STANDARDIZED COMMUNICATION FOR BIGDATA ANALYTICS THROUGH JSON

K.L.K Madushanka

(168244U)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

February 2020

STANDARDIZED COMMUNICATION FOR BIGDATA ANALYTICS THROUGH JSON

K.L.K Madushanka

(168244U)

Dissertation submitted in partial fulfilment of the requirements for the degree Master of Science specializing Data Science

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

February 2020

DECLARATION

I declare that this is my own work and this dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant the University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or another medium. I retain the right to use this content in whole or part in future works.

....

Kasun Madushanka Liyanage Date

I certify that the declaration above by the candidate is true to the best of my knowledge and the above candidate has carried out research for the Masters Dissertation under my supervision.

....

Dr. Amal Shehan Perera Date

Abstract

Big data is not a new terminology in the Information Technology sector anymore. With the emergence of big data, arise the need for analyzing large amounts of data that consist trillions of records. Additionally, big data have already penetrated multiple areas in data analytics. Therefore, different technological solutions were developed to handle these big data complexities. However, even after decades, contemporary solutions are unable to address complex issues and overcome several limitations.

Lack of a common communication standard has resulted in many issues in big data analytics. Presently, all the big data solution companies are using their in-house ad hoc communication methods to perform analytics. Unfortunately, this leads to limitations in integration and reusability of the solutions built. To overcome this, Microsoft introduced the XMLA (XML for Analysis), an industry standard for accessing data in analytical systems, namely OLAP (online analytical processing) systems. XMLA was well standardized and well designed for accessing data through Multi-Dimensional Expressions (MDX). Development of tailor-made query languages to access and analyze the stack of scattered data stores has caused the creation of different standards. This leads to the state where almost all big data services offering their proprietary query languages and APIs for data analysis.

This research is to propose a methodology for addressing the ad-hoc integration of these big data analytics endpoints through a JSON based specification by reusing XMLA structures. The research components are publishing a communication model using JSON specification and proposing to adopt the standards to existing stores. This solution will enable frontend tools to be fully independent of the backend storage model. Also, this will allow existing JSON standardized frontend tools to easily integrate with big data analytics through eliminating the necessity of a specific frontend tool aiming a data store.

Keywords: Big Data Communication, JSON Based Communication, JQA Specification

ACKNOWLEDGEMENT

I would like to take this time to sincerely appreciate the people who helped and guided me in this research. First, Dr. Amal Shehan Perera, my supervisor, for the supervision and guidance provided throughout the research and all the MSc module lecturers for encouraging and motivating me to complete this dissertation.

For Zone24x7.inc and the management for offering me flexibilities and support for the MSC initiation and related work and for my family for always trusting me and supporting me in my educational journey.

I also wish to thank all my colleagues and friends for all their help, support, interest and valuable advice. Finally, I would like to thank all others whose names are not listed particularly but have given their support in many ways and encouraged me to make this a success.

TABLE OF CONTENT

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

Abbreviation	Description
API	Application Programming Interface
ETL	Extract, Transform, and Load
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
MDX	Multidimensional Expressions
NoSQL	Not Only SQL
OLAP	Online Analytical Processing
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSAS	SQL Server Analysis Services
WSP	Web-Service Protocol
WWW	World Wide Web
XML	Extensible Markup Language
XMLA	XML for Analysis
JQA	JSON Queries for Analysis
DaaS	Data As a Service

CHAPTER 1

INTRODUCTION

The introduction chapter will briefly discuss the nature of big data and big data analytics. This chapter will elaborate on how the communication methods work in the current big data analytics and will discuss existing issues in current approaches. Then, Microsoft XMLA standards and OLAP architecture are explored to understand the communication model of XMLA. Finally, a solution is proposed to resolve existing issues in big data analytics.

1.1 Big Data and Analytics

Term big data is normally referred to as the data that is large in size, complex in structure and difficulty in handling. With the rise of IoT (Internet of things) and social media, a vast amount of data is being generated every second. The increased usage of social media networks has immensely contributed to the big data generation. Organizations with low-cost traditional storage devices motivate to store billions and trillions of records in stores. Unfortunately, once the data is collected, it is realized that processing on large volumes of data would be expensive and challenging. Even simple traditional SQL database select queries would take days to complete on such large data volumes.

First of all, storing such a large volume of data into a SQL database is been a problem. Additionally, above storage volume, the complex structure of the data also cause many issues in traditional databases. Unstructured textual data from social media networks have vastly scaled the complexity of big data processing. Nevertheless, with all these complexities, it was a necessity to solve these problems and explore the potentials of big data.

As a revolution, technology giants such as Google, Facebook, and other open source communities stepped into the picture and introduced big data solutions, [1] [2] [3] [4] tackling different problems with different approaches. Later, few promising technological solutions emerged to address the big data complexities and marked a name in the industry. Similarly, NoSQL was quite a buzzword when these solutions initially introduced. Finally, with all these evolutions the industry started to win the battle of handling data volume and the complexities in the big data processing. Once the initial complexities of the big data were stabilized, researchers began to realize the

potentials of big data, which led to the opening of opportunities for applications in many different fields of big data.

Data scientist role became the main authority in charge of performing analytical operations on top of this vast amount of data. Researchers developed different kinds of big data storage methods, algorithms, and tools to fulfil different analytical needs of big data. Furthermore, distinctive ways and methods are used to cover the diverse analytical aspects such as divide and conquer approach to perform analytics in scaled environments [5] [6] and analytics on unstructured textual data using indexes [7].

Collectively, all these approaches enabled opportunities for analytics on large and complex data with acceptable delay, fulfilling most of the need for big data analytics required at that period. Gradually, big data analytics soon spread to almost every sector where data was stored. The industry realized the power of analytics and the benefits of big data analytics [8] [9].

1.2 Big Data Analytics and Communication

In this era of big data, analytics has become the new hype among the industry. Whereas the past industrial approaches were only concerned about storage and retrieval of this vast volume of data, now the companies have visions towards big data analytics including data analytics and business intelligence. A significant amount of effort has been invested to analyze these large volumes of data generated through IoT devices, social media platforms (Facebook, Twitter) and multinational business organizations to mine these data to discover a lot of valuable information, statistics and insights.

With big data capabilities, there is the potential for analyzing a large range of historical data. To fulfil this necessity, various mining techniques have been developed on top of big data. Any insight is worthless unless it is visualized in a proper user-friendly manner and presented to end business consumer. Most of the big data analytics companies have their proprietary dashboards, widgets and pivot grids to accomplish such requirements.

Currently, big data analytics is systematized and has achieved the capabilities to analyze big data in real-time or near real-time with just milliseconds of delay. Moreover, most of these big data solutions have their proprietary query languages and API endpoints to perform different analytical operations with stable implementation and good performance. Regrettably, due to the proprietary on query methods, it is required to utilize different types of queries and requests to perform an identical operation in different data stores. As mentioned above, this is one of the major issues in big data analytics in recent times.

