Journal of Big Data

RESEARCH Open Access

Evaluation of high-level query languages based on MapReduce in Big Data

Marouane Birjali[1*†], Abderrahim Beni-Hssane[1†] and Mohammed Erritali[2†]

*Correspondence: birjali.marouane@gmail.com
†Marouane Birjali, Abderrahim Beni-Hssane and Mohammed Erritali are contributed equally to this work
[1] LAROSERI Laboratory, Department of Computer Science, Faculty of Sciences, University of Chouaib Doukkali, El Jadida, Morocco
Full list of author information is available at the end of the article

Abstract
MapReduce (MR) is a criterion of Big Data processing model with parallel and distributed large datasets. This model knows difficult problems related to low-level and batch nature of MR that gives rise to an abstraction layer on the top of MR. Therefore; several High-Level MapReduce Query Languages built on the top of MR provide more abstract query languages and extend the MR programming model. These High-Level MapReduce Query Languages remove the burden of MR programming away from the developers and make a soft migration of existing competences with SQL skills to Big Data. This paper investigates the very used—common High-Level MapReduce Query Languages built directly on the top of MR that translate queries into executable native MR jobs. It evaluates the performance of the four presented High-Level MapReduce Query Languages: JAQL, Hive, Big SQL and Pig, with regards to their insightful perspectives and ease of programming. The baseline metrics reported are increasing input size, scale-out number of nodes and controlling number of reducers. The experimental results study the technical advantages and limitations of each High-Level MapReduce Query Languages. Finally, the paper provides a summary for developers to choose the High-Level MapReduce Query Languages which fulfill their needs and interests.

Keywords: High Level MapReduce Query Languages, JAQL, Big SQL, Hive, Pig, Hadoop, Big Data, Performance comparison

Introduction

Since it was presented by Google in 2004, MapReduce (MR) [1] has been emerged as a popular framework for Big Data processing model in cluster environment and cloud computing [2]. It has become a key of success for processing, analyzing and managing large data sets with some number of implementations including the open-source Hadoop framework [3, 4]. MR has many interesting qualities, highly noticed in its design and plainness in writing programs. It has only two functions, known as Map and Reduce, written by developer to process key-value data pairs. Even though, MR is very simple to understand its principle and basic concepts but it is hard to develop, optimize, and maintain its functions especially in large-scale projects [5]. MR requires approaching any problem in terms of key-value pairs where each pair can be independently computed. Also, coding efficient MapReduce programs, mainly in Java, was non-trivial for those who interested to build large-scale projects even though their programming level. This meant that many operations need multiple inputs/outputs, both simple and complex, that were very hard to achieve without wasting programming efforts and time.

Springer Open

© The Author(s) 2018. This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

Furthermore, MR knows several limitations coming from its batch nature to handle streaming data [6, 7]. In addition, it is unsuitable for many operations with multiple inputs like Join operation [6, 8]. Therefore, the difficulty related to low-level programming of MR gives rise to high-level query languages (HLQL) based on MR, an abstraction layer constructed directly on the top of MR, designed to translate queries and scripts into executable native MR jobs.

High-Level MapReduce Query Languages (MapReduce-based HLQL) are built directly on the top of Hadoop MR to facilitate using MR as the low-level programming model [5]. They outline the absence of support that MR provides for complex dataflows and they provide explicit support for multiple data sources. Moreover, the choice of the Big Data processing model has a crucial role to play in the implementation of HLQL [9]. Several Big Data Processing model have been suggested in combined with MR, such as Resilient Distributed Dataset (RDD) and Bulk Synchronous Processing (BSP) [10, 11]. These two models can be combined with Hadoop but can not depend on MR. However, other HLQL based on RDD or BSP as Big Data programming model which offers an abstract model of parallel architectures and algorithms based on memory or processor computing. In fact, RDD shows a weak evaluation because of RDD data transformations [12]. RDD's data can be recomputed if they have lost in failure because it avoids data replication [12]. HLQL based on RDD do many iterative computations on the same data which necessities a lot of memory for keeping the data [13]. Therefore, the use of the main memory by HLQL based on RDD affects the cost of the infrastructure because memory resources are very expensive than disk. Data continue to increase, and HLQL based on RDD needs more expensive infrastructure compared to MapReduce-based HLQL [14]. In other words, the BSP model shows the same issues presented by RDD but in term of processor-memory pairs. In addition, BSP model is a sequence of super-steps and each super-step produces in three phases [15]. In each phase, data computed by processor from memory. Therefore, BSP needs to minimize the computation time of each phase partitions. Many interactions are requested between other processors and the synchronization between processors is also a potential source of errors in HLQL based on BSP [16].

In the light of the previous limitations mentioned in these processing models in terms of the exhaustive need for expensive memory and the need for successive data transformations, we are interested, in this paper, with the MR processing model [3]. There are many advantages in this model, such as it processes data on disk without requiring many transformations on data. MR processing model is based on a simple programming model and plainness in writing programs. It is cost-effective and flexible Big Data processing model which has access to various new sources of applications [15, 17]. Therefore, we investigate MapReduce-based HLQL that built directly on the top of MR, which translate all queries and scripts to be executed into native MR jobs. Their queries are automatically compiled to translate equivalent native MR jobs for execution, while other HLQL do not make the native translation into jobs. These MapReduce-based HLQL provide more abstract query languages, extending the MR programming model. They remove the burden of MR programming away from the developers and make a soft migration of existing competences with SQL skills to Big Data environment. JAQL, Big SQL, Hive and Pig are all the very used languages built on the top of MR to translate their queries into native MR jobs, named respectively JAQL [18], Ansi-SQL [19], HiveQL [20] and Pig Latin [21].

