Scalable Query and Analysis for Social Networks: An Integrated High-Level Dataflow System with Pig and Harp

Tak-Lon Wu, Emilio Ferrara, Judy Qiu

School of Informatics and Computing   
Indiana University, Bloomington, IN, USA   
taklwu, ferrarae, xqiu@indiana.edu

*Abstract*

Every day, vast amounts of data are being collected from social network applications, and in response there is a growing need r analysis methods that can handle this terabyte-size input. To provide an effective data processing environment, we need to support both query and complex analysis efficiently. Use of high-level scripting languages to solve Big Data problems has become a mainstream approach for sophisticated data mining and analysis. In particular, high level interfaces such as Pig, Hive, and Spark SQL are being used on top of the Hadoop framework. This simplifies coding of complex tasks in MapReduce-style systems while improving the flexibility of database systems through user-defined aggregations. In this chapter we will compare different approaches of building a high-level Dataflow Systems and propose an integrated solution with Pig and Harp (plugin to Hadoop) along with give extensive benchmarks.

1. Introduction

Social media is one important application which provides tremendous data from numerous social activities, with thousands of terabyte-level information streams going into storage. Many research projects are involved in performing intensive analysis on such data, and the outcome of this daily analysis is drawing the attention of various interests, from market sales analysts, social activities (including political polarization [1], congressional elections [2, 3], protest events [4, 5], and the spread of misinformation [6, 7]) and human sentiment diffusion [8] analysis. Compared to other fields of computing problems, social media analysis is special; it normally focuses on a subset of data related to a targeted social event within a specific time frame. To further investigate the inter-relationship of such subsets of data, various sophisticated algorithms and complex data transformations may be applied into a series of stages [9]. Therefore, developing a programmable solution for this social media data must include features like expressiveness, ability for data extraction, reusability and interoperability with different computation runtimes. Apache high level languages and Apache Hadoop [10] ecosystem are some of the existing building block solutions that match the requirements for social network analysis.

The use of high level language platforms is not just limited to social media data. Other fields of research such as workflow provenance [11], network traffic analysis [12, 13], and geographic data analysis [14] have proved the adaptation of these solutions boosts and scales up their historical data analysis. However, the lack of understanding of existing platforms makes it difficult for users to decide what language and low level runtimes best match their needs. Motivated by this absence, our goal is to provide a comprehensive survey of these high level abstractions involving experiments with real social media data examples and common mining applications.

The rest of the paper is organized as follows. Section 2 introduces the background of Hadoop and its ecosystem. It is impressive that there are some many interlocking tools in the Apache stack and of course Hadoop is best known example. In this section we also show how one can introduce plugins to improve performance and add functionality. Section 3 gives an overview of Apache high level languages, especially Pig [15], Hive [16] and Spark SQL [17, 18]. The first two build on Hadoop while Spark is an Apache iterative MapReduce offering an important different approach to parallel data systems. Section 4 provides a comparison of features of these languages – especially the important user-defined functions which make the MapReduce approach so powerful. Sections 5 and 6 introduce applications that are used for benchmarking in section 7. Section 5 introduces the Truthy project and types of queries that it needs to run on top of Twitter data. Section 6 discusses three data analytics use cases and how to express them in high level languages. Section 7 presents the performance evalutaion of the applications of sections 5 and 6 and the technologies of section 2 and 3. Section 8 is our conclusion.

1. Hadoop and Hadoop Ecosystem

Google introduced three key technologies that formed the foundation for their search engine infrastructure: Google File Systems (GFS), MapReduce and BigTable. Another well-known open source project is Hadoop. Hadoop [10] MapReduce was inspired by Google MapReduce [19]; it aims to support very large-scale data processing by decomposing data into partitions that execute in parallel on distributed commodity hardware. Hadoop Distributed File System (HDFS) [20] was first released in September 2007 as a Java-based open-source project that provides the interfaces for large-scale parallel implementations of algorithms and applications. Over the years, the Hadoop community has grown rapidly, and now Apache has many new open source projects contributed to this flourishing ecosystem. We will discuss a few key projects related to Hadoop.

* 1. Apache Hadoop Yet Another Resource Negotiator (YARN)

Apache Hadoop YARN [21] is a resource management system that provides the possibility for Hadoop ecosystem to run other runtimes or frameworks on a dedicated shared Hadoop cluster. Key idea are container-based tasks scheduling and system resource management interface. When a runtime requests resources via YARN’s API, the entire resource allocation for uch a runtime is handled by YARN, and overall computation resource usages on each compute node are monitored. Figure 1 shows an example of simultaneously running a MapReduce application and MPI application on the same shared cluster which is manged by YARN.

YARN splits the system resources into three categories: CPU cores, memory, and disk. Any runtime executed on YARN must run a YARN client program that extends from the resource manager interface. This client program communicates with YARN’s resource manager and requests resources specified in the program profile, e.g. number of containers, CPU and memory setting per container, etc. If the resource requests are accepted, YARN will start the specified application master(s) which dynamically negotiates with the resource manager to obtain access to resources (including scale up and down) and monitors the overall resource usages of containers via the NodeManager.

|  |
| --- |
|  |
| 1. YARN architecture: Hadoop and MPI runs on the same YARN cluster [22] |

* 1. Hadoop MapReduce and HDFS

As shown in Figure 2, the architecture of Hadoop MapReduce follows a traditional centralized server-client (master-slave) model. In Hadoop 2.0, the master node runs NameNode for HDFS, which handles most of the administrative work such as high-level metadata information handling, uploaded data splitting and distributing. Meanwhile YARN resource manager oversees job scheduling. Other compute nodes in the cluster run DataNode and NodeManager services which store the split data blocks, container allocation and scheduling, and monitor executing containers and their resource consumption. Application Master(s) is a replacement for Hadoop version 1.X’s JobTracker process. It negotiates resources from the Resource Manager and communicates with NodeManager to execute and monitor containers.

A MapReduce job consists of three different stages: Map, Shuffle, and Reduce. Map stage constructs the input data into key-value pair format, where ‘key’ is normally represented as a unique name and is used to perform in-memory local sorting. Shuffle stage takes these sorted key-value pairs and sends each record to an assigned reducer. Finally, reducers combine collected pairs and yield meaningful results to disk. Figure 3 depicts a process of data transformation in MapReduce.

