## DANSK REFERAT

AIS data er originalt tiltænkt at hjælpe skibe med at navigere mellem andre skibe og forhindringer til havs. Historisk AIS data giver et data grundlag, som kan bruges til forskellige typer af analyse, som fx at finde mønstre i skibs trafik.

Eftersom AIS data består af både manuel indtastning og automatisk sensor data, er der et stort behov for rensning inden nogen nyttefuld analyse kan udføres. Ligeledes er det nødvendigt at omstrukturere dataen fra temporale punkter til linjestrenge af skibs-ture, for at undgå tung konstruktion af linjestrenge under forespørgelser.

DIPAAL er en komplet platform der både renser, transformerer og indlæser den rå AIS data ind i et Data Warehouse (DW). DW'et er designet til at understøtte effektive distribuerede efterspørgsler. Denne artikel præsenterer en udvidelse af DIPAAL. Eftersom det ikke kan forventes at alle brugere af DIPAAL er DW eksperter, er der udviklet en API der muliggør meget analyse uden kendskab til det underliggende DW design.

DIPAAL introducerer en celle repræsentation, som simplificerer en linjestrengs-tur til et antal af celler som turen har krydset. Ved at konvertere turene tilbage til en punkt-baseret repræsentation, er det både hurtigt at søge i og aggregere data. For at undgå at efterspørgsler på store områder er unødvendigt tunge, men samtidig beholde en fin granularitet, understøtter DIPAAL fire granulariteter af celler med sidelængderne 50m, 200m, 1000m og 5000m.

Ud fra celle repræsentationen udregner DIPAAL heatmaps ved hjælp af spatielle aggregeringer som gemmes som rasters. Disse rasters kan derefter kombineres på tværs af tid, skibs type, mobil type, og om skibet var i bevægelse eller ej. DIPAAL udregner i øjeblikket fem typer af heatmaps, hvoraf en måler hvor lang tid skibe bruger i et område.

For at kunne distribuere disse spatielle aggregering, så er celle repræsentationen og heatmaps distribueret spatielt i et balanceret kd-træ.

Denne artikel præsenterer en omfattende evaluering af DIPAAL, som måler ikke blot hvor hurtigt en række af efterspørgsler kan udføres, men måler også hvor meget hurtigere efterspørgslerne udføres, når antallet af maskiner udvides fra en til fem. Resultaterne viser både fremragende resultater i forhold til aktuelt køretid, hvor selv de mest krævende efterspørgsler bliver færdige på under to minutter, men også at skalering fra en til fem giver en forbedring på køretiden mellem 354% og 1164% på store efterspørgsler. Dette viser at den spatielle opdelings metode er god til at udnytte de ekstra maskiner.

Med den spatielle distribuering er det forventeligt at data over tid vil blive ubalanceret. De spatielle opdelinger er baseret på data fra 2021. Evaluering af den spatielle distribuering viser at sekventielle år ikke har stor forskel i trafik mønstre, men over en periode på 10 år bliver de spatielle opdelinger lidt ubalanceret.

DIPAAL har i skrivende stund indlæst 5 år af dansk AIS data. Dette involverer 414 millioner tilbagelagte kilometer over 13.9 millioner linjestrenge af 57 238 skibe, som sammenlagt har været i bevægelse i 1 111 143 døgn.

## PREFACE

This paper is a continuation of the original DIPAAL paper [1], which explored creating a distributed platform for loading AIS data and performing analytics on the loaded data. The platform was implemented with a modular Python-based ETL process, and a PostgreSQL [2] data warehouse, using the extensions Citus [3] for distribution, PostGIS [4] for spatial and raster functionality, and MobilityDB [5] for spatio-temporal and temporal functionality. Furthermore, the original DIPAAL paper [1] introduced a cell representation which improves the query runtime of small spatial areas. This paper presents further contributions to this platform, where some contributions are fixes and improvements, while others are new features and extensive design changes.

For this reason, some of the content of this paper overlaps with the original DIPAAL paper [1]. To make it clear what is changed, each subsection is designated as either "minor changes", "major changes", or "new". Minor changes refer to reformulations, and small corrections. Major changes refer to subsections where the substance of the subsection differs from the original paper and thus include conceptual differences. New subsections are completely new sections or rewritten. Subsections with no markings indicate no changes.

| Section | Subsection | Change |
|---|---|---|
| Abstract | | Major Changes |
| Introduction | | Major Changes |
| Architecture | | Minor Changes |
| ETL Design | File Download Module | Minor Changes |
| | Data Cleaning Module | Minor Changes |
| | Trajectory Construction Module | Major Changes |
| | Bulk Inserter Module | Minor Changes |
| | Rollup Module | Major Changes |
| Data Warehouse Design | | Major Changes |
| Spatial Distribution | | New |
| API Design | | New |
| Data Preparation | | Minor Changes |
| Implementation | Line Simplification | Minor Changes |
| | Rollup | Major Changes |
| | Spatial Relationship | New |
| | Indices | Minor Changes |
| | API Implementation | New |
| | Reducing Storage Cost | New |
| Evaluation | | New |
| Related Work | | Major Changes |
| Conclusion | | New |
| Future Work | | New |
| Acknowledgements | | Major Changes |
| Appendix | Danish Geodata Agency | Major Changes |
| | Sample API Queries | New |
| | Benchmark Plans | New |
| | Relation Statistics | New |

TABLE I: Overview of the changes of specific sections and subsections.

The section Data Preparation is a new section, but consists entirely of re-used work from the implementation section in the original DIPAAL paper [1], with minor changes.

The AI-based search engine Phind[1] has been used to facilitate the development of DIPAAL in regards to Python, LaTeX , and SQL code by serving as an advanced search engine. None of the text in this paper is written by or facilitated with AI. Likewise, none of the published source code is written by Phind. GitHub Copilot[2] has been used during development, where it has been used to suggest and auto complete code snippets.

All aerial imagery used in the figures of this paper are supplied by Bing Maps[3].

[1]https://phind.com

[2]https://github.com/features/copilot

[3]https://learn.microsoft.com/en-us/bingmaps/rest-services/imagery/

# DIPAAL: DIstributed PostgreSQL-based AIS Analytics and Loading

Alex Skov Klitgaard
*Department of Computer Science*
*Aalborg University*
Aalborg, Denmark
aklitg13@student.aau.dk

Lau Ernebjerg Josefsen
*Department of Computer Science*
*Aalborg University*
Aalborg, Denmark
ljosef18@student.aau.dk

Mikael Vind Mikkelsen
*Department of Computer Science*
*Aalborg University*
Aalborg, Denmark
mimikk21@student.aau.dk

***Abstract*—AIS data show promise for analytical purposes, but as the data are not intended for analysis, the data need to be cleaned, processed, and stored before being usable. This paper presents an extension of DIPAAL, a system consisting of an efficient and modular ETL process for loading AIS data, as well as a distributed data warehouse storing the trajectories of ships. A spatially distributed data warehouse, with granularized cell and heatmap representations, is designed, developed, and evaluated. At the time of writing, DIPAAL stores 414 million kilometres of ship trajectories and more than 10 billion rows in the largest relation. It is found that the introduced granularized cell representation resolved out-of-memory errors of previous work, while improving the runtime of up to 324% compared to a trajectory-based query. It is also found that the spatially divided shards enable a consistently good scale up for both cell and heatmap analytics in large areas, ranging between 354% to 1164% with a 5x increase in workers. Lastly, it is found that the spatial divisions become slightly skewed over time, as traffic patterns evolve.**

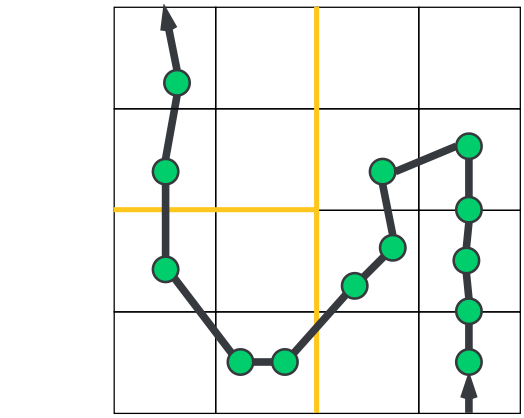
***Index Terms*—Spatio-temporal, AIS, Trajectory, Distributed, ETL, RDBMS, Moving object, Cell representation, Heatmaps, Spatial partitioning, Spatial distribution, PostgreSQL, PostGIS, MobilityDB, Citus**

## I. Introduction

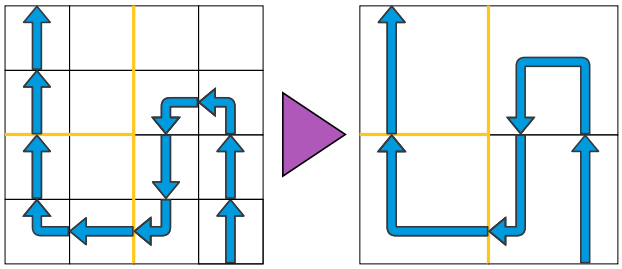
Although the automatic identification system (AIS) was introduced as a tool for automatic identification of vessels at sea, the data collected from it shows promise for analytics [6]. The explosions at the Nord Stream gas lines on the 26th of September 2022 [7] are examples of where analytics of AIS data shows promise, as it can be used to determine which ships were near the gas lines within a specific area and temporal span.

Although AIS data show great promise for analytics, sufficient care must be taken to overcome its inherent limitations [6]. One of the inherent limitations of AIS data are that it is dirty, as the protocol is originally designed to help increase the safety at sea by automatically identifying vessels. Therefore, before using AIS data for analytics, it is necessary to clean the data based on knowledge about the ship domain. Danish AIS data have been stored for more than a decade resulting in a large amount of available AIS data for analytics, such as traffic pattern mining [8] and minimum draught maps [9].

The large volume of available AIS data makes storing and processing the data on a single machine infeasible. Therefore, systems proposed for efficient handling of trajectory data, such as [10, 11, 1], use a distributed storage engine to store the data across a cluster of workers.



(a) Trajectory representation.



(b) Cell representation with rollup.

Fig. 1: Representations of AIS data in DIPAAL with spatial divisions.

The original DIPAAL platform [1] consisted of a modular ETL process responsible for cleaning and transforming the AIS data before loading the data into a PostgreSQL-based Data Warehouse (DW). The DW of the original DIPAAL platform introduced a cell representation, transforming complex trajectories into sets of simple geometries. This paper works toward extending the platform created in the original DIPAAL paper [1], while rectifying the shortcomings of the cell representation.

DIPAAL stores representations of the AIS data in the form of trajectories and cells, which are depicted in Figure 1. Trajectories are chronologically ordered sets of AIS points, as shown in Figure 1a, where the green circles are individual AIS data points. Whereas, cells are spatial areas that aggregate trajectory information, as shown in Figure 1b. The cell representation is updated to now consist of a hierarchy that stores the resulting cells at multiple granularities instead of solely consisting of 50m cells. This hierarchy alleviates the out-of-memory conditions and slow query runtimes experienced in the evaluation of the original DIPAAL paper [1].

The Citus PostgreSQL extension enables distribution by sharding the data in the DIPAAL DW across a cluster of workers. To enable spatial aggregations on the cell and heatmap representations, a spatial distribution approach is used. This approach creates spatial divisions which divides the cell and heatmap data across all available workers. These spatial divisions are created using a kd-tree approach and is based on an entire year's worth of AIS data. An example of spatial divisions are shown as the yellow lines in Figure 1 which shows a large division on the right and two smaller divisions on the left. It is evident from Figures 1a and 1b that the cell representation is more suited for spatial distribution than the trajectory representation, as trajectories can span multiple divisions, whereas cells fit within a single division.

To increase the analytical capabilities of DIPAAL both the design and implementation of the ETL process are extended to support querying heatmaps in the form of rasters. Adding support for heatmaps as an additional analytical capability originates from discussions with domain experts, see Appendix A.

The API proposed in the original DIPAAL paper [1] is implemented. It is implemented both to provide visualisation for the added heatmap functionality, as well as make parameterisable and pre-optimised queries available for non-DW users.

This paper explores how a kd-tree-based spatial distribution approach impacts the query runtime of different query types and the utilisation of the underlying hardware. Additionally, how well the spatial distribution approach scales when going from a one- to five-worker setup is evaluated for queries with areas of different sizes.

The storage requirements of the multiple representations stored in DIPAAL and the viability of lazily calculating the fine-grained representations are explored.

The contributions of this paper are:

- Implementation of a spatial distribution approach that reduces query runtime on the cell representation by ensuring spatial data locality on the workers and enabling push down of spatial aggregation to the workers.
- Extended analytical capabilities in the form of on-demand distributed heatmap creation.
- Extensive evaluation of the extended DIPAAL platform which offers insights into the query runtime performance of the platform and how well it scales on one- and five-worker setups.
- Show feasibility of the design of DIPAAL by loading, storing, and performing analytics on five years' worth of AIS data.

At the time of writing, 5 years are loaded into the DW, consisting of 414 million kilometres of ship trajectories between 13.9 million trajectories over a duration of 1 111 143 days. A total of 57 238 unique MMSIs have been observed. The largest relation contains 10 billion rows.

The rest of the paper is structured as follows; the definitions used and the architecture of DIPAAL are described in Section II. Section III explain the design behind the ETL process used to load data into the DIPAAL DW. The structure and design choices behind the DIPAAL DW are illustrated in Section IV. Section V discuss possible approaches to create spatial divisions for the DIPAAL DW. The principles and design choices behind the DIPAAL API are listed in Section VI. Implementation details of the DIPAAL platform are described in Sections VII and VIII. Section VII covers the interesting parts of the data preparation implementation for the ETL process, and Section VIII covers the parts related to insertion of data and methods to increase the efficiency of the DW. Section IX evaluates the efficiency of the DIPAAL DW. Related work are explained in Section X. Finally, Section XI concludes the paper and Section XII describes possible future work for DIPAAL.

## II. Architecture

In order to describe the architecture of DIPAAL, the key definitions are given.

### A. Definitions

*Definition 1: Point*: *A point* $p = \{t, lng, lat, MMSI, ais\}$*, where* $t$ *is the timestamp at which the point was created, lng and lat are abbreviations of the longitude and latitude coordinates. MMSI refers to the Maritime Mobile Service Identity (MMSI) of the transmitting ship, and* $ais$ *are additional optional AIS attributes.*

MMSI is a 9-digit identification number that is mandatory to use for identification [12]. The additional optional attributes $ais$, in Definition 1, are the attributes contained within the AIS message data related to the ship. Examples of additional attributes are the speed over ground (SOG), destination, heading, and navigational status, as explained by Bereta, Chatzikokolakis, and Zissis [12].

*Definition 2: Trajectory*: *A trajectory* $T$ *is a sequence of points sorted on their timestamp in ascending order* $T = \{p_1, p_2, ..., p_n\}$*, where* $n \geq 2 \wedge p_1.t < p_2.t < ... < p_n.t$.

A trajectory must consist of at least two points in DIPAAL as otherwise it is incapable of expressing movement over time. The points of the trajectory must be strictly temporally ordered.

Definitions 1 and 2 are data source independent and can be applied to any AIS dataset.

As both the sharding and partitioning techniques are used in DIPAAL, it is necessary to clearly define the difference between sharding and partitioning.

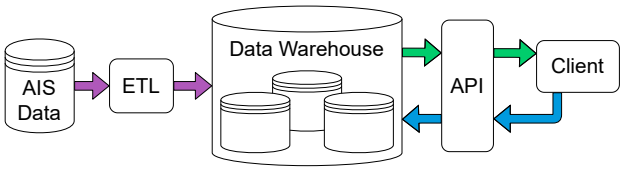



Fig. 2: The architecture of the DIPAAL platform.

*Definition 3: Shard*: *Relations are divided into shards to horizontally scale database queries. Each shard is a subset of the parent relation and may be located on different workers. The process of dividing a relation into shards is called sharding [13, sec. 4.2].*

*Definition 4: Partition*: *A partition, or the process of partitioning, refers to the process of splitting a logical PostgreSQL relation into multiple physical smaller PostgreSQL relations on the same worker [14, sec. 5.11].*

### B. Platform

The architecture of DIPAAL is seen in Figure 2, where purple arrows depict data processing, green arrows depict requests for data, and blue arrows depict the responses to the requested data. Cylinders indicate data storage, while boxes are software components.

AIS data are noisy and must therefore undergo data cleaning as part of the ETL process, to improve data quality [6]. The ETL process is modelled as a pipeline, which is further described in Section III.

For data cleaning to improve data quality, a set of cleaning rules are defined. The ETL process follows the same cleaning rules laid out by Nielsen et al. [9], which are designed to filter out noise from the AIS data. These cleaning rules are similar to the work of Graser [15].

The DW stores the AIS data processed by the ETL process, which results in multiple representations being available in the DW, with the finest representation being trajectories, as defined in Definition 2.

The DW can be queried directly by power users with a deep understanding of the DW design. Other users can send API requests to pre-optimized API endpoints without worrying about performance, removing the requirement of the user to understand the underlying DW design.

The design of the DIPAAL platform is not locked to a particular AIS data foundation. However, Danish AIS data[1] published by Danish Maritime Authority (DMA) are used for development, evaluation, and demonstration of DIPAAL.

### C. Data Warehouse

To create an efficient query platform for large-scale trajectory data, the underlying storage engine must support spatio-temporal data and distributed storage.

PostgreSQL [2] does not natively support spatio-temporal data or distributed storage. Nevertheless, PostgreSQL is extendable, and its extension API allows multiple extensions to co-exist and work in unison [14, sec. 38.1]. Utilising the extensions Citus [3], PostGIS [4], and MobilityDB [5], the missing support are added to PostgreSQL, making it a viable storage engine for DIPAAL.

*Distributed Storage Support:* Citus enables distribution of PostgreSQL by sharding relations and storing these shards across a cluster of workers. Citus support co-located relations [16, sec. 12.4], meaning rows with the same value in their distribution attribute are located on the same worker. Co-location increases data locality and enables local joins [16, sec. 14.4.4], which significantly reduces query runtime. In contrast, non-co-located joins must transfer data between workers to perform a join.

Distribution of the data also enables the workers in the Citus cluster to share the workload by each computing part of a query result from its local data, which are combined and presented to the user [17].

*Spatio-temporal Support:* PostGIS extends PostgreSQL with a range of spatial data types, operators, and functions. PostGIS is used for raster operations, to enable the heatmap functionality of DIPAAL, as well as provide spatial join conditions.

MobilityDB builds on top of the PostGIS extension and adds support for temporal and spatio-temporal data by lifting the operations and functions provided by PostgreSQL and PostGIS [18]. By defining multiple type constructors, MobilityDB enables working with both discrete spatio-temporal data, as well as providing Moving Object Database (MOD) capabilities [18].

## III. ETL Design

The ETL process consists of interchangeable and self-contained modules to facilitate the separation of concerns and improve modularity,

The pipeline of the ETL process is shown in Figure 3, describing the concern of each module and their respective inputs and outputs, with arrows depicting the data flow between modules. The base use case is the synchronous execution of the pipeline, as ordered by the numbers in Figure 3. Additional use cases are to clean the AIS data by running the DW-independent modules, indicated by the red box, or load previously cleaned data by running the DW-dependent modules, indicated by the blue box. This separation enables the DW-independent parts to be executed in parallel on multiple machines, with no coordination, besides transferring the result to the *bulk inserter module*. The individual modules are described in more detail below.

### A. File Download Module

This module provides a thin integration that enables the ETL process to find, download, and prepare the raw AIS source files automatically. An example of file preparation is the unzipping of archived files.