The four MapReduce-based HLQL presented in this paper have built-in support for data partitioning, parallel execution and random access of data. In this paper, we choose the four MapReduce-based HLQL because they offer the developers the opportunity to write small and easy programs, which are equivalent to MR programs. All these languages compile and execute their queries into native MR jobs, as well as they built on the top of MR [5, 19]. The developer can write programs in high-level other than writing in low-level MR programs. Furthermore, our study has a large impact in the MapReduce-based HLQL development communities, in terms of describing technical limitations and advantages, presenting results and providing recommendations for each MapReduce-based HLQL regarding their features, ease of programming and performance. The essential metrics used in our work are: increasing input size, scale-out number of nodes and controlling number of Reducer tasks. These metrics are the appropriate and canonical benchmarks used in the literature review of MR and HLQL [5, 22–24].

Moreover, to demonstrate how much shorter are the different queries of each MapReduce-based HLQL, we investigate specifically the concise language to give insight into programming languages perspectives, such as: ease of programming with the average ratio compared with MR, ease of configuration to link with Hadoop, and execution environment [5, 9]. Moreover, we make a comparison summary for developers to facilitate their choice of each MapReduce-based HLQL to fulfill their needs and interests.

The rest of the paper is organized as follows: After presenting literature review and related works in "Related work" section. In "Background and languages" section, we go through a synthetic study of these MapReduce-based HLQL in order to present their advantages, perspectives and architecture. Then, we compare the conciseness of each MapReduce-based HLQL using the source lines and instructions of code metric. In "High-level MapReduce query languages comparison" section, we further make a comparison of how expressive are the MapReduce-based HLQL, to determine whether or not that MapReduce-based HLQL pay a performance penalty for providing more abstract languages. In "Results and discussion" section, for each benchmark metric, we take a direct implementation using basis performance of MR, which allows to assess and evaluate the overhead of each MapReduce-based HLQL. Finally, "Conclusion" section concludes this paper.

Related work

Since its emergence publication in [1], MR processing model has become a criterion of certain analytical tasks. The same authors point out an extensive increase of the number of MR processes that ran in some very large companies during its first years of emergence [25]. Afterwards, MR becomes a popular Big Data processing model for large-scale data sets running on clusters in fault tolerant manner [1, 3]. In addition to MR processing model, several Big Data processing models have been proposed, such as RDD [11] and BSP [10], which are considered as other alternative Big Data processing models of some HLQL. Fegaras et al. [26] proposed MRQL as SQL-like query language for large-scale data analysis. MRQL is a HLQL based on BSP that can evaluate their queries in four modes [27]: MR mode using Apache Hadoop [1], BSP mode using Apache

Hama [28], Spark mode using Apache Spark [24], and in Flink mode using Apache Flink [29]. In MRQL-to-MR mode, MRQL translates MRQL queries in physical MR algebraic operators, which optimizes and translates the algebraic form to a physical plan consisting of the physical MR operators and not into native MR jobs [26]. In [28], the authors presented Apache Hama, BSP-based model processing, which provides not only pure BSP programming model but it is also constructed on the top of HDFS [30]. Further, the work implemented in [31] presents Apache Drill as an interactive ad-hoc analysis. It is created as Apache Incubator Project [32]. This work offers better low latency SQL engine but its application tool and visualization are very limited to customization. Moreover, it is absolutely not dependent on Hadoop [31]. Apache Phoenix is used only for HBase data and compiles queries into native HBase calls, without any interaction with MR [33]. Chang et al. [34] proposed the framework HAWQ which is another HLQL, it breaks complex queries into small tasks and distributes them to massively parallel processing (MPP) query processing units for execution [23].

In [35], the authors have presented Impala, an open-source MPP SQL query engine designed specifically to leverage the flexibility and scalability of Hadoop framework. However, Impala is an intensive memory and effectively not applicable for heavy data operations like Joins because it is not possible to transmit massive data into the memory. Furthermore, fault tolerance is not supported by Impala. The authors have proposed Llama as an intermediate component for integrated resource management within YARN [36]. Therefore, it can not translate their queries into native MR jobs. Moreover, the number of Hadoop-based systems has increased significantly. These systems use another framework to process SQL-like queries. Spark Core is one of the executive systems to process SQL-like queries of Spark framework. Thus, every other functionality is built on the top of Spark Core.

Michael et al. [37] presented Spark SQL as SQL-like data processing built on the top of Spark Core. The presented Spark SQL processes data upon RDD abstraction which has many limitations in computations and data in-memory nature [12, 13]. The use of the main memory by Spark SQL affects more cost-effective of the infrastructure, due to memory resources which are very expensive than disk while Hadoop processes data on disk [38]. Spark needs larger expensive infrastructure compared to Hadoop [14]. Further, developers are limited by using Spark SQL because is supported only in Spark [19, 37]. But in reality, interested companies need a generic solution that works with a lot of native SQL or like-SQL and is easily scalable to any future systems [19, 39]. As result, HLQL based on BSP or RDD show many limitations according to their processing models nature and the manner in which these HLQL based on BSP and RDD are implemented. They require more memory that is expensive and many complex computations on the data. These HLQL have a complex configuration to link with another open-source framework and do not provide easy programming.

Therefore, HLQL based on MR have built-in support for data partitioning, parallel execution and random access of data. They offer the opportunity to write small and easy programs, which are equivalent to MR programs. These HLQL based on MR compile their queries and scripts into executable native MR jobs, as well as they built on the top of MR, without many transformations on data. The developer can write programs in high-level other than writing in low-level MR programs. In fact, many works have been realized in MR query optimization: Pig Latin [21] and HiveQL [20] provide HLQL developed on the top of MR. They define an SQL-like high-level languages for processing large-scale analytic workloads. A performance comparison between three HLQL is conducted in the work [5], the authors show that Hive is more efficient than Pig using scaling input data size, processing units and execution time [40]. They used only two metrics to evaluate the performance of their three HLQL: JAQL, HiveQL and Pig Latin. In [21], the authors describe the challenges that have faced in developing Pig, and reported performance comparisons between Pig execution and raw MR execution. In [41], the authors have chosen two specific languages: Pig and JAQL. They compared them regarding two metrics. Another performance analysis of high-level query languages is implemented in [22]. The authors have analyzed the performance of the three high-level query languages Pig, Hive, and JAQL based on the processing time. All existing HLQL based on MR are not included in their work and they have used only one metric.

