Journal of Theoretical and Applied Information Technology
28th February 2018. Vol.96. No 4

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

DECLARATIVE STACK FOR DISTRIBUTED GRAPH PROCESSING

[1]RADWA ELSHAWI, [2]ARWA ALDHABAAN, [3]SHERIF SAKR

[1,2]Princess Nourah bint Abdulrahman University
Riyadh, Saudi Arabia
[3] King Saud bin Abdulaziz University for Health Sciences
[1]rmelshawi@pnu.edu.sa

ABSTRACT

Recently, people, devices, processes and other entities have been more connected than at any other point in history. In general, graphs have been used to represent data sets in various application domains including computational biology, social science, telecommunications, astronomy, semantic web and protein networks among many others. In practice, systemsstacks of large scale graph processing platforms are suffering from the lack of declarative processing interface. They are mainly relying on low level programming abstractions which can be only used by sophisticated software developers and are not adequate for many users. In order to tackle this challenge and improve the performance and user acceptance of large scale graph processing frameworks, we present a declarative querying framework that can seamlessly integrate with various big graph processing system platforms. Our experimental evaluation shows the effectiveness and efficiency of our proposed framework.

Keywords: *Big Data, Big Graph, Hadoop, Spark*

1. INTRODUCTION

The ubiquity of the Internet has dramatically changed the size, speed and nature of the generated data. Almost every human becomes a data generator and every business becames a digital business. Thus, we are witnessing a data explosion. In the last years, several technologies have contributed to this data explosion including mobile computing, Web 2.0, social media, social network, cloud computing and Software-as-a-Service (SaaS). In the future, it is expected that the Internet of Things will further amplify this challenge. Several things would be able to get connected to the Internet, and thus there will be lots of data passed from users to devices, to servers, and back. Hence, in addition to the billions of people who are currently using the Internet and daily producing lots of data, watches, cars, fridges, toaster, and many other devices will be online and continuously generating data as well. It is quite expected that in the near future, our toasters will be able to recommend types of bread based on suggested information from our friends on the Social Networks.

With the recent emerging wave of technologies and applications, the world has becoming more connected than ever. Graph is a popular and neat data structure which is used to model the data as an arbitrary set of objects (vertices) connected by various kinds of relationships (edges). With the tremendous increase on the size of the graph-structured data, large-scale graph processing systems have been crucially on-demand and attracted a lot of interest. In the last few years, several specialized platforms which are designed to serve the unique processing requirements of distributed large-scale graph processing have been introduced (e.g, Google Pregel [1], Apache Hama [2], Apache Giraph [3], GraphLab [4], PowerGraph [5] and Microsoft Trinity [6]). These systems provide low-level programmatic abstractions for performing iterative parallel computations, querying and analysis of large graphs on clustered systems. In practice, many programmers and data scientists prefer to express their analytic jobs declaratively. For example, in the early days of the Hadoop framework, the defacto standard in the domain of big data processing, the lack of declarative languages to express the large-scale data processing tasks has limited its practicality and the wide acceptance and the usage of the framework. Therefore, several declarative querying systems on top of the Hadoop framework (e.g., Pig [7], Hive [8]) have been introduced to fill this gap and provide higher-level languages for expressing large scale data analysis tasks on Hadoop. In practice, these languages have been widely adopted in the business and research communities. Currently the systems/stacks of large scale graph processing platforms are suffering from the same challenge. The aim of our work is to provide

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

higher-level languages for expressing large scale data analysis tasks in the domain of distributed and large-scale graph processing. In particular, we present a declarative query processing framework on top of the emerging distributed graph processing platform. The proposed framework relies on a declarative graph-based query language, Cypher, a metadata-based catalog for distributed graphs and cost-based query compiler that compiles the declarative graph-based queries and computation generates efficient execution plans using the low-level programming abstractions of the underlying distributed graph processing platforms.

BIG GRAPH PROCESSIG

Recently, people, devices, processes and other entities have been more connected than at any other point in history. In general, the complex relationships, interactions and interdependencies between objects are naturally modeled as graphs. Therefore, graphs have been used to represent data sets in various application domains including computational biology, social science, telecommunications, astronomy, semantic web, protein networks, and many more [3]. For example, in a social graph, nodes are commonly used to represent people while the friendship relationships among them are modelled via edges.