In software engineering, the best practice is to develop a reusable component with less cohesion. Therefore, to implement such standards, system architecture should be designed in a manner where each communication layer is defined with clear interfaces and protocols. Once a system is designed with such clear interfaces, it is very easy to replace and reuse existing components in the system with minimum effort and cost. Unfortunately, there is no standard query language such as SQL or proper communication contract for big data analytics. This research is to focus on designing a system for big data analytics to solve the above mentioned issues. Since most of the big data stores in the industry provide JSON based APIs to perform analytic operations, this research focuses on a more viable solution to resolve current communication issues involved with JSON based specification.

Primarily, the existing SQL based OLAP systems can be designed based on the Microsoft XMLA specification [10] [11]. To start with, the data layer can be separated from the data cube implementation with a proper ETL design. Which facilitates the data sources to be plugged or removed without affecting the data cube implementation. Also, the data cube and front end visualization widgets will be separated through the XMLA endpoints. Only the widget communicates through the XMLA endpoint regardless of the actual cube technology.

Figure 1.1 : OLAP Design with SQL Data Sources Separating each Layer

Figure 1.1 above depicts an overall picture of a proposed OLAP design with SQL data sources separating each layer. As in the figure, with the use of SQL technologies, systems that are less cohesive in each layer can be designed and the components can be replaced in each layer without affecting the other layers.

Even though this can be achieved through a similar design such as an SSAS with OLAP [9], designing less cohesive and reusable layers is still a challenging task. Big data analytics systems can also be separated in the data layers through ETL design, but the execution of data analytics to front end widgets is still tied up with the backend implementation. This is one of the major issues in the big data analytics that restricts the design for a less cohesive system. Each time when a backend data store is changed, all the front end tools and middle-tier components should be replaced due to specific communication implementations.

1.3 Research Problem

Big data is a famous research area among individuals, institutions, and universities in the world. Initially, to get a clear picture of the issues and challenges in the big data industry, a variety of studies were made on different past research. Current trends and technologies of big data industry were studied to identify issues and challenges which have not been reached or solved by anyone in the past [26] [28]. Similarly, the forthcoming phases of the massive use of big data technologies were also analyzed to find the possible future challenges and issues and to identify the challenges which should be addressed with more precaution to avoid difficulties in the future [27] [29]. Whereas, specific problems such as real-time data processing and data communication are major and rising challenges faced by the community due to the increased use of vehicular networks [30]. Information security, scalability, visualization of data, knowledge discovery, computational complexities and data storage and analysis are some of the other commonly discussed challenges [31]. Based on the knowledge gained the significant problem which has not been analyzed prior by anyone was chosen to be addressed in this analysis.

The main problem addressed in this research is that all of the data stores have their visualization tools integrated into it so that it cannot be reused with any other implementation. In recent times, modern databases such as Druid [12] [13] and MongoDB [14] [15] have their pivot and dashboard integration. With such integration approaches, it is unable to replace one of the backend stores without replacing the entire backend, frontend and middle-tier logic. Most of the business intelligence applications come in suites with the package having both front and backend. This causes a lot of limitations because some advanced backend stores come with very limited visualization tools and vice versa. As a result, this limit users from utilizing all the capabilities of one tool due to the limitations in another.

Making a standard endpoint for big data integration will increase the usability and popularity of big data engines since a lot of good backend models have been fallen out from usage due to lack of proper visualization tools. Proper and user-friendly visualizations are very essential from the business customer perspective. If the system

provider fails to offer a great package of both services then the customer tends to seek different solutions. On the other hand, if standards are involved consumers can rely on the standards and make their development based on such standards.

This research attempts to address the existing integration and communication inconsistencies in current big data analytics platforms and frontend tools. Current big data storage models and engines have their designs and platforms for storing, indexing and analyzing petabytes of data. However, all these engines need frontend visualization tools to represent insights. For integration and communication purposes, these big data communities have developed their query languages and APIs. Since there is no standard format or specification for these integrations, no one has attempted to implement a reusable frontend tool. The discrepancies in such systems will be discussed in detail in the Literature Survey section of the next chapter of this study.

1.4 Research Objectives

The main objective of this research is to design a communication specification to standardize the frontend and backend communication of big data analytics. Therefore, a frontend visualization tool adhering the standard is to be developed and facilitating reusability with any backend store. Both frontend and backend communication related implementations should follow the specification contracts once it is finalized. When the objective is achieved, any frontend client can communicate with any backend big data store and vice versa, enabling reusability in both parts.

It would be a great advantage to have the possibility to select the backend and the frontend separately based on the capabilities and requirements and being able to plug them together rather than to be bound to a single suite. This will make a huge impact on the big data technologies and the community since consumers will eventually start contributing to the community on the reusable components based on the standards. Then there will be no necessity for a proprietary company to do complete development from frontend to backend of a solution. Instead, the existing components could be customized and the solutions could be contributed to the community.

1.5 Summary

In this chapter, the background of big data and big data analytics were briefly discussed. Similarly, the communication components of big data analytics tools and the procedures of communication between them were elaborated along with the issues in the existing communication methods. Additionally, existing technologies in big data analytics and communication such as Microsoft XMLA standards and OLAP architecture were explored. Similarly, modern data stores such as Druid and MongoDB were analyzed. The limitations and issues faced by the users due to the drawback incompatibility were identified by examining these existing solutions. Finally, the main research problem to initiate this research was defined and also the objectives of this research to address the given problems were stated along with a proposed solution.

CHAPTER 2

LITERATURE SURVEY

This chapter provides a literature survey for this research explaining a detailed description of related research work. The literature survey is organized according to the following aspects. Initially, the study aids to understand the XMLA and JSON specifications. Similarly, the next section of the literature survey focuses on the areas of related work to investigate the existing integration techniques and to discover if anyone had attempted to implement JSON into big data analytics. Finally, the study chooses the identified big data stores and understands the existing communication behaviors.

2.1 JavaScript Object Notation

JavaScript Object Notation is a lightweight, text-based, language-independent data interchange format [16]. It stores information in a well-structured, organized and easily accessible manner. JSON was developed based on a subset of the JavaScript Programming Language. Since there is no fixed schema, JSON would be very flexible for communication needs [17]. In recent times, a common trend is being seen where XML preferred specifications are moving to text-based data format since JSON is known to be faster and lightweight than XML for communication purposes [18]. Since JSON has a language-independent format, many programming languages and scripting languages including Java, C#, PHP, and JavaScript use JSON as the data transfer medium. With the improvement in World Wide Web (WWW) JSON has a good performance with AJAX, enabling websites to load data quickly and asynchronously, or in the background without any hassle in delay for page rendering or refreshing.