* + 1. Data locality and Job scheduling

Data locality is one of the key features of MapReduce. For each split data block, their locality information is stored as metadata in a NameNode. Application Master utilizes this location information to assign “close to data” computation, which reduces the cost of transferring data over the network. Several researchers [23] have proven that with this key feature, data locality scheduling, MapReduce runs faster than random or fairly task scheduling.

|  |  |
| --- | --- |
|  |  |
| 1. Hadoop MapReduce Architecture | 1. Overview of MapReduce data processing [19] |

* 1. HBase

HBase [24] is an Apache open source key-value columnar database derived from Google BigTable [25] data model. Figure 4 gives an example of an HBase table which has two column families. Data in HBase tables is stored on HDFS, where each table has rows and columns. An HBase table contains multiple rows with one or more column families, each column family consists of one or many column qualifiers (name). Each row can be retrieved by a unique key (rowkey), and a cell value is the combination of rowkey, column family and column qualifier. Each cell of the same key can be overwritten with a different value and another timestamp, which is represented as record versioning control.

|  |
| --- |
|  |
| 1. HBase table data model |

The HBase architecture in Figure 5 has four crucial components: HDFS, Zookeeper(s), master service, and region servers. HDFS is the underlining file system used by HBase, and each data block on HDFS is part of an HBase Table stored as a series of data lines which contain ordered row keys, time stamps and columns. ZooKeeper [26] is a centralized service that maintains the META table of HBase and coordinates the machine (master and region servers) state consistency. Here, META is a single system table that keeps track of regions and their start keys in order to find the data location served by region servers. Each region is responsible for a range of row keys, and a region server maintains a set of regions. When a client query makes a request for a specific row key, data location lookup is handled by ZooKeeper and data scanning happens directly to the assigned region server. Meanwhile, HBase Master (HMaster) coordinates the HBase Cluster and is responsible for administrative operations such as region splitting and rearranging region location when a region server is down.

|  |
| --- |
|  |
| 1. HBase Architecture |

* + 1. IndexedHBase

IndexedHBase [9, 27] was a research project motivated by the widespread adoption of social media data, such as Facebook feeds [28] and Twitter tweets [29]. Many research involoes analyzing this social network data [30] but there’s a of lack fast searching support at scale. IndexedHBase is a storage system built with a customizable fast indexing framework that directly deploys on top of HBase for supporting fast data scanning, customizable MapReduce queries and sophisticated data analysis on the subset of data stores in HBase and HDFS.

|  |
| --- |
|  |
| 1. IndexedHBase Architecture [9] |

The dataflow of IndexedHBase is shown in Figure 6. A developer of IndexedHBase can load a set of raw data, e.g. Twitter tweets, into HBase as tables with user defined schemes; at the same time, the indexing module builds index tables based the configuration of where a user provides a list of fields of to build as inverted index (e.g. screen name and user id). Once the raw data table and index data are written to HBase, a query engine or command-line tool can be used to search and retrieve information of interests for further analysis with the help of runtimes that are compatible with YARN.

* 1. Harp

Harp [31] is a Hadoop plugin that enables loop awareness, fast in-memory caching, and collective communication patterns for iterative computation; an architecture diagram is shown in Figure 7. As illusrated in Figure 8, it replaces the default mapper interface with a long-running mapper that supports multi-threading and in-memory caching. Compared with process-based task scheduling in Hadoop, it can handle large intermediate data more efficiently in a shared memory. Harp provides MPI-like collective communication interfaces for customized network-based shuffling, in addition to disk-based shuffling with HDFS. These new features enable desirable processing capabilities and high performance for data intensive applications.

|  |  |
| --- | --- |
|  |  |
| 1. Architecture of Harp [31] | 1. Parallelism Model for Harp [31] |

* 1. Apache Tez

Apache Tez [32] is an Apache incubator project that optimizes Pig or Hive script compilers to construct a complex DAG dataflow (originally compiled as multiple MapReduce jobs) into a single job which boosts the performance and reuses the same set of mappers and reducers. However, this approach does not support loop-aware computation and in-memory caching from the default Pig or Hive language syntax, and the Pig community has started adding Tez support from the official release for version 0.14.0.

1. Apache high level language, syntax and its common features

Programming language has been developed for more than 50 years. Each language must have its own compiler and execute a physical plan on top of the low level (operating) system. Apache high level languages share the common features of the traditional programming languages. In many cases, a compiler built for such a language must support several key fundamental functions and operations: a syntax parser, type and compile time semantic checking, logical plan generator and optimizer, and physical plan generator and executor. Here ANTLR (ANother Tool for Language Recognition) [33] is the general syntax parser for Pig, Hive, and Spark SQL. Each language has its own types and plan generator and optimizer, but all of them use YARN as their resource management tool.

We’ll discuss details of Apache Pig, Apache Hive and Apache Spark SQL in the next section.

* 1. Pig

Pig is a high level dataflow system which composites simple data transformations in pipeline for large amounts of semi-structured data stored in Hadoop compatible file storage. Applications such as massive system log analysis and traditional Extract, Transform, and Load (ETL) data processing are performed regularly. Pig was first introduced by Yahoo!, and became one of the most popular Hadoop ecosystem projects in the Apache open source community. It uses its built-in procedural language, Pig-Latin, designed for large-scale data analysis with Hadoop MapReduce. The syntax is very straightforward, so long as the developer is familiar with UNIX bash script. Pig hides complicated MapReduce programs with simple notations for a dataflow program. Internally, Pig scripts are compiled into sequences of MapReduce jobs, which automates parallelization and makes the code easy to maintain. Pig also provides an interactive shell interface named GRUNT that generates MapReduce jobs which depend on the type-in lines. Figure 9 depicts an overall architecture of Pig. As shown, Pig is standalone and can run as a Java client on any worker node.

|  |
| --- |
|  |
| 1. Pig’s architecture |

* + 1. Pig Internal Operation Flow

When a user submits their Pig scripts in a batch mode or enters line-by-line data transformation commands in an interactive mode, a default compiler handles the overall execution flows. This compiler translates the entered Pig scripts into operators and forms top-down Abstract Syntax Trees (AST) in different stages. It then visits the last compiled AST from the MapReduce operators plan compiler and constructs MapReduce jobs in order. Figure 10 shows the dataflow and lists all major steps. Similar to any programming language, Pig checks syntax by parsing the user-submitted script into a parser written in ANTLR. It then generates a logical LOP (Logical Operator Plan) for further optimization. Generally a logical rules-based optimization is performed without looking at the real data (this is different from traditional SQL or SQL-like technologies that take data schema as part of the rules-based optimization). Pig’s main driver program converts each MapReduce operator from Map-Reduce Operator Plan (MROperPlan) objects into Hadoop JobControl objects with detailed descriptions, input/output linkages, and other parameters, which are then passed along to each worker node with a configuration in xml format. These translations generate Java .jar files that contain the Pig default Map and Reduce classes, including the user-defined functions. The packages of .jar files are submitted to Hadoop Job Manager, and job progress is monitored until completion of the tasks.

|  |
| --- |
| fig2-v2 |
| 1. Pig High Level Dataflow |

* + 1. A Pig example

Figure 11 shows an example of a WordCount program written in Pig Latin [34]. In a Pig dataflow, each line of code has only one data transformation, which can be nested. The WordCount program consists of seven lines of code, and the syntax is straightforward and easy to understand. Generanlly, data is loaded as records in a relation/outer bag, and each field in a record is defined according to Pig’s default data types: bag, tuple, and field. A bag is a set of unordered columnar tuples. A tuple is a set of fields, where tuples in a bag can contain flexible length of fields, and fields in the same column can have different data types. Lastly, a field is the basic type of a piece of data. Then, based on the supported data types, developer applies the desired data transformation and generate their results.

