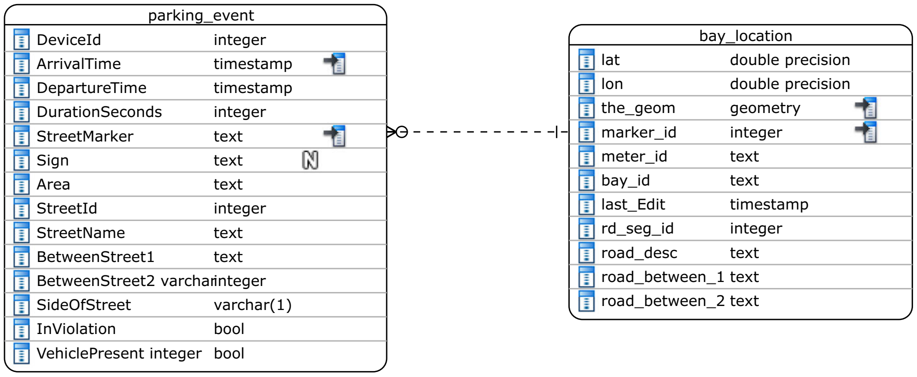


Fig. 2 Entity relation diagram for the considered mobility dataset

3 The empirical evaluation setup

This section describes the evaluation process of the considered COTS DBMSs with massive spatiotemporal data from different mobility contexts. In particular, Sect. 3.1 describes the two real-world datasets used during the assessment. Then, Sect. 3.3 describes typical data requests for each considered dataset and the metrics employed to evaluate results. Finally, Sect. 3.2 describes all the investigated settings designed on top of the considered COTS DBMSs.

3.1 The considered spatiotemporal datasets

As sensor networks can generate considerable amounts of spatiotemporal data at an unprecedented rate [49–51], it is not uncommon to have datasets of many Gigabytes per month or even per week. In this study, we employ two real-world spatiotemporal datasets with different geographical characteristics. The stationary dataset from Melbourne's parking monitoring system represents the typical fixed-position data stream produced in a Smart City. Then, the non-stationary dataset from a rail monitoring system is an example of the non-stationary data produced by IoT-based monitoring systems. Please note that both datasets have a massive temporal component, making it challenging to manage such data with standard databases.

The stationary dataset

The considered stationary dataset comes from the on-street parking monitoring system deployed in the Municipality of Melbourne, the capital city of the Australian state of Victoria. It consists of more than 4000 sensors and approximately 480 street segments across the Melbourne Central Business District (CBD). The collected data were made publicly available, both as real-time data through some APIs, both as aggregate historical data, in downloadable format [52]. Let us note that, like in other similar projects [53], in this case, the instrumented area is just a fraction of the total urban area.

We focused on a dataset containing such parking data, from 2011 to 2017, for a total of about 275 Million records. The conceptual model, expressed as an Entity Relationship Diagram (ERD), is reported in Fig. 2. More in detail, the database schema comprises two tables: *bay_location* and *parking_events*. The first one represents the parking stalls, containing information about the geographical location of each parking bay, the shape, the road segment it belongs to, the bay identifier, and the sensor's ID. The second entity models parking availability information collected by deployed sensors. Here, each tuple describes the state of the parking spot, whether it is free or occupied, plus some additional information to link it to a sensor and a parking bay. This table is richly populated with temporal data and represents the most prominent part of the dataset.

As a consequence, each spatiotemporal query performed on this dataset (e.g., retrieve all the parking events in a range of 500 mt from the Stadium on a specific day) requires joining these two tables, to get spatial filtering from *bay_locations* and temporal filtering and parking data from the *parking_events*.

When implemented in *PostgreSQL* with the spatial extension *PostGIS*, this dataset required about 40GB of storage space without considering the size for the indexing.

The non-stationary dataset

Nowadays, it is usual to use mobile probes to monitor roads, railways, movable objects and other infrastructures [54, 55]. Therefore, vast amounts of spatiotemporal data are continuously and constantly collected from ad hoc monitoring sensor networks [56, 57], probe vehicles [57–60] and crowd-sourced devices [2, 60].

The dataset we will use was collected from a rail probe monitoring system mounted on a train. It allows for continuous monitoring of rails and related elements as trains run over the railways during duty hours. The system comprises a number of 3-axial accelerometers, mounted on a wagon's axle-boxes recording expected and anomalous vibrations. Sensors are connected to a control unit which stores, combines and enriches acquired measurements with spatiotemporal pathways and contextual information. The dataset covers a period of 30 days and a stretch of about 2 kms, for a total of about 58 Million of records.

This dataset is a typical example of raw sensor measurements collected to monitor the health status of a railway infrastructure. In more general scenario, the payload of similar systems could be incremented with additional sensor sources such as such as gyroscopes, inertial measurement units, or optical measurement devices. In these dataset, one of the possible dimensional roll-up can take into account, as a possible independent variable, the distance of the central unit from a reference point on the railway line: the raw measures are then analyzed in ETL processes to derive quantitative information about the infrastructure as a function of space.

We designed a conceptual model composed of two parts expressed by ERDs in Figs. 3 and 4. The first part of the model is made by the tables *TripInfo* and *GeneralData* (see Fig. 3), reporting respectively generic information about the trip and measurements made from sensors during a trip. Hence, we extended the model with the *GeoPartitioning* table (see Fig. 4) which indicates spatial partitioning over specific routes, and it is meant to be used to filter *GeneralData*. The *GeoPartitioning* table joins with the *GeneralData* table through the tot_pk_m attribute, which expresses the extension of the railway from a reference point. The *GeneralData* table contains the prominent part of the dataset, the attributes *time* and *geog* indicate respectively a unix-like timestamp expressed in nanoseconds and geographical coordinates (expressed as a point) for each acquisition. Measurements from each accelerometer are indicated with the generic attributes x y z followed by $< id >$ referred to the sensor.

When ingested in PostgreSQL with the spatial and temporal extensions, this dataset required about 16GB of storage space. The storage space doesn't include the size of the indexes.

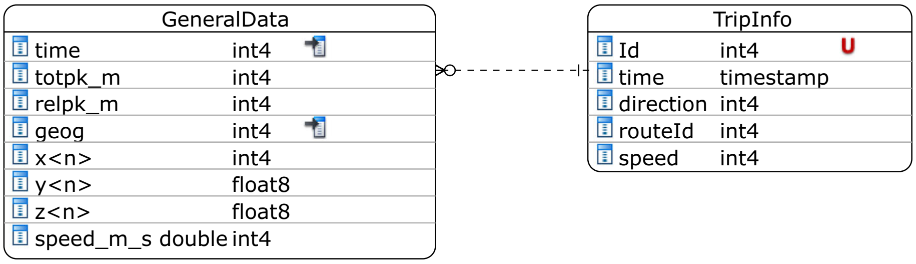


Fig. 3 Base entity relation diagram for the considered non-stationary dataset

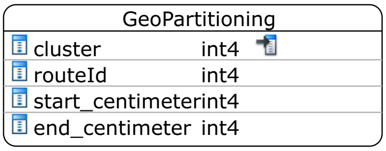


Fig. 4 ERD of the spatial partitioning table for the considered non-stationary dataset

3.2 The investigated COTS databases settings

To assess pros and cons of introducing a commercial TSMS like Timescale on top of established spatial databases, two combinations of the considered systems were defined, namely *PostgreSQL* + *PostGIS* (the Baseline) and *PostgreSQL* + *PostGIS* + *Timescale*. Let us note that, one of the main differences while using Timescale is a different ordering of the records within the physical files on the mass storage. This motivated us to investigate whether differences in performances (if any) might be due to the new algorithms and data structures of the TSMS or simply to the different physical ordering of the records. As a consequence, a further setting has been defined, exploiting the possibility offered by *PostgreSQL* to impose a physical ordering of the records in the files: such operation is named clustering.

