HEALER: A Data Lake Architecture for Healthcare

Carlo Manco[1], Tommaso Dolci[1,*], Fabio Azzalini[1], Enrico Barbierato[2], Marco Gribaudo[1] and Letizia Tanca[1]

[1]Dep. of Electronics, Information and Bionengineering, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano, Italy
[2]Dep. of Mathematics and Physics, Università Cattolica del Sacro Cuore, Via della Garzetta 48, 25133 Brescia, Italy

Abstract

With the growth of the Internet of Things and the rapid progress of social networks, everything appears to generate data. The ever-increasing number of connected devices is accompanied by a growth of the volume of data, produced at an ever-increasing rate, and this massive flow includes data types that are difficult to process using standard database techniques. One of the most critical scenarios is healthcare, whose activities need to store and manage a variety of data types – reports written in natural language, medical images, genomic data and waveforms of vital signs – which do not have a well-defined structure. In order to benefit from this large amount of complex data, Data Lakes have recently emerged as a solution to grant central storage and flexible analysis for all types of data. However, there is no Data Lake architecture that fits all the possible scenarios, since the architecture depends heavily on the application domain and, so far, there are no Data Lake architectures that support the specific needs of the healthcare domain. This work proposes HEALER: a Data Lake architecture that effectively performs data ingestion, data storage, and data access with the aim of providing a single central repository for efficient storage of different types of healthcare data. The architecture also enables the analysis and querying of the data, which can be loaded into the Data Lake regardless of their format and type. To verify the effectiveness of the architecture, a proof-of-concept of HEALER has been developed, that allows ingestion of various data, performs waveforms processing to make them more interpretable to researchers and analysts, grants access to the saved data and allows the analysis of natural language reports. Finally we studied the performance of the system in each of its main phases: ingestion, processing, data access and analysis. The results lead us to some important considerations to be taken into account when using and configuring the system components.

Keywords

Data Lakes, medical data, waveforms, Hadoop Distributed File System, Apache NiFi

1. Introduction

With the evolution and growing popularity of social media platforms and, in particular, with the birth of the Internet of Things (IoT), the data traffic has now reached the zettabyte threshold. For an increasing number of organizations, it is therefore critical to ingest, process and save the most important aspects of the generated data while removing the unnecessary parts.

One of the areas that contribute significantly to the generation of this data is healthcare. In fact, for the progress of medical research, and healthcare in general, it is of paramount importance that the data collected from patients are saved for immediate and future batch analyses, facilitating the identification of diseases allowing for precision treatments [1]. Valuable data about patients and medications are usually digitized and saved as Electronic Health Records (EHR). EHR on a large scale enable researchers to identify opportunities to move healthcare organizations toward personalized healthcare, that consists in using diagnostic tests to determine which medical treatments will work best for each patient [2]. Typically, only a portion of EHR contain data in a structured format that data scientists can easily use, while most of them consist in unstructured and semi-structured data (i.e., with a form of structure that does not follow the classical tabular one of relational models). It is therefore necessary to develop an efficient data administration pipeline to assist scientists in their efforts to provide customized prescriptions. An answer to this problem is represented by Data Lakes, that, according to [3] are a current and increasingly emerging trend for Big Data storage and management. However, despite the presence of well-designed examples of Data Lake architectures in the literature [4, 5], none of them has been specifically designed to meet the requirements of the healthcare scenario.

The goal of this work is to implement a proof-of-concept of a Data Lake for the healthcare scenario, that can ingest and process data from different types of sources (medical devices, clinical trial datasets etc.). Data must be saved in their raw format inside the Data Lake

DataPlat'23: 2nd International Workshop on Data Platform Design, Management, and Optimization, March 28, 2023, Ioannina, Greece
*Corresponding author.
carlo.manco@mail.polimi.it (C. Manco); tommaso.dolci@polimi.it (T. Dolci); fabio.azzalini@polimi.it (F. Azzalini); enrico.barbierato@unicatt.it (E. Barbierato); marco.gribaudo@polimi.it (M. Gribaudo); letizia.tanca@polimi.it (L. Tanca)
0000-0002-1403-7766 (T. Dolci); 0000-0003-0631-2120 (F. Azzalini); 0000-0003-1466-0248 (E. Barbierato); 0000-0002-1415-5287 (M. Gribaudo); 0000-0003-2607-3171 (L. Tanca)

storage area, and subsequently be processed and transformed in an easily usable way by scientists and analysts. The resulting system, named HEALER (HEalthcare dAta LakE aRchitecture), focuses on allowing analyses using both machine learning algorithms and conventional queries. Since healthcare data are very heterogeneous, for the moment we decided to limit our analysis to waveforms of various signals extracted from patients and to medical reports written in natural language, in addition to structured data management. These types of data, although being a limited part of the ones usually considered in healthcare, allow to characterize a complete workflow and thus to exemplify the proposed architecture.

The contributions of this work include: *a)* the design and definition of HEALER, a Data Lake architecture that can effectively manage both structured and unstructured medical data; *b)* the implementation of a proof-of-concept of HEALER on the basis of the proposed architecture, focusing on the management of waveforms and natural language reports; *c)* testing and performance analysis of the various components of the system.

