Towards an Understanding of Scalable Query and Data Analysis for Social Media Data using High-Level Dataflow Systems

Tak Lon (Stephen) Wu
School of Informatics and Computing
Indiana University Bloomington
taklwu@indiana.edu

Judy Qiu
School of Informatics and Computing
Indiana University Bloomington
xqiu@indiana.edu

ABSTRACT

Nowadays there is great research potential in analyzing the vast amount of data collected from social media and social network applications. In order to explore the correlations among this data and social activities, modeling techniques such as data mining and machine learning have been applied in combination with ad hoc query and complicated post-query analysis. Use of high-level platforms such as Pig, Hive, and Spark SQL to support this type of sophisticated analysis has become popular. However, the question remains: which of the available software building blocks can serve users best according to their data needs? This question motivated us to research the execution flow and performance characteristics of these platforms, focusing on our special interests of social media data, to provide a detailed comparison of high-level language frameworks for ad hoc queries and applications.

Categories and Subject Descriptors
H.3.3 [**Information Search and Retrieval**]: Search process

General Terms
Experimentation, Performance, Languages

Keywords
Social Media Data Analysis, Pig, Hive, Spark SQL, IndexedHBase, Dataflow

1. INTRODUCTION

Social media data and its applications have gained the attention of commercial, academic and research communities. Many interesting research applications [1-6] that deal with daily activities, events, and knowledge in human society have been yielded. The data collected by these metrics of social media is vast, far exceeding constraints found in the low-level storage, databases, and runtimes traditionally used to store and access historical data. In practice, social media service providers such as Twitter, Facebook and Instagram have accommodated users with their customized solutions. However, for those research scientists and application developers who subscribe to public social streams and build their research systems and prototypes, it is challenging to select appropriate software building blocks that can scalably store, serve, and customize data schema for such immense data.

Gao et al. [7-10] working with the *Truthy* [11] project demonstrates the usefulness of Apache software stacks with *IndexedHBase* [12]. This in turn led to Truthy's current deployment on a large-scale and large-storage private cluster, MOE. Though IndexedHBase and its Java API have met the fundamental requirements for accessing and processing data analysis, there are still unfulfilled areas where further research is viable, especially when integrating the existing analysis pipelines with Apache high-level language platforms such as Pig [13], Hive [14] and Spark SQL [15]. Other challenges include ad hoc queries and direct computation on top of storage and databases.

The scope of our research is outlined here in fine-grained low-level perspective such as I/O(s) consumption benchmark, comparison of system-level building blocks [16], and performance optimization corresponding to different types of data processing. Our goal is to understand the requisite background knowledge, review the existing research, and performance benchmarks. Based on these results we can identify the computation and performance characteristics for queries and applications run on these high-level platforms. Using these categories, we will further investigate the differences between social media data and general data analysis to identify the potential customizations for social data analysis using high-level platforms.

This paper focuses on understanding the requirements and boundaries of data systems that support various applications integrated with ad hoc queries and data analysis, especially for social media data. We benchmark query systems including Pig, Hive, IndexedHBase and Spark SQL. In particular, our previous work has investigated the possibility of using User Defined Functions (*UDF*) to support complicated iterative algorithms with fine-grained data aggregation and communication patterns [17]. We deploy an end-to-end pipeline for general scientific data and social media data processing.

The paper is structured as follows. Section 2 introduces our use cases and data model, system-level requirements, and infrastructure in supporting social media data. Section 3 describes the features of ad hoc queries, and our research using high-level languages with NoSQL databases for query and data analysis. Section 4 gives performance benchmarks on applications. Lastly, we draw our conclusion and summarize the research directions in Section 5 and Section 6.

2. TRUTHY - SOCIAL MEDIA OBSERVATORY

Truthy is a public social media observatory developed as a research project at Indiana University. It analyzes and visualizes information diffusion on Twitter. Truthy monitors and collects Twitter data in real-time directly through the Twitter public streaming API [18]. Much of our work has been accomplished to support this observatory, upon which researchers have yielded inferences of human society by analyzing the social activities in cyberspace.