So the considered off-the-shelf settings are:

(1) (Baseline) The relational DBMS *PostgreSQL* with the spatial extension *PostGIS*, with an R-tree on the spatial dimension. We will refer to this as *PostGIS*.
(2) The relational DBMS *PostgreSQL* with the spatial extension *PostGIS*, using advanced data sorting and indexing capabilities of the DBMS, with an R-tree on the spatial dimension. We will refer to this as *Clustered PostGIS*.
(3) The relational DBMS *PostgreSQL* with the spatial extension *PostGIS* and with the time-series management extension timescale which not requires an explicit ordering on time (see Sect. 2.1.3), a kind of storage solution specialized in handling temporal data, with an R-tree on the spatial dimension. We will refer to this as *PostGIS* + *Timescale*.

In the above-mentioned three settings, an R-tree index has been defined on the spatial field. This further index proved to be most effective in the presence of a fixed spatial dimension and to present higher selectivity [15]. However, the non-stationary dataset presents a massive and unrestricted spatial component, which is likely to induce performance deterioration in presence of an R-tree index. This motivated us to investigate further aspects regarding spatial indexing in the presence of massive, non-stationary, spatial data streams. To this end, we add more settings based on the *PostGIS* + *Timescale* configuration:

(1) To reduce the impact of indexing on storage, we used a BRIN index on the spatial attribute from the massive table. We will refer to this as *PostGIS* + *Timescale BRIN*.

(2) To evaluate the impact of a higher level of fragmentation, we partitioned the monitored route and accordingly the massive table. The spatial index is defined on the partitioning table rather than on the massive one. We will refer to this as *PostGIS + Timescale Secondary Partition (SP)*.
(3) Since secondary partitioning introduces a higher parallelization level, we decided to introduce an R-tree index also on the massive table to evaluate its impact on performances. We will refer to this as *PostGIS + Timescale Secondary Partition (SP) R-tree*.
(4) To reduce the impact on disk size, we decided to introduce a $BRIN$ index also on the massive table with secondary partitioning. We will refer to this as *PostGIS + Timescale Secondary Partition (SP) BRIN*.

In Tables 1 and 2 we summarize the characteristics of the investigated settings.

Database tuning

Firstly, the default parameters of the database have been tuned to leverage the workstation's resources with each configuration. In particular, following also official documentation guidelines, we increased: the number of parallel workers to 16, namely the maximum number of processor's threads; shared_buffers to 8GB; work_mem to about 13MB, which represents the amount of memory used by a single query for small sorts and hash tables; effective_cache_size to 49GB, which, according to the official documentation, *"Sets the planner's assumption about the effective size of the disk cache that is available to a single query.... it is used only for estimation purposes...."*, has the effect of favouring an index scan rather than a disk sequential scan. In order to present the results of the above proposed settings, it is first necessary to understand some peculiarities of the data indexing employed during experiments. The choice of the proper indexes is a crucial part of database tuning. Indeed, the lack of suitable indexes will lead to the scan of the entire files from the disk during the execution of a query, with catastrophic impact on performances. Conversely, too many indexes will slow down data ingestion and occupy relevant amount of space on the disk. In addition, executing a query over secondary index is one of the most difficult and common case. Indeed, as seen in [61], a secondary index scan may or may not work better than a full sequential scan, and the system needs to perform access path selection to choose the best access method, relying on run-time characteristics during optimization. In our study, we defined indexes as follows.

Stationary dataset indexing

Since the *bay_locations* table represent only the parking spot data and does not contain any temporal information, only two small size indexes are defined on it. The first is a B-tree on the *meter_id* attribute, used to search locations by the ID of the sensor. The second one is an R-Tree/BRIN on the attribute *the_geom*, used to speed up the spatial queries. Then we have the *parking_events* table, containing the massive amount of temporal data that is handled differently in each of the three settings. In the *PostGIS* setting, a B-tree index has been defined on the *ArrivalTime* attribute, to support time-based queries. Since this attribute is not the primary key, records in the files on the storage memory are not physically ordered by this field. As a consequence, the B-tree index on *ArrivalTime* is a secondary, dense index. A B-tree was defined on the *StreetMarker* attribute, which is a Foreign Key for the *bay_locations* table, to optimize the join. In the *Clustered PostGIS* setting, the attribute *ArrivalTime* becomes the physical ordering field of the records, and thus the corresponding index becomes clustered. Finally, in the *PostGIS + Timescale* setting, we transform the *parking_events* table in an

Table 1 Characteristics of the investigated settings for both the stationary and non-stationary datasets

Setting	Technologies	Characteristics
PostGIS	PostgreSQL, PostGIS,R-tree	B-Tree on time attribute. R-Tree on space attribute
Clustered PostGIS	PostgreSQL, PostGIS, R-tree	B-Tree on time. R-Tree on space attribute. Clustered on B-Tree
PostGIS + Timescale	PostgreSQL, PostGIS, Timescale, R-tree	B-Tree on time. R-Tree on space attribute. Hypertable natively ordered on time
PostGIS + Timescale BRIN	PostgreSQL, PostGIS, Timescale, BRIN	B-Tree on time. BRIN on space attribute. Hypertable natively ordered on time

Table 2 Characteristics of the additional investigated settings for the non-stationary dataset

Setting	Technologies	Characteristics
PostGIS + Timescale BRIN	PostgreSQL, PostGIS, Timescale, BRIN	B-Tree on time. BRIN on space attribute. Hypertable natively ordered on time.
PostGIS + Timescale Secondary Partition (SP)	PostgreSQL, PostGIS, Timescale, R-tre	B-Tree on time. R-Tree on partitioning table. Hypertable natively ordered on time. Secondary partitioning on space.
PostGIS + Timescale Secondary Partition (SP) R-tree	PostgreSQL, PostGIS, Timescale, R-tree	B-Tree on time. R-Tree on partitioning and massive tables. Hypertable natively ordered on time. Secondary partitioning on space.
PostGIS + Timescale Secondary Partition (SP) BRIN	PostgreSQL, PostGIS, Timescale, R-tree, BRIN	B-Tree on time. R-Tree on partitioning table. BRIN on the massive table. Hypertable natively ordered on time. Secondary partitioning on space.

Hypertable, optimized for time series filtering, with a default interval dimension of 1 week. Being mandatory for each Hypertable, chunks are physically ordered by the time index (i.e., *ArrivalTime*), improving both range and specific value searches on the temporal attribute.

Not stationary dataset indexing

In each considered setting, the *TripInfo* table has no indexes since it contains only information about the trip and the train. On the *GeneralData* table, containing massive spatiotemporal data, we defined a B-tree index on the *time* attribute to support time-based filtering. As for the stationary dataset, the B-tree index has been used as a clustering index in the *Clustered PostGIS* setting. Then in the *PostGIS + Timescale* setting, it has been used for Hypertable partitioning in 1-week chunks. As the *GeneralData* table contains also the spatial component, we defined an R-tree index on the *geog* to support spatial filtering (see Table 1). Since the spatial component has different nature in this context and impacts performances differently, we introduced further settings to be investigated(see Table 2). In particular, we introduce the BRIN spatial index in place of an R-tree on the GeneralData table. It is supposed to reduce resources used to store the index and increment performance if the index is defined on top of an attribute that follows table's physical ordering. Since the spatial dimension is as prominent as the temporal one, we introduce secondary partitioning on top of the *GeneralData* table.