[1] https://web.ais.dk/aisdata/

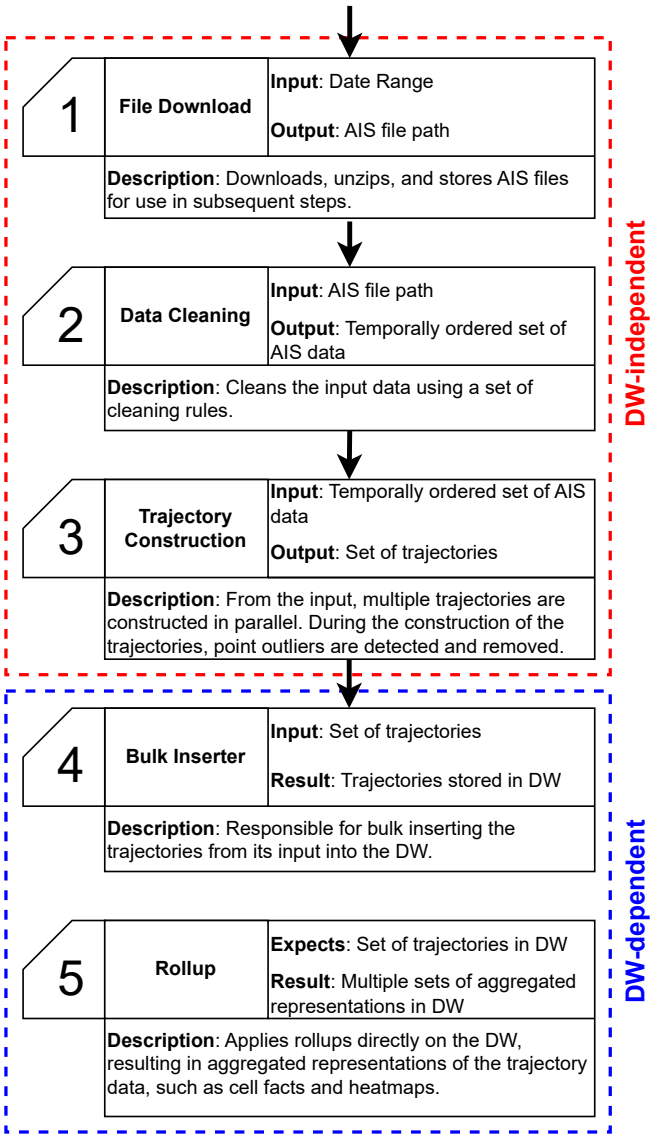



Fig. 3: Pipeline of the ETL process. The modules in the red box are DW-independent and can be run anywhere. The modules in the blue box are DW-dependent and must be run on the cluster.

### B. Data Cleaning Module

Due to the circumstances regarding AIS data described in Section I, it is necessary to improve the data quality. This module removes noisy AIS data by enforcing the following domain-specific cleaning rules, defined by Nielsen et al. [9]. These cleaning rules coincide with some of the steps in the protocol created by Graser [15], which helps identify common problems in moving data.

1) Remove duplicate rows
2) Remove ships with unnatural dimensions
3) Remove ships with an MMSI that does not follow the correct format
4) Remove ships with a position on land
5) Remove ships outside of defined geometric bounds

### C. Trajectory Construction Module

The *trajectory construction module* is responsible for constructing the trajectory facts to be stored in the DW, as defined in Definition 2.

The module creates trajectories from a temporally ordered AIS data set. To handle each ship in isolation, the input data are grouped based on the ship's MMSI value. MMSI is a mostly unique identifier, as it uniquely identifies a ship at a given time, but it may change under certain circumstances [19, 12]. Although the International Maritime Organisation (IMO) number is a unique identifier, it is not used for grouping, as it is not required for AIS transmissions [19, 12].

The trajectories are classified as either stopped or moving, enabling analysis to focus on either classification.

To construct trajectories, the *trajectory construction module* iterates over the temporally ordered AIS data set and examines the points pairwise. From this examination, the module determines which set of points represents moving and stopped ships, creating trajectories for each. The pairwise examination is further used to improve the data quality by removing spatio-temporal outliers, as defined in the work of Nielsen et al. [9].

### D. Bulk Inserter Module

The *bulk inserter module* is responsible for inserting data from the trajectories into the appropriate relations in the DW. The *bulk inserter module* is able to insert many rows at a time but also split extensive inserts into multiple batches. If an identical row already exists in the DW, the *bulk inserter module* instead returns the primary key identifier of the existing row, which can be used for referencing. The case where an identical row already exists only happens on dimension relations, such as previously inserted ships.

### E. Rollup Module

After the insertion of the produced trajectories, the *rollup module* is responsible for executing the rollup queries that aggregate existing data in the DW into new representations. An example of a rollup query is the transformation of trajectory facts into cell facts, as seen in Figure 1. The work is trivially distributed and parallelised using the DW, as Citus automatically creates a distributed query plan.

## IV. Data Warehouse Design

No golden standard exists for designing and implementing a moving object DW, especially in a distributed scenario. The design of DIPAAL is thus created from a collection of sources and recommendations and serves as a proposal on efficiently distributing analytics of large scale moving object data.

The DW design of DIPAAL is partially based on the recommendations of Kimball and Ross [20]. It is partially based since Kimball and Ross do not make recommendations for some aspects of the spatio-temporal and moving-object world, such as spatial references and representations. For these

aspects, the DW design is based on the recommendations made by Vaisman and Zimányi [21] and from consultations with domain experts, see Appendix A. The overall DW design is based on a star schema with an RDBMS as the physical foundation, which Kimball and Ross recommend as an option [20, chap. 1, p. 9].

The complete DW design is seen in Figure 4. The DW comprises six fact tables that individually are part of a partially snow-flaked star schema with conformed dimensions except for the time and date dimensions which are role-playing dimensions. The complete schema is a fact constellation schema [22].

Each attribute in the fact constellation schema is depicted using three sections, where the leftmost section specifies if the attribute is part of the primary key (PK) and/or a foreign key (FK). The middle section is the attribute's name, and the rightmost section is the datatype. Attributes that are both part of the PK and a FK are represented with a green colour, while attributes that are only part of a PK are represented with a blue colour. If a PK attribute is the distribution attribute, it has a purple gradient. The distribution attribute determines which Citus shard the row lies in. Each arrow points from a FK to the attribute it references, as seen in Figure 4. Arrows to `dim_spatial_division`, `dim_date`, and `dim_time` are not shown but are implicit for all attributes named with *division_id*, *date_id* and *time_id*. **Tgeompoint**, **stbox**, and **tfloat** are types from MobilityDB, where **tgeompoint** represents temporal geometric points, **stbox** represents a spatio-temporal box, and **tfloat** represents temporal floats [5].

Four granularities are chosen for the cell representation. These granularities are chosen as a result of collaboration with domain experts combined with the findings of the original DIPAAL paper [1]. The chosen granularities are 5000m, 1000m, 200m, and 50m.

### A. Facts

Kimball and Ross recommend the fact table foundation builds on atomic events [20, chap. 1, p. 17]. However, the data foundation of each row in the fact table `fact_trajectory` is a series of points traversed by a ship, i.e., a trajectory. This granularity is used for `fact_trajectory`, as no information is lost by aggregating points into trajectories; see Definitions 1 and 2.

`Fact_trajectory` contains measures that describe a ship's trajectory, as defined in Definition 2 and seen in Figure 1a. The measures `duration` and `length` represent the timespan of the trajectory and its length in meters, respectively, and are both additive measures [20, chap. 2, p. 47]. Lastly, the measure `infer_stopped` is a boolean where TRUE means the trajectory is inferred to represent a non-moving ship within a small area, for example, an anchored ship, and FALSE means the ship is moving.

The four cell fact relations are aggregate fact tables based on `fact_trajectory`, as shown in Figure 1, and contain measures for events within defined cells. These events describe the movement of a ship between it entering and exiting a cell. The `SOG` averages the ship's SOG during the cell event. `Delta course over ground` (COG) and `heading` are measures of how much the ship's course and heading changed inside a cell. The attribute `bounding_box` describes the spatio-temporal bound of a cell fact and is solely included for indexing to reduce the query runtime of spatio-temporal queries. The `bounding_box` is necessary for the index as PostgreSQL as of version 15 does not support cross-relation indices [14]. Lastly, `draught` is the minimum draught of the ship inside of the cell, as the minimum draught can be used to complete a minimum depth chart as performed by Nielsen et al. [9]. The draught of a ship can change inside of a cell, but if analysis of draught is desired on a finer granularity than the grid provides, the data in `dim_trajectory` should be used instead. All of these measures are additive, except draught, which is non-additive [20, chap. 2, p. 47].

The `fact_heatmap` relation contains entries defining a raster for a 5 by 5km cell. The 5 by 5km raster contains the finer granularities by storing multiple pixels in the raster. For example, the 50m resolution raster is a 5 by 5km raster containing 10 000 pixel values. This is a design choice, as it reduces query runtime by aggregating fewer, larger rasters, compared to many, smaller rasters [23, p. 6].

*Rast*, is an advanced [21] and semi-additive measure, as it cannot be aggregated across different heatmap types or granularities. The `fact_heatmap` relation serves as an intermediary pre-aggregate of the four cell fact relations, where entries of the relation can be aggregated further to create heatmaps. Pre-aggregation minimises the computation needed at query time, while preserving the ability to create parameterised heatmaps.

The `fact_heatmap` relation also contains measures for the spatial and temporal resolution in spatial reference units, which for the Danish data foundation is meters from EPSG:3034[2], and seconds, respectively. These measures serves to enable storing multiple resolutions. DIPAAL currently pre-aggregate heatmaps with spatial resolution of the four cell granularities, and one day as the temporal resolution, as per request by domain experts, see Appendix A.

### B. Dimensions

Both `fact_trajectory` and the four cell fact relations reference the dimension `dim_trajectory`. This dimension is responsible for storing the variable length fields of a trajectory. It stores a **tgeompoint**, representing the trajectory, but also **tfloat**s for the *rate of turn* (ROT), *heading*, and *draught*. Lastly, it has a **varchar** attribute representing a trajectory's *destination*.

The `dim_heatmap_type` stores metadata about different heatmap types, such as how to aggregate the rasters, and its name and description. DIPAAL, at the time of writing, pre-aggregates five heatmap types; count of ships crossing a cell, accumulated time spent in a cell, average delta change in heading in a cell, average delta change in COG in a cell,

[2]https://epsg.io/3034

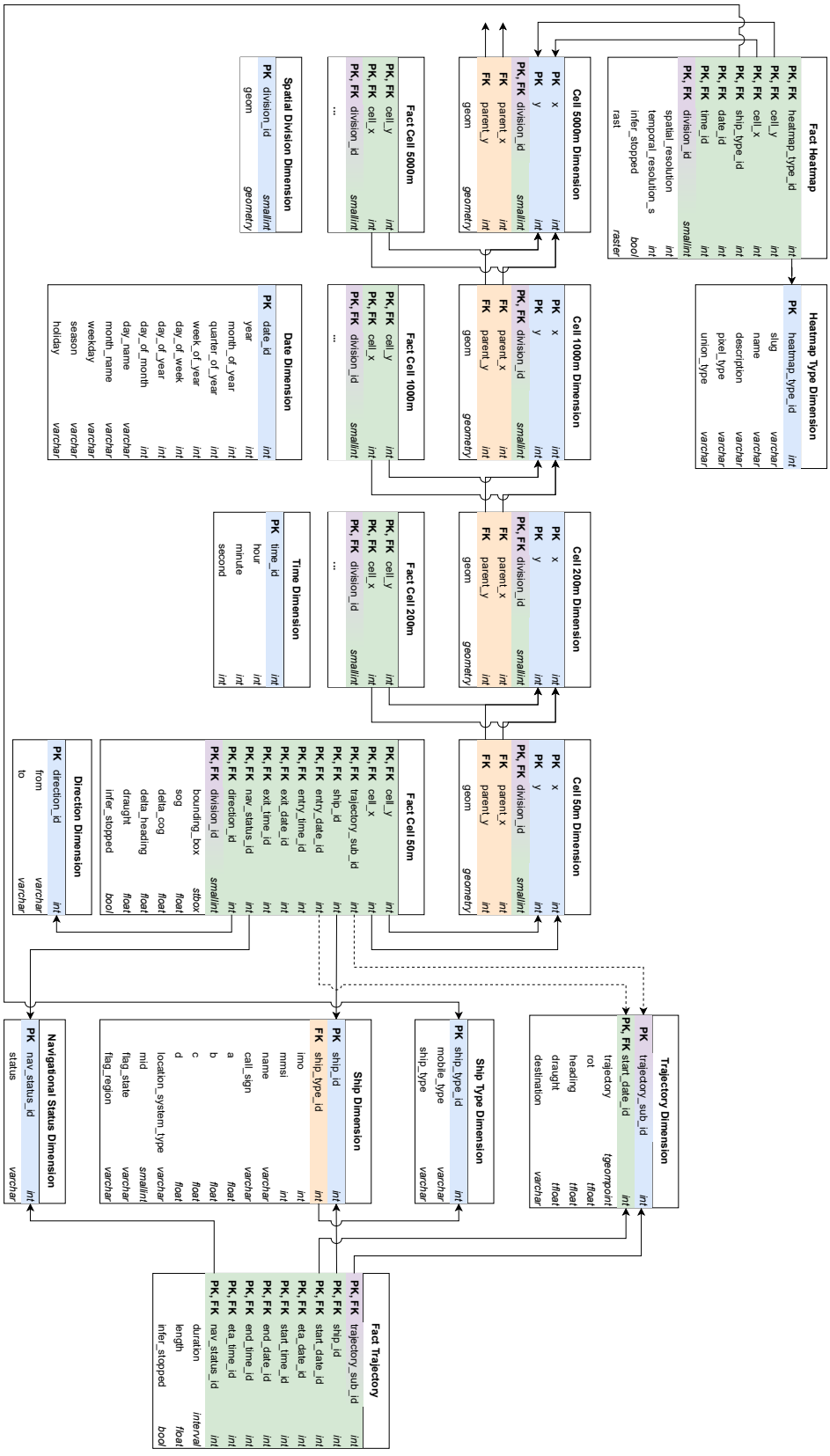



Fig. 4: Fact constellation schema for the DW.

and minimum draught in a cell. A two-band raster is used to support aggregating average values, with one band being the sum of the value, and the other being the count. The `dim_heatmap_type` enables the possibility of defining new heatmap types dynamically without any structural changes to the DW, thus only requiring trivial ETL changes.

Two dimensions describe the date and time of a fact. Splitting the temporal property into two dimensions reduces the number of rows needed as it incurs that `dim_date` only grows by one row per day, while `dim_time` is fixed in size. These dimensions use smart keys [20, chap. 3, p. 101] as primary keys to simplify queries and the ETL process.

`Dim_ship` contains the information about ships, except the *mobile type* and *ship type* attributes, which are stored in a fixed depth hierarchy in `dim_ship_type`. `Dim_ship` is a type-0 slowly changing dimension, where the original is stored and never changes [20, chap. 2, p. 54].

`Dim_nav_status` contains the navigational statuses available for the ship crew to set according to the AIS definition [12]. The navigational status is in its own dimension, as Kimball and Ross recommend avoiding textual data in the fact tables [20, chap. 1, p. 12].

The four cell dimensions all store a two-dimensional index of cells and their physical geometries using the PostGIS **geometry** datatype. The four cell dimensions use the same hierarchy and granularity levels as the four cell facts. The four cell dimensions are linked in a snow-flake hierarchy, with smaller cells referencing their larger parent cell.

A snow-flake approach is chosen to avoid the data explosion which occurs if the hierarchy is represented as a denormalised star schema hierarchy. Denormalising the hierarchy with 5000m and 50m cells results in the 5000m granularity being replicated 10 000 times due to a 5000m cell containing 10 000 50m cells. Furthermore, a snow-flake approach trivialises adjusting the hierarchy in the future.

`Dim_direction` contains the cross product of the four directions, which is formed by a cell's sides and an unknown direction. These are referenced by `dim_direction`'s *to* and *from* attributes, determining from which direction a ship entered and exited a cell, respectively. For example, in Figure 1b, in the right subfigure, the upper left cell has the direction south to north, while the upper right cell has the direction south to south.

`Dim_spatial_division` is a non-changing fixed dimension. It contains two attributes, consisting of an id and a geometry. This dimension is used for every relation that is spatially distributed. The spatial distribution scheme is described in depth in Section V.

Besides the fact and dimension relations, an audit table, not shown in Figure 4, is created to keep track of the ETL runs as recommended by Kimball and Ross [20, chap. 2, p. 66]. The audit table contains information such as the ETL version, the start time of the ETL process, the date that was loaded, and the runtime of each stage of the ETL process.

### C. Distributed Relations and Distribution Attributes

The dimensions `dim_nav_status`, `dim_direction`, `dim_spatial_division`, and `dim_time` are fixed size. The `dim_date` relation grows on average by one row for each loaded day, and `dim_ship` grows by the number of new ships every day but is expected to grow by less than a few hundred rows per day. All these relations are configured as reference relations, meaning all the data in these relations are replicated to all workers in the Citus cluster to ensure data locality [16, chap. 10, p. 56].

The relations `fact_trajectory`, `dim_trajectory`, the four cell facts, and `fact_heatmap` are expected to grow the fastest. For this reason, these are chosen to be distributed. The four cell fact and `fact_heatmap` relations are distributed spatially using the `dim_spatial_division` dimension, which is described in more detail in Section V. The four cell dimension relations are spatially distributed together with the four cell fact relations, ensuring data locality.

For the `fact_trajectory` and `dim_trajectory` relations to be distributed spatially, it requires the trajectories to be split into segments that fit inside the divisions defined in `dim_spatial_division`. Splitting the trajectories requires trajectory reconstruction if the complete trajectory is needed for a query, involving cross-node reconstruction and repartitioning. It is thus chosen to instead use random hash partitioning for the trajectories. This design choice means that to create cell facts from the trajectories, a repartitioning has to occur, as the cell relations are not co-located with the trajectory relations. This issue is discussed in detail in Section VIII-B.

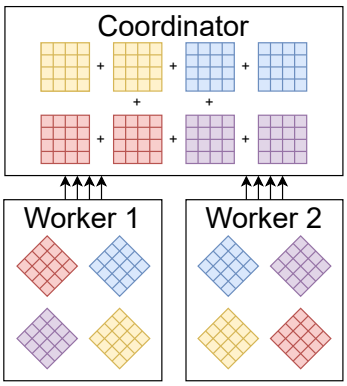
## V. Spatial Distribution

DIPAAL can produce a variety of heatmaps. The production of heatmaps is a computationally heavy task, scaling with the temporal and spatial spans. Therefore, the distribution of such tasks and the distribution method used are important.

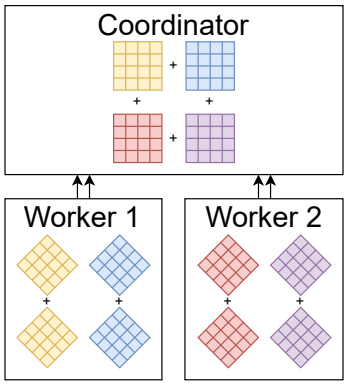
### A. Motivation

To facilitate efficient distributed calculation of heatmaps, all values of a pixel must be located in the same shard, as the summation can then be pushed down to the workers, which is visualised in Figure 5. Rasters of the same colours indicate rasters of the same area but may represent, for example, a different time, ship type, or mobile type. An example for two rasters of the same colour is one for the ship type "Cargo", and one for the ship type "Passenger". The notion "area" refers to the area covered by the raster.

In Figure 5a, a random distribution scheme is used. In this scheme, each worker node must transmit every raster to the coordinator, aggregating the rasters into the final raster. Aggregating the rasters on the coordinator is expensive in network overhead and limits the system to the computational power of the coordinator. It is, therefore, not horizontally scalable. In Figure 5b, a spatial distribution scheme is used. The spatial distribution guarantees that all rasters for a given area is located on the same worker. Thus, the workers perform the raster aggregation for each area, and only transmit one




(a) Visualisation of calculating rasters distributed randomly.




(b) Visualisation of calculating rasters distributed spatially.

Fig. 5: Comparison of the calculations in a spatially distributed scheme versus a randomly distributed scheme.

raster per area to the coordinator. The coordinator perform less work, as the only operation it has to do is align the areas into a large raster.