One example of end-to-end social media data analysis [2] involved utilizing the IndexedHBase queries [12] on top of data mining techniques, such as eigenvector modularity [19] and label propagation [20]. The analysis was carried out on two datasets about political discussion collected during the six weeks leading up to the 2010 U.S. congressional midterm elections and 2012 U.S. presidential elections. The results shown in [2, 12] prove that the retweet networks exhibited a highly segregated partisan structure; users of those tweets are mainly split into two homogenous communities corresponding to the political left and right leanings. Figure 1 shows the execution flow for getting the graph of political polarization. In 2012, the average amount of collected tweets each month was about 1 billion tweets. Such a large dataset proved problematic in terms of storage. Because of this we provide a fast processing layer to handle such high volume and complexity of data.

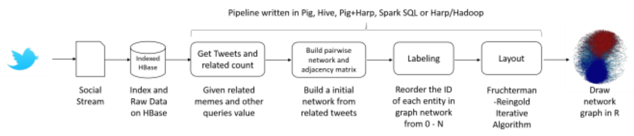
Pipeline written in Pig, Hive, Pig+Harp, Spark SQL or Harp/Hadoop
Indexed HBase
Get Tweets and related count
Build pairwise network and adjacency matrix
Labeling
Layout
Social Stream
Index and Raw Data on HBase
Given related memes and other queries value
Build a initial network from related tweets
Reorder the ID of each entity in graph network from 0 - N
Fruchterman -Reingold Iterative Algorithm
Draw network graph in R

Figure 1. Sophisticated pipeline for visualizing Political Polarization

This data observatory has been storing Twitter streaming data since July 2010; the current data size as of Aug 2015 is approximately 162TB. It includes raw compacted JSON files on HDFS, tweet fields and inverted indices stored as HBase tables. IndexedHBase [12] has been used to create inverted indices for raw tweets in JSON format. Various fields such as keywords, hashtags, geographical locations, user IDs, and retweet IDs have been stored as searchable rowkeys while the related tweet IDs are stored as associated (multi-column) values. Scientists and developers of Truthy perform ad hoc queries and post-query data analysis on these HBase tables, where tweet tables (JSON fields) and index tables are semi-structured with different amounts of columns.

2.1 System Challenges for Truthy

We have compared different NoSQL solutions to support indexing and fast queries on large-scale social media data. As a result, IndexedHBase was selected as the framework to store, serve, and perform data computation for data scientists [7-10] using YARN Hadoop and HBase as building blocks.

Utilizing an infrastructure supported by IndexedHBase, our work delves into the system support for ad hoc queries and post-query data analysis performed on large-scale social media data. Based on our studies, the three categories of challenges are data-related, system-related, and programming and computation-related. Data related issues involve storing and serving incremental data on a scale of at least TB level, sustaining or creating indices with customized formats, and supporting flexible data schema for structured and semi-structured data with less disk consumption. System-related issues offer multi-tenancy to query clients and application developers, as well as allowing commodity hardware failures with fast/auto recovery. Finally, programming and computation-related issues support ad hoc query interfaces such as Pig, Hive, and Spark SQL, in addition to supporting customized programming in imperative programming languages such as Java and Python. They offer different levels of parallelism and sophisticated data mining and machine learning applications.

3. AD HOC QUERY WITH NOSQL DATABASE

A key characteristic of social media data analysis is ad hoc queries that select a subset data of interest from a very large dataset in databases, which has time and spatial attributes. Each row/field of tweet data is stored with an associated timestamp and their related column values. An example query could be “Find all the related tweets with given hashtag #computing in the time range between June 15th 2015 and July 10th 2015”. This type of query can be rewritten as traditional Select-Project-Join (SPJ) ad hoc queries. They project and join the two datasets, the records within that specific time, and other sets of records within the target fields, such as hashtags. The size of projection data, amount of generated temporary tables, and the type of join operations depends on the target fields of each query within a single table. For instance, the execution flow of the example query given above firstly scans the entire raw data table and filters the required data by referring to the given predicates of time duration and hashtag. Then it generates two temporary tables and performs a single shared-key join. Due to the extra overhead of generating two tables separately, in addition to performing a join aggregation and scanning entire rows of each target record, Gao et al. [7-9] has shown that the overall performance does not meet our expectations. By comparison, the HBase solution scans the index and raw tables once and immediately filters the data with the support of built-in “create timestamp” for each stored row/column in a table. Even adopting NoSQL databases as backend storage, there are limited choices of database solutions that can efficiently store large datasets with fast (inverted) index access to the time spatial data. IndexedHBase has been developed as the backend inverted index layer, where the data and indices are stored on top of HBase to support these complicated social media data queries.