JSON format mainly consists of two structures:

- A collection of name key/value pairs
    - JSON allows declaring any number of properties using a "name": "value" format.
- An ordered list of values
    - List of values or object within square brackets and separated by commas.

```
{
        "code" : "200",
        "message" : "success"
}
```

Figure 2.1 : Simple JSON object with key/value pair

```
[
    "Druid", "MongoDB", "Hadoop", "HBase"
]
```

Figure 2.2 : Simple JSON List

Using simple notations and strings as in figure 2.1 and figure 2.2, JSON can be used to build complex but light-weighted structures for data exchange as in figure 2.3.

```
{
"items":
        {
        "item":
        [
                {
                "id": "0001",
                "type": "donut",
                "name": "Cake",
                "ppu": 0.55,
                "batters": {"batter": [{ "id": "1001", "type": "Regular" },
                        { "id": "1003", "type": "Blueberry" }]
                },
                        "topping":
                        [{“id": "5001", "type": "None”},
                        {“id": "5005", "type": "Sugar”}]

                },
                ...
        ]
        }
}
```

Figure 2.3 : Multi Types JSON Object

Due to the flexibility and the performance concerns, there is a variety of different JSON based communication models namely, protocols, description languages, and API contracts. For example, famous projects such as JSON-RPC, XMPP [19], SOAPjr and JSON-WSP proves that JSON is a good medium for developing specifications.

JSON-RPC (JSON Remote Procedure Call) is a remote procedure call protocol based on XML-RPC, encoded in JSON and it is a very simple protocol with minimum types and commands to be dealt with [20]. SOAPjr is a protocol specification designed based on a hybrid model of Simple Object Access Protocol (SOAP) and JSON-RPC, for exchanging structured information fast in the contract of Web services or AJAX-style APIs [21]. JSON-WSP (JavaScript Object Notation Web-Service Protocol) is a web-service protocol that uses JSON for service description, requests and responses namely, Web Service Definition Language (WSDL). Because of these advantages, JSON enables most of the modern data stores in moving towards JSON based APIs. A few selected data stores and their nature of communication with JSON based models are discussed in the latter of the literature review.

2.2 XML for Analysis and OLAP

XMLA was developed based on SOAP for universal data access of any standard multidimensional data sources exposed over HTTP [10]. This was designed by Microsoft, primarily targeting Microsoft analysis services in the year 2005 [11]. It was not designed just as a data interchange mechanism but also to handle and support metadata management, session management, and locking capabilities.

The XMLA request standard describes two generally accessible methods namely, Discover and Execute. Additionally, each of these structure serves a specific purpose. The ‘Discover’ method is used when a client needs to know about any Metadata information from backend such as data cube name, available dimensions, measures, etc. The ‘Execute’ method allows applications to execute specific queries against the data source and get the result. Originally, XMLA uses MDX as the query language to get the result from data cubes [22]. Figure 2.4 and 2.5 below demonstrates a sample discovery request and sample execute request format respectively.

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Header/>
    <Body>
        <Discover xmlns="urn:schemas-microsoft-com:xml-analysis">
                <RequestType>MDSCHEMA_DIMENSIONS</RequestType>
                <Restrictions>
                        <RestrictionList>
                        <CATALOG_NAME>Adventure Works DW 2008R2</CATALOG_NAME>
                        <CUBE_NAME>Adventure Works</CUBE_NAME>
                        </RestrictionList>
                </Restrictions>
                <Properties>
                        <PropertyList>
                                <Catalog>Adventure Works DW 2008R2</Catalog>
                        </PropertyList>
                </Properties>
        </Discover>
    </Body>
</Envelope>
```

Figure 2.4 : Sample Discovery request [23]

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
    <Header></Header>
    <Body>
            <Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
                    <Command>
                            <Statement> SELECT NON EMPTY
    {[Date].[Calendar],[Date].[Calendar].Children} DIMENSION PROPERTIES
    CHILDREN_CARDINALITY,PARENT_UNIQUE_NAME ON COLUMNS, NON EMPTY
    {[Geography].[City].[All Geographies]} DIMENSION PROPERTIES
    CHILDREN_CARDINALITY, PARENT_UNIQUE_NAME ON ROWS FROM [Adventure
    Works] WHERE ([Measures].[Reseller Freight Cost])</Statement>
                    </Command>
                    <Properties>
                            <PropertyList>
                                    <Catalog>Adventure Works DW 2008R2</Catalog>
                                    <Format>Multidimensional</Format>
                            </PropertyList>
                    </Properties>
            </Execute>
    </Body>
</Envelope>
```

Figure 2.5 : Sample Execute request [23]

2.3 JavaScript Object Notation and Big Data Stores

Since JSON is in full text-based and human-readable format, it makes exchanging data understandable by the receiving party, even without the intervention of the computer. Because of these properties, many analytical tools use JSON for data exchange. However, the structure or the specification of the JSON format depends on the tools being used. The ad-hoc nature of the formats can be visualized when each request and

response from the data store is analyzed. Initially, all the responses are truncated for brevity. Subsequently, this research scopes down on Apache Kylin, Druid and ElasticSearch for a detailed analysis of the existing ad-hoc communication models.

2.3.1 Apache Kylin

Apache Kylin is an open-source Distributed Analytics Engine designed to provide SQL interface and multi-dimensional analysis (OLAP) on Hadoop. Originally contributed from eBay Inc., Apache Kylin supports very large datasets and it is very popular since it supports data cube operations on big data [23] [24]. Apache Kylin supports multiple data source connections such as JDBC, ODBC and provides plugins for Tableau and Excel. Additionally, it also provides rich JSON based REST API for querying purposes.

As seen in figure 2.6 and 2.7 below, REST API consists of Apache Kylin specific query and response format. Due to this propriety communication format, Apache Kylin needs a dedicated frontend tool for visualization.

```
{
    "sql":"select * from TEST_KYLIN_FACT",
    "offset":0,
    "limit":50000,
    "acceptPartial":false,
    "project":"DEFAULT"
}
```

Figure 2.6 : Apache Kylin sample request [24]

```
{
  "columnMetas":[   {
       "isNullable":1,
       "displaySize":0,
       "label":"CAL_DT", …
     },
     {
       "isNullable":1,
       "displaySize":10,
       "label":"LEAF_CATEG_ID",
       "name":"LEAF_CATEG_ID", …
     }
  ],
  "results":[
    [
       "2013-08-07",
       "32996",
       "15",
       "Auction",
       "49.048952730908745",
       "1"
    ], …
  ],
  "cube":"test_kylin_cube_with_slr_desc",
  "affectedRowCount":0, …
}
```


Figure 2.7 : Apache Kylin sample response [24]

2.3.2 Druid

Druid is a high-performance, column-oriented, distributed and open-source big data store designed for real-time exploratory analytics on large data sets. Druid architecture and its advanced indexing structure allow arbitrary exploration of billion-row tables with sub-second latencies [12] [13]. Druid has its proprietary query language and accepts queries as POST requests. The body of the POST request is a JSON object containing various query parameters.

