Toward Smart Wireless Ocean Observation System using Distributed Computing

Ngoc Thanh Nguyen

14.11.2024

1 Introduction

Machine learning has achieved remarkable success across various domains, including healthcare [28], agriculture [26], and manufacturing [29]. These advancements are largely due to the progress in sensor technologies, which can collect vast amounts of data, reaching hundreds of gigabytes per day. Traditionally, machine learning models are trained using a centralized architecture, where data is gathered in a single supercomputer and the training process occurs on the same computational system. However, this approach presents several challenges, such as data storage, communication bottlenecks, and hardware limitations [31].

The SFI Smart Ocean project[1], established in late 2020, aims to develop an advanced wireless ocean observation system. As illustrated in Figure 1, the Smart Ocean system is based on a network of smart underwater sensors that can collect real-time and multi-parameter marine data. The collected data can be about environmental parameters, such as pressure, temperature, and ocean currents; structural parameters, including offshore structures integrity; and marine life parameters like animal communication sounds. The collected data is transmitted from the sensors to sea surface communication hubs via Underwater Acoustic Communication (UAC), then relayed to a central data management system using satellite signals. UAC is much cheaper than cabled systems, but it has low communication bandwidth which becomes a communication bottleneck of the system [16]. Finally, various application services will be developed to support data analytics for effective and sustainable ocean resource management use.

Our previous study points out that the challenging regarding communication bottlenecks is particularly evident in wireless ocean monitoring systems [16]. UAC suffers from low bandwidth, making it nearly impossible to transmit images or videos under the sea. Additionally, data transmission via UAC is highly energy-intensive, consuming up to 100 times more energy than data collection itself. Since most underwater sensors rely on limited battery power, the practice has applied various methods to reduce power consumption, especially when it comes to communication demand. It is also important to mention that maintaining these sensors can be both technically challenging and costly. Therefore, it is important perform more local computation as close to data collection sources and transmit less data across the network of the system. Furthermore, transmitting the data via satellite links, the only option for offshore areas, is expensive [16], demanding the reduction of communication overhead.

In order to reduce communication overhead as much as possible, only important events happening in the sea will be transmitted to the cloud. We need to realize this autonomous decision-making functionality at underwater sensors. This functionality is one of the crucial steps to make wireless ocean observation systems smart.

There are two problems to be addressed: where in the Smart Ocean network this decision-making component is implemented and how the functionality of filtering important is realized. The SFI Smart Ocean project has figured out that it is possible to deep learning models directly at underwater sensor hubs to facilitate real-time decision-making. The remaining goal is how to implement the functionality of filtering important events at underwater acquisition sources, which will be addressed in this project. Underwater sensor hubs are cluster of sensors. For example, in Figure 1, four sensors illustrated at the left of the sensor network can be grouped into one cluster.

Marine data is very contaminated due to many factors including limitations of sensors per se, biofouling, or even human errors [16, 17]. Basically, both bad data and events can be viewed as anomalies, which are significantly deviate from the normal patterns of the data [6]. It is possible to classify events and bad data by observing the data acquisition layer and how anomalies are supposed to be detected in

[1] https://sfismartocean.no/

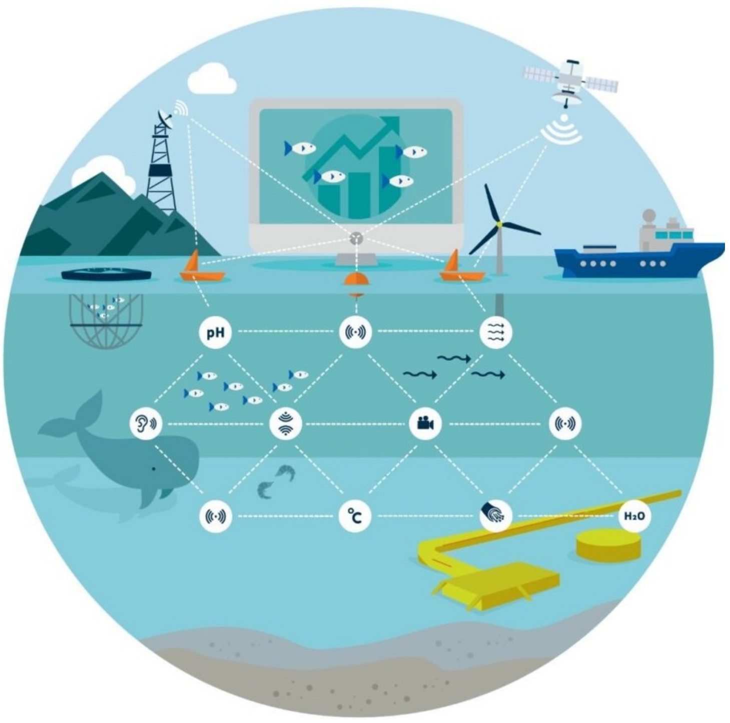


Figure 1: Topology of the Smart Ocean system [16]

the topology. Figure 2 illustrates how underwater sensors are distributed or deployed in wireless ocean observatory system. Assume that there are two events happen at the sea, oil leakage and the entry of a submarine. When these events occur, the sensors (for example, measuring ocean pressure) circled in red report anomalies. However, if a sensor encounters a problem such as communication issue or generating noise, and thus leading to bad data, only that sensor (circled in green) reports anomalies. This project leverages that observation to realize the functionality of filtering important events in the sea.

