A Scalable Model for Vessel-Generated Underwater Noise: Enhancing Efficiency through Parallelisation

Giulia Rovinelli[1,*], Esteban Zimányi[2], Marta Simeoni[1,3], Davide Rocchesso[4] and Alessandra Raffaetà[1]

[1]Ca' Foscari University of Venice, Venice, Italy
[2]Université Libre de Bruxelles, Bruxelles, Belgium
[3]European Centre for Living Technology (ECLT), Venice, Italy
[4]Università degli Studi di Milano Statale, Milano, Italy

Abstract

Underwater noise pollution by shipping activities is widely recognised as a significant threat to marine life. The noise emitted by vessels can have various detrimental effects on fish and marine ecosystems. Therefore, accurately estimating and analysing vessel-generated underwater noise is a critical challenge for the protection and conservation of marine environments. For this reason, we have built a model for the spatio-temporal characterisation of underwater noise generated by vessels. This paper builds on this model by optimising the code pipeline, implementing table partitioning and leveraging parallelisation techniques. These enhancements allow us to explore various partitioning methods while significantly improving the computational performance and enabling more efficient analysis of underwater noise. Our approach not only improves the computational efficiency but also preserves the accuracy of the noise calculations, offering a more scalable solution for large datasets.

Keywords

Spatio-temporal databases, underwater noise, parallelisation techniques

1. Introduction

Underwater noise generated by human activities, especially from shipping, is known to produce short and long term effects on marine animal species. This noise pollution can disrupt the natural acoustic environment, leading to several adverse consequences. Some of the negative impacts include interference with communication, changes in behaviour, stranding, and increased mortality rates [1, 2]. Therefore, characterising underwater noise is crucial for monitoring the health of aquatic life, assessing potential risks, and providing valuable information to ecologists and policy makers. This enables the development of effective strategies to maintain a productive and healthy ecosystem. However, measuring underwater noise is a complex and computationally demanding task. In addition to the installation of hydrophones, which requires specific resources and expertise for proper deployment and calibration, the analysis of the collected data is equally challenging. Once the data is acquired, it must be processed to extract meaningful insights, a process that can require substantial computational power, especially when monitoring large areas or extended time periods. Moreover, direct measurement of underwater noise is not always feasible, particularly in remote regions or deep waters. In these cases, acoustic models are employed to simulate sound propagation. However, these models also require a wide range of input data, including detailed environmental parameters and vessel-specific characteristics, in addition to huge computational effort to handle the complex calculations involved. For this reason, the development of sound propagation models that balance accuracy with computational efficiency is essential. Such models must be capable of providing reliable predictions while minimising resource consumption, enabling their application on larger scales or in data-intensive scenarios.

In this work, building upon the model developed in [3] and refined in [4], we introduce several enhancements aimed at improving the efficiency of its implementation. Specifically, we optimise the computational pipeline to handle large-scale spatio-temporal datasets more effectively while preserving the results of the previous model. The optimisations include the restructuring of the code to cope with time consuming operations and the implementation of table partitioning using PostgreSQL [5] and Citus [6], as well as leveraging parallelisation techniques to improve processing speed and scalability. The framework has been implemented in MobilityDB [7], an open-source platform for managing and analysing geospatial trajectory data. Our framework enables various analyses to estimate the impact of fishing activities on underwater noise pollution.

To demonstrate the potential of the developed system,

Published in the Proceedings of the Workshops of the EDBT/ICDT 2025 Joint Conference (March 25-28, 2025), Barcelona, Spain
*Corresponding author.
giulia.rovinelli@unive.it (G. Rovinelli); esteban.zimanyi@ulb.be (E. Zimányi); simeoni@unive.it (M. Simeoni); davide.rocchesso@unimi.it (D. Rocchesso); raffaeta@unive.it (A. Raffaetà)

we focus on the fishing activities in the Northern Adriatic Sea, one of the most heavily exploited areas of the Mediterranean Sea, where underwater noise pollution is a recognised consequence of intensive fishing activity. The dataset used in this study includes AIS data from Italian and Croatian fishing vessels for June 2020. Moreover, to determine the acoustic features of the vessel engines and refine the propagation model, we use direct acoustic measurements from the Interreg project SOUNDSCAPE[1], which conducted acoustic monitoring in the Northern Adriatic Sea from March 2020 to June 2021.

The paper is organised as follows. Section 2 overviews the sound propagation model introduced in [3] and refined in [4]. Section 3 focuses on the optimisation of the computational pipeline to enhance the model performance. Section 4 discusses the implementation of data partitioning techniques with PostgreSQL and explores the integration of the Citus extension to enable distributed processing. Finally, Section 5 presents some concluding remarks.

2. Underwater Noise Model

In this section, we briefly describe the model for underwater sound propagation based on our previous work [3] and significantly refined in [4] w.r.t. several aspects.