Our work differs from the previous studies in the literature by specifying MR as a Big Data processing model and adding the recent MapReduce-based HLQL, Big SQL which is native SQL query language that translates native Big SQL queries into native MR jobs. Also, we compare Big SQL with the very used MapReduce-based HLQL in their recent and stable version: JAQL, Hive and Pig. Therefore, we have added more than two metrics to evaluate the performance of the four MapReduce-based HLQL: JAQL, Hive, Pig and Big SQL. These metrics are: increasing input size, scale-out number of nodes and controlling number of Reducers. Besides the previous three metrics, we add other valuable one which is the language conciseness that gives insight into language programming perspectives, such as: ease of programming with the average ratio compared with MR and ease of configuration to link with Hadoop, the execution environment. Finally, we offer a summary for developers to choose the MapReduce-based HLQL which fulfill their needs and interests.

Background and languages

There are four MapReduce-based HLQL presented in this paper that have emerged out of the MR programing model. They also provide abstractions on the top of Hadoop framework. These abstractions are used to reduce the amount of low-level difficulties required for typical tasks and to translate queries into native MR jobs. Moreover, they allow developers to write programs using MapReduce-based HLQL abstractions that can be compiled into native MR jobs. These languages provide several operators, so developers can develop their own functions for reading, writing, and processing data. MapReduce-based HLQL are easy to be scripted, modified, and understood. Their relationship with Hadoop is shown in Fig. 1.

Hive: a data warehousing over Hadoop

Hive is a data warehouse infrastructure, built on the top of Hadoop framework that is developed by Facebook [20]. It provides a simple query language called HiveQL that supports queries expressed in a SQL-like declarative query language. Hive queries are compiled and translated into MR jobs that are executed on Hadoop. Hive provides a SQL-like, called Hive query language (HiveQL) for querying data stored in a Hadoop [42]. Having SQL-like features, HiveQL provides several functions and operations like

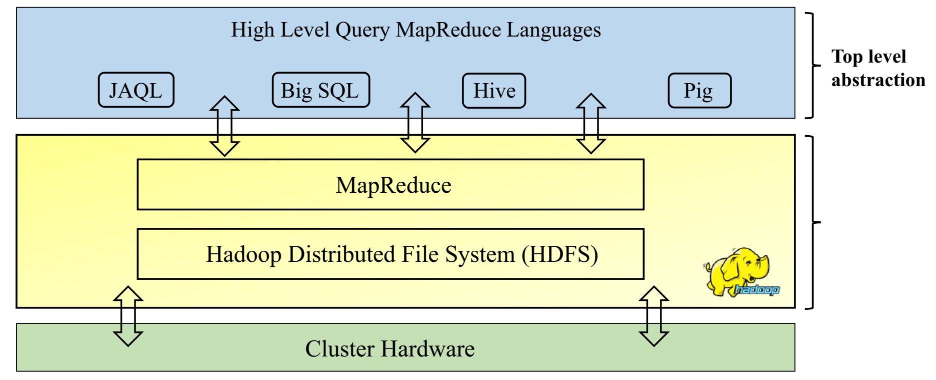
High Level Query MapReduce Languages
JAQL
Big SQL
Hive
Pig
Top level abstraction
MapReduce
Hadoop Distributed File System (HDFS)
Cluster Hardware

Fig. 1 MapReduce-based HLQL architecture and their position on top of MR

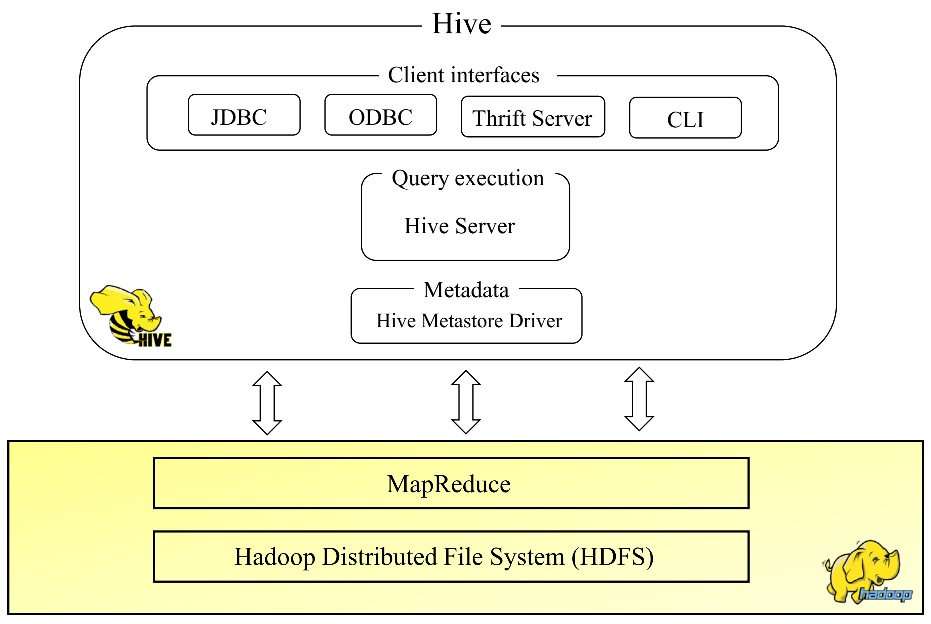

Fig. 2 Hive system architecture

group by, joins, aggregation etc. In other words, it provides an easy data summarization, ad-hoc querying and analysis of large volumes of data.

The Hive architecture presented in the Fig. 2 is mainly composed of four main components. The first component is the external interface that consists of sub-component: command line (CLI), web user interface, application-programming interface (API) shown either as JDBC or ODBC [20]. The next one is the driver manager, the life cycle of HiveQL statements during compilation and execution that receives the queries and creates a session handle [23]. The third component is the compiler invoked by the driver upon receiving HiveQL queries. It translates those statements for generating an execution plan. The fourth one is the metastore which is the system catalog for Hive. It performs the validation of the relational schema or query. All other components of Hive interact with the metastore [20].

