NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE STUDIES

MASTER THESIS

Big Data Visual Analytics Architecture

Panayiotis I. Vlantis

Supervisor: Alex Delis, Professor

ATHENS

JULY 2018

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αρχιτεκτονική Οπτικής Ανάλυσης Υψηλής Κλίμακας σε Υπολογιστικό Νέφος

Παναγιώτης Ι. Βλαντής

ΕΠΙΒΛΕΠΩΝ: Αλέξης Δελής, Καθηγητής

ΑΘΗΝΑ

ΙΟΥΛΙΟΣ 2018

MASTER THESIS

Big Data Visual Analytics Architecture

Panayiotis I. Vlantis
R.N.: M1387

Supervisor: **Alex Delis**, Professor

THESIS COMMITEE: **Alex Delis**, Professor
Maria Roussou, Assistant Professor

July 2018

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Big Data Visual Analytics Architecture

Παναγιώτης Ι. Βλαντής
Α.Μ.: M1387

ΕΠΙΒΛΕΠΩΝ: Αλέξης Δελής, Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Αλέξης Δελής**, Καθηγητής
Μαρία Ρούσσου, Επίκουρη Καθηγήτρια

Ιούλιος 2018

ABSTRACT

Analyses with data mining and knowledge discovery techniques are not always successful as they occasionally yield no actionable results. This is especially true in the Big Data context where we routinely deal with complex, heterogeneous, diverse and rapidly changing data. In this context, visual analytics play a key role in helping both experts and users to readily comprehend and better manage analyses carried on data stored in *Infrastructures as a Service* (*IaaS*). To this end, humans should play a critical role in continually ascertaining the value of the processed information and are invariably deemed to be the instigators of *actionable* tasks. The latter is facilitated with the assistance of sophisticated tools that let humans interface with the data through *vision* and *interaction*. When working with Big Data problems, both scale and nature of data undoubtly present a barrier in implementing responsive applications. In this thesis, we propose a software architecture that seeks to empower Big Data analysts with visual analytics tools atop large-scale data stored in and processed by *IaaS* infrastructures. Our key goal is to not only yield *on-line* analytic processing but also provide the facilities for the users to effectively interact with the underlying *IaaS* machinery. Although we focus on hierarchical and spatio-temporal datasets here, our proposed architecture is general and can be used to a wide number of application domains. The core design principles of our approach are: *a)* On-line processing on cloud with Apache Spark. *b)* Integration of *interactive programming* following the notebook paradigm through Apache Zeppelin. *c)* Offering robust operation when data and/or schema change on the fly. Through experimentation with a prototype of our suggested architecture, we demonstrate not only the viability of our approach but also we show its value in a use-case involving publicly-available crime data from the United Kingdom.

SUBJECT AREA: Big Data

KEYWORDS: big data, visual analytics, spatio-temporal data, cloud infrastructure, apache spark, interactive programming

ΠΕΡΙΛΗΨΗ

Η ανάλυση δεδομένων με τεχνικές εξόρυξης δεδομένων και γνώσης, δεν παράγει πάντοτε αποτελέσματα αξιοποιήσιμα στη λήψη αποφάσεων. Αυτό είναι ιδιαίτερα σύνηθες στην περιοχή επεξεργασίας δεδομένων Υψηλής Κλίμακας όπου τα υπό ανάλυση δεδομένα μπορεί να χαρακτηρίζονται από υψηλό βαθμό πολυπλοκότητας, ετερογένεια και υψηλό ρυθμό μεταβλητότητας. Σε ένα τέτοιο περιβάλλον, εργαλεία Οπτικής Ανάλυσης (Visual Analytics) μπορεί να επιτρέψουν σε ειδικούς περιοχής ενδιαφέροντος και αναλυτές δεδομένων Υψηλής Κλίμακας να κατανοήσουν και να εξάγουν συμπεράσματα από σύνολα δεδομένων σε υποδομές υπολογιστικού νέφους. Σε αυτή τη περίπτωση οι άνθρωποι αποκτούν αναβαθμισμένο ρόλο αφού γίνονται μέτοχοι και στην ανάλυση των δεδομένων εκτός από την αξιολόγηση και αξιοποίηση τους κατά τη λήψη αποφάσεων. Η Οπτική Ανάλυση, επιτυγχάνεται με τη χρήση σύνθετων εργαλείων τα οποία προσφέρουν στους ανθρώπους μια διεπαφή από και προς τα δεδομένα διαμέσου της όρασης και της αλληλεπίδρασης αντίστοιχα. Στην ανάπτυξη συστημάτων Υψηλής Κλίμακας, η κλίμακα και η φύση των δεδομένων αναμφίβολα εγείρουν εμπόδια στην επίτευξη ικανοποιητικής απόκρισης των συστημάτων αυτών. Σε αυτή την εργασία, παρουσιάζεται μια αρχιτεκτονική λογισμικού η οποία επιχειρεί να επιτρέψει στους αναλυτές να χρησιμοποιήσουν εργαλεία οπτικής ανάλυσης διασυνδεδεμένα με το υπολογιστικό νέφος για ανάλυση δεδομένων Υψηλής Κλίμακας. Έχει γίνει εστίαση σε δεδομένα με χωροχρονικά γνωρίσματα και ιεραρχική δομή, αλλά η αρχιτεκτονική είναι επεκτάσιμη σε άλλες περιοχές ενδιαφέροντος. Οι κεντρικές σχεδιαστικές αρχές της αρχιτεκτονικής είναι: *a)* Διασύνδεση σε υπολογιστικό νέφος για επεξεργασία δεδομένων μέσω Apache Spark. *b)* Υποστήριξη διαδραστικού προγραμματισμού μέσω ενσωμάτωσης Apache Zeppelin. *c)* Εύρυθμη λειτουργία συστήματος ανεξάρτητη από μεταβολές στη δομή και το περιεχόμενο των υπό επεξεργασία δεδομένων. Τέλος, μέσω ενός προχωρημένου πρωτοτύπου, εξετάζεται ένα παράδειγμα με δημόσια δεδομένα από το Ηνωμένο Βασίλειο και παρουσιάζεται η βιωσιμότητα και τα πλεονεκτήματα της προτεινόμενης αρχιτεκτονικής.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Δεδομένα Υψηλής Κλίμακας

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: δεδομένα υψηλής κλίμακας, οπτικοποίηση, χωροχρονικά δεδομένα, υπολογιστικό νέφος, διαδραστικός προγραμματισμός

CONTENTS

1. INTRODUCTION **11**

2. VISUAL ANALYTICS BACKGROUND **13**

3. RELATED WORK **15**

4. ARGUS-PANOPTES DESIGN PRINCIPLES **17**

4.1 On-line Big Data Processing 17

4.2 Interactive Programming 17

4.3 Robust Data/Schema Manipulations 18

4.4 Eliminate Unnecessary Re-computations via Caching 18

4.5 Expose System Internals to User 18

4.6 Promote Visualization Component Reusability with `React` 19

5. THE ARGUS-PANOPTES SYSTEM **20**

5.1 Internal System Architecture Concepts 20

5.1.1 Schema Convergence 20

5.1.2 Data Binning 21

5.1.3 Visualization Chunks 21

5.2 The Architecture 22

5.3 The *VA Client* Functionality 24

6. SKILLS TO OPERATE, ADAPT OR EXTEND `ARGUS-PANOPTES` **27**

7. ASSESSMENT WITH A GOVERNMENT *ASB* DATASET **29**

8. CONCLUDING REMARKS **33**

APPENDICES **34**

A. LIBRARIES **34**

A.1 Scala Libraries 34

A.2 JavaScript Libraries 35

A.3 Ruby libraries 36

LIST OF TABLES

Table1: Required Skills to operate, adapt or extend the system 27

Table2: UK ASB Dataset Key Characteristics 29

LIST OF FIGURES

Figure 1: `Argus-Panoptes` Architecture Layout 12

Figure 2: Visual Analytics Feedback Loop 13
Figure 3: The six phases of BRETAM model 14

Figure 4: *VA Client* UI. 20
Figure 5: `Argus-Panoptes` Full Architecture Layout 23
Figure 6: *VA Client* UI after a Drill-down Operation. 25
Figure 7: Flowchart of Visualization Chunk related operations 26

Figure 8: Evaluation dataset summary 30
Figure 9: Evaluation dataset summary after preprocessing 30
Figure 10: Visualization Chunk computation time / data row count plot 31
Figure 11: Visualization Chunk file size / data row count plot 31

1. INTRODUCTION

Datasets used by Big Data systems and applications are characterized by their complexity, heterogeneity, instant growth, and frequently, noise. These characteristics do affect the quality of automatic analyses performed in a negative way and occasionally, render analysis results to be of either limited or no value at all [1]. By providing appropriate tools, *visual analytics* can help users manually interact with data sets, proceed in an highly exploratory manner and shift the focus of the analyses as the occasion calls along the way [2–4]. However, the traditional use of visualization techniques on large scale datasets does become prohibitive as the volume of the underlying data grows [5]. To address this challenge, we have to adopt contemporary cloud-based computing environments that can accommodate voluminous data by incrementally enlarging the computing cluster (i.e., horizontal scaling).