In our shown example, the first line defines an outer bag input and loads a text document from HDFS, each line of this text file are declared as string (chararray in Pig Latin). The second and third line further converts each line into English words and create each individual word a single tuple by using the built-in function TOKENIZE and relational statement FILTER. The fourth line aggregates exact same word together, and constructs a two cells tuple for each word. Here, the first cell of this tuple stores a text of this word, where second cell stores a list of same word and list size is the total occurrences of this word. Line fifth counts the amount of word item in list and emit a word count pair for each word <word, occurrences>. Line sixth uses the built-in order statement and reorder the wordcount result with descending order, and finally, line seventh stores the ordered result into default file storage.

Other than the syntax shown in this paper, Pig provides operations and syntax patterns for various data transformations, although the current version of Pig does not support optimized storage structures such as indices and column groups.

|  |
| --- |
| 1 input = LOAD 'input.txt' AS  (line:chararray);  2 words = FOREACH input GENERATE  FLATTEN(TOKENIZE(line)) AS word;  3 filWords = FILTER words BY word MATCHES '\\w+';  4 wdGroups = GROUP filWords BY word;  5 wdCount = FOREACH wdGroups GENERATE group AS  word, COUNT(filWords) AS count;  6 ordWdCnt = ORDER wdCount BY count DESC;  7 STORE ordWdCnt INTO 'result'; |
| 1. WordCount written in Pig [35] |

* + 1. Pig in Support for Iterative applications

Pig does a good job for ETL applications, but it does not directly support for iterative computations. This implies that Pig can execute simple one pass algorithms but does not support complex functions that need to apply a computation repeatedly (e.g. for loop) which exists in graph, linear algebra, expectation and maximization computations. To write such general data analysis applications using Pig, the control flow should be similar to what is shown in Figure 12. An external wrapper script is required, because Pig syntax does not provide control flow statements. This causes extra overhead of job startup and cleanup time when a program runs in several rounds of MapReduce jobs. Furthermore, inputs of iterative applications are normally unchanged and cacheable between iterations, whereas Pig has a DAG framework that does not cache those inputs in memory and reuses data efficiently.

|  |  |
| --- | --- |
| fig3-v2 |  |
| 1. Iterative applications with Pig | 1. Iterative application with Pig+Harp |

* + 1. Pig with Harp

To generalize the usage of Pig for scientific applications, we need to enable loop-awareness computation and in-memory caching; our research project investigated a version of Pig for scientific applications based on the DAG computation model. There are several iterative MapReduce frameworks available as candidates to integrate with Pig, which include Twister [36], Spark [37], HaLoop [38], and Harp. We chose Harp as it is a an plug-in to Hadoop that supports our required iteration features, the result being referred to from here on as Pig+Harp [39]. With Harp integration, we replace the Hadoop Mapper interface with Harp’s MapCollective, a long-running mapper to support conditional loops. Subsequently, iterative applications implemented in Pig+Harp can cache reusable data and replace the default GROUP BY operation with Harp’s collective communication interface featuring high performance data movement. Figure 13 shows a dataflow that can be applied to iterative applications

* 1. Hive

Hive [16] is a data warehouse solution for ad-hoc queries from simple data summarization to business intelligence applications; and high-latency queries for extremely large structured data sets stored on top of Hadoop related file storage. Initially developed by Facebook’s data infrastructure team, it is used for filtering summarization information from their massive amount of stored social network data and support products associated with the collected data. Thousands of Hive jobs were submitted daily since 2010 [16]. Hive uses a SQL-like language named HiveQL which is very attractive for the traditional SQL community. Similar to Pig, HiveQL queries are compiled into MapReduce jobs and executed on top of Hadoop. Hive reintroduces a RDBMS technique - Metastore - that stores data schemas and statistics as a service of an in-memory system catalog to facilitate Hive’s compiler and data scanning. Figure 14 shows the architecture of Hive.

|  |
| --- |
|  |
| 1. Hive Architecture |

* + 1. Hive Internal Operation Flow

When a user submits HiveSQL statements via any supported APIs, Hive initially checks the syntax by an ANLTR parser, then cooperates with Metastore for further type checking and semantic analysis, and lastly generates an initial AST as a logical plan. This plan is then optimized through a rule-based optimizer involving the schema and indices metadata obtained from Metastore. Optimizations such as column pruning, pushdown, partition pruning, mapside joins, and join reordering are also performed. Finally, a physical plan is generated from the optimized logical plan and submits a sequence of MapReduce jobs to Hadoop cluster.

* + 1. A Hive Example

Figure 15 shows a WordCount program written in HiveQL in which only three lines of code are involved. Hive supports nested statements, and each statement represents a single data transformation. As shown in Figure 15, the first line declares a table named doc with only a string column line. The second line reads files from the given path and overwrites the table doc. The third line is a nested statement that splits all words of each record of lines in table ‘doc’, then it groups all emitted (word, 1) pairs from the temporary table ‘words’ and orders it decreasingly with their occurrence. As shown below, the overall syntax is very SQL friendly.

|  |
| --- |
| 1. CREATE TABLE doc (line STRING);  2. LOAD DATA INPATH '$documentsPath' OVERWRITE INTO TABLE doc;  3. INSERT INTO OVERWRITE DIRECTORY '$outputPath'  SELECT word, count(1) AS count FROM  (SELECT explode(split(line, '\s')) AS word FROM doc) words  GROUP BY word  ORDER BY word; |
| 1. WordCount written in Hive |

* + 1. Hive with HBase

By default, Hive is compatible with local file system, HDFS and HBase. A user is required to provide data schema by creating tables before accessing the files in storage. For instance, prior to reading existing tables in HBase, users need an additional step to make tables in Metastore and link the schema of Hive to the HBase tables, such as row key and column families of the reading tables.

* 1. Spark SQL/Shark

Spark SQL [18] or Shark [17] is another open source project directly inspired by the Hive projects. Both use Spark runtime and RDD [40] as the core engine to execute their physical plan on top of YARN. Spark SQL is the latest release replacing Shark, now merged as a branch project under Spark’s ecosystem.