In principle, graph analytics is considered as one of the most important big data discovery tool [3]. For example, it enables inspecting fraud operations in a complex interaction network, identifying influential persons in a social network and recognizing product affinities by analyzing community buying patterns. In practice, nowadays, graphs with millions and billions of nodes and edges have become very common. For example, in 2012, Facebook has reported that its social network graph contains more than a billion users (nodes) and more than 140 billion friendship relationships (edges) [3]. The continuous growth in the size of the graph datasets requires scalable computing resources to achieve the goal of effectively analyzing and utilizing them. In general, one of the most important challenges in processing large scale graphs, in addition to their size, is the inherent irregular structure and the iterative nature of graph processing and computation algorithms.

In practice, the popular MapReduce framework [9] and its open source realization, the Apache Hadoop project [10], together associated with its ecosystem (e.g., Apache Pig [7], Apache Hive [8]) has represented the pervasive technology for big data processing platforms [11]. In principle, the MapReduce framework provides a simple but powerful programming model that supports the developers to easily build parallel and scalable algorithms to analyze massive amounts of data on clusters of commodity machines. However, the MapReduce programming model has its own limitations [12]. For example, it does not provide a direct support for iterative data analysis (or equivalently, recursive) tasks. Instead, users need to design iterative jobs by manually chaining various MapReduce tasks and orchestrating their execution via a controller program [12].

Generally, graph processing algorithms are iterative and need to traverse the graph in some way (e.g., breadth first or depth first). In practice, graph algorithms can be mapped into a sequence of lined MapReduce jobs where the whole state of the graph get passed from one task to the next. However, such approach is not adequate for graph processing and commonly leads to inefficient performance because of the overhead on the communication costs which is also associated with the serialization overhead in addition to the need of coordinating the steps of a chained MapReduce. Several approaches have proposed Hadoop extensions (e.g., HaLoop [13], Twister [14], iMapReduce [15]) to optimize the iterative support of the MapReduce framework and other approaches have attempted to implement graph processing operations on top of the MapReduce framework (e.g. Surfer [16], PEGASUS [17]). However, these approaches remain inefficient for the graph processing case because the efficiency of graph computations depends heavily on inter-processor bandwidth as graph structures are sent over the network after each iteration. While much data might be unaltered from one iteration to another, the data must be reloaded and reprocessed at each iteration, resulting in the unnecessary wastage of I/O, network bandwidth, and processing power. In addition, the ending condition might involve the detection of when a fix point is reached. The condition itself might need to define an extra MapReduce task for each iteration which consequently increases the resource usage in terms of scheduling additional tasks, reading additional data from disk, and transmitting additional data through the network.

In order to tackle the inherent performance problem of the MapReduce framework, several specialized platforms which are designed to serve the unique processing requirements of large scale graph processing have recently emerged. These

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

systems provide programmatic abstractions for performing iterative parallel analysis of large graphs on clustered systems. In particular, in 2010, Google has pioneered this area by introducing the Pregel [1] system as a scalable platform for implementing graph algorithms. Since then, we have been witnessing the development of a large number of scalable graph processing platforms. For example, the Pregel system has been cloned by various open source projects such as Apache Giraph [3] and Apache Hama [2]. Pregel system has also been further optimized by other systems such as Pregelix [18], Mizan [20] and GPS [19]. In addition, a family of related systems has been initiated by the GraphLab system [4] as an open source project at Carnegie Mellon University. Furthermore, some other systems have been also introduced such as GraphX [21], Trinity [6], GRACE [22] and Signal/Collect [23].

In the early days of the Hadoop framework, the lack of declarative languages to express the large-scale data processing tasks has represented one of the main limitations towards its practical usage and wide acceptance by many users [12]. As a result, several declarative querying frameworks (e.g., Apache Pig, Apache Hive) have been introduced on top of the Hadoop stack in order to fill this gap. In practice, these frameworks have gained wide attention and adoption in the industry and research communities. Nowadays, the systems/stacks of large scale graph processing platforms are suffering from the same challenge of the early days of the Hadoop framework. Therefore, with the current momentum and increasing interest on building and using distributed graph processing platforms, we believe that it is beyond doubt that high level programming abstractions and declarative querying frameworks that ease the user's job for expressing their graph processing jobs and enable the underlying systems/stacks to perform automatic optimization are crucially required and represent an important research direction to enrich this domain.