The basic objective of noise modelling is to assess how much noise a particular activity will generate in the surrounding area. Specifically, the aim is to model the received noise level (RL) at a given point (or points), based on the sound source level (SL) of the noise source, and the amount of sound energy which is lost as the sound wave propagates from the source to the receiver (transmission loss or propagation loss, TL). The principal sources of underwater noise are machinery, propellers, and cavitation. Our AIS dataset includes some data of the fishing boats, such as the length overall (LOA) of the boat, the horsepower of the engine and also the fishing gear used. However, the dataset does not include direct measurements of the sound pressure levels of the fishing vessels. So, we infer such values considering the general literature about underwater noise and the measurements provided by the SOUNDSCAPE project [8], which conducted acoustic monitoring in the Northern Adriatic Sea from March 2020 to June 2021. In particular, we use the measurements of a hydrophone located in the middle of the Adriatic Sea, taken on March 31, 2021 between 5:40 pm and 5:55 pm. Here, there is a unique fishing vessel crossing nearby the hydrophone and taken as the reference boat. This allows us, by linear regression on sound pressure level measurements, to assign a vessel with an 835 Hp engine, when not trawling, an estimated source level of 136 dB at 63 Hz. In order to associate the source levels to all the other vessels, we need to relate the sound pressure level to the engine horsepower, the latter being available in our dataset. If we assume that a constant fraction of engine power gets converted into acoustic power (i.e. acoustic power scales linearly with horsepower), then 3 dB are added per doubling in engine power. We adopt such a linear progression on logarithmic scale of engine power and the resulting value is denoted with SL_0. For example, for engines between 100 Hp and 835 Hp, considering a frequency of 63 Hz, we obtain a range between 123 dB and 136 dB.

Differences in source level may result from variations in speed. Specifically, as noted in [9], the intrinsic factor of speed can influence the broadband source level of ships according to the following relation:

$$SL = \begin{cases} SL_0 & \text{if } v \leq v_0 \\ SL_0 + 15.39\ dB \times log_{10} \frac{v}{v_0} & \text{if } v > v_0 \end{cases} \quad (1)$$

where $v_0 = 3.9$ kn corresponds to the speed of the reference boat and v is the actual speed of the vessel.

Trawling vessels typically generate higher levels of radiated noise compared to free-running vessels operating under the same machinery settings. While published data on the radiated noise from operating trawling vessels are limited, some studies have reported increases in radiated noise ranging from 5 dB to 15 dB during trawling activities. Specifically, it is noted that the effect of trawling is minimal below 100 Hz and increases with frequency. Accordingly, we assign an increase of 5 dB at 63 Hz when the vessel is trawling.

To account for transmission loss, we adopt a combination of spherical propagation and mode stripping [10]. The resulting formula is:

$$TL = \begin{cases} 20\ log_{10}(r) & \text{if } r \leq r_{trans} \\ 15\ log_{10}(r) + 5\ log_{10}(r_{trans}) & \text{if } r > r_{trans} \end{cases} \quad (2)$$

The $15\ log_{10}(r)$ dependence on range is known as mode stripping because it results from the gradual erosion of steep ray paths (high-order modes) after multiple bottom reflections. To determine r_{trans}, we refer to the reference boat. At 63 Hz the transition is expected to occur at around 400 m, approximately 10 times the water depth.

Environmental absorption features may affect the transmission loss, especially for large distances and high frequencies. To take into account all the environmental aspects that influence the sound propagation underwater, we add a term proportional to distance from the source [11]:

$$TL_{tot} = TL + \alpha \times r \quad (3)$$

At frequency 63 Hz, α is on the order of 10^{-6} dB/m.

The classic sonar equation [12] provides an estimation of the received noise level (RL) by subtracting the trans-

[1] https://www.italy-croatia.eu/web/soundscape

mission loss (TL) from the sound source level (SL). However, it does not consider the ambient (or background) noise, which is present in the marine environment. The RL exceeding the ambient noise is the following:

$$RL = SL - TL_{tot} - AN \quad (4)$$

The SOUNDSCAPE measurements [13, 8] are also used to estimate the ambient noise. In particular, we employed the exceedance level L_{90}, which indicates the sound level that is exceeded 90% of the time. As mentioned in [13], L_{90} can be referred to as common natural acoustic conditions. To account for spatial and temporal variability, we partitioned the Northern Adriatic Sea into a 1 km $\times$ 1 km grid and assigned noise values based on L_{90} measurements at hydrophone stations. These values were interpolated using the Inverse Distance Weighting (IDW) in QGIS[2], producing maps that capture the heterogeneous underwater acoustic environment.

The implementation of the model to calculate the underwater noise generated by vessels is succinctly described below (for more details, see [4]). First, the Northern Adriatic Sea is partitioned into a regular grid composed of square spatial cells (1km$\times$1km). This grid, consisting of $43,508$ cells, is enriched with the ambient noise and some environmental features (such as the sea surface temperature or the salinity) which are essential for noise calculation. Then, starting from AIS data, we reconstruct the vessels trajectories and we deploy them in a spatio-temporal database [14]. These trajectories are equipped with semantic information, such as the acoustic characteristics of the vessel engines and the activities conducted along their paths, which are used to infer how the noise spreads in the area of interest. The entire trajectories reconstruction and their semantic enrichment leverage the temporal and spatio-temporal types of MobilityDB, as well as the functions provided by this spatio-temporal database. Subsequently, using the spatio-temporal functions of MobilityDB, we apply a sampling process on the vessel's trajectory at one-minute intervals to determine the boat's positions at specific temporal instants. For each position p, we estimate the decibels produced by the vessel, based on its activity and speed. Next, we calculate the propagation radius r, i.e. the distance at which the noise generated by the fishing vessel gets drowned into ambient noise, and we construct a buffer b with radius r around p. Then, we select all the grid cells whose centroids fall within b and compute the distance between the sampled point p and these centroids. This distance is used to determine the received noise in the selected cells. Finally, by grouping by cell id and time, we combine all the received sound levels to obtain the total noise level to be associated with the cell.