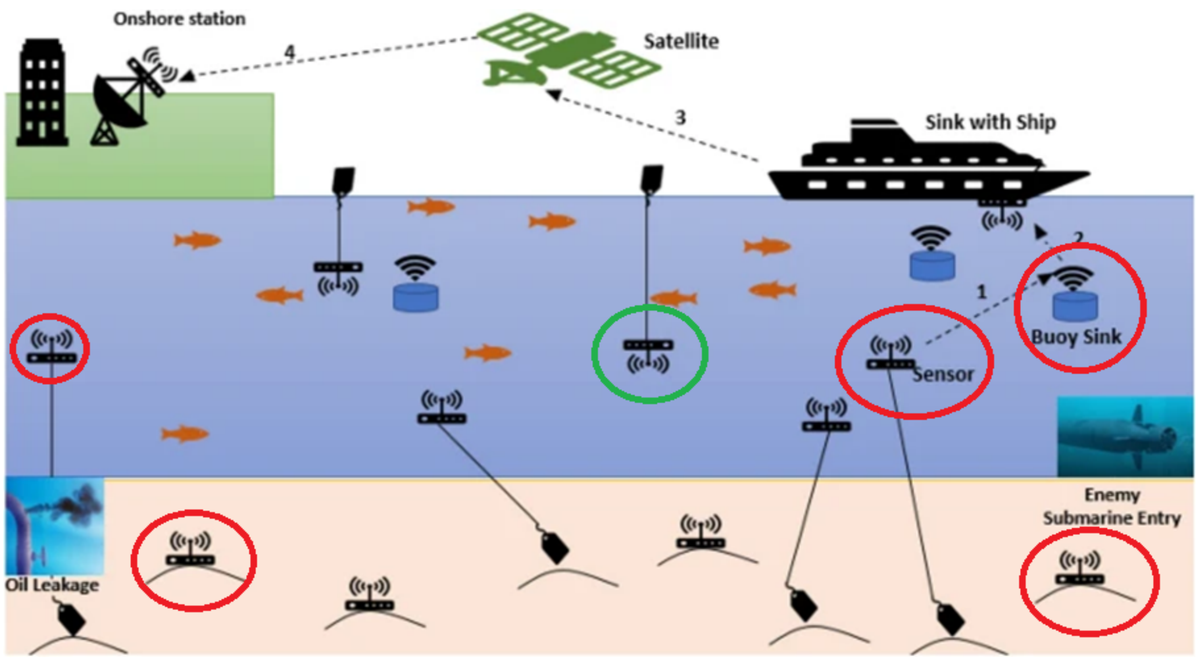


Figure 2: The Data Gathering Framework in Underwater Wireless Sensor Network (Figure adapted from [9])

The goal of performing the envisioned autonomous decision-making process at underwater data acquisition sources align very well with realizing a distributed computing network for the Smart Ocean system. With the use of the MapReduce programming model on the Hadoop framework, it is possible to perform computation at data nodes and then transfer results to worker nodes for final decision-making. If the transmitted information is small enough and sufficient, the goal of filtering events at underwater acquisition source can be fulfilled. To this end, this project aims to investigate MapReduce on Hadoop to realize the functionality.

The rest of the report is organized as follows: Section 2 discusses core concepts upon which the application is built in this project. Section 3 explains how the application is developed. Section 4 shows the performance of MapReduce jobs for the application when using and not using Combiner and discusses the impact of Combiner on the execution. Finally, Section 5 concludes the report.

2 Background

This section covers the core concepts behind the project. First, it introduces the Hadoop framework (Section 2.1). Next, it explains how marine data is collected and why automated data quality control is needed to filter important events correctly (Section 2.2). Finally, it discusses anomaly detection in time series data, which is essential for building the project's application (Section 2.3).

2.1 The Hadoop framework

Hadoop is an open-source framework developed under the Apache Software Foundation, designed for distributed storage and processing of large data sets across clusters of computers using simple programming models. It is widely adopted in big data analytics due to its robustness, scalability, and cost-effectiveness. Hadoop comprises several core components that enable efficient data handling. The Hadoop Distributed File System (HDFS) is responsible for storing data across multiple machines, dividing large files into smaller blocks that are distributed throughout the cluster. HDFS ensures data reliability through block replication across nodes, providing fault tolerance and high data throughput [30].

Hadoop operates by dividing large data sets into smaller, manageable chunks and distributing them across a cluster of machines for parallel processing, ensuring efficient handling and analysis of big data. It uses HDFS to store data in a distributed manner, breaking files into blocks and replicating them across multiple nodes to provide fault tolerance and high data availability. The MapReduce programming model, as illustrated in Figure 3, processes data in two main phases: the *Map* phase, which filters and processes the input data to create intermediate key-value pairs, and the *Reduce* phase, which aggregates and compiles the final output from the intermediate data.

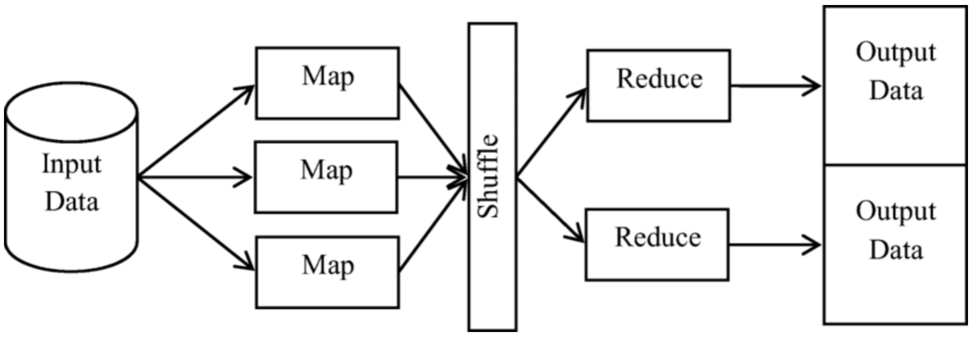


Figure 3: The MapReduce programming model [2]

The YARN component manages resources within the cluster, enabling dynamic allocation and efficient scheduling of tasks. This combination of distributed storage and parallel processing allows Hadoop to handle vast amounts of data quickly and reliably, making it a core technology in the big data ecosystem.

