ISSN 1975-8359(Print) / ISSN 2287-4364(Online)
The Transactions of the Korean Institute of Electrical Engineers Vol. 65, No. 7, pp. 1277~1282, 2016
http://dx.doi.org/10.5370/KIEE.2016.65.7.1277

빅데이터 분석을 위해 아파치 스파크를 이용한 원시 데이터 소스에서 데이터 추출

Capturing Data from Untapped Sources using Apache Spark for Big Data Analytics

Aaron Nichie* · 구 흥 서†
(Aaron Nichie · Heung-Seo Koo)

Abstract - The term "Big Data" has been defined to encapsulate a broad spectrum of data sources and data formats. It is often described to be unstructured data due to its properties of variety in data formats. Even though the traditional methods of structuring data in rows and columns have been reinvented into column families, key-value or completely replaced with JSON documents in document-based databases, the fact still remains that data have to be reshaped to conform to certain structure in order to persistently store the data on disc. ETL processes are key in restructuring data. However, ETL processes incur additional processing overhead and also require that data sources are maintained in predefined formats. Consequently, data in certain formats are completely ignored because designing ETL processes to cater for all possible data formats is almost impossible. Potentially, these unconsidered data sources can provide useful insights when incorporated into big data analytics. In this project, using big data solution, Apache Spark, we tapped into other sources of data stored in their raw formats such as various text files, compressed files etc and incorporated the data with persistently stored enterprise data in MongoDB for overall data analytics using MongoDB Aggregation Framework and MapReduce. This significantly differs from the traditional ETL systems in the sense that it is compactible regardless of the data formats at source.

Key Words: Apache Spark, MapReduce, Big Data, MongoDB, Analytics

1. Introduction

The potential to glean useful insight from big data through analytics has gained great momentum among researchers in both the academia and the industry. In a broad range of application areas, huge amount of data is being collected at an unprecedented scale. Among the big data properties of volume, variety and velocity, termed 3Vs of big data, volume is considered to be the first attribute of big data which needs much attention.

The Internet of Things, social-mobile-cloud among others have been named as the main cause for this explosion in data volume. Big data analytics allows researchers, developers, analysts and business organizations to make faster and better decisions based on insights gleaned from the data that was previously inaccessible or unuseable. Decision making was mostly based on guesswork or painstakingly constructed models of reality, But now decision can be made based on the data itself[1]. Every aspect of our modern society is powered by big data analytics including mobile services, retails, manufacturing, financial services to mention a few.

Using advanced big data analytics tools for text analysis, machine learning, data mining, statistics and natural language processing. Businesses can analyze previously untapped data sources independently or together with enterprise data to gain new insights resulting in significantly better and faster decision making [2].

To realize the outmost benefits from data analysis, there is an imperative need to fulfil especially in the era of big data. Fast-on-demand processing of data and effective persistent storage of data are two main hard-core issues which require maximum attention of data scientists. This is the era where problems related to data processing are interesting to solve. The ability of a developer to leverage available technologies and factor big data into integrated data processing solution remains a daunting task and requires enormous skills[3].

Although data sources may be of unstructured formats, they are required to be of a certain structure for persistent storage. ETL processes are key in restructuring data. However,

† Corresponding Author : Dept. of Computer and Information Engineering, Cheong-Ju Univ., Korea
E-mail: hskoo@cju.ac.kr
* Dept. of Computer and Information Engineering, Cheong-Ju Univ., Korea
Received : May 15 2016; Accepted: June. 20 2016

ETL processes incur additional processing overhead and also require that data are maintained in predefined formats at the source. Consequently, data in inconsistent formats are completely ignored by the ETL processes because designing ETL processes to cater for all possible data formats is almost impossible. Data from unstructured data sources are often ignored, and justifiably so, in an analytics process for two reasons. First, because they are not always consistent with already deployed ETL systems in a production environment and frankly it is impossible to design an ETL process that would encapsulate all data structures and formats. Second, prior to the development of already deployed ETL systems such technologies were unavailable. Now that technological tools such as Apache Spark are available, developers either lack the requisite technical know-how or do not find it necessary to change the existing processes to incorporate these data sources into their analytics solutions[4].