BIG SQL PROCESSING SYSTEMS

Several systems have been introduced to support the SQL flavor on top of the Hadoop infrastructure and provide competing and scalable performance on processing large scale structured data. For example, Hive [8] is considered to be the first system which has been introduced to support SQL-on-Hadoop with familiar relational database concepts such as tables, columns, and partitions. Hive has been widely used in many reputable organizations to manage and process large volumes of data, such as Facebook, eBay, LinkedIn and Yahoo! [8]. Hive supports all of the major primitive types (for example, integers, floats, and strings) and complex types (for example, maps, lists, and structs). It also supports queries that are expressed in an SQL-like declarative language, Hive Query Language (HiveQL[1]), which represents a subset of SQL92, and therefore can be easily understood by anyone who is familiar with SQL. These queries automatically compile into MapReduce jobs that are run by using Hadoop. HiveQL enables users to plug custom MapReduce scripts into queries as well. Recently, Huai et al. [24] have reported about the major technical advancements that have been implemented into the HIVE project by its development community. These advancements include a new file format, Optimized Record Columnar File (ORC File) [25], which is designed to provide high storage and data access efficiency with low overhead. In addition, the query planning component has been updated to provide more sophisticated optimizations for complex queries and significantly reduce unnecessary operations in the executed query plans. Hive has some limitations eg. Updating data is complicated because of using HDFS, no real time access to data and latency for Hive queries is generally very high.

Impala [26] is another open source project, built by Cloudera, to provide a massively parallel processing SQL query engine that runs natively in Apache Hadoop. Impala does not use Hadoop to run the queries. Instead, it relies on its own set of daemons, which are installed alongside the data nodes and are tuned to optimize the local processing to avoid bottlenecks. In principle, Impala is part of the Hadoop ecosystem and shares the same infrastructure (for example, metadata, Apache Hive). Therefore, by using Impala, the user can query data which is stored in Hadoop Distributed File System (HDFS) [27]. It also uses the same metadata, SQL syntax (HiveQL), that Apache Hive uses. One of the main limitations of Impala is that it relies on an in-memory join implementation. Therefore, queries can fail if the joined tables can't fit into memory. Impala does not replace Hive or other frameworks built on MapReduce for long-running batch-oriented queries. Impala is not fit as a query layer to support

[1] https://cwiki.apache.org/confluence/display/Hive/LanguageManual

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

operational/OLTP applications (No update/deletes, not optimized for point look-ups).

Big SQL [28] is the SQL interface for the IBM big data processing platform, InfoSphere BigInsights, which is built on the Apache Hadoop framework. In particular, it provides SQL access to data that is stored in InfoSphere BigInsights and uses the Hadoop framework for complex data sets and direct access for smaller queries. In the initial implementation of Big SQL, the engine was designed to decompose the SQL query into a series of Hadoop jobs. For interactive queries, Big SQL relied on a built-in query optimizer that rewrites the input query as a local job to help minimize latencies by using Hadoop dynamic scheduling mechanisms. The query optimizer also takes care of traditional query optimization such as optimal order, in which tables are accessed in the order where the most efficient join strategy is implemented for the query. The design of the recent version of the Big SQL engine has been implemented by adopting a shared-nothing parallel database architecture, in which it replaces the underlying Hadoop framework with a massively parallel processing SQL engine that is deployed directly on the physical Hadoop Distributed File System (HDFS). Therefore, the data can be accessed by all other tools of the Hadoop ecosystem, such as Pig and Hive. The system infrastructure provides a logical view of the data through the storage and management of metadata information. In particular, a table is simply a view that is defined over the stored data in the underlying HDFS. In addition, the Big SQL engine uses the Apache Hive database catalog facility for storing the information about table definitions, location and storage format. one of the fundamental challenges with the Big SQL is they are mainly designed for supporting analytical workload without any consideration of transactional workloads. I

Facebook has released Presto [29] as an open source distributed SQL query engine for running interactive analytic queries against large scale structured data sources of sizes up to gigabytes to petabytes. In particular, it targets analytic operations where expected response times ranges from sub-second to minutes. Presto allows querying data where it lives, including Hive, NoSQL databases (e.g., Cassandra), relational databases or even proprietary data stores. Therefore, a single Presto query can combine data from multiple sources. Presto currently has limited fault tolerance capabilities when querying. If a process fails while processing, the whole query must be re-run.

