exeley INTERNATIONAL JOURNAL ON SMART SENSING AND INTELLIGENT SYSTEMS
Article | DOI: 10.21307/ijssis-2018-003 Issue 1 | Vol. 11 (2018)

Patient-centric health-care data processing using streams and asynchronous technology

Kenneth Mbuthia, Jin Dai, Stavros Zavrakas, and Jize Yan*

School of Electronics and Computer Science, University of Southampton Healthera Ltd, Wellington House, East Rd, Cambridge CB1 1BH, United Kingdom.

*E-mails: kjkm1n15@soton.ac.uk, yanjize@gmail.com

Received for publication January 3, 2018.

Abstract

This paper describes a system in detail, which takes in depersonalized data collected by a patient medication management system, carries out reformatting and basic calculations on the data, then stores the resulting information into a database for retrieval, visualization, and further analysis. An investigation is also carried out to create a new way of developing a data analysis application that is efficient and simple to create/code, while avoiding the use of overly complicated libraries to boost performance or investing in expensive hardware. A description of the technologies, algorithms, software tools, and software development process shall be provided. The results from the creation of this application will also be covered while indicating relevant uses for the application, pitfalls/challenges, and future improvements that can be carried out to enhance and improve the system. This application makes use of data collected by an existing patient prescription tracking app that was built by a third-party company that provided dummy data that were used as a guide to develop the system. The application/system created consists of four main sections, a data model project that was used to create the database schema, a "cruncher" project that consists of scripts used to extract data from the existing database, then transforms it into the needed information to be stored, an application programming interface (API) that is used to easily query/retrieve information from the database, and lastly a set of interfaces that visualized the data stored once collected by the cruncher project for easy interpretation and investigation.

Disclaimers: The system described in this paper was developed and tested locally and not in any production environment. All data used for the creation of this paper are dummy data and not real user data. Therefore, all results are simulated and, in no way, violate any real user's privacy. All functionality proposed and developed in this solution represent the potential applications of an analytics tool of this nature and do not represent how any collaborator in this project currently uses the developed system in any real-world/production environment.

Keywords

data processing, prescription tracking, health-care data, analytics, data visualization.

As technology grows more ubiquitous, it has seen its fair share of improvement within the health-care industry as well. With the digitization of information and now the automation of various processes, the health-care industry has seen a face-lift in terms of how it operates and interacts with people, thanks to technology.

While systems become more advanced within different areas such as bookings with medical staff, patient profile management, and even automation of minor surgery, the aim of this paper is to tackle an issue that is growing more important in helping cure patients, finding ways to help patients ensure they take their medication on time. Several applications in the recent past have come up to try and tackle this issue; however, very few leverage on the data they collect in order to investigate further how the product built can be optimized to improve its interaction with the user so that they do indeed take their medication on time, or, on the other hand, anonymize the data to be used in conjunction with medical staff to improve clinical decision-making so that they can better understand and help avoid/mitigate the common pitfalls that hinder a patient's recovery when taking prescription medication. The main problem surrounding patient prescriptions that prescription tracking applications can help solve is antibiotic resistance.

The WHO currently declares antibiotic resistance as a major threat to global health and food security (World Health Organization, 2017). At the moment, the cause of antibiotic resistance is explained by not completing a prescribed dose of antibiotic medication which could lead to the bacteria or virus being treated becoming resistant to the drug. The spread of drug-resistant disease would then become very difficult to manage/resolve, hence the need for patients to ensure that they use the medication given to them by the medical staff according to the instructions they have been provided with. The issue is also present in the situation where antibiotics are overused. This is especially common in farming practices to ensure that animals are reared disease free. The Compassion in World Farming Organization provides several reports on this situation such as overuse of antibiotics in pig rearing or the overuse of fluoroquinolone antibiotics with poultry (World Farming, 2017), which points out the main danger of doing so is the development of resistant bacteria that end up infecting the poultry. Technological innovations have stepped in to attempt to solve this problem with prescription tracking systems, but there is still room for improvement.

Prescription tracking systems have been around for some time, but with the growth of cloud and mobile technology, new systems have come up that leverage this infrastructure to make more robust solutions for people. Recent solutions leverage the use of mobile apps to help patients keep track of their medication, one example being Healthera, a UK-based mobile app that helps patients record the drugs they have been prescribed and reminds them when they need to take it (Healthera, 2017). Another known approach is the use of smart bottles, one example being AdhereTech, a US-based company that developed a smart bottle that detects when the cap is open and when pills are removed from the bottle (Adheretech.com, 2017). It also utilizes Wi-Fi to send out reminders to the user to take their medication. The solution created and documented in this paper is meant to leverage the data collected by such applications to gain further insight on how to improve the service provided to the end user as well as contribute toward clinical decision-making when prescribing medication to patients. The visualization and analysis of adherence data collected by such systems are the core industry use of the developed application. Its other main benefit is analyzing and visualizing data that can be used to improve the product itself.

The objectives of creating this system, and this paper, can be broadly defined as follows:

- A tool that can help the intended creator of the prescription tracking system market their product better and personalize the user experience for customers that join and use the app.
- An investigation into creating a more efficient analytics application by taking advantage of modern technologies, specifically asynchronous calls, JavaScript promises, web sockets, and data streaming. The implementations of these technologies have successfully created an application that is easy to replicate as well as a more efficient (in terms of speed and memory consumption) way to develop a data analysis platform and these identified technologies shall be discussed in more detail in the analytics application section of this paper as well as in the results and discussion section.

The system created as part of this paper makes use of dummy data provided by Healthera, a UK-based company that has developed a prescription tracking mobile app for patients as well as a web dashboard for pharmacists. Thanks to their participation, a working system that carries out analysis and visualization of adherence and patient information was created and tested with dummy data provided by the company. The investigation into a more efficient analytics application that can handle large amounts of data with reasonable memory consumption and loading time was also fruitful. This document shall delve further into how the application was created, how the data models were created, and

how the data were analyzed and visualized as well as how the data can be useful to the company and clinicians alike. Future improvements and proposed features for further development shall also be discussed.

Background

Previous work

A lot has been done when it comes to research and development of prescription tracking systems, but not much is publicly available concerning how data collected by these systems can be used to help improve how they work by increasing customer acquisition and retention as well as aiding in clinical decision-making. These are but some of the potential uses proposed for the system built as part of this paper.

Dhiya et al. (Al-Jumeily et al., 2015) proposed a prescription tracking system in 2015 that suggested creating a website that would help users keep track of their medication. A web-based approach was chosen based on questionnaires they gave out to people to obtain feedback on their preferred platform to access the information, among other points of feedback. They also help point out the factors that comprise good e-health which include empowering the user, efficiency, and cost saving to use as well as ease of use, as some of the primary factors that make up good e-health. However as of 2017, according to Smart Insights, mobile phone usage worldwide has skyrocketed to overtake desktop in various countries across the globe (Smart Insights, 2017), with more than1.6 billion users estimated to be using mobile devices, surpassing the number on desktop in 2014. Due to this information, a mobile-ready solution is much more desirable than a web-based one as people will tend to move around with their phones as much as they would do with their medication when necessary.