Apache Spark of the Hadoop ecosystem offers a plausible choice as it can scale up when it comes to non-transactional data [6]. However, the use of Spark as the underlying processing engine of applications calling for high responsiveness is not an obvious choice. Spark introduces inherent latencies that cannot be avoided, only mitigated. Clearly, a number of compensating mechanisms have to be introduced to address this issue. Moreover, to further enhance the user experience, we advocate the integration of *interactive programming* in such Big Data environments. This choice may greatly assist the work of scientist(s) as it offers versatility in handling data and timely decision making during the early exploratory phase of working with datasets. It is worth mentioning however, that by introducing the interactive programming paradigm in this Big Data context, we cannot exploit pre-computation techniques; there are no guaranties as far as the stability of the data is concerned and the data schema remains highly volatile.

In this thesis, we propose a software architecture that helps users effectively interact with underlying *IaaS* stored data, manipulate information using Spark and last but not least, enable *on-line* analytic processing via interactive programming. We mitigate Spark-emanating overheads through the introduction of *1) visualization-chunks*, variable-size granules containing elements shipped over the network and ultimately rendered and presented to users, *2) schema convergence* techniques enabling the seamless transition among different data schemas used across multiple iterations in run-time, and *3)* deployment and intensive use of *caching* at all levels of our software architecture. The aforementioned features can work in tandem and take advantage of hierarchical datasets that we have worked with [7]. Figure 1 depicts the salient features of our proposed architecture. It is decoupled in two key parts: a cloud-based *IaaS* as well as a client-side component. At the server side, the Apache Sparkis used as the Big Data processing engine accepting requests from Zeppelin and *Visual Analytics Server (VA-Server)*. The Spark-server(s) undertake the actual computation and/or management of the stored datasets. Zeppelin is the interactive programming “notebook platform” essentially offering a Web-interface accessible to users through a browser. This notebook-style facility allows users to execute task in a way reminiscent to that of shell scripting and is the prime tool for direct interaction with the Spark engine and subsequently, for manipulating its *IaaS*-stored data. The *Visual Analytics Server* undertakes the central role of coordinating operations among all cloud components and maintains bi-directional `WebSocket` channels with the client side.

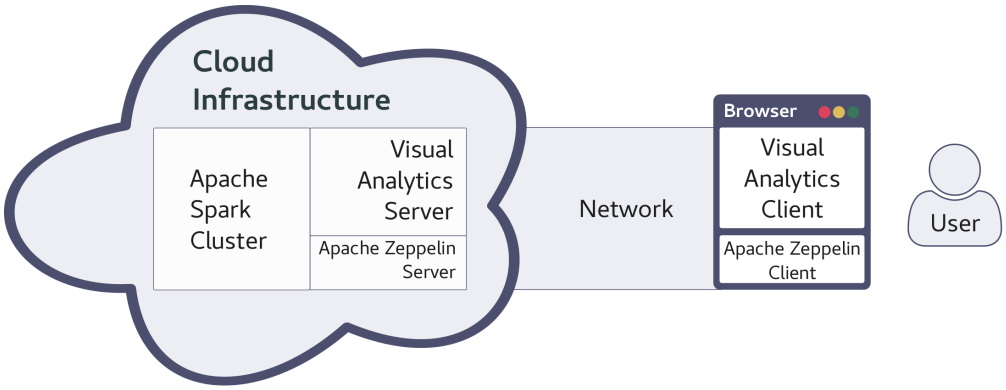


Figure 1: `Argus-Panoptes` Architecture for On-line Big Data Visual Analytics.

The client-side is a *JavaScript* Web application that runs on the user's browser and carries out the functionality of the *Visual Analytics Client*. The latter renders all server-emanating visualization chunks and accepts user-instigated requests through mouse interaction. All parts of the *JavaScript* web application are `React` components. Subsequently, they have to be adapted on a per-case basis however, the use of `React` places strong emphasis on the reusability of components already developed. The Apache Zeppelin window found at the client-side allows for the on-line interactive shell environment receiving explicit requests from users and displaying the outcome of its Zeppelin-server counterpart.

In realizing our proposed architecture, our core design principles have been: *a)* our visual analytics application to carry out its processing on-line on a Spark-cluster; hence, our application is independent of the volume of data utilized. *b)* dataset filtering, joining with others, and transformations are carried out through interactive programming and independently occur from all *VA* aspects. In this regard, users can simultaneously manipulate data through an Apache Zeppelin browser-window while at the same time the *VA Client* interface remains fully operational. We argue that this two-pronged approach can effectively overcome the challenges of pursuing visual analytics on Big Data while at the same time, it yields the basis for overcoming the occasional sluggish response times. Our approach seeks to empower the work of domain experts working along with Big Data counterparts to gain insights and a better understanding through visualization in sophisticated hierarchical datasets.

We have produced a fully-functional prototype that has served as the means to explore a number of Big Data use-cases. In this report, we discuss the effectiveness of our prototype using a real-world Big Data dataset pertinent to crime incidents curated and published from *UK Home Office* [7]. The dataset in question maintains geo-spatial and time-stamped records of incidents since late 2015. The rest of the report is organized as follows: Section 2 introduces some important background aspects of visual analytics and section 3 discusses related work whereas the rest sections are more focused on our work. Section 4 presents the rationale for the design of our architecture. Section 5 outlines the architectural components and their interaction whereas the intended audience for our platform and its skills are presented in Section 6. Section 7 briefly discusses our use-case and Section 8 provides concluding remarks.

2. VISUAL ANALYTICS BACKGROUND

Concept of Feedback Loop in Visual Analytics: *Visual Analytics* can be described as a process, a sense-making loop were insights that user gains in the current iteration, will be used to modify the whole system and proceed to the next iteration repeating the loop. This way, in every iteration the analysis could be driven towards a slightly different direction advancing the analysis further and leading to the discovery of new insights. To be able to modify a Visual Analytics application in such a fashion, the application itself must be designed to allow this kind of incremental development process. This kind of application development, is supported by the paradigms of *Continuous Development* and *Continuous Delivery* and all components included in the proposed architecture and implemented prototype are capable of being developed under such fashion.

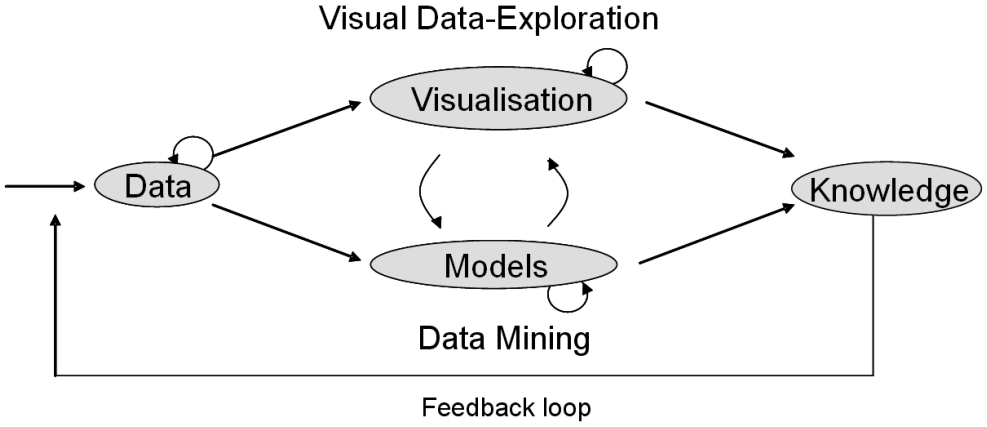


Figure 2: The Feedback Loop in Visual Analytics by D. Keim et al on ”Visual Analytics: Definition, Process and Challenges” [8]

Visual Analytics Evolution: Brian R. Gaines created a model of the development of the information sciences in ”Modeling and Forecasting the Information Sciences” [9]. According to it, there are six stages in an information technology field’s evolution: *Breakthrough*, *Replication*, *Empiricism*, *Theory*, *Automation* and *Maturity*. A short explanation of each stage can be seen in figure 3. Visual analytics, is in the *Replication* stage (2nd) [5, p. 87]. in the proposed architecture we compensate for the lack of VA specific technologies by leveraging technologies from much more mature fields of information sciences. Namely *Big–Data*, *Web technologies* and *Visualization* to compensate for the lack of Visual Analytics specific technologies. An indirect objective of our work in this thesis, is to present a novel way of working with Visual Analytics in the Big Data context and make a small contribution towards the advancement of Visual Analytics field in the next evolution phase.