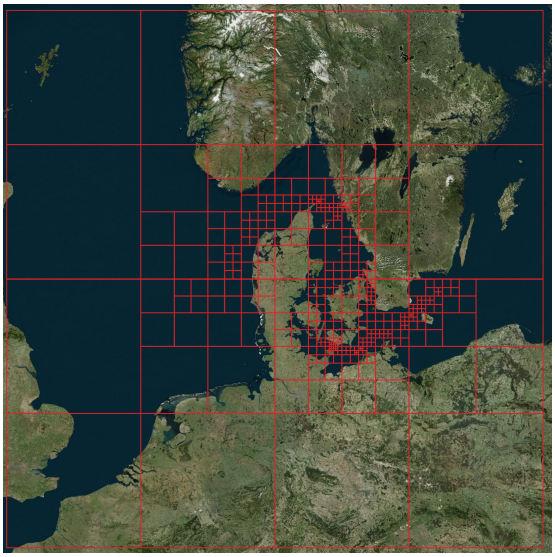
The example in Figure 5a only has two rasters per area, and four areas, which results in eight rasters being transmitted using the randomly distributed scheme. If, instead, there are 200 rasters per area, for example, 10 ship types over 20 days, it has to transmit 800 rasters. If horizontal scaling is desired, random distribution is thus unfeasible. For the spatial distribution scheme, the number of rasters transmitted is constant regarding the number of areas, as there is only one aggregated result from the workers per area.

To horizontally scale spatial aggregates, the cell facts and dimensions are spatially co-located with the heatmaps.

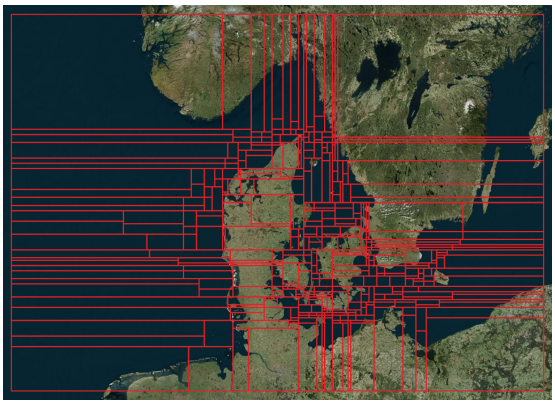
Quad-tree and kd-tree-based spatial distributions for the spatial domain of DIPAAL are seen in Figure 6. Figure 6 is explained in further detail in Section V-B.

### B. Spatial Distribution Approach

It is shown that spatial distribution is necessary to scale spatial aggregations, such as heatmaps, horizontally. Therefore, the next step is determining the approach to create the spatial distributions by dividing the spatial domain.



(a) The resulting quad-tree.



(b) The resulting kd-tree.

Fig. 6: The resulting divisions from the quad-tree and kd-tree approaches, respectively.

As DIPAAL has a maximum cell granularity of 5km, no spatial division can have a side length not divisible by 5km, and no division can thus be smaller than 5 by 5km.

The spatial distribution aims to create divisions, such that the counts of cell facts are balanced between the shards. This is because the four cell fact relations are the largest by far, see Appendix D. By creating equal shards in regards to cell count, it ensures that both the load, when querying the cell relations, as well as the storage requirements for each worker are balanced. As the count of the cell facts of different granularity is proportional, the count of 5000m cell facts are used as the foundation of the spatial distribution.

Two division approaches are considered for use in DIPAAL. The first is a region quad-tree approach [24], and the second is a kd-tree approach [25]. R-trees can also be used for spatial division, but since they do not spatially cover the domain, it is discarded as an option. Using R-trees for spatial division was explored by Li et al. [26], where the non-covering property

resulted in a work-around that ended up performing poorly compared to quad- and kd-trees.

For the quad-tree construction, a global spatial domain is chosen by finding the shortest side lengths that surround the global spatial domain of the data while having side lengths, $s$, be $s \in 2^n * 5000m$ where $n$ is a natural number. The side length must be 5 000m multiplied by a two exponent, as each quad-tree split divides the cell's side lengths in half. This multiplication ensures each cell, even after $n$ splits, is divisible by 5km. Consequently, $n$ defines the maximum depth of the quad-tree. For each split, the division with the highest number of cells is chosen until it reaches a maximum depth, such that splitting again results in divisions smaller than 5 by 5km.

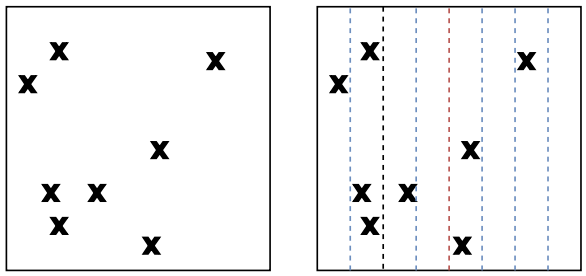
A similar approach is taken for the kd-tree construction with the pseudo-code seen in Algorithm 1. The input to the function is the global spatial domain and a limit on the number of divisions created. Line 1 counts the total number of cell facts in the global spatial domain. On line 2, *divisions* are initialised as a set consisting of the global domain. Each division in the *divisions* set is a tuple consisting of a geometry defining the division and a number describing the number of 5000m cell facts contained in the division.

The while loop on lines 3 through 7 loops as long as it has not reached the maximum number of divisions to create. Each iteration pops the division with the highest number of cell facts that is capable of splitting. Being capable of splitting means having any side length, that can be divided, such that the new side length is divisible by 5km. The best split for the largest division is found on line 5 based on a binary-search approach, where the two new divisions have the highest balance regarding the number of cell facts. The new divisions are added to the divisions set on line 6.

**Algorithm 1** Spatial Divisions Kd-tree Construction

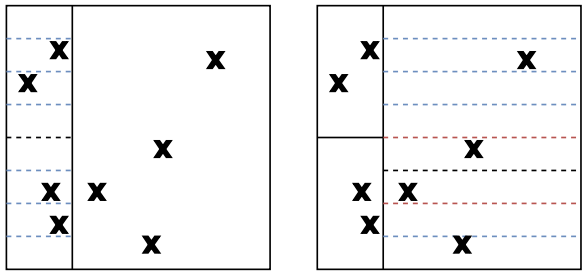
**Input:** $maxNumDivisions$
**Input:** $globalDomain$
**Output:** $divisions$
1: $maxPoints \leftarrow \textbf{count}(fact\_cell\_5000m)$
2: $divisions \leftarrow \{(\text{global domain}, maxPoints)\}$
3: **while** $|divisions| < maxNumDivisions$ **do**
4: $division \leftarrow$ *pop largest division capable of splitting*
5: $p1, p2 \leftarrow \textbf{getBestSplit}(division)$
6: $divisions \leftarrow division \cup p1 \cup p2$
7: **end while**
8: **return** divisions

An example of a kd-tree construction using the method in Algorithm 1 is seen in Figure 7. In Figure 7a, the initial spatial domain is shown. The crosses mark individual cell facts. The algorithm then pops the global domain and finds the best split along the x-axis in Figure 7b. The dashed lines indicate possible split lines. Not all x-coordinates are valid split lines, as it must align with the 5km grid. Since the best split is found using a binary-search approach, the centre line is checked first. It is determined from this check that the best split must be to the left of the centre line, as there are five



(a) The initial global spatial domain and the cell facts within.

(b) The first split is performed along the x-axis



(c) The second split is performed along the y axis in the left-most partition.

(d) The third split is performed along the y-axis in the right-most division.

Fig. 7: An example showing a global domain being split into four balanced divisions using the kd-tree algorithm in Algorithm 1.

cell facts to the left but only three to the right of the centre line. The line is marked as red to visualise the lines checked using the binary-search approach. The left half is then divided in half and checked. This line, marked as the black dashed line, equally splits the division with four points on each side and is thus chosen.

In Figure 7c, the new and smaller left division is chosen. As the kd-tree splits in alternating directions, this level of the kd-tree is split along the y-axis. The best split is found as the centre line, dividing the division into two divisions with two cell facts in each. Lastly, the same is done for the right division in Figure 7d. In Figure 7d, the two dashed red lines indicate the splits that are considered using the binary search approach. First the center line is tested, then the centre line of the bottom half, before testing the black dashed line, which is determined to be the best split.

Both the quad- and kd-tree approach has a parameter defining the maximum number of divisions created. 400 divisions are chosen as the maximum number of divisions based on experimentation on the year 2021. This experimentation shows that creating additional divisions requires splitting divisions into sizes less than 5km or introducing a higher imbalance of the divisions.

The resulting spatial divisions of both approaches do not change over time, i.e. they are static. The static property

enables push down of spatial aggregation to the workers, regardless of the query's temporal range. The consequence of static division is the possibility of divisions becoming skewed as traffic patterns evolve or new AIS receivers are installed. Data deviation over time is evaluated in Section IX-G.

The resulting quad- and kd-tree are shown in Figure 6, with the quad-tree approach in Figure 6a and the kd-tree approach in Figure 6b.

The data deviation is compared using the measures Standard Deviation (SD) [27] and Coefficient of Variance (CV) [28]. For both metrics, a lower value indicates better balanced shards. The 2021 data contain 9 million trajectories, travelling a total of 85 million kilometres by 28k unique ships.

After constructing the quad- and kd-trees on AIS data for 2021, the SD and CV are measured. For the quad-tree, the SD is 81 192, and the CV is 106%. For the kd-tree, the SD is 31 814, and the CV is 41%. Kd-tree-based divisions are better balanced and is used as the distribution approach for DIPAAL. This result is explained by the unevenness of the data foundation, which is due to the nature of navigation on water that mostly follow defined sea lanes, resulting in high- and low-traffic areas. The dynamic nature of the kd-tree approach better captures this unevenness compared with the more rigid quad-tree approach, and thus, these results may not be applicable to other spatial domains or datasets.

## VI. API Design

As mentioned in Section II-B, an API intermediary is designed which provides non-trivial pre-optimised queries, written with detailed knowledge of the DW design. To improve security, the API is restricted to a read-only DW user. The API is a stateless, OpenAPI[3]-compliant wrapper.

The API's queryable resources are restricted to the DW's core concepts. The core concepts of the DW are the result of collaboration with domain experts, as seen in Appendix A, and lists heatmaps, cell facts of all granularities, trajectories, and ships.

### A. Endpoints

For all queryable resources, besides heatmaps, there is an URL (Uniform Resource Locator), which links directly to a single entry of a resource, as well as a query endpoint that finds all matching resources. For example, **/api/v1/ships/123** finds the ship with **ship_id** 123. However, the query **/api/v1/ships?length_gt=100&ship_type=Passenger** find all ships with the ship type **Passenger** and a length greater than 100 meters.

All endpoints in DIPAAL use the GET HTTP method, and all parameters are thus either path- or query-parameters. This approach is chosen both in order to comply with the HTTP specification but also to support tools that are used by domain experts, such as QGIS's[4] raster functionality, which does not support body-parameters.

Parameters that refer to a non-aggregatable hierarchy in the DW design are path parameters. For example, all heatmaps with the resolution of 200m and the type **count** are found at the path **/api/v1/heatmaps/count/200m**. All other parameters are query parameters. For example, to add a temporal constraint to the previous example, the path would be **/api/v1/heatmaps/count/200m?start_timestamp=2022-01-01T00%3A00%3A00Z**. The **%3A** parts are the URL encoding of a colon, as the timestamp is of the ISO8601[5] Zulu time format.

The output format of the endpoints in DIPAAL is JSON, as there are well-supported JSON standards for moving features (MFJSON[6]) and geometries (GeoJSON[7]). For the heatmap endpoints, which return rasters, the output format is either a GeoTIFF, a pre-rendered raster in the form of a PNG image, or an MPEG-4 video. The pre-rendered results allow the endpoint users to see a result without having a specialised GIS-client installed.

The specification for the API and all available endpoints is available at https://dipaal.dk/docs. Direct SQL-access to the DW is required if ad-hoc access to non-core concepts are needed.

### B. Example of Usage

One example of the API usage is shown, with more examples found in Appendix B

This example covers how to get a heatmap from the API. For the sake of demonstration, the complete count heatmap for the years 2011, 2019, 2020, 2021, and 2022 for all ship types and mobile types is requested at a 1000m spatial resolution. This is done through the URL https://dipaal.dk/api/v1/heatmap/single/count/1000m?output_format=tiff&x_min=3480000&y_min=2930000&x_max=4495000&y_max=3645000&srid=3034&start_timestamp=2011-01-01T00%3A00%3A00Z&end_timestamp=2022-01-01T00%3A00%3A00Z, where the 1000m path parameter determines the 1000m resolution, the *output_format=tiff* query parameter requests the output as a GeoTiFF, and the rest of the query parameters determine the spatio-temporal bounds of the heatmap.

The GeoTiFF output is rendered using QGIS and is seen in Figure 8.

## VII. Data Preparation

This section covers the implementation of selected topics on AIS data preparation.

Data preparation covers the DW-independent sub-modules of the ETL process shown in Figure 3.

A custom geometric bound is created for the *data cleaning module* to limit the scope of the spatial domain of DIPAAL. The bound is created from open coastline data made available by OpenStreetMap[8]. Changing the area for the cleaning in DI-

[3]https://github.com/OAI/OpenAPI-Specification

[4]https://qgis.org/

[5]https://www.iso.org/obp/ui/#iso:std:iso:8601:-1:ed-1:v1:en

[6]http://docs.ogc.org/is/19-045r3/19-045r3.html

[7]https://datatracker.ietf.org/doc/html/rfc7946

[8]Data: https://osmdata.openstreetmap.de/data/coastlines.html, attribution: https://www.openstreetmap.org/copyright

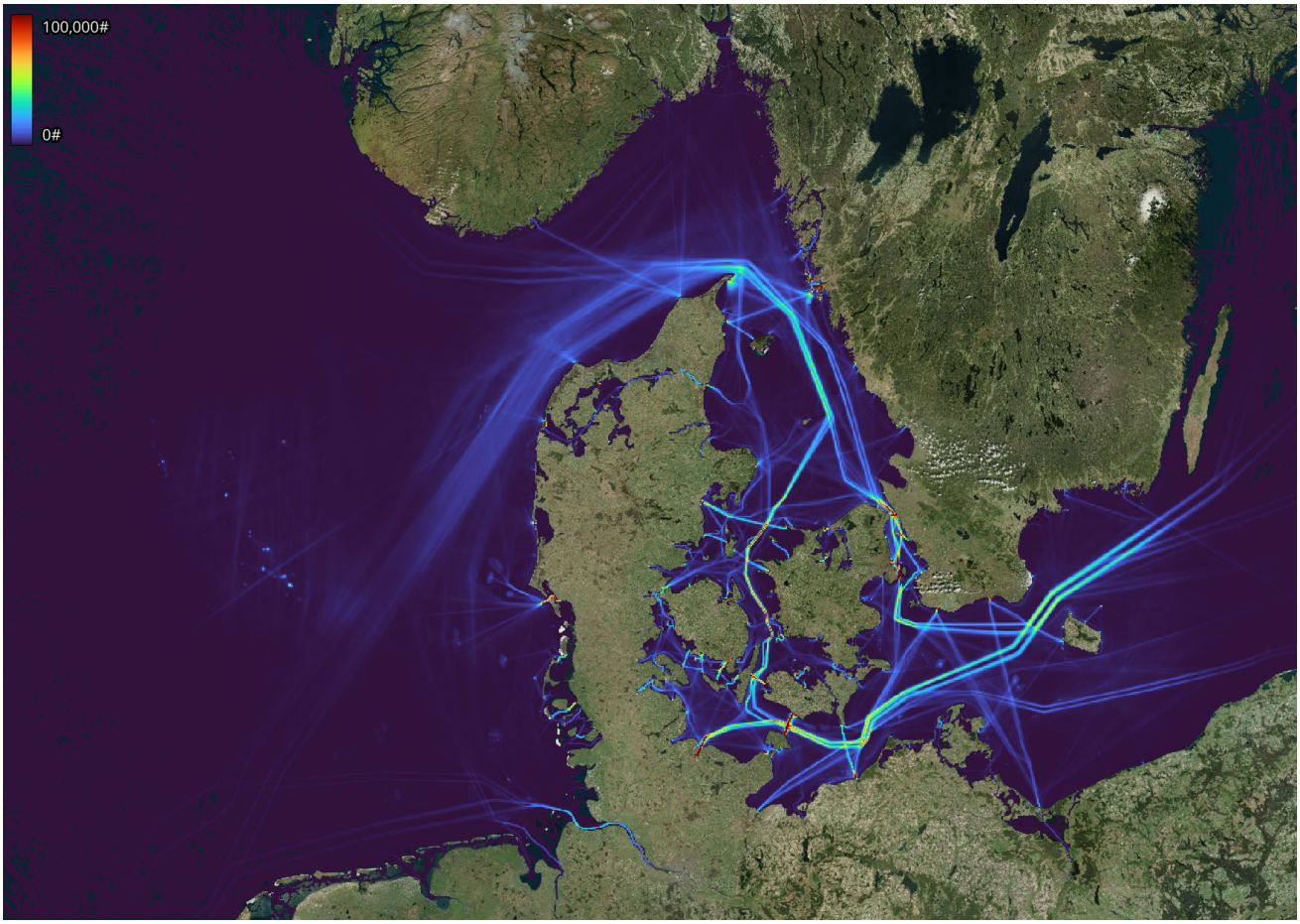



Fig. 8: Count heatmap of all ships in 2011, 2019, 2020, 2021, and 2022, as requested from the API in 1000m resolution.

PAAL is trivial and only requires defining the new geometric bound.

The ship's SOG is used to determine whether a trajectory is classified to represent either a moving or a stopped ship. If the SOG is above a threshold of 0.5 knots, the ship is considered to be moving, and if the SOG is below the threshold for at least five minutes, the ship is considered stopped [9, sec. 5].

### A. Data Cleaning

The *data cleaning module*, as seen in Figure 3, reads a single AIS data file and improves the quality of the AIS data using the cleaning rules defined in Section III-B.

Algorithm 2 describes how the AIS data are cleaned. Line 2 reads the AIS data described by the file path *aisfp* from storage. The AIS files must be temporally ordered in DIPAAL. The raw AIS data are looped through on lines 3 to 7, which removes data points that are not clean according to Section III-B.

### B. Trajectory Construction

The Algorithms 3 to 5 are implemented as part of the *trajectory construction module* seen in Figure 3.

Algorithm 3 describes the entry point for constructing trajectories. The input is a temporally ordered set of clean AIS points. The algorithm outputs a set of trajectories constructed from the input set. Line 2 groups the AIS points based on the ship while keeping the ordering for each group created. This grouping allows for complete parallelisation of outlier removal on line 4 and trajectory construction on line 5, as no trajectory spans multiple ships. The call to **constr_traj** on line 5 takes an additional two arguments, the index in the set at which to start trajectory construction and whether a moving or stopped trajectory is constructed. In the call to **constr_traj**, these arguments have been omitted for readability, but it is

**Algorithm 2** AIS Data Cleaning

**Input:** $aisfp \leftarrow file\ path\ to\ AIS\ data\ file$
**Output:** $cds \leftarrow set\ of\ clean\ AIS\ data\ points$
1: $cds \leftarrow \varnothing$ ▷ Initialise result set
2: $points \leftarrow \boldsymbol{read\_csv}(aisfp)$
3: **for** $p \in points$ **do**
4: **if** $\boldsymbol{is\_clean}(p)$ *and* $\boldsymbol{is\_in\_bounds}(p)$ **then**
5: $cds \leftarrow cds \cup p$
6: **end if**
7: **end for**
8: **return** $cds$

**Algorithm 3** Trajectory Construction: Overview

**Input:** $cds \leftarrow set\ of\ clean\ AIS\ data\ points$
**Output:** $ts \leftarrow set\ of\ trajectories$
1: $ts \leftarrow \varnothing$ ▷ Initialise result set
2: $grouped\_points \leftarrow \textbf{group\_by\_ship}(cds)$
3: **parallel for all** $grp \in grouped\_points$ **do**
4: $grp \leftarrow \textbf{rm\_outliers}(grp)$ ▷ Algorithm 4
5: $trajs \leftarrow \textbf{constr\_traj}(grp)$ ▷ Algorithm 5
6: $ts \leftarrow ts \cup trajs$
7: **end for**
8: **return** $ts$

implicitly called with 0 and TRUE. On line 6, the constructed trajectories for the ship are added to a shared set of constructed trajectories, which are outputted on line 8.