Hadoop offers significant advantages, such as scalability, allowing it to scale from a single server to thousands of machines, each contributing local computation and storage. Its fault tolerance ensures data is accessible even if a node fails, due to data replication across nodes. Hadoop is also cost-effective as it runs on commodity hardware, making it a practical solution for managing large-scale data. Additionally, it provides flexibility in handling structured, semi-structured, and unstructured data types.

Hadoop has a wide range of use cases, from data warehousing for business intelligence [1] to log processing for analyzing server logs and understanding web traffic [11]. It is also used in fraud detection by processing extensive data sets to identify anomalies in smart grid systems [15] and in machine learning to enable the training and processing of models on large-scale data [31].

A **Combiner** is an optional optimization component within the MapReduce paradigm that improves the efficiency of data processing. Combiner operates as a "mini-reducer" between the Map and Reduce phases to reduce the amount of data transferred across the network, thus minimizing network congestion and enhancing job performance. When the Map function generates intermediate key-value pairs, the Combiner is applied to these outputs locally on the mapper node. By performing local aggregation or summarization of the data before it is sent to the Reduce phase, the Combiner reduces the volume of data shuffling required.

Figure 4 illustrates the difference between the use of a combiner in a MapReduce process. On the left side, the MapReduce process is depicted without a combiner, while on the right side, the combiner is included. In the process without a combiner (left side), each mapper processes an input split and produces intermediate key-value pairs. These intermediate outputs are then directly sent to the reducers. The reducers perform the final aggregation and generate the output. Without a combiner, a large amount of intermediate data is transferred across the network from mappers to reducers, which can result in network congestion and affect the overall performance. In contrast, the process with a combiner (right side) includes an additional step where each mapper still processes input splits and produces intermediate key-value pairs. However, instead of sending all intermediate pairs directly to the reducers, a combiner is applied locally on the mapper nodes. The combiner aggregates intermediate data locally, significantly reducing the volume of data that needs to be transferred to the reducers. This approach not only minimizes network congestion but also improves the efficiency and speed of the MapReduce process.

By including a combiner, the overall performance of the MapReduce job can be enhanced, especially when there are large amounts of intermediate data. This optimization can be particularly beneficial in applications where data aggregation can be partially handled at the mapper level.

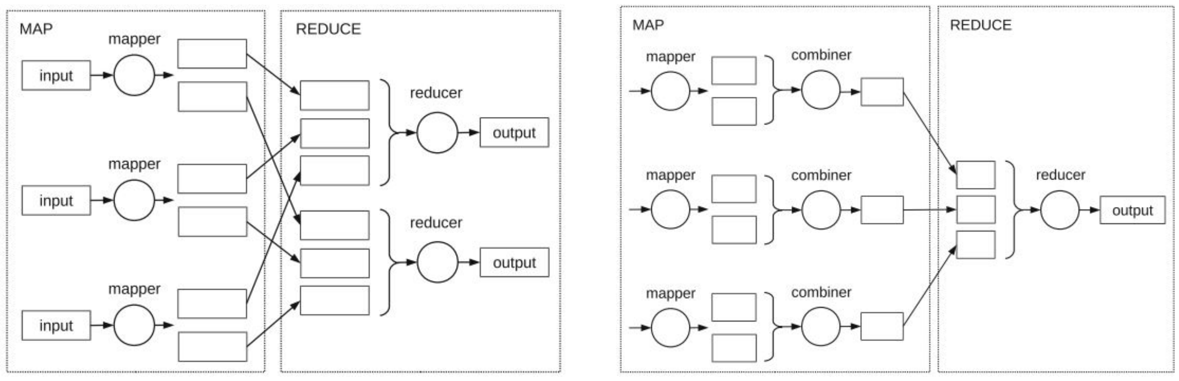


Figure 4: Data flow of MapReduce when not using Combiner (left) and using Combiner (right) [23]

2.2 Marine data

2.2.1 Data Collection

Since 2016, ocean industries have generated approximately 1.5 trillion dollars annually for the global economy and it will be double by 2030 [19]. We can find the use of marine data in applications like detecting subsea pipeline leaks [5], diagnosing offshore wind turbine failures [32], and enhancing maritime security by detecting illicit activities, such as human trafficking or smuggling [22]. They are critical to drive the success of marine industries.

Underwater sensors that collect marine data are battery-powered and they can operation from a few weeks to many years [3, 18]. In the past, collected marine data was stored locally and retrieved later by human operators [16]. Recently, the use of fiber optic cables or underwater acoustic communication enable the data to be transmitted in real time. Modern ocean observation systems typically consist of multiple sensors [25]. To optimize bandwidth, data collected from different sensors are aggregated before transmission, which can happen at communication hubs [13, 16].

2.2.2 Data quality control

Marine data is it more contaminated than data in other domains [17]. Underwater sensors generate more noisy data when the battery is low [8], and biofouling phenomenon can alter measurements unexpectedly [7]. Ensuring reliable data is critical for effective data-driven decision-making results. Checking the quality of the data is typically done by humans and this can take up to six months to complete [16, 18]. Our industrial and research collaborators have emphasized the need for automated data quality control [12, 16].

Our research group has proposed a novel anomaly detection called AdapAD to identify bad data in marine data streams [17]. The algorithm has been applied to perform data quality control from the data collected from the One Ocean Expedition automatically [20]. We are now collaborating with our industrial partner to deploy AdapAD into underwater sensors and the proposal can be found in the appendix. In the scope of this project, however, another anomaly detection algorithm will be used due to the high complexity of converting AdapAD from Python to Java, which is out-of-scope of this project.

2.3 Anomaly detection

A time series is a sequence of individual data point collected at equal-spaced timing. A time series can be unidimensional if each data point is a singular value, or multidimensional if the data point is represented as an array of values. This work assumes that each sensor collect one marine parameter, so the time series is univariate. **Anomaly detection in time series** is the process of identifying anomalies in a given time series. The anomalies can be subsequences of T which deviate from its frequent patterns. Readers are redirected to [6, 24] for detailed introductions and definitions.

