On-line Big–Data Processing for Visual Analytics with Argus-Panoptes

Panayiotis I. Vlantis and Alex Delis

University of Athens, Athens, 15703, Greece
{panosv,ad}@di.uoa.gr

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 *Infrastructure as a Service* (*IaaS*) cloud services. 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 undoubtedly present a barrier in implementing responsive applications. In this paper, 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*. 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 spatiotemporal 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 United Kingdom.

Keywords: Visual Analytics · Interactive Programming · Big–Data Processing · Apache Spark · *IaaS* Infrastructures

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 analyses results to be of either limited or no value at all [11]. By providing appropriate tools, *visual analytics* can help users manually interact with datasets, proceed in an highly exploratory manner and shift the focus of the analyses as the occasion calls along the way [19,8,16]. However, the traditional use of visualization techniques on large scale datasets does become prohibitive as the volume of the underlying data grows [3]. 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 [21]. 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 that accompanies working with unfamiliar datasets. It is worth mentioning however that, in our case, 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 paper, 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 [9].

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, Apache Spark is 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.

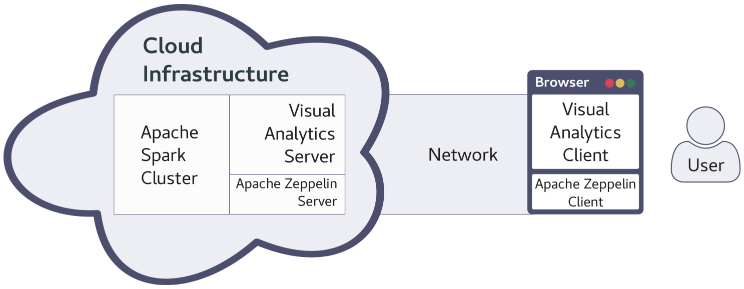


Fig. 1. Argus-Panoptes Architecture for On-line Big–Data Visual Analytics.

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* 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 interface window found in the client-side, allows for the on-line interactive execution of explicit user requests as well as the display of execution outcome which took place in 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 analysts to gain insights and a better understanding through visualization in sophisticated hierarchical datasets.

We have produced a fully-functional prototype, Argus-Panoptes [1], that has served as the means to explore a number of Big–Data use-cases We have published the source code repository [2] to allow further testing with other datasets

[1] Argus-Panoptes is a figure from Greek mythology, it was an ”all-seeing” giant having a watchman role.

[2] Source code repository is available at: https://github.com/panayiotis/visual_analytics

and comparisons with similar tools. In this paper, 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* [9]. The dataset in question maintains spatiotemporal records of incidents since late 2015. The rest of the paper is organized as follows: Section 2 discusses related work and 3 presents the rationale for the design of our architecture. Section 4 outlines the architectural components and their interaction. Section 5 briefly discusses our use-case and Section 6 provides concluding remarks.

2 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 [2,15,4,18,20,12]. Apache Zeppelin [1], Cloudera Hue [2] and Jupyter [10] are open-source initiatives that offer *built-in* visualization functionalities, commonly used in Big–Data exploration. All three 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. Visualization of pertinent results is realized with the help of built-in visualization libraries [13]. 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 off-line phase and examine the large space of possible visualization combinations during the on-line phase[18]. 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 use of *sampling*.

The use of an *RNN* neural-network is advocated in [4] 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 [15], 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 [12]. Similarly to our approach, data binning plays an important role as it is the key technique to attain dimensionality reduction. However, *imMens* overall operations are founded on the concept of *pre-computation* of all data-tiles. This pre-computation occurs in an off-line phase and the respective results are made available at runtime 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.

3 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:

• **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 quick look-up query. In contrast, typical responses in a visual interface are 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.

• **Interactive Programming:** for the user to enjoy the maximum benefit while interacting with our architecture, we introduce interactive notebook systems. Such systems include Zeppelin [1], Jupyter [10] 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 continue 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.

• **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 browser tab and start over. In all above cases, we should stress that the *VA Client* functionality is desired to remain strictly stateless.

