DOI: 10.34185/1991-7848.itmm.2026.01.064

DEVELOPMENT OF A HIGH-LOAD PIPELINE FOR TRANSFERRING MARINE TRAFFIC DATA FROM SOURCE TO END USER IN REAL TIME

Sofiia Burimova[1] [ORCID], Hanna Mykhalchuk[2] [ORCID]

[1]Oles Honchar Dnipro National University, student, Ukraine

[2]Oles Honchar Dnipro National University,

Cand. Phys.-Math. Sci., Assoc. Prof., Dept, Ukraine

Abstract. *In modern maritime industry, vessel traffic monitoring is the most important and often required task, where failures in data delivery or instability of software components can lead to colossal economic losses. This paper considers the problem of processing AIS (Automatic Identification System) data streams in large volumes and in real time. The author proposes a pipeline architecture on the .NET platform, with acceleration of object deserialization with source-generated JSON, moving data using gRPC and SignalR with MessagePack, temporary storage of data portions in Redis, and delegating client visualization of results to GPU. The implementation of the subsystem made it possible to achieve a stable display of 20-25 thousand dynamic ship objects – approximately a quarter of the world's maritime traffic – with an instantaneous speed, sometimes outpacing the loading of map tiles. The considered approach can be scaled to other geographic information systems with high data input loads.*

Keywords: *performance optimization, data streaming, maritime traffic, AIS, geographic information system, monitoring, high-load system, .NET, gRPC, Redis*

Introduction. Maritime logistics is a critically important industry for global trade and economies on the scale of countries. From 75% to 90% of all freight transport takes place on water routes. Currently, digital transformation is taking place aiming to standardize management, make faster decisions, monitor safety, and accelerate communication between ships, ports, stations, etc.

One of the most important tasks in the industry is monitoring vessels, in particular their location, technical condition and planned actions. To exchange such data between ships, the standard AIS (Automatic Identification System) communication protocol has long been used, where messages are sent at intervals from every few seconds to several minutes [1]. With the number of active ships in the world (slightly over 100 thousand), the need to receive real-time updates, and the

presence of a generally accepted protocol through which these updates are shared quite often, monitoring is probably the first task to undergo software automation.

At the same time, its improper implementation - software instability, inability to keep up with data volumes, data arrival delays - most directly leads to physical and operational losses, and as a result - to very significant financial ones. A day of vessel delay during delivery costs from $150 thousand, annually in total the delays cause $14 billion in losses to the global maritime industry, collisions - another $20 billion per year [2]. Taking into account the situation described above, high-performance and fault-tolerant systems capable of visualizing global AIS data flows without delays are a relevant type of software projects.

The main goal of this work is to develop the transmission of real AIS data streams in the existing software project of a geographic information system, from listening to communications to displaying ship markers on the map, with latency as close to real-time as possible and without harming the performance of other parts of the system.

To listen to AIS data streams, a separate microservice with a constantly running web socket was added into the architecture. This separation is decided due to a significant difference of the principle of how web sockets work, compared to Dependency Injection in ASP. NET Core [3], and therefore to exclude the interruption of listening by some DI component or delays due to the functioning of many other processes in the main API.

The source of AIS data in the project is an open-source API aisstream.io. It delivers AIS messages not from all vessels where the protocol is enabled, only from 20-25 thousand, which is at most a quarter of the world's traffic, but it is positively distinguished by simple connection setup and completely free access, which became the reason for this choice.

In the AIS specification, there are only 27 types of messages [1], but for the implemented subsystem, 6 are enough: 2 types for vessel locations, 2 for properties and 2 for safety incidents. Selection of these particular types was specified at the stage of connecting the web socket to the aisstream.io API.

Messages come from the data source already in JSON format. With a flow rate of several hundred messages per second, deserializing them the way common for .NET would be unproductive, since reflection is used there and it decreases performance the most. Therefore, a JSON context was defined for message classes so that the deserialization code would be generated at compile time. At this stage, the number and structure of classes fully correspond to the JSON that comes from the web socket. The specific type of each message is determined by reading the type name bytes at its beginning.

Next, AIS messages need to be sent to the main project API. The gRPC protocol was chosen for this channel due to its binary nature, which is 70-80% more efficient than HTTP, and its reputation as a good choice for communication between services, especially at high load of hundreds of objects per second [5]. When using gRPC, it is necessary to define data models using the proto3 syntax [5], from which the C# classes are generated at compile time.