Anomaly detection in time series is a very active research topic [21], with 100+ algorithms proposed in the last 60 years [27]. Anomaly detection algorithms can be groupped based on how they learn to classify normal and anomalous data: unsupervised, semi-supervised, and supervised learning. Unsupervised learning algorithms identify anomalies without labeled training data. They assume that anomalies have distinct characteristics from the normal data like shapes, or different data distributions. Supervised learning algorithms require labeled training data, i.e., both normal and anomalous data examples. Then, they learn how to separate those examples for detect anomalies in test data. Semi-supervised learning algorithms are trained on normal data. They flag data points too different from the learned patterns. Readers are redirected to [4, 27] about existing algorithms. In this project, Isolation Forest—introduced below—will be applied to detect anomalies and identify significant events in the data. Experiments in [27] show that this algorithm performs well on diverse datasets. Note that evaluating the accuracy of anomaly detection algorithms is beyond the scope of this project.

Isolation Forest (iForest) is an efficient algorithm for unsupervised anomaly detection, introduced by Liu et al. [14]. The key idea behind iForest is that anomalies, which are rare and distinct, are easier to isolate compared to normal data points. The algorithm constructs multiple decision trees–Isolation Trees–through the recursive partitioning of data points using randomly selected features and split values. The depth at which a data point is isolated within these trees serves as an indicator of its anomaly score. Data points that are isolated at shallow depths are considered anomalies, while those requiring deeper partitions are likely to be normal.

Isolation Forest has several key advantages that make it particularly suitable to the goals and challenges addressed by the Smart Ocean system. One major benefit is its efficiency for large datasets, as its time complexity is linear with respect to the number of samples. This characteristic is crucial for the Smart Ocean system, which aims to collect a vast amount of data in real-time [16]. Additionally, unsupervised learning is another strong point of iForest. It does not require labeled training data, making it effective for detecting anomalies where defining "normal" behavior is difficult. This feature aligns well with the challenges of the marine domain, where data quality control and detecting anomalies often rely on manual processes that can take up to six months to complete [18].

3 Methodology

The high-level architecture of the Smart Ocean system, as illustrated in Figure 1 and further detailed in Figure 5, provides a foundation for implementing a distributed computing network. Data is collected by various underwater sensors, which are organized into sensor hubs. In the context of a distributed computing network, these hubs function as data nodes. Each node can communicate with others to support tasks such as fault tolerance. The data collected at each node is transmitted to communication gateways—viewed as worker nodes in the distributed computing framework—via acoustic signals.

A key challenge arises at this stage due to the low bandwidth of acoustic signals, creating communication bottlenecks. Once the data reaches the communication gateways, it can be transmitted further via satellite links to cloud service gateways and eventually to cloud-based data spaces using the satellite. The satellite links offer significantly higher bandwidth compared to acoustic signals, enabling smoother data transmission beyond the underwater segment. However, it should be note that transmitting data via satellite is expensive. For the matter of economy, the amount of data transmitted should be reduced. In summary, one of the primary challenges for the Smart Ocean system is minimizing communication overhead, which is the central objective of the project. Finally, the data spaces shown in the figure contain software services which facilitate big data management.

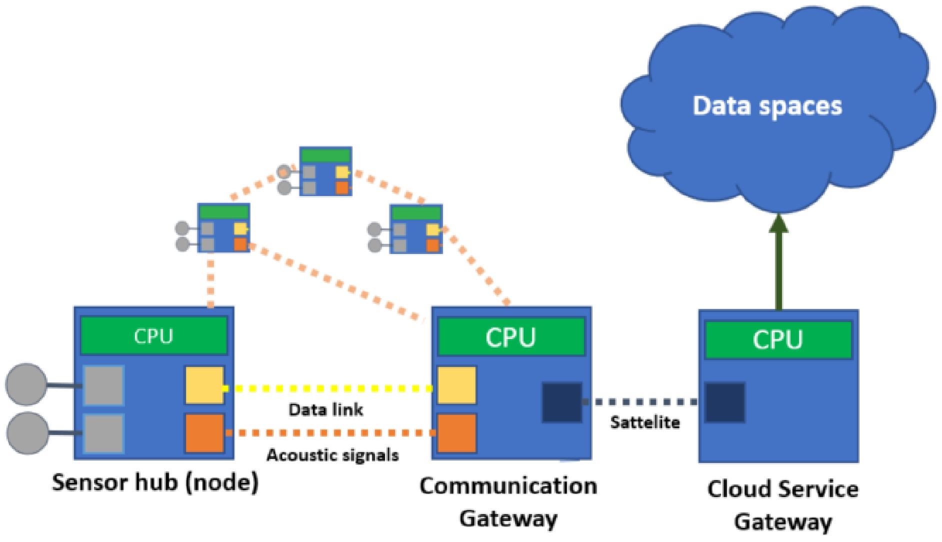


Figure 5: Topology of wireless smart ocean observation systems [10]

Sensor hubs and communication gateways are equipped with a Central Processing Unit (CPU), allowing for the possibility of processing data directly at the source of acquisition. Each of the sensor hubs can transmit the data collected by itself and forward from that of the peers to communication gateways. The role of communication gateways is to relay collected data further.

In this project, the primary goal is to develop a solution for the Smart Ocean system that reduces network communication overhead by leveraging a distributed computing framework. To achieve this, MapReduce on Hadoop will be employed to distinguish between bad data and significant events. Therefore, the Smart Ocean system, and more specifically the underwater acquisition component, will have the autonomy to decide which data worth transmitting to the cloud and thus contributing to the reduction of communication overhead. The code implementation of this application is available on GitHub at https://github.com/ntnguyen-so/SmartOcean-EventFiltering-Hadoop.

The workflow of the application is outlined in Algorithm 1. This algorithm assumes the use of a Combiner to optimize commuication need.