• **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 SQLite 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 [3] thus requests for fetching locally existing chunks from *VA Server* are eliminated.

• **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. Such control mechanism is the invalidation of caches at either Spark or *VA Server* components. By adopting such

[3] around $200MBytes$ in total.

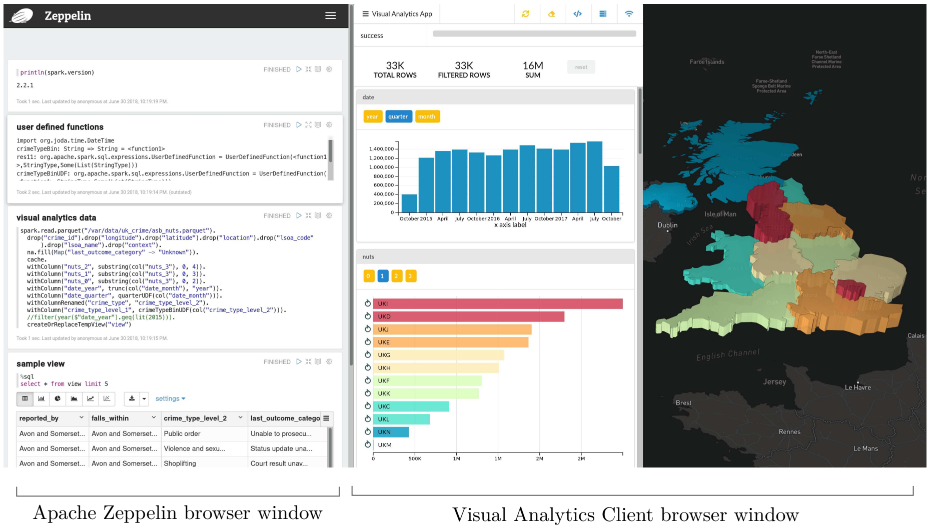
a design choice, we avoid error-prone implementation issues that bear little if any significance to the visual analytics domain. Moreover, we argue that making available Spark/*VA* server real-time status information is helpful to the experienced user.

• **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 [7] 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.

4 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 2 depicts the interface that the *VA Client* of the system realizes. The UI panel consists of two portions: the first is the Zeppelin browser 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 2, the *VA Client* browser window displays chart-based outcomes and generated map-related elements. The latter depicts generated graphs that help demonstrate trends and assist users gain insight with regards to investigated datasets. The functionality of Argus-Panoptes is built around these three concepts which make the architecture feasible: *Schema Convergence*, *Data Binning* and *Visualization Chunks*. In particular:

- ***Schema Convergence*** allows the architecture to be be fault-tolerant when the schema of the examined dataset is being actively manipulated. This mechanism is always utilized when the *VA Server* ships code to Spark for execution. When Spark engine receives a *VA Client* request, it compares the schema embedded in the request with the current schema of the dataset. 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 on the user requests as those are often consistent with a state of the data at an earlier point in time. The data requests instigated from 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 the 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.



```
println(spark.version)
2.2.1
```

Took 1 sec. Last updated by anonymous at June 30 2018, 10:19:19 PM.

user defined functions

```
import org.joda.time.DateTime
crimeTypeBin: String => String = <function1>
res11: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(<function1
>,StringType,Some(List(StringType)))
crimeTypeBinUDF: org.apache.spark.sql.expressions.UserDefinedFunction = UserDefinedFunction(
```

Took 2 sec. Last updated by anonymous at June 30 2018, 10:19:14 PM. (outdated)

visual analytics data

```
spark.read.parquet("/var/data/uk_crime/asb_nuts.parquet").
  drop("crime_id").drop("longitude").drop("latitude").drop("location").drop("lsoa_code"
      ).drop("lsoa_name").drop("context").
  na.fill(Map("last_outcome_category" -> "Unknown")).
  cache.
  withColumn("nuts_2", substring(col("nuts_3"), 0, 4)).
  withColumn("nuts_1", substring(col("nuts_3"), 0, 3)).
  withColumn("nuts_0", substring(col("nuts_3"), 0, 2)).
  withColumn("date_year", trunc(col("date_month"), "year")).
  withColumn("date_quarter", quarterUDF(col("date_month"))).
  withColumnRenamed("crime_type", "crime_type_level_2").
  withColumn("crime_type_level_1", crimeTypeBinUDF(col("crime_type_level_2"))).
  //filter(year($"date_year").geq(lit(2015))).
  createOrReplaceTempView("view")
```