2. Related Work

Multiple Data Lake architectures have been proposed by the literature, but most of them do not qualify as comprehensive solutions for the described scenario. The Data Lake is defined in both the scientific and industrial worlds as a repository storing raw data in their native format; however, different definitions have different emphases, for example in [6] governance and metadata management are highlighted, while [7] places importance on the users, presenting an architecture focused on researchers and analysts. Furthermore, the design of a Data Lake is strongly influenced by the kind of scenario in which it is placed.

Many Data Lakes designed for non-healthcare scenarios present a robust architecture, but disregard certain requirements that are essential for healthcare. For instance, CoreDB [8] is an open-source Data Lake to organize, index and query data and metadata, but does not guarantee the storage of data in their native format. Hydria [9] is a Data Lake for cultural heritage data: it provides an integrated framework that enables easy deployment of data acquisition services, dataset sharing with other stakeholders, search, filtering and analysis of data via visualization tools. Constance [10] manages structural and semantic metadata, but scenario-specific features must be included in order for it to be used in real-world applications. ArchaeoDAL [5] has been developed using a very effective multi-layer approach; however, it is designed to mainly handle relational databases. FEDDL (Flexible Energy Denmark Data Lake) [4] is a Data Lake with the purpose of collecting and sharing energy-related data in Denmark, with integrated AI and machine learning tools to analyze them. FEDDL proposes a solid architecture and provides a technical overview of the whole structure. In fact, their findings played an important role for the definition of our architecture.

In the healthcare field, many cloud-based solutions offered by cloud providers such as AWS (Amazon Web Services) and Azure are emerging [11] and some Data Lake have been implemented leveraging their services. A medical Data Lake is proposed in [12], whose authors suggest a design that relies on cloud services. Similarly, [13] proposes an IoT-Cloud–based framework for real-time processing of Big Data in the healthcare domain. It is developed implementing AWS. Also [14] describes a cloud architecture that make use of Azure functionalities. Here, huge amounts of data are efficiently stored using Azure Data Lake, to support physicians in detecting heart diseases.

However, the main challenge of cloud technology applications in the healthcare sector is the ownership of sensitive data. Proper security measures must be implemented to protect data at each level of data management. Despite cloud providers offering various security services, it is not always clear how to integrate them with the proposed architecture. Moreover, working on proprietary and not on open source software greatly limits the customization of the solutions, and in addition cloud providers usually require monetary payments based on software usage and the hardware made available.

Recently, despite the absence of a full-fledged Data Lake for healthcare, researchers are working towards this direction, for instance by developing query engine components for clinical data [15, 16]

3. Goals and Requirements

In order to retrieve valuable insights and helpful knowledge easily from data, healthcare researchers require a unified system [17] that allows to:

- manipulate, process, and analyze different types of data, from fully structured to unstructured, allowing their storage inside the system in raw native format;
- handle different types of data effectively, with open possibilities for integrating information from IoT devices, with the system serving as a centralized location for all data collected by various injector systems;
- provide a distributed file system where data can be stored in files and directories to allow for efficient data loading, while guaranteeing high availability and fault tolerance without impacting application performance;

- meet clinical requirements by allowing fast and efficient analysis of new combinations of data;
- give the rights to design how to access the data: access must be regulated for protection of personal data and privacy.

As a consequence, a Data Lake is the most appropriate framework to adequately manage all these factors. In fact, Data Lakes can provide a unified platform for all relevant data generated by healthcare systems, operating as a repository for both structured data collected from traditional databases and unstructured data derived from various other sources. Data Lakes are extremely fast and versatile because they implement a scalable architecture to store data in their native form, while remaining easily accessible and centralized for end users. Furthermore, Data Lakes can be fully equipped with various security layers to ensure data integrity and compliance with privacy and regulations, which is particularly important in the healthcare context.

This work focuses primarily on the definition of the physical components of a data lake architecture, and conceptual aspects such as security and privacy are left for future work since their components are outside the main data management pipeline [18].

3.1. Multi-layered Architecture

In our case study, we want to define, as a first step, a functional implementation similar to [19], which proposes a multi-layered approach based on incorporating layers with separation of concerns. In fact, this type of architecture has the advantage of clearly highlighting the functions to implement for a given Data Lake, which allows for an easy matching to the corresponding required technologies: the key concept is that each layer communicates with the adjoined ones, and the data follow a pipeline over all the layers. Figure 1 illustrates the specific layers considered in the proposed architecture.

In particular, starting from the top of the diagram, the **Data Ingestion** layer has the task of ingesting heterogeneous data in raw format from various data sources and into the Data Lake. The two main options to load data into a Data Lake are *batch* and *streaming*. The choice of implementing a batch-oriented or stream-oriented data ingestion layer depends very much on the context to which the Data Lake is applied: in fact, as the context changes, we may have more or fewer data sources providing streaming data.

The **Data Storage** layer is the core of the Data Lake. It contains raw-data repositories, but also transformed data; it provides support for different forms and structures of the data, including file storage and raw-record storage.

The **Data Transformation** layer provides the potential for the scalable execution of operations such as data cleaning and data transformation, in order to achieve a predefined final form. In this layer, different data representations are created in order to transform raw data into easily accessible data formats, based on algorithms' and users' needs.