Algorithm 1 MapReduce Job for Classifying Bad Data and Events

```
Input: Time series data from multiple sensors in an ocean observatory system. The data collected of
each sensor is presented by a text file with each line format as follows: dataIndex, dataValue.
Output: List of data points classified as events.

Mapper Phase:
Input: Key-value pair, where key is the data index and value is the time series data from a sensor
Output: Intermediate key-value pairs where anomalies are identified

function MAPPER(dataIndices, dataValues)
  // Assume that iForest is used in this project
  anomalyScores ← CALCULATEANOMALYSCORE(dataValues)
  for each dataIndex in dataIndices do
    emit(dataIndex, dataValue_anomalyScore)
  end for
end function

Combiner Phase:
Input: (i) List of data points together with anomaly scores and (ii) Threshold to decide when a data
point is anomalous threshold.
Output: Aggregated list of anomalous data points.

function COMBINER(dataIndices, dataPoints with anomalyScores)
  for each dataPoint in dataPoints do
    dataValue, anomalyScore ← extract(dataPoints with anomalyScores)
    if anomalyScore > threshold then
      emit(dataIndex, dataValue)
    end if
  end for
end function

Reducer Phase:
Input: List of anomalous data points.
Output: List of data points identified as events.

function REDUCER(anomalousDataIndices, anomalousDataValues)
  for each anomalousDataIndex in anomalousDataIndices do
    if anomalousDataIndex occur more than once in anomalousDataIndices then
      emit(anomalousDataIndex, anomalousDataValue as event)
    end if
  end for
end function
```

If a Combiner is not used, the operations performed in the Combiner phase are incorporated directly into the Reducer phase as explained in Algorithm 2.

Algorithm 2 MapReduce Job for Classifying Bad Data and Events without Combiner

```
Input: Time series data from multiple sensors in an ocean observatory system. The data collected of
each sensor is presented by a text file with each line format as follows: dataIndex, dataValue.
Output: List of data points classified as events.

Mapper Phase:
Input: Key-value pair, where key is the data index and value is the time series data from a sensor
Output: Intermediate key-value pairs where anomalies are identified

function MAPPER(dataIndices, dataValues)
    // Assume that iForest is used in this project
    anomalyScores ← CALCULATEANOMALYSCORE(dataValues)
    for each dataIndex in dataIndices do
        emit(dataIndex, dataValue_anomalyScore)
    end for
end function

Reducer Phase:
Input: (i) List of data points together with anomaly scores and (ii) Threshold to decide when a data
point is anomalous threshold.
Output: List of data points identified as events.

function REDUCER(dataIndices, dataPoints with anomalyScores)
    for each dataPoint in dataPoints do
        dataValue, anomalyScore ← extract(dataPoints with anomalyScores)
        if anomalyScore > threshold then
            insertTo(dataIndex, anomalousDataIndices)
        end if
    end for

    for each anomalousDataIndex in anomalousDataIndices do
        if anomalousDataIndex occur more than once in anomalousDataIndices then
            emit(anomalousDataIndex, anomalousDataValue)
        end if
    end for
end function
```

4 Results and Discussions

To evaluate the performance of MapReduce jobs with and without the use of a Combiner, six input files, each containing 200,000 data points, were used. These files represent different marine parameters, with each file containing approximately 30% anomalous data points. The threshold to decide when a data point is anomalous is above 70%. Table 1 summarizes the performance of the MapReduce jobs in both scenarios across various performance metrics.

Table 1: Comparison of MapReduce performance while Combiner is used and not used

Field	**Without Combiner**	**With Combiner**
Number of bytes read (FILE)	26,085,041 bytes	605,225 bytes
Number of bytes written (FILE)	54,102,288 bytes	3,143,923 bytes
HDFS bytes read	800,580 bytes	12,534,210 bytes
HDFS bytes written	12,534,210 bytes	41,963 bytes
Launched map tasks	6	6
Launched reduce tasks	1	1
Data-local map tasks	5	6
Total time spent by all maps (ms)	8,231,809 ms	7,895,215 ms
Total time spent by all reduces (ms)	689,108 ms	493,992 ms
Map input records	1,200,000	1,200,000
Map output records	1,200,000	1,200,000
Map output bytes	23,685,321 bytes	23,685,321 bytes
Map output materialized (bytes)	2,800,024 bytes	605,255 bytes
Combine input records	0	1,200,000
Combine output records	0	41,909
Reduce input groups	17	38429
Reduce shuffle (bytes)	2,800,024 bytes	605,255 bytes
Reduce input records	200,000	41,909
Reduce output records	9,141	3,320
Spilled Records	400,000	83,818
GC time elapsed (ms)	6,717 ms	62,926 ms
Peak Map Physical Memory (bytes)	774,889,472 bytes	840,192,000 bytes
Peak Reduce Physical Memory (bytes)	213,295,104 bytes	300,785,664 bytes
Shuffle Errors	0	0

Based on the results obtained, observations with focus on communication overhead, computational demand, and runtime efficiency are made:

- **Communication overhead**
  - **File system**: The job without a Combiner reads significantly more data (26,085,041 bytes) compared to the job with a Combiner (605,225 bytes). This reduction indicates that the use of a Combiner effectively minimizes the data processed between the map and reduce phases. Furthermore, the job without a Combiner writes 54,102,288 bytes, whereas the job with a Combiner writes only 3,143,923 bytes. This reflects the Combiner's role in reducing intermediate data before the shuffle and sort phase.
  - **HDFS counters**: The job with a Combiner has fewer HDFS bytes written (41,963 bytes) compared to without a Combiner (12,534,210 bytes). This indicates less intermediate data transfer, leading to more efficient use of HDFS bandwidth and improved overall job performance.
- **Computational demand**
  - **Data volume**: There is a significant reduction in map output materialized bytes from 26,085,041 bytes (without Combiner) to 605,255 bytes (with Combiner), demonstrating the Combiner's efficiency in aggregating data early. Additionally, with the Combiner enabled, there are 1,200,000 input records and only 41,909 output records, suggesting that the Combiner effectively reduces the volume of intermediate records before reaching the reduce phase.