In particular, we define the dimensional table *GeoPartitioning* that contains rail's spatial partitions referenced by the spatial attribute from *GeneralData*. We obtained this table by partitioning the route in sections of similar length, based on the characteristics of the trips and of the railway section. After some investigation, we assumed an average section length of 120 ms based on some railway characteristics. Table 2 summarizes also the features of the additional investigated settings.

3.3 Experimental setup

To evaluate the consequences of introducing a TSMS on top of a GIS for managing massive mobility data, we first empirically compared the achievable performances using the stationary dataset, consisting of 275 million records, in which the temporal dimension is the most prominent. Then to assess the performance in the presence of both massive spatial and temporal dimensions, we employed the non-stationary dataset, consisting of 58 million records, in which the spatial component varies along with time.

The experiments were executed on a workstation equipped with an eight-core Ryzen 3800x processor, 64GB RAM, and a Samsung M.2 SSD with a storage capacity of 1TB.

For the stationary dataset, the results will report the average of five runs of the spatiotemporal query reported in Listing 1, for each parameter combination and each setting. Note that, based on previous studies regarding the use case of a user searching for a parking space [3, 30, 62] and that of an officer looking for parking violations [63], we derived a query and a set of parameters representing the considered use cases. The considered ERD, along with the query and the related set of parameters, are aimed at a practical comparison of the considered architectures under conditions depicted in the considered use cases.

For the non-stationary dataset, we will report the average of five runs of the spatiotemporal queries showed in Listing 2,3,4, for each parameter combination, each setting and each spatial indexing technique. Note that, in this case, the considered combinations of queries and parameters were defined together with the industrial partner of the research, being domain experts of the railway monitoring scenario. Queries are designed to let an analyst recover enough data to evaluate the railway's health state. To this end, the temporal filter resolution varies to let the railway sections' health be assessed during different periods (E.g., days, weeks, months) based on data acquired by various instrumented trains. In the same way, the spatial filter resolution varies to consider interactions with nearby railway sections or related equipment like railway switches [64, 65].

To let every considered storage solution operate in the same initial conditions and eliminate the caching policies' effects, the workstation was rebooted after each single execution.

To summarize, we will compare three basic configurations, in which the main difference resides in the physical arrangement of the records on disk. Starting from these configurations, we will compare the impact of two spatial index types. On the one hand, we rely on a tree-structured index (i.e., the R-tree) usually indicated for queries aimed at selecting well-defined (relatively small) quantities of data. On the other hand, we will consider a block index (i.e., the BRIN) which misses the refined superstructure of tree-based indexes and is usually employed for retrieving large-grained data collection. This comparison is based on two real-world examples, whose considered queries and parameters represent a wide range of relatively typical situations, ranging from very fine-grained data retrieval to larger-grained data retrieval.

Based on configurations defined in Sect. 3.2, we are going to compare the impact of two spatial index types, namely a tree-structured index, i.e., the R-tree, mainly suited for relatively small data selections and a block index, i.e., the BRIN, which conversely is primarily recommended for large-grained data recovery. Note that a block index usually misses a fine-grained search superstructure, which speeds up the search in tree-based indexes, thus resulting in a thinner and lightweight (in terms of required disk space) data structure. So a naive assumption would be to expect that the execution of a query on a range index should take far more time than the exact same query executed on a tree-based index. Still, the assumption is not necessarily true and it depends on a number of conditions in concrete contexts. Therefore, we will evaluate the impact of both indexes under different real-world constraints represented by datasets, queries and parameter settings described in this section, which span from the most selective (i.e., fine-grained) queries to coarser ones.

Stationary dataset retrieval

We investigated many spatiotemporal queries, obtaining homogeneous results. To simplify, we will describe in detail only one query, reported in Listing 1, meant to represent typical data retrievals that a Decision Maker of a Smart City might perform to get a better vision on mobility phenomena. Such a location-based query is designed to retrieve all the parking events that happened in a specific *Range* from a given *Center*, in a time interval going from *StartInterval* to *EndInterval*.

Listing 1 The parametric spatiotemporal query we used in our experiments on the Melbourne dataset.

```
SELECT *
FROM parking_events AS PE JOIN bay_locations AS BL
    ON PE.StreetMarker = BL.marker_id
WHERE PE.ArrivalTime >= StartInterval
    AND PE.ArrivalTime < EndInterval
    AND PE.DepartureTime < EndInterval
    AND st_dwithin(BL.the_geom, Center, Range)
```

To evaluate the proposed settings, we fixed the *center* of each spatiotemporal query in *Lonsdale Street*, then we varied spatial and temporal parameters. In particular, the range parameter varies among 100, 500 and 1000 ms, while the length of time period varies among 1,2,3,7,14 and 21 days. Note that, characteristics such as event density can impact performance, for example, when increasing the search radius and time interval or similarly when moving the search to denser areas. For this reason, the search area is centred on *Lonsdale Street*, which is located inside Melbourne's central business district, an area with a high parking events density and the first area where parking monitoring sensors have been installed.

Non-stationary dataset retrieval

To evaluate the performances of the mentioned systems, we designed a query (See Listing 2) representing the typical retrieval task that a railway maintenance decision-maker would perform. The query is designed to retrieve all the measurements made during trips happened between *StartInterval* to *EndInterval*, in a certain *Range* from a point of interest (*Center*). We employ such query to evaluate all the mentioned systems.

As Timescale offers the possibility to add a secondary partitioning dimension over a timestamp or an integer attribute, we partitioned the *GeneralData* table over the tot_pk_m attribute. To take advantage of partitioning while filtering the dataset with spatial operators, we extended the schema with the *GeoPartitioning* table. To evaluate such setting, we defined two queries (See Listings 3 and 4) expressing the same information need from that in Listing 2, such queries are meant to be used in a two-step filtering. Experimental settings with this type of filtering will be marked as *2step*. In particular, the query in Listing 3 performs the spatial filtering on the *GeoPartitioning* table, obtaining the parameters MaxPk and MinPk (i.e., maximum and minimum values referred to tot_pk_m). Then, these parameters are used in the query 4 to filter out the *GeneralData* table, in combination with time filtering parameters *StartInterval* and *EndInterval*, and with spatial filtering parameters *Range* and *Center*.

Listing 2 The parametric spatiotemporal query we used in our experiments on the non-stationary dataset, for the configurations R-tree and BRIN.

```
SELECT *
FROM GeneralData
WHERE time BETWEEN StartInterval AND EndInterval
      AND st_dwithin(geom, Center, Range)
```

Listing 3 The parametric Spatial filtering query we used in our experiments on the non-stationary dataset to retrieve spatial partitions, on configurations with secondary partitioning.

```
SELECT MIN(start_centimenr), MAX(end_centimeter)
FROM GeoPartitioning
WHERE st_dwithin(cluster, Center, Range)
```

Listing 4 The parametric spatiotemporal query we used in our experiments on the non-stationary dataset to retrieve data from the main table in combination with the query in Listing 3,on configurations with secondary partitioning.

```
SELECT *
FROM GeneralData
WHERE time BETWEEN StartInterval AND EndInterval
      AND tot_pk_m between MinPk AND MaxPk
      AND st_dwithin(geom, Center, Range)
```

During the experimental evaluation, the center of each query was fixed at the beginning of the monitored route. As for the stationary dataset, the spatial range varies among 100, 500 and 1000 ms (about half of the monitored route), while the time period size varies among 1,3,5,7,9,11,13,15 days. The process consists of the execution of 5 runs, for each combination of queries and parameters, and then of the evaluation of performances. Let us note that, for the two-step filtering experiments, we included in the retrieval time calculation also the time occurring between the execution of the two queries (see Listings 3 and Listing 4).