Spark SQL reuses Hive’s query parser to generate a logical operator plan. With this compatibility support, general Hive queries can run on Spark SQL without any changes to the execution script. Spark SQL has its own ruled-based logical operator plan optimizer for matching the physical operators that run on Spark. As claimed by Spark runtime developers, this allows Spark SQL queries to run better on RDD operations and best match the Spark execution model, rather than tuning Spark low-level execution to support Hive’s Hadoop implementation. Figure 16 shows the architecture of Spark SQL.

|  |
| --- |
|  |
| 1. Spark SQL architecure [18] |

* 1. Spark SQL’s limitation in supporting Hive statements

Spark SQL can support most of the HiveQL statements with several limitations; e.g., bucketed tables in Hive are not currently supported in Spark SQL.

1. Pig, Hive and Spark SQL Comparison

Before comparing the differences among between Pig, Hive and Spark SQL, we need to look into two fundamental terms: dataflow and data warehouse system.

Dataflow system is a type of data processing system where data is transformed from one format to another via different processing units in a directed path. Data can be structured (with a predefined schema) or unstructured (e.g. logs); it therefore requires customized data selection and operations to extract meaningful information. Pig falls into this category. Data warehouse is a system that handles cleaned, structured and cataloged data in organized hierarchical data storage units. Data is available to observers for conducting data analysis. Hive and Spark SQL are designed for the data warehouse community.

Table I gives a comprehensive comparison between Pig, Hive and Spark SQL. Even though they are designed for different systems and applications, these three tools share many common features, operations and functions.

Pig can be used for unstructured raw data batch processing and simple statistical analytics, especially massive logs and text mining. It is more like an alternative to Hadoop MapReduce applications in high level abstraction with an extensible subset of general data operations. Data is stored in HDFS or HBase with high-latency data scanning operations. Pig scans data entirely (e.g. must scan all data for filtering fields with numeric type less than 10) without the help of data indices. As such it is considered ‘slower’ in supporting ad-hoc queries than Hive.

Meanwhile, Hive and Spark SQL are SQL-like distributed systems that run high-latency queries for data sets stored on top of MapReduce (HDFS) file system. Hive still scans data from disk or HDFS directly for assigned map/reduce tasks. Metastore provides the data schema and indices while data scanning. Spark SQL uses Hive’s query parser and Metastore generates the operator plan, but it then uses Catalyst as its logical plan generator and optimizer (Shark uses Hive’s query planner), executes with Spark, and stores the processed/queried data into DataFrame (columnar RDD in Shark) instead of files on HDFS. Here rule-based optimizations of Hive/Shark and Spark SQL are expandable. The use of RDD provides in-memory reusable access to the scanning data. It saves plenty of disk I/O and jobs restart overhead if the data is hit frequently, especially when the cases of mixing ad-hoc queries and further sophisticated applications are involved within the same Scala script. Spark SQL is still a newly released ongoing project, so some query plan optimizations of Hive/Shark are not included in Spark SQL such as block level bitmap indices and virtual columns. Catalyst is the core difference between Shark and Spark SQL. DataFrame is a special type of row objects RDD that has associated data schema such as column field name and data type as a collection of named and typed tuples. It can then support operations from the submitted relational queries in line. In addition, with the help of the known type of the row objects, DataFrame can be cached with better compression than general RDD objects.

* 1. User-Defined Function

All of Pig, Hive and Spark SQL introduce User-Defined Functions (UDF) for advanced tuple/record-based data transformation, which enables the possibility to implement special computation and sophisticated algorithms in addition to the basic queries.

TABLE I. CROSS COMPARSION FOR PIG, HIVE AND SPARK SQL

|  |  |  |  |
| --- | --- | --- | --- |
|  | Pig | Hive | Spark SQL |
| Target system | Dataflow | Data warehouse | Data warehouse, then data analytic applications |
| Syntax | Pig Latin | HiveQL (SQL-like) | HiveQL (SQL-like) |
| Script parser | ANTLR | ANTLR | ANTLR |
| Logical Plan compiler | Script -> AST -> Operator Trees | Script -> AST -> Operator Trees  (DML DDL by tables) | Catalyst |
| Logical Plan optimizer | Operators Trees (Rules based) | Operator Trees (Rules based) | Operator Trees (Rules based) |
| Physical / MR compiler | Operators Trees -> MR jobs | Operators Trees -> MR jobs | Operators Trees -> Spark jobs on YARN |
| Structured or unstructured data | Unstructured, structured, nested Structured raw data | Structured tabular data | Structured tabular data |
| Catalog services | HCatalog (Optional) | Metastore and HCatalog | Metastore and HCatalog |
| Primitives data type | INT, LONG, FLOAT, DOUBLE, CHARARRAY, etc. | TINYINT, SMALLINT, INT, BIGINT, FLOAT, DOUBLE, etc. | ByteType, ShortType, IntegerType, LongType, FloatType, DoubleType , etc.  And most of Hive’s Primitives DataType |
| Non-Primitives data type | map, tuple, bag | maps, arrays, structs, union | ArrayType, MapType, StructType |
| Relational Statements | GROUP, DEFINE, FILTER, FOREACH, JOIN, UNION, ORDER BY, SAMPLE, etc. | SELECT, GROUP BY, ORDER BY, CLUSTER BY, DISTRIBUTE BY, JOIN, UNION, TABLESAMPLE, etc. | SELECT, GROUP BY, ORDER BY, CLUSTER BY, JOIN, UNION, TABLESAMPLE, etc. |
| Math operators | ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, MODULO, etc. | ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, MODULO, etc. | ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION, MODULO, etc. |
| Logical Operators | AND, OR, IN, NOT, EQUAL, NOT EQUAL, LESS THAN, GREATER THAN, PATTERN MATCHING | AND, OR, NOT, IN, EQUAL, NOT EQUAL, LESS THAN, GREATER THAN, PATTERN MATCHING, EXISTS, IF, COALESCE, CASE | AND, OR, NOT, IN, EQUAL, NOT EQUAL, LESS THAN, GREATER THAN, PATTERN MATCHING, EXISTS, IF, COALESCE, CASE |
| Collection and Aggregate Functions | AVG, SUM, COUNT, CONCAT, MAX, MIN, SIZE, SUBSTRACT, etc. | AVG, SUM, COUNT, CONCAT, MAX, MIN, SIZE, SUBSTRACT, etc. | AVG, SUM, COUNT, CONCAT, MAX, MIN, SIZE, SUBSTRACT, etc. |
| String functions | Yes | Yes | Yes |
| DateTime Functions | Yes | Yes | Yes |
| UDF support | Yes | Yes | Yes (partially Hive UDF) |
| JDBC/Thrift support | Partial (No Thrift API) | Yes | Yes |
| Index Table | No | Yes | Yes |
| Storage Layer | Local Disk, HDFS, HBase | Local Disk, HDFS, HBase (Optional) | Local Disk, HDFS, HBase (Optional) |
| Applications | Data filtering, ETL, log analysis, general statistic applications, text processing | Ad-hoc queries, ODBC/JDBC applications, high-latency queries | Ad-hoc queries, ODBC/JDBC applications, low-latency queries |

1. Ad-hoc queries – Truthy and Twitter data

Ad-hoc queries are the most common benchmark for ETL applications. This also applies on Apache high level languages, which are mainly designed for supporting ETL operations. Here we use Truthy project and Twitter social media data to evaluate the general ad-hoc query performance among these runtimes.