**Algorithm 4** Trajectory Construction: Remove Outliers

**Input:** $sscps \leftarrow single\ ship\ sorted\ clean\ AIS\ point\ set$
**Output:** $ofcps \leftarrow outlier\ free\ and\ clean\ AIS\ point\ set$
1: $ofcps \leftarrow \varnothing$ ▷ Initialise result set
2: $prv\_pnt \leftarrow \textbf{null}$
3: **for** $pnt \in sscps$ **do**
4: **if** $prv\_pnt \neq \textbf{null}$ **then**
5: $time\_diff \leftarrow pnt.t - prv\_pnt.t$
6: $speed \leftarrow \textbf{calc\_speed}(prv\_pnt, pnt, time\_diff)$
7: **if** $time\_diff == 0 \vee speed \geq SPD\_THR$ **then**
8: **continue**
9: **end if**
10: **end if**
11: $prv\_pnt \leftarrow pnt$
12: $ofcps \leftarrow ofcps \cup pnt$
13: **end for**
14: **return** $ofcps$

Algorithm 4 describes how outliers are removed during trajectory construction. Algorithm 4 takes as input a temporally ordered set of clean AIS points *sscps*. The output is a temporally ordered set of clean AIS points without outliers *ofcps*.

Line 1 initialises the result set *ofcps* as an empty set, and line 2 initialises variable *prv_pnt*, which tracks the last non-outlier point checked. The for-loop spanning lines 3-13 loops through the input keeping the temporal order. If it is the first iteration, then *prv_pnt* is null. In that case, the if-statement on line 4 jumps directly to lines 11-12, which adds the first point to the result set. Otherwise, the time difference between the point's timestamps is calculated on line 5.

The time difference is used along with *pnt* and *prv_pnt* by **calc_speed** to calculate the *speed* between the points. The speed is either computed based on the points positions and *time_diff*, or the SOG attribute of *pnt*. If SOG is undefined, the computed speed is always used. Otherwise, the difference between the computed speed and the SOG of *pnt* is used to decide which is used. If the difference is above a configurable threshold, SOG is used, otherwise, the computed speed is used. Nielsen et al. determines that SOG is more trustworthy if the difference between the speeds is above a two knots threshold [9, sec. 5].

Line 7 checks if *pnt* is an outlier or a duplicate point. *pnt* is regarded as a duplicate if it has the same timestamp as *prv_pnt*, i.e., *time_diff* = 0, as no two AIS points for the same ship should have the same timestamp. *pnt* is an outlier if the sailing speed exceeds a configurable *SPD_THR* threshold. In the current implementation, this threshold is set to 100 knots based on the work of Nielsen et al. [9, sec. 5]. If *pnt* is neither an outlier nor a duplicate, it is added to the *ofcps* result set and set as *prv_pnt* on lines 12 and 11, respectively.

**Algorithm 5** Trajectory Construction: Build Trajectory

**Input:** $grps \leftarrow set\ of\ grouped\ points$
**Input:** $fm\_idx \leftarrow from\ index$
**Input:** $cstr\_mv \leftarrow boolean\ state\ if\ constructing\ moving\ trajectories$
**Output:** $trajs \leftarrow set\ of\ constructed\ trajectories$
1: $trajs \leftarrow \varnothing$ ▷ Initialise result set
2: $stop\_idx \leftarrow -1$
3: **for** $idx \in \{fm\_idx, ..., \textbf{len}(grps)\}$ **do**
4: $moving \leftarrow \textbf{is\_moving}(idx)$
5: **if** $cstr\_mv$ **then**
6: $stop\_idx \leftarrow \textbf{updt\_stop}(stop\_idx, idx, moving)$
7: **if** $\textbf{is\_stopped\_traj}(stop\_idx, idx)$ **then**
8: $tj \leftarrow \textbf{fin\_traj}(grps, fm\_idx, stop\_idx, T)$
9: $rest \leftarrow \textbf{constr\_traj}(grps, stop\_idx, F)$
10: **return** $trajs \leftarrow tj \cup trajs \cup rest$
11: **else if** $\textbf{split\_traj}(idx, SPLT\_THR)$ **then**
12: $tj \leftarrow \textbf{fin\_traj}(grps, fm\_idx, idx, T)$
13: $rest \leftarrow \textbf{constr\_traj}(grps, idx, T)$
14: **return** $trajs \leftarrow tj \cup trajs \cup rest$
15: **end if**
16: **else if** $moving$ **then**
17: $tj \leftarrow \textbf{fin\_traj}(grps, fm\_idx, idx, F)$
18: $rest \leftarrow \textbf{constr\_traj}(grps, idx, T)$
19: **return** $trajs \leftarrow tj \cup traj \cup rest$
20: **end if**
21: **end for**
22: $tj \leftarrow \textbf{fin\_traj}(grps, fm\_idx, \textbf{len}(grps), cstr\_mv)$
23: $trajs \leftarrow trajs \cup tj$
24: **return** $trajs$

Algorithm 5 describes how trajectories are constructed from a set of AIS points. Its input is a temporally ordered set of AIS points for a single ship *grps*, as well as an index *fm_idx*, which determines where in *grps* the construction begins. As a third input, it takes whether it is constructing a moving or a stopped trajectory. Its output is a set of constructed trajectories. Capital T and F in the algorithm represent True and False, respectively.

The algorithm is initialised on lines 1-2, where the set of constructed trajectories *trajs* is the empty set, and *stop_idx* is a variable that holds the index where the ship is suspected of having stopped moving. On line 3, the points in *grps* are iterated through in order beginning with *fm_idx*. The function

**is_moving** on line 4 determines whether the ship is classified as moving at index *idx*. Similar to the outlier detection, seen in Algorithm 4, DIPAAL uses the computed speed instead of SOG, if the SOG is undefined.

Lines 5-15 handle if the algorithm is currently in the state of constructing a moving trajectory. On line 6, the *stop_idx* is updated if need be. A value of $-1$ in *stop_idx* means that the algorithm currently does not suspect the trajectory might be stopped. **updt_stop** uses the *stop_idx*, *idx*, and *moving* variables to determine whether the currently constructed trajectory is suspected of being stopped. In case *moving* is true, *stop_idx* is reset to $-1$, otherwise it is updated to *idx* if *stop_idx* already is $-1$.

On line 7, the algorithm checks if the temporal delta between the AIS points at *stop_idx* and *idx* are above a threshold, which is set to five minutes [9, sec. 5]. If that is the case, line 8-10 finish the current moving trajectory and returns the finished current moving trajectory unioned with a recursive call, but with the *cstr_mv* state inverted.

On line 11, the function **split_traj** uses the current index *idx* and a configurable threshold *SPLT_THR* to check whether the trajectory currently under construction is to be split. Whether the trajectory is split depends on whether the difference between the timestamps of the current point *idx* and its predecessor is below *SPLT_THR*. In the current implementation *SPLT_THR* is defined as five times the slowest reporting rate of the AIS transmission, which is three minutes [12, sec. 1.2.4]. Splitting at the *SPLT_THR* threshold help reduce the number of trajectories crossing land caused by ships missing multiple reporting rates.

On line 16, if constructing a stopped trajectory and the current index is a moving point, it stops constructing the current stopped trajectory. Afterwards, it outputs the union of the finished current stopped trajectory and a recursive call, again with the *cstr_mv* inverted on line 17-19. Lines 22-24 finishes the current trajectory if no more points are present.

## VIII. Implementation

This section describes the more interesting implementation details of DIPAAL. All the described parts are part of the DW-dependent modules seen in Figure 3.

The source code of DIPAAL is released as Open Source Software with the MIT license and are found on GitHub[9].

The number of lines of code used for developing the DIPAAL platform are seen in Table II.

| Part\CLOC | Python | SQL | Total |
|---|---|---|---|
| ETL | 1 587 | 2 085 | 3 672 |
| API | 2 854 | 998 | 3 852 |
| Benchmark | 888 | 446 | 1 334 |
| Total | 5 329 | 3 529 | 8 858 |

TABLE II: The number of lines of code for each part of the DIPAAL platform.

[9]ETL: https://github.com/DIPAAL/etl, API: https://github.com/DIPAAL/qpi

### A. Line Simplification

During the implementation of the ETL process, it is observed that constructed trajectories contain points with high spatio-temporal similarity, especially for stopped ships. Each trajectory is stored as a MobilityDB sequence of points. MobilityDB sequences imply a linear interpolation between points with consecutive timestamps, allowing the spatial position of a ship to be inferred at any given time within a trajectory. Therefore, removing data points that contribute little to the precision of the trajectory can thus reduce the query runtime of DIPAAL without significant precision loss.

Expressing trajectories using fewer points while preserving their original shape as much as possible is called line simplification [29]. To increase the performance of DIPAAL, line simplification is performed for each trajectory in the `dim_trajectory` relation. DIPAAL performs line simplification with an error bound of 10 meters, reducing the number of points stored by a factor of six. The Douglas–Peucker algorithm [30] with Synchronized Euclidean Distance [31] (SED) is used for line simplification through the `douglasPeuckerSimplify` MobilityDB function. SED differs from the Euclidean distance as it also considers the temporal aspect [31].

As a consequence of the simplification process, the trajectories are introduced to an error of 10 meters. Domain experts approve of introducing this error, as the benefits of a x6 data reduction outweigh the minor error bound introduced, as described in Appendix A.

Due to the line simplification utilising a MobilityDB function, the data must first be inserted by the *bulk inserter module*. Thus, line simplification is applied as part of the *rollup module* seen in Figure 3.

### B. Rollup

The *rollup module* is responsible for applying rollup, resulting in aggregated representations of the trajectory data. The functionality of this module is implemented as a set of SQL queries, as this module only interacts with the DW, thereby making the module highly declarative.

*Cell Rollup:* Trajectories are aggregated to populate the `fact_cell` relations for each cell granularity in the DW.

Experiments show that populating `fact_cell` by directly joining `dim_cell` to the geometry representing the trajectory in `dim_trajectory`, results in poor performance. This is due to a combination of PostGIS functions necessary for such a join, `ST_Crossing`, `ST_Contains`, and `ST_Touches`, suffering from an inefficient index filtering. Index filtering gets inefficient when the trajectories' Minimum Bounding Rectangle (MBR) is large.

For example, a trajectory representing a ship sailing from the Danish Island Bornholm through Skagerrak has a bounding box encapsulating 63 million 50m cells. In contrast, only 21 thousand cells intersect the trajectory. This poor candidate elimination leads to the R-tree-based index of `dim_cell` to be very slow.

To reduce the query runtime, the trajectories are split to minimise their MBR. To facilitate this, a staging relation populated with split trajectories is created. These split trajectories are spatially distributed as described in Section VIII-C to avoid repartitioning on future joins. Trajectories are split using the `spaceSplit` MobilityDB function. A size parameter is provided to the `spaceSplit` function, which determines the size of the resulting tiles. The optimal value of this parameter is found through experimentation, where the results are seen in Figure 9. A tile size of ∼1 km incurs the best trade-off between splitting too many times versus having a larger MBR. The 1 km tile size split takes 16.8 seconds to join on the data from January 1st 2022.

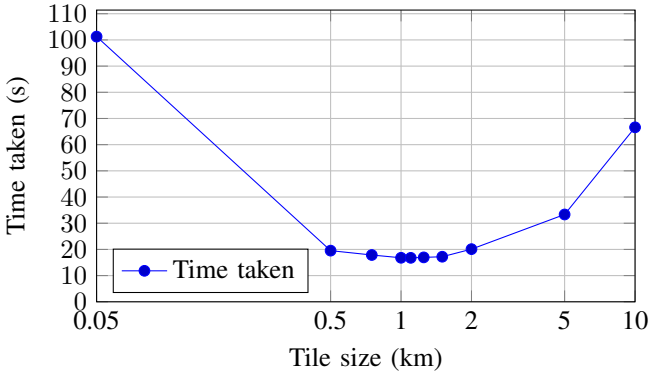



Fig. 9: The impact of changing the tile size parameter of the MoblityDB function `spaceSplit` before joining trajectories onto the 50m cell grid.

To avoid re-construction of trajectories, the size of the split trajectories must correspond to the coarsest cell granularity in the DW. It is seen that the query runtime of `spaceSplit` is increasing after the lowest point at 1.1km. Thus, 5km is chosen as the tile size, as it is the smallest tile size that encompasses the 5km cell granularity.

Note that line simplification, as mentioned in Section VIII-A, is performed before using the `spaceSplit` function, increasing the performance of `spaceSplit` by reducing the complexity of the trajectories.

Finally, once the staging relation is populated, each granularity of the `fact_cell` relations is populated by performing joins between the staging relation and the `dim_cell` relation of the appropriate granularity.

*Heatmap Rollup:* Cell facts are aggregated into rasters to populate the `fact_heatmap` relation with heatmaps.

The efficient creation of heatmaps is facilitated by the spatial distribution of the `fact_cell` relations, as explained in Section V, as it enforces data locality. The spatial distribution allows each worker to locally create a section of the total heatmap, a raster. These rasters populate the `fact_heatmap` relation and are created for each granularity of `fact_cell` in the DW. Additionally, different types of rasters are created for each granularity, each aggregating a particular type of information from `fact_cell`. DIPAAL implements the following types of rasters, where each pixel of those rasters represents aggregated information from all cell facts within the corresponding cell:

- **Count:** Pixels represent the count of all cell facts.
- **Delta COG:** Pixels represent the average change in COG within the cell.
- **Delta Heading:** Pixels represent the average change in heading within the cell.
- **Maximum Draught:** Pixels represent the maximum draught recorded within the cell.
- **Time:** Pixels represent the sum of time spent within the cell.

To support aggregation of averages, two bands are stored for the delta heading and delta COG heatmap types. One band represents the sum of change, and the other represents the count of events. The sum of these two bands is then divided to calculate the average.

### C. Spatial Relationship

The transformation from the trajectory representation to the cell representation, as seen in Figure 1, is a multi-step process. The steps are as follows:

1) Split the trajectories into trajectory segments along the borders of the 5km grid. An analogy is to imagine pressing a cookie cutter with 5km squares onto a plate of spaghetti, where the spaghetti represents the trajectories.
2) Assign a spatial division, see Section V, to each of the segments by spatially joining the division geometries from `dim_spatial_division`.
3) Rollup from the segments to each of the four cell granularities.

Steps 2 and 3 are interesting, as the spatial joins require a non-trivial join condition to avoid double-counting and artefacts along division edges. For example, consider a segment that travels on a division's left edge. If an intersect join is used, this segment intersects both the division to the left and the actual division, resulting in double counting.

*Spatial Relationship Between Trajectory Segments and Spatial Divisions:* As the spatial relationship of intersects results in double counting, the spatial join when joining divisions to the segments in step 2 is instead defined as:

$$\begin{aligned} &ST\_Covers(division, segment) \textbf{ AND} \\ &ST\_YMax(division) \neq ST\_YMin(segment) \textbf{ AND} \\ &ST\_XMax(division) \neq ST\_XMin(segment) \end{aligned} \quad (1)$$

This definition of *ST_Covers*[10] is that no point of the segment is outside of the division. As divisions always have at least 5km side lengths, it is guaranteed that a segment is counted in exactly one division, except in the edge case where a segment may be entirely on the line between two divisions. To prevent double counting in this scenario, the remainder of the join conditions ensure that only a division's left and bottom

[10]https://postgis.net/docs/ST_Covers.html

edges are inclusive. For example, suppose a segment follows a horizontal line between two divisions. In this case, it belongs only to the top division, since the segment follows the bottom edge of the top division.

*Spatial Relationship Between Trajectory Segments and Reference Cells:* In step 3, the spatial join follows mostly the same definition as in step 2. Except this time, the *ST_Covers* condition cannot be used since the fine cell granularities are much smaller than the segments. Instead, a combination of *ST_Crosses*, *ST_Contains*, and *ST_Touches* is used, as seen below.

$$\begin{aligned} &ST\_Crosses(cell,\ segment)\ \textbf{OR} \\ &ST\_Contains(cell,\ segment)\ \textbf{OR} \\ &(ST\_Touches(cell,\ segment)\ \textbf{AND} \\ &ST\_XMax(cell) \neq ST\_XMin(segment)\ \textbf{AND} \\ &ST\_YMax(cell) \neq ST\_YMin(segment)) \end{aligned} \tag{2}$$

The *ST_Crosses* or *ST_Contains* conditions catch all cases where a segment is partly in a cell or entirely in a cell, respectively. The third condition uses *ST_Touches* to check if the segment is perfectly aligned along a cell edge, and the rest of the third condition checks if it is along the left or bottom edge to avoid double counting, as with divisions.

### D. Indices

Several indices are created for some DW relations seen in Figure 4 to reduce the query runtime when running analytical queries on DIPAAL.

An SP-GiST index [32] is created on the *geom* attribute in the four cell dimensions. These indices enable bounding box candidate elimination for a range of spatial join conditions.

Two B-tree indices are created for the ship identification attributes *mmsi* and *imo* of `dim_ship`. These reduce the query runtime of analytical queries that look for specific ships by either their *mmsi* or *imo*.

In order to quickly look up and compare trajectories, a GiST index [33] is created on the *trajectory* attribute of `dim_trajectory`. This index reduces the query runtime for spatio-temporal query conditions on trajectories.

To speed up queries involving the lookup of facts for specific trajectories, a composite B-tree index is created for the attributes *start_date_id* and *trajectory_sub_id*, as these are used as the composite primary key of `dim_trajectory`.

SP-GiST indices are created for the *bounding_box* attribute of the four fact cell relations, which help reduce the lookup time for spatio-temporal bound queries directed at the cell representation. As the cell representation is intended as the central feature for analysis in DIPAAL, these indices are considered the most important.

An SP-GiST index is created from the *geom* of `dim_spatial_division`, which helps speed up the check to see which spatial division a trajectory segment belongs to, as described in Section VIII-C.

Besides the above-mentioned indices, b-tree indices are created for the primary key of each relation in DIPAAL, as these are automatically created by PostgreSQL [14, sec. 11.6].

### E. API Implementation

The API endpoints provide pre-optimised queries as an intermediary to the DW. To ease implementation, the FastAPI[11] framework is used. FastAPI is OpenAPI-based, which enforces the API to comply with the OpenAPI specification, and eases the documentation.

Due to the custom data types and the spatio-temporal operators from MobilityDB, it is infeasible to implement the API using Object-Relational Mapping (ORM). Instead, SQL queries are built using a query-builder.

To facilitate rendering PNG images and MPEG-4 videos, as mentioned in Section VI, the heatmap endpoints are implemented using the rasterio[12], imageio[13], and Pillow[14] libraries. Together, these libraries provide the necessary functionality to convert the GeoTiff data result from the DW into the requested format.

### F. Reducing Storage Cost

From investigating the size of the DIPAAL relations, it is found that `fact_cell_50m` is by far the largest relation. The `fact_cell_50m` relation occupies 62% of the storage space occupied by the continuously growing relations[15] when looking at the data for 2021. During a meeting with domain experts, see Appendix A, it was discussed whether the analytics on the 50m cell granularity is interesting in all areas. It was concluded that analytics on 50m cell granularity is only interesting in select areas, mainly harbours, as that level of detail is unnecessary in other areas due to the scale in which ships operate on open waters. Therefore, keeping 50m cell facts for the spatial boundary of DIPAAL is unnecessary and the reduction in storage cost results in more space to load additional AIS data.