  - **Demands on the Reduce phase**: The job with a Combiner receives fewer input records (41,909) compared to without a Combiner (200,000), alleviating the load on the reducer and enabling more efficient processing. In addition, the number of spilled records is significantly lower with a Combiner (83,818) compared to without it (400,000), indicating reduced memory pressure and less need for intermediate data spilling to disk.
  - **Memory usage**: The peak map physical memory usage is slightly higher when using a Combiner (840,192,000 bytes) compared to without it (774,889,472 bytes), which may be due to the additional computation for combining data at the mapper level. The peak reduce physical memory usage also increases with the Combiner, from 213,295,104 bytes to 300,785,664 bytes.
  - **System load**: The job with a Combiner results in fewer spilled records (83,818 vs. 400,000), which suggests that the overall system load is lighter. This can lead to better use of system resources and reduced disk I/O, contributing to more efficient performance in high-data-volume environments.

- **Runtime efficiency**
  - **Total Time Spent by All Map and Reduce Tasks**: The job with a Combiner shows a reduction in total time spent by map and reduce tasks, indicating an improvement in runtime efficiency. Specifically, the total time spent by all map tasks is 8,231,809 ms without a Combiner and 7,895,215 ms with a Combiner. The total time spent by all reduce tasks also drops from 689,108 ms to 493,992 ms. This highlights how using a Combiner can shorten the overall execution time by decreasing the amount of data shuffled to reducers and processed.
  - **GC Time Elapsed**: The garbage collection (GC) time is higher when using a Combiner (62,926 ms) compared to without a Combiner (6,717 ms). This increase might be attributed to the additional overhead of combining operations at the map phase, which could lead to higher memory consumption and more frequent garbage collection.

Using a Combiner in MapReduce jobs can bring substantial benefits but also comes with certain trade-offs. Below is an analysis of the advantages and potential downsides:

- **Advantages of Using a Combiner**
  - **Reduced Data Transfer**: The use of a Combiner significantly reduces the volume of intermediate data that needs to be transferred between the map and reduce phases. This leads to lower network I/O, as demonstrated by the reduction in the number of bytes written and read (e.g., 26,085,041 bytes read without a Combiner versus 605,225 bytes with a Combiner).
  - **Lower Spilled Records**: With fewer intermediate records, the Combiner helps minimize the number of records spilled to disk, reducing memory pressure and improving processing speed (e.g., 400,000 spilled records without a Combiner versus 83,818 with a Combiner).
  - **Reduced Load on Reducers**: By aggregating data early, the number of reduce input records is lowered, which can lead to faster processing in the reduce phase (e.g., 200,000 reduce input records without a Combiner compared to 41,909 with a Combiner).
  - **Improved Efficiency**: The decrease in data transfer contributes to shorter job execution times, as seen by the reduced time spent by all maps and reduces (e.g., total map time of 8,231,809 ms without a Combiner compared to 7,895,215 ms with a Combiner).

- **Potential Trade-offs of Using a Combiner**
  - **Increased Complexity**: Implementing a Combiner can add complexity to the MapReduce program. Developers need to ensure that the Combiner logic correctly preserves the associative and commutative properties required for accurate data aggregation.
  - **Memory and Resource Utilization**: While the Combiner can reduce data transfer, it may increase the memory usage of map tasks, as intermediate aggregation is performed before data is written out. This is reflected in the higher peak physical memory usage (e.g., 840,192,000 bytes with a Combiner versus 774,889,472 bytes without a Combiner).
  - **Overhead of Combining**: Although the overall job performance improves, there might be slight computational overhead from running the Combiner logic, which could impact scenarios where map output size is already minimal.

5 Conclusions

This project used the MapReduce model in the Hadoop framework to enable automated decision-making to filter important ocean events for the Smart Ocean system. Key findings from this implementation include:

- **Communication and Data management**: MapReduce has two main phases: map and reduce. Adding a Combiner phase helps by reducing the amount of data that passes from map to reduce. This leads to fewer bytes being read and written, and less data being shuffled and processed, which lowers communication costs and improves data handling across the network.
- **Processing efficiency**: The Combiner phase reduces the number of records the reducer has to process, which reduces memory load during sorting and transferring. This means fewer records are saved to disk and reduce tasks take less time. However, using a Combiner can increase memory usage and garbage collection (GC) time, especially in the mapper nodes. So, while overall processing time is faster, mappers handle a heavier load.
- **Task-specific requirements**: Choosing to use a Combiner should depend on the task's needs. For this project, reducing on communication costs and shuffle size is crucial, so the Combiner phase is helpful. However, it is important to consider the added memory use and gabage collection time.

Finally, assessing the computational demands on underwater sensor nodes with Hadoop has not yet been done. One should note that this assessment is out-of-scope of the project, and this will be a key area for future research to determine if Hadoop can be used efficiently on limited-resource underwater systems. Furthermore, the work of this project has opened up new research opportunity after my PhD.

References

[1] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop-gis: A high performance spatial data warehousing system over mapreduce. In *Proceedings of the VLDB endowment international conference on very large data bases*, volume 6. NIH Public Access, 2013.

[2] Shiva Asadianfam, Mahboubeh Shamsi, and Abdolreza Rasouli Kenari. Tvd-mrdl: traffic violation detection system using mapreduce-based deep learning for large-scale data. *Multimedia Tools and Applications*, 80(2):2489–2516, 2021.

[3] Fei Chai, Kenneth S Johnson, Hervé Claustre, Xiaogang Xing, Yuntao Wang, Emmanuel Boss, Stephen Riser, Katja Fennel, Oscar Schofield, and Adrienne Sutton. Monitoring ocean biogeochemistry with autonomous platforms. *Nature Reviews Earth & Environment*, 1(6):315–326, 2020.