Evaluation metrics

As already done in some works on DBMS benchmarks (e.g.: [31]), our experimental protocol aims at the assessment of specific performance indicators, evaluated in some typical situations. In this case, the most used indicator is the *Retrieval Time* (see Table 3) expressed in seconds, which allows evaluating performances of typical retrieval tasks. Another key metric to assess the effectiveness of a Data Management solution is the *Disk Occupancy* (see Table 3) for each considered dataset, including also the space required for indexes. However, to allow the comparison of resources used by both datasets, we express the disk usage of indexes as a percentage of the raw data part stored in PostgreSQL.

Table 3 Selected Evaluation Metrics

Metrics	
Retrieval Time	Time to retrieve data and return records with respect to the baseline, i.e., PostGIS, (Percentage)
Disk Occupancy	Percentage of space required to store indexes with respect to data (Percentage)

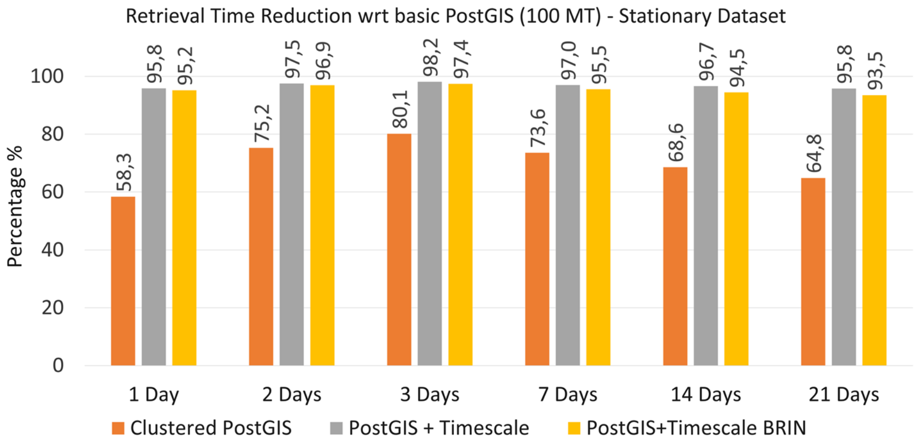


Fig. 5 Average retrieval time reduction for queries on stationary data with range of 100 ms, expressed in Percentage with respect to PostGIS configuration

4 Results and discussion

In Figs. 5, 6, 7 and in Table 4, are reported the results achieved on the stationary datasets, with the query reported in Listing 1. As we can see, the improvement in terms of performances due to the introduction of Timescale is remarkable. In fact, the setting *PostGIS + Timescale* requires on average 96% less time than the baseline (i.e., *PostGIS* setting) with each parameter combination. Considering the maximum query extension, i.e., with a range of 1000mt and 21 days, the *PostGIS + Timescale* setting took 98% less time with respect to the baseline. On the other hand, with filtering parameters set to 100mt range and a period of 1 day, the *PostGIS + Timescale* setting required 96% less time compared to the baseline.

It is worth noting that also the *Clustered PostGIS* setting generates a significant improvement over the baseline, highlighting that the physical ordering of the records has a deep impact on the execution time, as expected. On average, *Clustered PostGIS* reduces the time required to run the query by about 62%. Nevertheless, the performances of *PostGIS + Timescale* setting are by far better than *Clustered PostGIS*, so the optimization offered by Timescale makes an important difference in the presence of a prominent temporal dimension. Although both *PostGIS + Timescale* and *PostGIS + Timescale BRIN* configurations showed a comparable performance increment with respect to the baseline, the BRIN index induces a performance increment which is proportional to the spatial filter size.

In Fig. 8, we report indexes disk requirements of the three settings in terms of disk occupancy percentage with respect to the data portion. Let us remember that, when implemented in PostgreSQL, the stationary dataset occupies 40GB without indexes. As we can see, configurations involving Timescale need about 4% more than other settings to store temporal indexes. It is due to the table partitioning (chunks of one week) operated by Timescale. In

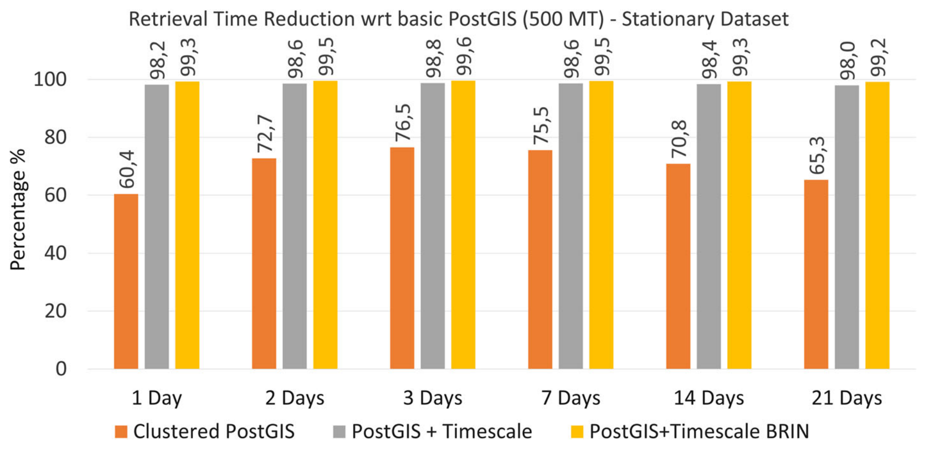


Fig. 6 Average retrieval time reduction for queries on stationary data with range of 500 ms, expressed in Percentage with respect to PostGIS configuration

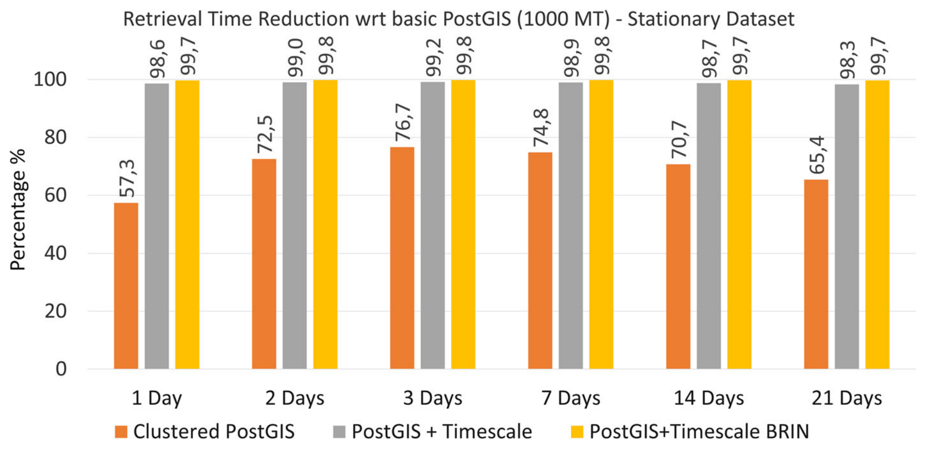


Fig. 7 Average retrieval time reduction for queries on stationary data with range of 1000 ms, expressed in Percentage with respect to PostGIS configuration

this case, the spatial index requires an unimportant percentage of space, which is subject to a reduction when switching from an R-tree to a BRIN index. On the other hand, the indexes used to filter the massive data streams based on the parking spot (i.e. Foreign Key Index in Fig. 8) require almost the same space as the temporal index.