2. DECLARATIVE BIG GRAPH PROCESSING FRAMEWORK

Currently the systems/stacks of large scale graph processing platforms are suffering from the same challenge. The aim of our work is to fill this gap and introduce a flexible and extensible declarative querying framework on top of the emerging distributed graph processing platforms. Our proposed framework consists of declarative graph-based query language, metadata-based catalog for distributed graphs and cost-based query compiler that compiles the declarative graph-based queries and computation generates efficient execution plans using the low-level programming abstractions of the underlying distributed graph processing platforms. In general, for the graph pattern matching operations, it is necessary to express a query graph declaratively. To achieve this goal, we rely on Cypher, the graph query language of Neo4j [30]. In practice, there is an ongoing effort to standardize Cypher as a graph query language within the open Cypher project[2].

Figure 1 illustrates our proposed framework architecture with the following main components:

1- A declarative graph-based query language, Cypher, that can express various graph querying requirements (e.g., pattern matching, shortest path) of different application domains (e.g., Web graphs, social networks).
2- A cost-based query optimizer for distributed graph storage. The query optimizer collects metadata and statistics about the stored graph partitions, generate different execution plans and select among them based on statistical cost-based model.
3- The framework is designed in a form of being agnostic towards the underlying distributed graph processing platform. Therefore, the query compiler of the framework will be designed in a flexible and extensible fashion that enables compiling the generated execution plans into the low-level programming abstractions of the various distributed graph processing platforms. For example,

[2] https://www.opencypher.org/

Journal of Theoretical and Applied Information Technology
28th February 2018. Vol.96. No 4

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

our framework can translate Cypher queries to SQL statements that are evaluated using Big SQL systems [31] or BSP implementation for graph pattern match queries [32].

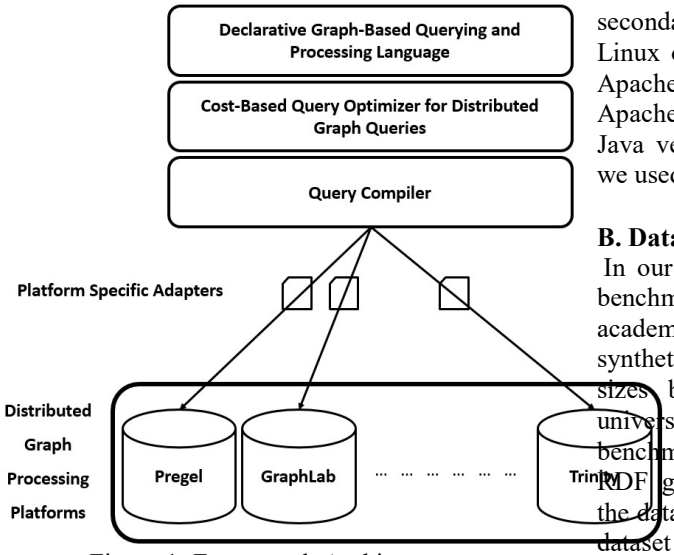

Figure 1: Framework Architecture

In practice, for any declarative query, there are always various possible execution plans to evaluate such query. Thus, our framework is equipped with a query optimizer that seeks to optimize the query execution time for any input query. In particular, among a wide space of alternative possible query plans for executing the user input query, the query optimizer employs a cost model to predict the time execution cost of each plan then selects the execution plan that with the minimum cost for actual execution. In order to achieve this goal, the query coordinator node maintains a set of graph statistics (e.g., structural indices, selectivity information of value-based predicates on the attributes of graph nodes and edges) which are utilized by the query optimizer to estimate the time execution cost of each possible query plan.

In practice, the query optimizer starts by compiling the user input query (Q) into a logical query plan using a defined set of algebraic operators. Using the statistical information and the cost model, the query optimizer compiles the logical query plan into a set of sub-query physical query execution plans. Finally, our framework relies on a set of cost-based query optimization techniques that attempt to estimate the cost of the various possible execution plans and predicts the one with the lowest-cost or at least a closest one to it.