Potentially, these unconsidered data sources can provide useful insights when incorporated into big data analytics. Time series facts reflecting the history of data changes are more reliable and valuable than the only current state these historical changes may end up in text file which are likely to be ignored[5]. For instance, a social networking website keeps the history of “adding” and “removing friend” events for all the users. This raw data carries more information than the current friend list of any user. In fact, the current state of friend list can be derived by aggregating all these “adding” and “removing friends” events. One may argue that executing the aggregate function to derive a friend list is slow and costly, an option is to create a snapshot of the friend list during certain time interval and save it as a materialized view, which can be merged with the new events created to generate the up-to-data friend list[6].

The example given above underscores the importance of historical data which maybe scattered around in pieces but have a huge potential of providing useful insights when brought together and then analyzed. In this project, using big data solution, Apache Spark, we tapped into other sources of data stored in their raw formats such as various text files, compressed files and incorporated the data with persistently stored enterprise data in MongoDB. Overall data analytics was then carried out using MongoDB Aggregation Framework and MapReduce. The objective was not only to establish the fact that often ignored data sources could be beneficial to gleaning useful insights when incorporated into data analytics processes but also to ascertain which of the two analytics systems, MongoDB Aggregation Framework and MapReduce, that ships with MongoDB database engine is the most efficient interface to use as part of a big data processing solution.

2. Related Work

Using Spark’s DataFrames to explore MongoDB through spark-mongodb package from Stratio[7]. The paper used SQL and DataFrames Library from Apache Spark to represent MongoDB collection as a DataFrame. That involved a configuration object being created, an SQL parser object, and then DataFrame representing the collection in 1-minute windows to read from MongoDB which was called 1-minute bars. An SQL sub-query was execute on the 1-minute bars in the table minbars and then a query was executed to aggregate the results to create 5-minute bars, the result was saved in a DataFrame which was finally written to MongoDB. Thus, in the paper, MongoDB was used both as source and destination through the Apache Spark SQL and DataFrames Library.

Collecting data in operational systems and then relying on nightly batch-Extract, Transform and Load processes to update the Enterprise Data Warehouse is no longer sufficient. The paper questioned the efficiency of traditional ETL processes[8]. Although MongoDB natively offers rich analytics capabilities, the paper extended MongoDB’s analytics capacities with Apache Spark. The paper presented some circumstances where integrating the Spark engine could improve the real-time processing of operational data managed by MongoDB, and allow users to operationalize results generated from Spark within real-time to support business processes. By taking advantage of MongoDB’s rich secondary indexes to extract and process only the range of data it needs, Spark was used to analyse all customers located in a specific geographical areas.

This was a project that created a staging environment to mirror a production environment. The aim was to reveal performance issues of test codes that were not evident in development environment. Code changes were taken to staging, where data was pulled from MMS to determine if feature X would impact performance[9]. The test focused on Map/Reduce and the MongoDB aggregation framework to compare performance and to what extent would the overall MongoDB system will be impacted.

3. System Design and implementation

In this project, we used Apache Spark to access raw, unstructured, data stored in various text formats both compressed and uncompressed via the local file system. The tapped data were formatted and persistently stored in MongoDB together with existing enterprise data for further