Figures 9, 10, 11 and Table 5, report the most significant results achieved per combinations of system setting, parameters and queries (reported in Listings 2,3,4). The setting with the "2step" mark is referred to two-step filtering experiments made with queries in Listings 3,4. For brevity, we omitted the results achieved with settings *PostGIS + Timescale Secondary Partition (SP) R-tree* and *PostGIS + Timescale Secondary Partition (SP)*, also in the two-step filtering version, as they showed performances comparable or worse than BRIN based settings. Compared to the baseline, the introduction of Timescale let us to achieve an improvement of 17% in the worst (see *PostGIS + Timescale* configuration in Fig. 11) and 89% in the best case (see *PostGIS + Timescale BRIN* configuration in Fig. 10). Let us

Table 4 This table describes the achieved standard deviation (in seconds) during experiments on the stationary dataset

Time	Solution	100 MT	500 MT	1000 MT
1 day	Clustered PostGIS	0.05	0.01	0.02
1 day	PostGIS + Timescale	0.003	0.009	0.04
1 day	PostGIS + Timescale BRIN	0.003	0.008	0.05
2 day	Clustered PostGIS	0.01	0.03	0.01
2 day	PostGIS + Timescale	0.006	0.006	0.04
2 day	PostGIS + Timescale BRIN	0.005	0.0067	0.05
3 day	Clustered PostGIS	0.01	0.02	0.02
3 day	PostGIS + Timescale	0.008	0.026	0.24
3 day	PostGIS + Timescale BRIN	0.008	0.024	0.23
7 day	Clustered PostGIS	0.01	0.02	0.04
7 day	PostGIS + Timescale	0.019	0.023	0.078
7 day	PostGIS + Timescale BRIN	0.021	0.026	0.08
14 day	Clustered PostGIS	0.007	0.02	0.12
14 day	PostGIS + Timescale	0.015	0.035	0.049
14 day	PostGIS + Timescale BRIN	0.019	0.034	0.05
21 day	Clustered PostGIS	0.08	0.08	0.21
21 day	PostGIS + Timescale	0.015	0.058	0.062
21 day	PostGIS + Timescale BRIN	0.014	0.061	0.067

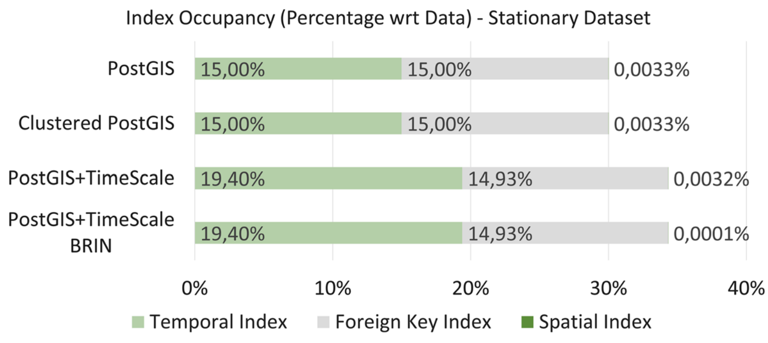


Fig. 8 Indexes' disk occupancy in percentage with respect to disk space required by the data component—stationary dataset

note that, with the not-stationary dataset, the *Clustered PostGIS* setting performed worse than any other, with a slight performance deterioration with respect to the baseline. After an analysis of the query execution plans, we faced that both *PostGIS* and *Clustered PostGIS* settings access data mainly through the spatial index, being considered the most selective one, even if the temporal dimension reflects the physical ordering of records, or if we are retrieving half of the dataset. On the other hand, since Timescale splits data in multiple fragments, it accesses data through the temporal index. Then it filters records based on the spatial condition or vice versa. Choosing the right spatial filtering approach is fundamental also with improvements achieved by Timescale. In fact, BRIN-based settings showed better performances in terms of retrieval time and disk usage, with respect to R-tree based R-tree

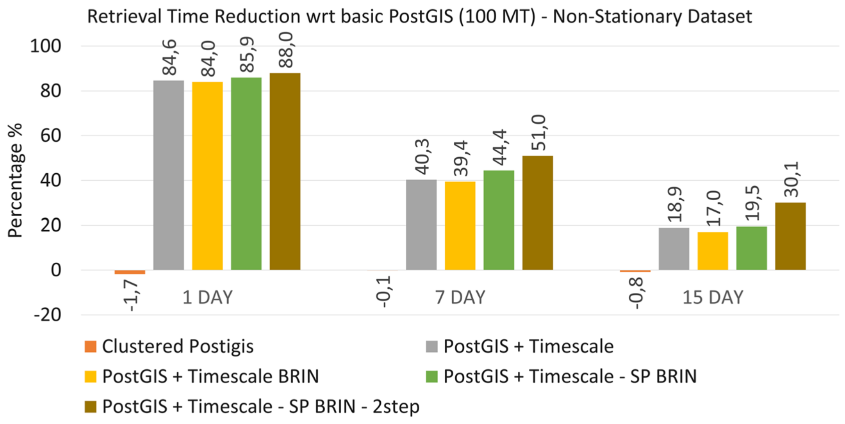


Fig. 9 Average retrieval time reduction for queries on not-stationary data with range of 100 ms, expressed in Percentage with respect to PostGIS configuration

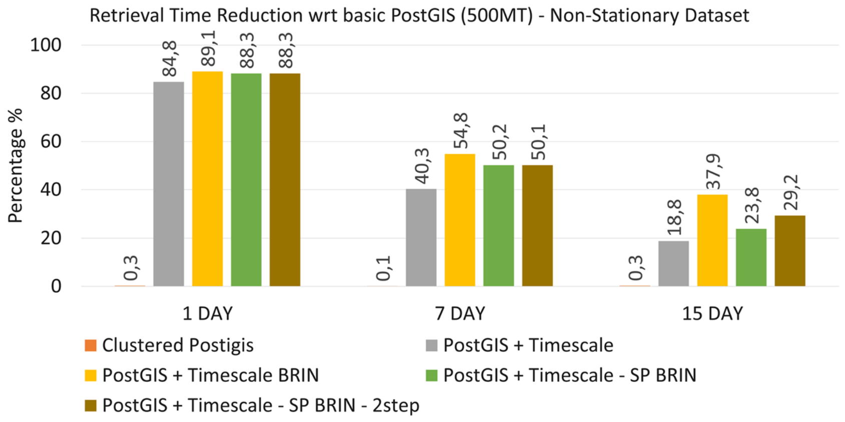


Fig. 10 Average retrieval time reduction for queries on not-stationary data with range of 500 ms, expressed in Percentage with respect to PostGIS configuration

index. Even with secondary partitioning, Timescale brings performance improvements with respect to the baseline setting, with an increment spanning from a minimum of about 20% to a maximum 88%.

In Fig. 12, we reported the space requirements for the not-stationary dataset. Let us note that in this case the *PostGIS + Timescale BRIN* setting, brings a considerable disk usage reduction with respect to *PostGIS* and *Clustered PostGIS* settings. As expected, the *PostGIS + Timescale BRIN* and *PostGIS + Timescale - SP BRIN* settings performed better than any other, as BRIN indexes are known for the low disk occupancy and complexity.