3. EXPERIMENTS AND EVALUATION

A. Experimental Environment

Our experiments have been conducted on a cluster of 20 nodes of Amazon EC2 Computing nodes[3]. Each server has an Intel QuadCore 2.9 GHz CPU, 32 GB of main memory storage, 1 TB of SCSI secondary storage and runs the 64-bit Fedora 13 Linux operating system. For the comparison with Apache Giraph Systems, we have been using Apache Hadoop 2.6.0, Apache Giraph 1.1.0 and Java version 7. For the comparison with Impala, we used version 2.5.

B. Datasets

In our experiments, we used the popular LUBM benchmark [33] which provides an ontology for academic information (e.g., universities). This is a synthetic dataset that can be generated with various sizes by controlling the number of generated universities. The original data generator of the benchmark generates the dataset according to the RDF graph model. Therefore, we have modified the data generator of the benchmark to generate the dataset according to the attributed graph model6. In order to evaluate the scalability of our system, we generated a dataset with 50K universities (1.2 TB).

C. Workload

In practice, there is no defined standard benchmarks for evaluating the performance of query engines. Therefore, we defined our own categories queries which we used in our evaluation. In particular, we used two main categories of queries:

- *Highly Selective Pattern Matching Queries*: This category represents a connected graph pattern (e.g., path, star, subgraph) with highly selective predicates that matches to a small set of answers.
- *Low Selective Pattern Matching Queries:* This category represents a connected graph pattern with low selective predicates that matches to a large set of answers.

Our main performance metric is the query execution time. In particular, each query instantiation of the experimental workload has been executed 5 times and execution times were collected. All times are in seconds. In order to ensure that any caching or system process activity would not affect the collected results, the longest and shortest times for each instantiation were

[3] https://aws.amazon.com/ec2/

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

dropped and the remaining three execution times for the 20 instantiations were averaged.

D. Experiments

In our evaluations, we conducted two main experiments:

- The focus of our first experiment is to evaluate the efficiency of our query optimization techniques. To achieve this goal, we mapped our graph experimental dataset into relational data which has been stored on the Impala cluster. In addition, we have translated the Cypher experimental workload into SQL queries with and without using our query optimization techniques.
- Our second experiment focused on evaluating the usability of the declarative interface on the productivity of users. We compared the required time for the users to express their graph pattern matching queries using the Cypher language and

E. Results

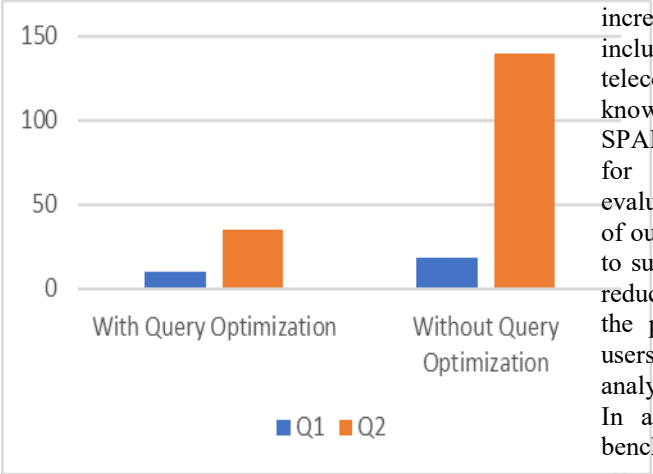

Figure 2: Impact of Query Optimization Techniques

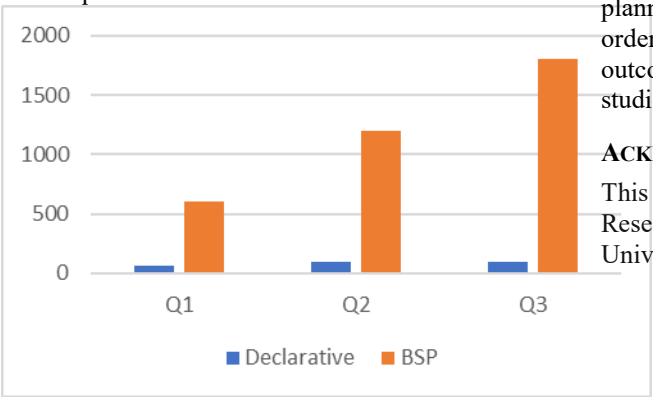

Figure 3: Declarative Querying vs BSP Querying

Figure 2 illustrates the results of our first experiments. The results show the high impact of our query optimization techniques on improving the query performance. The impact of the low selective pattern matching query category is higher due to the effectiveness on avoiding the generation of bigger intermediate results. Figure 3 show the outcome for the comparison on user productivity on implementing the graph pattern matching queries using the declarative interface and the low-level APIs of the BSP programming interface. Clearly, the declarative interface is much easier and faster for the end users to express their queries. The more number of predicates and the more complex the query, the higher the effectiveness of the declarative interface on improving the user productivity.