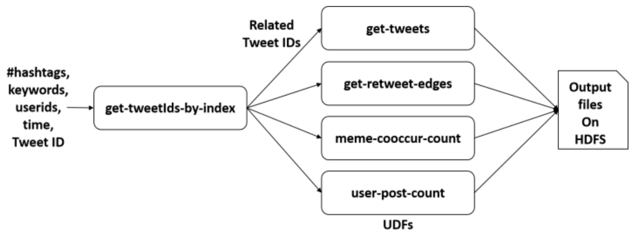

Figure 2. Dataflow for Ad hoc queries of social media data

Most of our queries are HBase I/O intensive, which mainly perform random data access by specified row keys, e.g. tweet IDs to tweet table and keywords to text index tables. Each query must first retrieve the related tweet IDs from index tables by a given time range and queried keys. It then obtains the required columns from the tweet table and may perform a UDF to yield a stage-ready result output on HDFS for further data analysis as shown in Figure 2. This differs from SQL database procedure. IndexedHBase must build the indices as separate tables on HBase, and it considers extra overheads when loading data into HBase. Based on the execution flow and different type of data transformation of these queries, we have identified four categories and a total of 17 queries as summarized in Table 1:

1. *Read-One-Write-One*: Obtain one related tweet ID from Index Table by the given queried key (e.g.

hashtag), dump the whole tweet as result, e.g. get-tweets-with-meme.

2. *Read-One-Transform-One*: Obtain one related tweet ID from Index Table by the given queried key (e.g. hashtag), generate single output entry (e.g. user pair) from the obtained tweet, e.g. get-retweet-edges.
3. *Read-One-Transform-Many*: Obtain one related tweet ID from Index Table by the given queried key (e.g. hashtag), generate multiple output entries as ArrayList, e.g. meme-cooccur-count.
4. *Single-Scan*: read the statistic information directly from HBase table.

Type	**Query**	**Exe. Steps**
Read-One-Write-One	get-tweets-with-meme, get-tweets-with-text, get-tweets-with-userid, get-retweets, get-tweets-with-time, get-tweets-with-phrase	2
Read-One-Transform-One	get-retweet-edges, get-mention-edges	2
Read-One-Transform-Many	meme-post-count, text-post-count, userid-post-count, user-post-count, user-post-count-by-text, meme-cooccur-count	2
Single-Scan	meme-timestamp-count, text-timestamp-count, userid-timestamp-count	1

Table 1. Classification of support social queries

3.1 Query Execution with High-level Languages

IndexedHBase includes Java MapReduce implementations driven by a wrapper bash shell. Despite this, it is not easy to add new queries or UDF without understanding the background of Hadoop MapReduce. Specifically, all the supported social media data queries are very straightforward ad hoc queries executed with common database operations such as *FILTER*, *GROUP BY*, *JOIN*, and FOR EACH with built-in or UDF functions. This motivated us to investigate the integration with high-level abstractions such as *Pig*, *Hive*, and *Spark SQL* for day-to-day query and data analysis.

Most of these systems are considered as *Dataflow* type of system or *Dataflow programming* model, which is a paradigm that models a program as a directed graph of data [21]. In both cases, data flows among a series of components such as operators and functions which serve as a "*black-box*" unit (the detailed implementations are already defined) to transform the incoming data from its original format into another. Data in the execution flow is clearly defined as either being input or output to every atomic component, independently handled on each and inherently run in parallel.