[2]https://qgis.org/en/site/

3. Noise Modelling Optimisation

In this section, we first describe the setting of our experiment concerning the implementation of the underwater noise model presented in [4]. Then, we propose some optimisations of the process, and discuss the benefits obtained in terms of time efficiency.

For our experiment we focus on June 2020, one of the months with the highest fishing activity in 2020. During this period, there are 642 fishing vessels, generating $9,841,079$ AIS data points and completing $7,462$ trips. Since the AIS data are limited to the Northern Adriatic Sea, we consider the projected coordinate system for Italy, specifically the spatial reference identifier (SRID) 6876. To process this data and build our model, we used a machine that features 32 Intel(R) Xeon(R) CPU E5-4610 v2 processors running at 2.30 GHz, offering multithread performance. It is equipped with 256 GB of DDR4 ECC RAM and it utilises a 500 GB RAID 5 storage configuration. On this machine we deployed PostgreSQL 16.6, PostGIS 3.5, and MobilityDB 1.3.

By using the approach from [4] recalled above, the reconstruction of the fishing vessels trajectories takes 46 minutes, while the pipeline to calculate the underwater noise propagation requires approximately 44 hours. The latter running time, referred as *Original Pipeline* in Figure 3, is the target of our optimisations.

We now outline the improvements to such a pipeline to enhance efficiency, support scalability, and reduce computational overhead. One of the most costly operations is the selection of the cells affected by the noise propagation. In fact, every 60 seconds we get all the fishing vessel positions, compute the noise generated by the vessels (SL) and then propagate it. To accomplish this task for each point we build a buffer using the propagation radius r. Then, we perform a `JOIN` operation with the table `Grid` storing the grid cells, followed by an `ST_Intersects` operation to determine the cells affected by the noise, i.e., those inside the buffer. Since the `ST_Intersects` operation involves the `geometry` type, it inherently requires computationally expensive spatial operations, which can significantly impact the model performance. To avoid this computational overhead, we make two significant changes: (i) restructure the table `Grid` and (ii) use a bounding box instead of a buffer in noise propagation. The aim is to find the cells involved in the noise propagation without using the expensive operation `ST_Intersects`.

Grid table restructuring. We add two new attributes to the cell of the grid: `grid_r` and `grid_c`, which indicate the row and column numbers within the grid. Hence, starting from the lower-left corner, the grid cells are numbered sequentially, so they are identified as $(1,1)$, $(1,2)$ and so on. This grid-based system allows for an efficient

identification of the cells within a bounding box, without the need for costly spatial operations. The table `Grid` includes also the x and y coordinates of the cell centroid, which will be used for calculating sound propagation. The structure of the table `Grid` is as follows.

```
CREATE TABLE Grid (
    grid_id integer PRIMARY KEY,
    grid_r integer NOT NULL,
    grid_c integer NOT NULL,
    centroid_x double precision,
    centroid_y double precision,
    elevation real,
    ambient_noise real,
    alpha tfloat );
CREATE INDEX idx_grid_r ON Grid (grid_r);
CREATE INDEX idx_grid_c ON Grid (grid_c);
```

Note that we also add two indexes to the table `Grid` on the columns `grid_r` and `grid_c`, to improve the efficiency of spatial query operations.

Bounding box for Noise Propagation. To compute the total received noise level for each cell of our grid, we proceed as illustrated in Figure 1. After reconstructing the vessel trajectories from the AIS data, we get the positions of all the fishing vessels at the same time instants, i.e., every 60 seconds (Step 1 in Figure 1). For each point p, we determine the cell c it belongs to, by comparing the coordinates of p with the grid cell boundaries which are computed by adding or subtracting 500 meters from the coordinates of the cell centroid. We calculate the noise generated by the fishing vessel obtained by adding to the sound level associated with the horsepower of the boat, a contribution related to the actual speed of the vessel in p (see Equation (1)), and the noise due to the fishing activity, if it occurs in p. Then, we compute the propagation radius r (expressed in meters) and we build the *sound propagation bounding box* (Step 2 in Figure 1), defined by the minimum and maximum row and column identifiers that enclose all the cells affected by the noise generated by the vessel at p. These boundaries are obtained simply by adding or subtracting r from the row and columns identifiers of the cell c, `grid_r` and `grid_c`. Thanks to the row and column identifiers of the grid cell we avoid the use of the `ST_Intersects` operation, which is very time consuming. This approach allows retrieving the cells involved in the noise calculation in just 10 seconds for the entire dataset of June 2020. Next, we select all cells inside the bounding box and compute the distance between p and the cell centroids (Step 3 in Figure 1). We use this distance to estimate the transmission loss, which allows us to determine the received noise level in the selected cells. By grouping by cell id and time, we combine all the contributions of the points of the different trajectories (Step 4 in Figure 1), thus obtaining for each cell the received noise level (RL). These optimisations led to a more time-efficient pipeline that produces the same results as the implementation described in Section 2. In fact, the new execution time for June 2020 is reduced to 7 hours, making the code over six time faster than the original version, saving 37 hours of execution time (see Figure 3, where this is called *Optimised Pipeline*).