4. CONCLUSION AND FUTURE WORK

A graph is a popular data model that has become pervasively used for modeling structural relationships between objects. It has been increasing used in several application domains including social network, computer networks, telecommunication networks, the Web, and knowledge bases. In this article, we presented DG-SPARQL, an efficient and declarative framework for querying big graphs. Our experimental evaluation validated the efficiency and scalability of our approach. As a future work, we are planning to support visual query interfaces that can further reduce the burden of query formulation and ease the process for different types of non-technical users. Another future direction is to support the analysis of unstructured and semi-structured data.

In addition, there is a clear lack of standard benchmarks for evaluating the performance of query engines and for building the required depth and common global understanding. We are planning to tackle this problem as a future work in order to guide and improve the significance of the outcomes of such evaluation and benchmarking studies.

ACKNOWLEDGMENT

This research was funded by Deanship of Scientific Research at Princess Nourah bint Abdulrahman University. (Grant no: 227- ص -38)

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REFERENCES

[1] Malewicz, Grzegorz, Matthew H. Austern, Aart JC Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. *Pregel: a system for large-scale graph processing*. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 135-146. ACM, 2010.

[2] Seo, Sangwon, Edward J. Yoon, Jaehong Kim, Seongwook Jin, Jin-Soo Kim, and Seungryoul Maeng. *Hama: An efficient matrix computation with the mapreduce framework*. In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on, pp. 721-726. IEEE, 2010.

[3] Sakr, Sherif, Faisal Moeen Orakzai, Ibrahim Abdelaziz, and Zuhair Khayyat. *Large-Scale Graph Processing Using Apache Giraph.* Springer, 2016.

[4] Low, Yucheng, Joseph E. Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E. Guestrin, and Joseph Hellerstein. *Graphlab: A new framework for parallel machine learning*. arXiv preprint arXiv:1408.2041 (2014).

[5] Gonzalez, Joseph E., Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. *PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs*. In OSDI, vol. 12, no. 1, p. 2. 2012.

[6] Shao, Bin, Haixun Wang, and Yatao Li. *Trinity: A distributed graph engine on a memory cloud.* In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 505-516. ACM, 2013.

[7] Olston, Christopher, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. *Pig latin: a not-so-foreign language for data processing.* In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1099-1110. ACM, 2008.

[8] Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. *Hive: a warehousing solution over a map-reduce framework*. Proceedings of the VLDB Endowment 2, no. 2 (2009): 1626-1629.

[9] Dean, Jeffrey, and Sanjay Ghemawat. *MapReduce: simplified data processing on large clusters*. Communications of the ACM 51, no. 1 (2008): 107-113.

[10] White, Tom. *Hadoop: The definitive guide*. O'Reilly Media, Inc.", 2012.

[11] Sakr, Sherif. *Big Data 2.0 Processing Systems: A Survey*. Springer, 2016.

[12] Sakr, Sherif, Anna Liu, and Ayman G. Fayoumi. *The family of mapreduce and large-scale data processing systems*. ACM Computing Surveys (CSUR) 46, no. 1 (2013): 11.

[13] Bu, Yingyi, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. *HaLoop: Efficient iterative data processing on large clusters*. Proceedings of the VLDB Endowment 3, no. 1-2 (2010): 285-296.

[14] Ekanayake, Jaliya, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. *Twister: a runtime for iterative mapreduce*. In Proceedings of the 19th ACM international symposium on high performance distributed computing, pp. 810-818. ACM, 2010.

[15] Zhang, Yanfeng, Qixin Gao, Lixin Gao, and Cuirong Wang. *imapreduce: A distributed computing framework for iterative computation*. Journal of Grid Computing 10, no. 1 (2012): 47-68.

[16] Chen, Rishan, Xuetian Weng, Bingsheng He, and Mao Yang. *Large graph processing in the cloud*. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, pp. 1123-1126. ACM, 2010.