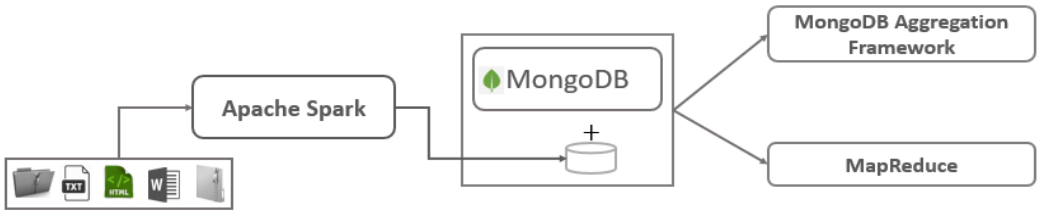

Fig. 1 System General Overview

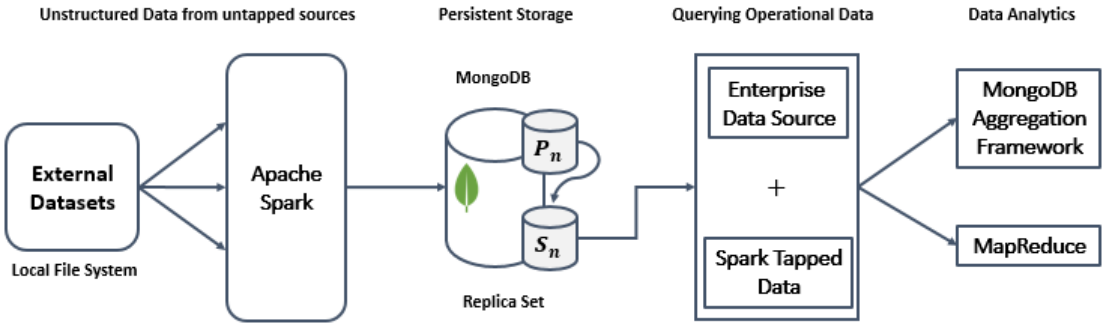

Fig. 2 System Implementation

data analytics using Aggregation Pipeline and MapReduce. Fig. 1 provides a broad overview of the system.

The overview shown in Fig. 1 was then translated into technical implementation with Python being the language of implementation. Analytics done with MongoDB Aggregation Framework and MapReduce were both implemented in Python via MongoDB Python API, PyMongo, while Spark programming was done using PySpark. Fig. 2 provides a detailed system implementation.

3.1 Tapping Data from Unstructured Data Sources

With all the target raw data stored on the local system, Spark was used to access the data by creating *Resilient Distributed Datasets (RDD),* the basic abstraction in Spark. With PySpark, it is possible to create RDD from any storage source supported by Hadoop, including local file systems, HDFS, Cassandra etc. Spark's Python API also supports text files, SequenceFiles and other file formats. A text file RDD was created using *SparkContext.wholeTextFiles* method [10]. The method allowed reading a directory containing multiple small text files, and returned each of them as filename-content pairs. For some other files, text file RDDs were created using *textFile("/path/to/textfile/*.txt ")* method. The URI for the files: a local path on the local machine was supplied to the method and read it as a collection of lines. The data was thus captured and saved persistently in MongoDB database discussed next.

3.2 Saving the Captured Data

PySpark provides a means to save data persistently to any database system that it is interfaced with through connectors. Two connectors are available to integrate MongoDB with Spark: MongoDB Connector for Hadoop and Spark and the Spark-MongoDB Connector, developed by Stratio, a Big Data start-up. In this project, the MongoDB Connector for Hadoop and Spark was used. The connector is a plugin for both Hadoop and Spark that provides the ability to use MongoDB as an input source and/or an output destination for jobs running in both environments. The connector directly integrated Spark with MongoDB and had no complex dependencies. The *core jar* and a valid *OSGi* bundle: an *uber jar* containing bson library, the core library and MongoDB Java Driver, were the only two dependencies for installing the connector.

The method *saveAsNewAPIHadoopFile(params)* helped to output the Python RDD of key-value pairs of the form *RDD([k,v])* to MongoDB using the new Hadoop Output Format API. The method takes at most seven parameters, however not all of them were needed to perform the operation. Inputs such as *inputFormatClass, keyClass, valueClass* and *conf* were basically needed.