One way to reduce the storage cost of the 50m cell fact relation is to remove cell facts outside areas of interest. The definition of cells within Electronic Nautical Charts (ENC)[16] are used to identify potential areas of interest and their geometrical bounds. Two types of ENC cells are considered; "Harbour" and "Approach". The spatial coverage of the Danish set of "Harbour" and "Approach" type ENCs are seen in Figures 10a and 10b, respectively.

Table III estimates how many years can be loaded with the 10TB disk space available to DIPAAL. These estimates are based on the storage cost of loading a single year.

Based on the input from domain experts and because it allows additional years to be loaded into the DW, it is recommended that 50m cell facts are kept only for the harbour

[11]https://fastapi.tiangolo.com/

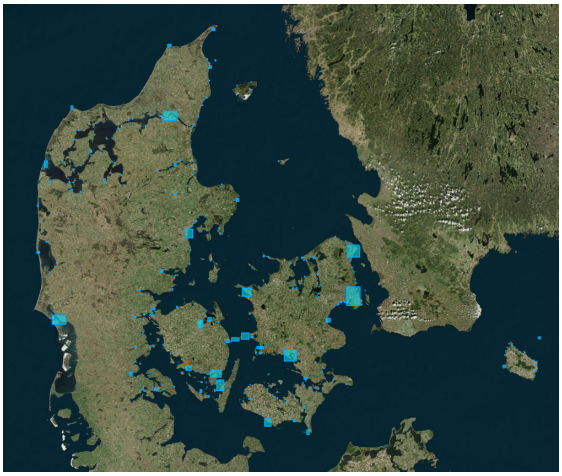
[12]https://pypi.org/project/rasterio/

[13]https://pypi.org/project/imageio/

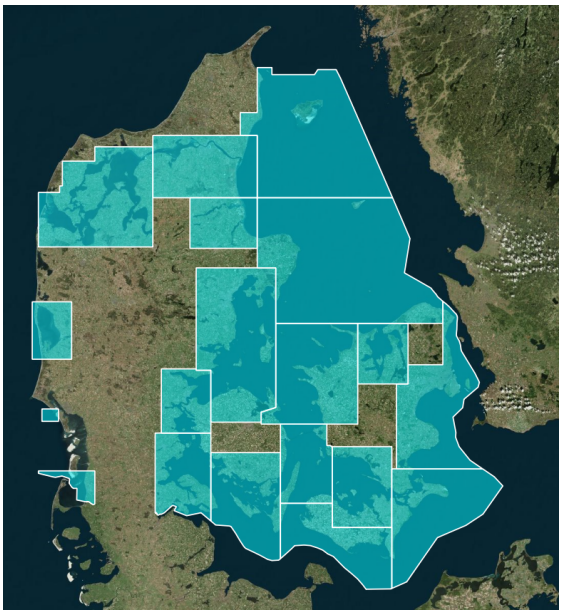
[14]https://pypi.org/project/Pillow/

[15]The continuously growing relations refer to all fact relations, as well as the ship, trajectory, and date dimensions.

[16]ENC data used are from: https://www.geogarage.com/page/catalog



(a) Danish ENCs with type “Harbour”.



(b) Danish ENCs with type “Approach”.

Fig. 10: Candidate ENC types to define where to keep 50m cell facts.

areas. In order to still provide 50m cell facts for non-harbour areas, a dynamic version of the trajectory to cell rollup query is defined, which returns transient cell facts based on the spatial and temporal bounds given.

The removal of 50m cell facts is a responsibility of the final step of the *rollup module*, seen in Figure 3, as the 50m cell data are used for the heatmap rollups.

## IX. Evaluation

This section evaluates the performance of a range of queries on DIPAAL and the data skewness of the spatial distribution approach with new and old data.

| Change | Loadable Years (Estimated) |
|---|---|
| No Change | 6 years |
| Approach Only | 10 years |
| Harbour Only | 19 years |
| Remove All | 21 years |

TABLE III: Estimate of how many years can be loaded into DIPAAL with the current disk space available, based on which 50m cell facts are kept.

The evaluation focus on query runtime in a single-user scenario and thus does not provide any evaluation of a concurrent multi-user scenario.

### A. Hardware/Software

The DW cluster of DIPAAL is a Citus cluster and consists of six identical machines in a setup of one coordinator and five workers. The cluster is configured without high availability since the primary focus of DIPAAL is improving performance.

Each machine has an 8-core, 16-thread AMD Ryzen 7 5800x processor with a base clock speed of 3.8Ghz. Each machine has 32GB of DDR4 system memory and a 2TB nVME Solid State Drive. The machines are connected through Ethernet on a gigabit connection.

Each machine runs a containerised instance of PostgreSQL 15.3 with the extensions PostGIS 3.3.2, MobilityDB 1.1.0 (unreleased; forked[17] from the develop branch on April 22th with cherry-picked fixes, 2023), and Citus 11.3.0 in Kubernetes [34]. Both the container base image and host OS is Ubuntu 22.04. The evaluation uses Python 3.11.

### B. Data Foundation

The dataset, which DIPAAL is evaluated upon, is the publicly available AIS dataset published by DMA[18]. The year 2021 is loaded for all benchmarks unless otherwise stated in the benchmark section.

### C. Runtime Evaluation Metrics and Procedure

Evaluations that measure runtime all use two metrics; runtime and average Worker Idle Fraction (WIF)

The runtime is the duration from sending the query to the DW to receiving a complete result. WIF is defined as the fraction of the runtime a worker spends idling. A high WIF indicates that the computational power of the worker is wasted. WIF is calculated as defined in Equation (3)

$$\text{WIF}_j = \frac{\max\left(\text{Worker Time}\right) - \text{Worker Time}_j}{\max\left(\text{Worker Time}\right)} \tag{3}$$

In Equation (3), $\text{Worker Time}_j$ is the time that Worker $j$ took to finish all shards located on that worker and is defined in Equation (4)

$$\text{Worker Time}_j = \max_{i \in \text{Worker Shards}_j} \text{Shard Time}_i \tag{4}$$

[17]https://github.com/DIPAAL/MobilityDB/tree/fix-srid-binary-representation-stbox

[18]https://web.ais.dk/aisdata/

Average WIF has a lower bound of 0%, but an upper bound of $1 - \frac{1}{|workers|}$. The upper bound is due to a single worker always being non-idling. For this reason, the average WIF is normalized to a range from 0% to 100%, as seen in Equation (5)

$$\text{Average WIF} = \frac{\sum WIF}{|workers|} * \frac{1}{1 - \frac{1}{|workers|}} \tag{5}$$

Each query is evaluated through ten iterations in an effort to remove outliers from the result. The metrics are then aggregated using a trimmed mean, by removing the two lowest and two highest results, before taking the mean [35].

To facilitate consistent results, all PostgreSQL instances in the cluster are restarted, and the OS file cache is cleared before each query iteration. The cache is then pre-warmed by running random queries, as explained in Appendix C-A, before the evaluation query is executed.

Multiple designated areas are tested as part of the run time benchmarks, which are found in Figure 11.

### D. Heatmap Evaluation

This section covers the heatmap performance evaluation of DIPAAL. A detailed benchmark plan is found in Appendices C-A and C-B. The heatmap benchmark serves two purposes; To find the best performing physical storage scheme and to evaluate the scale up when comparing a one-worker setup to a five-worker setup.

Three physical storage schemes are evaluated; Row-based storage, Citus Columnar, and Partitioned Citus Columnar. The Partitioned Citus Columnar storage scheme uses Citus Columnar but is partitioned on heatmap type, resolution, and month, instead of global monthly partitions. The Partitioned Citus Columnar scheme aims to reduce the Columnar runtime, by storing data that are likely to be queried together in the same stripe.

These three physical storage schemes are evaluated on the five-worker setup. The best performing physical storage scheme is evaluated on the one-worker setup to compare the scale up. This results in four configurations.

The evaluation of a configuration consists of a series of 36 queries. The 36 queries are the cross product of the four cell granularities, three temporal spans, and three spatial areas, and are seen in Figure 12. The three temporal spans are 1 day (February 28th, 2021), 30 days (January 26th through February 24th, 2021), and 1 year (2021). The three spatial areas are seen in Figure 11, with Aarhus Harbour ($46.78km^2$) in red (3), The Great Belt (3 $071km^2$) in green (7), and the spatial domain of DIPAAL (725 $725km^2$) in blue (2).

$$\begin{Bmatrix} Aarhus\ Harbour \\ The\ Great\ Belt \\ Spatial\ Domain\ of\ DIPAAL \end{Bmatrix} \times \begin{Bmatrix} 1\ day \\ 1\ month \\ 1\ year \end{Bmatrix} \times \begin{Bmatrix} 5000m \\ 1000m \\ 200m \\ 50m \end{Bmatrix}$$

Fig. 12: Overview of the 36 heatmap configurations to be evaluated.

The results of the evaluation of heatmaps are seen in Table IV. The 1 day queries are not included as they are always below one second and are thus uninteresting. Likewise, the resolutions of 1000m and 5000m are not included due to space constraints and the finer granularities being more interesting.

*Evaluation of Physical Storage Scheme:* In Table IV, the fastest configuration for each query is marked in bold. It is noticeable that the five-worker setup with row-based storage is the fastest at the majority of the queries, besides the queries with a runtime at or below one second, where the difference is insignificant.

As it is more important to reduce the runtime of the slow queries, the row-based physical storage scheme is chosen as the physical storage scheme used in DIPAAL. The row-based physical storage scheme is chosen as it has the best performance across all queries with a significant runtime.

*Evaluation of Scale Up:* The row-based physical storage scheme is applied to a one-worker setup and compared to the row-based physical storage scheme of a five-worker setup. This result is seen in the right-most column in Table IV. The scale up is measured in percent, with 100% indicating the same runtime and 200% indicating a runtime two times faster on the five-worker setup.

It is seen in Table IV, that the scale up varies significantly. This variation is expected as smaller areas only engage a few shards, which is also indicated by the high WIF in the "Aarhus Harbour" query. It is seen that the WIF is lower the larger the area queried, as more shards are engaged in calculating the results. In the "Aarhus Harbour" area, five shards are engaged; likewise in the "The Great Belt" area, 15 shards are engaged. In the "spatial domain of DIPAAL " area, all 400 shards are engaged.

Even with a low scale up on the small areas, this is deemed satisfactory as the small areas' runtimes are fast enough without a significant scale up. The scale up is consistently good in the spatial domain of DIPAAL query, where it is hovering around 400%. This is a satisfactory result, as the desire to scale up is higher on the slowest queries.

On the queries with a lower runtime, i.e. the queries in the areas of Aarhus Harbour and the Great Belt, on the five-worker setup with row configuration, the WIF is consistently lower for 200m heatmaps than 50m. Likewise, it is observed that the scale up is worse for 50m than 200m heatmaps. By investigating the explain analyse dump, it is clear this is due to shard imbalance. For example, in the one year Great Belt query, 14 shards are involved in the calculation. The largest shard holds 2.4GB of raster data, while the next-largest holds 1.9GB of raster data, i.e. there is a 26% size difference. By investigating the query plan and the individual shard timings, it is observed that the runtime of the spatial aggregation of the rasters is linearly proportional to the sum of pixels in the shard. Thus, a larger time span is expected to be spent during spatial aggregation in the large shards. This imbalance is relatively equal in the 200m query with a 24% size difference between the largest shard holding 170MB of raster data and the next-largest 137MB of raster data. However, as the absolute size

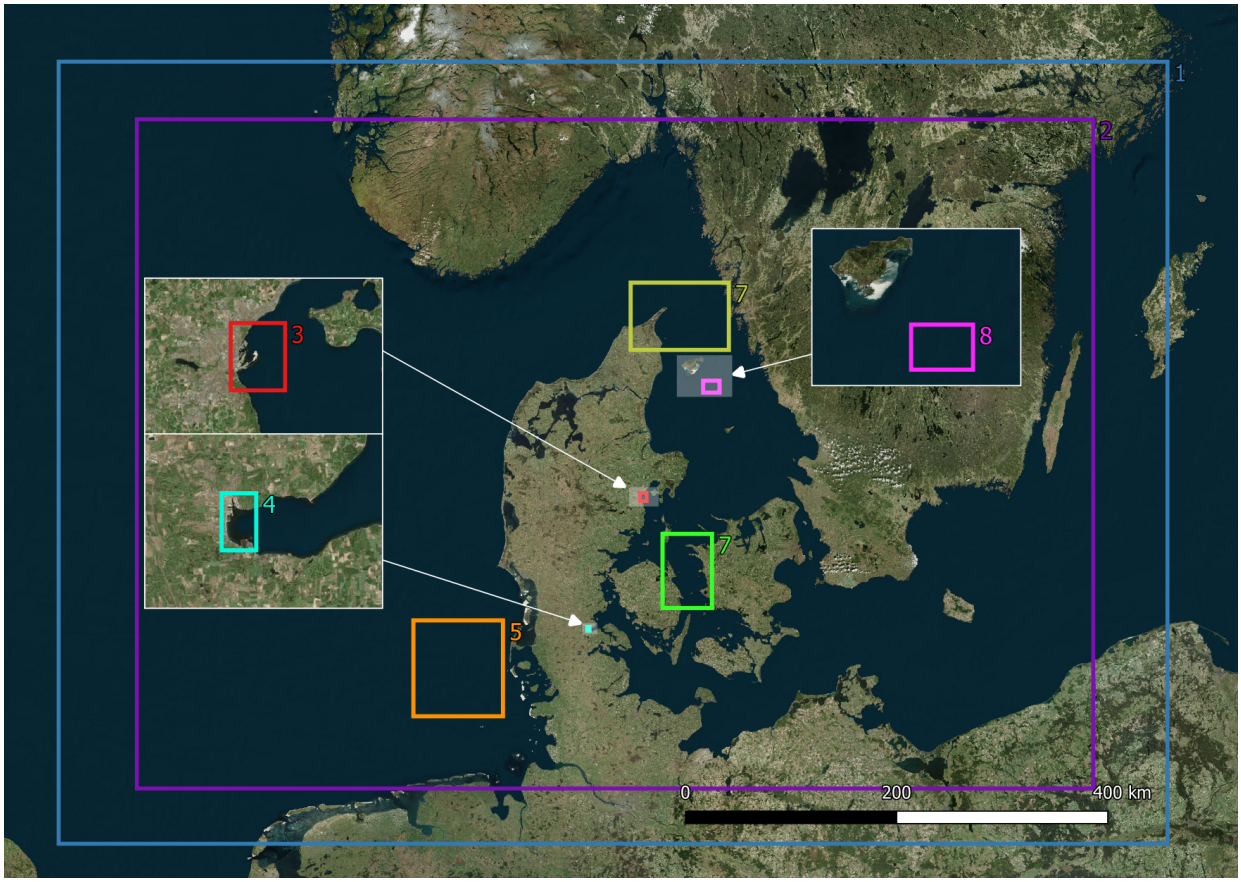



Fig. 11: The eight spatial areas evaluated as parts of the runtime benchmarks.

difference is much lower, it introduces less runtime imbalance relative to other query plan nodes and thus results in a lower WIF and higher scale up.

### E. Cell Evaluation

This section evaluates the performance difference of running spatio-temporal range queries on the cell representation versus the trajectory representation. This evaluation is run on both the one- and five-worker setups to evaluate the scale up. A more detailed plan for this benchmark is found in Appendices C-A and C-C.

For this evaluation, a number of queries are evaluated, made up of 12 spatio-temporal ranges, which is the cross product of four spatial areas and three temporal spans. The three temporal spans consist of a single day (January 10th, 2021), 30 days (January 26th through February 24th, 2021), and 90 days (January 1st through March 31st, 2021). The four spatial areas are seen in Figure 11, with the Aabenraa area ($6.6km^2$) in teal (4), Aarhus Harbour ($46.78km^2$) in red (3), Skagerrak ($5529km^2$) in Yellow (6), and danish seaborders ($535\ 557km^2$) in purple (2). The spatio-temporal parameters are seen in Figure 13

$$\left\{\begin{matrix} Aabenraa \\ Aarhus\ Harbour \\ Skagerrak \\ Danish\ Seaborders \end{matrix}\right\} \times \left\{\begin{matrix} 1\ day \\ 30\ days \\ 90\ days \end{matrix}\right\}$$

Fig. 13: Overview of 12 spatio-temporal parameters of the cell evaluation.

Additionally, the cell queries are repeated on multiple levels of the cell hierarchy to evaluate how the cell granularity impacts the runtime on different spatio-temporal ranges. Not all cell granularities are run on all areas, as this was proven infeasible in previous work [1]. The details of which granularities are used for each area are found in Appendix C-C and Table VIII.

Figure 14 shows the runtime results of the cell evaluation for the four defined areas. It is seen from the figure that the five-worker setup mostly outperforms the one-worker setup, though to a varying degree. An example of this variation is seen in Figure 14a for the 1 day trajectory query between the one- and five-worker setup, where the five-worker setup is more than twice as slow as the one-worker setup.

From investigating the query plans, the cause is narrowed down to network transmission. Out of the 10 query iterations, six of the results have a fast execution time below 555ms,

| | | | Five-worker | | | | | | One-worker | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Row | | Columnar | | Columnar partitioned | | Row | |
| | | | Time (s) | WIF (%) | Time (s) | WIF (%) | Time (s) | WIF (%) | Time (s) | Scale Up (%) |
| Aarhus Harbour | 30 day | 200m | 0.37 | 27 | **0.35** | 64 | 0.35 | 47 | 0.59 | 160 |
| | | 50m | 0.36 | 65 | 0.34 | 66 | **0.34** | 53 | 0.55 | 150 |
| | 1 year | 200m | **0.92** | 56 | 0.94 | 80 | 1.89 | 35 | 6.37 | 693 |
| | | 50m | 1.10 | 68 | **1.01** | 82 | 2.83 | 42 | 4.69 | 424 |
| The Great Belt | 30 day | 200m | **0.36** | 21 | 0.47 | 22 | 0.36 | 19 | 0.93 | 258 |
| | | 50m | **1.17** | 39 | 1.36 | 37 | 1.28 | 43 | 1.80 | 154 |
| | 1 year | 200m | **2.19** | 20 | 3.36 | 27 | 2.53 | 23 | 7.68 | 350 |
| | | 50m | **12.50** | 37 | 14.4 | 38 | 13.74 | 42 | 18.17 | 145 |
| Spatial Domain of DIPAAL | 30 day | 200m | **1.27** | 19 | 4.17 | 18 | 1.58 | 15 | 8.09 | 639 |
| | | 50m | **11.90** | 13 | 14.4 | 14 | 12.06 | 14 | 42.10 | 354 |
| | 1 year | 200m | **9.18** | 9 | 35.58 | 16 | 9.36 | 12 | 40.19 | 438 |
| | | 50m | **102.35** | 11 | 126.37 | 10 | 102.81 | 10 | Out of memory | |

TABLE IV: Heatmap Benchmark Results.

whereas the remaining three iterations have a slow execution time above 5700ms. When comparing one of the slow and one of the fast iterations, it is found that the difference between the longest running Citus task is minimal, with $\sim 10.8ms$ for the slow iteration and $\sim 10.4ms$ for the fast iteration. The remaining Citus tasks are insignificant, as their runtimes are roughly the same, as only one shard has any data for the result. After receiving the 142 bytes of data from the Citus task, the coordinator node's final work takes 1.7ms for the slow iteration. Therefore, the root cause of these outliers must be a delay in the network data transmission between the workers and the coordinator for some iterations. Similar observations are observed for other results as well.

In general, the runtime results are as expected; the increase in runtime is directly proportional to the size of the spatio-temporal bounds.

Table V shows a selected set of results from the cell evaluation. To remove any bias, the selected results include some of the best and worst results for a given metric. In Table V, the best of the selected results are highlighted in bold, while the worst results are underlined.

As expected, both the lowest scale up and highest WIF are found for the area that occupies only a single shard. The WIF is high on these queries because the data for the result is stored on a single worker. This results in one worker calculating the result, while the other workers ensure they contain no data for the result and then idle. For the same reason queries for the single shard area (Aabenraa) also have the worst scale up, as the difference between running it on single versus five workers are insignificant.