Clearly, physical ordering on the primary search dimension is crucial in improving performances of retrieval tasks over massive temporal data. Although PostgreSQL can order records physically, it is essential to note that sorting (i.e., clustering) data are a costful operation that must be performed after every update on the ordering attribute. In fact, sorting the stationary dataset in the *Clustered PostGIS* setting required about 79 min. On the other hand, Timescale acts like an instance of PostgreSQL in continuous sorting of the records, splitting

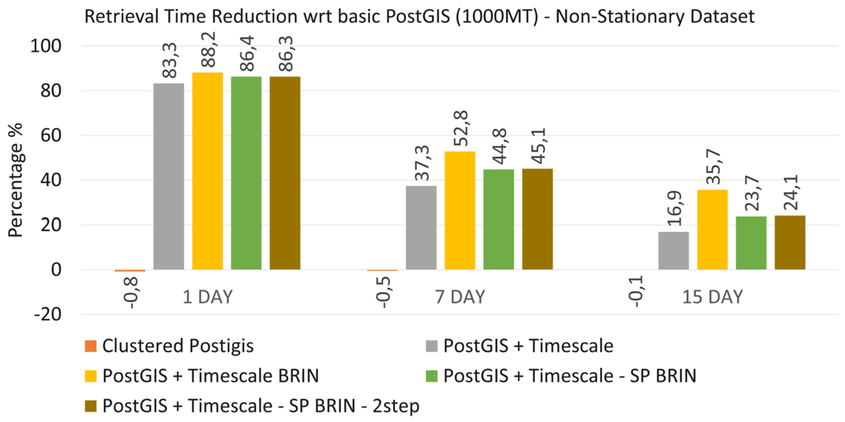


Fig. 11 Average retrieval time reduction for queries on not-stationary data with range of 1000 ms, expressed in Percentage with respect to PostGIS configuration

Table 5 This table describes the achieved standard deviation (in seconds) during experiments on the non-stationary dataset

Time	Solution	100 MT	500 MT	1000 MT
1 Day	Clustered PostGIS	0.11	0.45	0.87
1 Day	PostGIS + Timescale	0.03	0.12	0.21
1 Day	PostGIS + Timescale BRIN	0.06	0.14	0.15
1 Day	PostGIS + Timescale − SP BRIN	0.08	0.10	0.11
1 Day	PostGIS + Timescale − SP BRIN 2step	0.03	0.06	0.19
7 Day	Clustered PostGIS	0.28	0.35	1.34
7 Day	PostGIS + Timescale	0.11	0.72	0.39
7 Day	PostGIS + Timescale BRIN	0.15	0.46	2.17
7 Day	PostGIS + Timescale − SP BRIN	0.17	0.36	0.71
7 Day	PostGIS + Timescale − SP BRIN 2step	0.06	0.32	1.21
15 Day	Clustered PostGIS	0.49	0.57	0.66
15 Day	PostGIS + Timescale	0.13	1.01	3.07
15 Day	PostGIS + Timescale BRIN	0.60	1.13	2.17
15 Day	PostGIS + Timescale − SP BRIN	0.42	0.64	2.4
15 Day	PostGIS + Timescale − SP BRIN 2step	0.27	0.98	2.53

the overhead of the ordering phase in the ingestion phase on smaller time-ordered tables. This way, Timescale guarantees a collection of physically time-ordered tables, rapidly searchable and able to fit working memory completely. As a minor drawback, Timescale requires to set some parameters (i.e., the *chunk_time_interval*) to achieve and maintain such performances without excessive disk usage increment.

Therefore, Timescale demonstrated to be effective in managing both stationary and not-stationary spatial data. It shows better performances compared to both *PostGIS* and *Clustered PostGIS* settings with any parameters combination. Furthermore, Timescale also shows its effectiveness with state-of-the-art spatial indexes on secondary search dimensions by significantly increasing retrieval performances.

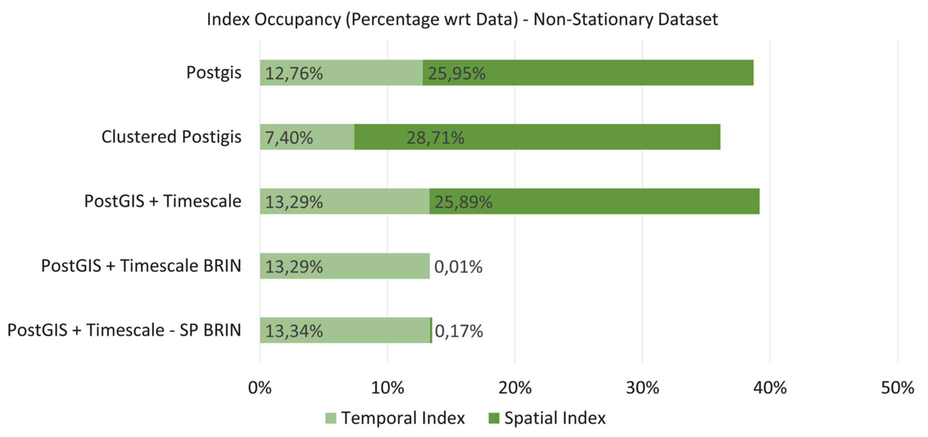


Fig. 12 Indexes' disk occupancy in percentage with respect to disk space required by the data component-not stationary dataset

5 Threats to validity

In this section, we report on threats about this first pilot experience, that may have influenced results and their generalizability.

Threats to internal validity These threats regard experimental-related factors that could limit the confidence in results presented.

The choice to perform five executions for each possible query represents a trade-off between the time required to execute each query and the large number of parametric configurations for each query in each use case considered for each storage solution. However, to mitigate the effects of this choice, we performed a complete restart of the workstation after every single execution (see Sect. 3.3), thus obtaining similar execution conditions and removing the effects of caching policies. This study used a single hardware optimization for all storage settings considered. On the one hand, this leaves open further optimization possibilities that depend on different hardware configurations. On the other hand, this allowed us to compare the solutions considered in the same operating conditions, exploiting in the same way available hardware resources.

Threats to external validity These threats limit the generalizability of the experimental results. We considered two different types of spatiotemporal data. The respective investigated configurations, as well as the employed datasets, are based on real use-cases and, therefore, may show similar biases to some extent. As we considered two very specific types of spatiotemporal data, in which the geographical component is fixed (like in the stationary dataset) or, in any case, is constrained to specific geographical areas (like rail sections in the non-stationary dataset), the results hold for similar scenario. Thus, different forms of spatial data, such as trajectories, may exhibit different behaviours. Furthermore, we considered a generic query aimed only at massive data points retrieval based on filters which vary over massive dimensions (i.e., spatial and temporal). However, under different analytic conditions which require data aggregation or ad hoc filtering (e.g., subqueries), system performance might be affected differently.

In this study, we considered one hardware configuration, along with some ad hoc optimizations. This can limit the generalizability of our findings on configurations with different CPU architectures, reduced working memory (RAM) availability or multi-node configurations.

In addition, we focused on the performance improvements, in terms of running time, over a baseline configuration rather than measuring actual resource usage (e.g. CPU/RAM/Disk usage). This allowed us to compare the performance of considered storage solutions from a hardware-agnostic point of view, while leading to a somewhat uncertain performance comparison in terms of physical resources required.

6 Conclusions

The way we see cities, factories, houses and many other structures is undergoing a significant transformation due to the integration of state-of-the-art ICT technologies like IoT, Cloud, Big Data, etc. To succeed in such transformation, the capacity to manage and analyze data effectively is of paramount importance.

In this work, we reported on an empirical experience in handling massive spatiotemporal datasets with both stationary and non-stationary spatial components. The stationary dataset covers about seven years of on-street parking availability from the municipality of Melbourne (AU). The non-stationary dataset covers about a month of rail monitoring through a train-mounted system. In particular, we compared an established setting for spatial data management, i.e., PostgreSQL and PostGIS, with an optimized version (i.e., clustered) and with a third method based on an advanced Time Series Management System (i.e., Timescale), providing some techniques optimized to manage massive temporal data in the modern technological context depicted by IoT. Furthermore, we assessed the effectiveness of two state-of-the-art spatial indexing techniques in the presence of massive, non-stationary data. Then we compared them with a simple approach aimed at improving both performances and resource usage on top of a TSMS.