3.3 Creating Replica Sets

By simulating a typical production, there was the need to create replica sets to isolate the Analytics from operational workloads. In this case the operational workloads were spark processes which brought in huge amount of data classified as unstructured data. The data were pumped to the primary replica set labelled P_n in Fig. 2 while analytics took place at the secondary replica sets labelled S_n. In a fully-fletched production environment the primary P_n will have to be sharded in order to distribute the weight, so to speak, and aid in faster query throughput but there was no need to shard the replica sets just to keep the systems configurations simple. Integration of the data pumped in by spark and the enterprise data took place at the primary replica set and were then replicated to the secondary set. Analytics were done on the secondary replica set as discussed next.

3.4 Performing the Analytics

Analytics tools used were MongoDB's Aggregation Framework and MapReduce via the python API. Analytics included but not limited to filtering data that satisfied the criteria. For instance, Since data were social media reviews about products, it had country of origin, the product's name or the name of the company the manufactured the product. First, data filtering was done to obtain data that contained certain specific words such as names of companies or name of products those companies manufactured, data were then grouped by countries of origin and then were sub-accumulated by the various social media portals through which they were posted. Fig. 3 provides an implementation flowchart and further details about the analytical processes. Three companies: Apple, Samsung and LG were featured in this project. We monitored the elapsed time of both analytics solution which served as a performance yardstick between MongoDB's Aggregation Framework and MapReduce. The processes are discussed next.

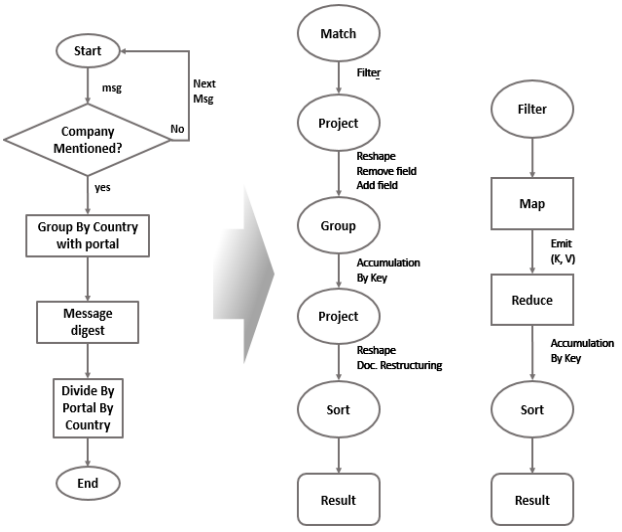

Fig. 3 Flow Chart, MongoDB Aggregation Pipeline & MapReduce Phases

4. Experiment Setup and Results

Using review comments collected from various social media networks, Facebook, Twitter handles and others provided freely on the internet as html blog posts, text files etc for public downloads. Data size ranged from one million to thirty million review comments about a product from selected companies: LG, Samsung and Apple.

Beginning the tests, one million test data; integrated enterprise and Spark-tapped data, each analytics solution was executed and elapsed time record. The test was repeated with data size increased from one million to five, ten and thirty million in succession. Table 1 provides details about the test environments, computer specifications, operating systems, MongoDB and Spark versions and configurations.

4.1 Results

Fig. 4. presents the result: the elapsed time from the two analytic solutions, MongoDB Aggregation Framework and MapReduce. Table 2 shows a snapshot of the output from the two solutions. Origin is the country or state code where the review was originally posted, the portal through which the comments were posted and the analyzed data size

Table 1 System Technical Specifications

Computer System
Operating System: Windows 10 Home 64-bits(10.0, Build 10586) Processor: Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz (4CPUs), ~3.3GHz Memory: 8192MB RAM Drive Type: Solid State Drive(SSD)
Database System and Add-Ons
MongoDB for Windows 64-bit Version 3.2.3 Default Engine: WiredTiger Apache Spark PySpark Version 1.5.2 Pre-built for Hadoop 2.4