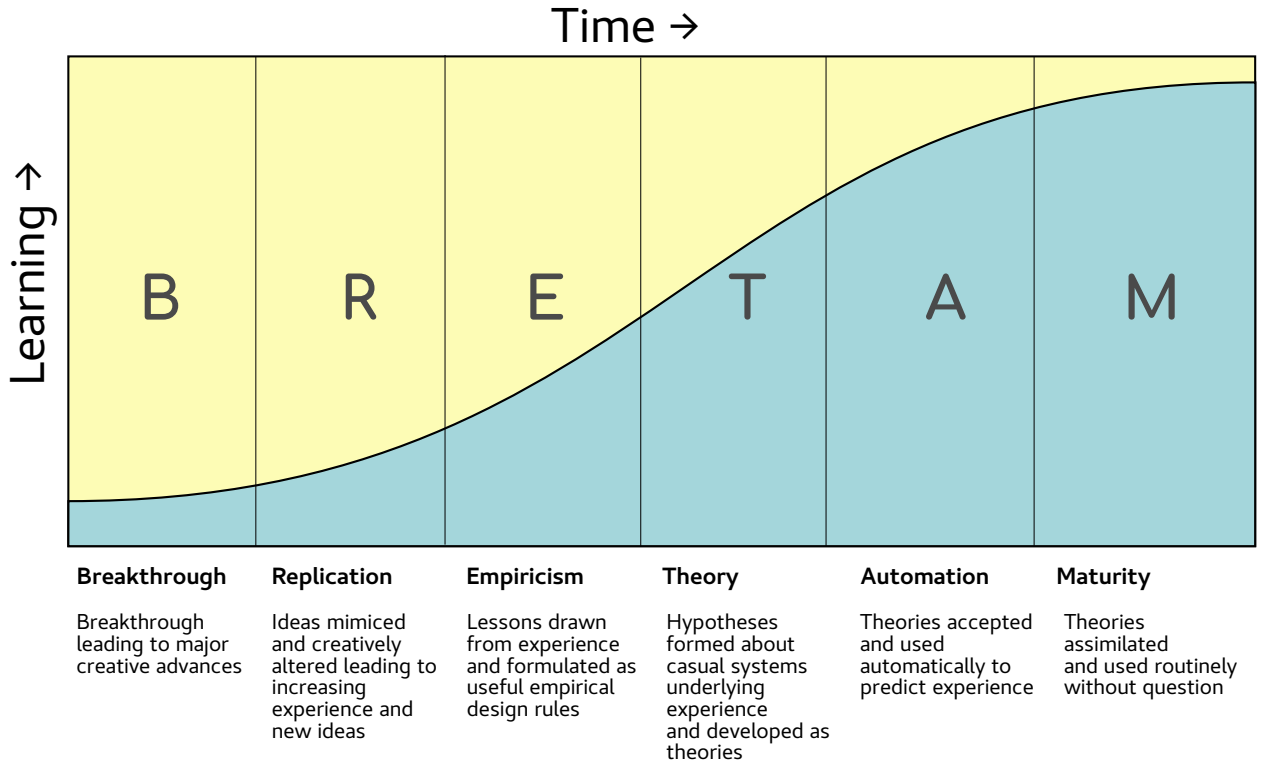


Figure 3: The six phases of BRETAM model by Brian Gaines on ”Modeling and forecasting the information sciences”. [9].

3. RELATED WORK

There has been a flurry of research activity recently in the areas of visualization for Big Data, visual analytics, and visualization recommendation systems [10–15]. Apache Zeppelin [16], Cloudera Hue [10] and Jupyter [17] are open-source initiatives that offer *built-in* visualization functionalities, commonly used in Big Data exploration. All systems allow their users to "send in" either high-level source code such as *Python* and *Scala* modules or dispatch SQL-queries for execution to Spark. They visualize pertinent results with the help of built-in visualization libraries [18]. The main difference between our platform and the aforementioned projects is that we strive to offer user a more immersive experience in visual exploration without calling for continual editing of source code or SQL statements in order to bring about changes in rendered visualizations. In contrast, we let users directly interact with the visualizations produced and the respective interface (i.e., *VA Client*). Moreover, we do not strip the ability to directly manipulate data through high-level source code as our platform does also integrate Zeppelin in its components.

In the area of visual analysis of high-dimensional datasets, *Visualization Recommendation (VisRec)* systems offer a novel approach as they suggest feasible visualizations without major user involvement. These systems, automatically designate and interactively suggest visualization choices for specific tasks at hand. In this regard, such recommendations are particularly useful during the initial phase(s) of exploratory analyses through the creation of a series of alternative visualizations. *VisRec* systems operate by performing pre-computations to analyze the dataset during the offline phase and examine the large space of possible visualization combinations during the on-line phase [13]. While these systems are primed for high-dimensional datasets, their computational intensive on-line phase may make them unsuitable for large scale data without extensive *sub-sampling*.

The use of an *RNN* neural-network is advocated in [12] as the means to help novice users start with visualization. The *RNN* network "examines" a corpus of human-created visualization configurations known so far and along with the schema of the used data it automatically generates a `json`-based visualization configuration. The latter is ultimately consumed by `JavaScript` libraries to display the expected output. In [11], the ZQL-language is proposed as the vehicle to help users designate visual patterns. Such patterns have been extracted from diverse disciplines including biology, engineering, meteorology and commerce. Although this is certainly a novel approach, the number of ZQL-produced possible visualizations remains very high, yielding a somewhat questionable route when it comes to dealing with Big Data. Evidently, the overall ZQL process presents overheads that would be hard to overcome when on-line processing is sought.

The *imMens* project seeks to provide interactive visualization for Big Data in real-time [15]. Similarly to our approach, data binning plays an important role as it is the key technique to attain dimensionality reduction. However, *imMens* overall operations is founded on the concept of *pre-computation* of all data-tiles. This pre-computation occurs in a off-line phase and the respective results are made available at run time to help user fulfill her/his visual analytics tasks. In contrast, our approach performs the respective data tiling by incrementally and dynamically producing `json` *chunks* that can be created on-the-fly

empowering so the on-line mode of operation.

In a recently published work, common patterns that people use to sequence static visualizations for the purpose of storytelling through visualization is studied to determine if there is a natural cognitive tendency towards specific sequencing strategies [19]. In their findings they determine that there is clear preference over hierarchical structuring strategies for presenting and receiving a story consisting of a limited number of static visualization images. The types of hierarhies that were utilized include spatio-temporal dimensions, use-case relevant measures and different levels of aggregation. In our platform, the hierarchical nature of data is exploited in two ways. Segmenting datasets according to their inherent hierarchical structure to allow incremental data exploration on Big Data and allow users to actively manipulate the hierarchical structures to better facilitate the reasoning process. Further research in this area will be of great importance for platforms such as ours as it could provide many opportunities for performance optimizations residing from the fact that we could anticipate the users' next steps as they visualy explore the data.

4. ARGUS-PANOPTES DESIGN PRINCIPLES

We intend on furnishing a software architecture that best serves the merged operations of visual analytics and Big Data analysis. In doing so, `Argus-Panoptes` should not be restricted by the scale of data while at the same time, the architecture should incorporate core visual analytics principles and should display satisfactory responsiveness. Our design leans towards accommodating the experienced user-base as we would like to create a highly-versatile and efficient architecture. In this context, `Argus-Panoptes` maintains an open aggregation and exposes internal components/subsystems to the user. Our design addresses the misgivings of contemporary systems that offer visual analytics on Big Data today. We aspire to address the following 6 design principles while designing `Argus-Panoptes`:

4.1 On-line Big Data Processing

Weaving a platform such as Spark for data processing along with the visual analytics application atop is not a straightforward effort. This is due to the fact that it might take several seconds for the Spark-cluster to respond to even a single look-up operation. In contrast, a typical response in a visual interface is expected to be within the 200*msecs* range. Should we be able to *bridge* the above performance gap, we are to successfully address the design principle in question. Instead of downsizing data through sampling, we advocate *elasticity* of Spark-workers requested by the `Argus-Panoptes` user. We take advantage of the fact that larger datasets call for horizontal scaling of the cluster as applicable operations (i.e., filtering, aggregations etc.) are highly-parallelizable. By delegating all data operations to the Spark-cluster, `Argus-Panoptes` reaps the following benefits: ***1)*** the *VA* application (both client and server components) becomes yet another component in the Hadoop ecosystem. consequently, a large number of tools can be readily integrated into our processing pipeline. ***2)*** users do not have to code in order to export datasets to specific formats required by the *VA* application. The *VA* application has direct access to `DataFrames` in memory as it functions along with Spark. Hence, data pre-processing, cleaning, and *VA* transformations can all be instigated through the Spark programming `API` using `Scala, R, Python`, or `Java`.

4.2 Interactive Programming

For the user to enjoy the maximum benefit while interacting with our architecture, we introduce interactive notebook systems. Such systems include Zeppelin [16], Jupyter [17] and the proprietary Databricks. In general, they all offer a Web-interface in which a user may write code split into *paragraphs* or blocks each pertaining to a specific job or set of jobs. Paragraphs can be executed either sequentially or individually. Individual execution means that if the notebook crashes in the k^{th} paragraph for example, after it has performed some expensive computations, the user can edit the code on the k^{th} paragraph and con-

tinue execution from that point on. This paradigm of code writing and execution is very much desired in our architecture for it facilitates the work of the analyst.

4.3 Robust Data/Schema Manipulations

To attain flexibility, the *VA* application has to operate under uncertainty as far as the current data and its schema is concerned. Robustness of this type is particularly desired as `Argus-Panoptes` deploys interactive programming. In a normal operational work-flow, a user simply manipulates a dataset by either adding or removing features (columns). In erroneous circumstances, issues that may ensue include: *1)* The *VA Client* does not show any data for a user has simply committed a mistake; here, the respective piece of code has to be revised and the query cycle to to be repeated. *2)* The *VA Client* becomes unable to cope with a voluminous visualization chunk consisting in the order of more than $1M$ rows; the user has to reload the tab and start over. In all above cases, we should stress that the *VA Client* functionality is desired to remain strictly stateless.

4.4 Eliminate Unnecessary Re-computations via Caching

It often takes Spark several seconds to compute a visualization chunk. This overhead can be reduced but can not be avoided if identical chucks are requested time and again. Thus, Spark re-computations should and are avoided in our architecture through the adoption of 3 levels of caching: *1)* Apache Spark: when a task is dispatched by the *VA Server*, Spark can avoid execution should it maintain a pool of executed so far jobs. If a `json` file is identified as existing in this pool, its re-computation is bypassed. *2)* the *VA Server* tracks with the help of an SQL database all chunks produced so far and in this manner, contact with Spark is successfully prevented. This caching layer is possible to return stale data due to this reason, a cache invalidation has to be provisioned. *3)* the *VA Client* maintains in-memory a limited number of chunks [1] and the request for fetching the chunk in question from the *VA Server* can be eliminated.

