Universitat Politécnica de Catalunya

Big Data Management

Spring Semester 2019-2020

BDM
Project

Authors:
Johannes Bogensperger
Student ID:
Todi Thanasi
Student ID:
Sofia Yfantidou
Student ID: 590152

Supervisors:
Prof. Dr. Alberto Abelló
Sergi Nadal

May 12, 2019

Contents

1 Introduction

The purpose of this report is to propose a Big Data solution for a component of a hospital management system, which is responsible for the online cardiac monitoring of individual patients. Specifically, the hospital has installed vital signs monitors for every patient in the Intensive Care Unit (ICU). These monitors provide among others continuous electrocardiogram (ECG) measurements, namely electrical signals produced by the heart each time it beats. Our system is responsible for capturing and visualizing the ECG stream of every patient, which then serves a twofold purpose. Firstly, the system detects anomalies in the patients' ECGs and alerts the responsible physicians. At the same time the system uses stream approximation techniques to compress the patients' ECGs to enable storing the whole patient history of the hospital more efficiently. At the same time, the system provides batch functionalities as well. At doctor's request, e.g. once an anomaly is detected in an ECG, the system can provide a full approximated history of the patient, as well as the patient's last 24h of intact ECG measurements. This information is presented to the physician through an interactive dashboard. More details on the functionality and use cases of the system will be given in Sections 2 and 4 for better understanding.

1.1 Data Sources

Our system has two external data sources:

1. The **vital signs monitoring system sensors** providing the patients' **ECGs** in an unstructured format. There is an individual ECG signal for each ICU patient. Typical ECGs have a frequency of 1,000-Hz [2], i.e. the pace equals to 1000 measurements per second. This means that, if we calculate a modest 4 bytes per measurement, we need 4 bytes $\times$ 1000 measurements/second $\times$ 60 seconds $\times$ 60 minutes $\times$ 24 hours = 345.6 MB/day per patient. For a medium-sized hospital with an average of 30 patients in the ICU daily, we would need 3.8 TB of storage space per year. These exemplary figures show case that this source is characterized both by immense **volume** and **velocity**. At the same time, different vital signs monitoring systems within the same hospital might have different device firmware versions, due to rolling upgrades or different models, and each firmware revision may have slightly different schemas, e.g. adding new sensors to the system. Thus, this data source is also characterized by increased **variety**.

2. The patients' data, which include the patient ID, the ICU bed ID, which correlates the patient with specific ECG measurements (acts like a foreign key), and the patient's admission and release dates. While this structured data source is not characterized by velocity or variety, the data volume is expected to grow, given the daily patient admission and release rates. Nevertheless, it should not exceed a few GBs.

1.2 Data Model

To understand how to model our data, we first need to discuss how we will query them. Our system needs to enable the visualization of three different data views per patient. Specifically, (1) the streaming ECG data along with their anomaly scores, (2) the last 24h of actual patient ECG data, and (3) daily views of the patient's approximated history for the last 4 visits/hospitalizations. For (2) and (3) we obviously require to store our data for later (not online) visualization. Given the velocity problem of our source, we believe that a data lake, utilizing a distributed file system, is ideal for storing raw files coming from the source, following the load-first model-later principle for cases (2) and (3).

For (2), we need to store the raw data in files named as pID_sID_currentTimestamp.avro, where pID is the patient's ID and sID is the VSMS ID. The naming conventions help us locate the files faster simply by specifying the prefix pID_sID. View (3) has two requirements. First, we need to store the shift coefficients produced online by the F-Shift algorithm (See Section 1.3) in files named as pID_sID_dateIn_currentTimestamp.avro, where dateIn is the patient's admittance date. This naming convention (with dateIn) enables us to locate the last 4 visits faster, avoiding complicated temporal range queries in the patients' table. Second, we need to prefetch the data and precompute the approximations for the last 4 patient's visits for all *current* patients every night, in order to enable faster visualization upon doctor's request. These precomputed approximations are stored in files named as pID_sID_date.avro to enable daily views granularity. We utilize Avro file format because it is more efficient for full-scan jobs, such as data visualization, incorporates its schema, and requires less space compared to plain text files.