Took 1 sec. Last updated by anonymous at June 30 2018, 10:19:15 PM.

sample view

```
%sql
select * from view limit 5
```

settings

reported_by	falls_within	crime_type_level_2	last_outcome_catego
Avon and Somerset...	Avon and Somerset...	Public order	Unable to prosecu...
Avon and Somerset...	Avon and Somerset...	Violence and sexu...	Status update una...
Avon and Somerset...	Avon and Somerset...	Shoplifting	Court result unav...

Fig. 2. *VA Client* UI.

Such event is an excellent example of a case where users should explicitly initiate a cache invalidation procedure from the *VA Client* to avoid data inconsistencies.

- ***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 spatiotemporal 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 [5].
- ***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` processing is a chunk and its main characteristics may either help improve or adversely impair system performance. Visualization chunks are tracked by the *VA Server* and in this respect, their invalidation, if needed, has to be an 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.

4.1 The Architecture

Fig. 3 outlines the architecture of our *IaaS*–based system: the server side is hosted on virtual computing systems as the left half of the figure shows, while the *VA Client* functionalities are shown to the right hand side. At the core of the server layout, the Spark-engine is referenced as a single entity although it may consist of a whole cloud cluster. It may also involve auxiliary services from the Hadoop ecosystem. In a minimal configuration, the computing cloud consists of an Apache Spark Master service deployed in standalone mode. A more common configuration would involve a Spark Master node, a number of Spark workers and a compatible distributed file system for storing and retrieving data such as HDFS. This example configuration could be extended with the addition of Apache Zookeeper for attaining high-availability as well as YARN or Mesos for for cluster resource management.

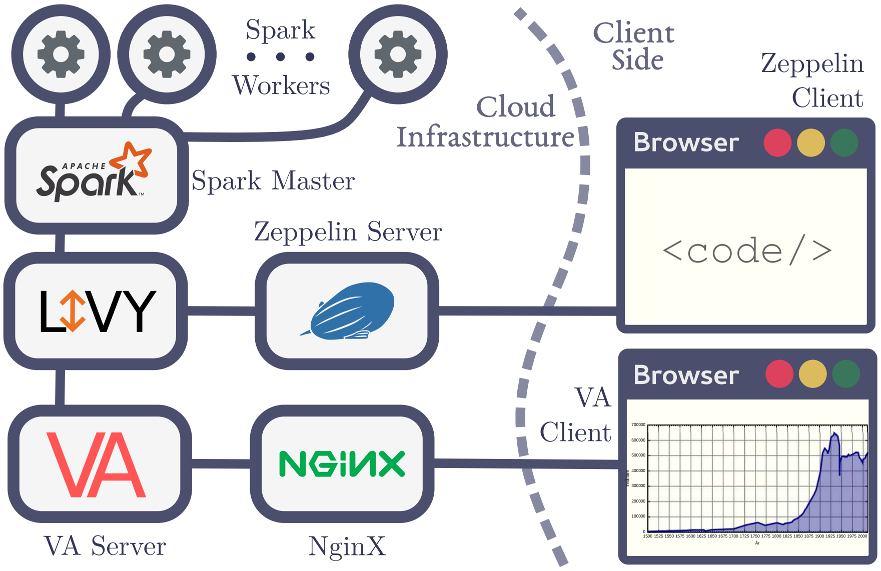


Fig. 3. Argus-Panoptes Architecture Layout

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 an *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 channelling tasks. On the left side of Fig. 2, 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 Apache 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 *VA Client* (Fig. 3). 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. This leads to a non obvious performance improvement: The decompression of json files is handled automatically by a browser thread separate from the one that the JavaScript application is running thus the UI responsiveness is not halted during the decompression process.

