Archiving AIS messages in a Geo-DBMS

Martijn Meijers
Delft University of Technology
Julianalaan 134
Delft, The Netherlands
b.m.meijers@tudelft.nl

Wilko Quak
Delft University of Technology
Julianalaan 134
Delft, The Netherlands
c.w.quak@tudelft.nl

Peter van Oosterom
Delft University of Technology
Julianalaan 134
Delft, The Netherlands
p.j.m.vanoosterom@tudelft.nl

Abstract

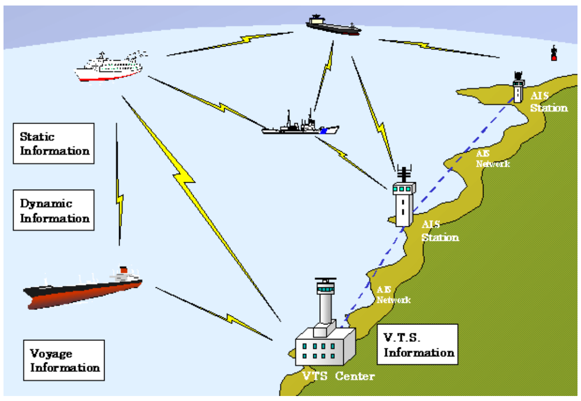
This paper reports on the result of two studies for using a geographical database management system for archiving Automated Identification System (AIS) message data. In this paper, we analyse the storage (using MongoDB and PostgreSQL) and we give a more in-depth description of a possible data model for archiving messages based on the bit vector type and functional indexes in PostgreSQL.

Keywords: AIS, MongoDB, PostgreSQL, database management system, bit vector, functional index

1 Introduction

AIS stands for Automated Identification System. The AIS system is mainly used for improving safety at sea and inland waters. Figure 1 illustrates that the system consists of different components. Depending on the cruising speed of the vessel, a transponder broadcasts its identity and position in intervals ranging from 2 seconds to 3 minutes.

Figure 1: Components of the Automated Identification System (AIS)



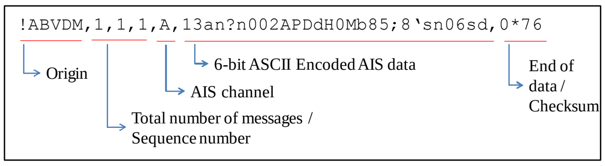
Source: www6.kaiho.mlit.go.jp/kanmon/

Rijkswaterstaat (RWS) maintains the main waterway network in The Netherlands. Within RWS Automated Identification System (AIS) messages are received in real time with the Dutch Inland AIS Monitoring Infrastructure (DIAMONIS) network. The current architecture of this system is not suited for archiving copious amounts of historic AIS messages.

2 AIS messages

Figure 2 shows an example AIS message. Figure 3 shows two different encodings and some parameters after decoding the message of Figure 2.

Figure 2: Sample AIS message encoded as NMEA sentence (cf. Raymond 2016 for details on decoding)



```
!ABVDM,1,1,1,A,13an?n002APDdH0Mb85;8‘sn06sd,0*76
```

Origin
Total number of messages / Sequence number
AIS channel
6-bit ASCII Encoded AIS data
End of data / Checksum

Figure 3: Data from the sample message in Figure 2, encoded as 6-bit ASCII, bit vector and some decoded parameters

The raw AIS data (6 bit ASCII encoded):
`13an?n002APDdH0Mb85;8'sn06sd`

As bit vector of 0’s and 1’s:
000001 00 001110100111011000111111011000 000...100

Some decoded parameters:

Key	Value	Binary representation
Message Type	1	000001_2 (= 1)
Repeat Indicator	0	00_2 (= 0)
MMSI	245207000	$001…0000_2$ (= 245207000)
…	…	…

3 Using a Geo-DBMS for storing historic AIS messages

The question we try to answer in this research is:

> What is a suitable data management strategy for archiving AIS message data in a Geographical Database Management System (Geo-DBMS)?

The following requirements were considered:

1. Volume of storage must be on par with size of raw data
2. Spatial-temporal queries answered reasonably efficient
3. We prefer that full history is archived, and often used parameters can be accessed efficiently.

In the research 2 systems were selected: MongoDB and PostgreSQL. MongoDB has been tested in a MSc thesis project (De Vreede 2016). The testing of PostgreSQL is described in a technical report (Meijers et al. 2016).

3.1 Initial test – Storage size requirements

As AIS messages are received frequently for many vessels, the total data volume is significant. Per week more than 80 million messages are received (leading to over 1.5GB of raw message data per week). Our initial test therefore focused on the storage requirements. A limited test set was loaded into MongoDB and PostgreSQL.

Table 1: Storage size requirements for storing AIS messages

Solution	Storage data type	Size (MB)	Factor to File
File	Raw AIS message + time stamp (tab separated)	27	–
MongoDB	JSON (WiredTiger)	58	2.1x
PostgreSQL	JSON	213	7.9x
PostgreSQL	JSONB	273	10.1x
PostgreSQL	Varchar	35	1.3x
PostgreSQL	Bit vector	35	1.3x

As can be concluded from Table 1, both PostgreSQL and MongoDB offer compact storage. However, the JSON based types in PostgreSQL require quite some storage, and our conclusion is that this data type is not suited for the use case of archiving unpacked AIS messages.

3.2 Indexing the data model based on the bit vector type in PostgreSQL

Indexing will also require storage space. We defined database functions to access the parameters of the AIS messages (stored as bit vector). Figure 4 shows an example function for decoding the MMSI. We defined 4 *functional indexes* on the table (see Table 2 for size).