It was decided to use new message classes for gRPC, since the classes from the previous stage were too complex, had a lot of nesting, and barely differed within one purpose (location/ship properties/safety). For gRPC 4 message classes have been developed: one for each purpose and a wrapper with fields common for any message.

AIS data comes to the main API into one of the background services constantly as it arrives. But the same continuous flow cannot be channeled further to the browser for visualization, because that approach would lead to almost instant memory exhaustion. Batches must be gathered on the server side before being sent further in between some time intervals [3], so there is a need for temporary data storage. The interval between batches was decided as 2 seconds, and to work with streaming loads so significant, the high-performance NoSQL database Redis was undoubtedly chosen as the storage.

The data needed for visualization of markers includes coordinates, type of a vessel (for color coding), and the key that logically joins all this is MMSI (Maritime Mobile Service Identity) - the identifier of a vessel in communications. Coordinates are taken from messages about vessel locations, types - from messages about their properties, and safety messages do not participate in the formation of markers. In

Redis, the data is organized into one hash table, so that all the information for filling the map can be retrieved with a single command [4]. The keys are composed of MMSI and property names, and the corresponding values are atomic, as shown in Fig. 1. These key-value pairs are created, or more often - updated, constantly as AIS objects arrive from the gRPC channel.

```
46029) "367480140:type"
46030) "18"
46031) "232006684:type"
46032) "11"
46033) "538010698:type"
46034) "20"
46035) "244020442:long"
46036) "5.0640266666666669"
46037) "257093820:long"
46038) "6.7857783333333339"
46039) "319173000:long"
46040) "9.4014266666666675"
46041) "314836000:lat"
46042) "44.367333333333335"
46043) "244690959:type"
46044) "7"
46045) "244013558:long"
46046) "4.7560966666666662"
46047) "538010198:long"
46048) "-15.398741666666666"
46049) "255768000:lat"
```

Figure 1 – Output fragment of the contents of a Redis hash table with visualization data

For sending data further to the client side, a SignalR channel was used, since in the current project its textbook usage scenario takes place: an ASP.NET application which should send a large number of records to clients at a speed as close to real-time as possible [3]. Another performance improvement by 40-60% more was achieved by configuring the SignalR channel to use the binary protocol MessagePack.

For simplicity in the client-side processing, this stage of data transfer was intended so that an object directly corresponds to a marker on the map. The answer is an only class with the fields MMSI, latitude, longitude and vessel type identifier. To serialize its objects with MessagePack, fields must have ordinal numbers set. The objects arrive to the client side as arrays of field values in that set order.

Entries of the Redis hash table are aggregated into objects of the marker class described above in another background service every 2 seconds, as mentioned earlier. Since the values in pairs are atomic, no connection of properties related to the same ship is implemented at the DB level, and the order of pairs is random, it is necessary

to construct objects in the server-side code. MMSI is extracted from keys, which in Redis are strings [4], and a dictionary is filled, where the values are marker objects described above. When receiving another entry from the hash table from Redis, the value of the corresponding property is set on the object in the dictionary by the current MMSI. After iterating through all the hash table entries, collection of values of the dictionary is sent to client side via SignalR.

When received at the client side, data arrays are not converted to GeoJSON collections in the main thread. Given their amount, which in a couple of minutes of application operation exceeds 20 thousand, there would be lags occuring every time - that is every 2 seconds. The conversion is outsourced to WebWorker, where it is performed asynchronously and then a fully prepared GeoJSON is returned to the main thread.

An attempt to display 20-25 thousand markers on the map with the usual DOM or even SVG tools from the Leaflet GIS framework used in the project is far beyond its limit, which immediately led to a crash of the client part or even the entire browser. Linking the leaflet.glify plugin was critical to the operability of the entire data transmission pipeline. It delegates GeoJSON rendering to WebGL, which in turn means GPU rendering, and this way finally displays the working ship markers on the map. With each new data arrival, all previous markers are deleted, because via WebGL it is not possible to change the position of existing images. But this constant deletion with redrawing does not harm the performance of the finished visualization, the most densely filled fragment of which is shown in Fig. 2. Markers are the best performing map layer, loading properly even during fast interactions, when other layers, even the base map tiles, may update with a slight delay.