A more specific solution that was reviewed in 2016 by Perez et al. (Martínez Pérez et al., 2016) looked at a radiofrequency identification (RFID)-based tracking system for patients with intravenous medication within a hospital. The aim of the system, according to the study, is to mitigate risks involved with treating patients within the hospital such as mistakenly providing the wrong medication to a patient. Preventing adverse events, identifying them, and mitigating their impact were the core objectives the system aimed to achieve. The RFIDs helped do this by storing information about the patient such as treatments they have received and approved medication they are meant to receive. This was available to the medical staff whenever they scanned the RFID tag beside a patient's bed. The system works well for the intended setting but falls short when it comes to empowering the user with accessibility to this information once they leave the hospital. It is also less feasible for patients to carry around RFID tags for their medication wherever they go. A mobile-ready solution in this setting would not be ideal either but would be very well placed to complement such as system by connecting the user to this information so that they can have access to it for future consultation should they need to visit a different hospital or practice.

Creating modern software for the medical field is no trivial process due to the nature of the medical field itself. When dealing with human life and patient privacy, extra safeguards and standards need to be put in place to ensure the highest level of protection and service is provided to both patients and practitioners. Less room for error can be entertained especially with software that helps either the patient or the practitioner, guide their decision-making. It is in this spirit that Boulos et al. (2014) carried out a survey to investigate the quality of modern health-care apps in 2013 and went further to propose a body that could govern and approve apps specifically for health care be formed to regulate and protect patients and practitioners alike. The main aspects identified by the paper in terms of what app developers need to do to meet healthcare standards were legal content quality and usability. Boulos et al. recommended for legal content quality, authoring information, and requiring consent to share medical images, for example, be a part of apps being built for the health-care industry. Ensuring people with disabilities are able to use their apps was the other crucial point given in the paper, and this could be implemented in ways such as an enlarged text option for visually impaired users, a screen reader for the blind, or voice assist for those unable to use hands to interact with the application.

As much as a mobile-based application is at the moment a strong candidate as the best solution to allowing patients to keep track of and remember to take their medication on time, security is an important concern, specifically in reference to protection of sensitive, private, and confidential medical data that the patient will have available on their device. Despite the fact that there are several ways to protect this information nowadays, many users have not been sensitized to the issue or even made aware of it in some cases. A survey done over a two-year period ending in 2016 by Zhang and Costa (2016) showed that 34% of young users (age 17-24) in UK did not use any kind

of pin authentication for their mobile devices. Another 45% of them never change their code and a further 42% of them have only changed it once. Therefore, should their device be lost or stolen, a part of their private medical information would be accessible to potentially malicious third parties. Greater steps then need to be taken to ensure that mobile-ready solutions allowing people to keep track of the medication they take, should be taken to ensure that user information is not just easily accessible to the desired party, but also easy to protect from an undesired party. The suggested solution in this case would be to automate the security layer of applications that need to keep such information ready on a user's device, for example, enforcing the use of a pin to enter the prescription tracking app, encryption of any data cached on the phone, as well as ensuring the app communicates with its server via an encrypted channel.

To continue further concerning security, malware and other software-based exploits would also be a significant risk affecting mobile-based solutions for prescription tracking. Thankfully, most security concerns surrounding the solution can be addressed as suggested above, as well as with elements such as end-to-end encryption, token and two-factor authentication, and database encryption. The current applications that leverage cloud and mobile technology should make use of these tools to protect user information from being compromised by malicious third parties. This is discussed further in the future work and in the improvements section of this paper.

Data analysis and modeling

In order to create a system that visualizes data collected by a prescription tracking system, an understanding of data modeling and analysis is needed. Data modeling can be described as the exploration of data-oriented structures (Agiledata.org, 2017). It differs from class modeling where with data modeling, one identifies entity types whereas with class modeling, one identifies classes and their corresponding subclasses and attributes.

Data modeling had to be carried out in order to not only understand how to manipulate the data, but also in order to extract the relevant information out of it in a way that is also cost effective and scalable. Therefore the "cruncher" section of the application was the most crucial one as it would be the center of all information that the overall system would use. Conceptual, logical, and physical data models all had to be developed in order to create the working solution. The first step was the conceptual models which helped define the domain of what the solution developed would solve, that is, what capabilities it would have, as well as limitations. In order to do that, some business use cases for the data had to be developed which are stated below:

- Marketing purposes: Knowing when and where to market the prescription tracking system to customers. The final platform could give advice on certain activities such as where and when to campaign for pharmacies or patient customers for the app, targeted advertising via email and location-based marketing.
- Personalizing the experience: Data provided by the platform can be integrated into the notification aspect of the existing prescription tracking system to provide a customized experience for users of the app, for example, customizing notifications for patients according to the type of medication they take based on preexisting adherence data for that drug captured by the created platform.
- Assisting with clinical decision-making: At the time the company collaborating on the paper did not seem interested in this as a potential use case, but the developed system can potentially be used in this manner. The data captured by the system could aid the medical staff and/or drug manufacturers in gaining further insight into why people forget to take certain drugs hence allowing them to target the drugs in need of a different approach of administration.

These use cases helped define the areas to focus on for data collection and analysis. The actual data captured is discussed in further detail in the analytics cruncher section of this document.

Logical modeling was also carried out to identify the relationships between different features in the datasets. This was especially useful for the visualization layer of the system because it was important to form comparisons between different data features so as to make more sense out of the data. This is the stage where some data processing and analysis were required. Data analysis is defined as the process of systematically applying logical and/or logical techniques to describe, recap, condense, evaluate, and illustrate data (Ori.hhs.gov, 2017). A lot of evaluation and transformations on the data had to be carried out during the process of the data by the cruncher section of the system in order to store a more desirable format of the data that could be easily manipulated by any application programming interface (API) logic that would be created as part of this system later on.

The other step was creating the physical models which are the database schema that was used by the cruncher section of the system to store the information it processed in the logical models. These schemas had to be created before any code logic from the logical modeling could be written.

Data visualization

Friendly (2006) defines data visualization simply as the graphical representation of quantitative information. This process encompasses the third and final section of the developed system and is the main visual component of the application. Though visualization of information has been around for very many years as documented by Friendly (Friendly, 2006), the techniques used in the process have changed and improved over time. These techniques have changed form and medium, moving from crude drawings on rock or paper, to (what we currently perceive as) sophisticated digitized graphs and charts. Visualization techniques have also greatly benefited the health-care industry largely through magnetic resonance imaging and X-ray technology. Now with the growing popularity of cloud technology and Internet of Things (IoT) devices, other types of data require visualization and in this particular case, data collected by a prescription tracking system. In the health-care field, one of the primary benefits of visualizing data is to help medical practitioners improve treatment of patients. The developed system is no different and has the potential to do so beyond simply providing marketing insights and personalized user experiences for the prescription tracking app itself.

One crucial benefit of visualizing data as asserted by SAS is to help discern important relationships between features in data/datasets (Title WHITE PAPER Data Visualization Techniques, 2017). The developed system took an exploratory approach to the aspect of visualization by seeking out relationships based on best guess such as the number of recorded prescriptions versus the number of recorded appointments or adherence of a drug versus the manner in which the drug is taken by the patient. Advice is also given in the white-paper by SAS (Title WHITE PAPER Data Visualization Techniques, 2017) when handling big data, which is to find a different approach to collapse and condense the results in an intuitive manner. This was very much applied in this case and is necessary so as to avoid any bottlenecks in the data processing pipeline.

Building up on this point, Baldwin et al. (2017) present useful information closely related to this paper with regard to making data collected by health-care applications more useful to their stakeholders. However, the difference between Baldwin et al. and this paper is that this paper is focused on a system that is not customer/patient facing. However, the principles discussed by Baldwin et al. are shared with this paper's objective in the sense that as more information is availed by health-care systems and apps, more work needs to be done in order to make it more easily interpretable by both savvy and non-savvy users hence essentially customizing reports and interfaces to each target audience's needs. One of the main suggestions given by Baldwin et al. to make patient portals more user friendly is providing as much room as possible for displaying explanations or extra information along with the data itself to make it more understandable to a non-savvy user/normal patient.