4. Partitioning and Parallelisation

To further optimise the performance of the pipeline we present an analysis of various partitioning and parallelisation techniques. In particular, selecting the cells affected by noise propagation for each point p (Step 3 in Figure 1) remains a computationally expensive operation. This complexity arises from the need to perform a `JOIN` operation between the table `PointBoundingBox`, which contains each vessel position along with its sound propagation bounding box, encompassing over 4 million points, and the table `Grid`, which consists of 43,508 cells. Consequently, the `JOIN` involves a computational effort equivalent to approximately 4 million $\times$ 43 thousand operations, making it inherently costly.

In Section 4.1, we examine table partitioning techniques in PostgreSQL, applying both range and hash partitioning strategies. In Section 4.2, we extend this approach by combining PostgreSQL partitioning with multidimensional tiling, focusing on the spatial dimension. Finally, in Section 4.3, we leverage the Citus extension of PostgreSQL to apply sharding and take advantage of its parallel query execution capabilities.

4.1. PostgreSQL Partitioning

The first technique we explore to enhance the execution of our code is *Table Partitioning* in PostgreSQL. This method consists in dividing a logically large table into smaller physical segments, with each partition being an independent table that stores a specific subset of the original data. PostgreSQL natively supports three forms of partitioning [5]: (i) *Range partitioning*, where the table is divided into *ranges* based on a key column or set of columns, with each partition containing non-overlapping ranges of values; (ii) *List partitioning*, which explicitly assigns specific key value(s) to each partition, allowing precise control over data distribution; and (iii) *Hash partitioning*, where the table is divided by applying a hash function to the partition key.

Table partitioning offers several advantages that significantly improve both performance and data management. It enhances query execution by allowing the database management system to filter out irrelevant partitions, thus speeding up query processing, especially for large datasets. Additionally, partitioning simplifies data management tasks such as archiving, purging, backup and restore operations. Furthermore, data loading is also

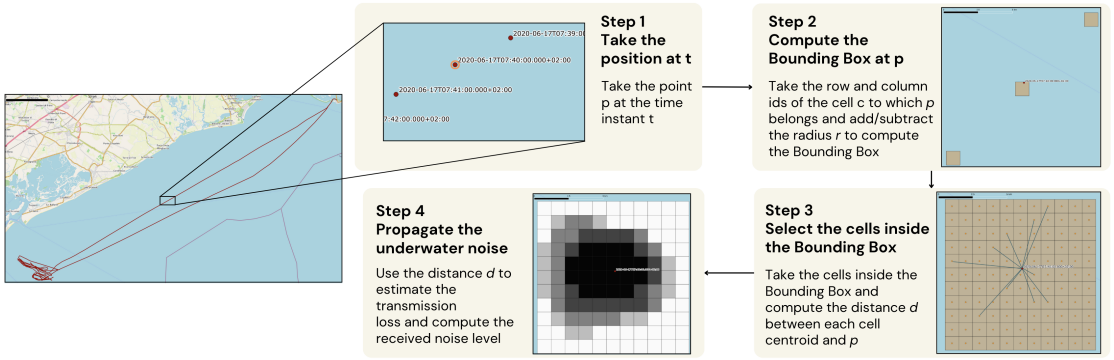


Figure 1: Main steps in the calculation of the noise maps.

more efficient since it can be parallelised, and indexing becomes faster as partitions reduce the scope of the data being indexed [15].

Range Partitioning on Time. To enhance the performance of our pipeline, as we have already remarked, we can improve the time execution of the `JOIN` operation between the table `PointBoundingBox` and the table `Grid`. To accomplish this task we partition the table `PointBoundingBox`, which is defined as follows:

```
CREATE TABLE PointBoundingBox AS (
    SELECT point_id,trip_id,mmsi,x,y,time,db_boat,
        grid_r-radius AS r_min,
        grid_r+radius AS r_max,
        grid_c-radius AS c_min,
        grid_c+radius AS c_max
    FROM UnnestTripWithCell );
```

where the `point_id` identifies the spatio-temporal point, `trip_id` is the identifier of the trip to which the point belongs, `mmsi` refers to the vessel performing the trip, `x` and `y` are the coordinates of the point, `time` specifies the date and hour of the point, and `db_boat` denotes the decibel level generated by the vessel at that point, based on its speed and activity. The remaining attributes represent the row and column identifiers used to construct the sound propagation bounding box including all the cells affected by the noise generated by the vessel at `point_id`.