Finally, for our patients' data, since they have a structured format and limited volume, we utilize a relational database solution with indexes on patient ID, which allow us to perform point queries on pID to locate the respective sIDs and admittance dates, which are utilized by both (2) and (3).

1.3 Algorithms and Queries

First of all we are going to apply data cleaning techniques to the incoming data, to validate that our Logical Model rules (See Section 2) apply, e.g. not null restrictions, to ensure **veracity** in our architecture. Moreover, data cleaning will help us deal with occasional erroneous data if sensors are damaged or have drifted out of spec, in which case measurements should be ignored. Note though, that we should track consecutive erroneous measurements, in order to take required action, e.g. change of sensor. Following data cleaning, we are going to use two algorithms to exploit our streaming data. Both have linear complexity and their logic is easily parallelizable to run on top of a distributed data processing systems.

1. The **Robust Random Cut Forest** (RRCF) algorithm [1] for anomaly detection. In our scenario, it is important to identify abnormal behavior in patients' ECGs and notify the responsible physician. RRCF uses an ensemble approach and manages to isolate anomalous observations closer to the root of each tree

in the ensemble and to calculate an anomaly score based on the average length of the path between the root and the leaf observation. RRCF is among the state-of-the-art algorithms in streaming anomaly detection and is the official anomaly detection algorithm of Amazon Web Services (AWS).

2. The **F-Shift** algorithm [3] for data approximation. F-Shift achieves a compressed view of the streaming data by constructing unrestricted, error-bounded Haar wavelet synopses. Haar wavelet is one of the most known and uncomplicated wavelet decomposition techniques, and it has already been successfully applied for the compression of ECGs [4]. In our scenario, it is important to approximate patients' historic data to decrease the massive volume requirements as discussed in Section 1.1.

Beyond our PoC, there are more algorithms we could utilize to exploit our data. For instance, we could use a classification algorithm for ECG heartbeat classification, e.g. to classify different types of arrhythmia, based on past ECG annotated data. Moreover, we could use root cause analysis methods, to not only detect anomalies, but identify the reasons behind an ECG anomaly.

Apart from the Machine Learning methods we are going to use to exploit the streaming data, we are going to perform certain queries on batch data, too. Specifically, we will require point queries to retrieve patients' data based on patient ID, e.g. the period of hospitalization and the vital signs monitoring system he was connected to, as well as full-scan queries for the visualization of a patient's history in the system's dashboard.

2 Functional Architecture

2.1 Speed Layer

The speed layer is responsible for ingesting (Stream Ingestion Component), dispatching (Dispatcher Component) and processing real-time data (Stream Processing Component). In our case, the data that fall under this category are the ones from the vital signs monitoring systems (VSMS), namely the ECGs. The ingestion component maintains an event queue for each VSMS where raw data for each device are placed for the dispatcher to handle, to ensure a no-event-loss-policy. Each ingested event should be paired with its source, so it can be mapped to the logical model later. The dispatcher component is responsible for two things; first it ensures that the events map to the specified schema (Veracity) and secondly it routes the events either to the stream processing component and/or to the data lake. In our case the ECG data need to be dispatched to the stream processing component to apply both the RRCF and the F-Shift algorithms, as well as the data lake, where we maintain intact patients' ECG data for the last 24h (older data are only stored in a compressed format through F-Shift).

2.2 Batch Layer

The batch layer is responsible for ingesting (Batch Ingestion Component), storing (Data Lake Component) and processing batch data (Batch Processing Component). In our case, even though we do not require batch data ingestion, we do need to utilize batch processing. Specifically, the batch processing component is responsible for converting the data lake data into views to be presented to the user later. For instance, every night, the system executes a batch cron job, that reconstructs approximations of the patients' data based on the stored F-Shift coefficients produced in the speed layer. These approximations are only constructed for the patients that are currently hospitalized, in order to enable faster visualization of a patient's history upon doctor's request.