JAQL: a JSON query language to MR

JAQL [18] is a functional scripting language with data model based on JavaScript Object Notation Language (JSON) [43]. JAQL facilitates parallel processing by translating high-level queries into low-level ones [44]. It is a dataflow language that manipulates semi-structured data using JSON values. It is able to exploit massive parallelism in a transparent manner using Hadoop framework. It started as an open-source project at Google, but IBM took the latest releases as primary data processing language [5, 18].

JAQL is extendable with integration point for various data sources like local and distributed file systems, NoSQL (HBase) and relational databases. When comparing JAQL with developing dataflows directly with MR framework, JAQL have exhibited similar advantages in relational database systems. It provides a proven abstraction for data management systems. At the point of running, JAQL transforms the parsed statement into equivalent optimized statement. The optimized query can be transformed back to JAQL code. The latter is useful for debugging. Moreover, JAQL provides SQL with native JAQL functions that can parameterize and package any JAQL logic. Furthermore, JAQL can be extended with UDF (user defined functions), written in JAQL itself. However, it is possible to be used as a general-purpose programming language, JAQL is the foundation for Big SQL presented in the following subsection. The Fig. 3 illustrates the architecture system of JAQL [18].

Big SQL: native SQL access on the top of MR

Hive provides HiveQL [20], a SQL-like, but not SQL. It has some limitations in query access for Hadoop at the level of the data types: no varchar, no decimal and others. HiveQL have not join support, and despite the JDBC/ODBC driver limitations, all queries are executed in MR jobs. All those limitations require a native SQL access for Hadoop, namely Big SQL.

Big SQL [19, 45] is MapReduce-based HLQL designed for providing native SQL for querying data managed by Hadoop, and developed mainly by IBM [19]. Big SQL provides massively parallel processing SQL that can deploy directly on the Hadoop Distributed File System (HDFS) [30]. It is able to use a low-latency parallel execution processing that access directly on the Hadoop data natively for reading and writing SQL queries. Big SQL is able to run on the top of Hadoop and to translate all queries to native MR jobs. It

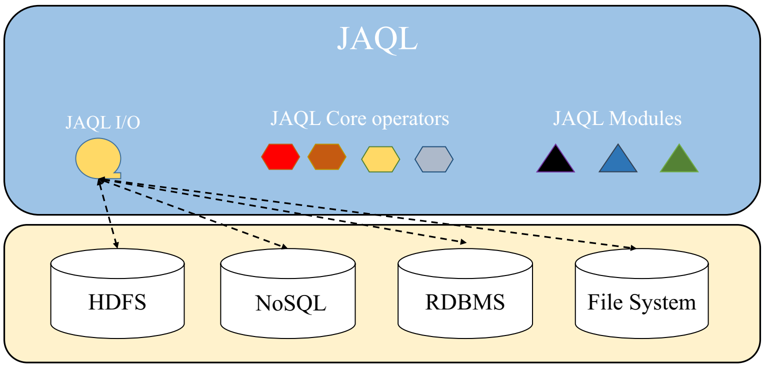

Fig. 3 The architecture system of JAQL

supports queries expressed in native SQL declarative language, Ansi-SQL. These queries are compiled into native MR jobs. The following architecture diagram shows how Big SQL fits with the Hadoop ecosystem.

Figure 4 illustrates the architecture of Big SQL and how it fits the Hadoop ecosystem. It supports JDBC/ODBC driver access from Linux and Windows platforms. Big SQL uses HCatalog (metastore) of Hbase for data access and the Hive storage engines to read/write data. In addition, it provides the ability to create virtual tables because the data is synthesized via JAQL scripts. Big SQL provides its own HBase storage handler due to its capacity to execute all queries in parallel execution through MR processing model. By its Ansi-SQL language, Big SQL provides direct access for low-latency queries. The Big SQL engine supports joins, unions, grouping, common table expressions, and other familiar SQL expressions.

Pig: a high-level data flow language for Hadoop

Pig [21] is a high-level data flow language for data transformation which used to analyze massive datasets and represent them as data flows. The language used for expressing these data flows is Pig Latin. This language is an abstraction of the MR programming model which makes it a HLQL constructed on the top of Hadoop. It includes many traditional data operations (sort, join, filter, etc.), as well as the ability for programmers to develop their own functions for accessing, processing, and analyzing data [23]. Pig provides an engine for executing data flows in parallel manner using Hadoop framework. The architecture of Pig is shown in Fig. 5. It shows that Pig Latin scripts are firstly handled by the Parser which checks the syntax and instance of the script. The output of the