* + 1. Truthy project

Truthy [30] 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’s daily data directly through Twitter’s public steaming API. The overall size of historical raw data from 2010 till April 2015 is about 3.2TB. IndexedHBase is used to store, load, and index this data as tables into HBase on a private large storage, high performance, and large memory cluster MOE. As of today, the overall data including the raw data tables and index tables (includes the standard 3 replicas) on HBase is near 133 TB. We expect to continue store more historical data, and the Truthy group aims to perform innovative and large-scale social network research and analysis to understand how information propagates through this complex socio-technical data. For instance, many researchers [2-7] have built their prototype, model and analysis results based upon this huge and complicated infrastructure, which shows the capability to capture the spread of information, from pontifical discourses to commercial sales trends, from social topics to scientific analysis, with many unexpected findings.

The data collected from Twitter streaming API is mostly Twitter tweets with various fields of attributes, and the most common attributes used for intensive analysis are hashtags, user information, text and media content, retweets information, user mentions information, and specific time interval of social events. Truthy identifies this information and utilizes the concept of ‘meme’, a piece of data that corresponds to specific topics, communication channels, or shared elements by people in a social network, to construct a set of temporal queries for extracting tweets’ information for further data intensive analysis. These queries can be classified into two categories [27]: ad-hoc queries for simple tweet retrieval with the help of index tables, and combination of tweets retrieval with extra data transformation. We only discuss the ad-hoc queries here because it is the best practice for matching the common features of high level languages. Table II shows four different queries that firstly search the related index table and then redirect the obtained tweets back to user.

TABLE II. Truthy’s Ad-hoc queries for simple tweets retrieval

|  |  |  |
| --- | --- | --- |
| Queries | Description | Index Table Name |
| get-tweets-with-meme | Search tweets with given memes such as hashtags, user-mentions, and URLs | memeIndexTable |
| get-tweets-with-text | Search tweets with given keywords | textIndexTable |
| get-tweets-with-user | Search tweets with given user information, e.g. user id and screen name | userTweetsIndexTable |
| get-retweets | Search retweets with given tweet Ids | retweetIndexTable |

1. Iterative Scientific Applications

Many domain scientists who work on scientific applications use programming languages such as Python, Matlab and R to do daily analysis. Often such analysis involves sophisticated data mining and machine learning techniques that must run in several rounds of computation to complete a full task. With the built-in mathematics and statistical operations in Pig and Hive, these two runtimes could be candidate tools to support scientific applications run with very large-scale data. Due to the fact that Pig and Hive do not support iterative applications directly, we need to extend them with an external wrapper script/program to handle the loop control and link the program inputs from HDFS between iterations. On the other hand, Pig+Harp integrated Harp’s collective communication API to support iterative application as well as single pass in common Hadoop jobs for both high level language tool. Here we present two popular algorithms, K-means clustering and PageRank computing, to show the difference if implementations by using Pig and Hive.

* 1. K-means Clustering

Pig K-means consists of three components: a python control-flow script, a Pig data-transform script for a single iteration, and two K-means user-defined functions written with a Pig-provided Java interface. Figure 17 shows a single iteration of K-means written in Pig Latin. During each iteration, our customized Loader in each Mapper loads the aggregated centroids into memory as vector objects from the distributed cache on disk before computing the Euclidean distances for data points. Each loader outputs assigned centroids and data points as fields in a single bag; each field in a bag is defined as string data type which further splits into tuples to match Pig’s GROUP operation and collect partial centroid vectors from mappers. It takes the average of all partitions, emits to a final centroids file and saves it to HDFS.

|  |
| --- |
| 1 raw = LOAD $hdfsInputDir using  PigKmeans('$centroids',  '$numOfCentroids') AS (datapoints);  2 dptsBag = FOREACH raw GENERATE  FLATTEN(datapoints) as dptInStr;  3 dpts = FOREACH dptsBag GENERATE  STRSPLIT(dptInStr, ',', 5) AS  splitedDP;  4 grouped = GROUP dpts BY splitedDP.$0;  5 newCens = FOREACH grouped GENERATE  CalculateNewCentroids($1);  6 STORE newCens INTO 'output'; |
| 1. Pig K-means script for a single iteration |

Hive K-means is written in SQL-like syntax and uses a UNIX bash script as the loop conditions wrapper for supporting iterations. Figure 18 depicts a single iteration of K-means written in HiveQL. Data points and centroids are originally stored on HDFS during each iteration, and they overwrites the intermediate centroids to HDFS. The default INPUT\_\_FILE\_\_NAME field provided by Hive is used, and our K-means UDF directly loads entire files for computation instead of using Hadoop InputFormat with a series of input splits. General data aggregations such as GROUP BY and SUM are executed, while Euclidean distance computation is handled by the K-means UDF.

|  |
| --- |
| 1 CREATE EXTERNAL IF NOT EXISTS $INPUTTABLE (filename String)  LOCATION '$hdfsInputDir ';  2 DROP TABLE IF EXISTS interKmeansTable;  3 CREATE TABLE interKmeansTable(x double, y double, z double, beta double) LOCATION '$hdfsOutputDir;  4 INSERT OVERWRITE TABLE interKmeansTable  Select sum(KmeansTable.ret.x)/sum(KmeansTable.ret.count),  sum(KmeansTable.ret.y)/sum(KmeansTable.ret.count),  sum(KmeansTable.ret.z)/sum(KmeansTable.ret.count),  0.0  FROM  (Select explode(Kmeans(INPUT\_\_FILE\_\_NAME,  '$initCentroidOnHDFS', '$centroidSize')) as ret  FROM $INPUTTABLE T) KmeansTable  GROUP BY KmeansTable.ret.assignedcentroid;" |
| 1. Hive Kmeans script for a single iteration |
| 1 centds = LOAD $hdfsInputDir using  HarpKmeans('$initCentroidOnHDFS',  '$numOfCentroids', '$numOfMappers',  '$iteration', '$jobID', '$Comm') as  (result);  2 STORE centroids INTO '$hdfsOutputDir; |
| 1. Pig+Harp K-means script |

Pig+Harp K-means script in Figure 19 illustrates a similar idea using R. Users only provide the parameters, such as number of mappers, total amount of iterations, and communication patterns used for global data synchronization. In the case of executing Pig+Harp K-means, a customized Loader in each Mapper first loads the initial centroids and data points from HDFS to memory and cache the data points for all iterations. Then the UDF computes Euclidean distances and emits partial centroids locally. Harp’s communication layer then exchanges these partial centroids in each mapper. By default, HarpKmeans UDF uses AllReduce to synchronize among all partitions. The program reuses the same set of mapper processes until exit conditions have been reached. Note that users need to have a good understanding of Hadoop and Harp frameworks in order to achieve optimal performance.