Pig [13] is a dataflow system built on top of Hadoop MapReduce, which aims to serve as a high level abstraction interfacing with SQL database and MapReduce computation systems. Pig itself is a declarative DAG-flow system, but it uses *Pig-Latin* [22], a procedural language. This makes it flexible and allows users to choose different implementations of the same relational operator (e.g. *JOIN* and *GROUPBY*) in execution. Other than the built-in operators, a developer can apply their own sophisticated algorithm to the dataflow in Pig via its UDFs. *Hive* [14] is another high-level platform, but it differs from Pig by supporting data warehouse ad hoc queries and simple MapReduce applications for structured data stored on *HDFS* [23]. It provides a SQL-like language, *HiveQL*, to execute on top of Hadoop. Most of the implementation concepts of Hive derive from SQL RDBMS. *Spark SQL* [15] is another open source project inspired by Hive. Instead of being coupled with the Hadoop MapReduce engine, it uses Spark as its low-level runtime, with DataFrame schema RDD as its major in-memory data structure embedded with named column (table-like) schema. The extensible query optimizer Catalyst is written in Scala, a different model from Hive and its predecessor *Shark* [24].

Although we have not yet linked the ad hoc query with the post-query analysis, we recognize the need for chaining this intermediate data to next-generation compute resources and fulfilling the dataflow of the entire analysis pipeline. Our previous work [17] has demonstrated the importance of in-memory computation and resource reuse for sophisticated machine learning applications with iterations. We have shown that incorporating the Hadoop plugin Harp allows general ETL queries to seamlessly integrate with sophisticated analysis applications [15]. This would save significant job restart overhead and enable fast resource allocation and reusability. Furthermore, it enables intuitive development writing prototypes of end-to-end pipelines in a single environment. Spark SQL has proposed a similar idea that uses the same platform and data abstractions for both queries and analysis, yielding meaningful results for sophisticated algorithms. Meanwhile, *Apache Tez* [25] shows the importance of resource reusability for complex DAG tasks on top of high-level platforms run on YARN Hadoop.

4. PERFORMANCE RESULTS

We measure the overhead of using these high-level platforms for the target ad hoc queries. Our experiments run on MOE, a large-storage, large-memory and high-performance private cluster at Indiana University dedicated to the Truthy project [11, 26]. It consists of 3 login nodes and 10 compute nodes, where each login node is set up with two Intel(R) Xeon(R) CPU E5-2620 v2 CPUs, 64 GB memory, and each compute node has two Intel(R) Xeon(R) CPU E5-2660 v2 CPUs, 128 GB memory, 48TB HDD and 120GB SSD. All nodes are interconnected with a 10Gb Ethernet. We perform our tests on top of a Hadoop 2.5.1 cluster with different high-level platforms such as Pig 0.14.0, Hive 1.0.0, and Spark SQL 1.5.0. Meanwhile, IndexedHBase 0.2.0 is the Java MapReduce baseline.

We have implemented a total of 17 ad hoc queries [12] written in Pig, Hive, IndexedHBase and Spark SQL. Other than the initial stage of searching related tweet IDs from index tables, we select three query implementations and benchmark them on different platforms to examine the runtime behaviors as shown in Figures 3-5. Each query submission runs with a total of 587858 tweet IDs obtained from meme index tables by being given the most common hashtag, *"Follow Friday" #ff* and are equally assigned to 9 workers; the default parallelism (the amount of reducers) is set to 4.

Since all of these are I/O intensive queries using HBase, a major overhead is the data retrieval time communicated with the HBase tweet table which stores the original tweet fields. Other than get-tweets query, which dumps the entire tweet to HDFS, every implemented UDF only scans a subset of columns and yields a specified format such as edge pair (user ID and retweet user ID) and a list of mentioned hashtags in the related tweet. The transformed data are collected and accumulated by using the standard data aggregation operations, e.g. *GROUP BY* and *reduceByKey*. Compared with traditional row-based databases, we reduce significant I/O overhead with the help of the columnar scanning in HBase. Note that the computation time of Spark SQL takes longer as Spark performs "map-only" worker execution; it includes the UDF transformation time (from DataFrame RDD to Java RDD) and cross-worker data aggregation/communication time, along with the output to HDFS time.