4.2 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 component reusability and failure resistance. Fig. 4 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 from quarterly to monthly.

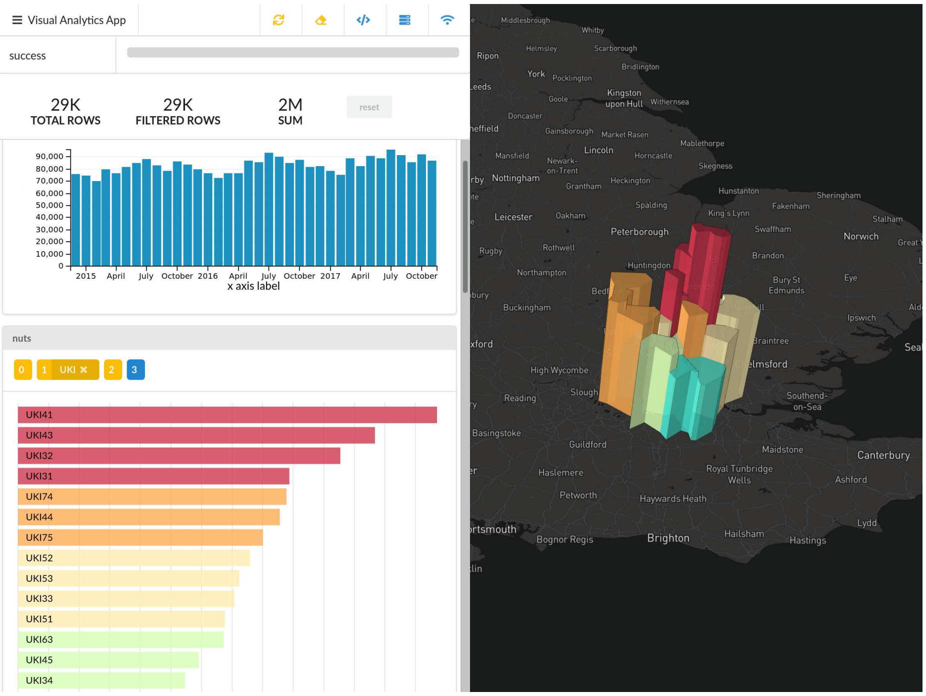


Fig. 4. *VA Client* UI after a Drill-down Operation.

The *VA Client* uses the `Redux` framework as a mechanism of managing the local application state. `Redux` helps the application become independent from prior states. Similarly to the functional programming paradigm, the application interface generated at any point in time 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. 4, every depicted visual element is a React component. Among others, there are components that deal with server connectivity, initiate cache invalidation, and refresh visualizations. There are also React Bar/Row *chart* components that help synthesize complex chart dashboards. The portion of UI to the right of Fig. 4 depicts map-based information and is constructed using a third-party React Map component [17].

5 Assessment with a Government *ASB* Dataset

Salient Argus-Panoptes features evolved during the prototype development. Experimentation with different real-world datasets 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)*** temporal features, 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* [9]. 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 1 shows all features among with the number of distinct and null values for each feature.

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

Table 1. UK AST Dataset Key Characteristics

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

We also transform the UK dataset by joining it with the NUTS classification scheme of Eurostat [6]. 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[14] 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 which has only 178 distinct values, whereas the distinct values of the prior coordinate features were $760K$. (Fig.6) This geo-join operation is CPU-intensive but it occurs only once and so, we make the data persistent for further processing.

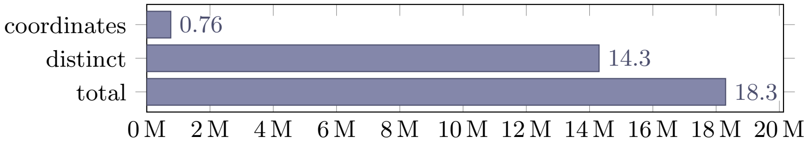


Fig. 5. Dataset has 18.3M rows: 14.3M distinct rows and 760K distinct coordinates.