* 1. PageRank

For Pig PageRank, we use a model with fewer UDF functions by leaveraing Pig built-in operators. Figure 20 shows a single iteration of the PageRank algorithm, which is created and iteratively invoked by a Java wrapper. The script involves the following steps: a) Load the given input file using the custom loader into variable raw; b) Extract the outgoing URLs and emit both outgoing URL and partial page rank from the source URL; c) CO-GROUP above two aliases to calculate new page rank and store it in an alias newPgRank; d) Store new page rank in a HDFS temp file, which will be the input file for the next iteration. One drawback of this program is that the default Pig runtime optimizer creates extra mappers for the final STORE step when it calls the raw and prePgRank variables for CO-GROUP operators, which utilizes extra computing and memory resources.

Hive PageRank follows a similar logic as Pig PageRank does, but the HiveQL script uses tables as data abstraction and nested queries for computation, as well as OUTER LEFT JOIN in Figure 21.

In Pig+Harp PageRank implementation, we write a new data loader UDF to calculate probabilities for each web page. For the first iteration, data is loaded in a graph data structure where vertices are partitioned across all worker nodes. Each vertex receives all in-edges information by calling regroupEdges collective communication, and the number of out-edges is sent to all vertices by calling an AllMsgToAllVtx operation. The vertex and edge information is cached in memory for all iterations. Subsequently the page rank values of each vertex are updated during each iteration, and distributed by an AllGather communication until the program satisfies break conditions, e.g. the end of iterations. The script shown in Figure 22 is similar to that of Pig+Harp K-Means.

|  |
| --- |
| 1 raw = LOAD '$InputDir' USING  CmLoader('$noOfURLs','$itrs') as  (source,pagerank, out:bag);  2 prePgRank = FOREACH raw GENERATE FLATTEN(out)  as source, pagerank/SIZE(out) as  pagerank;  3 newPgRank = FOREACH (COGROUP raw by source,  prePgRank by source OUTER)GENERATE  group as source, (1-$dpFactor) +  $dpFactor\*(SUM(prePgRank.pagerank)  IS NULL?0:SUM(prePgRank.pagerank))  as pagerank, FLATTEN(raw.out)  as out;  4 STORE newPgRank INTO '$outputFile'; |
| 1. Pig PageRank script for a single iteration |
| 1 CREATE EXTERNAL TABLE pageRankInput(line String)  location '$INPUTDIR';  2 CREATE TABLE PageRankComputeTable(pagerankCell struct<source:int,pagerank:double,outLinks:array<int>>)  CLUSTERED BY(pagerankCell)  INTO $MAP\_SIZE BUCKETS  location '$tmpPageRankResult';  3 INSERT OVERWRITE TABLE PageRankComputeTable  select InitialPageRank(line, '$numOfUrls') as ret FROM pageRankInput;  4 INSERT OVERWRITE TABLE PageRankComputeTable  SELECT named\_struct('source',T1.pagerankCell.source,  'pagerank', PageRank(T2.pagerank, $dpFactor, $noOfURLs),  'outLinks', T1.pagerankCell.outLinks) as cell  FROM  PageRankComputeTable T1  LEFT OUTER JOIN  (SELECT outlink,  sum(pagerankCell.pagerank/size(pagerankCell.outlinks)) as pagerank  FROM PageRankComputeTable  LATERAL VIEW explode(pagerankCell.outlinks) outLinkTable as outlink  Group by outlink) T2  ON (T1.pagerankCell.source = T2.outlink); |
| 1. Hive PageRank script for a single iteration |
| 1 pagerank = LOAD '$InputDir' using  HarpPageRank('$totalUrls',  '$numMaps', '$itrs', '$jobID')  as (result);  2 STORE pagerank INTO '$output'; |
| 1. Pig+Harp PageRank script |

1. Benchmark and evaluation

We have performed a set of extensive ad-hoc queries against Twitter’s social network data using these high level languages to illustrate their overheads and performance difference. We compare two scientific applications, K-means and PageRank, to evaluate the language expressiveness and performance in support of generic scientific algorithms in regards to high level data abstractions, operations and execution flows. Currently we are not able to perform Spark SQL tests as exisiting Spark SQL (latest version 1.3.1 as of April 2015) only supports a subset for HiveQL query and is best compatible with Hive 0.13.1. This limited compatibility causes our tests to fail. For example, when Spark SQL scans data from HBase, although the high level abstraction StringType is used, Spark SQL in low level execution retrieves HBase’s record as String instead of LazyString in Hive, which causes a data loss to our ad-hoc queries test cases.

We have provided our tests online at github: https://github.com/taklwu/apache-high-level-languages-survey

* 1. Machine Setting

Our experiments are conducted under cluster MOE for the Truthy project [30]. MOE is a large storage, large memory and high performance private cluster at Indiana University with 3 login nodes and 10 compute nodes. 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 five 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. Table III shows the specifications of MOE cluster.

TABLE III. HARDWARE SPECIFIATION OF MOE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CPU | RAM | Disk | Network |
| Login Node | 2 x Intel(R) Xeon(R) CPU E5-2620 v2 | 64GB | 120 SSD | 10Gb Ethernet |
| Computer Node | 5 x Intel(R) Xeon(R) CPU E5-2660 v2 | 128GB | 48TB HDD + 120GB SSD | 10Gb Ethernet |

YARN Hadoop cluster on MOE is configured with a master on an independent login node. Meanwhile HBase uses another login as the master node and runs a ZooKeeper on each login node. YARN’s NodeManager, HDFS’s DataNode and HBase’s RegionServer run on individual node and memory is shared among these processes. Table IV specifies the software and runtime settings of MOE.

TABLE IV. RUNTIME SOFTWARE SPECIFIATION OF MOE

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | YARN | Pig | Hive | Pig+Harp | HBase | IndexedHBase |
| Version | 2.5.1 | 0.14.0 | 1.0.0 | 0.14, 0.1.0 | 0.94.23 | 0.94 branch |
| Memory | 66GB per node | 2GB per worker | 2GB per worker | 2GB per worker | 30GB per node | 2GB per worker |
| Disk | 48TB per node | Data on HDFS | Data on HDFS | Data on HDFS | Data on HDFS | Data on HBase |

* + 1. Results of Ad-hoc Queries

Figure 23 shows the results of running Truthy’s queries on Twitter data using IndexedHBase. The query get-tweets-with-X initiates two steps; first it searches for tweet IDs from the related index table on HBase by given keys such as meme, text or user id under a specified time interval. The second step reuses the obtained tweet IDs to scan related tweets from the raw tweets table in HBase and stores the retrieved tweets on HDFS. The overall performance is dominated by the total amount of retrieved tweet IDs. Table V displays the number of records obtained from each query; we use hashtag ‘#ff’ as meme, keyword ‘NBA’ as text, and randomly choose a user ID to search tables in December 2012 dataset.