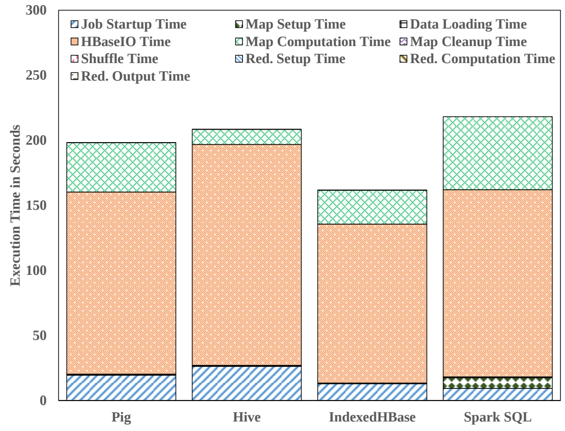

Figure 3. Performance breakdown for get-tweets

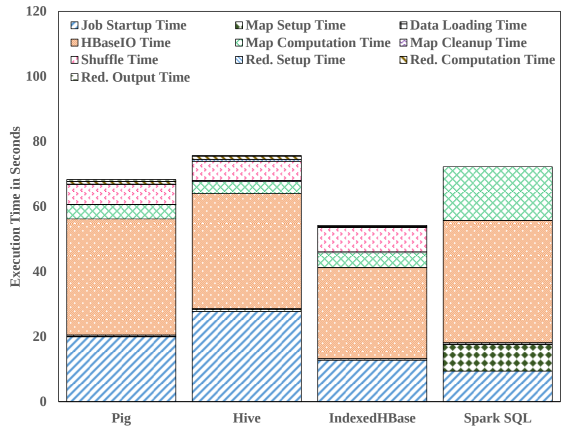

Figure 4. Performance breakdown for get-retweet-edges

As shown in Figures 3-5, IndexedHBase performs the best as it has minimum startup overhead for constructing the high-level execution flow. Also, before writing data to the output buffer, each worker is optimized with an in-memory combination; if there is any intermediate data, they share the same emit key. Note that the computation time of Spark SQL takes longer as Spark performs "map-only" worker execution; it includes the UDF transformation time (from DataFrame RDD to Java RDD) and cross-worker data aggregation/communication time, along with the output to HDFS time.

Since all our queries in Table 1 are compiled and run as YARN or Hadoop jobs, we evaluate the local write bytes (except Spark SQL which does not have a reduce stage) and investigate the data aggregation overhead. Figure 6 shows that for queries with reduce stages (*get-retweet-edges* and *meme-cooccur-count*), Pig and Hive implementations generate more intermediate data that matches the increasing time of execution. This is due to these high level abstractions using tuple-based computations and emitting each processed tuple to the output buffer. IndexedHBase is a Java MapReduce implementation with which the output of the mapper is optimized, combining the emitted values that share the same output key. We observe this behavior from the intermediate record sizes as shown in Table 2.

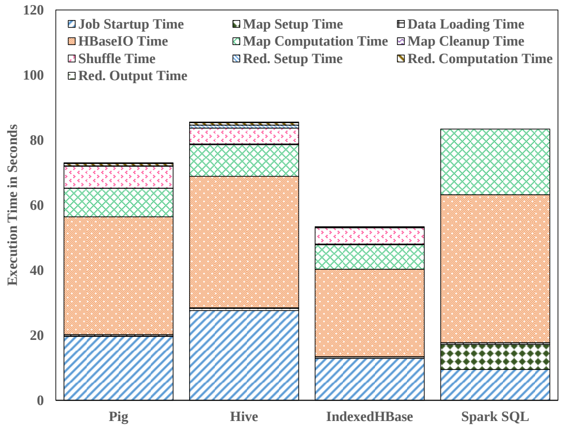

Figure 5. Performance breakdown for meme-cooccur-count

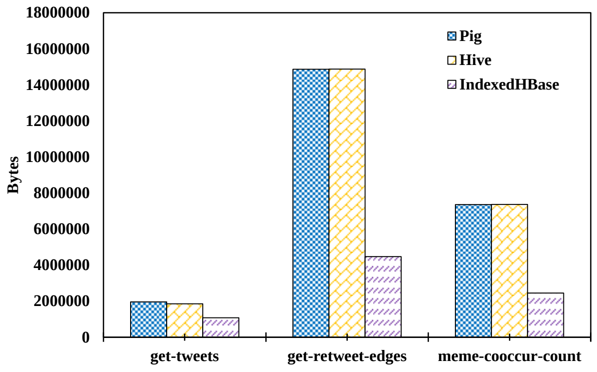

Figure 6. Intermediate Local write in bytes

Query	Pig	Hive	IndexedHBase
get-tweets	587858	587858	587858
get-retweet-edges	179486	179463	167740
meme-cooccur-count	90216	90125	63524