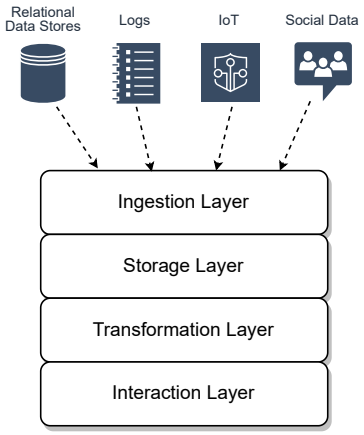

Figure 1: Overview of the multi-layered Data Lake architecture described in [19].

Finally, the **Data Interaction** layer provides end-users with access to the data created in the transformation layer. Users can access data in order to perform exploration tasks, create and apply analytical queries and visualize the stored data using various visualization tools.

4. HEALER's Architecture and Implementation

4.1. Overview of the Architecture

Data Lake architectures are usually classified into Data Pond Architectures and Zone Architectures [20, 21]. In the healthcare domain, as suggested in [22], a Zone Architecture is preferred. In fact, data should go through different levels of refinement while maintaining a copy of the data in their raw format, characteristics that Pond Architecture does not provide [23, 20]. These functionalities are essential for healthcare researchers to allow them and analysts to perform more than a single transformation and analysis on the same data. More in detail, the proposed architecture is composed of five different zones: Transient Landing Zone, Raw Zone, Process Zone, Refined Zone and Consumption Zone (Figure 2).

Figure 3 shows a high-level view of the proposed system, illustrating the main layers of the architecture, where each layer is represented by its corresponding component. Data are first ingested by the Ingestion component in the Transient Landing Zone, and then per-

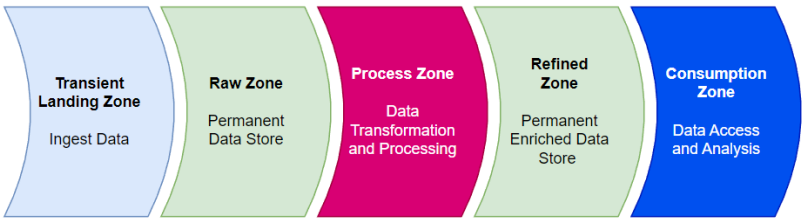

Figure 2: Overview of the adopted zones.

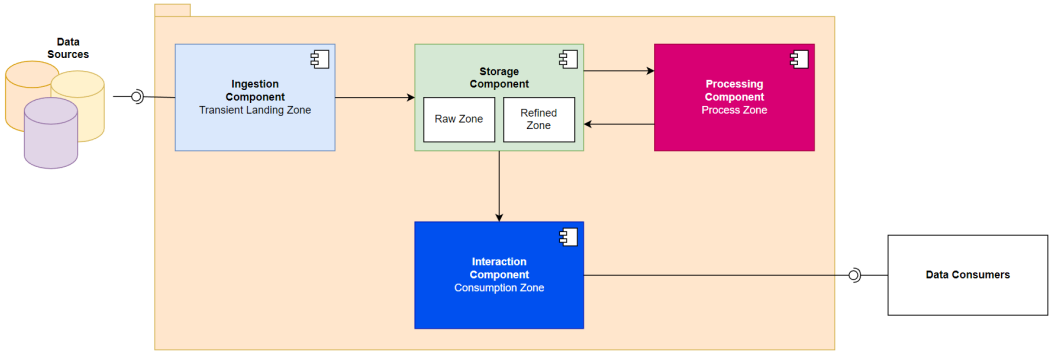

Figure 3: Implemented components of the proposed Data Lake architecture.

manently saved in the Raw Zone by the Data Storage Component. Then, they are processed inside the Process Zone in different ways according to their format. Moreover, the output of the transformation performed by the Processing Component, known as "refined data", is stored in the Refined Zone. The Interaction Component is the module that deals with data analysis and querying; it has free access to both the Raw and Refined zones of the Data Storage, and produces data in the Consumption Zone.

From Figure 3 it can also be observed that each component is associated with at least one zone. In particular, the **Data Ingestion Component** represents the Transient Landing Zone: batch and streaming data land here after being extracted from the data sources and pushed into the Data Lake. Data are transferred without any transformations, in accordance with the first steps in the Extract-Load-Transform (ELT) paradigm. Vital signs from hospitalized patients, reports written in natural language, and structured data on patients and lab events are all ingested regardless of their format. In this work, we focused on a batch-type data ingestion.

The **Storage Component** represents the central repository where large volumes of medical data are stored. This module must provide a distributed file system while guaranteeing high availability and fault tolerance. In particular, it contains two zones, namely the Raw Zone, i.e., the area where the data are permanently stored in their native raw format, and the Refined Zone, where data are stored in a form suitable for the end-users and where processed data are saved. In order for scalability to be guaranteed, this component should be able to increase its storage capacity when needed.

The **Processing Component** represents the Process Zone. This is where data are brought to be prepared and processed. In our case, we manage time series data before saving them in the Refined Zone, exposing them to a transformation process that makes them more easily interpretable by the final user.

Finally, the **Interaction Component** contains the Consumption Zone that is in charge of granting access to the data in the Refined and in the Raw Zone. Here users can perform queries on structured data by directly using Python libraries, or apply various data analysis techniques. For instance, reports written in natural language can be subject to various language-understanding tasks. Results obtained from this component can be returned to the Raw Zone for later use. Saving results is very important to allow further analysis: for example, keywords extracted from reports can later be mined to find patterns or apply clustering techniques.