Despite the large amount of data collected by the application from users across the country, speed and general good performance when processing the data and when retrieving it to present in the visualization layer were the priority for the purposes of this system. This was another obstacle that had to be overcome and is explained in more detail with the solution in the methodology of the analytics cruncher section of this paper.

The analytics application

Overview

The first objective of the paper was to create an application that could carry out analysis and visualization of data collected by a patient prescription tracking system. This would be the first step in a larger plan to use the data to gain further insights and eventually guide business decision-making. How the data that can be used is already discussed in the data analysis and modeling section of this paper. The other objective of this paper was to take a novel approach toward creating a big data analysis platform that implemented modern technology. The purpose of this new approach is to define an easier way of creating such applications, while maintaining or improving their efficiency in terms of loading time and memory consumed during execution.

In order to deliver a scalable and effective solution, the paper had to be split into three sections. This decision was largely controlled by the company and their needs but from the outcome of the paper,

the approach has proven to be not only a good one, but also reusable by others that may have a similar objective. The approach used was to first copy data from the existing database into a new database that was optimized for the task of carrying out complex queries needed to carry out data visualization. This process of copying data would take the form of scripts that would be executed daily and would stream data from the main database, apply transformations to that data, and then store the results in the new database which will be referred to as the analytics database from now on. From there another set of scripts would act as the API that communicated with the analytics database, querying the necessary information that a user requests from the front end of the application.

The solution was therefore divided into four sections:

- Analytics models: This section or folder of the solution contained the code that created the necessary schemas for the analytics database. These had to be carefully designed to avoid being unable to make the needed comparisons between features of the data further down the road.
- Analytics cruncher: This section of the solution is a back-end system that carries out the core data processing of the health-care data as well as transfers and stores the processed or "crunched" data from the main mobile app database into the analytics database.
- Analytics API: This is the system that communicates with the data in the analytics database once it is stored there by the analytics cruncher. It also carries out some minor data transformations and formatting before sending the results to the browser. This happens in order to complete a last-mile step of ensuring the final data structure complies with the format needed by the charting library used in the solution. This approach also allows flexibility should a different charting library be used, therefore rather than storing the data in the exact format needed by a specific charting library, an intermediate format is stored instead that can be easily manipulated for different purposes should the charting library or business requirements change.
- Analytics portal: This is basically the front end of the application and contained the interface code as well as request logic for calling data through the API, and filtering logic for sorting the data loaded onto the page should the user need to interact further with the currently loaded data.

The approach to separate the solution into sections makes it easier to manage the workflow and scale up any section of the overall application or manage/maintain it in future. Concerning the tools and technologies used to develop the application, the main ones are listed below and shall be discussed further in the sections to follow:

- Cassandra: This is the database engine used by the company and therefore the desired tool that needed to be used for the analytics database as well.
- Node.js: This is the back-end scripting language based on JavaScript that was used to create all back-end logic for the entire application.
- Socket.io: This is a JavaScript library that helps provide socket-based communication between the server and the client. This was used to send data to and from the server and helped create a data stream that minimized the amount of time taken for data to arrive at the front end of the application.
- Highcharts: This is a JavaScript-based charting library that was used to create the visualizations on the front end of the application.

HTML, CSS, and JavaScript were also used in the development of the front-end section of the application. A number of libraries were also used in the development of the application both in the front end and the back end and shall be expounded on in the sections to follow of this paper.

Elements that were considered out of scope for the purposes of developing this application were security and authentication layers. This is due to the time constraints present when developing the system. Furthermore, at the time of development, this application would only be for internal use hence time and effort were better spent analyzing the data and creating the core functionalities of the application as well as ensuring the application was built in a scalable way.

Concerning the workflow, a scrum-based approach was taken to develop the platform. Therefore, virtual meetings took place every two weeks between the company and myself and a weekly meeting with my supervisor. This approach helped ensure a large amount of work was completed within the given time frame by aiming to have completed small features by the end of each week, which eventually led to the creation of a large application.

It is also worth noting that though dashboards for data analysis are nothing new in this day and

age, the implementation and overall approach used to create this particular application is what one would consider to be novel. The algorithms used to copy out, transform, and copy in data from one Cassandra database to another could be considered a relatively new approach as it involves the use of streaming technology and performing calculations within a stream, as well as parallelizing certain events that do not rely on each other's output, meaning they were coded to be executed asynchronously. The manner in which data is sent from the server to the client via the analytics API can also be considered novel due to the fact that it does not rely on the traditional request–response approach that most applications have come to use over the years. It instead uses web sockets to create a continuous stream of the desired result set requested by the client. Multiple streams are created with the use of promises during an API request by the client and are sent back asynchronously by the server. These and other implementations of new technologies will be explained in detail in the sections to follow.

The analytics models

This was the most crucial step in the system's development. Despite the many challenges faced in the development of the application, this remained the most challenging step and all other issues that came up later on had solutions thanks to the solid structures created in this step. Healthera provided some fake/dummy data as well as their existing database schema so that it could be used as a reference to create the analytics database.

In order to come up with the table schemas for the database, some experiments and exploratory data analysis had to be carried out. This was preceded by first deciding the overall objectives the application would aim to achieve. From there the experimentation began and this was done by simply querying the data directly from the sample data provided by the company and trying to see if the conceptualized visualizations could be created. A significant amount of time was spent in the experimentation stage to make sure that the final created schemas could produce the desired results while still remain flexible enough to accommodate minor adjustments further down the road.

Cassandra is the database that the application utilizes. This was decided on due to the preference of the company because the final system was handed over to them and they preferred to work with tools that they were familiar with. That being said, Cassandra has its fair share of advantages as a database engine. The most notable two being that it supports data replication out of the box and currently has the best availability in terms of uptime and node failure management (Scalegrid.io, 2017). However, when it comes to querying data and manipulation for the purpose of analytics, it does not provide a very favorable or easy-to-work-with environment. The querying layer of the database tool is quite rigid, and queries will not execute at all unless they meet a number of preset conditions. This, on the other, forces one to have a deep understanding of data modeling and how to go about structuring the data you have in a way that is optimized to accept a specific query you give it. Based on the research online, it seems that people have more success using Cassandra for data analysis when they combine it with tools like Apache Spark; however, this paper was carried out without using Spark because it did not have any JavaScript drivers for connecting to it at the time of development. The company that would take over the management of the analytics application did not have any staff conversant with the languages that Spark currently supports; hence, creating the application without using this tool was one major challenge that was overcome.

Thanks to the querying experiments that were carried out, the tables below showcase the final structure of the tables created for the analytics database. We summarize the features of interest that were considered as part of the scope of what functionality and data the application would be able to have and show, respectively.

Name	Resource_counts
Fields	Type (text) (primary key) date (text) (primary key, clustering column) count (int).
Use	This is useful for counting the number of pharmacies, services, appointments, users, etc.
Comments	This table is not suitable for graphing historical data points, but is useful for showing state at specific points in time, e.g. summation that the widgets in the portal need.

Name	User_locations
Fields	Gender (text) (primary key) city (text) (primary key) date_created (text) (primary key, clustering column) last_login (text) (primary key, clustering column) postcode (text) data (object).
Use	This is useful for viewing user activity by location. You can view locations of active users within a given time period and determine areas that have high or low usage for marketing purposes. Adding a heatmap layer will also bring out high- and low-usage hotspots.
Comments	Created index on postcode as well.