The results in Figure 23 show that the IndexedHBase API command-line script performs the best. This is because it calls an optimized Hadoop MapReduce job directly, and even the ‘search for tweet IDs’ step is run as a local processes. Pig and Hive solutions execute these two steps in MapReduce jobs, therefore their performance is similar. Hive requires more time for setting up the Table schema (includes DROP and CREATE statement) in Metastore and Hive related parameters in script for each query. As a result Hive performs the slowest in all our tests. Table VI lists the lines of code in script and the amount of submitted Hadoop jobs for each runtime based on query ‘get-tweets-with-X’.

|  |
| --- |
|  |
| 1. Truthy’s get-tweets-with-X queries on Twitter data |

TABLE V. SIZE OF RECORDS OBTAINED BY ‘get-tweets-with-X’

|  |  |  |  |
| --- | --- | --- | --- |
|  | get-tweets-with-meme | get-tweets-with-text | get-tweets-with-userid |
| # of Records | 1570261 | 202076 | 22 |

TABLE VI. SCRIPT AND EXECUTION COMPARSION OF ‘get-tweets-with-X’

|  |  |  |  |
| --- | --- | --- | --- |
| get-tweets-with-X | IndexedHBase CMD | Pig | Hive |
| Lines of code in script | 1 | 11 | 17 |
| Hadoop job(s) | 1 | 2 | 2 |
| Map(s)/Reduce(s) | 24/0 | 1/0, 24/0 | 1/24, 24/0 |

* + 1. Results of scientific applications

We useK-means and PageRank to evaluate the difference of performance. Both algorithms are implemented in the same dataflow logic but using different syntax in Pig, Hive and Pig+Harp implementations.

* + 1. Results of K-means

The tests for K-means algorithm is shown in Table VII, where we compute 10 iterations for two data sets: 100 million 3-dimensional data points against 500 centroids, and 100 million 3-dimensional data points against 5000 centroids. The dataset is split into 128 partitions, running 128 mappers and 8 reducers. Each mapper or reducer runs on 1 CPU core with 2GB memory.

Figure 24 shows the total execution time for the K-means algorithm with each runtime. Pig+Harp outperforms the other two runtimes due toin-memory objects cache for loaded data points and centroids, fast network I/O for data aggregation, and reduced overheads of the job restart between iterations. In contrast, Pig and Hive implementations have a huge cost due to reloading intermediate data points and centroids from HDFS at each iteration. The execution plans for Pig and Pig+Harp are similar. Pig K-means generates 1 Hadoop job per iteration and Pig+Harp K-means generates a single job for all iterations.

* + 1. Results of PageRank

In the PageRank test, we compute 10 iterations for two data sets: 1 million and 2 million numeric URLs. The data is split into 64 partitions, running 64 mappers and 64 reducers . Each mapper or reducer has 1 CPU core and 2GB memory.

Figure 25 presents the execution time of PageRank algorithmwhere Pig+Harp performs the best by storing the adjacency matrices as objects in memory, exchanging partial PageRank values via network I/O, and long running tasks.

As shown in Table VIII, a maximum of 128 mappers (rather than our expected 64 mappers) are invoked for the partitions. This is due to the use of LEFT OUTER JOIN both in Pig and Hive implementation, and each partition is separately loaded in an extra mapper and prepared for the JOIN operations. In the case of Hive PageRank, although the HiveQL logic is same as Pig’s, Hive’s physical plan executor generates a total of 4 Hadoop jobs per iteration, which results in a dramatic performance loss.

|  |  |
| --- | --- |
|  |  |
| 1. K-means result with running 128 mappers | 1. PageRank result with running 64 mappers |

TABLE VII. SCRIPT AND EXECUTION COMPARSION OF K-MEANS

|  |  |  |  |
| --- | --- | --- | --- |
| K-means | Pig+Harp | Pig | Hive |
| Lines of code in script | 3 | 11 | 13 |
| Hadoop job(s) per iteration | 1 | 1 | 1 |
| Map(s)/Reduce(s) | 128/0 | 128/8 | 128/8 |

TABLE VIII. SCRIPT AND EXECUTION COMPARSION OF PAGERANK

|  |  |  |  |
| --- | --- | --- | --- |
| PageRank | Pig+Harp | Pig | Hive |
| Lines of code in script | 3 | 8 | 16 |
| Hadoop job(s) per iteration | 1 | 1 | 4 |
| Map(s)/Reduce(s) | 64/0 | 128/64 | 64/64, 64/64, 128/64, 64/64 |

1. Conclusion

In this paper, we surveyed the Apache high level languages and several runtimes of Hadoop ecosystems. We conduct tests onreal world applications with social media data. Terabytes of data streams are collected every day and stored on different large scale storage systems such as HDFS and HBase. Pig, Hive, and Spark SQL have been widely adopted by developers and domain scientists for rapidly building their prototypes and performing daily analysis tasks on both new and historical data. The results presented in this paper shows that we can use these high level abstractions not only to run ad-hoc queries but also involves the appropriated execution engines to support multiple stage dataflow and a series of data aggregations on a variety of applications. These high level languages makes it easier to invoke data analysis when being used on top of Hadoop framework improves the flexibility of database systems through user-defined aggregations. We compare different approaches of building a high-level Dataflow Systems and propose an integrated solution with Pig and Harp (plugin to Hadoop) with our benchmarks to achieve high performance.

Reference

1. Conover, M., J. Ratkiewicz, M. Francisco, B. Gonçalves, F. Menczer, and A. Flammini. *Political polarization on twitter*. in *ICWSM* 2011.

2. DiGrazia, J., K. McKelvey, J. Bollen, and F. Rojas, *More tweets, more votes: Social media as a quantitative indicator of political behavior.* PloS one, 2013. 8(11): p. e79449.

3. Conover, M.D., B. Gonçalves, A. Flammini, and F. Menczer, *Partisan asymmetries in online political activity.* EPJ Data Science, 2012. 1(1): p. 1-19.

4. Conover, M.D., E. Ferrara, F. Menczer, and A. Flammini, *The digital evolution of occupy wall street.* PloS one, 2013. 8(5): p. e64679.

5. Conover, M.D., C. Davis, E. Ferrara, K. McKelvey, F. Menczer, and A. Flammini, *The geospatial characteristics of a social movement communication network.* PloS one, 2013. 8(3): p. e55957.

6. Ratkiewicz, J., M. Conover, M. Meiss, B. Gonçalves, S. Patil, A. Flammini, and F. Menczer. *Truthy: mapping the spread of astroturf in microblog streams*. in *Proceedings of the 20th international conference companion on World wide web* 2011: ACM.