4.2. Technologies

Many data management tools can be implemented inside a Data Lake [24]. In HEALER, two are the key tools: a file system responsible for managing data collection in a distributed storage, and a tool for performing ELT-type data ingestion that can easily connect to various data sources and the distributed storage.

In particular, to fulfill the file system requirements expressed in Section 3, we choose HDFS (Hadoop Distributed File System) [25]. In fact, HDFS provides high-performance access to data across highly scalable Hadoop clusters, and it can handle big volumes of both structured and unstructured data. Additionally, it is possible to specify the number of copies of a file that should be maintained by HDFS, guaranteeing the replication of data. In each cluster there is a single main node, called Namenode, and a variable number of secondary nodes which are called Datanode.

Secondly, we selected Apache NiFi [26] as tool for data ingestion. NiFi has many advantages compared to its competitors [27]. Among them, it provides both batch and real-time data transfer, it can easily interact with the Hadoop ecosystem and particularly with HDFS, it supports the most widely used communication protocols, and it scales linearly to many nodes. Additionally, every node of NiFi provides the same functionality as every other node: this makes the system very robust to node failures.

4.3. Data Flow

The first stage in the data flow is the ingestion into the Transient Landing Zone. This is where raw data are extracted from the sources to start their path inside HEALER. This phase ends when the raw data is placed within the storage area provided by HDFS. The entire process is handled by Apache NiFi. The flow diagram in Figure 4 shows the two NiFi processors used to manage the ingestion phase: the GetFile processor fetches data from the data source and then hands them to the PutHDFS processor; PutHDFS establishes the connection to HDFS and inserts data into the distributed storage. Once distributed memory is reached, data are divided into blocks and replicated in different Datanodes for a number of times equal to the replication factor, in our case equal to three. HDFS ensures that this operation is performed automatically to increase data availability and make the system more robust to individual Datanode failures.

At this point, the raw data saved within the distributed storage are ready to be processed. This task is performed inside the Process Zone, and the processing procedure depends on the data format and type; in this context, we discuss the processing of time series extracted from

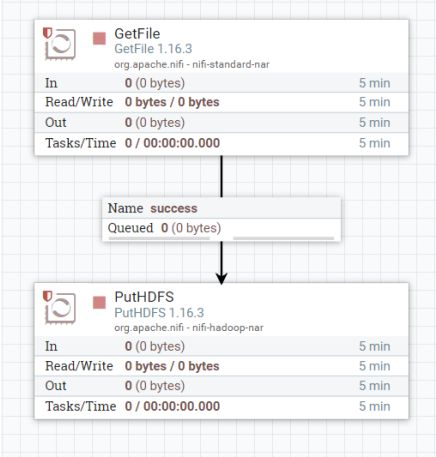

Figure 4: Data flow in Apache NiFi.

MIMIC-III [28], a medical dataset that contains both structured and unstructured data, such as information on patients' vital signs in waveforms and discharge notes in natural language. In particular, we define a processing tasks for waveforms, meant to transform them into a more easily interpretable format. This task involves the addition of information - such as the identifier of the patient and the starting date and time of the measures - to the file containing the signal measurements. Details on time series processing are presented in Appendix A. The process takes advantage of the *hdfs* Python library[1], used to handle communication with the storage, not only when the processor node needs to take data to be processed, but also when there is the need to rewrite data into the Refined Zone after processing. For what concerns the execution of data requests in the system, the following two scenarios were studied.

Scenario 1 In this scenario, the data requested by the user have already been processed and therefore they are already present in the Refined Zone. In this case, the request is immediately addressed and the data are directly retrieved from the Refined Zones.

Scenario 2 This scenario involves real-time processing of the data requested by the user, due to the fact they have not been processed yet. In this case, the data are searched in the Refined Zones; however, some, or all of the data are not available in the Refined Zone and thus the request must be forwarded to the Raw Zone. If the requested data are found in the Raw Zone, they are sent to the Process Zone, ready to be processed in real-time. After the processing, following the normal flow of data,

[1] https://pypi.org/project/hdfs

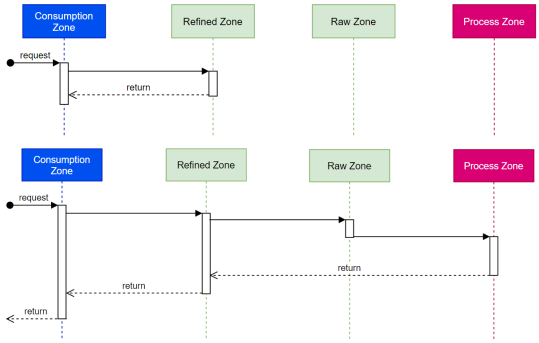

Figure 5: Two sequence diagrams illustrating user requests to access data. On the top diagram, data are already available in the Refined Zone. On the bottom diagram, data are not yet stored in the Refined Zone, thus need be pre-processed.

they are saved into the Refined Zone, which is finally ready to meet the user request. This is shown in Figure 5.

Concerning natural language data, the analysis involves the extraction of keywords that best identify the key concepts of the text. To perform this kind of text processing, two different Python libraries for natural language processing are compared: Spacy [29] and KeyBERT [30]. This work will use the reports in MIMIC's *NoteEvents* table as input dataset for analysis.