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. 6 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 is highly desirable as it affords the user to instantly experiment with the introduced hierarchies on-line and if needed, realign them on the spot.

The aforementioned generation of hierarchical dimensions results to a maximum of 24 distinct chunks. Fig. 7 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. 8 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 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 pro-

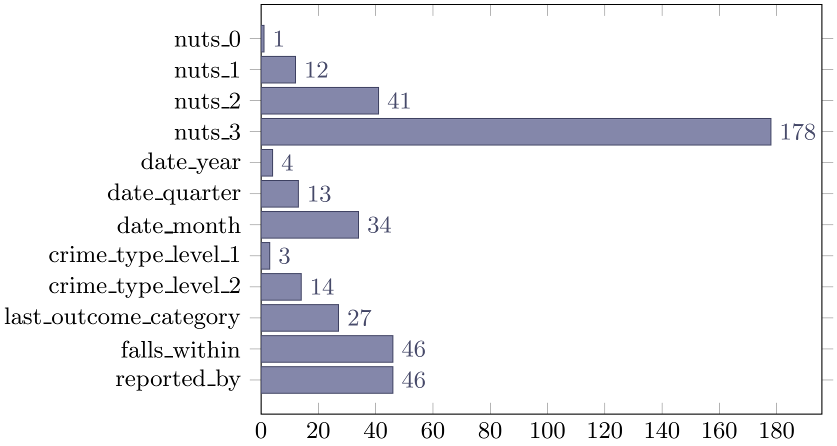


Fig. 6. Distinct counts for every feature resulting from geo-joining and data binning.

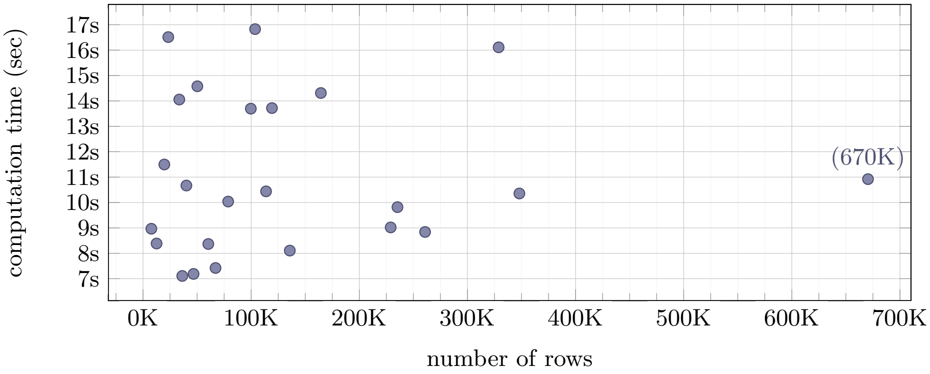


Fig. 7. Visualization chunk computation times in relation to the row count of the aggregated data. Big–Data aggregation operations are taking several seconds to complete and are independent of amount of data returned.

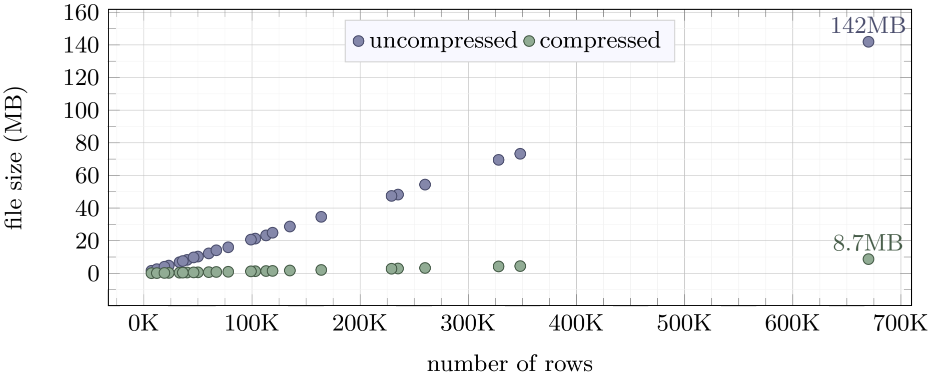


Fig. 8. Chunk file sizes in relation to the row count of the data they contain. The `gzip`-ped `json` files are those that are transfered from the cloud to the browser.

totype. 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*.