We partition the table `PointBoundingBox` into four partitions based on time ranges to reflect the recurring weekly pattern: fishing activity is intense from Monday to Thursday, while significantly lower from Friday to Sunday. Additionally, this partitioning ensures a balanced disk usage across the partitions (see Table 1). We can create the partitioned table as follows.

```
CREATE TABLE PointBb_RangePart(LIKE PointBoundingBox)
      PARTITION BY RANGE(time);
```

Next, we create four time-based partitions corresponding to the four weeks of June 2020. After inserting the data into the partitioned table, the entries are automatically routed to the appropriate partition. Some statistics regarding the number of rows in each partition, along with their disk usage, are presented in Table 1 (left).

The query we want to optimise, which involves the partitioned table `PointBb_RangePart`, is the following.

```
SELECT eg.grid_r,eg.grid_c,pbb.trip_id,pbb.time,
    pbb.db_boat, SQRT(POWER(pbb.x-eg.centroid_x,2) +
      POWER(pbb.y-eg.centroid_y,2)) AS dist,
    eg.elevation,eg.ambient_noise,
    valueAtTimestamp(eg.alpha,time::DATE) AS alpha
FROM PointBb_RangePart pbb, Grid eg
WHERE eg.grid_r>=r_min AND eg.grid_c>=c_min AND
    eg.grid_r<=r_max AND eg.grid_c<=c_max;
```

This query returns, for each spatio-temporal point (`pbb.x`, `pbb.y`, `pbb.time`), the cells that are affected by the noise generated at that point by the fishing vessel, and computes the distance between the point and the centroids of these cells (Step 3 in Figure 1).

The query plan involves a combination of parallel and sequential scans to optimise the data retrieval process. The first step is a parallel append operation, which processes multiple partitions of the table `PointBb_RangePart` in parallel. Each partition (corresponding to a different time range) is accessed through a parallel sequential scan. The second part of the plan involves a bitmap heap scan on the table `Grid`, where rows are selected based on conditions that compare the grid's row and column identifiers with the corresponding bounding box identifiers from the partitions. Specifically, the query checks that the cells, identified by row `grid_r` and column `grid_c`, lie within the minimum and maximum row and column values of the bounding box. This comparison is optimised through bitmap index scans on `idx_grid_r` and `idx_grid_c`, each filtering the data based on the row and column values. In essence, the query plan performs a parallel scan of partitioned data,

Table 1
Statistics for the partitions by range on the `time` column (left) and by hash on the `mmsi` column (right).

	Range Partitioning		Hash Partitioning	
N. partition	Disk Usage	Rows	Disk Usage	Rows
1	81 MB	888,849	97 MB	1,090,196
2	115 MB	1,269,182	87 MB	977,776
3	90 MB	990,148	94 MB	1,059,732
4	93 MB	1,019,383	92 MB	1,039,858

followed by an efficient indexed search of the grid, ensuring faster query execution by narrowing down the relevant data points through partitioning and indexing.

By partitioning the table `PointBb_RangePart` while leaving the rest of the code unchanged, the entire pipeline now completes in just 2 hours and 25 minutes. The computation of sound propagation is 18.2 times faster than the first implementation (which took 44 hours) and 2.9 times faster than the optimised version without partitioning (which took 7 hours).

Hash Partitioning on MMSI. As a second partitioning experiment, we use the *hash partitioning* on the `mmsi` column of the table `PointBoundingBox`. We aim to divide the table `PointBoundingBox` into four partitions based on a hash function. The partitioned table can be created as follows.

```
CREATE TABLE PointBoundingBox_HashPart (LIKE
    PointBoundingBox) PARTITION BY HASH(mmsi);
CREATE TABLE PointBb_HashPart_1 PARTITION OF
    PointBoundingBox_HashPart FOR VALUES WITH (
    MODULUS 4, REMAINDER 0);
```

We have only reported the creation of the first hash partition. Next, we insert the values into the table `PointBoundingBox_HashPart`, which are automatically distributed across the partitions. Table 1 (right) presents some statistics on the number of rows and the disk usage of each partition. In this case, we can observe that the data distribution across the four partitions is more balanced compared to the partitions obtained through time-based range partitioning.

Now we use table `PointBoundingBox_HashPart`, instead of table `PointBb_RangePart`, in the query we want to optimize, presented in the previous subsection. The query plan is the same as that described for range partitioning and consists of a Parallel Seq Scan across the four partitions of the hash-partitioned table and a Bitmap Heap Scan on the table `Grid`. The execution time for June 2020 is 2 hours and 20 minutes, which is slightly faster than the range partitioning approach.

4.2. Space Tiling and Partitioning

Multidimensional tiling is a technique that partitions an n-dimensional domain into tiles of varying dimensions. This approach has several applications. For instance, multidimensional tiling can be applied to partition and/or distribute datasets across a cluster of servers. One key advantage of this partitioning mechanism is that it preserves spatial and temporal proximity, unlike traditional hash-based partitioning methods. This distribution reduces the amount of data that needs to be exchanged between nodes during query processing, a process commonly known as *reshuffling* [15].

In our work, we focus on tiling with respect to the spatial dimension. Specifically, we partition the positions of vessels based on their spatial locations. The tiling can be either *regular*, where all tiles are of equal size in each dimension, or *adaptive*, where the size of the cells may vary across dimensions. In the first case, we employ a regular tiling, constructing a uniform grid consisting of 4×3 cells, as shown in Figure 2a. To generate this grid, we used the MobilityDB function `spaceTiles`. The grid size was manually tuned to balance the trade-off between the number of partitions and the data distribution within each partition. Then we create the partitioned table along with the corresponding tables for the space tiles, by using the *List partitioning technique*.