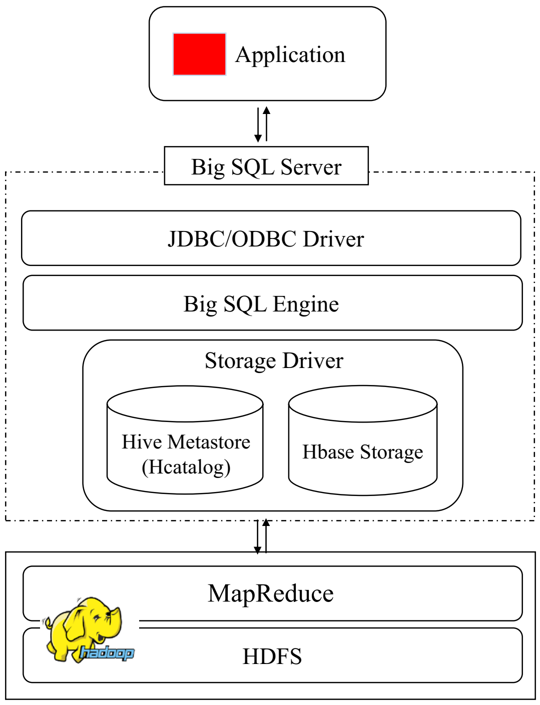

Fig. 4 Big SQL architecture

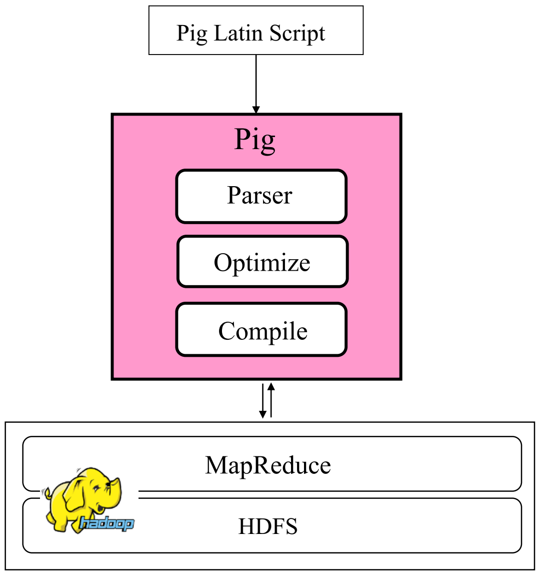

Fig. 5 The Pig architecture

parser is a logical plan, a collection of vertices where each vertex executes a fragment of the script. The parser is also a representation of the Pig Latin statements and logical operators. This logical plan is compiled by MR compiler to submit Pig Latin scripts, as native MR jobs, to Hadoop job manager for execution.

High-level MapReduce query languages comparison

The four specific MapReduce-based HLQL presented in this paper establish an abstract layer for utilizing the MR programming model. We have chosen to compare JAQL, Big SQL, Hive and Pig regarding their features, ease of programming and speed of programming. To differentiate the insightful perspective of MapReduce-based HLQL, we used the WordCount benchmark of each language, which is the appropriate canonical benchmark used in the literature review of MR and MapReduce-based HLQL [5, 22–24]. In each MapReduce-based HLQL, we investigate specifically the conciseness of each language, which gives insight into MapReduce-based HLQL perspectives and draw a comparison, on the one hand, to examine how expressive are the MapReduce-based HLQL, on the other hand, to determine whether or not MapReduce-based HLQL pay a performance penalty for providing more abstract languages.

Programming languages with MapReduce-based HLQL

The Hive query language (HiveQL) includes a subset of SQL-like and some useful extensions in this comparison. Traditional SQL characteristics such as: sub-queries, group by and aggregations, various types of joins, union all and other functions make the language very SQL-like. The structure of HiveQL in database's notions are tables, columns and partitions. It supports all the important primitive types such as doubles, integers, strings and collection types such as lists, structs and maps. Hive comprises a query compiler, responsible for compiling HiveQL into a directed cyclic graph of MR jobs. The Hive word count query is given in Listing 1.1.

Listing 1.1. HiveQL Word Count query

```
load data inpath '/user/wcdirectory/' into table myinput;
create table wordcount as
select word, count(1) as count
from (select explode(split(lcase(regexp_replace(line,'[\\p{punct},\\p{cntrl}]','')),' ')) as word from myin-
  put) words
group by word;
```

Concerning the principal characteristics of JAQL and its capability to read/write from different storage types, mainly the HDFS, local file system, HTTP and HBase. JAQL is extendable with user defined functions (UDF), either written in many popular programming languages or in JAQL itself. Its values, arguments, variables and return values, named key are JSON objects or JAQL functions. The JAQL word count is given in Listing 1.2.

Listing 1.2. JAQL Word Count query

```
$InputData = read(lines("/user/Jaql/wcdirectory/"))
$InputData → expandstrSplit($, " ")
$InputData → group by $w = ($) into [$w, count($)]
→ write(seq("/user/jaql/outputJaqlWC/JAQLOutput"));
```

Big SQL requires creating tables and familiarizing them with data. It supports a Create Table statement and a LOAD command for moving data. Although, the fundamental syntax of Big SQL query is similar to SQL, there are some aspects of Big SQL, table creation and data loading, that probably will not. The Load command of Big SQL reads and moves data simply and directly from several relational DBMS as well as from files stored locally or in HDFS. The Big SQL word count query is given in Listing 1.3.

Listing 1.3. Big SQL WordCount query

```
create table "wctable" ("word" varchar(32768) )
row format delimited fields terminated by ';'
lines terminated by '\n';
load hive data local inpath '/tmp/wctable.csv'
overwrite into table wordcount.wctable;
select word, count(word) as wordcount from wctable group by word;
```

With its language Pig Latin, Pig takes three key aspects in account [35] for its development. The first key allows developers to create programs which are easy to write, understand, and maintain. The second key is the optimization whose tasks are to transform and allow system optimizing their execution automatically. The last one is the extensibility where developers can create their own functions by the UDF. The Pig Latin word count is shown in Listing 1.4.

Listing 1.4. Pig Latin Word Count query

```
myinput = Load '/user/wcdirectory/' Using TextLoader AS (line:CHARARRAY);
words = Foreach myinput Generate Flatten (Tokenize(Replace(Lower(Trim(line)), [\\p{Punct},\\
  p{Cntrl}],'')));
grpd = Group words By $0;
cntd = Foreach grpd Generate $0, Count($1);
unmix = Order cntd BY $1 Desc, $0 ASC;
Store unmix Into '/user/unmixPIG.dat' Using PigStorage( );
```

Comparative analysis of MapReduce-based HLQL

This subsection outlines the concise language of the four MapReduce-based HLQL. It gives a general insight from these languages perspectives with the average ratio compared with MR. For each benchmark, we compare the conciseness between the four MapReduce-based HLQL with MR using the source lines of code metric (Fig. 7). We further evaluate the four MapReduce-based HLQL based on the use of computational power to achieve the performance comparison (Fig. 6).

Figure 6 shows the computational power of JAQL, Big SQL, Hive and Pig, MR and MR-Merge model [46]. The principal idea of MR-Merge is to bring relational operations into parallel data processing. It can be used to implement some derived relational operators. Thus, MR-Merge is relationally complete; whereas, MR is not relationally complete. Neither Pig Latin nor HiveQL provides loop structures required to be established as Turing Complete languages. Pig Latin and HiveQL can both extend UDF. JAQL provides not only recursive function but it can be also defined as Turing Complete. Big SQL can extend UDF in the context of the “Select” list of queries or in a “where clause” to filter data.