4.4. Implementation

We developed a proof-of-concept of HEALER, implementing the considered nodes on several container based virtual machines, running on a single physical machine using a quad core CPU Intel Core i7-4790S 3.2 GHz, 16 GB RAM, GPU NVIDIA GeForce GTX 745, and 2 TB Hybrid HDD. Despite the hardware limitations, the system was perfectly suited for our tests. The technical overview of the system is shown in Figure 6. The system was tested to ensure the architecture's effectiveness and also to evaluate the components' compatibility and communication. Listed below are the specific nodes implemented:

- a single ingestion node implementing Apache NiFi and representing the Transient Landing Zone;
- a single node implementing an HDFS Namenode for managing data storage;
- a cluster of five nodes, each implementing an HDFS Datanode instance, representing the Storage Component and incorporating both Raw Zone and Refined Zone;
- a single process node to execute waveform processing (Process Zone);
- a single analysis node, with access to the GPU, equipped with useful libraries to perform analysis and to access data (Consumption Zone).

We adopted Docker[2] to virtualize the nodes, and configured the system with the required tools. The Hadoop cluster can extend or reduce the number of Datanodes, by leveraging the potential of an orchestration tool such as Docker Compose. The considered setup can be easily migrated to run across multiple virtual or physical machines, using for example Kubernetes[3].

Additionally, the following requirements are essential for the proper execution of the system: *(a)* the nodes must be instantiated within the same network; *(b)* the Namenode and Datanodes communicate with each other properly: the Namenode must hold information about the IP addresses of the Datanodes in the network, and each Datanode must be familiar with the IP address of the Namenode and fellow Datanodes; *(c)* the ingestion node should have all the information needed to access the Storage Component (IP address, port number etc.), and the Hadoop namespace configuration files contained in the Namenode must be copied to the Apache NiFi node; *(d)* the processing node and the analysis node implements the *hdfs* library to establish and manage communication towards the distributed storage.

5. Evaluation and Results

5.1. Dataset

The dataset used for the evaluation is the MIMIC-III Database [28]. MIMIC stands for Medical Information Mart for Intensive Care, and is a large, freely-available database comprising de-identified health-related data associated with over forty thousand patients who stayed in critical care units.

It includes a structured clinical database and a related waveform database [31]. The clinical database includes 26 structured tables, with one of them (named *NoteEvents*) including medical reports written in natural language. In the waveform database, records typically contain ten or more time series of patients' vital signs sampled once per second or once per minute. Each of this records is composed of a header file, which shows information about the measurements (number of samples, start time of the measurement session, description about the observed signals) and a matching signal file.

5.2. Evaluation

This section presents an evaluation of the components of HEALER, focused on the execution of the main stages.

[2]https://www.docker.com
[3]https://kubernetes.io

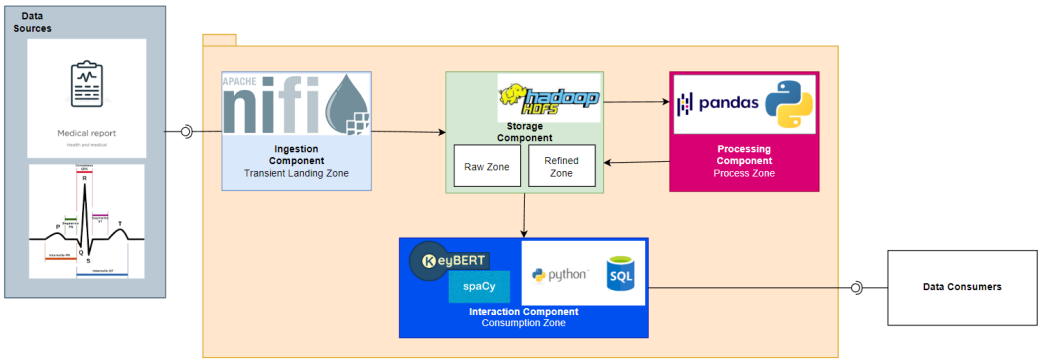

Figure 6: Technical overview of the system: for each component of the architecture, the corresponding technology/software is specified.

Table 1
Execution time and transfer rate for the ingestion phase.

Batch (MB)	GetFile (s)	PutHDFS (s)	Full Time (s)	Transfer Rate (MB/s)
11	0.5	4.5	5.0	**2.20**
156	5.0	33.0	38.0	**4.11**
305	8.0	44.0	52.0	**5.87**
1024	35.0	75.0	110.0	**9.31**
1741	52.0	106.0	158.0	**11.02**
3738	86.0	194.0	280.0	**13.35**

The ingestion phase transfers data to the HDFS cluster using Apache NiFi. In particular, we evaluate the ingestion time of structured data (from MIMIC III clinical database) and waveforms (from MIMIC III waveform database). The experiment begins when the flowchart described in Figure 4 is executed, and it ends when all the files saved in the Apache NiFi node are ingested into the system. Data of different batch sizes are transferred, repeating measurements 20 times for each batch size. Table 1 shows that the average transfer rate goes from 2.20 MB/s for a batch size of 11 MB, to 13.35 MB/s for 3.7 GB. Thus we infer that, as the batch size increases, the transfer rate also becomes higher.