Name	Appointment_count
Fields	Status (text) (primary key) count (int) date (text) (primary key, clustering column).
Use	This is useful for horizontal and vertical comparisons of appointment data. Useful for pie/line charts showing comparison of appointments booked through the app. It is also useful for targeting pharmacies that have a high number of mobile app customers to reward them or look at areas with low app usage and increase marketing there.
Comments	Potentially other uses besides marketing.

Name	Adherence_count
Fields	Status (text) (primary key) count (int) date (text) (primary key, clustering column).
Use	This is useful for horizontal and vertical comparisons of adherence data. Useful for pie/line charts showing the percentage of adherence status, or line graph for how the taken status has been growing.
Comments	Helpful when identifying fluctuation in adherence so as to investigate further.

Name	Pharmacy_counts
Fields	Name (text) (primary key) events (text) (primary key) count (int) date (text) (primary key, clustering column).
Use	This is useful for viewing interactions that pharmacies have with patients through the app over time. It is good for identifying trends in interaction between pharmacies and their customers, e.g. if the number of appointments decrease, you can reach out to users to remind them they can do so through the app.
Comments	

Name	Adherence_drug
Fields	Medicine_name (text) (primary key) totals (object) date (text) (primary key, clustering column).
Use	This is useful for determining which drugs get low or high adherence. Allows you to profile the performance of the app with respect to different drugs. For example: Does mobile app help ‘drug x’ patients more than ‘drug y’, then investigate the cause from there.
Comments	Will need to stream and map remedy ids to medicine name within this cruncher (note: for viewing all drugs within the time period, use adherence_count with “status IN” clause).

Name	Pharmacy_appointments
Fields	Name (text) (primary key) city (text) (primary key) app_date (text) (primary key, clustering column) totals (object) details (object).
Use	This is useful for viewing which pharmacies are actively receiving appointments through the mobile app and which are not. This info can therefore be used to incentivize pharmacies to keep using the app to organize or communicate with patients.
Comments	

Name	Remedy_count
Fields	Dose_type (text) (primary key) count (int) date (text) (primary key, clustering column).
Use	This is useful for horizontal and vertical comparisons of remedy data. Can be used in line or bar graph to show the comparison of different types of drugs users' record in the app. This info can be used to create marketing segments/ targeted marketing, e.g. if there is low usage of people recording injections, you can create a marketing campaign to attract this target group to the app.
Comments	Potentially other uses besides marketing.

In order to manipulate the data by different filter types, for example, date, medicine name or location, clustering columns, indexes, and composite keys had to be used to ensure that the queries made to the created analytics database would be accepted by Cassandra, which in turn meant that they would be as efficient as possible. Clustering columns in Cassandra facilitate ordering results during query construction; defining primary and/or composite keys ensure duplicate entries are avoided and existing records are updated with the most current results, and indexes allowed searching for specific results in an efficient manner.

It is also worth noting that the models do not follow a typical entity relationship structure that SQL databases usually have. This is because Cassandra is classified as an NoSQL database hence data are stored in a loosely or more flexibly structured manner. Cassandra specifically encourages that any table schema created be optimized in a manner that supports a specific query that you would like the table to handle ("A deep look at the CQL WHERE clause", 2017). Therefore, any relationships you form between tables require you to keep the type of query you would like to execute in mind.

Analytics cruncher

This section of the system contains the logic used to extract data from the existing mobile app prescription tracking application, apply some analysis techniques on it such as transformations and tallying calculations, then storing the results in the database tables described in the above section of this paper.

Technologies used

Below are the main libraries used to develop the analytics cruncher:

- Cassandra Node.js driver: this was used to connect to the Cassandra database instance and insert, update, and query data needed by the analytics application.
- Async: this is a JavaScript library that, as the name suggests, provides programmers with straightforward and powerful functions for working with asynchronous JavaScript. Caolan. github.io (2017)
- Hashmap and Superhash: these are Node.js libraries that helped create hashmap objects needed in order to store intermediate data structures during data processing when extracting data from the mobile app to store in the analytics database. Hashmap creates standard key value pairs and comes with helper API calls that allow one to easily manipulate the data stored in a hashmap. Superhash on the other hand allows one to define and store multiple keys within a single entry, hence one can add or update an entry of data in the hash if all the defined keys match (in the case of updating an entry) or are unique (in the case of adding an entry).
- Moment.js: this is a JavaScript library used for date formatting and easily capturing past, present, and future dates by providing simple calls to functions within its library.
- Geocoder: this is another JavaScript library that was used to carry out geocoding of postcodes for the dummy user data that Healthera provided. Using postcodes to obtain latitude and longitude was needed to display user location on a map in the analytics portal section of the system. The implementation of this library depends on Google maps API.

Methodology

A diagrammatic representation of the algorithm used to stream data out, transform/process it, and store it in the desired format can be seen below (Fig. 1).

Using the Cassandra driver, its in-built stream function is used to loop through all records stored in a target table. Streaming through the data was more advantageous than carrying out a "select all" query because it minimizes the amount of memory used to process all the entries any one table may have especially as time goes by and as entries start to increase into

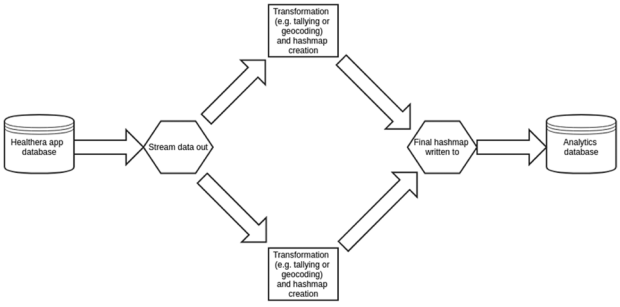

Figure 1: Cruncher algorithm: A simple representation framework based on minimum description length for automatically forming a set of concepts from attribute-value data sets.

hundreds of thousands. Streaming minimizes this memory usage by loading entries one at a time into memory, making them available to be processed or manipulated in any way one may see fit before that entry is discarded and the next entry is loaded. Cassandra's Node.js driver furthermore handles throttling and back pressure when looping through records and although these parameters can be tweaked to enhance performance when dealing with large numbers of records, there is still a fair amount of abstraction the driver provides making it easier to use streaming and hence making the solution ideal for manipulating a large result set.

Figure 1 provides an example of the process that the data go through when any one of the cruncher scripts that were created is run/executed. However, in some cases, transformations were carried out in series rather than in parallel as the diagram suggests. This is because the results of one process were needed in order to carry out the next process. Despite the fact that intermediate data structures had to be created in the process by using the hashmaps, these structures would have a significantly smaller number of entries than the total number of records that the cruncher scripts would pass through when querying the mobile app database. This is because transformations on the data such as tallying or reformatting were carried out within the stream before storing the result in a hashmap or superhash. Therefore, if for example 2,500 records were needed to be streamed from a certain table, if every 10 records happened within the same day, the intermediate hashmap that would be created would have a size of 250 entries and a tally for each day that contained data in the same category would be created within the stream. It was agreed upon that the cruncher scripts created as part of the system would be run every day, hence data collected by the analytics database would be by day and not by the very minute they happened as this level of accuracy was not needed.