Figure 7 illustrates the source lines of the four MapReduce-based HLQL’s code metric, with a direct implementation of Java MR, to focus on the abstract nature of these languages in order to compare the used conciseness in the WordCount, Join and web log processing benchmark.

Programs in all four MapReduce-based HLQL (JAQL, Big SQL, Hive and Pig) are shorter than the equivalent Java MR:

- WordCount: MR is 39 lines and all MapReduce-based HLQL are smaller than 6 lines.
- Join: MR is 95 lines and all MapReduce-based HLQL are smaller than 8 lines.
- Web log processing: Java is 140 lines and all MapReduce-based HLQL are smaller than 6 lines.

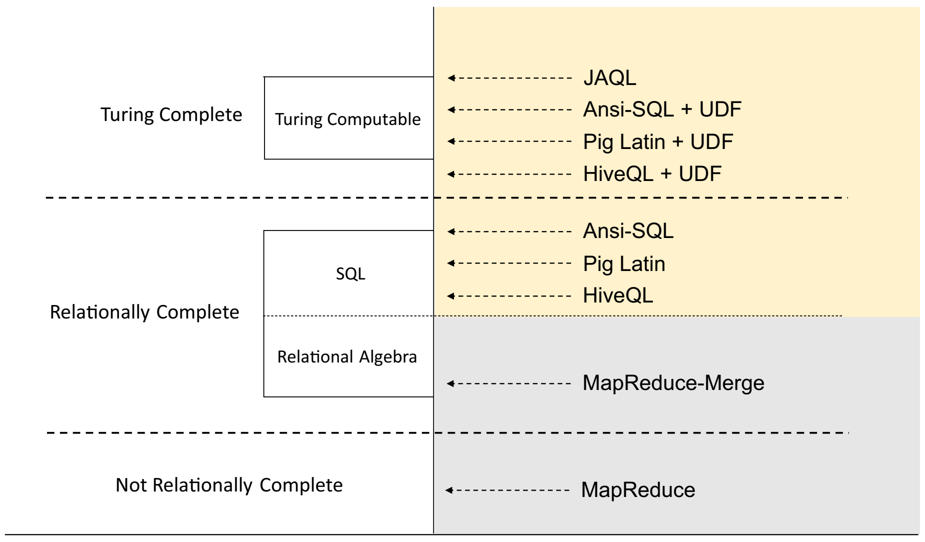

Fig. 6 Computational power comparison

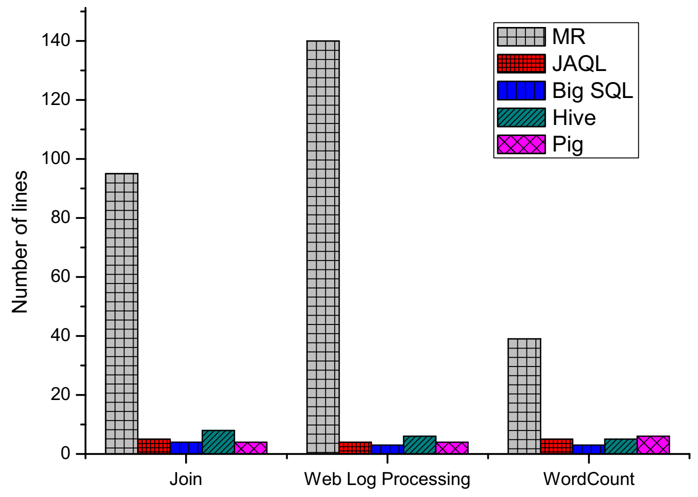

Fig. 7 Source lines of code comparison of the four MapReduce-based HLQL

Developers spend more time in writing the MR manually, debugging large applications, and hence require a magnitude program size. Figure 7 highlights the abstract nature of these four languages through presenting the source lines of code for each benchmark metric which is much smaller than the equivalent Java MR. Pig and Hive are unable to check and control the number of Reducer tasks within the language syntax. They can tune this parameter to improve the runtime performance. JAQL has proved to be both the most computationally powerful language and Turing Complete. Moreover, Big SQL is the shortest high-level language compared to all MapReduce-based HLQL and MR.

Results and discussion

This section outlines the runtime environment used for the performance comparison of the four MapReduce-based HLQL. The metrics used for making a systematic performance comparison are: increasing input size, scale-out number of nodes and controlling reducer tasks. The performance baseline for each benchmark is a direct implementation of MR, which allow the developers to assess the overhead of each MapReduce-based HLQL. Moreover, WordCount is one of the applicable canonical benchmarks in the literature of Hadoop MR. Furthermore, the Web Log benchmark is another standard used for processing purposes. It is based on the processing of queries to count the average time spent on the website pages by each visitor on a set of web files in a textual format. The Join benchmark consists of two sub-tasks that perform selecting data from two tables. These three benchmarks are the appropriate traditional ones founded in the literature review of MR and HLQL [5, 22–24].

All experiments utilize only Hadoop MR v2.7.2 (latest stable version) as execution engine, Hive v2.3.2, Pig v0.17.0, JAQL v0.6 and Big SQL v4.2.4 were running on cluster consisting of one dedicated to NameNode and ten DataNodes. Each node was equipped with processor Intel Core i5-5300U CPU 2.30 GHz and 8 GB of RAM. We have used 10 datasets, for x = 1,...,10 of size x = 2GBs. That is, the largest dataset is x10 = 20GBs. The input format used for our experiments in HDFS is a textual file. Our experiments do not include data preparation and loading time.

Increasing input size metric

To perform the scaling input size measurement of each MapReduce-based HLQL, the size of the cluster is fixed at one NameNode and ten DataNodes, and the dataset size has been doubled for each experiment to get the required results. This experiment is based on the analysis of the processing time by using the WordCount, web log processing and the Join program. All the used programs are written with the four MapReduce-based HLQL namely JAQL, Ansi-SQL, HiveQL and Pig Latin.