The processing node in our simulation is in charge of running a Python script that takes a batch of waveforms in order to perform the transformation process. HEALER makes use of the *pandas* library to perform processing on waveforms and the *hdfs* library to handle communication with the Storage Component. Our evaluation in this case involves measuring the time taken by the node to fetch batches of waveforms from storage, transform them, and finally save them into the Refined Zone. We perform the processing of different batch sizes, obtaining the results described in Table 2. Evidently, the system always perform with an average rate of 1 MB/s, regardless of the data batch size. Therefore, we reach the conclusion that the batch size of has minimal effect on the processing rate.

Table 2
Execution time and transfer rate of the processing phase.

Batch (MB)	Process Time (s)	Rate (MB/s)
9	9.0	**1.00**
127	120.0	**1.06**
305	290.0	**1.05**
1024	1034.0	**0.99**

Concerning the access phase, the main objective is to allow users to access data from the Storage Component. Our simulation involves the evaluation of both scenarios described in Section 4.3, i.e., we evaluate the amount of time that the analysis node takes when data is already in the Refined Zone, and when it is not. We consider data of different sizes and assess the efficiency of the system based on response time. Table 3 clearly shows that requests of data not yet processed are much more time-consuming than those that have already been processed and stored in the Refined Zone. In fact, the data access phase is very dependent on the speed of real-time

Table 3
Comparison between access times of data already available in the Refined Zone and yet to be processed in the Raw Zone.

Batch (MB)	Refined Zone (s)	Raw Zone (s)
0.1	0.030	0.095
1	0.094	1.213
11	0.467	10.789
53	1.330	51.342
108	3.713	107.432

processing: since the processing phase is the slowest step inside the pipeline, accessing data in the Raw Zone is also very slow.

Finally, we evaluated the extraction of keywords from natural language reports. In this experiment, the goal is to compare two different Python libraries: Spacy [29] and KeyBERT [30]. The main difference between the two libraries is that KeyBERT leverages the GPU, while Spacy does not. In our proof-of-concept, it is the analysis node that performs the extraction operation from the reports. The extraction performance of the two libraries is analyzed on different samples of reports extracted from MIMIC-III *NoteEvents* table. We perform the tasks first on a set of 20 reports, then on a set of 200, using both libraries. Each experiment is repeated ten times. Spacy takes, on average, 16s to analyze 20 reports and 157s for 200 reports, while KeyBERT takes 27s to analyze 20 reports and 233s for 200 reports. With KeyBERT, GPU utilization peaked at over 50%, therefore, with a higher performing unit, it might be advantageous to use this library. On the contrary, if the system lacks a GPU, Spacy will be more efficient in terms of execution time.

5.3. Discussion

By analyzing the results and observations derived from the simulation, we observe how Apache NiFi and HDFS work very well with large batches of data. On the other hand, it is not recommended to use small batch sizes due to the risk of lower transfer rates. Moreover, regarding HDFS, due to its approach of dividing data into fixed-size blocks, it is suggested to not ingest a large amount of data with a much smaller size than the block size. This is to avoid flooding the Datanodes with unfilled blocks.

Lastly, for what concerns the access phase, it is appropriate to process larger data asynchronously with respect to user requests. Instead, real-time processing should be left for accessing data of smaller size.

6. Conclusions and Future Work

In this work we proposed HEALER, an architecture for a Data Lake that allows for efficient data storage and access to a variety of types of data in the context of healthcare. HEALER allows for analysis and query, and can manage data regardless of their generation speed or volume. In HEALER, the main emphasis is placed on the storage of non-structured data, such as medical waveforms and natural language reports, which are seldom considered in traditional systems.

Moreover, we implemented a proof-of-concept of HEALER, composed by a Data Ingestion Component, a Storage Component, a Processing Component and a Data Access and Interaction Component. We tested the system for a general evaluation of its various stages: ingestion, processing, data access, data analysis. From our experiments, we discovered that both HDFS and Apache NiFi perform well at high batch sizes. In addition, the study showed how the processing phase represents the bottleneck of HEALER and how it also impacts the data access phase in responding to user requests.

For future work, the plan is to expand the system by introducing horizontal layers, such as a layer for Data Governance, to manage metadata and enable the system to be more organized and faster when accessing data. Furthermore, an additional layer to manage user access and ensure data security is also planned, considering the extreme importance of securing the large amount of personal information present in medical data.

Last but non least, in a more realistic context, it is appropriate to test HEALER with a streaming-type ingestion of data, and to expand the Process Zone to a cluster of nodes performing data transformations at a distributed level. To perform distributed operations, we will use an appropriate tool, such as Apache Spark [32].

Finally, we plan to perform workload tests on HEALER with both real-world and simulated data, to obtain more realistic results on response time in different workload situations.

Acknowledgments

This work has been partially supported by the Health Big Data Project (CCR-2018-23669122), funded by the Italian Ministry of Economy and Finance and coordinated by the Italian Ministry of Health and the network Alleanza Contro il Cancro.

Additionally, we are grateful to Giuseppe Serazzi for his advice during the definition of this work and the support in the revision of this paper.