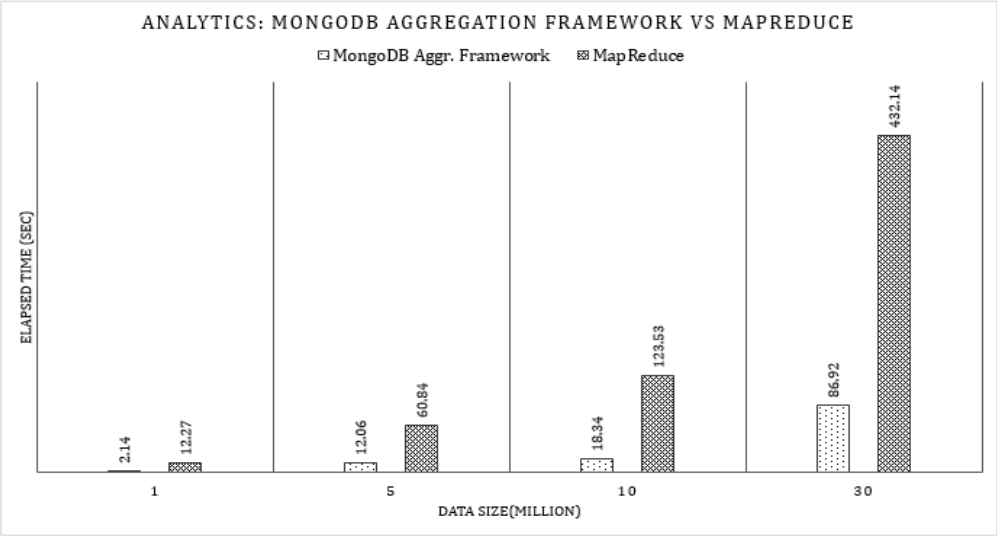

Fig. 4 MongoDB Aggregation Framework vs. MapReduce

grouped according to the portals. From the table it can be seen that the results are same but the processes through the results were obtained were entirely different: MongoDB Aggregation Framework and MapReduce.

4.2 Discussion of Results

In this section, experiment results are discussed and analyzed; results are compared with known theoretical facts to confirm the validity or otherwise of the results. Conclusions are then drawn in the next chapter. First, considering Fig. 4, MongoDB Aggregation Framework vs. MongoDB MapReduce. MongoDB Aggregation Pipeline showed an incredible performance which was very difficult for the MongoDB MapReduce process to compare. At the maximum data size of thirty million, the performance difference was over 140 percent. The average performance percentage difference between the two solutions was approximately 140%. The same results but different approaches. From Fig. 3 aggregation pipeline had an extra stage compared to MapReduce but the aggregation pipeline was impressively faster in analysing same amount of data. For most analytic operations, aggregation pipeline provides better performance and more coherent interface[11]. Aggregation pipeline provides all functionalities needed for data analytics with MongoDB. MongoDB acknowledges the performance supremacy that the aggregation pipeline has over the MapReduce interface. The aggregation framework provides better performance and more coherent interface. However, map-reduce operations provide some flexibility that is not presently available in the aggregation pipeline[12].

Table 2 Analyzed Result Output

Aggregation Pipeline		
Origin	Portal	Analyzed Data Size
Kr	Facebook	1120
Kr	Kakao	748231
Kr	Twitter	2296
Kr	Flat-File	1277
Kr	Whatapp	554
MapReduce		
Origin	Portal	Analyzed Data Size
Kr	Kakao	748231.0
Kr	Facebook	1120.0
Kr	Flat-File	1277.0
Kr	Twitter	2296.0
Kr	Whatapp	554.0

5. Conclusion

With the average percentage difference between the two solutions approximately being 140 percent, it shows that MongoDB Aggregation Framework analytical solution was 140 percent much more efficient than the MapReduce alternative. Our experiment result confirm MongoDB's assertion that the aggregation framework is superior to MapReduce. MapReduce's operational advantage is that MapReduce operations provide additional flexibility that is not presently available in Aggregation Pipeline; at the map phase more than one key and value mapping can be created. Furthermore, map-reduce operations can make final

modification to the results at the end of the operation, such as perform additional calculation using the finalize phase, an extended addition to MapReduce operation. Such operation would require two separate pipelines in order to replicate that on the Aggregation Framework. The first pipeline saves its results in MongoDB while the second pipeline accesses it to perform additional operation. This two-way pipelining could be slow with respect to huge volume of data and also not forgetting the fact that the intermediate result is also restricted to 16M BSON document size limitation.