As expected the highest scale up and the lowest WIF are found for the largest areas queried, i.e. Skagerrak and spatial domain of DIPAAL. The high scale up is explained by these areas intersecting a large number of shards, thereby ensuring that all workers are utilised and, as a result, the idle time is reduced.

Table V shows that the Skagerrak area of the trajectory type has the highest scale up of 992.55%. The high scale up is partly explained by the random distribution used to distribute trajectories. The random distribution provides a better balanced distribution of the work between the workers that allow faster calculation of the query result. Since the scale up is super linear, it is also likely that the higher system cache and memory available on the five-worker setup contributes to the scale up, in a way that is not present in other queries.

From the numbers seen in Table V, it is evident that the 90 days Danish Sea Borders query has a 324% runtime improvement when queried on the 5000m cells compared to the trajectory-based query. This improvement shows that the granularized cell representation can significantly improve runtimes compared to the more complex trajectories.

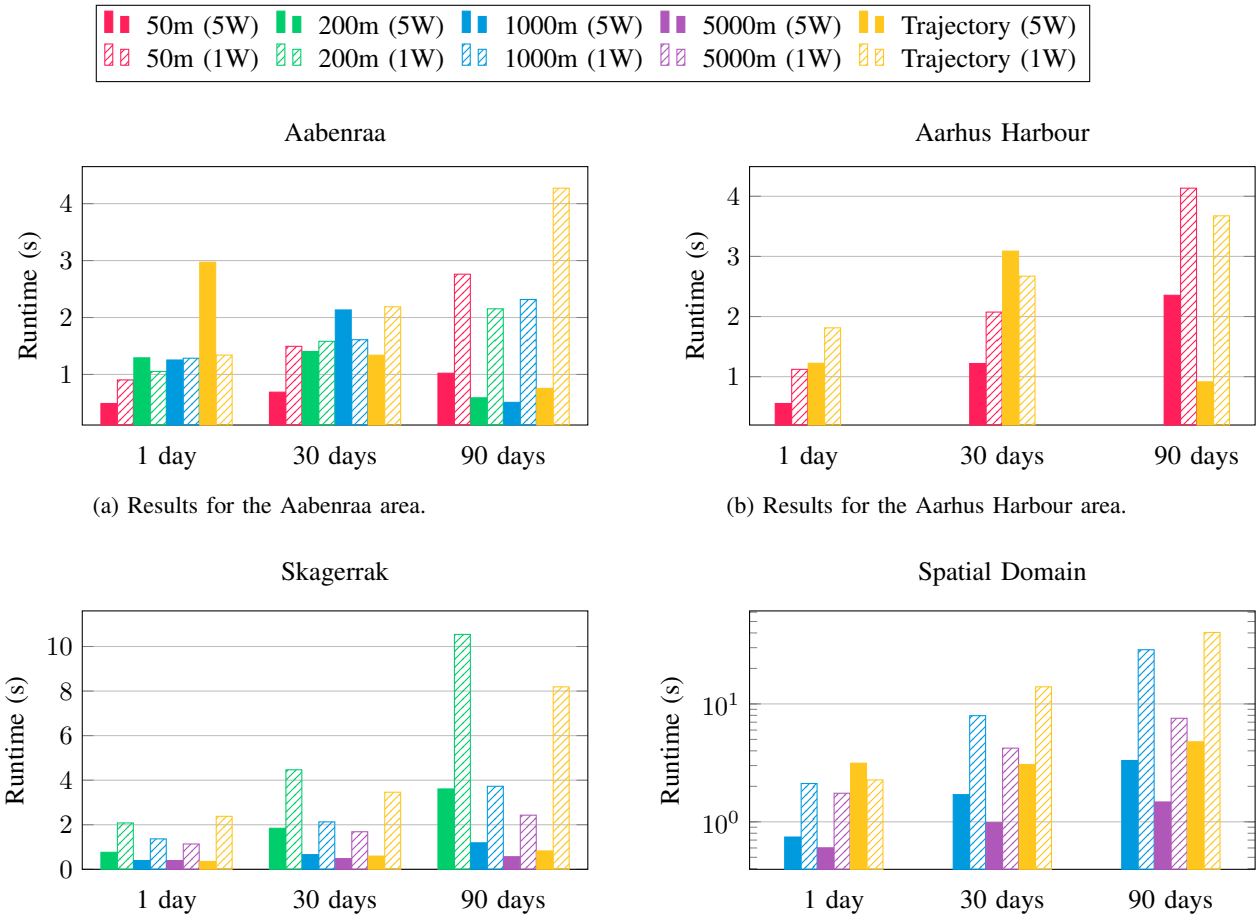
### *F. Lazy Versus Eager 50m Cell Calculation*

This evaluation serves as proof of the viability of deleting 50m cell facts outside of ENC harbours, as described in Section VIII-F, by evaluating the runtime if 50m cells are requested in areas where they are deleted.

Two types of queries are compared. Both are spatio-temporal range queries of 50m cell facts. However, one is eagerly evaluated by querying `fact_cell_50m`, and the other is lazily evaluated by querying `fact_trajectory` and constructing transient cell facts on demand.

These two query types are executed with six spatio-temporal ranges, which are the cross product of two spatial areas, and three temporal spans. The two spatial areas are "Near Heligoland" (7 $189km^2$) and "South of Laesoe" ($164km^2$), and are seen as the orange (5) and the pink box (8) in Figure 11, respectively. The three temporal spans serve to evaluate a range of temporal spans and are 1 day (May 4th, 2021), 1 week (June 21st through 28th, 2021), and 1 month (October, 2021).

These six spatio-temporal ranges are evaluated for both query types, and on both the one- and five-worker setup. This results in a total of 24 queries, and are shown on Figure 15




(a) Results for the Aabenraa area.

(b) Results for the Aarhus Harbour area.

(c) Results for the Skagerrak area.

(d) Results for the DIPAAL spatial domain.

Fig. 14: The cell benchmark runtime results for the one- and five-worker setups.

| | | | Scale Up (%) | Five-worker runtime (s) | One-worker runtime (s) | WIF (%) |
|---|---|---|---|---|---|---|
| Cell 1000m | 1 day | Aabenraa | 103 | 1.25 | **1.29** | 35 |
| Cell 1000m | 30 days | Danish Sea Borders | 467 | 1.70 | 7.95 | **13** |
| Cell 50m | 90 days | Aabenraa | 270 | 1.02 | 2.76 | 65 |
| Cell 200m | 90 days | Aabenraa | 365 | **0.59** | 2.15 | 38 |
| Cell 1000m | 90 days | Danish Sea Borders | 870 | 3.31 | 28.82 | 13 |
| Cell 5000m | 90 days | Danish Sea Borders | 583 | 1.47 | 7.55 | 14 |
| Trajectory | 90 days | Skagerrak | **993** | 0.83 | 8.19 | 14 |
| Trajectory | 90 days | Danish Sea Borders | 858 | 4.78 | 40.37 | 13 |

TABLE V: Selected cell benchmark results.

$$\left\{\begin{matrix}Near\ Heligoland\\South\ of\ Laesoe\end{matrix}\right\} \times \left\{\begin{matrix}1\ day\\1\ week\\1\ month\end{matrix}\right\} \times \left\{\begin{matrix}lazy\\eager\end{matrix}\right\} \times \left\{\begin{matrix}five\ workers\\one\ worker\end{matrix}\right\}$$

Fig. 15: Overview of 24 queries of the lazy cell evaluation.

It is, as expected, much faster to query pre-calculated results, as seen in Table VI. The speed-up measures how much faster the eager query is compared to the lazy query. It is seen the speed-up ranges from 737% through 4177%. It is also noteworthy that the lazy evaluation scales better horizontally than the eager evaluation. The better scale up is due to the lazy query querying `fact_trajectory`, which is randomly distributed, compared to the spatial distribution of `fact_cell_50m`. Since the areas queried are small, the spatial distribution does not allow for as large a scale up. This also indirectly means, the lazy query induces a much higher worker utilisation, and thus DW load.

Lastly, a key takeaway is that the lazy query is shown to finish within 14 seconds with the given parameters. This result

| | | | Five-worker | | One-worker | |
|---|---|---|---|---|---|---|
| | | | Time | WIF | Time | Scale Up |
| Near Heligoland | 1 day | eager | 0.23s | 72% | 0.44s | 188% |
| | | lazy | 9.68s | 20% | 39.63s | 409% |
| | | speed-up | | 4177% | | 9077% |
| | 1 week | eager | 1.8s | 70% | 2.38s | 130% |
| | | lazy | 13.4s | 20% | 57.95s | 430% |
| | | speed-up | | 737% | | 2430% |
| | 1 month | eager | 1.36s | 57% | 1.94s | 142% |
| | | lazy | 13.49s | 20% | 59.62s | 443% |
| | | speed-up | | 987% | | 3071% |
| South of Laesoe | 1 day | eager | 0.15s | 73% | 0.20s | 137% |
| | | lazy | 1.33s | 8.2% | 4.90s | 369% |
| | | speed-up | | 900% | | 2429% |
| | 1 week | eager | 0.15s | 65% | 0.32s | 218% |
| | | lazy | 2.26s | 15% | 9.27s | 411% |
| | | speed-up | | 1560% | | 2936% |
| | 1 month | eager | 0.16s | 55% | 0.27s | 168% |
| | | lazy | 4.62s | 16% | 19.77s | 428% |
| | | speed-up | | 2837% | | 7219% |

TABLE VI: Lazy versus eager cell fact evaluation results.

provides a foundation for the argument that deleting 50m cell facts for non-harbour areas is feasible, as it is a non-frequent query to request 50m cells in areas outside points of interest, as argued in Section VIII-F.

### G. Spatial Distribution Skew

The spatial distribution skew benchmark differs from the other benchmarks by evaluating how skewed the spatial distribution, defined in Section V, becomes over time rather than measuring runtime.

To measure the skewness, Standard Deviation (SD), as well as Coefficient of Variance (CV) are used, as introduced in Section V, where lower values for both metrics indicates better balanced shards. The metrics are measured for each of the four cell granularities regarding the count of cell facts and the sum of the size of heatmap rasters. The evaluation is performed for 2011, 2021, and 2022, and is shown in Table VII. The year 2021 is chosen as the reference year, which the spatial divisions are built upon since it was the last fully available year at the beginning of the development. 2022 is chosen to measure the skew in a consequent year, and 2011 is chosen to get a measurement of the skew across a large temporal gap, i.e. ten years.

In Table VII, it is seen that the CV of both the count of cell facts and the size of heatmap rasters for all granularities are higher in 2011 and 2022 than in 2021. It is also seen that 2022 is much closer to the 2021 measurement than 2011, which is to be expected, as traffic patterns are more likely to have changed over ten years versus one year. The evaluation shows that querying data of recent years is likely faster than querying older data.

Another observation is that the CV is generally higher in the finer cell granularities. The higher CV is caused by the spatial divisions being built on the foundation of 5000m cells.

## X. Related Work

With more and more devices generating trajectory data [36], systems handling trajectory data must scale with the growing

| | | 2011 | | 2021 | | 2022 | |
|---|---|---|---|---|---|---|---|
| | | SD | CV | SD | CV | SD | CV |
| Cell | 50m | 3359k | 66% | 2720k | 50% | 2734k | 51% |
| | 200m | 836k | 65% | 671k | 49% | 6756k | 50% |
| | 1000m | 164k | 61% | 127k | 43% | 129k | 45% |
| | 5000m | 36k | 56% | 32k | 41% | 34k | 44% |
| Heatmap | 50m | 2388MB | 77% | 2002MB | 61% | 1997MB | 61% |
| | 200m | 164MB | 73% | 136MB | 56% | 137MB | 57% |
| | 1000m | 18MB | 56% | 14MB | 40% | 15MB | 42% |
| | 5000m | 11MB | 52% | 9MB | 38% | 9MB | 40% |

TABLE VII: Spatial Distribution Skewness.

volume of data. Likewise, as the amount of data increases, these systems must implement efficient algorithms to query the trajectory data.

Efficiently processing and storing trajectory data has been an area of research for many years, but it is not purely research-oriented, with some systems being used in the industry [37]. These systems differ in how they handle trajectory data, with some systems using a DW approach [9, 38] and others using a more general trajectory management approach [37, 10, 11, 39].

The underlying storage system varies between using an RDBMS [9], NoSQL databases [37, 39], and processing frameworks based on either in-memory storage [11] or storage in distributed file systems [10]. After discussions with domain experts, it was ultimately chosen that DIPAAL is based on the PostgreSQL RDBMS, which is distributed using the Citus extension.

DIPAAL uses the trajectory-based storage model, storing the full trajectories as a single row. Other systems that utilise or support the trajectory-based storage model include Lan et al. [37], Nielsen et al. [9], and Li et al. [39]. Additional storage models are also available, such as the point-based model [37, 9], which stores each point of a trajectory individually, and the segment-based model [37, 10, 11], which stores trajectories in segments. The trajectory-based model is chosen for DIPAAL as it incurs no reconstructing cost during query execution and has reduced storage cost compared to the point-based model [37]. Segmentation of trajectories is used during the *rollup* step of the ETL process, as the reduced MBRs increases the performance of the *rollup* step as no reconstruction is needed.

DIPAAL uses spatial distribution to reduce the runtime when querying data based on the underlying cell grid using Citus shards to distribute and parallelise queries. DIPAAL opted to create the spatial divisions utilising a kd-tree approach from one year's worth of data, whereas other approaches were adopted by similar systems, such as Li et al. [26] and Aji et al. [40]. Li et al. [26] chose to use a quad-tree-based approach to construct their divisions, as they divide non-point data, which is more difficult to find kd-splits for [26]. Aji et al. [40], instead use a custom grid division approach based on thresholds, where divisions are continuously split as long as the amount of data within a split is above the threshold. The spatial distribution approach used in DIPAAL can result in some divisions spanning very large areas. However, compared to Aji et al., it is ensured that the largest divisions are split

first, and all divisions created contain data, enabling a better balance of data across the cluster.

The static spatial distribution approach of DIPAAL results in the amount of data stored in the shards becoming skewed over time. This approach is opposed to the spatio-temporal distribution approach proposed by Li et al., which results in changing spatial divisions over time. Spatio-temporal distribution solves the problems of divisions becoming skewed over time, but for DIPAAL it results in an increase in runtime as spatial aggregation across temporal divisions can no longer be pushed down to the workers. Both Li et al. and Aji et al. use global indices to look up which shards contain candidate data. DIPAAL, instead relies on the shards' local indices to quickly evaluate whether a shard contain relevant data for a query.

The API for DIPAAL consists of endpoints which all are GET HTTP-method endpoints, with separate endpoints for querying the core elements of DIPAAL. Looking at other AIS data APIs used in production, namely MarineTraffic [41] and VesselFinder [42], they support requests for live and historical information about ships and events surrounding ships. DIPAAL does not support live data but supports querying trajectory, cell, and heatmap representations of AIS data, while MarineTraffic and VesselFinder only support AIS point data. MarineTraffic and VesselFinder both support JSON as an output format. Similarly, JSON is one of the supported output formats of DIPAAL.

Various heatmaps are generated as part of DIPAAL's ETL process. DIPAAL utilises a binning approach similar to Liu, Jiang, and Heer [43] by creating small heatmaps based on the 5km cell grid and spatial divisions, which can then be combined into a variety of heatmaps.

## XI. Conclusion

The DIPAAL platform consists of a modular ETL pipeline which enhances the data quality through well-defined cleaning rules, as well as running the cleaned AIS data through a transformation process. The transformation process starts by creating simplified trajectories from the AIS point data that are then aggregated into cells, and finally into heatmaps. The ETL process's cleaning rules and modularity enables DIPAAL to be applied to any AIS dataset with minimal changes. DIPAAL runs on a cluster of six commodity machines, with the data preparation part of the ETL process capable of running asynchronously and independently of the DW.

This paper presents an improved version of DIPAAL. First and foremost, the DW design is updated with a new heatmap representation of the AIS data, as well as modelling the cell representation as a snow-flake hierarchy of multiple granularities. The snow-flake hierarchy reduces the replication of the coarsest granularity by 10 000 times compared to a denormalized star-schema. Choosing a coarser cell granularity for large areas shows that previous out-of-memory errors are eliminated and up to 324% improvement in runtime compared to the trajectory-based query.

Secondly, a spatial distribution approach is used for the raster-based representations, i.e. cells and heatmaps. The spatial distribution approach uses a kd-tree-based approach to statically divide the spatial domain of DIPAAL into shards, which stores the cells and heatmaps within each division.

The kd-tree-based approach proves to balance the AIS data the best. The dynamic splits of the kd-tree approach find better splits than the fixed splits of the quad-tree approach, due to the unevenness of the data foundation. The Coefficient of Variance (CV) of the kd-tree-based built approach with 400 partitions on the year 2021 is 41%, compared to the 106% of the quad-tree-based approach.

The spatial distribution ensures data locality and enables spatial aggregates to be pushed down. The pushed down of spatial aggregates enables horisontal scaling and thus reduces the query runtime. For example, a 200m resolution heatmap of the year 2021 for the spatial domain of DIPAAL ($725\ 725 km^2$) shows a 438% increase in runtime on a single-worker setup compared to a five-worker setup, with runtimes of 40.19 and 9.18 seconds, respectively.

Evaluation of the spatial distribution show that static divisions result in the balance between shards becoming skewed over time. The divisions built on 2021 are compared with the 2011 and 2022 data, with larger skews observed the larger the temporal distance from the reference year used to create the divisions. Furthermore, it is found that the utilisation of the workers and scale up are higher when querying large areas ($> 5\ 000 km^2$), as large areas generally intersect with more spatial divisions, resulting in multiple workers being engaged in calculating the result. Inherently, this results in the slowest queries scaling the best.

Finally, in collaboration with domain experts, it is found that a significant reduction in the storage space used can be achieved by removing cells from the finest cell granularity, i.e. 50m, outside areas of interest. The evaluation shows that it is viable to lazily calculate cell facts with a maximum runtime of 13.49 seconds on a five-worker setup. Although there is a significant difference compared to querying eagerly stored 50m cell facts, it is still considered viable as querying the 50m granularity outside small areas of interest, such as harbours, is an infrequent query.

## XII. Future Work

This section proposes future work to improve aspects of DIPAAL.

### A. Split Trajectories

As mentioned in Section VIII-B, querying large trajectories result in a poor filter performance of an r-tree-based index. An improvement could be to split trajectories into segments to improve the bounding box index filtering significantly.

DIPAAL experimented with splitting trajectories using the MobilityDB function *spaceSplit*, and then combine the resulting segments into the MobilityDB type *tgeompoint_seqset*, i.e. a set of sequences of geometric points. Storing the segments of a trajectory in a set enables easy reconstruction of the segments and allows the segments to be stored without changing the DW design.

This experimentation discovered that the r-tree based index does not benefit from the segmented trajectories. This is because all the segments of a trajectory reside in a single row of the `dim_trajectory` relation, and PostgreSQL GiST index not supporting multiple bounding boxes per row[44]. To circumvent this, either a DW redesign of the trajectory dimension is needed, with each trajectory segment being a row, or PostgreSQL needs to be extended with multi-row GiST indexing, such as the early work of MGiST[19].

Further experimentation weighing the advantages and consequences is needed to determine whether a restructuring of the trajectory dimension is feasible.

### B. Multi-User Evaluation

The evaluation of DIPAAL evaluated the performance of DIPAAL in a single-user scenario. While the work of this paper evaluated the isolated performance of the DW, it is interesting to evaluate how the DW performs with multi-user concurrent access in the future.

A multi-user concurrent evaluation is especially interesting regarding the trajectory versus cell queries evaluated in Section IX-E. Due to the random distribution of the trajectory queries, it can use all the compute resources in the cluster, resulting in the trajectory-based query slightly beating the cell-based queries on small areas. However, the cell-based query is significantly more efficient in regards to compute resources, and thus would outperform the trajectory-based queries in a concurrent multi-user scenario.