[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[5] Tong-Chuan Che, Huan-Feng Duan, and Pedro J Lee. Transient wave-based methods for anomaly detection in fluid pipes: A review. *Mechanical Systems and Signal Processing*, 160:107874, 2021.

[6] A. Cook, G. Mısırlı, and Z. Fan. Anomaly detection for IoT time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2019.

[7] M. C. Domingo. An overview of the internet of underwater things. *Journal of Network and Computer Applications*, 35(6):1879–1890, 2012.

[8] A. Fawzy, H. Mokhtar, and O. Hegazy. Outliers detection and classification in wireless sensor networks. *Egyptian Informatics Journal*, 14(2), 2013.

[9] Osho Gupta and Nitin Goyal. The evolution of data gathering static and mobility models in underwater wireless sensor networks: a survey. *Journal of Ambient Intelligence and Humanized Computing*, 12(10):9757–9773, 2021.

[10] Lars Michael Kristensen and Roald Otnes. An internal technical document of the sfi smart ocean project. Technical report, SFI Smart Ocean Project, 2022. Internal Technical Document.

[11] Yeonhee Lee and Youngseok Lee. Toward scalable internet traffic measurement and analysis with hadoop. *ACM SIGCOMM Computer Communication Review*, 43(1):5–13, 2012.

[12] Keila Lima, Ngoc-Thanh Nguyen, Rogardt Heldal, Eric Knauss, Tosin Daniel Oyetoyan, Patrizio Pelliccione, and Lars Michael Kristensen. Marine Data Sharing: Challenges, Technology Drivers and Quality Attributes. In *International Conference on Product-Focused Software Process Improvement*, pages 124–140. Springer, 2022.

[13] Keila Lima, Ngoc-Thanh Nguyen, Rogardt Heldal, Lars Michael Kristensen, Tosin Daniel Oyetoyan, Patrizio Pelliccione, and Eric Knauss. A data-flow oriented software architecture for heterogeneous marine data streams. In *2024 IEEE 21st International Conference on Software Architecture (ICSA)*, pages 146–157. IEEE, 2024.

[14] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

[15] Suzanne J Matthews and Aaron St Leger. Leveraging mapreduce and synchrophasors for real-time anomaly detection in the smart grid. *IEEE Transactions on Emerging Topics in Computing*, 7(3):392–403, 2017.

[16] Ngoc-Thanh Nguyen, Rogardt Heldal, Keila Lima, Tosin Daniel Oyetoyan, Patrizio Pelliccione, Lars Michael Kristensen, Kjetil Waldeland Høydal, Pål Asle Reiersgaard, and Yngve Kvinnsland. Engineering challenges of stationary wireless smart ocean observation systems. *IEEE Internet of Things Journal*, 10(16):14712–14724, 2023.

[17] Ngoc-Thanh Nguyen, Rogardt Heldal, and Patrizio Pelliccione. Concept-drift-adaptive anomaly detector for marine sensor data streams. *Internet of Things*, page 101414, 2024.

[18] Ngoc-Thanh Nguyen, Keila Lima, Astrid Marie Skålvik, Rogardt Heldal, Eric Knauss, Tosin Daniel Oyetoyan, Patrizio Pelliccione, and Camilla Sætre. Synthesized data quality requirements and roadmap for improving reusability of in-situ marine data. In *2023 IEEE 31st International Requirements Engineering Conference (RE)*, pages 65–76. IEEE, 2023.

[19] Oecd. *The ocean economy in 2030*. OECD, 2016.

[20] Judith Thu Ølberg, Patrik Bohlinger, Øyvind Breivik, Kai H Christensen, Birgitte R Furevik, Lars R Hole, Gaute Hope, Atle Jensen, Fabian Knoblauch, Ngoc-Thanh Nguyen, et al. Wave measurements using open source ship mounted ultrasonic altimeter and motion correction system during the one ocean circumnavigation. *Ocean Engineering*, 292:116586, 2024.

[21] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

[22] Aungon Nag Radon, Ke Wang, Uwe Glässer, Hans Wehn, and Andrew Westwell-Roper. Contextual verification for false alarm reduction in maritime anomaly detection. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 1123–1133. IEEE, 2015.

[23] Claudio Reggiani, Yann-Aël Le Borgne, and Gianluca Bontempi. Feature selection in high-dimensional dataset using mapreduce. In *Benelux Conference on Artificial Intelligence*, pages 101–115. Springer, 2017.

[24] Martha Rodríguez, Diana P Tobón, and Danny Múnera. Anomaly classification in industrial internet of things: A review. *Intelligent Systems with Applications*, page 200232, 2023.

[25] Camilla Saetre, Astrid Marie Skålvik, Kjell-Eivind Frøysa, and Marie Bueie Holstad. A smart ocean observation system for reliable real-time measurements. In *OCEANS 2023-Limerick*, pages 1–5. IEEE, 2023.

[26] T Saranya, C Deisy, S Sridevi, and Kalaiarasi Sonai Muthu Anbananthen. A comparative study of deep learning and internet of things for precision agriculture. *Engineering Applications of Artificial Intelligence*, 122:106034, 2023.

[27] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797, 2022.

[28] Shahab Shamshirband, Mahdis Fathi, Abdollah Dehzangi, Anthony Theodore Chronopoulos, and Hamid Alinejad-Rokny. A review on deep learning approaches in healthcare systems: Taxonomies, challenges, and open issues. *Journal of Biomedical Informatics*, 113:103627, 2021.

[29] Hasan Tercan and Tobias Meisen. Machine learning and deep learning based predictive quality in manufacturing: a systematic review. *Journal of Intelligent Manufacturing*, 33(7):1879–1905, 2022.

[30] TutorialsPoint. Hadoop - tutorial. `https://www.tutorialspoint.com/hadoop/index.htm`, 2024. Accessed: 2024-11-03.

[31] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *Acm computing surveys (csur)*, 53(2):1–33, 2020.