Modern big data tools have made it possible to capture data from sources which hitherto were completely ignored or not encapsulated into ETL processes because of sheer huge diversity in data formats. Successfully, we have been able to tap into such data sources in their raw formats and incorporated them into data analytical process while avoiding the use traditional ETL processes. This was made possible by Spark because it is able to capture a broad spectrum of diverse data and file formats. This broke us free from the restrictions imposed on data scientists by traditional ETL systems thus giving us the leverage to incorporate other data sources, thus providing access to much big data to glean insights from.

References

[1] D. Agrawal, P. Bernstein, E. Bertino, S. Davidson, U. Dayal, M. Franklin, J. Widom, "Challenges and Opportunities with Big Data: A white paper prepared for the Computing Community Consortium committee of the Computing Research Association". pp. 1-17, Nov.2012. [Online]. Available:http://cra.org/ccc/resou rces/ccc-led -whitepapers/Downloaded: Feb. 14, 2016.

[2] Bringing Big Data to The Enterprise. [Online]. Available: http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html accessed on Feb. 05, 2016.

[3] P. Nathan, "Intro to Apache Spark", Chicago International Software Conference 2015. pp. 1-188, May 14, 2015. [Online]. Available: http://training.dat abricks.com/ workshop/sparkcamp.pdf Downloaded: Ja n. 03, 2016.

[4] L. Neal, "Will NoSQL Databases Live Up to Their Promise?", Technology News, IEEE Computer Soci- ety, pp. 12-14, Oct. 2010.

[5] Big Data Analytics. [Online]. Available: http://www - 01.ibm.com/software/data/infosphere/doop/what-is-big-data-analytics.html Accessed: Feb. 14, 2016.

[6] A. Hafiz, O. Lukumon, B. Muhammad, A. Olugbenga, O. Hakeem, A. Saheed, "Bankruptcy Prediction of Construction Businesses: Towards a Big Data Analytics Approach", IEEE Conf. Pub., pp.1-5, Mar. 09, 2015.

[7] M. Kalan, "Tutorial for Operationalizing Spark with MongoDB", [Online]. Available: https://www.mongodb. com/blog/post/tutorial-for-operationalizing-spark-with -mongodb Accessed Dec. 12, 2015.

[8] MongoDB, "Apache Spark and MongoDB Turning Analytics into Real-Time Action", A MongoDB White Paper, Aug. 2015.

[9] QAing New Code with MMS: Map/Reduce vs. Aggregation Framework, available at http://blog.mong odb.org/post/ 62900213496/qaing-new-code-with-mms-mapreduce-vs accessed on Mar. 01, 2016.

[10] How Apache Spark Is Transforming Big Data Processing, Development. [Online]. Available: http:// www.eweek.com/enterprise-apps/how-apache-spark-is -transforming-big-data-processing-development.html Accessed: Feb. 16, 2016.

저 자 소 개

Aaron Nichie

Graduated from University of Ghana, Legon with B.S. degree in Computer Engineering in 2012. He is currently pursuing his M.S. degree at Cheong-Ju University, Korea. His research interests include Internet of Things (IoT), Data Mining, Big Data, Data Analytics and Data Engineering.

E-mail: nichieaaron@gmail.com

구 흥 서 (Heung-Seo Koo)

Reference to the Transactions of The KIEE Vol.65, No.3, 2016.