A prominent characteristic of these three measurements is about the total running time, which increases with the input size. It can be observed that Pig and JAQL achieve similar performance, though both MR Java and Hive perform considerably better. This can be seen clearly from experiment results shown in Figs. 8, 9, whereas, Big SQL is the most powerful processing time in the three experiments presented previously (Figs. 8, 9, 10). In the Join benchmark, from the smallest (x1) to the largest data input size (x10), JAQL has the highest running time compared to other MapReduce-based HLQL and MR Java (Fig. 10). Big SQL achieves the lowest running time among all other languages with 42% quicker than JAQL (Fig. 8).

As a result, in Figs. 9 and 10, a weak performance is delivered by Pig in every running time performance of Join benchmark compared to its high performance in web log processing. While comparing these MapReduce-based HLQL at the level of the running time, we find that the Join benchmark of the MR java, Hive and Pig take almost the same execution time (Fig. 10). On one hand, JAQL takes a long running time. On the other hand, Big SQL is characterized by less running time in increasing input size metric.

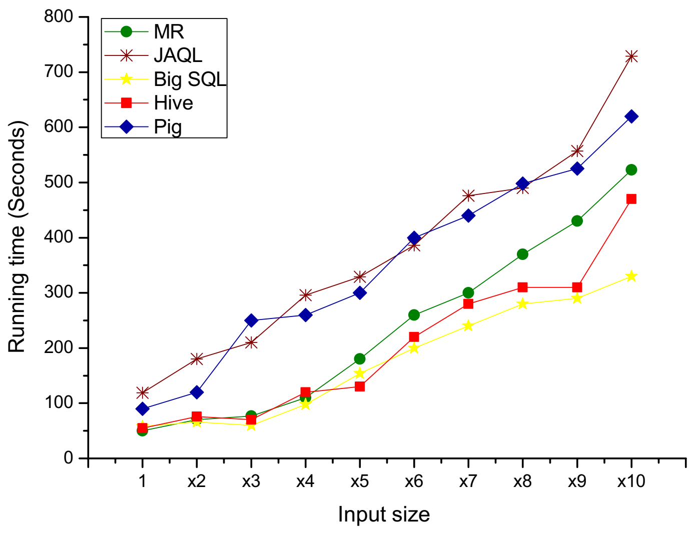

Fig. 8 WordCount benchmark metric in increasing input size

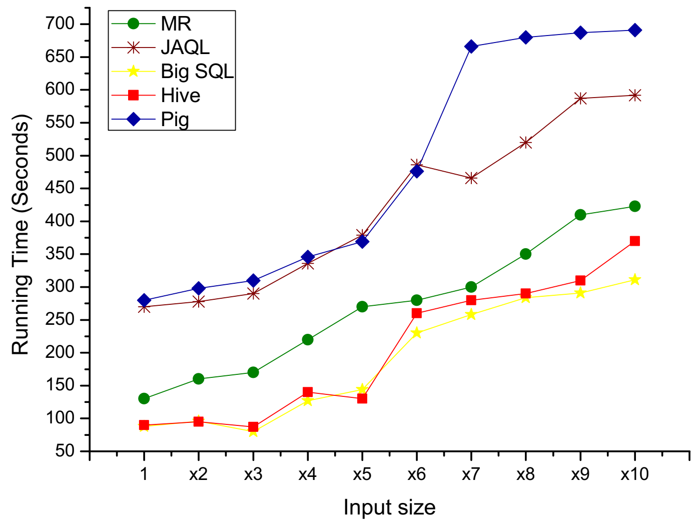

Fig. 9 Web log processing benchmark metric in increasing input size

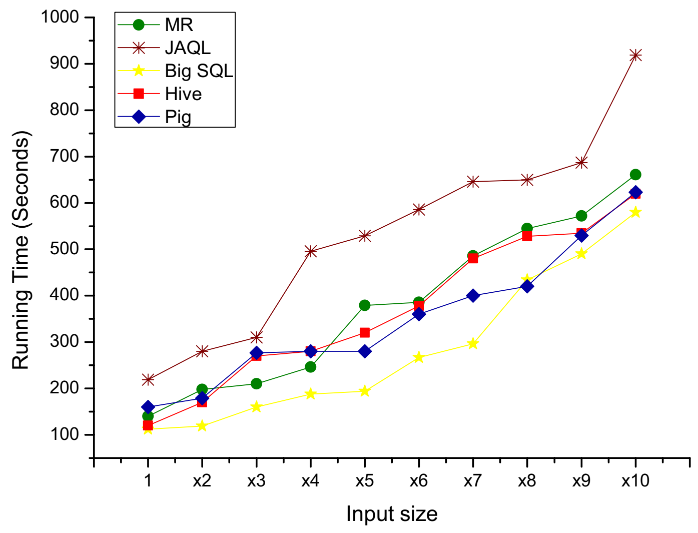

Fig. 10 Join benchmark metric in increasing input size

Scale-out number of nodes

We carry out another experiment where we have focused on the number of nodes (DataNodes). These number will be increased from 1 NameNode/DataNode to 1 NameNode and 10 DataNodes. The size of input datasets is fixed in x10. The results of these experiments, scale-out number of nodes, are represented in Figs. 11, 12.

All these results clearly illustrate one common design challenge for parallel system. In the WordCount benchmark, beyond the first added node (Fig. 11), there is a notable improvement for Pig, Hive and JAQL. These three MapReduce-based HLQL are all capable to use the additional processing capacity up to 2 nodes. At each point, we notice no further expansion in three MapReduce-based HLQL running time performance. However, Big SQL does not benefit from adding nodes. As result, the running time performance is decreased by 43% from one to ten nodes (Fig. 11).