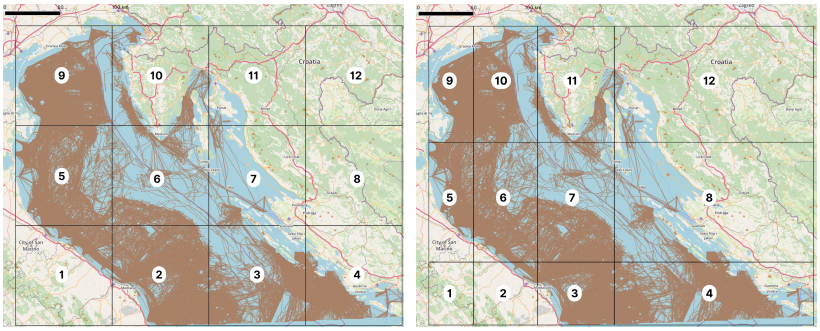
```
CREATE TABLE PointBoundingBox_RegGrid(LIKE
    PointBoundingBox) PARTITION BY LIST(TileId);
CREATE TABLE PointBb_RegGrid_1 PARTITION OF
    PointBoundingBox_RegGrid FOR VALUES IN (1);
```

Only the creation of the first tile is specified. Once the data is inserted into the partitioned table, the entries are automatically directed to their corresponding partitions. The limitation of this type of tiling is that it does not ensure balanced workload distribution across the tiles.

A possible solution to this issue is to use an *adaptive* grid, as illustrated in Figure 2b. In this case, we create a grid that divides the region based on the distribution of vessel points in the Northern Adriatic Sea. It is worth noting that some cells are smaller, as they contain a higher density of data points. Then, we partition the table `PointBoundingBox` according to the adaptive grid structure. The process of creating the partitioned table, along with the corresponding tables for the spatial tiles, follows the same steps as for the regular grid.

Table 2 presents statistics on the number of rows in each tile, as well as their respective disk usage, for both the regular and adaptive grids. The table clearly shows that the data partitioned according to the adaptive grid exhibits a more balanced distribution across the tiles compared to the regular tiling. However, certain tiles (specifically, tiles 1, 2, and 12) contain noticeably fewer data points, because they mostly cover the mainland.

The query we aim to optimise is the one presented



(a) Regular grid. (b) Adaptive grid.

Figure 2: Partitioning of vessel trip data with a regular grid and an adaptive grid.

Table 2
Statistics for the partitions by list on the `tileId` column.

	Regular Grid		Adaptive Grid	
Tile	**Disk Usage**	**Rows**	**Disk Usage**	**Rows**
1	18 MB	168,403	32 kB	0
2	95 MB	882,540	4000 kB	35,144
3	61 MB	571,392	53 MB	494,276
4	34 MB	314,051	92 MB	859,769
5	77 MB	715,011	48 MB	445,491
6	23 MB	212,307	40 MB	373,537
7	6176 kB	54,928	44 MB	404,563
8	32 kB	0	18 MB	165,596
9	117 MB	1,090,983	64 MB	596,401
10	17 MB	152,747	68 MB	633,210
11	688 kB	5,200	15 MB	143,051
12	32 kB	0	1944 kB	16,524

in Section 4.1. The query plan, like the previous ones, combines parallel and sequential scans to optimise data retrieval. The first step is a parallel append operation, which processes multiple partitions of the table `PointBoundingBox_RegGrid` concurrently. This is followed by a bitmap heap scan on the table `Grid`, where rows are selected based on conditions that compare the grid's row and column identifiers with the corresponding bounding box identifiers from the partitions. By tiling the space with the regular grid, the full pipeline is executed in 2 hours 46 minutes, while using the adaptive grid it completes in just 2 hours and 16 minutes, which slightly improves the techniques in Section 4.1.

4.3. Using Citus for parallelisation

Citus[3] is an extension of PostgreSQL designed to ease horizontal scaling, making it suitable for handling large datasets across multiple machines. It distributes both data and queries across a cluster, allowing users to leverage the power of a distributed system while maintaining compatibility with existing PostgreSQL tools. By using sharding and replication Citus scales PostgreSQL across several servers. *Sharding* is a method employed in distributed systems to divide data horizontally across multiple servers or nodes. It involves splitting a large dataset into smaller, more manageable pieces known as shards. Each shard holds a portion of the data, and collectively, they represent the entire dataset. Citus enables timeseries data to be scaled by combining PostgreSQL single-node declarative table partitioning with its distributed sharding capabilities, creating a scalable time-series database.