[32] Shuangyuan Wang, Yixiang Huang, Lin Li, and Chengliang Liu. Wind turbines abnormality detection through analysis of wind farm power curves. *Measurement*, 93:178–188, 2016.

AI-powered embedded software for real-time marine data quality control

The ocean is vital for human beings as it provides food and energy, regulates climate, or enables transportation. An unhealthy ocean can have severe consequences for both the climate and humankind. In 2019, the United Nations warned that ocean health had been declining and called for efforts to reverse it. To foster sustainable ocean utilization, obtaining accurate and dependable data regarding the ocean's state is of importance. A comprehension of the ocean can improve awareness and motivate efforts towards monitoring and conservation, benefiting both society and companies involved in marine activities.

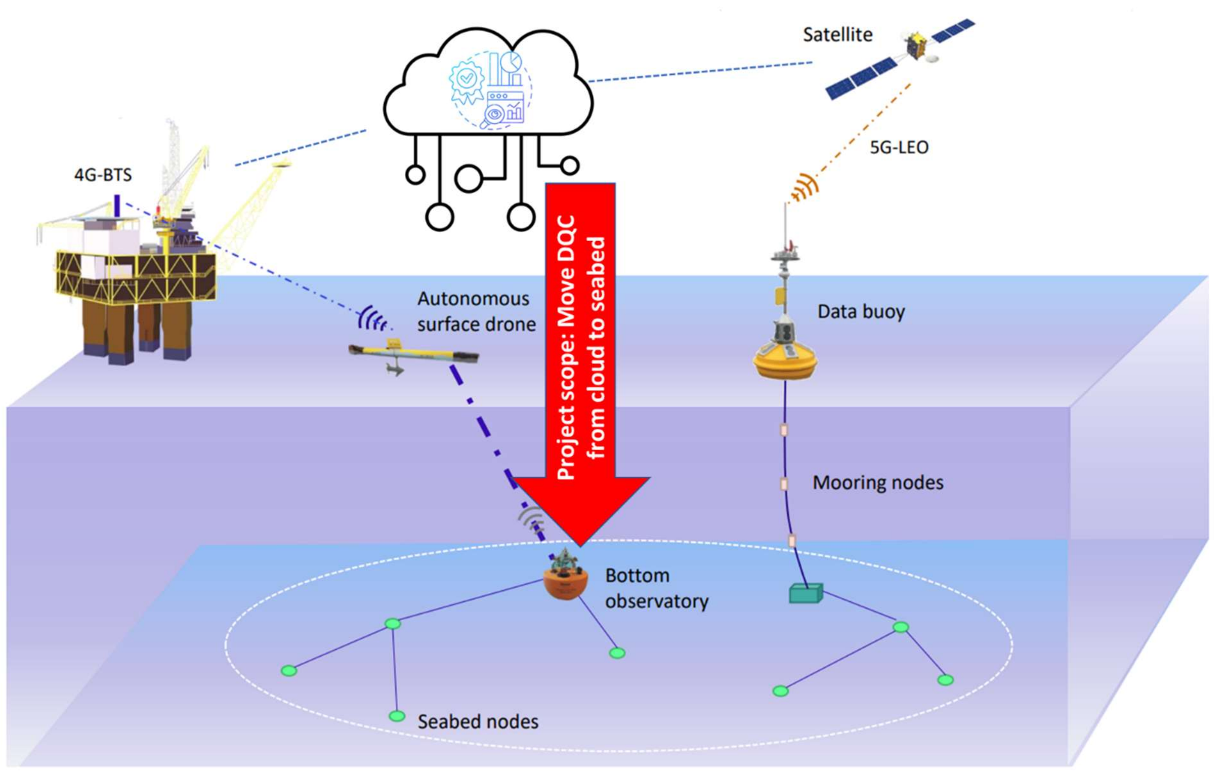
Marine data is prone to error due to several factors. Underwater sensors may produce a lot of anomalies when powered by low-batteries. Biofouling is an accumulation of microorganisms, plants, algae, or small animals on underwater sensors, resulting in unexpected measurement results. If low-quality data is used in decision-making processes, the results will be misleading or suboptimal; thus, Data Quality Control (DQC) must be done to ensure data reliability.

Today's underwater sensory technologies allow us to collect vast amounts of data. For example, Copernicus (https://www.copernicus.eu/en),a one of the world's largest marine data providers, delivers 16 terabytes per day, and the first 13 months of the One Ocean Expedition (https://oneoceanexpedition.com/) resulted in 8.6 terabytes. Despite some initiatives and guidelines for automatic DQC, assessing data quality of data is still mainly done by domain experts in practice. Data publication typically is delayed from 6 months to a year due to DQC. Such delays undermine the validity of decision-making results. In many cases, expensive collected data must be abandoned due to labour shortage.

There is pressing need from industry for software solutions to automate data quality control activities and Artificial Intelligence (AI) appears to be a promising solution. Our research group has developed an effective anomaly detection algorithm using deep learning based on Python for that purpose. However, we have to deal with another emerging challenge that it is difficult and costly to transmit data from underwater to the shore; hence, DQC must be done at source. In this case, the algorithm must be embedded into underwater sensors, which is a crucial step to make the whole ocean smart. The figure below shows how DQC is done currently and expected by this project.

The project does not need to invent a new algorithm as it is readily available; however, different kinds of knowledge are required, including embedded programming, understanding of hardware, knowledge of machine learning, and code and resource optimization. In other words, the project will require and cover knowledge of the whole computing system.

The project is conducted in collaboration with an underwater sensor manufacturing company based in Bergen; therefore, some visits to the company may be required. Evaluation/development hardware and necessary expertise will be provided by the company and also supported by the supervising team. The data is collected from different sites and is readily available for study.



Supervisors:

- Rogardt Heldal, professor of Software Engineering.
- Ngoc Thanh Nguyen, PhD candidate in Software Engineering.
- Jan Flatlandsmo, Senior Software Engineer, Aanderaa.