Figure 2.8 and Figure 2.9 below illustrate the current ad-hoc nature of request and response format of the Druid. One of the objectives of this research is to standardize the request and response formats using a JSON specification, including Druid.

```
{
  "queryType": "select",
  "dataSource": "wikipedia",
  "descending": "false",
  "dimensions":[],
  "metrics":[],
  "granularity": "all",
  "intervals": [
    "2013-01-01/2013-01-02"
  ],
  "pagingSpec":{"pagingIdentifiers": {}, "threshold":5}
}
```

Figure 2.8 : Druid sample request [13]

```
[{
  "timestamp" : "2013-01-01T00:00:00.000Z",
  "result" : {
    "pagingIdentifiers" : {
      "wikipedia_2012-12-29T00:00:00.000Z_2013-01-10T08:00:00.000Z_2013-01-
       10T08:13:47.830Z_v9" : 4
    },
    "events" : [ {
      "segmentId":"wikipedia_editstream_2012-12-29T00:00:00.000Z_2013-01-
       10T08:00:00.000Z_2013-01-10T08:13:47.830Z_v9",
      "offset" : 0,
      "event" : {
        "timestamp" : "2013-01-01T00:00:00.000Z",
        "robot" : "1",
        "namespace" : "article",
        "anonymous" : "0",
        "unpatrolled" : "0",
        "page" : "11._korpus_(NOVJ)",
        "language" : "sl",
        "newpage" : "0",
        "user" : "EmausBot",
        "count" : 1.0,
        "added" : 39.0,
        "delta" : 39.0,
        "variation" : 39.0,
        "deleted" : 0.0
      } } …
      } ] }
} ]
```

Figure 2.9 : Druid sample response [13]

2.3.3 ElasticSearch

Elasticsearch is one of the main unstructured text processing service available in the industry. It is available as a package with related components namely, components to read from data sources, components to transform data and components to visualize data. Indeed, Elasticsearch is very flexible to search & analyze data in Real-Time [7]. Additionally, Elasticsearch has its proprietary query language based on JSON. Therefore, any query needs to be requested as a GET or POST request to the exposed endpoint.

By analyzing figure 2.10 and 2.11 below, it is concluded that the request and response format of Elasticsearch is different from Druid or Apache Kylin, discussed previously in this research. Unfortunately, such issues make big data analytics challenging and limited. Once the standard specification for communication is published, these limitations could be avoided and the power of analytics could be explored to a greater extent.

```
{
    "query" : {
      "term" : { "message" : "search" }
    }
}
```

Figure 2.10 : ElasticSearch sample request [7]

```
{
 "matches" : true,
 "explanation" : {
  "value" : 0.15342641,
  "description" : "fieldWeight(message:search in 0), product of:",
  "details" : [ {
   "value" : 1.0,
   "description" : "tf(termFreq(message:search)=1)"
  }, {
   "value" : 0.30685282,
   "description" : "idf(docFreq=1, maxDocs=1)"
  }, {
   "value" : 0.5,
   "description" : "fieldNorm(field=message, doc=0)"
  } ] } }
```

Figure 2.11 : ElasticSearch sample response [7]

2.3.4 Data Services

Data services or Data as a Service (DaaS) is a cloud computing aspect that provides client access to plain data or data analytics and also it is charged as per usage. Data services are one of the trending data access methods because of its scalability and cost-efficiency. User doesn't have to maintain a hardware platform or manage data issues since all the maintenance and data cleaning happens in the cloud. Depending on the cloud agreement user will be charged for the data load or the access frequency [37].

Data services expose access endpoints to third party users. Users can consume data through a provided endpoint regardless of the backend data storage technology. The data service communication model is lightly decoupling. Users can just focus on managing the frontend based on the data services response. This approach won't be a problem as long as the user binds to a single or the same data provider. Developers can reuse the same frontend tools as long as the contract remains. For any reason, if the data service provider or the cloud platform has to be changed, a new vendor contract has to be made. Therefore, the method the data is consumed has to be changed and the approach has to be redesigned [38][39]. This problem happens since there is no standard communication model for data services. Hence, data service providers should use a universal communication format like JQA. Subsequently, if the data service provides the data in the same request-response model, the same research objective can be achieves using the Data Services itself. The proposed solution of the research can be expanded to be used in such cloud services, imposing the standards in every backend, frontend communication methods.

2.4 Summary

In the literature survey, available communication specifications, namely, JSON and XMLA were analyzed to select the most suitable technology for our proposed solution. Related work existing in past were studied to discover whether anyone had attempted to bring JSON into big data analytics. Further, the integration techniques existing in the industry for communication of big data analytics tool were examined. Finally, three big data stores such as Apache Kylin, Druid, and Elasticsearch were selected and each of these big data store technologies was analyzed to understand the core structure and the procedures. This thorough analysis provided a clear knowledge of the existing solution and its drawbacks which supported in determining the necessity of the proposed solution.

CHAPTER 3

METHODOLOGY

Under the Methodology, the approaches to solve the big data analytics problem using JQA is discussed. This section mostly focuses on the JQA communication model, components, syntaxes, and usage. It provides an overview of the architecture and the implementation methods of the proposed solution.

3.1 JSON Queries for Analysis

To solve the problems discussed in previous sections of this research, a properly designed specification for communication is the appropriate solution. In this research, the main focus is to build a JSON based specification to act as the communication contract. All existing major big data stores seem to support JSON as a communication method. Additionally, considering all the benefits of JSON, using JSON to solve this research problem is more viable.

The JSON specification is based on the existing Microsoft XMLA specification design components. As an example, dividing the XMLA request to read the Metadata or to read the actual data in starting tag level is a good design in XMLA. This research inherits such good design approaches in the specification of XMLA. The main challenge is to cover all the features and functions that are required in analytics within the specification. For instance, if the client needs to acquire the distinct count of a given field, there should be a unique way to request for aggregated distinct count without being ambiguous from the normal count.

Beyond the normal analytical operations, the design should support other functional and non-functional aspects such as session handling, error handling and security. Each of this feature is important and should be carefully designed. With the initial specification release of version 1.0, the research will only facilitate the basic features. Other aspects would be covered in subsequent versions of this research over time.

The high-level communication architecture of JQA would be a typical client-server model. The client implements the JQA query generator on its business layer based on the data representation layer, user interactions and events (Bar chart, Click on pie chart slice, Grid, etc.). Then, the generated JQA queries are sent as JSON HTTP request to the server.

Figure 3.1 below depicts the communication specification overview of JQA.