[17] Kang, U., Charalampos E. Tsourakakis, and Christos Faloutsos. *Pegasus: A peta-scale graph mining system implementation and observations*. In Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on, pp. 229-238. IEEE, 2009.

[18] Bu, Yingyi, Vinayak Borkar, Jianfeng Jia, Michael J. Carey, and Tyson Condie. *Pregelix: Big (ger) graph analytics on a dataflow engine*. Proceedings of the VLDB Endowment 8, no. 2 (2014): 161-172.

[19] Salihoglu, Semih, and Jennifer Widom. *GPS: A graph processing system*. In Proceedings of the 25th International Conference on Scientific and Statistical Database Management, p. 22. ACM, 2013.

[20] Khayyat, Zuhair, Karim Awara, Amani Alonazi, Hani Jamjoom, Dan Williams, and Panos Kalnis. *Mizan: a system for dynamic load balancing in large-scale graph processing*. In Proceedings of the 8th ACM European Conference on Computer Systems, pp. 169-182. ACM, 2013.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[21] Xin, Reynold S., Joseph E. Gonzalez, Michael J. Franklin, and Ion Stoica. *Graphx: A resilient distributed graph system on Spark*. In First International Workshop on Graph Data Management Experiences and Systems, p. 2. ACM, 2013.

[22] Wang, Guozhang, Wenlei Xie, Alan J. Demers, and Johannes Gehrke. *Asynchronous Large-Scale Graph Processing Made Easy*. In CIDR, vol. 13, pp. 3-6. 2013.

[23] Stutz, Philip, Abraham Bernstein, and William Cohen. *Signal/collect: graph algorithms for the (semantic) web*. The Semantic Web–ISWC 2010 (2010): 764-780.

[24] Huai, Yin, Ashutosh Chauhan, Alan Gates, Gunther Hagleitner, Eric N. Hanson, Owen O'Malley, Jitendra Pandey, Yuan Yuan, Rubao Lee, and Xiaodong Zhang. *Major technical advancements in apache Hive*. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, pp. 1235-1246. ACM, 2014.

[25] He, Yongqiang, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang, and Zhiwei Xu. *RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems*. In Data Engineering (ICDE), 2011 IEEE 27th International Conference on, pp. 1199-1208. IEEE, 2011.

[26] Bittorf, M. K. A. B. V., Taras Bobrovytsky, Casey Ching Alan Choi Justin Erickson, Martin Grund Daniel Hecht, Matthew Jacobs Ishaan Joshi Lenni Kuff, Dileep Kumar Alex Leblang, Nong Li Ippokratis Pandis Henry Robinson, David Rorke Silvius Rus, John Russell Dimitris Tsirogiannis Skye Wanderman, and Milne Michael Yoder. *Impala: A modern, open-source SQL engine for Hadoop*. In Proceedings of the 7th Biennial Conference on Innovative Data Systems Research. 2015.

[27] Borthakur, Dhruba. *HDFS architecture guide*. Hadoop Apache Project 53 (2008).

[28] Gray, S., F. Özcan, H. Pereyra, B. van der Linden, and A. Zubiri. *IBM Big SQL 3.0: SQL-on-Hadoop without compromise*. (2014).

[29] Traverso, Martin. *Presto: Interacting with petabytes of data at Facebook*. Retrieved February 4 (2013): 2014.

[30] Webber, Jim. *A programmatic introduction to Neo4j*. In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity, pp. 217-218. ACM, 2012.

[31] Chong, Eugene Inseok, Souripriya Das, George Eadon, and Jagannathan Srinivasan. *An efficient SQL-based RDF querying scheme*. In Proceedings of the 31st international conference on Very large data bases, pp. 1216-1227. VLDB Endowment, 2005.

[32] Fard, Arash, M. Usman Nisar, Lakshmish Ramaswamy, John A. Miller, and Matthew Saltz. *A distributed vertex-centric approach for pattern matching in massive graphs.* In Big Data, 2013 IEEE International Conference on, pp. 403-411. IEEE, 2013.

[33] Guo, Yuanbo, Zhengxiang Pan, and Jeff Heflin. *LUBM: A benchmark for OWL knowledge base systems.* Web Semantics: Science, Services and Agents on the World Wide Web 3, no. 2 (2005): 158-182.