References

[1] V. Jagadeeswari, V. Subramaniyaswamy, R. Logesh, V. Vijayakumar, A study on medical internet of things and big data in personalized healthcare sys-

tem, Health information science and systems 6 (2018) 1–20.
[2] N. V. Chawla, D. A. Davis, Bringing big data to personalized healthcare: a patient-centered framework, Journal of general internal medicine 28 (2013) 660–665.
[3] P. P. Khine, Z. S. Wang, Data lake: a new ideology in big data era, in: ITM web of conferences, volume 17, EDP Sciences, 2018, p. 03025.
[4] H. B. Hamadou, T. B. Pedersen, C. Thomsen, The danish national energy data lake: Requirements, technical architecture, and tool selection, in: 2020 IEEE International Conference on Big Data, IEEE, 2020, pp. 1523–1532.
[5] P. Liu, S. Loudcher, J. Darmont, C. Noûs, Archaeodal: A data lake for archaeological data management and analytics, in: 25th International Database Engineering & Applications Symposium, 2021, pp. 252–262.
[6] C. Walker, H. Alrehamy, Personal data lake with data gravity pull, in: 2015 IEEE Fifth International Conference on Big Data and Cloud Computing, IEEE, 2015, pp. 160–167.
[7] C. Madera, A. Laurent, The next information architecture evolution: the data lake wave, in: Proceedings of the 8th international conference on management of digital ecosystems, 2016, pp. 174–180.
[8] A. Beheshti, B. Benatallah, R. Nouri, V. M. Chhieng, H. Xiong, X. Zhao, Coredb: a data lake service, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, 2017, pp. 2451–2454.
[9] K. Deligiannis, P. Raftopoulou, C. Tryfonopoulos, N. Platis, C. Vassilakis, Hydria: An online data lake for multi-faceted analytics in the cultural heritage domain, Big Data and Cognitive Computing 4 (2020) 7.
[10] R. Hai, S. Geisler, C. Quix, Constance: An intelligent data lake system, in: Proceedings of the 2016 international conference on management of data, 2016, pp. 2097–2100.
[11] T. S. Hukkeri, V. Kanoria, J. Shetty, A study of enterprise data lake solutions, International Research Journal of Engineering and Technology (IRJET) Volume 7 (2020).
[12] M. Sha M, M. P. Rahamathulla, Cloud-based healthcare data management framework, KSII Transactions on Internet and Information Systems (TIIS) 14 (2020) 1014–1025.
[13] N. C. Taher, I. Mallat, N. Agoulmine, N. El-Mawass, An iot-cloud based solution for real-time and batch processing of big data: Application in healthcare, in: 2019 3rd international conference on bioengineering for smart technologies (BioSMART), IEEE, 2019, pp. 1–8.
[14] E. Maini, B. Venkateswarlu, A. Gupta, Data lake-an optimum solution for storage andanalytics of big data in cardiovascular disease prediction system, International Journal of Computational Engineering & Management (IJCEM) 21 (2018) 33–39.
[15] T. Heinis, A. Ailamaki, Data infrastructure for medical research, Now Publishers, 2017.
[16] G. M. Weber, S. N. Murphy, A. J. McMurry, D. MacFadden, D. J. Nigrin, S. Churchill, I. S. Kohane, The shared health research information network (shrine): a prototype federated query tool for clinical data repositories, Journal of the American Medical Informatics Association 16 (2009) 624–630.
[17] C. Cappiello, M. Gribaudo, P. Plebani, M. Salnitri, L. Tanca, Enabling real-world medicine with data lake federation: a research perspective, in: Heterogeneous Data Management, Polystores, and Analytics for Healthcare, 2022. To appear.
[18] C. Giebler, C. Gröger, E. Hoos, R. Eichler, H. Schwarz, B. Mitschang, The data lake architecture framework, BTW 2021 (2021).
[19] T. Hlupić, D. Oreščanin, D. Ružak, M. Baranović, An overview of current data lake architecture models, in: 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), IEEE, 2022, pp. 1082–1087.
[20] P. Sawadogo, J. Darmont, On data lake architectures and metadata management, Journal of Intelligent Information Systems 56 (2021) 97–120.
[21] C. Giebler, C. Gröger, E. Hoos, H. Schwarz, B. Mitschang, Leveraging the data lake: Current state and challenges, in: Proceedings of the 21st International Conference on Big Data Analytics and Knowledge Discovery (DaWaK), 2019, pp. 179–188.
[22] E. Barbierato, M. Gribaudo, G. Serazzi, L. Tanca, Performance evaluation of a data lake architecture via modeling techniques, in: Performance Engineering and Stochastic Modeling, Springer, 2021, pp. 115–130.
[23] B. Inmon, Data Lake Architecture: Designing the Data Lake and avoiding the garbage dump, Technics publications, 2016.
[24] A. Agrahari, D. Rao, A review paper on big data: technologies, tools and trends, Int Res J Eng Technol 4 (2017) 10.
[25] A. S. Foundation, Hdfs architecture guide, 2006. URL: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.
[26] A. S. Foundation, Apache nifi, 2006. URL: https://nifi.apache.org/.
[27] J. Alwidian, S. A. Rahman, M. Gnaim, F. Al-Taharwah, Big data ingestion and preparation tools, Modern Applied Science 14 (2020) 12–27.
[28] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. An-

thony Celi, R. G. Mark, Mimic-iii, a freely accessible critical care database, Scientific data 3 (2016) 1–9.
[29] M. Honnibal, I. Montani, S. Van Landeghem, A. Boyd, spaCy: Industrial-strength Natural Language Processing in Python, 2020. URL: https://doi.org/10.5281/zenodo.1212303.
[30] M. Grootendorst, Keybert: Minimal keyword extraction with bert., 2020. URL: https://doi.org/10.5281/zenodo.4461265.
[31] B. Moody, G. Moody, M. Villarroel, G. Clifford, I. Silva III, Mimic-iii waveform database matched subset, PhysioNet (2017). URL: https://doi.org/10.13026/c2294b.
[32] S. Salloum, R. Dautov, X. Chen, P. X. Peng, J. Z. Huang, Big data analytics on apache spark, International Journal of Data Science and Analytics 1 (2016) 145–164.