The empirical experiment was conducted with complex spatiotemporal queries, specifically designed for each dataset, involving multiple location-based and temporal parameters. The results show that Timescale can outperform by far any other option. On the other hand, it requires more disk space and some additional parameters to be set to create the Database. However, it is possible to combine Timescale's characteristics with simple partitioning approaches to mitigate resource usage and improve performances. It is worth noting that if the primary search dimension is also the physical ordering one, the possibility offered by PostgreSQL to specify the sorting attribute impacts the results in a dramatic way.

As future evolution, we are interested in evaluating the effectiveness of the proposed solution using different online data sources, eventually combined with streams of other incoming data. Another interesting aspect would be the comparison with DBMS based on non-relational data models, like the graph-based ones, which also support spatial primitives.

Acknowledgements This work is funded by the Italian Ministry for Economic Development, in the framework of the OCTOPUS project (Funding Program Fabbrica Intelligente - #B61B20000300005)

Author Contributions VNV wrote the main manuscript text. Performed experiments on stationary data. He contributed to results analysis. SM contributed to Sects. 1 and 2. He contributed to data analysis. He contributed to the definition of constraints regarding the stationary dataset. Reviewed the manuscript. AP contributed to Sects. 1 and 2. He contributed to the analysis of the results and helped in analyzing data. Reviewed the manuscript. MR made part of the experiments regarding the non-stationary data; He contributed to data analysis and Sect. 3.2. EB represents the industrial partner; he contributed to Sects. 3.1 and 3.3 for the parts concerning the non-stationary dataset. Reviewed the manuscript.

Funding Open access funding provided by Università degli Studi di Napoli Federico II within the CRUI-CARE Agreement.

Data availability The non-stationary dataset analysed during the current study is not publicly available because it is part of an ongoing industrial project. The stationary dataset analysed during the current study is available from the municipality of Melbourne [52].

Declarations

Conflict of interest The authors declare no competing interests.

References

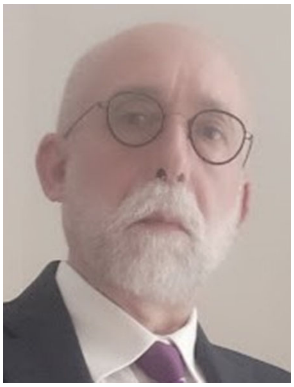
1. Ahmed E, Yaqoob I, Gani A, Imran M, Guizani M (2016) Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges. IEEE Wirel Commun 23(5):10–16
2. Habibzadeh H, Qin Z, Soyata T, Kantarci B (2017) Large-scale distributed dedicated-and non-dedicated smart city sensing systems. IEEE Sens J 17(23):7649–7658
3. Di Martino S, Vitale VN, Bock F (2019) Investigating the influence of on-street parking guidance strategies on urban mobility. In: 2019 6th international conference on models and technologies for intelligent transportation systems (MT-ITS). IEEE, pp. 1–6
4. Bhattarai BP, Paudyal S, Luo Y, Mohanpurkar M, Cheung K, Tonkoski R, Hovsapian R, Myers KS, Zhang R, Hao P et al (2019) Big data analytics in smart grids: state-of-the-art, challenges, opportunities, and future directions. IET Smart Grid 2(2):141–154
5. Hermann M, Pentek T, Otto B (2016) Design principles for industrie 4.0 scenarios. In: 2016 49th Hawaii international conference on system sciences (HICSS). IEEE, pp. 3928–3937
6. Carvalho TP, Soares FAAMN, Vita R, da Francisco RP, Basto JP, Alcalá SGS (2019) A systematic literature review of machine learning methods applied to predictive maintenance. Comput Ind Eng 137:106024
7. Henning K, Wolfgang W, Johannes H (2013) Recommendations for implementing the strategic initiative industrie 4.0. Final report of the Industrie 4: 82
8. Kim TH, Ramos C, Mohammed S (2017) Smart city and iot
9. Das S (1994) Time series analysis. Princeton University Press, Princeton
10. Villalobos K, Ramírez-Durán VJ, Diez B, Blanco JM, Goñi A, Illarramendi A (2020) A three level hierarchical architecture for an efficient storage of industry 4.0 data. Comput Ind 121:103257
11. Chaudhry N, Yousaf MM, Taimoor Khan M (2020) Indexing of real time geospatial data by iot enabled devices: opportunities, challenges and design considerations. J Ambient Intell Smart Environ 1:32. https://doi.org/10.3233/AIS-200565
12. Vatsavai RR , Ganguly A, Chandola V, Stefanidis A, Klasky S, Shekhar S (2012) Spatiotemporal data mining in the era of big spatial data: algorithms and applications. In: Proceedings of the 1st ACM SIGSPATIAL international workshop on analytics for big geospatial data, pp. 1–10
13. Lee J-G, Kang M (2015) Geospatial big data: challenges and opportunities. Big Data Res 2(2):74–81
14. Guo D, Onstein E (2020) State-of-the-art geospatial information processing in NoSQL databases. ISPRS Int J Geo Inf 9(5):331
15. Yu J, Sarwat M (2017) Indexing the pickup and drop-off locations of nyc taxi trips in postgresql–lessons from the road. In: International symposium on spatial and temporal databases. Springer, pp. 145–162
16. Yang C, Clarke K, Shekhar S, Tao CV (2020) Big spatiotemporal data analytics: a research and innovation frontier. Int J Geogr Inf Sci 34(6):1075–1088
17. Jawarneh A, Mashhour I, Bellavista P, Corradi A, Foschini L, Montanari R (2020) Big spatial data management for the internet of things: a survey. J Netw Syst Manage 28(4):990–1035
18. Sisinni E, Saifullah A, Han S, Jennehag U, Gidlund M (2018) Industrial internet of things: challenges, opportunities, and directions. IEEE Trans Industr Inf 14(11):4724–4734