4.5 Expose System Internals to User

As `Argus-Panoptes` operation is intended for the experienced user, the system should make available both internal control mechanisms and system information. A fundamental such control mechanism is the invalidation of caches at either Spark or *VA Server* components. By adopting such a design choice, we avoid error-prone implementation issues that bear little if any significance to the visual analytics domain. Moreover, we argue that making available Spark/*VA* server real-time status information is helpful to the experience user.

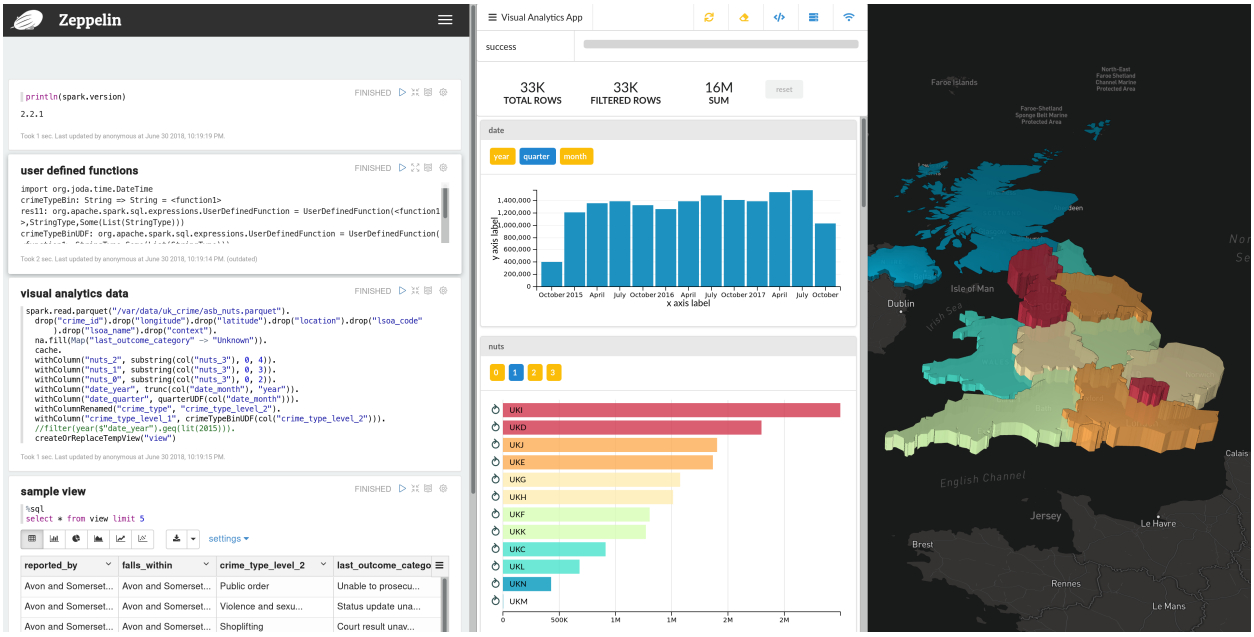
[1] around $200MBytes$ in total.

4.6 Promote Visualization Component Reusability with `React`

Creating a new *VA* application calls for a substantial amount of work most of which is geared towards the development of visualization elements of the interface. This is due to the fact that the effectiveness of a *VA* platform highly depends on tailor-made visualization components. The `React`-framework [20] promotes and provides reuse of `JavaScript`-components in order to built interfaces across multiple *VA* applications. As we strive for `Argus-Panoptes` to not be a one-size-fits-all solution, we resort to the accumulated set of `React` reusable components to rapidly create and/or adapt VA customized interfaces.

5. THE ARGUS-PANOPTES SYSTEM

Argus-Panoptes follows the interactive work-flow of operations that filter, aggregate, summarize and help visually explore diverse aspects of datasets under examination. Fig 4 depicts the interface that the *VA Client* of the system realizes. The UI panel consists of two portions: the first is the Zeppelin window that serves as the means to interact with Argus-Panoptes when it comes to launching of work-flow tasks. On the right side of Fig 4, the *VA Client* browser window displays chart-based outcomes and generated map-related elements. The latter depict generated graphs that help demonstrate trends and assist users gain insight with regards to investigated datasets. The functionality of Argus-Panoptes



Apache Zeppelin browser window

Visual Analytics Client browser window

Figure 4: *VA Client* UI.

is built around three concepts that make the architecture feasible that are discussed next: *Schema Convergence*, *Data Binning* and *Visualization Chunks*.

5.1 Internal System Architecture Concepts

5.1.1 Schema Convergence

Schema Convergence allows the architecture to be be fault-tolerant when the schema of the examined dataset is being actively manipulated. The mechanism is utilized when the *VA Server* ships code to Spark for execution on one side and on the other to *VA Client*. When Spark engine receives a *VA Client* request, it compares the requested with the existing schema of data. Should discrepancies be identified, Spark deals with convergence

so that every element on the stack of the system “perceives” a consistent view. In this process, Spark imposes no restrictions as user requests are offer guidance of what is deemed consistent at any point in time. The data requests of the *VA Client* are predominantly based on what the user has seen last. For instance, if there are new columns in the dataset, they will be included in the converged schema; the same is true when certain columns get dropped. This schema convergence mechanism disengages the architecture from having to deal with state information in its request-response cycle. Evidently, the user can replace the entire dataset through the UI and the virtual analytics platform will continue to operate trouble-free. In this last case, users should also explicitly invalidate caches with the help of the *VA Client*.

5.1.2 Data Binning

Data Binning is heavily used in work-flow processing carried out by our platform. It is a critical mechanism to attain graceful dimensionality reduction for discretizing continuous features; oftentimes, data gets summed before sent for visualization to the *VA Client*. Moreover, binning significantly affects the formation of information hierarchies (or datasets organized in tree-like fashion) that may influence the user’s analytical and reasoning process. Although spatio-temporal data are by and large inherently hierarchical datasets, this is not the case for many others. Binning can effectively assist in the generation (or re-regeneration) of datasets initially featuring no explicit or simply flat structure. We should point out that in our case (re-)generating hierarchies from flat datasets is as vital as feature-engineering is in machine learning [21].

5.1.3 Visualization Chunks

Visualization Chunks are used as the internal unit of information exchanged between the different `Argus-Panoptes` components. A chunk is a `json` file containing the aggregated data for a given dataset hierarchy and the respective dataset schema. Over time, the schema may apparently change. To this end, the *VA Client* receives a visualization chunk once a request has been launched. The outcome of the Spark is a chunk and its main characteristics may either help improve or adversely affect the performance of the system. Visualization chunks are tracked by the *VA Server* and in this respect, their invalidation if needed has to be a explicit user action.

We should point out that the process of of (re-)generating features hierarchies in datasets is linearly correlated to both number and size of the produced visualization chunks. In this respect, we have established that in our experimentation discussed in this paper, the size of the largest size chunk generated is 142*MBytes* and features $670K$ of data-rows. This chunk maintains the highest possible resolution and visualizes all features of the dataset.