7. Ratkiewicz, J., M. Conover, M. Meiss, B. Gonçalves, A. Flammini, and F. Menczer. *Detecting and Tracking Political Abuse in Social Media*. in *ICWSM* 2011.

8. Hutto, C. and E. Gilbert. *Vader: A parsimonious rule-based model for sentiment analysis of social media text*. in *Eighth International AAAI Conference on Weblogs and Social Media* 2014.

9. Gao, X. and J. 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.

10. Apache, *Apache Hadoop*. <http://hadoop.apache.org/>. <http://hadoop.apache.org/core/>.

11. Amsterdamer, Y., S.B. Davidson, D. Deutch, T. Milo, J. Stoyanovich, and V. Tannen, *Putting lipstick on pig: Enabling database-style workflow provenance.* Proceedings of the VLDB Endowment, 2011. 5(4): p. 346-357.

12. Hegeman, T., B. Ghit, M. Capota, J. Hidders, D. Epema, and A. Iosup. *The btworld use case for big data analytics: Description, mapreduce logical workflow, and empirical evaluation*. in *Big Data, 2013 IEEE International Conference on* 2013: IEEE.

13. Lee, Y. and Y. Lee, *Toward scalable internet traffic measurement and analysis with hadoop.* ACM SIGCOMM Computer Communication Review, 2013. 43(1): p. 5-13.

14. Aji, A., X. Sun, H. Vo, Q. Liu, R. Lee, X. Zhang, J. Saltz, and F. Wang. *Demonstration of hadoop-gis: A spatial data warehousing system over mapreduce*. in *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems* 2013: ACM.

15. Gates, A.F., O. Natkovich, S. Chopra, P. Kamath, S.M. Narayanamurthy, C. Olston, B. Reed, S. Srinivasan, and U. Srivastava, *Building a high-level dataflow system on top of Map-Reduce: the Pig experience.* Proc. VLDB Endow., 2009. 2(2): p. 1414-1425.

16. Thusoo, A., J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, *Hive: a warehousing solution over a map-reduce framework.* Proc. VLDB Endow., 2009. 2(2): p. 1626-1629.

17. Xin, R.S., J. Rosen, M. Zaharia, M.J. Franklin, S. Shenker, and I. 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. pages. 13-24. DOI: 10.1145/2463676.2465288.

18. *Spark SQL*. Available from: <https://spark.apache.org/sql/>.

19. Dean, J. and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters.* Sixth Symposium on Operating Systems Design and Implementation, 2004: p. 137-150.

20. Shvachko, K., H. Kuang, S. Radia, and R. Chansler. *The hadoop distributed file system*. in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* 2010: IEEE.

21. Vavilapalli, V.K., A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, *Apache Hadoop YARN: yet another resource negotiator*, in *Proceedings of the 4th annual Symposium on Cloud Computing*. 2013, ACM. Santa Clara, California. pages. 1-16. DOI: 10.1145/2523616.2523633.

22. Vavilapalli, V.K., A.C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, and S. Seth. *Apache hadoop yarn: Yet another resource negotiator*. in *Proceedings of the 4th annual Symposium on Cloud Computing* 2013: ACM.

23. Guo, Z., G. Fox, and M. Zhou, *Investigation of Data Locality in MapReduce*, in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. 2012, IEEE Computer Society. pages. 419-426. DOI: 10.1109/CCGrid.2012.42.

24. Apache. *Hbase implementation of Bigtable on Hadoop File System*. 2010 [accessed 2010 June 5]; Available from: <http://hbase.apache.org/>.

25. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and R.E. Gruber, *Bigtable: A Distributed Storage System for Structured Data*, in *OSDI'06: Seventh Symposium on Operating System Design and Implementation*. 2006, USENIX. Seattle, WA. <http://labs.google.com/papers/bigtable-osdi06.pdf>.

26. Hunt, P., M. Konar, F.P. Junqueira, and B. Reed. *ZooKeeper: Wait-free Coordination for Internet-scale Systems*. in *USENIX Annual Technical Conference* 2010.

27. Gao, X. and J. Qiu, *Supporting End-to-End Social Media Data Analysis with the IndexedHBase Platform.* 2013.

28. Borthakur, D., J. Gray, J.S. Sarma, K. Muthukkaruppan, N. Spiegelberg, H. Kuang, K. Ranganathan, D. Molkov, A. Menon, S. Rash, R. Schmidt, and A. Aiyer, *Apache hadoop goes realtime at Facebook*, in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 2011, ACM. Athens, Greece. pages. 1071-1080. DOI: 10.1145/1989323.1989438.

29. *Twitter Inc.* Available from: <https://twitter.com/>.

30. McKelvey, K. and F. 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. pages. 1351-1358.

31. Bingjing Zhang, Yang Ruan, and J. Qiu. *Harp: Collective Communication on Hadoop*. in *IEEE International Conference on Cloud Engineering (IC2E) conference*. March 9-12 2015. Tempe, AZ.

32. *Apache Tez*. 2014 Available from: <http://tez.incubator.apache.org/>.

33. Parr, T.J. and R.W. Quong, *ANTLR: A predicated‐LL (k) parser generator.* Software: Practice and Experience, 1995. 25(7): p. 789-810.

34. Olston, C., B. Reed, U. Srivastava, R. Kumar, and A. 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. pages. 1099-1110. <http://portal.acm.org/citation.cfm?id=1376726>.

35. *Pig Programming Tools*. 2014 Available from: <http://en.wikipedia.org/wiki/Pig_(programming_tool)>.

36. J.Ekanayake, H.Li, B.Zhang, T.Gunarathne, S.Bae, J.Qiu, and G.Fox, *Twister: A Runtime for iterative MapReduce*, in *Proceedings of the First International Workshop on MapReduce and its Applications of ACM HPDC 2010 conference June 20-25, 2010*. 2010, ACM. Chicago, Illinois. <http://grids.ucs.indiana.edu/ptliupages/publications/hpdc-camera-ready-submission.pdf>.

37. Zaharia, M., M. Chowdhury, M.J. Franklin, S. Shenker, and I. Stoica, *Spark: Cluster Computing with Working Sets*, in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud '10)*. June 22, 2010. Boston. <http://www.cs.berkeley.edu/~franklin/Papers/hotcloud.pdf>.

38. Bu, Y., B. Howe, M. Balazinska, and M.D. Ernst, *HaLoop: Efficient Iterative Data Processing on Large Clusters*, in *The 36th International Conference on Very Large Data Bases*. September 13-17, 2010, VLDB Endowment: Vol. 3. Singapore. <http://www.ics.uci.edu/~yingyib/papers/HaLoop_camera_ready.pdf>.

39. Wu, T.-L., A. Koppula, and J. 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.

40. Zaharia, M., M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M.J. Franklin, S. Shenker, and I. Stoica, *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing*, in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. 2012, USENIX Association. San Jose, CA. pages. 2-2.