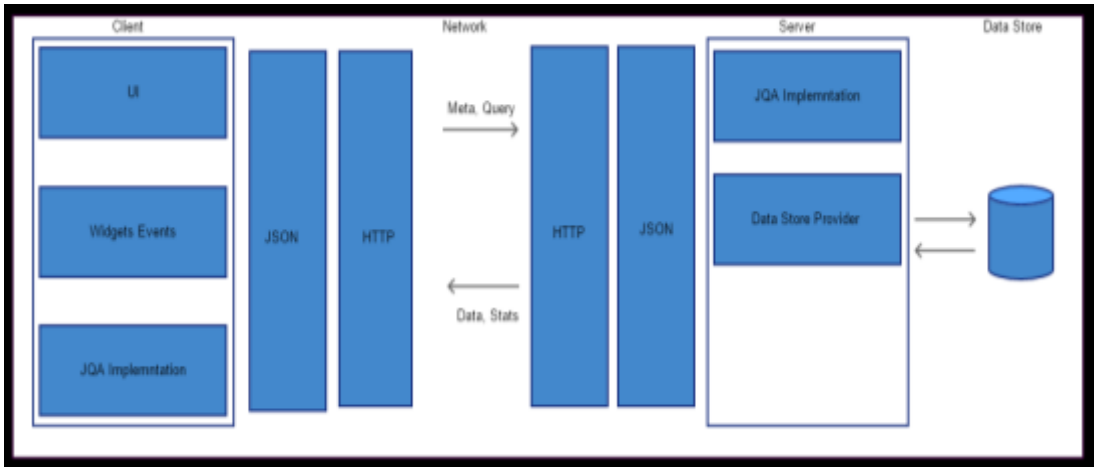

Figure 3.1 : JQA communication overview

Once the server has received the request, it extracts the query and passes it to the data storage provider. Then, the data store provider parses the query and generates raw queries to execute on the data store to get the result. Subsequently, the result is transformed back to satisfy the JQA specification and is passed back to the client as a JSON response. Finally, the client validates and renders the data based on the frontend visualization widget.

Proposed JSON specification is named JQA. It is a simple and light-weighted JSON API, designed specifically for standardizing the data analytical interaction between a client application and the data store. Additionally, it provides universal data access to any data store that is supported by JQA communication model. In general, it is built upon the open Internet standards of HTTP and JSON, and it is not bound to any specific language or technology. Precisely, the main objective is to define a standard to provide reliable data access to any client applications regardless of the underlying dependencies such as operating system, hardware or device.

The primary goals of this specification include the following:

- Provide a standard data access and analyses API to data stores so the same query can be executed in any data store.
- Enforce reusability in front widgets, dashboards and queries.
- Simplify the query model so users can easily develop a query with minimum effort and technical knowledge.
- Support technologically independent implementations using any tool, programming language, technology, hardware platform or device.
- Build on open Internet standards such as JSON and HTTP.
- Use the power of free and open-source community for growth and popularity.
- Work efficiently with standard data stores supporting full functionality.

3.2 JQA Implementation

JQA specification primarily defines three methods namely, Meta, Query and Custom, which define and send JSON for data access and analysis.

Meta section defines how a client needs to write a query to get the meta-information such as datastore name, catalogue details, version information, etc. from the data store. Figure 3.2 below shows a sample JQA store Meta method.

```
{
    "meta": {
            "store": {}
    }
}
```

Figure 3.2 : JQA store Meta method

Query API provides a simple and powerful way to access and perform analytics on the data stores. Query API is mainly divided into four sections namely, SEARCH, AGGREGATIONS, FILTER and OPTIONS. Search API is responsible for getting non-analytic results such as selecting all or a specific set of data columns. It is a typical way of searching for specific data without using any analytical calculations.

Figure 3.3 below illustrates a sample search API list method.

```
{
    "query": {
        "search": {
            "list": {}
        }
    }
}
```

Figure 3.3 : Search API list method

Aggregation API supports standards and analytical functionalities such as COUNT, COUNT_D, SUM, AVG and so on. This is the primary endpoint to perform a simple or complex analytical operation through JQA. Since the specification is fully based on JSON, it is flexible to build queries by removing or nesting JSON properties and objects into the query.

The sample aggregation syntax is as defined in figures 3.4 and 3.5 below. Full JQA specification and related information are publically available in the JQA GitHub repository [25].

```
{
    "query": {
        "aggregation": {
            "function": "<AGGREGATION FUNTION>",
            "field": "<FIELD NAME>",
            "grouping": "<FIELD NAME>",
            "having": {
                "aggregation": {
                    "function": "<AGGREGATION FUNTION>",
                    "field": "<FIELD NAME>",
                    "filters": "<FIELD NAME>"
                }
            }
        }
    }
}
```

Figure 3.4 : Aggregation API Syntax

```
{
    "query": {
            "aggregation": {
                    "function": "count",
                    "field": "<FIELD NAME>"
            }
    }
}
```

Figure 3.5 : Aggregation Count Syntax

3.3 Summary

This chapter thoroughly explained the proposed JQA solution and the main components related to it. The underlying technologies used to build the JQA specification is also explained with reasons in selecting those technologies. Similarly, the high-level architecture of the JQA communication was discussed and JQA implementation methods were explained in detail along with a detailed discussion on JQA syntaxes and relevant usages. Additionally, the primary goals of the specification were also discussed to give an overall knowledge on the proposed solution and to elaborate on the usefulness of the proposed solution.

CHAPTER 4

CASE STUDY, RESULTS AND OBSERVATIONS

The nature of big data applications makes it a challenging task to switch between different technologies with minimum effort. The rapid development of existing and upcoming big data technologies make consistency and maintenance in big data communication much harder. This section focuses on a use case to minimize these problems to derive results based on the observations.

One of the main issues with non-standard big data communication is the massive development effort invested to endeavor. This use case precisely focuses on this issue of massive development effort.

4.1 Case Study

This research has introduced the JSON Queries for Analytics to overcome most of the major communication issues in big data communication spectrum. This big data communication problem was originally raised while developing an actual enterprise big data project. The raised original use case was unable to be used due to software licensing agreements and client data confidentiality.

A different case study was performed to demonstrate the practical usage of JQA in reducing development time and effort when backend data stores keep changing due to requirement complexities. According to the results that have been collected from the case study, the proposed communication standard reduced development effort and time drastically when a need arises to change the big data backend store in the middle of the development life cycle or to change the existing functioning system.

This section further describes the requirements of the intended use case, the practical usage of the JQA, with evidence of the demonstration that proves that JQA reduces development time. This use case also introduces a JQA integrated dashboard to display the analytical summary in the form of simple graphical representations using graphs and charts which helps to generate reports. This is a general requirement for a typical organization namely, Senior Management, BU Heads and Accounts Department and so on. Additionally, this section describes the non-functional and functional requirements and other necessary facts to provide a complete description of the conducted use case study.