To optimise our pipeline, we first apply PostgreSQL range partitioning based on time, followed by distributing the partitions using Citus sharding mechanism. Here, we utilise Citus in a single-node cluster configuration,where a single PostgreSQL server employs Citus to locally shard the data (with the coordinator also acting as a worker). This configuration has been implemented on the machine described in Section 3 running Citus 12.1.6. As outlined in Section 4.1 we want to partition the `PointBoundingBox` table based on time ranges. The partitions can be defined using the following Citus function.

```
SELECT create_time_partitions (
    table_name := 'PointBoundingBox_RangePart',
    partition_interval := '1 week',
    start_from := '2020-06-01 00:00:00',
    end_at := '2020-06-30 23:59:59' );
```

The function above creates weekly partitions starting from the dates specified. Furthermore, the tables `PointBoundingBox` and `Grid` are distributed using Citus functions as follows.

```
SELECT create_distributed_table(
    'PointBoundingBox_RangePart', 'point_id');
SELECT create_reference_table('Grid');
```

The first function distributes the table `PointBoundingBox` into multiple horizontal shards on the `point_id` column. The second function distributes

[3]https://www.citusdata.com/

Table 3
Statistics for the partitions by range on column `time` with Citus.

Partition Table	Disk Usage	Rows
PointBoundingBox_RangePart_p2020w23	104 MB	704,384
PointBoundingBox_RangePart_p2020w24	137 MB	948,703
PointBoundingBox_RangePart_p2020w25	148 MB	1,033,152
PointBoundingBox_RangePart_p2020w26	145 MB	1,002,853
PointBoundingBox_RangePart_p2020w27	73 MB	478,470

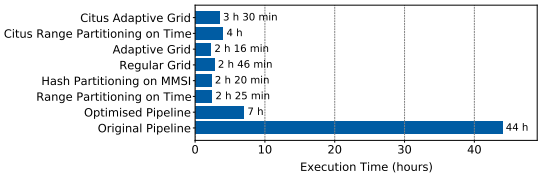


Figure 3: Execution times (in hours) of the implementations.

the table `Grid` into a single shard and replicates the shard to every worker node. Tables distributed in the second way are called *reference tables* and are employed to store data that requires frequent access by multiple nodes within a cluster. Table 3 presents statistics on the number of rows in each partition, along with their respective disk usage.

The objective, as in the previous cases, is to optimise the query described in Section 4.1. When executed using Citus, the query plan reveals that the workload is distributed across multiple tasks, with a total of 32 tasks created. Each task is assigned to a specific execution node, ensuring efficient parallel processing. Within each task, a gathering operation takes place, using multiple worker threads to further parallelise the workload. The query plan performs two main operations: the Parallel Append retrieves data from multiple partitioned tables, and the Bitmap heap scan identifies the relevant grid cells by verifying that their positions fall within the bounding box. This step is optimised by index-based filtering on the row and column attributes, further enhancing the performance. Using Citus the entire pipeline is executed in 4 hours. The computation of sound propagation is 1.75 times faster than the optimised pipeline without partitioning in Section 3) but it takes about 1.65 times longer than the partitioned PostgreSQL version (presented in Section 4.1).

We also utilise Citus for the space tiling presented in Section 4.2. Specifically, we partition the `PointBoundingBox` table according to the adaptive grid structure and distribute it using the Citus function previously discussed. The query plan is clearly similar to the case described above, with the workload distributed across multiple tasks. The main difference lies in the presence of 12 partitioned tables. The execution time for the entire pipeline, using Citus and distributing the points according to the adaptive grid, is 3 hours and 30 minutes, which is slightly faster than the partitioning by the `time` column. However, the pipeline incorporating Citus did not yield better performance compared to partitioning alone. As detailed in Cubukcu et al. [6], a single-node Citus configuration does not provide immediate performance benefits. Thus, single-node Citus is slightly slower than single server PostgreSQL due to distributed query planning overhead.

5. Concluding Remarks

Monitoring underwater noise pollution caused by human activities is crucial for preserving a healthy marine ecosystem. In this paper, we presented several optimisations to the underwater noise propagation pipeline presented in [3, 4]. The goal was to enhance efficiency, support scalability and reduce computational overhead.

Figure 3 collects the results of our experiments on June 2020 described in the previous sections. A clear improvement is observed between the original pipeline implementation presented in [3, 4] and the optimisations proposed in this work. In particular, the space tiling technique based on an adaptive grid provided the best result, which is over 19 times faster than the original running time. The pipeline incorporating Citus (single-node) did not yield better performance compared to partitioning alone, mainly due to distribution planning overhead.

As future work, we would like to investigate the Citus deployment in a multi-node cluster, to fully leverage its distributed processing capabilities. Additionally, we aim to conduct experiments with different partition numbers (e.g., 2, 4, 8, 16) to determine whether performance improves as the number of partitions increases, or if overhead dominates at some point. Moreover, in addition to space tiling with both regular and adaptive grids, quadtree-based spatial partitioning could be explored. Finally, we plan to analyse the entire year of 2020 to gain deeper insights into how partitioning and parallelisation perform with a larger volume of data, where their advantages are likely to become more pronounced.

This work enhances our original underwater sound propagation model with greater computational efficiency, offering a scalable solution for modelling underwater noise. By balancing estimation accuracy with computational effort, it can provide a convenient alternative to existing approaches, which often rely on hydrophone measurements or acoustic simulations and require extensive input data along with significant computational resources to manage complex calculations.