2.3 Semantic Layer

The semantic layer (Metadata Repository Component) is an important component of our architecture because it provides useful information about the sensors' data schema and limitations. The semantic layer component uses RDF technologies which facilitate mapping different attributes or concepts to the same entity. The usage of such component is very important for our use case because the ECG sensors are produced from different companies, have different attributes and even when they are of the same brand they may have different firmware versions. Such heterogeneity introduces different data formats that may create difficulties for the other components of the architecture if the semantic layer is missing. Also, the Metadata Repository Component contains knowledge for the pre-processing steps e.g. data cleaning for the ECG sensor's data and related algorithms that we apply on the data. Figure 1 shows the content of the metadata repository.

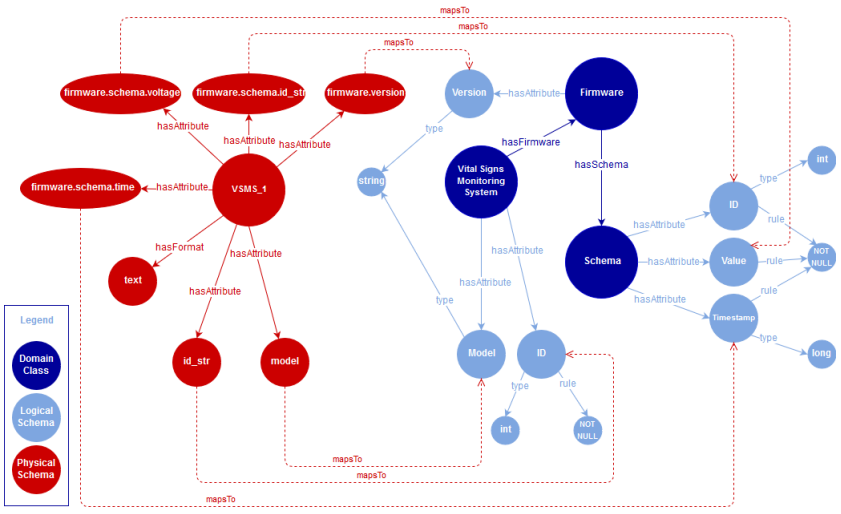

Figure 1: Excerpt of the content in the Metadata Repository.

2.4 Serving Layer

[Redo] Finally, the serving layer is responsible for storing, indexing and presenting data from its Batch Views (Batch View Component) and Real-time Views (Real Time View Component) through its Search Engines (Search Engine Component). The batch view component is responsible for storing and indexing data to allow random reads and aggregations e.g. identification of busy rooms via aggregation of human presence. The real view component is responsible for storing and indexing dynamic and changing data sets, that's why we require low latency, hence we prefer in-memory database solutions. In our case the current dust accumulation data should be available upon user's request in real time. Finally, the query engine component would be slightly different in our case than a typical Bolster SRA. Since we are not dealing with Data Analysts but end-users the queries to be ran in the component should be predefined and only the visualizations should be made available to the user upon request e.g. through a mobile application.