The scope of this use case is to identify the usage and benefits of JQA specification in a practical environment and also to demonstrate and explain the specific scenario taken from the software engineering industry. This use case was performed merely to demonstrate the usage of JQA and to show the methods to overcome such communication issues by using JQA and a properly designed dashboard.

4.2 Dashboard, Results and Observations

The dashboard application developed allows displaying the analytical summary of selected datasets in the form of simple graphical representation. Also, provides enhancement requirements to demonstrate usage of JQA in engineering.

The big data analytical dashboard was developed using Angular2, Bootstrap and ChartJS, to facilitate rich visualization charts and graphs. Dashboard charts were implemented adhering the JQA specification for communication. The main idea of the case study is to demonstrate the usage of JQA in big data analytics and highlight key advantages such as ease of use, minimum development time and effort.

Additionally, the dashboard is customizable to add data retrieving queries runtime and to configure data source endpoints dynamically. This isolates the effort and time spent on frontend development when data stores are changed in the backend. This is the achieved extent in solving the existing limitations in big data communication. The backend services use Atmo to deliver the mocked responses to the dashboard requests.

Figure 4.1 below represents the simple dashboard build, including a few widgets. These simple widgets are the basic build blocks of any complex dashboard or graphical report. Here only a limiting number of widgets in a dashboard is shown to cater to the use case requirement.

Figure 4.1 : Dashboard Overview

To build this dashboard a small development team consisting of five members was chosen. The man-hours consumed is represented in Table 4.1 below. The dashboard is developed in a reusable manner. Whereas, all human hours spent on this project by the engineers does not have to be spent again for a new dashboard to support a different backend data source.

Table 4.1 : Frontend development effort in hours

Role	**Effort in hours**
Project Manager	40
Technical Lead Engineer	80
Software Engineer	200
UI Engineer	100
QA Engineer	60

Practically, to setup and configure a big data store with backend development requires considerable human effort and time. Table 4.2 below is a fair industrial estimation for developing a minimal operational druid based big data backend.

Table 4.2 : Backend development effort in hours

Role	**Effort in hours**
Project Manager	40
Technical Lead Engineer	160
Data Science Engineer	400
DevOps	100
Software Engineer	200

These hours are subject to change depending on the use case, the complexity of the requirements, the experience of the developers and the technology stack used. The main argument this use case highlights is that human efforts and time spent is important in software engineering. Also, investing time and effort for building reusable components would be an advantage in the future.

The use case reveals that changing the frontend for a change in the backend store takes very less effort due to two facts namely, the reusable design of the frontend dashboard and the JQA specification. Changing a widget or the entire dashboard for a change in backend takes only a few minutes with the configurations. Since all the queries are written according to the JQA specification, no changes have to be done in query implementation. Similarly, new backend related tasks need to be incorporated and deployed for the backend to be functional. Existing frontend components and query templates can be reused because of the JQA integration. This makes development a whole lot easier for the engineers.

The configurator is designed in such a way that the underlying data source, query or the endpoint is decoupled from the widget. This gives the flexibility of changing one backend endpoint to another without any development aid if the endpoint is following the same communication standards. Similarly, the query to fetch the data from backend could be changed without any development support. Since this dashboard was developed using JQA specification. In total, 480 frontend development hours are saved in with this use case. Considerable backend efforts were saved by skipping the learning efforts and query implementations.

Figure 4.2 below displays widget configurator, the configuration panel for the widget of the dashboard.

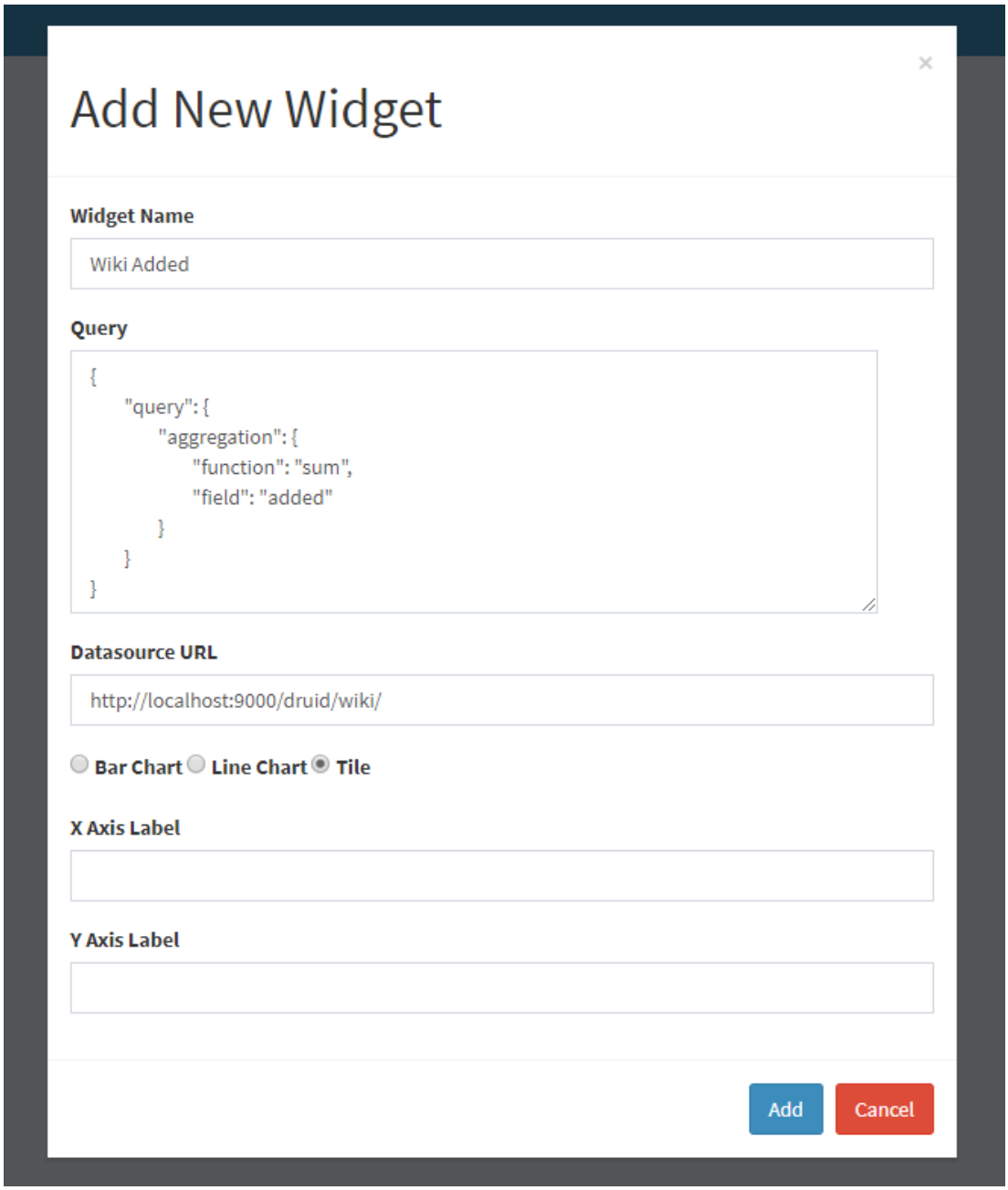

Figure 4.2 : Widget Configurator

The JQA implementation should be integrated into every big data technology communication. Plugging it in separately or having an intermediate proxy will have performance impacts and complications. Since the standards are well defined beforehand accepting it and implementing it to the big data vendor's specific technology is much easier to be done.