Other than tallying and reformatting data, other types of transformations that were carried out include geocoding, date formatting/conversion from timestamp, as well as matching entries across different tables, for example, matching the medicine id in one table with a medicine name in another table so as to identify the adherence by drug of different medications. In some instances, a created hashmap would need to go through more than one transformation before the desired results were obtained. Finally, storing the final hashmap data was a tricky step that was sorted out with the help of the async library and involved looping through the entries in the hashmap using an async function and storing records into the database table one after the other. Thanks to the clever design of the table schemas, any records that already existed would have their data updated rather than duplicated. Storing the entries one after the other was opted for rather than creating batch queries due to potentially reaching the limit on the number of batch queries that Cassandra can handle executing. The end result was five crunchers that were created to handle storing data into the newly formed analytics database tables and are described below (note that the resource_counts table stored a tally of the total records processed from all tables that were queried each day):

- Adherences_import: this script carried out transformations for the adherence data and stored the results in the resouce_counts, adherence_count, and adherence_drug tables.
- Appointments_import: this script carried out transformations for appointment-related data and stored the results in the resource_counts and appointment_counts tables.
- Pharmacies_import: this script carried out transformations for pharmacy-related data and stored the results in the resource_counts, pharmacy_counts, and pharmacy_appointment tables.
- Remedies_import: this script carried out transformations on recorded medicine types (e.g. tablets, injections, liquids) taken by patients and stored the results in the resource_counts and remedy_count tables.
- Users_import: this script carried out transformations on user data and stored its results in the resource_counts and user_location tables.

Analytics API

This section of the system contains the logic used to query data stored in the analytics database once the

cruncher has refined and formatted the data it collected from the existing app database into the desired format. The API also carried out any last-mile processing that needed to occur in order to return a compatible data structure for the charting library to use when rendering the data onto charts. The APIs significance is brought out by the fact that this section of the platform needs to be able to efficiently query information stored in the target database, ensuring the ability to deliver large result sets in a reasonable amount of time, handle large amounts of traffic, and make efficient use of CPU power when operational.

Technologies used

Below are the main libraries used in the development of the analytics API:

- Socket.io: as the name suggests, socket.io is a JavaScript library that makes it easier for developers to get up and running with web socket integration for their applications. Sockets enable real-time communication between the server and the client by maintaining a constant open connection between the two. This shall be discussed in more detail in the methodology section.
- Hashmap: as mentioned previously in the analytics cruncher section, this is a JavaScript library that helps make key-value pairs and provides simplified API calls used to manipulate the data one wishes to store in a hashmap.
- Promise: this is a Node.js library that provides an implementation of promises, a technology that allows the concurrent execution of JavaScript code. This shall also be discussed further in the methodology section.
- Moment.js: as mentioned previously in the analytics cruncher section, this is a JavaScript library used for date formatting and easily capturing past, present, and future dates by abstracting code into simple API calls.
- Async: as mentioned previously in the analytics cruncher section, this is a Node.js library implementation of asynchronous technology.

Methodology

The flowchart below gives a pictorial description of the algorithm used by the analytics API to query data and send it to the client (Fig. 2).

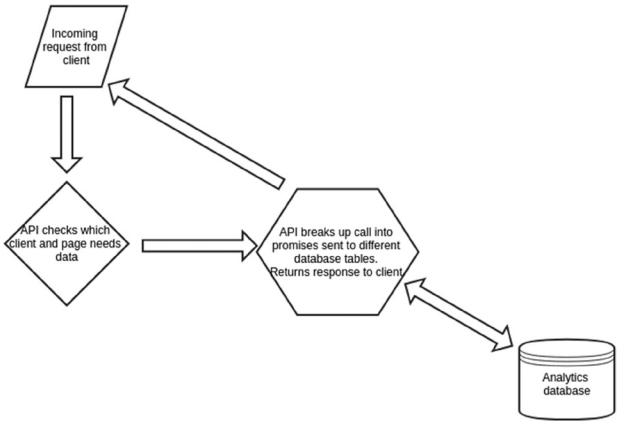

Figure 2: API algorithm.

Once the client initiates a socket connection with the server, a request is sent to fetch data for a particular page and is dependent on a selected date range. Once the server receives this request from a client, a unique id for the client that made the request is obtained from the request that the client sent and is used later to ensure the right client receives the right data. For each page, a number of different datasets needs to be sent back in order to populate the charts on the page. To make these multiple requests happen faster, promises come in to reduce the amount of time it takes to return the results from each query that needs to be executed.

Promises are not that new and were proposed by Liskov and Shrira in 1988 who actually coined the term “promise” and their proposal was the first to suggest incorporating asynchronous calls into a programming language (Liskov and Shrira, 1988). They propose the foundational aspects that are still used in promises today such as the defining of a state that a promise should have. In their paper, these states are defined as blocked or ready but today they are now referred to as pending, fulfilled, and rejected. Promises however started receiving attention within the JavaScript language after the year 2000 when third-party libraries started gaining traction in implementing them using the language until eventually, JavaScript started supporting it natively. The concept is based on allowing functions to be executed without being concerned with when/how long they will take to return their result. This approach then allows other functions that do not rely on each other’s output to be executed in parallel. Promises can also schedule the order in which results from each promise should be stored so as to control the output that different functions within different promises provide. Both these functionalities are used within the analytics API to ensure it operates in an efficient manner.

It is important to emphasize that multiple functions can be executed at the same time and their results

received only when you want them, if the functions are wrapped in promises. This is where the real power of promises lies. Within each query that the promise has helped break apart, any final transformations are carried out on the data so as to ensure the exact format of the response matches that of the charting library used in this system (which is highcharts). From there the API is capable of returning the result either as a JSON response, or by using sockets to send back the data record by record as a continuous stream once a function has finished transforming the data into the correct format for the client charting library. Streaming the data back to the client using web sockets rather than building a large result set before sending out the data saves time and memory because the results hit the page immediately and a large array does not have to be created as output in order to send back the results of any query.

Web sockets are relatively new, and the concept was formally documented and proposed by Fette and Melnikov in 2011, by expounding in detail on a protocol that enables two-way communication between a client running untrusted code in a controlled environment that has opted in to communications from that code (Fette and Melnikov, 2011). The process has largely remained the same to date and is based on using a single Transmission Control Protocol (TCP) to establish two-way communication. An initial handshake is carried out traditionally between a client and server so as to identify each other for future communication. From there an open channel is created between the client and the server to allow them to transfer data between each other without the need for reconnecting. Today it is widely used in gaming, streaming real-time data as well as instant chat applications. This was chosen as the method of communication between the API and the front end of the application for its speed and reduction of memory consumption should large result sets need to be transferred between the server and the client. A socket connection would guarantee that data would arrive on a client's browser faster as each result could be sent as soon as it had been processed into the desired format. Memory on the server side can also be saved by this approach because the server does not have to construct a large, formatted result set before sending it back to the client and instead can send results in chunks in fractions of seconds hence matching the speed of a traditional request–response approach.

Analytics portal

This is the front-end section of the system and contains all code related to the interfaces of the system which consisted of charts, maps, and other widgets for displaying information. The front end also contained some logic in JavaScript that was used to create interactive filters on data that had already been loaded onto the page. The significance of the portal is to provide a real-world example of a simplified method of visually representing the data collected by Healthera's prescription tracking system.