3 Tool Selection

3.1 Batch Ingestion Component

3.2 Data Lake Component

3.3 Batch Processing

3.4 Batch and Stream View Component

3.5 Semantic Layer

3.6 Query Engine Component

3.7 Stream Data Ingestion

3.8 Dispatcher Component

3.9 Stream Processing

4 Use Cases and Data Flows

5 Proof of Concept

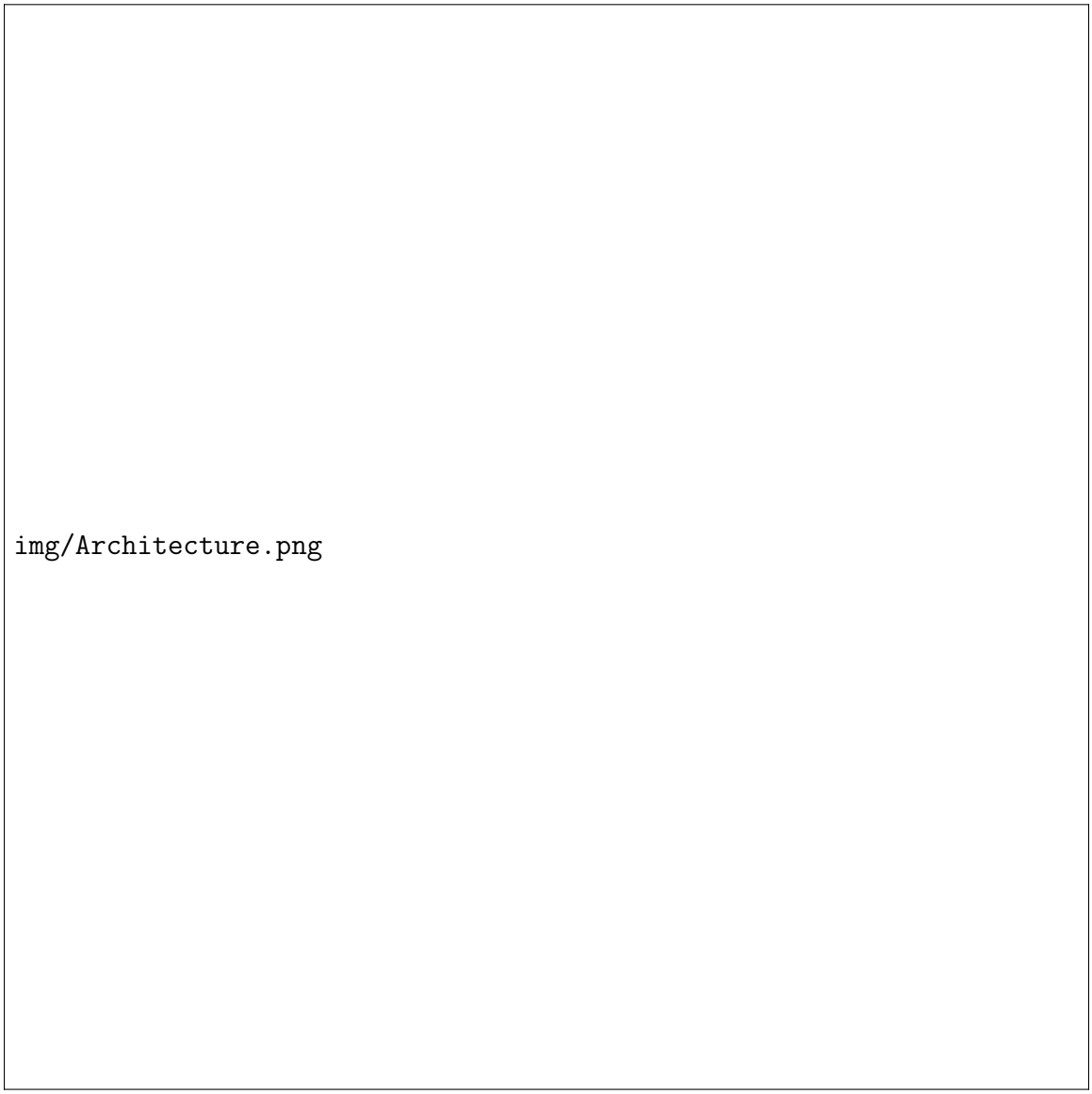

Figure 2: The architecture of the system based Bolster architecture.