Figure 4: A PL/PGSQL function for decoding the MMSI number (vessel identification)

```
-- get the MMSI number as integer
CREATE OR REPLACE FUNCTION ais_mmsi
(payload bit varying) RETURNS integer AS $$
BEGIN
  RETURN
  substring(payload
     from 9 for 30)::integer;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

Table 2: Functional indexes and their size

Index	Type	Column / Function	Size (MB)	Share (%)
MMSI	B-Tree	ais_mmsi(payload)	11	19
Type	B-Tree	ais_type(payload)	11	19
Time stamp	B-Tree	ts	11	19
Geometry	R-Tree	ais_point(payload)	25	43
			58	100

Table 3: Comparing bitvector storage together with the four indexes against raw file storage

	Size		Factor
Table with bit vector	35 MB	38%	
Indexes	58 MB	62%	
Total	93 MB		3.5x

Table 3 provides insights in the amount of space used for the bit vector data type, including indexes. Compared to the raw tab separated text file storage, 4 times more space is consumed.

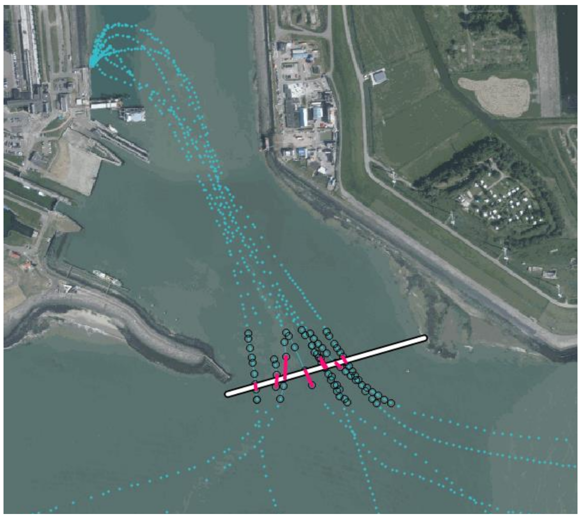
3.3 Querying the data model

For a complete range of use cases two queries are a starting point:

1. Find the last known position of a vessel.
2. Give the trajectory of a vessel (ordered by timestamp and subsequently connected with straight segments), see Appendix A.

The results of these queries can be visualized by using QGIS, an off-the-shelf GIS package. With the limited test data set we experienced real time query performance. Figure 5 shows that we count how many vessels crossed a line by extending the basic queries (the trajectory query is used).

Figure 5: Selecting tracks overlapping a line for counting how many vessels crossed. For performing this analysis, the trajectory query (Appendix A) is used as a building block.



4 Discussion

MongoDB offers compact data storage for AIS messages. The data model developed for PostgreSQL is a compact and viable option. Based on the bit vector data type a factor 4 more storage space than raw file storage is used. It allows efficient spatial queries (as messages are indexed).

5 Future work

In this study, we have dealt with AIS data that was already collected and then bulk loaded. However, archiving real-time data is different. An option is to structure the data in 2 tables: A heap (with not-yet-indexed recent data) and an indexed and clustered historic archive. How often to perform re-organization of the heap table into the historic archive?

Acknowledgements

This work has been carried out in the framework of the 'RWS-TUD Raamovereenkomst betreffende Samenwerking en Kennisuitwisseling op gebied van Ruimtelijke Informatievoorziening' (Reference 31103836).

References

Meijers, M, van Oosterom, P, Quak, W 2016, Management of AIS messages in a Geo-DBMS, GISt Report No. 71, in opdracht van de Raamovereenkomst Rijkswaterstaat - TU Delft, Technical report, Delft University of Technology, Delft, pp. 33, 2016.

Raymond, ES 2016, AIVDM/AIVDO protocol decoding. Available from: catb.org [1 February 2016]

de Vreede, I 2016, Managing historic Automatic Identification System data by using a proper Database Management System structure. Master's thesis, TU Delft.

Appendix A: Trajectory query

```
SELECT * FROM
(
-- step 1. Get the data of the vessels,
-- ordered by MMSI and then by timestamp
WITH ais_data AS
(
SELECT ts, ais_mmsi(payload) AS mmsi, ais_point(payload) AS
geometry
FROM
ais_bits_rws_tiny
WHERE
ais_type(payload) in (1,2,3) AND ais_mmsi(payload) = [MMSI]
AND
-- between certain period
ts > '[TMIN]' AND ts < '[TMAX]'
-- exclude records with (91,181) readings
AND
ais_point(payload) && st_setsrid('BOX(-90 -180, 90
180)'::box2d, 4326)
ORDER BY
mmsi, ts
)
-- step 3. Calculate distance + speed
SELECT
mmsi, start_ts, end_ts, happened_ts, duration_secs,
dist, CASE WHEN duration_secs <> 0 THEN dist / duration_secs
ELSE NULL END AS speed, geo_segment
FROM
(

-- step 2. Find next point in trajectory
-- and make line segment for every
-- part of the trajectory
SELECT
mmsi, start_ts, end_ts, tstzrange(start_ts,end_ts) AS
happened_ts, extract(epoch from (end_ts - start_ts)) AS
duration_secs, st_distance(st_transform(geom1, 28992),
st_transform(geom2, 28992)) AS dist,
st_makeline(geom1,geom2)::geometry(LineString, 4326) AS
geo_segment
FROM (
SELECT
mmsi, ts AS start_ts, lead(ts) OVER w AS end_ts,
geometry AS geom1, lead(geometry) OVER w AS geom2
FROM
ais_data
WINDOW w AS (PARTITION BY mmsi ORDER BY ts)
) AS q
) AS qq
) AS r;
```