Table 2. Mapper output (combined if any) record sizes

5. CONCLUSION

This paper compares social media data query performance on a large-scale data observatory. To address the challenges in various levels of this observatory, we proposed to use inverted indices generated by IndexedHBase with different high-level abstractions to perform efficient query and post-query data analysis. The benchmarks show that major overhead of ad hoc queries come from I/O and job startup and IndexedHBase can further improve query performance compared to other high level platforms. Our future work will be centered on an integrated solution that makes it simpler and more efficient for users to conduct query and analysis. This implies that programming interface, computation extension, and locality-aware data linkage are constructed within an interoperable platform. We have achieved better resource utilization by reducing the resource allocation overhead, and fast data access with in-memory caches for frequently used data within a pipeline [17]. With optimized query execution flow, we can support real-time and statistical data metrics of data processing.

Our research currently does not investigate the *query optimization* of databases [27-30] with optimization strategies such as predicates move-around [29], which have been implemented in many database [31-33, 14] and dataflow [13] systems, especially for Select-Project-Join (SPJ) ad hoc queries. However, the social media data queries can prove challenging for traditional SPJ database systems. Our implementation therefore avoids the SPJ complexity by using inverted indices with associated timestamps within the same cell of data.

6. FUTURE WORK

We have integrated Harp with Pig [17] to show the advantages of using customized data aggregation and in-memory computation for ad hoc query and analysis applications. We have produced a survey [16] to evaluate the basic features and fundamental differences among Pig, Hive and Spark SQL. Based on these efforts, we plan to extend our research from state-of-the-art Apache high-level language platforms to end-to-end solutions that link multiple compute components into a single development and platform. We will evaluate these high-level platforms versus domain-specific languages such as R and Matlab.

7. ACKNOWLEDGEMENT

We gratefully acknowledge support from the National Science Foundation (grant OCI-1149432) and would like to thank our colleagues in the SALSA team, Prof. Filippo Menczer and our collaborators in the Truthy team at Indiana University for their support and comments.

8. REFERENCES

[1] Joseph M Hellerstein and Michael Stonebraker, Predicate migration: Optimizing queries with expensive predicates. Vol. 22. 1993, ISBN: 0897915925: ACM.

[2] Michael Conover, Jacob Ratkiewicz, Matthew Francisco, Bruno Gonçalves, Filippo Menczer, and Alessandro Flammini. Political polarization on twitter. in ICWSM. 2011.

[3] Jacob Ratkiewicz, Michael Conover, Mark Meiss, Bruno Gonçalves, Alessandro Flammini, and Filippo Menczer. Detecting and Tracking Political Abuse in Social Media. in ICWSM. 2011.

[4] Michael D Conover, Bruno Gonçalves, Alessandro Flammini, and Filippo Menczer, Partisan asymmetries in online political activity. EPJ Data Science, 2012. 1(1): p. 1-19.

[5] Joseph DiGrazia, Karissa McKelvey, Johan Bollen, and Fabio Rojas, More tweets, more votes: Social media as a quantitative indicator of political behavior. PloS one, 2013. 8(11): p. e79449.

[6] Mohsen JafariAsbagh, Emilio Ferrara, Onur Varol, Filippo Menczer, and Alessandro Flammini, Clustering memes in social media streams. Social Network Analysis and Mining, 2014. 4(1): p. 1-13.

[7] Xiaoming Gao, Investigation and Comparison of Distributed NoSQL Database Systems.

[8] Xiaoming Gao and Judy Qiu, Scalable inverted indexing on NoSQL table storage. 2010.