5.2 The Architecture

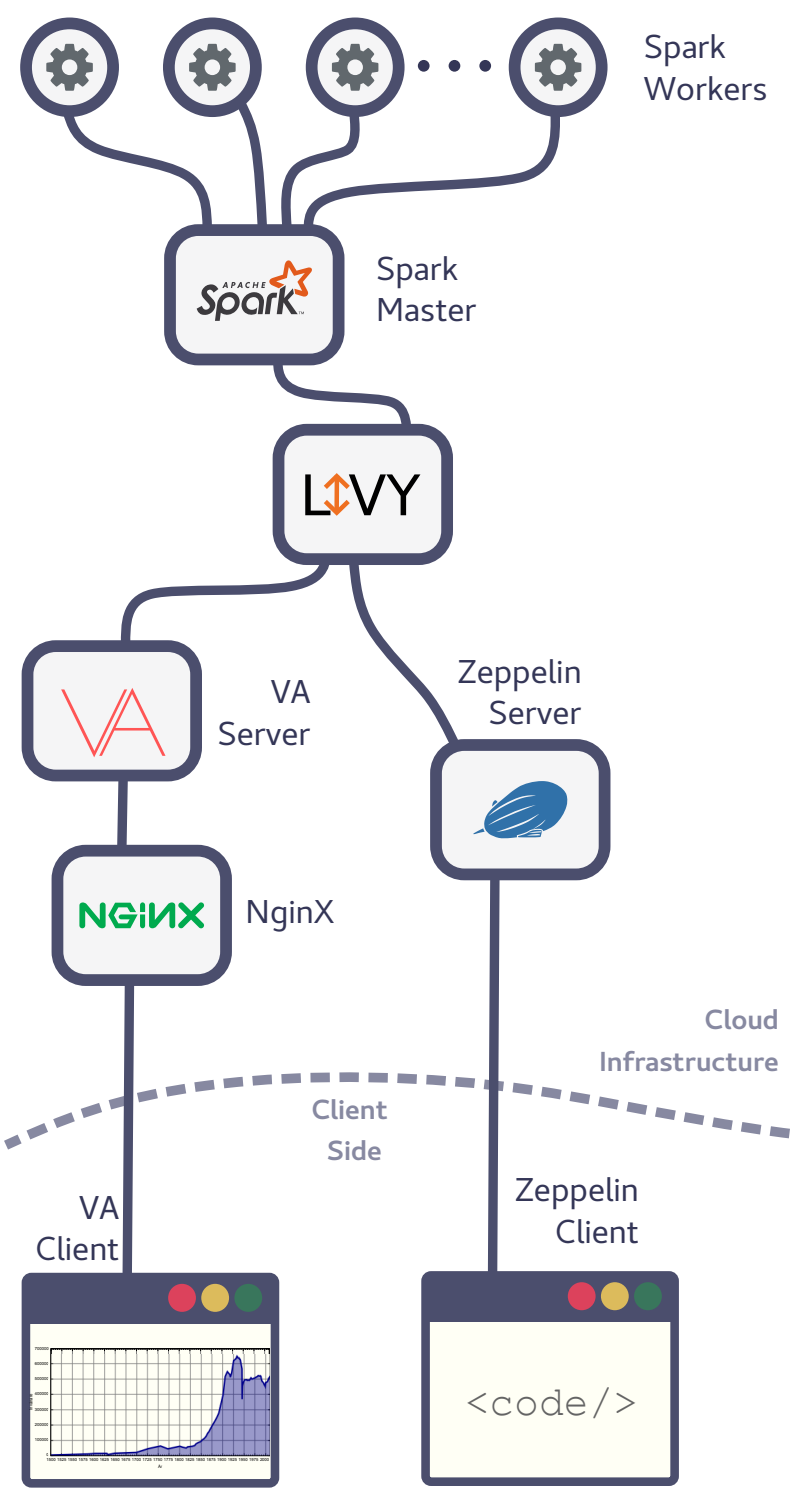
Fig. 5 outlines the architecture of our *IaaS*–based system: the server side is hosted on virtual computing systems as the upper half of the figure shows, while the *VA Client* functionalities are shown to the bottom side. At the core of the server layout, the Spark-engine is referenced as a single entity that may consist of a cloud cluster. It may also involve services from the Hadoop ecosystem. In a minimal configuration, computing cloud consists of an Apache Spark `Master` service deployed in stand-alone mode. A common configuration might involve a Spark `Master`, an number of workers that all use a distributed file system for storing and retrieving data such as HDFS. A more sophisticated configuration may involve the Apache `Zookeeper` for attaining high-availability as well as YARN or `Mesos` for for cluster resource management.

As Spark is not designed to offer a *REST API*, its connectivity with both Zeppelin and *VA Server* presents a point mismatched interface. Natively, Spark may only receive `.jar`, `.py` or .r jobs for Big Data processing over the network. Here, our main concern is to maintain a single *SparkSession* at all times so that multiple clients dispatching jobs to the same visibility scope can be accommodated. Thus, continuity and on-line fashion processing for the client can be warranted. Apache Livy addresses the above issue as it can both provide a *SparkSession* and receive network-requests on behalf on the engine via its *REST API*. In this way, code submitted to Livy can be executed in the same visibility scope. For instance, if a user launches the code `val a="hello"` with the help of a Zeppelin server connected to a Livy session, she can subsequently perform a *HTTP POST* request to Livy via the curl command-line tool and obtain the value of variable a. In our architecture, both *VA Server* and Apache Zeppelin are connected to Livy accessing the same *SparkSession*.

The Apache Zeppelin helps us realize the notion of interactive programming in the context of `Argus-Panoptes`. It is a platform that through its *paragraphs* allows for channeling tasks. On the left side of Fig. 4, we show the paragraphs as well as the controls of Zeppelin. This dialog-based portion of the Zeppelin-UI panel is the main interface facility for users to access the system.

Found in front of Livy, the *VA Server* carries a number of tasks and plays a central role for the coordinated operation of `Argus-Panoptes`. More specifically, the *VA Server*: *1)* serves the *VA Client* with the `JavaScript` Web application. *2)* dispatches code-segments for execution to Spark. *3)* receives visualization chunks from the Apache Spark. The latter are the outcome of Big Data jobs executed at the engine. *4)* maintains two-way communication channels with the *VA Client* with the help of `WebSockets`. *5)* monitors the status of the *IaaS* computing resources and sends pertinent information update snippets to *VA Client* over time. *6)* tracks visualization chunks produced so far and if requested explicitly by the user, it does carry out cache invalidation. *7)* manages chunk-related information and offers an interface for profiling purposes. Developed with `Ruby-on-Rails`, the *VA Server* remains at all times “agnostic” in terms of specific characteristics of datasets under examination.

NginX is a reverse-proxy placed between our cloud-based components and our VA client



Big data processing is performed in Spark Worker nodes

Spark Master is accepting processing jobs, distributes them to the computing cluster and collects results.

Livy server connects to Spark Master, creates and mainttains a persistent SparkSession and makes it available over REST API

Apache Zeppelin and the VA server, execute code in the same visibility scope in the SparkSession maintained by Apache Livy

NginX intercepts visualization data and compresses it to minimize network traffic.

User acesses the visual analytics client-side application and zeppelin web interface through his browser.

Figure 5: `Argus-Panoptes` Full Architecture Layout

(Fig. 5). The proxy is an additional layer for control and abstraction of resources/services and warrants smoother traffic flow between the interconnected servers and clients. In this manner, NginX has an invisible but crucial role as it effectively minimizes network traffic and consequently, enhances the perceived responsiveness of our platform. The main role of NginX is to forward *VA Client* chunk-requests to our server and let the client receive corresponding `json` files through *HTTP*. NginX transparently intercepts all outgoing `json` files and dispatches their `gzip`–ped versions.

One performance improvement that is not obvious is that the decompression of `json` files is handled invincibly by a browser thread different from the one that executes `JavaScript` application code. The *VA Client* `JavaScript` application could perform the decompression itself using a library. However in this case, the execution should take place in a `JS Web Worker` so as the UI does not get halted during processing.

5.3 The *VA Client* Functionality

Our *VA Client* is a `JavaScript` Web application that produces the entire visualization output interface. In this context, the `React/Redux` frameworks have been heavily used as they both promote failure resistance and component reusability. Fig. 6 shows the output window of the UI after two operations have been requested: a drill-down for displaying crime in the London region and enhancement of the date dimension to from quarterly to monthly.

The *VA Client* uses `Redux` as the mechanism to manage the local state. `Redux` helps the application become independent from prior states. Similarly to functional paradigm, the interface we build given a specific state, is always the same. As the schema of the data to be visualized next cannot be predicted, the interface cannot be constructed using information from the current state. We predominantly use the `React` framework for componentization. In a visual analytics application that caters for sophisticated users, the UI is an essential part of the architecture and invariably calls for much customization so that a application is both useful and timely. Hence, the one-fits-all solution approach is infeasible here. By offering components that can be readily reconfigured and reused, `React` plays a vital role in helping us put together effective UIs. In Fig. 6, every depicted visual element is to a `React` component. Among others, there are components that deal with server connectivity, initiate cache invalidation, and refresh visualizations. There are also `React` *Chart* components that help synthesize complex chart dashboards. The portion of UI to the right of Fig. 6 depicts map-based information and is constructed using a third-party `React` Map component [22]. Figure 7 depicts a series of visualization chunk related operations that occur throughout the software stack, as a result of a user's interaction with the interface.