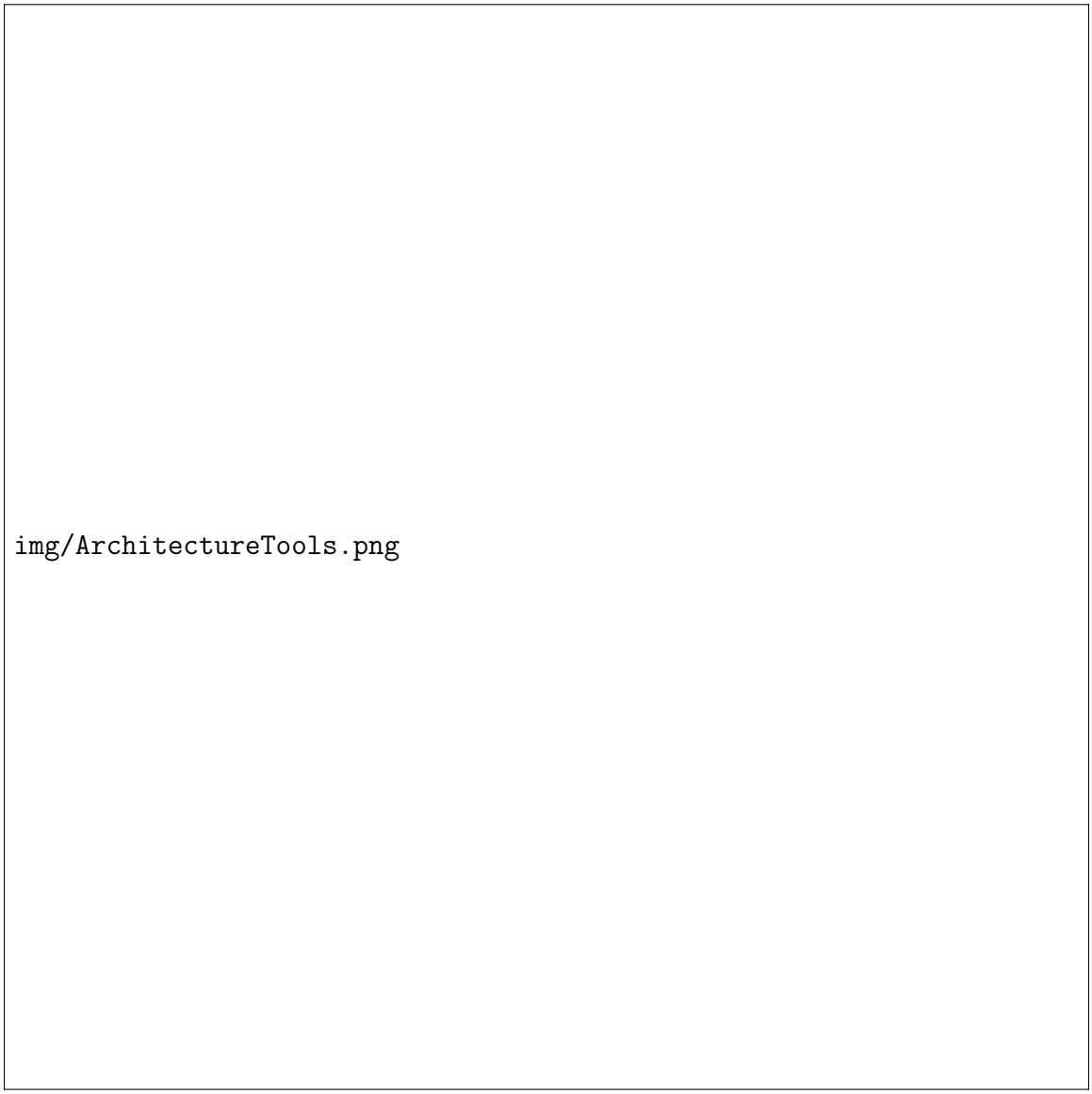

Figure 3: The tools selection for each component.

References

[1] Guha, S., Mishra, N., Roy, G., and Schrijvers, O. Robust random cut forest based anomaly detection on streams. In *International conference on machine learning* (2016), pp. 2712–2721.

[2] Kwon, O., Jeong, J., Kim, H. B., Kwon, I. H., Park, S. Y., Kim, J. E., and Choi, Y. Electrocardiogram sampling frequency range acceptable for heart rate variability analysis. *Healthcare informatics research 24*, 3 (2018), 198–206.

[3] Pang, C., Zhang, Q., Hansen, D., and Maeder, A. Unrestricted wavelet synopses under maximum error bound. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (2009), ACM, pp. 732–743.

[4] Ramakrishnan, A., and Saha, S. ECG coding by wavelet-based linear prediction. *IEEE Transactions on Biomedical Engineering 44*, 12 (1997), 1253–1261.