Technologies used

The main tools used to build the front-end of the application besides HTML, CSS, and JavaScript are as follows:

- Helper template: a helper template was used to beautify the user interface (UI) and therefore credit for the design goes to the author of the Minoral HTML template (ThemeForest, 2017) and was purchased for use in this system. The template came with no functionality whatsoever and was purely an aesthetic aid.
- Bootstrap: this was used to make the creation of widgets easier, particularly the date range picker which a user on the front end would use to filter data that they wanted to see according to time.
- Highcharts: this is a JavaScript library that is used to easily create charts of various types. Its robust functionality and customization capability made creating the needed visualizations easier to do.
- Socket.io: as mentioned earlier this is a JavaScript library that provides simplified functions for initiating web socket communication between a client and a server.
- Leaflet.js: this is another JavaScript library that allows one to easily create beautiful maps that are easy to customize and most importantly, add data to. It uses the concept of layers to easily manage the data that gets placed on a map. Some additional libraries were also used along with Leaflet, namely Leaflet's GoogleMutant library for rendering a Google maps layer to provide more details about any location, and the MarkerClustering library that helps group multiple markers that appear close to each other when you zoom into or out of the map.

Methodology

The diagram below represents the general algorithm used by the analytics portal to show/render data onto the web page of a user (Fig. 3).

Figure 3: Portal algorithm.

When any page on the analytics portal begins to load, an initial handshake is sent to the server to establish the web socket tunnel between the client and the server. Once a connection is established, the web page then requests the initial data that should be loaded to the page. The system has been coded to request data for the last three months by default, starting from the date that the user is accessing the page. When this request arrives at the server, the specific page making the request is identified and all needed data for the page is gathered by means of promises as described in the analytics API section.

The response is sent back to the client via web sockets again and each result from each promise emits data to the client asynchronously. Each result emitted is also in turn collected asynchronously and loaded into the relevant chart containers on the page. Once all loading is complete a final response is sent by the server to the client to inform the web page that all data have now been sent successfully. This final notification is used to dismiss any loading animations that may still be on the page. The page then makes use of the third-party libraries to create the visualizations needed which included bar charts, line graphs, pie charts, maps, and animated widgets. All visualizations on each page were also interactive, meaning that a user can hover or click on any item to obtain more information about it. Some of the graphs were also linked and/or contained drill downs that used some JavaScript written from scratch to apply filters on data already on the page.

Results and discussion

To recap on the previously stated core goals mentioned in the introduction, an application that carries out analysis and visualization of patient prescription data was indeed created. An investigation into a more efficient way of building an analytics application that could handle large result sets when queried by a web client was also successful. However, the platform still needs some last-mile integrations and functionality if it is to be able to achieve the goal of customized marketing and user experience for people that join the app that this system will be linked to. This aspect shall be discussed in more detail within the improvements and future work section of this paper.

Created views

Below are screenshots of the web interface of the analytics application.

Figure 4: The portal home page (split into two images above) is the presentation layer of the application that showcases the overall statistics collected by the patient prescription tracking application. In a real-world setting, data are meant to be anonymized and in this demo application, this has been put into effect. The totals and general information presented are handpicked metrics that seemed most useful to an application administrator. They include total users, prescriptions recorded, pharmacies registered, and medication taken on time all within a specified date range. Note that all data shown are not real but for demonstration purposes only and all views are simply

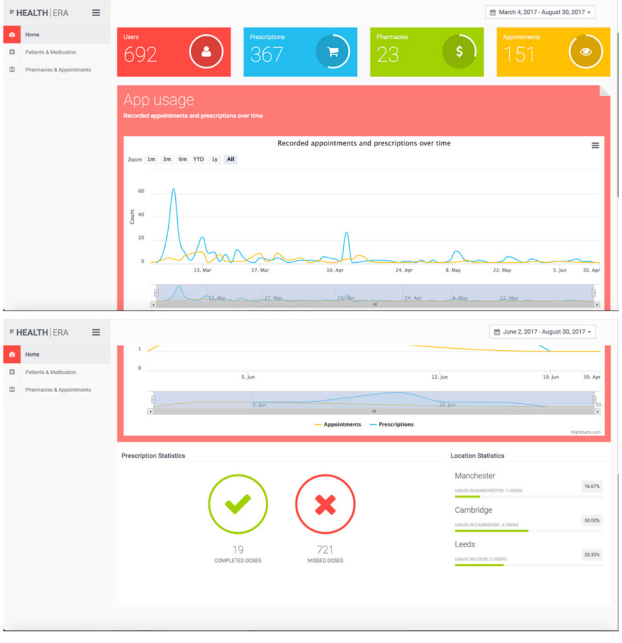

Figure 4: Portal home page.

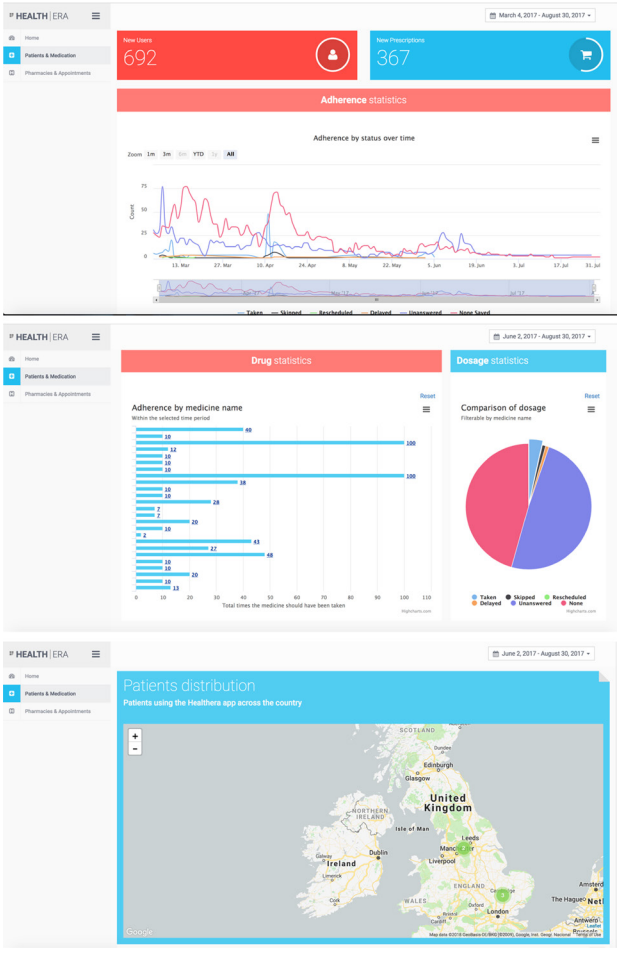

Figure 5: Portal patients' page.

proposals of what the system in a real-world setting could look like and showcase.

Figure 5: The portal patients page (split into three images above) is meant to dive deeper into anonymized user data by displaying information such as the total users on the system, prescriptions recorded, status of the medication they are meant to be taking, and tracking registration locations across the country all within a certain period of time. Most interesting is the comparison of medication status (adherence) that will allow an administrator to see how well users adhere to their medication over time using the multiline graph depicted. The vertical bar chart shows the same information but by medicine name which could potentially also be shared with healthcare professionals to investigate why users adhere better or poorly with certain drugs. Note that all data shown are not real but for demonstration purposes only and all views are simply proposals of what the system in a real-world setting could look like and showcase.

Figure 6: The portal pharmacies page (split into three images above) takes a deep dive into the registered pharmacies data that the patient prescription tracking application collects. The views shown give an administrator information about total pharmacies registered, appointments booked through the app, appointments by pharmacy, and distribution of registered pharmacies across the country all within a certain period of time. Missed appointments and pharmacy distribution can help influence healthcare policy (such as where new pharmacies should be set up or campaigns to spread awareness on the importance of refilling medication on time) if enough patients utilize the app in many parts of a target country. Note that all data shown are not real but for demonstration purposes only and all views are simply proposals of what the system in a real-world setting could look like and showcase.