## Acknowledgements

Thanks to the two representatives, Ove Andersen and Jonas Madsen from the Danish Geodata Agency, whom helped as external domain experts. Their contribution and feedback during meetings provided a deeper insight into the domain of maritime navigation, which helped shape the development and contributions of DIPAAL.

Thanks to Esteban Zimanyi and the rest of the MobilityDB team[20] at Université libre de Bruxelles, for facilitating the work of this paper by supporting and solving issues faced while using MobilityDB.

## References

[1] Alex Skov Klitgaard, Lau Ernebjerg Josefsen, and Mikael Vind Mikkelsen. *DIPAAL: DIstributed PostgreSQL-based AIS Analytics and Loading*. Aalborg University. 2023. URL: https://projekter.aau.dk/projekter/da/studentthesis/dipaal-distributed-postgresqlbased-ais-analytics-and-loading(a2beff25-1e33-46c3-a0a5-491c12f227e0).html (visited on 04/14/2023).

[2] *PostgreSQL: The world's most advanced open source database*. Nov. 30, 2022. URL: https://www.postgresql.org/ (visited on 11/30/2022).

[3] *Citus Data*. Accessed 20/10/2022. 2022. URL: https://www.citusdata.com/.

[4] *PostGIS*. Accessed 20/10/2022. 2022. URL: https://postgis.net/.

[5] *MobilityDB*. Accessed 20/10/2022. 2022. URL: https://www.mobilitydb.com/.

[6] Ties Emmens et al. "The promises and perils of Automatic Identification System data". In: *Expert Systems with Applications* 178 (2021), p. 114975. ISSN: 0957-4174. URL: https://doi.org/10.1016/j.eswa.2021.114975.

[7] *Possible third blast at the same time as the second blast*. Accessed on 02/06/2023. 2023. URL: https://eng.geus.dk/about/news/news-archive/2022/september/newspage.

[8] Zhe Xiao et al. "Traffic Pattern Mining and Forecasting Technologies in Maritime Traffic Service Networks: A Comprehensive Survey". In: *IEEE Transactions on Intelligent Transportation Systems* 21.5 (2020), pp. 1796–1825. DOI: 10.1109/TITS.2019.2908191.

[9] Kasper Suamchiang Hvitfeldt Nielsen et al. "Analyzing Billions of AIS Records in a Multi-Purpose AIS Data Warehouse". MA thesis. Aalborg University, 2022. URL: https://projekter.aau.dk/projekter/en/studentthesis/analyzing-billions-of-ais-records-in-a-multipurpose-ais-data-warehouse(bddd4314-a197-45b3-8f05-68184d92e312).html (visited on 05/20/2022).

[10] Mohamed Bakli, Mahmoud Sakr, and Taysir Hassan A. Soliman. "HadoopTrajectory: a Hadoop spatiotemporal data processing extension". In: *Journal of Geographical Systems* 21.2 (June 2019), pp. 211–235. ISSN: 1435-5949. URL: https://doi.org/10.1007/s10109-019-00292-4.

[11] Zhigang Zhang et al. "TrajSpark: A Scalable and Efficient In-Memory Management System for Big Trajectory Data". In: *Web and Big Data*. Ed. by Lei Chen et al. Cham: Springer International Publishing, 2017, pp. 11–26. ISBN: 978-3-319-63579-8.

[12] Konstantina Bereta, Konstantinos Chatzikokolakis, and Dimitris Zissis. "Maritime Reporting Systems". In: *Guide to Maritime Informatics*. Cham: Springer International Publishing, 2021, pp. 3–30. ISBN: 978-3-030-61852-0. URL: https://doi.org/10.1007/978-3-030-61852-0_1.

[13] Pramod J Sadalage and Martin Fowler. *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[14] The PostgreSQL Global Development Group. *PostgreSQL 15.1 Documentation*. Accessed 19/12/2022. 2022. URL: https://www.postgresql.org/docs/15/index.html.

[15] A. Graser. "An exploratory data analysis protocol for identifying problems in continuous movement data". In: *Journal of Location Based Services* 15.2 (2021), pp. 89–117. eprint: https://doi.org/10.1080/17489725.2021.1900612. URL: https://doi.org/10.1080/17489725.2021.1900612.

[19] https://github.com/mschoema/mgist

[20] https://github.com/orgs/MobilityDB/people

[16] Citus Data. *Citus Documentation — Citus 11.1 documentation*. Accessed on 15/12/2022. 2022. URL: https://docs.citusdata.com/_/downloads/en/v11.1/pdf/.

[17] Umur Cubukcu et al. "Citus: Distributed PostgreSQL for Data-Intensive Applications". In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD '21. Virtual Event, China: Association for Computing Machinery, 2021, pp. 2490–2502. ISBN: 9781450383431. URL: https://doi.org/10.1145/3448016.3457551.

[18] Mohamed Bakli, Mahmoud Sakr, and Esteban Zimanyi. "Distributed Moving Object Data Management in MobilityDB". In: *Proceedings of the 8th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. BigSpatial '19. Chicago, Illinois: Association for Computing Machinery, 2019. ISBN: 9781450369664. URL: https://doi.org/10.1145/3356999.3365467.

[19] *Assignment and use of identities in the maritime mobile service*. Accessed 07/11/2022. URL: https://www.itu.int/rec/R-REC-M.585-9-202205-I/en.

[20] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. 3rd. Wiley Publishing, 2013. ISBN: 1118530802.

[21] Alejandro Vaisman and Esteban Zimányi. "Spatial and Mobility Data Warehouses". In: *Data Warehouse Systems: Design and Implementation*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2022, pp. 437–486. ISBN: 978-3-662-65167-4. URL: https://doi.org/10.1007/978-3-662-65167-4_12.

[22] Deepti Mishra, Ali Yazici, and Beril Pinar Basaran. "A casestudy of data models in data warehousing". In: *2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*. 2008, pp. 314–319. DOI: 10.1109/ICADIWT.2008.4664365.

[23] *Raster Data in ArcSDE 8.3 An ESRI ® White Paper*. Accessed on 07/06/2023. 2003. URL: http://downloads.esri.com/support/whitepapers/sde_/arcsde83_raster.pdf.

[24] Hanan Samet. "An overview of quadtrees, octrees, and related hierarchical data structures". In: *Theoretical Foundations of Computer Graphics and CAD* (1988), pp. 51–68.

[25] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. URL: https://doi.org/10.1145/361002.361007.

[26] Ruiyuan Li et al. "Distributed Spatio-Temporal k Nearest Neighbors Join". In: *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*. 2021, pp. 435–445.

[27] Sekander Hayat Khan M. "Standard Deviation". In: *International Encyclopedia of Statistical Science*. Ed. by Miodrag Lovric. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 1378–1379. ISBN: 978-3-642-04898-2. URL: https://doi.org/10.1007/978-3-642-04898-2_535.

[28] Hervé Abdi. "Coefficient of variation". In: *Encyclopedia of research design* 1.5 (2010).

[29] Martin Fleischmann. *Line simplification algorithms*. Accessed on 16/12/2022. Apr. 2020. URL: https://martinfleischmann.net/line-simplification-algorithms/.

[30] David H. Douglas and Thomas K. Peucker. "ALGORITHMS FOR THE REDUCTION OF THE NUMBER OF POINTS REQUIRED TO REPRESENT A DIGITIZED LINE OR ITS CARICATURE". In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10 (1973), pp. 112–122. DOI: 10.3138/FM57-6770-U75U-7727.

[31] M. Potamias, K. Patroumpas, and T. Sellis. "Sampling Trajectory Streams with Spatiotemporal Criteria". In: *18th International Conference on Scientific and Statistical Database Management (SSDBM'06)*. 2006, pp. 275–284. DOI: 10.1109/SSDBM.2006.45.

[32] Walid G. Aref and Ihab F. Ilyas. "SP-GiST: An Extensible Database Index for Supporting Space Partitioning Trees". In: *Journal of Intelligent Information Systems* 17.2 (Dec. 2001), pp. 215–240. ISSN: 1573-7675. URL: https://doi.org/10.1023/A:1012809914301.

[33] Jeffrey F Naughton and Avi Pfeffer. "Generalized search trees for database systems". In: *Readings in database systems* (2005), p. 467.

[34] The Kubernetes Authors. Accessed 02/12/2022. 2022. URL: https://kubernetes.io/docs/home/.

[35] Anirban DasGupta. "The Trimmed Mean". In: *Asymptotic Theory of Statistics and Probability*. New York, NY: Springer New York, 2008, pp. 271–278. ISBN: 978-0-387-75971-5. URL: https://doi.org/10.1007/978-0-387-75971-5_18.

[36] Sheng Wang et al. "A Survey on Trajectory Data Management, Analytics, and Learning". In: *ACM Comput. Surv.* 54.2 (Mar. 2021). ISSN: 0360-0300. URL: https://doi.org/10.1145/3440207.

[37] Hai Lan et al. "VRE: A Versatile, Robust, and Economical Trajectory Data System". In: *Proc. VLDB Endow.* 15.12 (Sept. 2022), pp. 3398–3410. ISSN: 2150-8097. URL: https://doi.org/10.14778/3554821.3554831.

[38] Tariq Alsahfi, Mousa Almotairi, and Ramez Elmasri. "A Survey on Trajectory Data Warehouse". In: *Spatial Information Research* 28.1 (Feb. 2020), pp. 53–66. ISSN: 2366-3294. URL: https://doi.org/10.1007/s41324-019-00269-x.

[39] Ruiyuan Li et al. "TrajMesa: A Distributed NoSQL Storage Engine for Big Trajectory Data". In: *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. 2020, pp. 2002–2005. DOI: 10.1109/ICDE48307.2020.00224.

[40] Ablimit Aji et al. "Hadoop gis: a high performance spatial data warehousing system over mapreduce". In: *Proceedings of the VLDB Endowment* 6.11 (2013), pp. 1009–1020.

[41] *AIS API Documentation*. Accessed on 17/05/2023. 2023. URL: https://servicedocs.marinetraffic.com/.

[42] *API for AIS Data*. Accessed on 17/05/2023. 2023. URL: https://api.vesselfinder.com/docs/.

[43] Zhicheng Liu, Biye Jiang, and Jeffrey Heer. “imMens : Real-time Visual Querying of Big Data”. In: *Computer graphics forum* 32.3pt4 (2013). ISSN: 0167-7055.

[44] Maxime Schoemans. personal communication. Developer of MobilityDB. June 12, 2023.

[45] *CLAAUDIA New Compute Cloud Strato*. Accessed 30/11/2022. 2022. URL: https : / / www . strato - docs . claaudia.aau.dk/%5C#what-is-claaudia-compute-cloud-strato.

## APPENDIX A
## DANISH GEODATA AGENCY

This appendix contains the summaries of meetings with representatives from the Danish Geodata Agency (DGA), whom acted as external domain experts during the development of DIPAAL.

### A. October 2022 Meeting

During a meeting in October 2022 with a representative from the DGA, the ideas behind DIPAAL were presented and feedback was provided. The feedback from the meeting was collected and documented into the following observations.

DGA introduces itself as a governmental organisation. Consequently, this means that they are currently motivated by green initiatives; therefore the design for DIPAAL should account for tracking the speed of ships to analyse CO2 emissions.

DGA states that their experience with REST API architecture has previously encountered issues and performance decrease when trying to send requests that have a long response time. During further discussion it was determined to be an issue with higher load of requests that could be solved by limiting access to DIPAAL.

DGA showed an interest in trying a new REST API solution, but note that as a governmental organisation, they have several restrictions on what network ports they are allowed to use, which the API for DIPAAL should account for.

DGA uses QGIS for analysis, an open-source geographical information system. DGA showed interest in DIPAAL being able to integrate with it.

DGA's current solutions all use the Lambert conformal conic (LCC) projection EPSG:3034[21] and would like DIPAAL to use the same. This projection encompasses all the waters surrounding Denmark, which is the area DGA is responsible for.

DGA is experienced with PostgreSQL databases and is using it for their back-end for at least one solution. DGA have no experience with the PostgreSQL extension Citus, but showed interest in seeing how it will affect performance for DIPAAL.

DGA directly stated that DIPAAL must support the creation of heatmaps. DGA uses heatmaps for many of its tasks, especially when considering where to perform detailed depth measurements, as only a fraction of the map can be updated by their organisation each year. Using heatmaps depicting the density of ship traffic was given as an example for this context.

DGA was presented with the idea of joining trajectories onto a grid to increase the performance of specific queries, such as the creation of rasters used in heatmaps projections. This idea was approved by DGA, who specified that the grid resolution must be at least 50m, as that is the smallest size allowed for draught sea maps, one of their main products. DGA showed scepticism for larger aggregation of grids and stated that aggregations of aggregations should be done with care and for a clear purpose.

DGA are currently creating most of their heatmaps using a system that supports point clouds, which are currently aggregated into 5m grids. DGA are actively working on reducing the grid size to 1m grids. Therefore, a smaller grid size for DIPAAL is desirable.

DGA stated that automatic identification system (AIS) data could be assumed not to receive updates nor changes once published, allowing DIPAAL to be a read-only with the exception of a daily extract, transform, load (ETL) run.

### B. December 2022 Meeting

During a meeting in December 2022 with two representatives from the DGA, the progress made with DIPAAL was presented and feedback was provided. The feedback from the meeting was collected and documented into the following observations.

DGA would like performance benchmarks to conclude whether any bottlenecks are CPU-bound or I/O bound.

DGA showed little interest in row counts for each relation, as they are more interested in the actual data size, with statistics over the data sizes loaded during the ETL process. DGA elaborated that they would like to see statistics for data size of the input files with the time it took to complete for days and months of data.

DGA want recommendations on if and when Citus' columnar compression is useful.

DGA wants to see the benchmarks be performed on deterministic hardware, as the lack of performance guarantees from the service Claaudia [45], produced some doubt about the presented benchmark results.

DGA stated that the frequent updates to Citus could be a concern for them, as they prefer to be dependent on few and stable releases.

DGA expressed concerns with the PostgreSQL extension MobilityDB, as it appears to be more of an academic project than an actual product. These concerns seemed to be alleviated as they were assured the developers behind MobilityDB are working hard towards a more finished product.

DGA would like benchmarks that evaluate how the performance of DIPAAL scales with the increase of nodes in the cluster, starting from a single node.

DGA approved DIPAAL using line simplification, as described in Section VIII-A, with an error bound of 10m to reduce data size of trajectories, showing no concern for the minor loss in precision given the gain in storage space and performance.

DGA would value that DIPAAL is able to query on several years of AIS data, enabling analysis on how ship routes changes over seasons, as part of future analytical efforts.

DGA was impressed with the execution time of the presented queries during the live demonstration. The DGA noted how this proved that the backend is well-built and runs effectively.

[21] https://epsg.io/3034

### C. February 2023 Meeting

During a meeting in February 2023 with two representatives from the DGA, the first version of the paper was sent in advance of this meeting and a presentation of DIPAAL, and potential future work was given. The two representatives asked questions during the presentation and provided feedback. The feedback from the meeting was collected and documented into the following observations.

DGA was presented with examples of Citus columnar ability to compress data in DIPAAL with observations on how the compression is handled. DGA then stated they would like to see a comparison on the query performance of DIPAAL using Citus columnar without spatial indices and row-major with spatial indices.

DGA was concerned when presented with an issue found in MobilityDB and asked whether any or all extensions used in PostgreSQL conflicted with each other. DGA was informed that the issue presented, which involved the loss of SRID during serialisation after shard rebalancing, was an issue within MobilityDB.

DGA want MobilityDB to be evaluated for whether it is ready to be used in production.

DGA questioned DIPAAL's availability if made available to them or others, especially during load from the ETL process. DGA were told this was outside the scope of the project and that no such testing had been performed. DGA was also informed that Citus refers to regular PostgreSQL replication for this purpose.

DGA was asked the temporal extend they expected to analyse using DIPAAL, hence how much data should be loaded in as a minimum. DGA where uncertain but suggested at least a couple of years.

DGA stated they currently use 50 meter cells to create depth charts but are currently working towards reducing that to 1 meter cells.

DGA was asked what the largest area a cell in the cell representation could be while still being useful for analytical purposes. DGA stated each region of the world would benefit from different sizes of cell representation, with some regions benefiting from being very coarse, with 10 kilometres as an example.

DGA stated that observing the Electronic Navigational Chart (ENC) cells, which is provided and updated by the International Hydrographic Organisation (IHO), would provide insight into what regions of the world would benefit from a finer or coarser cell representation.

DGA stated that sailing routes changes quite a bit between seasons, hence they are very interested in analysing these differences. DGA further stated that a video showcasing these differences with several heatmaps is very desirable.

DGA stated they are more interested in seeing the results from analysing years' worth of data rather than the implementation of more features for DIPAAL.

### D. March 2023 meeting

The meeting in March was a status meeting, where the current state of DIPAAL was demonstrated, inviting discussion of design changes.

DGA was inquired how they would like to measure the density of ships, i.e. different types of heatmaps. DGA stated they would like to have moving and non-moving heatmaps separated. DGA also brought up the possibility of "reverse heatmaps", i.e. a heatmap indicating where ships are sailing abnormally.

DGA was asked what the minimum temporal resolution of heatmaps is, while still being useful for their analytics needs. DGA could not imagine any useful analysis from having a temporal resolution lower than one day. Hourly heatmaps were discussed, but DGA concluded that it is not interesting, as the data foundation is too sparse for this granularity.

The static spatial distribution scheme was presented to DGA. DGA was asked if they could see flaws in the scheme's assumptions. DGA stated that it was a fair assumption that ship routes change seldomly. The seldom changes could, however, be new harbours and industries opening or closing or just changes to the mandated ship routes. The concern of low-traffic areas being susceptible to large relative changes was brought up, but as low-traffic areas should not be the computationally heavy areas anyway, this concern was discarded.

DGA was presented that one year of loaded data occupied approximately 1.3 TiB of storage. Since the new cluster has limited storage, it was discussed if reducing the largest relation would be beneficial in order to process and store more years' worth of data. DGA expressed that fast 50m responses are not needed everywhere and only in specific areas of interest, such as harbours or problem areas. It was suggested to use Electronic Nautical Charts[22] as a guideline to what areas of interest should be kept.

## APPENDIX B
## SAMPLE API QUERIES

This appendix covers some examples of queries the DIPAAL API supports, including their meaning and result.

### A. Spatio-temporal Range Queries

To ask the DIPAAL API which ships were near one of the Nord Stream sabotages in 2022 in the bounding rectangle represented by the EPSG:3034 coordinates pairs *(4353000 3196100), (4354100 3197200)*[23] during the timespan from a week up until the sabotage, the following query is made: https://dipaal.dk/api/v1/ships/?search_method=cell_200m&start_timestamp=2022-09-19T17%3A03%3A00Z&end_timestamp=2022-09-26T17%3A03%3A00Z&x_min=4353000&y_min=3196100&x_max=4354100&y_max=3197200

[22]https://eng.gst.dk/danish-hydrographic-office/nautical-charts/electronic-charts

[23]https://nautiskinformation.soefartsstyrelsen.dk/details.pdf?messageId=8166360e-4f40-4b43-b2a1-4aa81f9f1552&language=en, https://nautiskinformation.soefartsstyrelsen.dk/details.pdf?messageId=72b2c3b1-0d0d-4a14-9ee0-be96c0a699fb&language=en

```
[
  {
    "ship_id": 13294,
    "name": "NIKE",
    "callsign": "9HA2013",
    "mmsi": 249803000,
    "imo": 9431032,
    "mid": 249,
    "flag_region": "europe",
    "flag_state": "Malta",
    "mobile_type": "Class A",
    "ship_type": "Tanker",
    "location_system_type": "AIS",
    "a": 98.0,
    "b": 24.0,
    "c": 10.0,
    "d": 7.0,
    "length": 122.0,
    "width": 17.0
  }
]
```


Listing 1: JSON output of querying distinct ships spatio-temporally near one the Nord Stream sabotages.

Please note the *search_method* query parameter, which determines what representation and granularity is used to fetch the result. The supported methods are the four cell granularities and a trajectory based search method. Cell 200m is chosen as some spatial buffer is allowed, and 50m cells are not stored for this area.