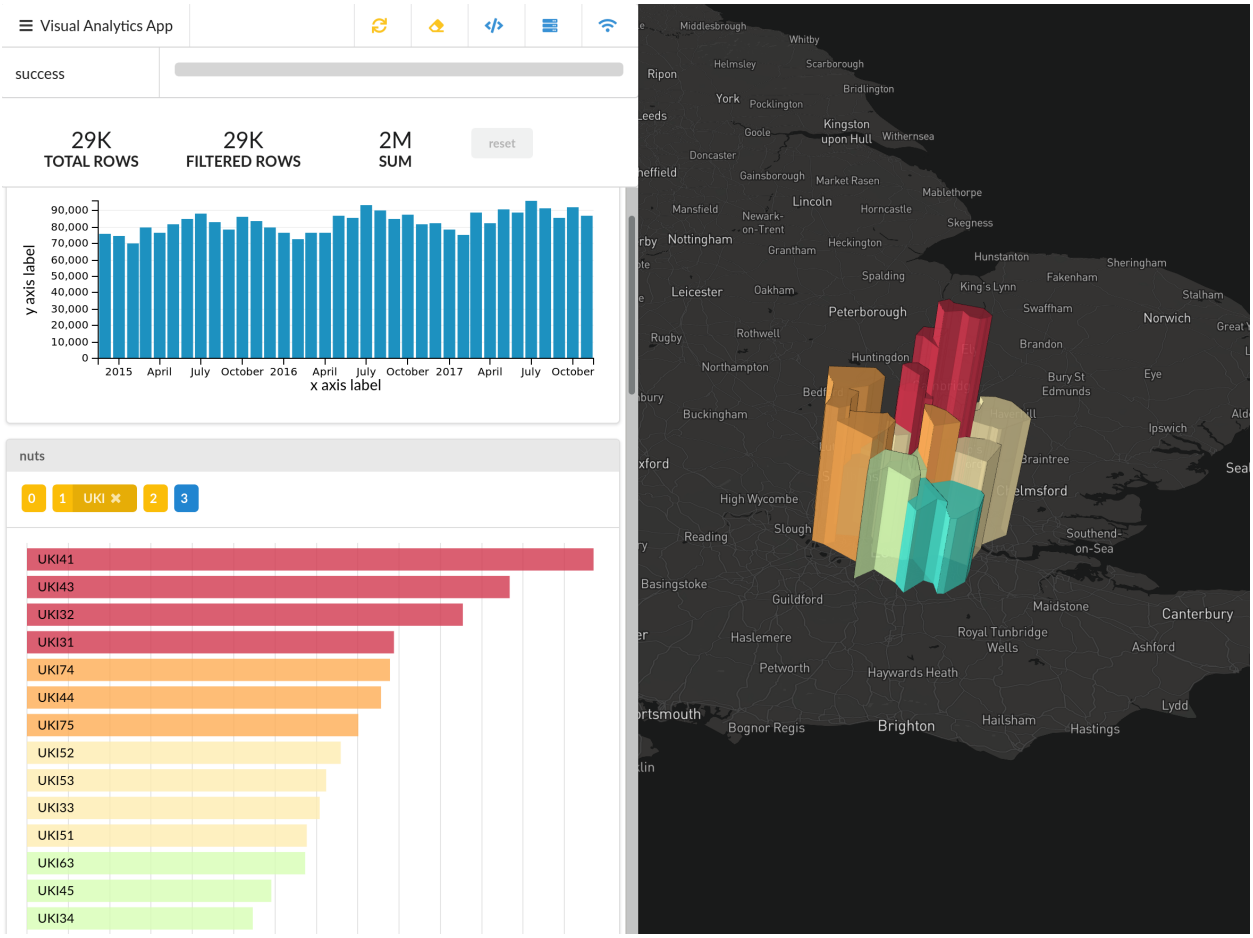


Figure 6: *VA Client* UI after a Drill-down Operation.

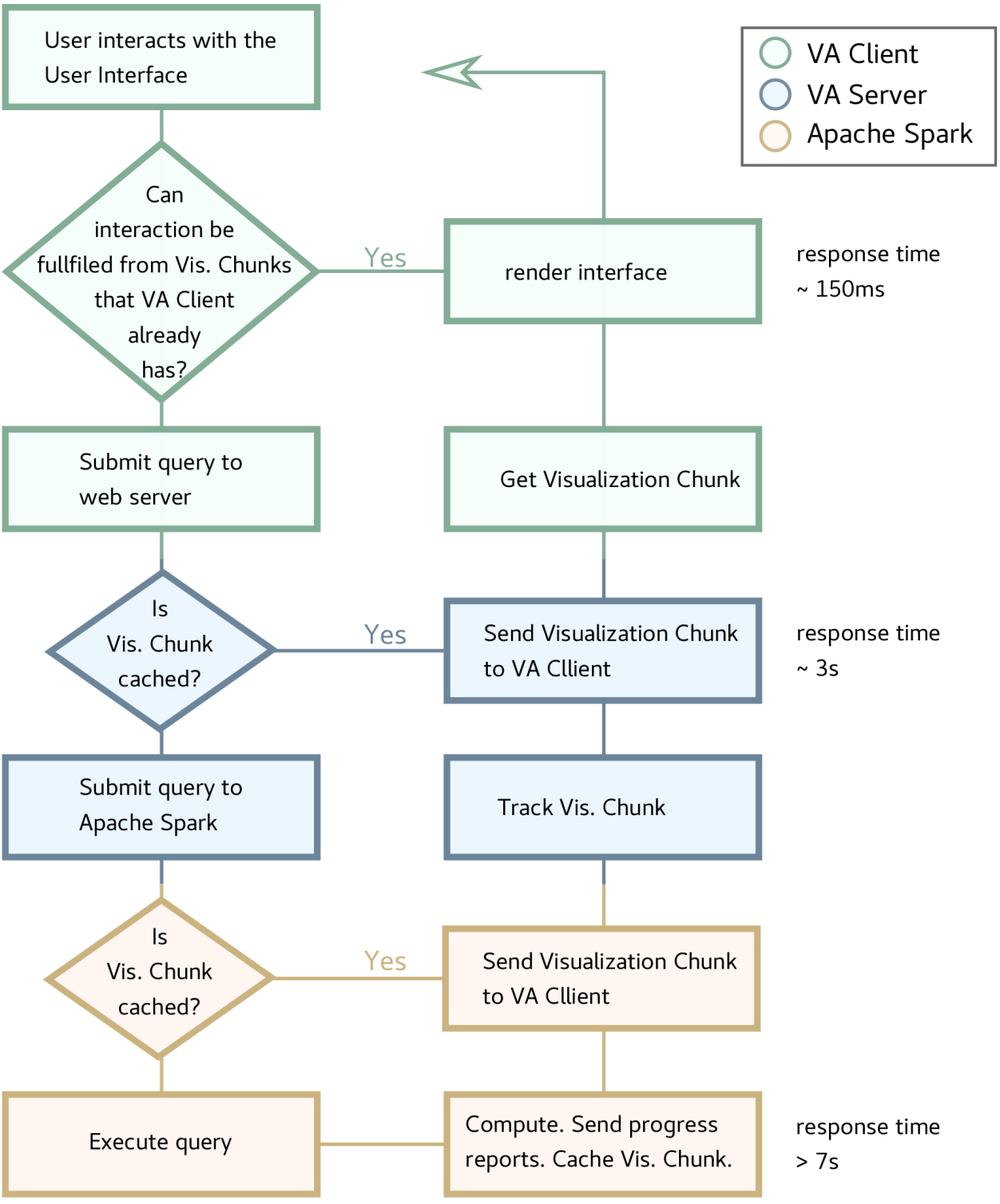


Figure 7: Flowchart of Visualization Chunk related operations that occur throughout the software stack, as a result of a user's interaction with VA Client's interface.

6. SKILLS TO OPERATE, ADAPT OR EXTEND `Argus-Panoptes`

The proposed architecture and `Argus-Panoptes`, the implemented prototype, are comprised of several pieces of software components, coordinated by the VA server application. Different components require different skill-sets to be operated, adapted to accommodate new use-cases or extended to support new functionality. A comprehensive table of user procedures and a description of their respective prerequisite skills, is listed in Table 1.

Table 1: Required Skills to operate, adapt or extend `Argus-Panoptes`. The *required skills level* field indicates if the level of expertise for a skill is very basic, basic, average or advanced.

User Procedure	Required Skills Level	Required Skills Description
Data Exploration	○ ○ ○	Data exploration requires no significant computer skills, it can be performed through intuitive interaction with the UI.
Data Exploration with simultaneous data manipulation.	● ○ ○	Basic programming and data-science skills are required. Basic Scala, Python, R or Java programming skills are needed to make use of the Apache Spark Programming API.
Import a new dataset	● ● ○	Basic data-science skills are needed to preprocess the dataset and basic Big Data engineering skills to import the dataset in a Big Data store.
Adapt - Reconfigure React Components to support the VA Feedback Loop process	● ● ○	Average JavaScript web development skills and working knowledge of the React framework is needed.
Extend the platform with new Visual UI components	● ● ●	Advanced JavaScript web development skills and good knowledge of the React framework is needed.

One objective that we wanted to achieve with this Visual Analytics architecture design, is to allow domain experts to be able to use it without possessing advanced computer skills. Obviously their lack of data-science skills should be compensated by working (even remotely) with another team member that has acquired such skills. Nevertheless this design allows an organization or team to utilize or search for domain experts in a broader pool of available talent.

Apart from data science skills, a fundamental notion in Visual Analytics is the *Feedback Loop* process described in section 2. This is an iterative process (loop) where each iteration leads to adapting the platform to enable the revelation of more insights in the next iteration. Obviously basic data-science skills are not sufficient to adapt the platform. Adapting the Visual components of VA Client requires knowledge of JavaScript and working knowledge of the React framework. However, creating new UI React components and extending the visualization and interaction capabilities of VA Client requires advanced JavaScript development skills and advanced knowledge of the React framework.

Last but not least, the architecture requires advanced Big Data engineering skills to be fully operational. However, those skills are a requirement for any kind of Big Data analysis and not specific to our architecture. On the contrary, due to our design choice of integrating the platform with Apache Spark we allow Big Data practitioners to readily import datasets to our platform conveniently with a minimal amount of work.

7. ASSESSMENT WITH A GOVERNMENT *ASB* DATASET

Salient Argus-Panoptes features evolved during the prototype development. Experimentation with different real-world data-sets from various disciplines also contributed to the realization of the system. In general, dataset features entail: ***1)*** sized textual data, ***2)*** raw tuple-based data for each incident that has received no aggregation, ***3)*** Geo-spatial features, ***4)*** timing information, and ***5)*** other continuous or discrete features. In this section, we briefly present our experience with a publicly available dataset about crime. We use Argus-Panoptes as a spatial decision support analysis tool. Below, we discuss the pre-processing, the (re-)generation of feature hierarchies and our profiling of Argus-Panoptes.

The utilized dataset is curated and published by *UK Home Office* [7]. It maintains individual crime and anti-social behavior (ASB) incidents including street-level location information and is published in CSV format. All features in the dataset, are textual with *longitude* and *latitude* being numeric. Table 2 shows all features among with the number of distinct and null values for each feature.