The analytics portal turned out to be satisfactory in terms of look and feel as well as functionality based

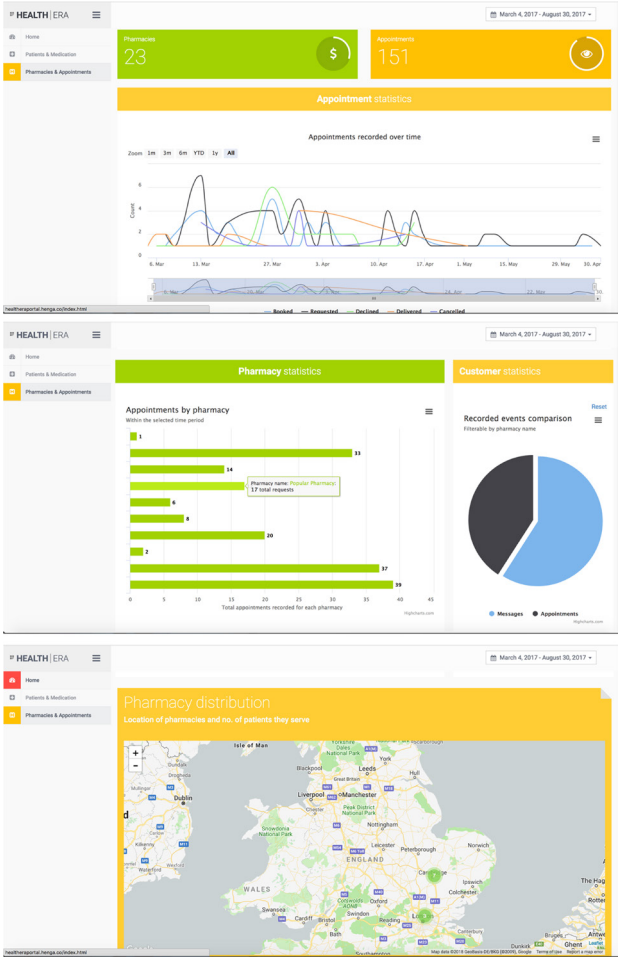

Figure 6: Portal pharmacies page.

on feedback from the developer team at Healthera. The charts visualized their predefined objectives which should allow the company to identify areas of interest in their data. This at least covers the first step in using the analytics to obtain actionable intelligence that the business can use to achieve the goals that the system has been designed to target, which are marketing and personalizing user experience.

The API is built to support both request–response connections to the server it would be hosted on, as well as socket connections. This dual operability was created in order to contrast the efficiency of either option as part of the investigation into creating an analytics portal that is more efficient in terms of speed and memory usage compared with other applications built without utilizing the technologies identified in this paper. The use of sockets and promises was crucial to proving that this approach is better than the widely popular ajax-based, request–response approach and some proof of that is provided in the application performance section of this paper. A web socket connection minimized the amount of time it took before a client requesting data started receiving a response and also took care of reducing the amount of memory needed to transmit a large number of records by sending results record by record rather than waiting for a large array to be constructed in server memory, then sending it down to the client that requested it. Promises took care of breaking apart a request that required a large amount of data from different tables in the analytics database, into manageable chucks. When combined with streaming, this significantly improved the efficiency of the platform because data from different asynchronous calls that promises create could be sent back to a client requesting data simultaneously. This also takes advantage of the fact that modern browsers now support web socket connections.

The cruncher was also able to successfully collect the data that were initially decided upon. Some scripts were much trickier to create than others as they required mapping values stored in different locations. For example, to capture medicine names of drugs that patients were taking, the table that stored patient drug adherence only contained medicine ids; therefore, the names of these medications had to be obtained by mapping the id against the names which are stored in a different table. This is not that straightforward to do considering the fact that Cassandra has a rigid query structure and that this mapping had to be carried out in an efficient way as this needed to happen for a large number of records, every day. These and other challenges are discussed in the challenges section of this paper.

The solution to such problems mentioned above lay in the use of streams and asynchronous technology. Streams were quite helpful in querying the existing mobile app database for all records stored in different tables because by streaming out the records one by one, one avoids the other option which is selecting all records from a table that in turn creates an array of results that places pressure on server memory. Asynchronous calls were quite helpful by allowing multiple functions to be executed concurrently hence saving time and also when storing the results to the new analytics database. The storing of results as mentioned in the analytics cruncher section was done by using an asynchronous call to loop through every fully formatted record and storing them one by one in the new database. Using an asynchronous call was more advantageous than a traditional for-loop because the async library used to carry this out provides different methods one can use to throttle the pace at which insert queries inside the loop get executed at. This helps avoid hitting any limits that a database engine may have when it comes to executing statements while still maintaining speed.

Application performance

It was possible to track the execution time of the request–response approach versus the socket connection approach using the in-built JavaScript function, console.Time(). The screenshots of these tests are shown below (Figs. 7 and 8).

One can notice that the difference in execution time is not that large but the amount of data this was tested on is relatively small. The more expensive queries took dummy data from tables containing roughly up to 3,000 rows. What is more interesting to point out is that that the socket approach does not take longer despite the fact that records are being sent back to the client one by one using asynchronous loops. One must also remember that with the socket method, data can be

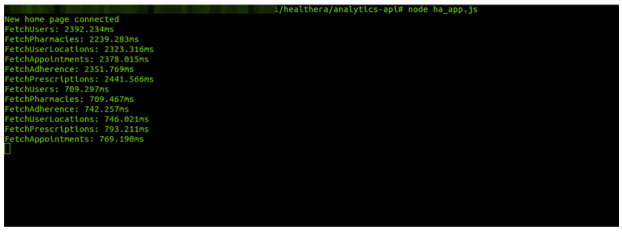

Figure 7: Execution time of functions when using the socket–connection approach.

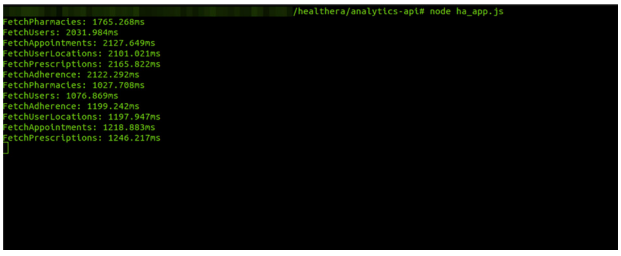

Figure 8. Execution time of functions when using the request–response approach.

sent back to the client as soon as each record is ready to be added to a chart or loaded into a variable on the client side, as opposed to waiting for the entire result set to be processed and then sent back. The socket approach is therefore faster and helps eliminate the process of using extra memory to create arrays containing the desired data. However, an investigation into the amount of memory used to execute the function to emit data to the client multiple times was not carried out as this was a trickier process to try and test accurately. Therefore, an assumption is made on this based on observing the CPU usage when simulating multiple clients connecting to the analytics API and no significant spikes that would raise concerns on the scalability or performance were observed.