The output of the above query is a JSON array of ships that were within the spatio-temporal range. A small sample of the output is seen in Listing 1

Cell and trajectory resources can also be requested within a spatio-temporal range. For example to get stopped trajectories intersecting the spatio-temporal range, the following query is made: https://dipaal.dk/api/v1/trajectory/trajectories/?offset=0&limit=10&x_min=4353000&y_min=3196100&x_max=4354100&y_max=3197200&srid=3034&start_timestamp=2022-09-19T17%3A03%3A00Z&end_timestamp=2022-09-26T17%3A03%3A00Z&stopped=true&time_series_representation_type=GeoJSON. The query parameter *time_series_representation* determines how the trajectory is encoded into JSON. Currently, GEOJSON and MF-JSON are supported. The output of this trajectory query is an empty array, as no stopped trajectories have been observed around this location. If instead queried with *stopped=false*, the result is seen in Listing 2

Likewise, to get the cell fact that spatio-temporally intersect, the following query is made: https://dipaal.dk/api/v1/cells/200m?x_min=4353000&y_min=3196100&x_max=4354100&y_max=3197200&cell_size=200m&srid=3034&start_timestamp=2022-09-19T17%3A03%3A00Z&end_timestamp=2022-09-26T17%3A03%3A00Z
The output of this query is a JSON-array of seven cell facts, all for the ship seen in Listing 1, and a small sample of the output is seen in Listing 3

### B. Other Filters

Besides spatio-temporal range queries, cell facts, trajectories, and the ship resources can all be queried by a range of filters.

For example, to find all ships registered in Africa, but not in Liberia, with a length over 100m, the following query is used: https://dipaal.dk/api/v1/ships/?flag_region_in=africa&flag_region_nin=Liberia%20%28Republic%20of%29&length_gt=100. Like before, the output of the ship endpoint is JSON, and is seen in Listing 4

### C. Heatmap Queries

In addition to the example covered in Section VI, a differential heatmap example is shown.

If the user want the difference of time spent per pixel, in the Great Belt between July and January 2021 for the ship type "Pleasure" in 50m resolution, the query is https://dipaal.dk/api/v1/heatmap/mapalgebra/time/50m?output_format=png&map_algebra_expr=%5Brast1.val%5D-%5Brast2.val%5D&map_algebra_no_data_1_expr=%5Brast2.val%5D&map_algebra_no_data_2_expr=-%5Brast1.val%5D&enc_cell=Storeb%C3%A6lt%2C%20N-lige%20del&first_mobile_types=Class%20B&first_ship_types=Pleasure&first_start_timestamp=2021-01-01T00%3A00%3A00Z&first_end_timestamp=2021-02-01T00%3A00%3A00Z&second_mobile_types=Class%20B&second_ship_types=Pleasure&second_start_timestamp=2021-07-01T00%3A00%3A00Z&second_end_timestamp=2021-08-01T00%3A00%3A00Z

To break down the above query, first it asks for a time heatmap of 50m as part of the URL path. Then it asks for an output format of PNG. Then it supplies the needed map algebra expressions. The map algebra expression defines how to calculate a pixel value based on the two heatmaps. In this example, the output heatmap is the difference, i.e. *rast2.val-rast1.val*. The MapAlgebra NoData expressions defines what to do, when either raster value is missing. In this case, just keep the other's raster's value. Then the spatial bound of the query is specified, to be equal to the spatial definition of the ENC-cell "Storebælt N-lige del". Lastly, the temporal spans of the two rasters are defined, which in this case are August for the first, and July for the second. The ship type is also limited to "Pleasure" ships. This results in an output where negative values indicate more traffic in August, and positive indicate more traffic in July.

The output of this query is a GeoTiFF, which is rendered in QGIS and is seen in Figure 16

## Appendix C
## Benchmark Plans

This appendix covers the detailed plans for execution of the performed evaluations in Section IX.

```
[
  {
    "trajectory_sub_id": 1380374154,
    "start_timestamp": "2022-09-24T16:10:44",
    "end_timestamp": "2022-09-24T23:59:56",
    "eta_timestamp": "2022-09-25T15:30:00",
    "trajectory": {
      "trajectory": {
        "crs": { "type": "name", "properties": { "name": "EPSG:4326" } },
        "type": "LineString",
        "datetimes": [
          "2022-09-24T17:16:25.120919+00:00",
          ...,
          "2022-09-24T17:17:42.713692+00:00"
        ],
        "coordinates": [
          [ 15.785330243, 55.550880443 ], ..., [ 15.777899126, 55.55150635 ]
        ]
      }
    },
    "rot": {
      "type": "MovingFloat",
      "values": [ 0, 0 ],
      "datetimes": [ "2022-09-24T16:10:44+00", "2022-09-24T23:59:56+00" ],
      "lower_inc": true, "upper_inc": true, "interpolation": "Step"
    },
    "heading": {
      "type": "MovingFloat",
      "values": [277, ..., 269],
      "datetimes": [ "2022-09-24T16:10:44+00", ..., "2022-09-24T23:59:56+00" ],
      "lower_inc": true, "upper_inc": true, "interpolation": "Step"
    },
    "draught": {
      "type": "MovingFloat",
      "values": [ 5.4, 5.4 ],
      "datetimes": [ "2022-09-24T16:10:44+00", "2022-09-24T23:59:56+00"],
      "lower_inc": true, "upper_inc": true, "interpolation": "Step"
    },
    "destination": "SEGOT", "duration": 28152.0, "length": 164207,
    "stopped": false, "navigational_status": "Under way using engine",
    "ship": {
      "ship_id": 13294, "name": "NIKE",
      "callsign": "9HA2013", "mmsi": 249803000,
      "imo": 9431032, "flag_region": "europe", "flag_state": "Malta",
      "mobile_type": "Class A", "ship_type": "Tanker",
      "location_system_type": "AIS",
      "a": 98.0, "b": 24.0, "c": 10.0, "d": 7.0,
      "length": 122.0, "width": 17.0
    }
  }
]
```


Listing 2: Truncated JSON output of the spatio-temporal query on trajectory facts.

```
[
  {
    "x": 21765,
    "y": 15980,
    "trajectory_sub_id": 1380374154,
    "entry_timestamp":
        "2022-09-24T17:17:09+00:00",
    "exit_timestamp":
        "2022-09-24T17:17:42+00:00",
    "navigational_status":
       "Under way using engine",
    "direction": {
      "begin": "Unknown",
      "end": "Unknown"
    },
    "sog": 11.784928920198922,
    "delta_cog": 0.0,
    "delta_heading": 1.0,
    "draught": 5.4,
    "stopped": false,
    "ship": {
      "ship_id": 13294,
      "name": "NIKE",
      "callsign": "9HA2013",
      "mmsi": 249803000,
      "imo": 9431032,
      "flag_region": "europe",
      "flag_state": "Malta",
      "mobile_type": "Class A",
      "ship_type": "Tanker",
      "location_system_type": "AIS",
      "a": 98.0,
      "b": 24.0,
      "c": 10.0,
      "d": 7.0,
      "length": 122.0,
      "width": 17.0
    }
  },
```


Listing 3: Truncated JSON output of the spatio temporal query on 200m cell facts.


```
[
  {
    "ship_id": 12037,
    "name": "TIN ZIREN",
    "callsign": "7THG",
    "mmsi": 605086070,
    "imo": 9697325,
    "mid": 605,
    "flag_region": "africa",
    "flag_state": "Algeria",
    "mobile_type": "Class A",
    "ship_type": "Cargo",
    "location_system_type": "AIS",
    "a": 130.0,
    "b": 17.0,
    "c": 17.0,
    "d": 4.0,
    "length": 147.0,
    "width": 21.0
  },
  ...
]
```


Listing 4: Truncated JSON output of querying African ships, that are not from Liberia, with a length greater than 100m.

### A. Runtime Benchmarks

In order to get repeatable measurements of query runtime performance, a procedure to reduce the impact of the cache has been implemented.

Before every iteration of a query runtime benchmark, the cache is cleared and then pre-warmed. The cache is cleared by clearing the operating system file cache and restarting all PostgreSQL instances. A range of random queries are run to avoid testing on a cold cache. The random queries hit every relation in the data warehouse and uses random query parameters. The purpose of the random query execution is to produce a system cache that is realistic in a real use scenario, without being influenced by the result of previous query iterations.

All query runtime benchmarks are run with the statement **EXPLAIN (ANALYZE, VERBOSE, TIMINGS, BUFFERS, FORMAT JSON)**[24] prefixed. Running with explain analyse does not only eliminate client-server transmission of the result, but the result of the explain analyse is saved and facilitates later analytics of the query performance, such as the Worker Idle Fraction (WIF), described in Section IX. Besides the WIF, all query runtime benchmarks produce a runtime metric in seconds, which does not include server-to-client latency and communication.

Each query runtime benchmark is evaluated ten times. To avoid the impact of outliers, a trimmed mean is used, removing the two fastest and two slowest results.

### B. Heatmap Runtime Benchmark

The heatmap runtime benchmark serves three purposes.

1) Evaluate the spatial distribution regarding scale up from one to five workers.
2) Evaluate the pre-aggregated heatmap performance.
3) Compare the columnar and row-based physical access method.

[24]https://www.postgresql.org/docs/current/sql-explain.html

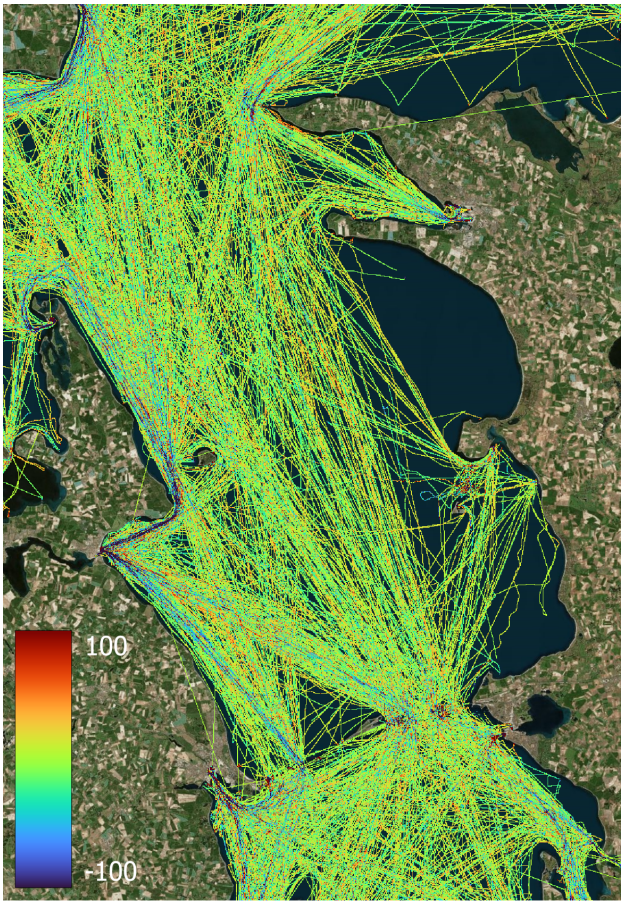



Fig. 16: Differential time heatmap between “Pleasure” ships in July and August.

The heatmap runtime benchmark evaluates the query runtime across a range of heatmap queries. This range of queries is then evaluated across different worker setups and DW configurations. Two queries are evaluated with different parameters. The first query, single heatmap, is to construct a heatmap within a spatio-temporal range, and the second query, mapalgebra heatmap, is to construct a heatmap as a combination of two heatmaps using the mapalgebra operation.

Both queries have a range of parameters. During the evaluation, some of the parameters are constant while others are variable. The variable parameters test different scenarios and data intensities.

Ship type is a constant parameter. The single heatmap is always calculated on Cargo ships only. The mapalgebra heatmap query is the difference between Pleasure and Cargo ships. Both queries always calculate the heatmap type of “count”.

The variable parameters are the spatio-temporal range and resolution of the heatmap. The spatio-temporal range is a cross product of three spatial ranges; Aarhus Harbour ($46.7km^2$), The Great Belt ($3\ 071km^2$), and spatial domain of DIPAAL ($725\ 725km^2$), as well as three temporal spans; 1 day (February 28th, 2021), 30 days (January 26th through February 24th, 2021), and 1 year (2021). This results in nine queries. Each query is then evaluated across each of the four granularities in DIPAAL, resulting in 36 queries. Each query is evaluated on the single heatmap query and the mapalgebra heatmap query, resulting in 72 queries.

These 72 queries are initially run against a cluster with five workers. The 72 queries are tested for three different relation access methods; row-based (regular PostgreSQL relation), columnar-based (Citus Columnar), and columnar-based but with partitions for each type and granularity. This is evaluated to find the most optimal access method.

The best-performing access method is then evaluated on a one-worker setup to evaluate the scale up factor of the heatmap queries. In total, the 72 queries are evaluated four times, resulting in 288 results.

The scale up is expected to be close to linear, as the spatial distribution approach distributes the workload evenly. Larger spatio-temporal ranges are expected to run slower than smaller spatio-temporal ranges. Lastly, the row-based access method is expected to be faster than Citus columnar, as the columnar-approach cannot use index scans. It is expected that Columnar performs better if partitioned by heatmap type, granularity, and month.

### C. Cell Runtime Benchmark

The cell runtime benchmark serves two purposes.

1) Evaluate the spatial distribution regarding scale up from one worker to five workers.
2) Evaluate the cell representation and cell hierarchy versus the trajectory representation, as seen in Figure 1.

All queries are run both on a one-worker and five-worker setup to evaluate the spatial distribution regarding scale up. If the scale up is close to linear, the spatial distribution approach evenly distributes the work across all workers.

A range of query parameters are used to evaluate the cell representation and cell hierarchy versus the trajectory representation. All queries in the cell runtime benchmark is answering the question “What ships were in this spatio-temporal range?”, with the spatio-temporal range being variable.

The spatio-temporal ranges evaluated are the cross product of four areas and three temporal spans. The four areas are chosen strategically. First, a really small area is chosen, as it is contained within a single shard, which is an interesting edge-case, as Citus cannot utilise more than one worker, and thus this query should not be affected by the number of workers in the cluster. The area chosen is the Aabenraa Harbour, with an area of $6.6km^2$. The rest of the areas chosen are Aarhus Harbour, with an area of $46.7km^2$, Skagerrak, with an area of $5\ 528.8km^2$, and the danish seaborders, with an area of $535\ 556km^2$. These areas are chosen to evaluate how the performance differs in different sizes.

To evaluate a range of temporal span sizes, a total of three temporal spans are chosen; one day (January 10th, 2021), 30 days (January 26th through February 24th, 2021), and 90 days (January 1st through April 30th, 2021).

Earlier experimentation has shown that querying a large area in a large temporal span of 50m cell size resulted in out-of-memory conditions. To combat this, the cell hierarchy is utilised. For this reason, not all cell granularities are evaluated for all spatial areas. Likewise, the coarsest granularities are not evaluated for the smallest areas, as the spatial error is too high. The configurations evaluated are seen in Table VIII.

| | 50m | 200m | 1000m | 5000m |
|---|---|---|---|---|
| Aabenraa Habour ($6.6km^2$) | ✓ | ✓ | | |
| Aarhus Harbour ($46.7km^2$) | ✓ | | | |
| Skagerrak ($5528.8km^2$) | | ✓ | ✓ | ✓ |
| Danish Seaborders ($535\ 556.5km^2$) | | | ✓ | ✓ |

TABLE VIII: The different configurations chosen of the four areas and four cell granularities.

The overall ruleset for constructing the matrix seen in Table VIII is to ensure at least two areas per granularity and at least two granularities per area. This is to compare how the query runtime and/or WIF changes dependent on the area and/or granularity.

For all configurations in the matrix, all three temporal spans are evaluated. This results in a total of 30 queries, which are evaluated against both a one- and five-worker setup.

Small areas, especially Aabenraa Harbour, is expected to result in a high WIF, as it fails to engage many shards. The large areas should have a near zero WIF and close to linear scale up, if the spatial distribution approach works as intended. A coarser granularity is expected to always yield a faster query runtime result, and a larger area or temporal span slow the query down.

### *D. Lazy Cell Calculation Runtime Benchmark*

To evaluate whether it is feasible to delete infrequently queried 50m cell facts to save disk space, a lazy-load query is evaluated against the eagerly loaded fact cells. This comparison gives an idea of how much slower it is to query on lazy-loaded areas compared to eager-loaded, and thus gives an understanding of the trade-of of disk space versus query runtime. This understanding helps decide what areas to delete 50m cell facts from, if any.

A total of six spatio-temporal ranges are compared. These ranges are the cross-product of two spatial areas and three temporal spans. The two areas are South West Laesoe, with an area of $164km^2$ and Near Heligoland, with an area of $7,189km^7$. These two areas are chosen as one small and one medium-sized area, with low traffic.

The expected result is that the eagerly loaded cell query runtime is much faster than the lazy loaded cell query runtime, as no calculation is needed. It is also expected that, due to the small areas, that the WIF is high on both the eager and lazy loaded queries.

### *E. Spatial Distribution Skewness Evaluation*

To evaluate the consequences of a static spatial distribution approach, the shards built on the year 2021 are compared to older and newer data. The years 2011 and 2022 are chosen as comparisons.

2011 is chosen as a worst case of the skewness. There is a 10-year difference between the year it was built on, and 2011. Even though DMA have published AIS data since 2007, with 2008 being the first complete year, 2011 is chosen. 2011 is chosen over the previous years, as it was uncovered, that the first couple of years of AIS data are inconsistent, with complete days missing, and intermittently including international data.

To compare the skewness across the years, the metrics Standard Deviation (SD) [27] and Coefficient of Variance (CV) [28] is used. For both metrics, a lower value indicates better-balanced shards.

The SD and CV of 2022 is expected to be closer to the SD and CV of 2021 measured in Section V than 2011, as 2022 data are temporally nearer 2021. However, as DGA explained in Appendix A, major ship traffic pattern changes rarely happens, which is why it is expected to be close to the SD of 2021, but not as close as 2022.

| Relation name | Relation Size | Total Relation Size | Row Count (thousand rows) |
|---|---|---|---|
| fact_cell_50m | 1799 GB | 4153 GB | 10 589 794 |
| fact_cell_200m | 455 GB | 1085 GB | 2 676 381 |
| fact_cell_heatmap | 752 GB | 935 GB | 2 282 717 |
| fact_cell_1000m | 96 GB | 243 GB | 567 293 |
| dim_trajectory | 37 GB | 165 GB | 40 254 |
| fact_cell_5000m | 25 GB | 65 GB | 145 524 |
| dim_cell_50m | 18 GB | 29 GB | 116 092 |
| fact_trajectory | 3896 MB | 9703 MB | 40 254 |
| dim_cell_200m | 1399 MB | 2274 MB | 8 767 |
| dim_ship | 62 MB | 146 MB | 88 |
| dim_cell_1000m | 68 MB | 140 MB | 413 |
| dim_cell_5000m | 5 MB | 37 MB | 18 |
| Total | 3187 GB | 6687 GB | 16 467 595 |

TABLE IX: The size of the relations in DIPAAL and the row count.

## APPENDIX D
## RELATION STATISTICS

Table IX shows significant relations' size, total size, and row count. It is measured using the Citus procedures *citus_relation_size* and *citus_total_relation_size*. Relation size excludes indices, free-space map, and visilibity map. The measurements are taken with 1379 days of AIS data loaded (2022, 2021, 2011 and most of 2020)

The numbers for `fact_cell_50m` in Table IX does not reflect the count after deletion of cell facts outside of areas of interest. After deletion of 50m cell facts outside Harbor ENCs, as described in Section VIII-F, only 338 million cell facts are kept. By assuming the space occupied by the relation is linear to the count of rows, it is estimated the deletion reduces the total relation size of the `fact_cell_50m` relation to approximately 132GB, thus reducing the total footprint of the five years from 6.7TB to 2.7TB.