A. Time Series Processing

In this section, we explain in detail the processing task involving time series. The goal of this operation is to transform data into a more easily interpretable format for analysts and researchers. In this scenario, we consider waveforms extracted from MIMIC-III Waveform Database [31]. Each individual time series of the dataset consists of two files: a header file with information about the measurements (Figure 7), and a signal file containing the values of the measurements, shown in (Figure 8).

In particular, the header file contains a first row with the filename (‘p000020-2183-04-28-17-47n’), the number of signal sampled (‘15’), the frequency at which the signal is sampled (‘0.01666.../125’), the number of rows in the file (‘1315’), the exact start time (‘17:47:59.486’) and start date (‘28/04/2183’) of the sampling sessions. The filename additionally is composed of the patient identifier (‘p000020’), the starting date (‘2183-04-28’) and the starting time (‘17-47’). The following rows indicate which signals are sampled in the file, with a description of the main parameters of the instruments adopted to take the measurements.

The processing task involves the addition of information such as the patient identifier and the start date and start time of the measurements to the file containing the signal measurements. The information is extracted from the header file. The output of the processing for the waveform in Figure 7 and 8 is illustrated in Figure 9.

As it can be observed that each line in the signal file has information regarding the patient identifier and the exact time at which that sample is taken. The exact timestamp of the measurement is obtained after extracting the start date and time of the session from the header file and adding the elapsed time of the measurement to it.

```
p000020-2183-04-28-17-47n 15 0.0166666666667/125 1315 17:47:59.486 28/04/2183
3544749n.dat 16 10/bpm 16 0 986 -15343 0 HR
3544749n.dat 16 10/mmHg 16 0 1157 -6771 0 ABPSys
3544749n.dat 16 10/mmHg 16 0 542 -7302 0 ABPDias
3544749n.dat 16 10/mmHg 16 0 729 13252 0 ABPMean
3544749n.dat 16 10/mmHg 16 0 358 -24414 0 PAPSys
3544749n.dat 16 10/mmHg 16 0 192 -13540 0 PAPDias
3544749n.dat 16 10/mmHg 16 0 262 -8068 0 PAPMean
3544749n.dat 16 10/mmHg 16 0 125 -13706 0 CVP
3544749n.dat 16 10/bpm 16 0 777 -32012 0 PULSE
3544749n.dat 16 10/pm 16 0 98 12894 0 RESP
3544749n.dat 16 10/% 16 0 1000 -26227 0 SpO2
3544749n.dat 16 1/mmHg 16 0 -32768 1238 0 NBPSys
3544749n.dat 16 1/mmHg 16 0 -32768 437 0 NBPDias
3544749n.dat 16 1/mmHg 16 0 -32768 704 0 NBPMean
3544749n.dat 16 100/Lpm 16 0 0 5200 0 CO
```

Figure 7: Example of the header file of the Time Series contained in MIMIC-III Waveform Database.

Elapsed time	HR	ABPSys	ABPDias	ABPMean	PAPSys
(seconds)	(bpm)	(mmHg)	(mmHg)	(mmHg)	(mmHg)
0.000	98.600	115.700	54.200	72.900	35.800
60.000	80.100	113.200	54.500	72.100	34.100
120.000	80.000	114.200	55.200	72.900	34.400
180.000	80.000	118.400	56.800	75.500	34.600
240.000	80.000	123.100	59.600	79.200	34.400
300.000	80.000	130.100	61.600	82.500	34.900
360.000	80.000	133.800	62.500	84.200	37.300
420.000	80.000	130.400	61.000	82.000	33.800

Figure 8: Example of the signal file of the Time Series contained in MIMIC III Waveform Database.

PatientID	TimeStamp	HR(bpm)	ABPSys(mmHg)	ABPDias(mmHg)	ABPMean(mmHg)	PAPSys(mmHg)
20	2183-04-28 17:47:59	98.6	115.7	54.2	72.9	35.8
20	2183-04-28 17:48:59	80.1	113.2	54.5	72.1	34.1
20	2183-04-28 17:49:59	80.0	114.2	55.2	72.9	34.4
20	2183-04-28 17:50:59	80.0	118.4	56.8	75.5	34.6
20	2183-04-28 17:51:59	80.0	123.1	59.6	79.2	34.4
20	2183-04-28 17:52:59	80.0	130.1	61.6	82.5	34.9
20	2183-04-28 17:53:59	80.0	133.8	62.5	84.2	37.3
20	2183-04-28 17:54:59	80.0	130.4	61.0	82.0	33.8
20	2183-04-28 17:55:59	80.0	125.5	59.0	79.1	32.6

Figure 9: Example of processed waveform.

Essentially, the waveform transformation involves putting all relevant information into the single signal file. Each row in the resulting file has the patient identifier and the exact timestamp of the measurement, making it unique in the entire dataset. After being processed, the data are stored in the Refined Zone of the Data Lake ready to be analyzed and/or accessed.