Challenges faced

One challenge that was overcome was the data modeling stage. This is due to the rigid nature of querying data stored in Cassandra tables. In order to obtain data in the desired way, for example, using where clauses, sorting, and applying filters, the schema had to be optimized to specifically allow this to happen on the fields that you would like to apply these types of queries to within a database table. The solution to this was carrying out “query modeling”, whereby experiments were first done by writing the query needed to obtain the data for a specific chart and from there working backwards to create the needed table schema that would accommodate that query. This rigidity in how Cassandra works presents one of its biggest drawbacks. However as per the documentation online, the reasoning behind rejecting queries that are not efficient is because the tool supports data replication out of the box; therefore, a badly written query would become painfully slow and hog memory if it needed to call data that is replicated across different nodes. This coincided with the needs of the company, hence their decision to work with the tool.

A potential problem identified by the Healthera developer team at the end of the system was that ids for names, for example, medicine names, were not stored in the analytics database. Only the medicine name was used as one of the unique identifiers. The potential problem of this is that if a medicine was to change its name or a pharmacy was to change names, the analytics would consider this entry as a new record, hence it would create fresh statistics for it. This seems to have been the only major design flaw of the system and as the system moves toward integration and deployment within Healthera or in any other environment, it would need to be addressed and resolved.

Closely tied to this flaw was another challenge that had to be overcome, which was mapping data stored in different tables. Though this is relatively trivial when working with other database tools, with Cassandra it involves a bit more thinking, again due to the rigidity when it comes to querying data. This issue was overcome simply by carrying out careful data modeling. As mentioned earlier in this paper, online forums have also shown that people have had success using Cassandra when carrying out data analysis if they combine it with Apache Spark, but since they were no drivers for this software available in JavaScript, it was not feasible to implement it given both the time constraint to learn it, as well as the fact that the developer team at Healthera had not used it either before and were not proficient in any of the languages that it currently supports. However, deployment of this system in other environments that support languages used by the existing Apache Spark drivers is encouraged, but not absolutely necessary as it may only complicate development further.

One other issue faced was how to go about writing large amounts of data into a database. This needed to happen in the cruncher section of the system and as mentioned earlier, the most viable option identified was not batch querying but rather using an asynchronous loop to write each record into the database.

Improvements and future work

The first improvement to make would be adjusting the models created to identify items that use names with their id as well so as to avoid name changes that would skew any results.

As for future work, one suggestion would be to carry out a last-mile integration of the system with

tools that can help the team carry out targeted marketing and personalizing the user experience for customers of the mobile app. This can take the form of email integration and custom notifications to users through the mobile app.

Another area to work on as mentioned earlier in this paper would be the security aspect of the system. For the current deployment scenario, the system would be for internal use and set up on a secured server within an internal network to prevent unwanted parties from trying to access it externally.

The use of the platform to assist in health-care planning would also be a great contribution to local councils and potentially governments as well. To illustrate how this could work, one example would be to extend the data that the system has access to and combine this analyzed data with other datasets such as mortality rates in different parts of the country. A comparison like this could advise medical professionals on what diseases need more attention in terms of treatment. Another example would be using pharmacy locations to determine areas in the country where pharmacies are needed ensuring that everyone has adequate access to health-care services. However, these scenarios depend on the situation where the mobile app has a large enough database of users and pharmacies in order to make these comparisons.

Conclusion

Creating analytics applications can be a challenging task especially when trying to optimize performance. This can sometimes lead to using multiple tools and languages which end up bloating the code base, making it difficult to maintain, all for the sake of achieving an efficient and scalable result. However, with the approach documented in this paper, we believe that leveraging the right concepts, which now have ready implementations in current programming languages, will make it much easier to create applications capable of carrying out analysis of big data.

By making use of modern techniques such as promises and asynchronous functions, one is able to drastically improve the performance of almost any application that needs to process large amounts of data, not just necessarily health care. With sockets and streams, the performance improves further, and concerning the visual layer, making use of more modern charting libraries and tools would make visualizing the data not just easier from a technical perspective, but also easier to read and interpret for health-care professionals and patients alike.

Literature Cited

"A deep look at the CQL WHERE clause", 2017. *DataStax: always-on data platform | NoSQL | Apache Cassandra*. [Online]. Available at: www.datastax.com/dev/blog/a-deep-look-to-the-cql-where-clause [Accessed: 05-Sep-2017].

Adheretech.com, 2017. "AdhereTech|home", [Online]. Available at: www.adheretech.com/ [Accessed: 02-Sep-2017].

Agiledata.org, 2017. "Data modeling 101", [Online]. Available at: www.agiledata.org/essays/dataModeling101.html [Accessed: 03-Sep-2017].

Al-Jumeily, D., Hussain, A., MacDermott, A., Tawfik, H. and Murphy, J. 2015. Improving communication between healthcare professionals and their patients through a prescription tracking system. 2015 International Conference on Developments of E-Systems Engineering (DeSE).

Baldwin, J.L., Singh, H., Sittig, D.F. and Giardina, T.D. 2017. Patient portals and health apps: Pitfalls, promises, and what one might learn from the other. *Healthcare* 5(3): 81–5.

Boulos, M.N.K., Brewer, A.C., Karimkhani, C., Buller, D.B. and Dellavalle, R.P. 2014. Mobile medical and health apps: state of the art, concerns, regulatory control and certification. *Journal of Public Health Informatics* 5(3).

Caolan.github.io. 2017. "Aync - documentation", [Online]. Available at: https://caolan.github.io/async/ [Accessed: 04-Sep-2017].

Fette, I. and Melnikov, A. 2011. The WebSocket protocol.

Friendly, M. 2006. A brief history of data visualization, *Handbook of computational statistics: data visualization*, vol. 3.

Healthera, 2017. "The app - healthera", [Online]. Available at: http://healthera.co.uk/ [Accessed: 02-Sep-2017].

Liskov, B. and Shrira, L. 1988. Promises: linguistic support for efficient asynchronous procedure calls in distributed systems, Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation - PLDI '88.

Martínez Pérez, M., Vázquez González, G. and Dafonte, C. 2016. Evaluation of a tracking system for patients and mixed intravenous medication based on RFID technology. *Sensors* 16(12): 2031.

Ori.hhs.gov, 2017. "Data analysis", [Online]. Available at: https://ori.hhs.gov/education/products/n_illinois_u/

datamanagement/datopic.html [Accessed: 03-Sep-2017].

Scalegrid.io. 2017. "Cassandra vs. MongoDB", [Online]. Available at: https://scalegrid.io/blog/cassandra-vs-mongodb/ [Accessed: 04-Sep-2017].

Smart Insights, 2017. "Mobile marketing statistics compilation", [Online]. Available at: www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/ [Accessed: 03-Sep-2017].

ThemeForest. 2017. "Minoral - responsive admin template", [Online]. Available at: https://themeforest.net/item/minoral-responsive-admin-template/6867448 [Accessed: 05-Sep-2017].

Title WHITE PAPER Data Visualization Techniques 2017. SAS [Online]. Available at: www.sas.com/content/dam/SAS/en_us/doc/whitepaper1/data-visualization-techniques-106006.pdf [Accessed: 04-Sep-2017].

World Farming, 2017. Why the use of fluoroquinolone antibiotics in poultry should be banned. Compassion in World Farming, 2017 [Online]. Available at: www.ciwf.org.uk/media/7427394/why-the-use-of-fluoroquinolone-antibiotics-in-poultry-must-be-banned.pdf [Accessed: 02-Sep-2017].

World Health Organization, 2017. "Antibiotic resistance", [Online]. Available at: www.who.int/mediacentre/factsheets/antibiotic-resistance/en/ [Accessed: 02-Sep-2017].

Zhang, S., and Costa, S. 2016. A survey study of young generation's mobile phone usage and security concerns. *IEEE Computer Society*.