Table 2: UK ASB Dataset Key Characteristics

Name	Distinct	Null	Description
crime_id	12665725	5510538	Incident identifier string
longitude	737765	301155	Longitude
latitude	731070	301155	Latitude
location	280694	0	Human-readable approximate location
lsoa_code	35921	778773	UK-designated area code
lsoa_name	35065	778773	UK-designated area name
reported_by	46	0	Reporting department
falls_within	46	0	Department with jurisdiction
month	35	0	Date string formatted as %Y-%m
last_outcome_category	26	5806479	Last outcome category
crime_type	14	0	Category of crime
context	0	18268085	Deprecated field

In our pre-processing phase, we transform and store the dataset in a format suitable for our analyses. In particular, both Spark Master and Spark Worker nodes should be able to import the format in question correctly. For the dataset of Table 2, we carry out the following preprocessing steps: *1)* transform date found in the *month* field to Date datatype, *2)* drop the deprecated field *context* as well as the *crime_id*. *3)* drop fields *lsoa_code*, *lsoa_name*

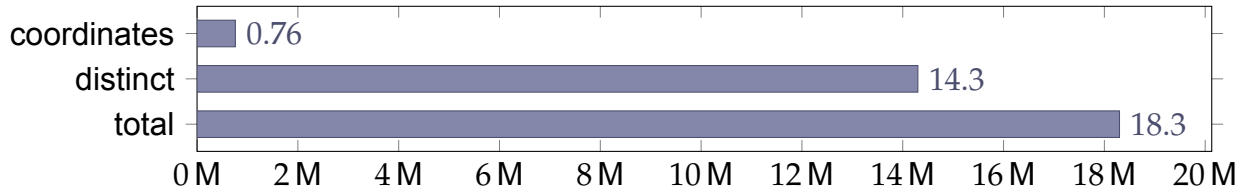


Figure 8: Dataset has 18.3M rows: 14.3M distinct rows and 760K distinct coordinates.

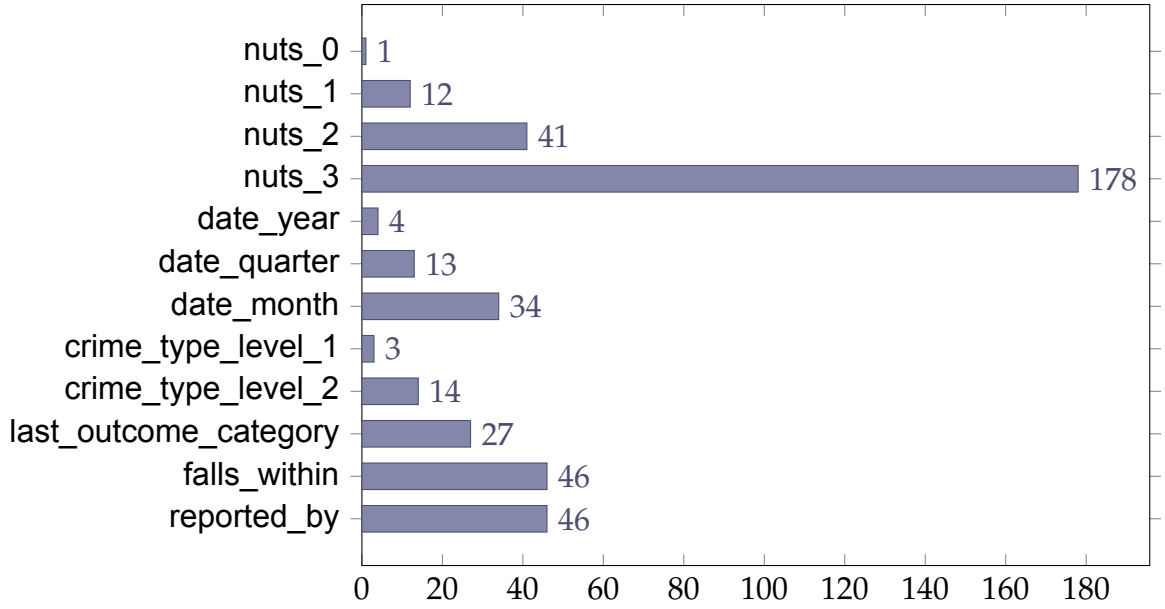


Figure 9: Distinct counts for every feature resulting from geo-joining and data binning.

and *location* deemed as redundant information, *4)* save the `DataFrame` in an efficient columnar data representation like Apache Parquet.

We also transform the UK dataset by joining it with the NUTS classification scheme of Eurostat [23]. This enhancement offers varying granularity in regional information that has the following 4 levels: country (NUTS_0), major socio-economic region (NUTS_1), basic region (NUTS_2), and small region (NUTS_3). We use the Magellan [24] Spark–library to perform the geo-join between the coordinates of each point and the *area polygon* of each region of the NUTS scheme. The geo-join helps us obtain the NUTS dimension having only 178 distinct values, whereas the distinct values of the prior coordinate features were $760K$. (Fig.9) This geo-join operation is CPU-intensive but it occurs only once and so, we make the data persistent for further `Argus-Panoptes` processing.

We also tinker with two more dimensions: *crime_type* and *date*. Through binning, we create 3 distinct types of crime: theft-related, anti-social behavior, and others. Then, we map the original 14 crime types to populate the 3 new bins. Similarly, we bin the time attribute of the dataset to populate quarterly and yearly levels. Fig. 9 reveals distinct counts for all features of the dataset after introducing hierarchies. In contrast to the geo-joining, binning is an inexpensive operation and can be carried out on-line without affecting the system responsiveness. The latter affords the user to experiment with the introduced

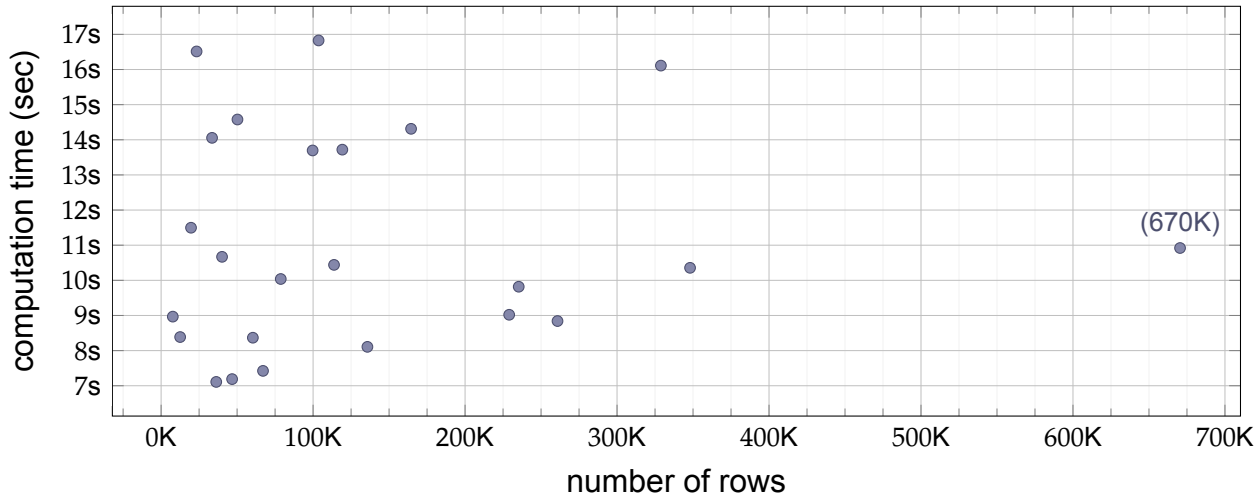


Figure 10: Visualization chunk computation times in relation to the row count of the aggregated data. Big Data aggregation operations are taking several seconds to complete and are independent of amount of data returned.

hierarchies on-line and if needed, realign them.

The aforementioned generation of hierarchical dimensions results to a maximum of 24 distinct chunks. Fig. 10 shows the computation time required for each of these chunks in conjunction with the number of visualization tuples each one contains. It takes anywhere between 7.10 and 16.80 *seconds* for chunks to be computed. The above range represents an acceptable delay as the computation of each chunk occurs only once. Through caching, subsequent accesses to already computed chunks is only dependent to the volume of the data ultimately transported over the network to the client.