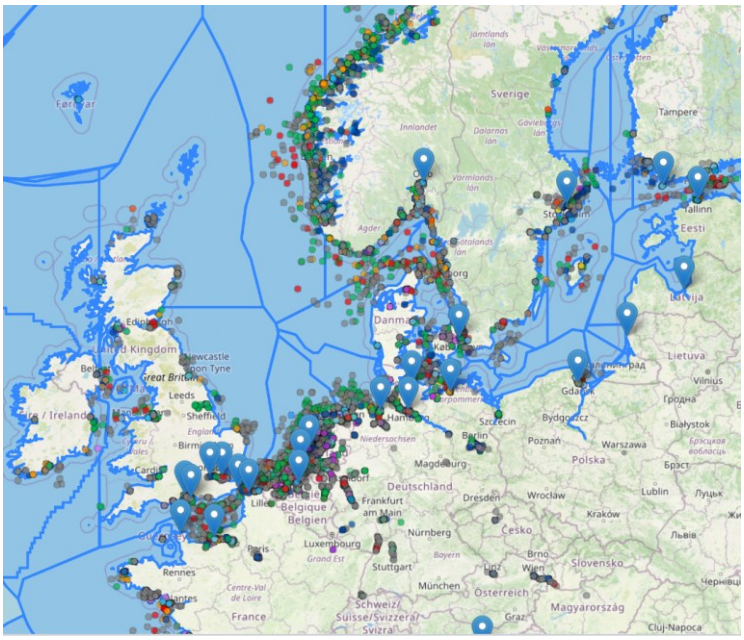

Figure 2 – Map fragment filled with markers the most

Conclusions

Data streaming pipeline architecture of the developed in the work completely solves the problem of visualization of a significant part of the global AIS traffic in real time. The achieved performance is way beyond limits of more traditional data transfer protocols and web visualization tools capabilities. It is expected that if AIS sources with greater coverage, up to the complete maximum, get included in the project, the developed subsystem will demonstrate the same speed and stability.

ЛІТЕРАТУРА / REFERENCE

1. B. J. Tetreault, Use of the Automatic Identification System (AIS) for maritime domain awareness (MDA). Proceedings of OCEANS 2005 MTS/IEEE. – 2005. P. 1590 – 1594. DOI:10.1109/OCEANS.2005.1639983.
2. J. Verschuur, J. Lumma & Jim W. Hall, Systemic impacts of disruptions at maritime chokepoints. Nature Communications – 2025. P. 3 – 8. DOI:10.1038/s41467-025-65403-w.
3. Kleppmann, M. Designing Data-Intensive Applications. O'Reilly. – 2017. P. 451 – 456, 459 – 467. ISBN: 9781491903063. URL: https://www.oreilly.com/library/view/designing-data-intensive-applications/9781491903063/
4. Dr. Josiah L C., Redis in Action. O'Reilly. – 2017. P. 48 – 54. ISBN: 9781935182054. URL: https://www.oreilly.com/library/view/redis-in-action/9781617290855/
5. A. Giretti, Beginning gRPC with ASP.NET Core 6. Apress – 2022. P. 85 – 91, 122 – 148. DOI:10.1007/978-1-4842-8008-9.

РОЗРОБЛЕННЯ ВИСОКОНАВАНТАЖЕНОГО КОНВЕЄРА ПЕРЕДАЧІ ДАНИХ МОРСЬКОГО ТРАФІКУ ВІД ДЖЕРЕЛА ДО КІНЦЕВОГО КОРИСТУВАЧА В РЕЖИМІ РЕАЛЬНОГО ЧАСУ

С.К. Бурімова, Г.Й. Михальчук

Анотація. *В сучасній морській справі моніторинг руху суден є найважливішою і найчастіше затребуваною задачею, де збої у доставці даних або нестабільність компонентів ПЗ можуть призвести до колосальних економічних втрат. В роботі розглядається проблема обробки потоків даних AIS (Automatic Identification System) у великому обсязі та режимі реального часу. Автором запропонована архітектура конвеєра на платформі .NET, з прискоренням десеріалізації об'єктів завдяки source-generated JSON, переміщенням даних за допомогою gRPC та SignalR з MessagePack, тимчасовим збереженням порцій даних у Redis та переведенням клієнтської візуалізації результатів на GPU. Впровадження підсистеми дозволило досягти стабільного зображення 20-25 тисяч динамічних об'єктів суден – приблизно чверті світового морського трафіку – з миттєвою швидкістю, що інколи випереджає завантаження фрагментів мапи. Розглянутий підхід може бути масштабований на інші геоінформаційні системи з великими навантаженнями надходження даних.*

Ключові слова: *оптимізація продуктивності, потокова обробка даних, морський трафік, AIS, геоінформаційна система, моніторинг, високонавантажена система, .NET, gRPC, Redis.*