[9] Xiaoming Gao, Vaibhav Nachankar, and Judy Qiu. Experimenting lucene index on HBase in an HPC environment. in Proceedings of the first annual workshop on High performance computing meets databases. 2011: ACM.

[10] Xiaoming Gao, Evan Roth, Karissa McKelvey, Clayton Davis, Andrew Younge, Emilio Ferrara, Filippo Menczer, and Judy Qiu, Supporting a Social Media Observatory with Customizable Index Structures: Architecture and Performance, in Cloud Computing for Data-Intensive Applications. 2014, Springer. p. 401-427.

[11] Karissa McKelvey and Filippo Menczer, Design and prototyping of a social media observatory, in Proceedings of the 22nd international conference on World Wide Web companion. 2013, International World Wide Web Conferences Steering Committee: Rio de Janeiro, Brazil. p. 1351-1358.

[12] Xiaoming Gao and Judy Qiu. Social Media Data Analysis with IndexedHBase and Iterative MapReduce. in Proc. Workshop on Many-Task Computing on Clouds, Grids, and Supercomputers (MTAGS 2013) at Super Computing. 2013.

[13] Alan F. Gates, Olga Natkovich, Shubham Chopra, Pradeep Kamath, Shravan M. Narayanamurthy, Christopher Olston, Benjamin Reed, Santhosh Srinivasan, and Utkarsh Srivastava, Building a high-level dataflow system on top of Map-Reduce: the Pig experience. Proc. VLDB Endow., 2009. 2(2): p. 1414-1425.

[14] Ashish Thusoo, 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. Proc. VLDB Endow., 2009. 2(2): p. 1626-1629.

[15] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia, Spark SQL: Relational Data Processing in Spark, in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. 2015, ACM: Melbourne, Victoria, Australia. p. 1383-1394.

[16] Tak-Lon (Stephen) Wu, Bingjing Zhang, Clayton Davis, Emilio Ferrara, Alessandro Flammini, Filippo Menczer, and Judy Qiu, Scalable Query and Analysis for Social Networks: An Integrated High-Level Dataflow System with Pig and Harp, in Big Data in Complex and Social Networks, My T. Thai, Hui Xiong, and W. Wu, Editors. 2015.

[17] Tak-Lon Wu, Abhilash Koppula, and Judy Qiu. Integrating Pig with Harp to support iterative applications with fast cache and customized communication. in Proceedings of the 5th International Workshop on Data-Intensive Computing in the Clouds. 2014: IEEE Press.

[18] Twitter Inc.; Available from: https://twitter.com/.

[19] Mark EJ Newman, Finding community structure in networks using the eigenvectors of matrices. Physical review E, 2006. 74(3): p. 036104.

[20] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara, Near linear time algorithm to detect community structures in large-scale networks. Physical Review E, 2007. 76(3): p. 036106.

[21] Dataflow programming; Available from: https://en.wikipedia.org/wiki/Dataflow_programming.

[22] Christopher Olston, 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. 2008, ACM: Vancouver, Canada. p. 1099-1110.

[23] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. in Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on. 2010: IEEE.

[24] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica, Shark: SQL and rich analytics at scale, in Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. 2013, ACM: New York, New York, USA. p. 13-24.

[25] Apache Tez, 2014; Available from: http://tez.incubator.apache.org/.

[26] Xiaoming Gao and Judy Qiu, Supporting End-to-End Social Media Data Analysis with the IndexedHBase Platform. 2013.

[27] Matthias Jarke and Jurgen Koch, Query optimization in database systems. ACM Computing surveys (CsUR), 1984. 16(2): p. 111-152.

[28] Johann Christoph Freytag, A rule-based view of query optimization. Vol. 16. 1987, ISBN: 0897912365: ACM.

[29] Alon Y Levy, Inderpal Singh Mumick, and Yehoshua Sagiv. Query optimization by predicate move-around. in VLDB. 1994.

[30] Surajit Chaudhuri. An overview of query optimization in relational systems. in Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. 1998: ACM.

[31] Spark SQL; Available from: https://spark.apache.org/sql/.

[32] PortageSQL; Available from: http://www.postgresql.org/.

[33] MySQL; Available from: https://www.mysql.com/.