Fig. 11 depicts the `json` and the corresponding compressed file sizes for all 24 different types of chunks. If `json` files were transported uncompressed, their size would range

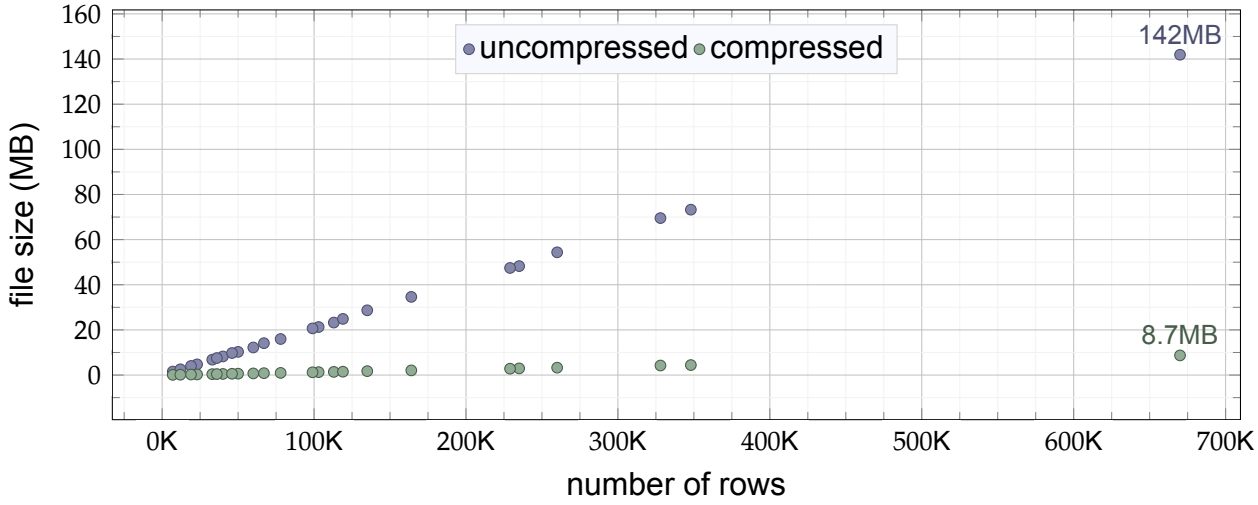


Figure 11: Chunk file size with row count of data they contain. The `json gzip`-ped files are transfered from the cloud to the browser.

from 1.56*MBytes* upto 140.90*MBytes*. In actuality, all such files are transfered `gzip`-ped and their sizes ranges between 0.08*MBytes* and 8.70*MBytes* with average chunk size being less than 2.00*MBytes*. Such sizes facilitate both the sought on-line type of operation and accomplish responsiveness for our prototype. Last but not least, we should indicate that a large number of visual interface interactions can be immediately served by already cached content in *VA Client*.

8. CONCLUDING REMARKS

In this thesis, we propose Argus-Panoptes, a visual analytics system that incorporates cloud-based Big Data processing in its core. Our key objective has been to combine Big Data processing with visual analytics so as to further empower both domain experts and data analysts. Our proposed architecture offers a number of novel mechanisms that entail interactive programming for direct manipulation of both datasets and operations, on-line processing through the use of Spark-clusters, robust operations through dataset schema convergence and use of highly reconfigurable UI components. Our system design involves both home-grown virtual analytics server and client components as well as state-of-the-art systems such as Zeppelin, Livy, Spark and NginX. We have evaluated Argus-Panoptes using an enhanced spatio-temporal crime dataset from the U.K. Home Office and have ascertained the effectiveness of our prototype through profiling of its operations. Finally, there are certain extensions to the application that could greatly improve its functionality and responsiveness. One obvious improvement is to make existing UI React components more responsive to the data they visualize as well as enrich the collection of existing components. Another important improvement is related to the current implementation of the Visualization Chunk mechanism. Since there is no restriction on the amount of data a Visualization Chunk can contain, browser crashes are expected when attempting to visualize a large Visualization Chunk. By giving the user a configurable option to restrict the visualization of a Chunk with more rows than an arbitary value, we make User Experience much smoother. This improvement is data-agnostic so it is beneficial to all use-cases.

APPENDIX A. LIBRARIES

A.1 Scala Libraries

organization	artifact	version
org.apache.spark	spark-core	2.2.1
org.apache.spark	spark-sql	2.2.1
org.apache.spark	spark-mllib	2.2.1
joda-time	joda-time	2.9.7
org.json4s	json4s-jackson	3.2.11
org.scalatest	scalatest	3.0.5

A.2 JavaScript Libraries

package name	semantic version
actioncable	^5.1.5
babel-preset-react	^6.24.1
colorbrewer	^1.1.0
d3	^3
dc	https://github.com/dc-js/dc.js.git#d06bd9
deck.gl	^5.1.1
lodash	^4.17.5
node-forge	^0.7.5
prop-types	^15.6.0
react	^16.2.0
react-dom	^16.2.0
react-map-gl	^3.2.4
react-redux	^5.0.7
react-split-pane	^0.1.77
react-vis	^1.9.4
redux	^3.7.2
redux-actions	^2.2.1
redux-thunk	^2.2.0
semantic-ui-css	^2.3.0
semantic-ui-react	^0.78.3
babel-eslint	^8.2.2
babel-jest	^22.4.1
eslint	^4.18.1
eslint-plugin-jest	^21.12.2
eslint-plugin-prettier	^2.6.0
eslint-plugin-react	^7.7.0
jest	^22.4.2
prettier	^1.10.2
react-test-renderer	^16.2.0
redux-mock-store	^1.5.1
webpack-dev-server	^2.11.1

A.3 Ruby libraries

gem name	semantic version
rails	~> 5.2.0
sqlite3	
puma	~> 3.11
sass-rails	~> 5.0
uglifier	>= 1.3.0
webpacker	
coffee-rails	~> 4.2
turbolinks	~> 5
jbuilder	~> 2.5
redis	~> 4.0
bcrypt	~> 3.1
bootsnap	>= 1.1.0
haml-rails	~> 1.0
pandoc-ruby	
forgery	
awesome_print	
colorize	
devise	
byebug	
factory_bot_rails	
rspec-rails	~> 3.7
listen	>= 3.0.5 & < 3.2
web-console	>= 3.3.0
spring	
spring-commands-rspec	
spring-watcher-listen	~> 2.0.0
guard	
guard-ctags-bundler	
guard-livereload	
guard-rspec	
guard-shell	
rack-livereload	
rubocop	
capybara	~> 3.0
selenium-webdriver	
chromedriver-helper	
tzinfo-data	

REFERENCES

[1] D. A. Keim, “Visual Exploration of Large Data Sets,” *Communications of the ACM*, vol. 44, no. 8, pp. 38–44, 2001.

[2] P. C. Wong, H. W. Shen, C. R. Johnson, C. Chen, and R. B. Ross, “The top 10 Challenges in Extreme-scale Visual Analytics,” *IEEE Computer Graphics & Applications*, vol. 32, no. 4, pp. 63–67, 2012.

[3] J. D. Fekete, “Visual Analytics Infrastructures: from Data Management to Exploration,” *Computer*, vol. 46, no. 7, pp. 22–29, 2013.

[4] J. J. Thomas and K. A. Cook, *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. IEEE Computer Society, 2005.

[5] K. Daniel, J. Kohlhammer, G. Ellis, and F. Mansman, Eds., *Mastering the Information Age Solving Problems with Visual Analytics*. Eurographics Association, 2010.

[6] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing,” in *Proc. of 9th USENIX Conf. on Networked Systems Design and Implementation (NSDI’12)*, San Jose, CA, 2012.

[7] Home Office, UK, “ASB Incidents, Crime and Outcomes,” 2015, accessed on 30-06-2018. [Online]. Available: https://data.police.uk/about/

[8] D. Keim, G. Andrienko, J. D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, “Visual analytics: Definition, process, and challenges,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4950 LNCS, pp. 154–175, 2008.

[9] B. R. Gaines, “Modeling and forecasting the information sciences,” *Information Sciences*, vol. 57-58, no. C, pp. 3–22, Sep 1991.

[10] Cloudera, “Hue is an open source Analytics Workbench for self service BI.” 2009, accessed on 30-06-2018. [Online]. Available: http://gethue.com/

[11] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran, “Effortless Data Exploration with Zenvisage : An Expressive and Interactive Visual Analytics System,” *Proc. VLDB Endowment*, vol. 10, no. 4, pp. 457–468, November 2016.

[12] V. Dibia and Ç. Demiralp, “Data2Vis: Automatic Generation of Data Visualizations Using Sequence to Sequence Recurrent Neural Networks,” arxiv.org/abs/1804.03126, April 2018.

[13] M. Vartak, S. Huang, T. Siddiqui, S. Madden, and A. Parameswaran, “Towards Visualization Recommendation Systems,” *ACM SIGMOD Record*, vol. 45, no. 4, pp. 34–39, 2017.

[14] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer, “Voyager 2,” in *Proc. of 2017 CHI Conf. on Human Factors in Computing Systems (CHI ’17)*, Denver, CO, May 2017, pp. 2648–2659.

[15] Z. Liu, B. Jiang, and J. Heer, “ImMens: Real-time Visual Querying of Big Data,” *Computer Graphics Forum*, vol. 32, no. 3, pp. 421–430, 2013.

[16] Apache Zeppelin, “Zeppelin: Web-Based Notebook,” 2009, accessed on 30-06-2018. [Online]. Available: https://zeppelin.apache.org

[17] Jupyter Team, “Jupyter Project,” 2009, accessed on 30-06-2018. [Online]. Available: https://jupyter.org

[18] Novus Partners, “NVD3: Reusable Charts for d3.js,” 2014, accessed on 30-06-2018. [Online]. Available: http://nvd3.org

[19] H. Jessica, K. Robert, and L. Heidi, “Finding a clear path: Structuring strategies for visualization sequences,” *Computer Graphics Forum*, vol. 36, no. 3, pp. 365–375, 2017.

[20] Facebook Inc., “React: A JavaScript Library for Building User Interfaces,” 2009, accessed on 30-06-2018. [Online]. Available: https://reactjs.org

[21] P. Domingos, “A Few Useful Things to Know About Machine Learning,” *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.
[22] Uber, “Deck.gl Large-scale WebGL-powered Data Visualization.” [Online]. Available: https://uber.github.io/deck.gl
[23] EUROSTAT, “NUTS - Nomenclature of Territorial Units for Statistics,” 2016, accessed on 30-06-2018. [Online]. Available: http://ec.europa.eu/eurostat/web/nuts/background
[24] Ram Sriharsha, “Magellan: Geospatial Analytics Using Spark,” 2015, accessed on 30-06-2018. [Online]. Available: https://github.com/harsha2010/magellan