6 Concluding Remarks

In this paper, 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 Big–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 spatiotemporal crime dataset from the U.K. Home Office and have ascertained the effectiveness of our prototype through profiling of its operations.

References

1. Apache Zeppelin: Zeppelin: Web-Based Notebook (2009), `https://zeppelin.apache.org`, accessed on 30-06-2018
2. Cloudera: Hue is an open source Analytics Workbench for self service BI. (2009), `http://gethue.com`, accessed on 30-06-2018
3. Daniel, K., Kohlhammer, J., Ellis, G., Mansman, F. (eds.): Mastering the Information Age Solving Problems with Visual Analytics. Eurographics Association (2010)

4. Dibia, V., Demiralp, Ç.: Data2Vis: Automatic Generation of Data Visualizations Using Sequence to Sequence Recurrent Neural Networks. arxiv.org/abs/1804.03126 (April 2018)
5. Domingos, P.: A Few Useful Things to Know About Machine Learning. Communications of the ACM **55**(10), 78–87 (2012)
6. EUROSTAT: NUTS - Nomenclature of Territorial Units for Statistics (2016), `http://ec.europa.eu/eurostat/web/nuts/background`, accessed on 30-06-2018
7. Facebook Inc.: React: A JavaScript Library for Building User Interfaces (2009), `https://reactjs.org`, accessed on 30-06-2018
8. Fekete, J.D.: Visual Analytics Infrastructures: from Data Management to Exploration. Computer **46**(7), 22–29 (2013)
9. Home Office, UK: ASB Incidents, Crime and Outcomes (2015), `https://data.police.uk/about/`, accessed on 30-06-2018
10. Jupyter Team: Jupyter Project (2009), `https://jupyter.org`, accessed on 30-06-2018
11. Keim, D.A.: Visual Exploration of Large Data Sets. Communications of the ACM **44**(8), 38–44 (2001)
12. Liu, Z., Jiang, B., Heer, J.: ImMens: Real-time Visual Querying of Big Data. Computer Graphics Forum **32**(3), 421–430 (2013)
13. Novus Partners: NVD3: Reusable Charts for d3.js (2014), `http://nvd3.org`, accessed on 30-06-2018
14. Ram Sriharsha: Magellan: Geospatial Analytics Using Spark (2015), `https://github.com/harsha2010/magellan`, accessed on 30-06-2018
15. Siddiqui, T., Kim, A., Lee, J., Karahalios, K., Parameswaran, A.: Effortless Data Exploration with Zenvisage : An Expressive and Interactive Visual Analytics System. Proc. VLDB Endowment **10**(4), 457–468 (November 2016)
16. Thomas, J.J., Cook, K.A.: Illuminating the Path: The Research and Development Agenda for Visual Analytics. IEEE Computer Society (2005), `http://vis.pnnl.gov/pdf/RD_Agenda_VisualAnalytics.pdf`
17. Uber: Deck.gl Large-scale WebGL-powered Data Visualization, `https://uber.github.io/deck.gl`
18. Vartak, M., Huang, S., Siddiqui, T., Madden, S., Parameswaran, A.: Towards Visualization Recommendation Systems. ACM SIGMOD Record **45**(4), 34–39 (2017)
19. Wong, P.C., Shen, H.W., Johnson, C.R., Chen, C., Ross, R.B.: The top 10 Challenges in Extreme-scale Visual Analytics. IEEE Computer Graphics & Applications **32**(4), 63–67 (2012)
20. Wongsuphasawat, K., Qu, Z., Moritz, D., Chang, R., Ouk, F., Anand, A., Mackinlay, J., Howe, B., Heer, J.: Voyager 2. In: Proc. of 2017 CHI Conf. on Human Factors in Computing Systems (CHI '17). pp. 2648–2659. Denver, CO (May 2017)
21. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: 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)