19. Pradeep P, Krishnamoorthy S, Vasilakos AV (2021) A holistic approach to a context-aware IoT ecosystem with adaptive ubiquitous middleware. Pervasive Mob Comput 72:101342
20. Jensen SK, Pedersen TB, Thomsen C (2017) Time series management systems: a survey. IEEE Trans Knowl Data Eng 29(11):2581–2600
21. Naqvi SNZ, Yfantidou S, Zimányi E (2017) Time series databases and influxdb. Université Libre de Bruxelles, Studienarbeit
22. Baralis E, Valle AD, Garza P, Rossi C, Scullino F (2017) SQL versus NoSQL databases for geospatial applications. In: 2017 IEEE international conference on big data (Big Data). IEEE, pp. 3388–3397
23. Zhang L, Yi J (2010) Management methods of spatial data based on PostGIS. In: 2010 second pacific-asia conference on circuits, communications and system. IEEE, vol. 1, pp. 410–413
24. Timescale (2019) Timescale simple, scalable SQL for time-series and IoT. https://www.timescale.com/. Accessed 15 Dec 2019
25. PostgreSQL. PostgreSQL the most advanced open-source object relational database
26. Pandey V, Kipf A, Neumann T, Kemper A (2018) How good are modern spatial analytics systems? Proc VLDB Endow 11(11):1661–1673
27. Ali ME, Eusuf SS , Islam KA (2020) An efficient index for contact tracing query in a large spatio-temporal database. arXiv preprint arXiv:2006.12812
28. Graser A, Dragaschnig M, Koller H (2021) Exploratory analysis of massive movement data. In: Handbook of big geospatial data, p. 285
29. Mahmood AR, Punni S, Aref WG (2019) Spatio-temporal access methods: a survey (2010–2017). GeoInformatica 23(1):1–36
30. Di Martino S, Vitale VN (2020) Massive spatio-temporal mobility data: an empirical experience on data management techniques. In: International symposium on web and wireless geographical information systems. Springer, pp. 41–54
31. Di Martino S, Peron A, Riccabone A, Vitale VN (2021) Benchmarking management techniques for massive IIoT time series in a fog architecture. Int J Grid Util Comput 12(2):113–125
32. Influx Data (2023) Influxdb. https://docs.influxdata.com/influxdb/cloud/query-data/flux/geo/#shape-data-to-work-with-the-geo-package Accessed 06 Apr 2023
33. S2geometry community (2023) S2geometry. https://s2geometry.io/ Accessed 06 April 2023
34. Zhang D, Wang Y, Liu Z, Dai S (2019) Improving NoSQL storage schema based on Z-curve for spatial vector data. IEEE Access 7:78817–78829
35. Makris A, Tserpes K, Anagnostopoulos D, Nikolaidou M, de Macedo JAF (2019) Database system comparison based on spatiotemporal functionality. In: Proceedings of the 23rd international database applications & engineering symposium. ACM, pp. 21
36. PostGIS (2019) PostGIS spatial database extender for postgresql. https://postgis.net/docs/. Accessed 15 Dec 2019
37. Kaur K, Rani R (2015) Managing data in healthcare information systems: many models, one solution. Computer 48(3):52–59
38. Liu X, Nielsen PS (2016) A hybrid ICT-solution for smart meter data analytics. Energy 115:1710–1722
39. Gilbert S, Lynch N (2012) Perspectives on the cap theorem. Computer 45(2):30–36
40. Open Geospatial Consortium et al. (2010) Opengis implementation specification for geographic information-simple feature access-part 2: SQL option. OpenGIS Implementation Standard
41. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. ACM, vol. 14
42. PostgreSQL. BRIN index
43. Michel O, Sonchack J, Keller E, Smith JM (2019) PIQ: persistent interactive queries for network security analytics. In: Proceedings of the ACM international workshop on security in software defined networks & network function virtualization. ACM, pp. 17–22
44. Bayer R, McCreight E (2002) Organization and maintenance of large ordered indexes. In: Software pioneers. Springer, pp. 245–262
45. Lehman PL et al (1981) Efficient locking for concurrent operations on b-trees. ACM Trans Database Syst (TODS) 6(4):650–670
46. Corbett JC, Dean J, Epstein M, Fikes A, Frost C, Furman JJ, Ghemawat S, Gubarev A, Heiser C, Hochschild P et al (2013) Spanner: Google's globally distributed database. ACM Trans Comput Syst (TOCS) 31(3):1–22
47. Papadias D, Tao Y, Kanis P, Zhang J (2002) Indexing spatio-temporal data warehouses. In: Proceedings 18th international conference on data engineering. IEEE, pp. 166–175
48. Borodin A, Mirvoda S, Kulikov I, Porshnev S (2017) Optimization of memory operations in generalized search trees of PostgreSQL. In: International conference: beyond databases, architectures and structures. Springer, pp. 224–232

49. Jin J, Gubbi J, Marusic S, Palaniswami M (2014) An information framework for creating a smart city through internet of things. IEEE Internet Things J 1(2):112–121
50. Kwoczek S, Di Martino S, Rustemeyer T, Nejdl W (2015) An architecture to process massive vehicular traffic data. In: 2015 10th international conference on P2P, parallel, grid, cloud and internet computing (3PGCIC). IEEE, pp. 515–520
51. Sun Y, Song H, Jara AJ, Bie R (2016) Internet of things and big data analytics for smart and connected communities. IEEE Access 4:766–773
52. Victoria State Government (2014) City of melbourne. melbourne parking sensor. https://www.melbourne.vic.gov.au/SiteCollectionDocuments/parking-technology-map.pdf. Accessed 15 Dec 2019
53. SFMTA (2014) SFPark: putting theory into practice. Pilot project summary and lessons learned. Accessed 24 June 2016
54. Du R, Santi P, Xiao M, Vasilakos AV, Fischione C (2018) The sensable city: a survey on the deployment and management for smart city monitoring. IEEE Commun Surv Tutor 21(2):1533–1560
55. McMahon P, Zhang T, Dwight R (2020) Requirements for big data adoption for railway asset management. IEEE Access 8:15543–15564
56. Hodge VJ, O'Keefe S, Weeks M, Moulds A (2014) Wireless sensor networks for condition monitoring in the railway industry: a survey. IEEE Trans Intell Transp Syst 16(3):1088–1106
57. Cong D, Susom D, Pradeep K, Tzuyang Yu, Xingwei W (2020) A review of railway infrastructure monitoring using fiber optic sensors. Sens Actuators A Phys 303:111728
58. Li C, Luo S, Cole C, Spiryagin M (2017) An overview: modern techniques for railway vehicle on-board health monitoring systems. Veh Syst Dyn 55(7):1045–1070
59. Alahakoon S, Sun YQ, Spiryagin M, Cole C (2018) Rail flaw detection technologies for safer, reliable transportation: a review. J Dyn Syst Meas Control 140(2):020801
60. Bock F, Di Martino S, Origlia A (2019) Smart parking: using a crowd of taxis to sense on-street parking space availability. IEEE Trans Intell Transp Syst 21(2):496–508
61. MS Kester, M Athanassoulis, S Idreos (2017) Access path selection in main-memory optimized data systems: should i scan or should i probe? In: Proceedings of the 2017 ACM international conference on management of data, pp. 715–730
62. Liu KS, Gao J, Wu X, Lin S (2018) On-street parking guidance with real-time sensing data for smart cities. In: 2018 15th annual IEEE international conference on sensing, communication, and networking (SECON). IEEE, pp. 1–9
63. Shao W, Salim FD, Gu T, Dinh NT, Chan J (2017) Traveling officer problem: managing car parking violations efficiently using sensor data. IEEE Internet Things J 5(2):802–810
64. Tsunashima H (2019) Condition monitoring of railway tracks from car-body vibration using a machine learning technique. Appl Sci 9(13):2734
65. Maes K, Van Meerbeeck L, Reynders EPB, Lombaert G (2022) Validation of vibration-based structural health monitoring on retrofitted railway bridge kw51. Mech Syst Signal Process 165:108380

Vincenzo Norman Vitale got his Ph.D. on Data Management in Cloud-Fog Environments at the University of Naples Federico II. He is currently an Assistant Professor at the Department of Electrical Engineering and Information Technology of the University of Naples Federico II, Italy. His research interests include natural language processing, machine learning, automatic speech processing, big data management and cloud-edge continuum.

Sergio Di Martino is Full Professor in Computer Science at the Department of Electrical Engineering and Information Technology of the University of Naples Federico II, Italy, where he is also the co-chair of the Knowledge Management and Engineering (Knome) Lab. He has published more than 130 papers in journals, conference proceedings and books. His research interests focus on management and analytics of complex and massive datasets, software engineering and software architectures.



Andriano Peron received the MS in Computer Science from the University of Udine and the PhD in Computer Science from the University of Pisa. He is currently director of the BS Program of Artificial Intelligence and Data Analytics the University of Trieste. His research interests include formal specification and verification of safety critical systems, model checking, logics for computer science, software testing, big data management.

Massimiliano Russo is currently a Master's student in Computer Science at the University of Naples Federico II. He got his Bachelor's degree in Computer Science in 2020 at the University of Naples Federico II. His research interests primarily revolve around Environmental and Geospatial Applications.

Ermanno Battista achieved a Ph.D. focusing on Software Engineering and Security at University of Naples Federico II. He completed his education with a period at the George Mason University, USA, where he proposed novel solutions for the protection of cyber-physical systems. With a strong technology background and passion for innovation, he started an entrepreneurship path.