Acknowledgments

This publication was supported by the European Union - Next Generation EU - Project ECS000043 - Innovation Ecosystem Program *"Interconnected Northeast Innovation Ecosystem (iNEST)"*, CUP H43C22000540006.
This work took place within the framework of the DoE 2023-2027 (MUR, AIS.DIP.ECCELLENZA2023_27.FF project).

References

[1] H. Slabbekoorn, N. Bouton, I. van Opzeeland, A. Coers, C. ten Cate, A. N. Popper, A noisy spring: the impact of globally rising underwater sound levels on fish, Trends in ecology & evolution 25 (2010) 419–427.

[2] R. Williams, A. Wright, E. Ashe, L. Blight, R. Bruintjes, R. Canessa, C. Clark, S. Cullis-Suzuki, D. Dakin, C. Erbe, P. Hammond, N. Merchant, P. O'Hara, J. Purser, A. Radford, S. Simpson, L. Thomas, M. Wale, Impacts of anthropogenic noise on marine life: Publication patterns, new discoveries, and future directions in research and management, Ocean & Coastal Management 115 (2015) 17–24.

[3] G. Rovinelli, D. Rocchesso, M. Simeoni, A. Raffaetà, Using semantic trajectories for spatio-temporal characterisation of underwater noise, in: Proceedings of the 6th International Workshop on Big Mobility Data Analytics (BMDA 2024) - EDBT/ICDT Workshops, volume 3651, 2024.

[4] G. Rovinelli, D. Rocchesso, M. Simeoni, E. Zimányi, A. Raffaetà, Spatio-temporal characterisation of underwater noise through semantic trajectories, arXiv (2025). URL: http://arxiv.org/abs/2501.11131.

[5] The PostgreSQL Global Development Group, PostgreSQL 16.6 Documentation, 2024. URL: https://www.postgresql.org/files/documentation/pdf/16/postgresql-16-A4.pdf.

[6] U. Cubukcu, O. Erdogan, S. Pathak, S. Sannakkayala, M. Slot, Citus: Distributed postgresql for data-intensive applications, in: Proceedings of the 2021 International Conference on Management of Data, 2021, pp. 2490–2502.

[7] E. Zimányi, M. Sakr, A. Lesuisse, MobilityDB: A mobility database based on PostgreSQL and PostGIS, ACM Trans. Database Syst. 45 (2020).

[8] A. Petrizzo, A. Barbanti, G. Barfucci, M. Bastianini, I. Biagiotti, S. Bosi, M. Centurelli, R. Chavanne, A. Codarin, I. Costantini, M. Cukrov Car, V. Dadić, F. M. Falcieri, R. Falkner, G. Farella, M. Felli, C. Ferrarin, T. Folegot, R. Gallou, D. Galvez, M. Ghezzo, A. Kruss, I. Leonori, S. Menegon, H. Mihanović, S. Muslim, A. Pari, S. Pari, M. Picciulin, G. Pleslić, M. Radulović, N. Rako-Gospić, D. Sabbatini, G. Soldano, J. Tęgowski, T. Vučur-Blazinić, P. Vukadin, J. Zdroik, F. Madricardo, First assessment of underwater sound levels in the Northern Adriatic Sea at the basin scale, Scientific Data 10 (2023) 137.

[9] C. Chion, D. Lagrois, J. Dupras, A Meta-Analysis to Understand the Variability in Reported Source Levels of Noise Radiated by Ships From Opportunistic Studies, Frontiers in Marine Science 6 (2019) 714.

[10] M. Ainslie, Principles of Sonar Performance Modelling, Springer Praxis Books, Springer Berlin, Heidelberg, Berlin, Heidelberg, 2010.

[11] C. Erbe, A. Duncan, K. J. Vigness-Raposa, Introduction to Sound Propagation Under Water, Springer International Publishing, Cham, 2022, pp. 185–216.

[12] R. J. Urick, Principles of underwater sound 3rd edition, Peninsula Publising Los Atlos, California 22 (1983) 23–24.

[13] M. Picciulin, A. Petrizzo, F. Madricardo, A. Barbanti, M. Bastianini, I. Biagiotti, S. Bosi, M. Centurelli, A. Codarin, I. Costantini, V. Dadić, R. Falkner, T. Folegot, D. Galvez, I. Leonori, S. Menegon, H. Mihanović, S. Muslim, A. Pari, S. Pari, G. Pleslić, M. Radulović, N. Rako-Gospić, D. Sabbatini, J. Tegowski, P. Vukadin, M. Ghezzo, First basin scale spatial–temporal characterization of underwater sound in the Mediterranean Sea, Scientific Reports 13 (2023) 22799.

[14] B. Brandoli, A. Raffaetà, M. Simeoni, P. Adibi, F. K. Bappee, F. Pranovi, G. Rovinelli, E. Russo, C. Silvestri, A. Soares, S. Matwin, From multiple aspect trajectories to predictive analysis: a case study on fishing vessels in the Northern Adriatic sea, GeoInformatica 26 (2022) 551–579.

[15] M. Sakr, A. Vaisman, E. Zimányi, Mobility Data Science, Data-Centric Systems and Applications, 1 ed., Springer Cham, 2025. Due: 04 March 2025 (Hardcover), 04 March 2026 (Softcover), 04 March 2025 (eBook).