4.3 Summary

In this chapter, a case study on the use case performed and the observations received from the use case were discussed. Similarly, the methods used to arrive at the results were explained. Further, this chapter highlighted the drastic effort saved by adhering to the JQA specification. Also, the time and effort saved in using reusable dashboard design over the traditional designs were argued. An overall idea of the final system of the proposed solution is presented with the aid of captures of the built system is presented. This would help the reader to get a better understanding of the easiness and user-friendliness of the system.

CHAPTER 5

CONCLUSION

Originally, when big data phenomena were initiated, technologies such as Hadoop and HDFS were chosen by all, in that period. Those few technologies became the solutions for all the big data problems. But sooner people realized that there are a lot of areas not explored in the big data spectrum. New opportunities, new problems, and new solutions were born when people started analyzing further deeper into big data analytics. Technologies such as Google Cloud Platform [32] [33], Spark Data Processing Engine [34], Flink Stream Processing [35], Beam [36] and Airflow immerged and took control over big data analytics. Communication is an important factor in any big data application. Currently, most of the communication protocols are vendor-based and follow different standards and specifications. This causes many problems for the developers when the requirement arises for changes in the backend stores of the system.

Here in this research, a standard communication named JQA (JSON Queries for Analysis) is introduced to overcome most of the problem faced in the communication protocols. This helps to save time and effort when engineering big data solutions. The case study discussed in this research showed the easiness in switching to a new backend without many hassles when adhering to the proposed solution, a standard communication specification. JQA currently supports only the main communication methods under the initial release of version 1.0 and is planned to be enhanced in future versions with other functional and non-functional aspects such as session handling, error handling, and security.

Initially, the background study about big data, big data analytics and big data communication was made to understand the current technologies available, trends and also the drawbacks in the industry. Then, the difficulties faced in the big data industry by using ad-hoc communication mechanisms were identified and the necessity to standardize communication was proposed along with well-defined objectives to be achieved with the proposed solution. Subsequently, existing research in the big data communication sector was studied and analyzed to gain overall knowledge before initializing the project. Later, the proposed solution was implemented with a development team and the complete methodologies used were elaborated with

examples. Additionally, a sample case study was also presented along with the results and observations from the dashboard built.

5.1 Challenges and Limitation

Big data spectrum is an evolving and ever-changing track. If the growth of the big data industry is carefully observed, the rapid change and improvement it has undergone in the past few decades could be identified. New technologies, tools, and concepts have emerged and have started to dominate the industry. New vendors and new job opportunities have resulted in the immense growth of the big data spectrum. With all the rapid changes and progress it is a very much difficult task to retain all communities into one standard communication.

Since the new trend in the industry is to make money by tweaking open source projects, supporting troubleshoots, modelling for service-oriented payments, vendors prefer to keep proprietary communication standards within the organization to secure the proprietary products and to standardize the market value of the product. Making a universal communication standard may threaten some of the business models of these organizations and that may cause hesitations in the industry to use or develop a standard communication protocol for big data communications. That could be the main reason that so far no individual had tried to develop standards as proposed in this project. That explains the reason for communication standards being ad-hoc until this research even though the big data industry has grown in many areas.

JQA is a specification that defines how both parties should communicate to maintain an industry standard. It is not a built-in driver or any implemented package or module that users can directly integrate into their application. To implement, there is a bit of work involved. Users must implement the request and response handling mechanism defined by the specification standards using a preferred language. But reusable modules and packages could be build that can handle these dined request-response models and could be made available for the community. That is up to the open-source community to decide how to handle the implementation aspects once the JQA specification is publicly available. A small open source project targeting this purpose has been initiated by the development team of this research.

Despite the proposed solution, JQA specification is proved to be beneficial for the industry by saving time and effort of the development life cycle plus built-in packages and modules implemented are being publicly available, it cannot be enforced into any existing solution. There is a lot of hesitations and obstacles to make this research practically a success in the market. Consequently, it has been published under a free and open-source license and is available for anyone to use on their own will.

Additionally, since most of the large scale industries stick to their proprietary solutions and deliver them as a package, companies using solutions of these massive industries tend to use proprietary solutions than adapting to new open source solutions. This isn't an easy task since so much scattered technologies and vendors are still developing proprietary products for big data consumers. It will take a lot of effort to earn publicity to market and popularize such newly published specifications.

5.2 Future Work

This research is the very first attempt to unify the communication standards of all the big data giants in the market. But eventually, big data developers may come to realize that existing ad-hoc communication standards are wasting a lot of development effort and time that could be utilized for any other productive work. At the time when the developer communities are looking for alternative approaches to unify the communications standards having a specification developed and published at the moment may give a starting point and a boost to the communities to try out the JQA specification as a solution for the main issues in big data communication.

Currently, the JQA version 1.0 supports the basic set of methodologies to interact with the backend stores. The specification provides a simplified API to communicate and analyze large data stores with easily designed syntaxes. JQA 1.0 covers all the basic syntaxes to support any big data analytical needs in the current software industry. Even though JQA currently supports most of the analytical and query needs of the users but to promote a new solution to the market it has to cover not only basic functionalities but also should satisfy other non-functional requirements. Therefore, the future versions of JQA are planned to support other functional and non-functional features such as security, session management, advanced error handling and other advanced

methods of statistics and analytics. With step by step version upgrades, each of these functionalities is planned to be covered in the future to roll out a complete industry-standard communication specification.

Presently, an ongoing open-source project has been initiated, named JQA-Dashboard to specifically promote the JQA specification. It is the same dashboard used in the use case discussed in this research. This is a JQA based dashboard supporting pluggable visual components for rich visualization and presentations. This initial repository is made with few basic widgets with limited supports for alterations. The future plan is to provide full flexible widgets so developers can alter the dashboards and widgets easily for their personal needs.

This project is an initiative to support the JQA specification in the industry. This will boost up the usage of JQA and would be a preliminary point for the developers who are looking into standardized communications for big data. Anyone interested in building this community can contribute to the project by developing and integrating rich JQA based components into the online repository [25]. This facilitates the open source community to contribute to this initiative and support and enrich the future of big data communication. This research would contribute to the growth of development of JQA based applications.

REFERENCES

[1] F. Chang et al., “Bigtable,” *ACM Transactions on Computer Systems*, vol. 26, no. 2, pp. 1–26, Jan. 2008.

[2] A. Lakshman and P. Malik, “Cassandra,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, p. 35, 2010.

[3] K. Shvachko et al., “The Hadoop Distributed File System*,” 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[4] “Apache Hadoop,” *Apache Hadoop*. [Online]. Available: http://hadoop.apache.org/. [Accessed: 11-Jan-2018].

[5] S. Ghemawat et al., “The Google file system,” *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, p. 29, Jan. 2003.

[6] J. Dean and S. Ghemawat, “MapReduce,” *Communications of the ACM*, vol. 51, no. 1, p. 107, Jan. 2008.

[7] “Elasticsearch,” *Elastic*. [Online]. Available: https://www.elastic.co/products/elasticsearch. [Accessed: 12-Jan-2018].

[8] H. Chen et al., “Business Intelligence and Analytics: From Big Data to Big Impact,” *MIS Quarterly*, vol. 36, no. 4, pp. 1165–1188, Dec. 2012.

[9] P. Russom, "Big Data Analytics", TDWI Best Practices Report, 2011.

[10] Natalia, “XML for Analysis Specification,” *Microsoft Docs*. [Online]. Available: https://msdn.microsoft.com/en-us/library/ms977626.aspx. [Accessed: 04-Jan-2018].

[11] Archiveddocs, “SQL Server Analysis Services,” *Microsoft Docs*. [Online]. Available: https://technet.microsoft.com/en-us/library/ms175609. [Accessed: 05-Jan-2018].

[12] F. Yang, et al., “Druid,” Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD 14, 2014.

[13] A. S. Foundation, “Interactive Analytics at Scale,” Druid. [Online]. Available: https://druid.apache.org/. [Accessed: 20-Oct-2018].

[14] “The most popular database for modern apps,” *MongoDB*. [Online]. Available: https://www.mongodb.com/. [Accessed: 03-Aug-2018].

[15] “Introducing *JSON*,” JSON. [Online]. Available: https://www.json.org/json-en.html. [Accessed 07-Dec-2018].

[16] D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON),” 2006.

[17] N. Nurseitov et al., “Comparison of JSON and XML Data Interchange Formats: A Case Study”, 2009.

[18] “JSON Encodings for XMPP,” *XMPP*. [Online]. Available: https://xmpp.org/extensions/xep-0295.html. [Accessed: 07-Dec-2018].

[19] “RPC 1.0 Specification (2005),” *JSON*. [Online]. Available: http://json-rpc.org/wiki/specification. [Accessed: 07-Dec-2018].

[20] “SOAPjr - SOAP Junior makes clean, fast AJAX API's using JSON! - specs,” archive.li, 06-Dec-2008. [Online]. Available: http://archive.li/tPAJn. [Accessed: 08-Dec-2018].

[21] G. Spofford, *MDX solutions with Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. Indianapolis, IN: Wiley Pub., 2006.

[22] “PivotGrid Control - Kendo UI with support for jQuery,” *Telerik.com*. [Online]. Available: http://www.telerik.com/kendo-ui/pivotgrid. [Accessed: 12-Jan-2018].

[23] L. Han, “Announcing Kylin: Extreme OLAP Engine for Big Data,” *Tech Blog - eBay Inc.*, 20-Oct-2014. [Online]. Available: http://www.ebaytechblog.com/2014/10/20/announcing-kylin-extreme-olap-engine-for-big-data/. [Accessed: 11-Jan-2018].

[24] “Apache Kylin: OLAP engine for big data,” *Apache Kylin | OLAP engine for big data.* [Online]. Available: http://kylin.apache.org/. [Accessed: 11-Jan-2018].

[25] Kasun, “Kasun88/JQA,” *GitHub*. [Online]. Available: https://github.com/Kasun88/JQA.

[26] T. Kolajo et al., “Trends and Technologies in Big Data Analytics: A Review,” Confluence Journal of Pure and Applied Sciences (CJPAS), vol. 1, no. 1, Nov. 2017.

[27] M. D. Assunção et al., “Big Data computing and clouds: Trends and future directions,” *Journal of Parallel and Distributed Computing*, vol. 79-80, pp. 3–15, 2015.

[28] M. M. Alani et al., *APPLICATIONS OF BIG DATA ANALYTICS: Trends, Issues and Challenges*. S.l.: SPRINGER NATURE, 2018.

[29] A. Londhe and P. P. Rao, “Platforms for big data analytics: Trend towards hybrid era,” *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017.

[30] E. Bourdy et al., “Big Data: An incoming challenge for vehicular ad-hoc networking,” *Internet Technology Letters*, vol. 2, no. 2, Apr. 2018.

[31] A. Rahaman et al., “Challenging tools on Research Issues in Big Data Analytics,” *International Journal of Engineering Development and Research (IJEDR),* vol. 6, no. 1, 2018.

[32] D. Sullivan, “Overview of Google Cloud Platform,” *Google Cloud Certified Associate Cloud Engineer Study Guide*, pp. 1–14, 2019.

[33] I. Shabani and A. K. A. A. Dika, “The Benefits of Using Google Cloud Computing for Developing Distributed Applications,” *Journal of Mathematics and System Science*, vol. 5, no. 4, 2015.

[34] “Apache Spark™ - Unified Analytics Engine for Big Data,” *Apache Spark™*. [Online]. Available: https://spark.apache.org/. [Accessed: 20-Jan-2018].

[35] “Stateful Computations over Data Streams,” *Apache Flink*. [Online]. Available: https://flink.apache.org/. [Accessed: 30-Dec-2019].

[36] “Apache Beam,” *Brand*. [Online]. Available: https://beam.apache.org/. [Accessed: 30-Dec-2019].

[37] S. Kempe, “Data as a Service 101: The Basics and Why They Matter,” *DATAVERSITY*, 20-Nov-2013. [Online]. Available: https://www.dataversity.net/data-as-a-service-101-the-basics-and-why-they-matter/. [Accessed: 24-May-2020].

[38] “Data as a Service,” *MongoDB*. [Online]. Available: https://www.mongodb.com/initiatives/data-as-a-service. [Accessed: 24-May-2020].

[39] “Data Services - Meet the Demand for Faster, More Agile Integration.,” Talend Real-Time Open Source Data Integration Software, 24-Mar-2020. [Online]. Available: https://www.talend.com/solutions/information-technology/service-oriented-architecture/?type=solutionpage. [Accessed: 24-May-2020].