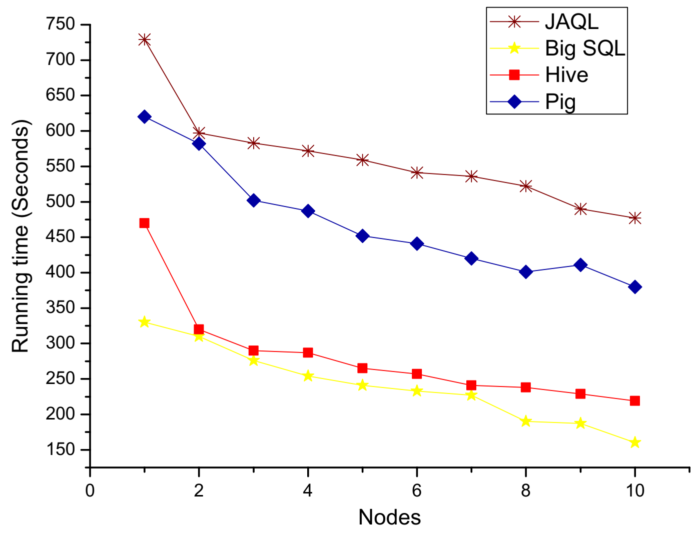

Fig. 11 WordCount benchmark for scale-out nodes

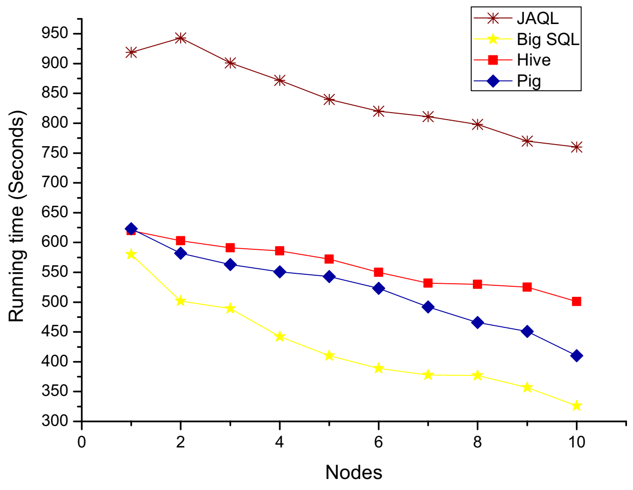

Fig. 12 Join benchmark for scale-out nodes

By considering on Fig. 12, which corresponds to the different running time performances, we can see that this experiment has different improvements compared to WordCount benchmark. This experiment, scale-out for Join benchmark, can not demonstrate that the addition of the nodes is beneficial for JAQL, Hive and Pig which achieve similar change. Therefore, Big SQL knows a good feedback with the addition of nodes. It has managed to decrease the running time performance by 60% (Fig. 12).

Controlling number of reducers

All MR jobs split into Map and Reduce tasks. Reducer tasks minimize a set of intermediate values associated with each intermediate key, generated by the map function and breaks down the output to a smaller set of values. A reducer writes output (key, value)

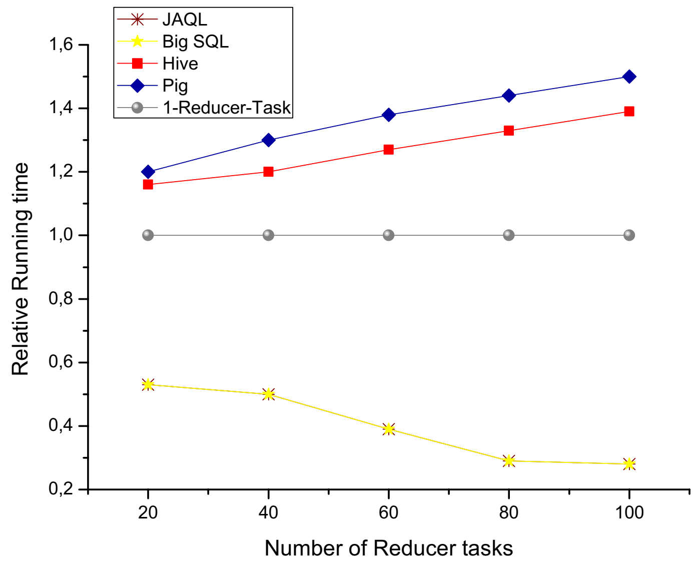

Fig. 13 Controlling the number of reducers: WordCount benchmark

pairs to both disk and memory via the namespace abstraction for further processing. The developers decide the number of reducers. Even if they have too many reducers and they can not determine the optimal number of reducers, context-switching overhead of reducer tasks will have a significant impact of performance.

Whilst the number of mappers is determined automatically, based on the split size which is 64 MB for each block. By default, one reducer is set in Hadoop configuration. There is a performance impact associated with MR jobs when investigating and controlling the number of reducer tasks. Many accomplished works have focused on Hive, Pig and MR, aiming to control reducers and to specify their required number while Big SQL and JAQL did not. JAQL and Big SQL will be added in this benchmark. Figures 13, 14 depict the results of controlling the number of reducers. Beyond 40 Reducers tasks, there is an increase in running time performance for all MapReduce-based HLQL. In parallel, Reduce tasks parameter increases to 100, whereas JAQL and Big SQL are all able to utilize the increasing number of Reducer tasks for the optimization of runtime capacity up to 40 Reducers tasks. WordCount program shows these results in Fig. 13.

An important contrast between the WordCount and the join benchmark, shown in Figs. 13, 14 respectively, proves that the three MapReduce-based HLQL, Big SQL, Pig and Hive achieve the fastest running time performance at 50 reducers, after which, performance grows gradually (Fig. 14). To summarize this experiment, the additional expressive of Reducer number can optimize performance with 40% (Fig. 14). In other words, controlling Reducers tasks provide more performance penalty with an increase of 50% in runtime performance (Fig. 13).

MapReduce-based HLQL summary and discussion

An important characteristic of all the four MapReduce-based HLQL, JAQL, Big SQL, Pig and Hive presented in this paper is that they are not limited to the core functionality of each language, which means they are all extendable by the used of UDF. These allow

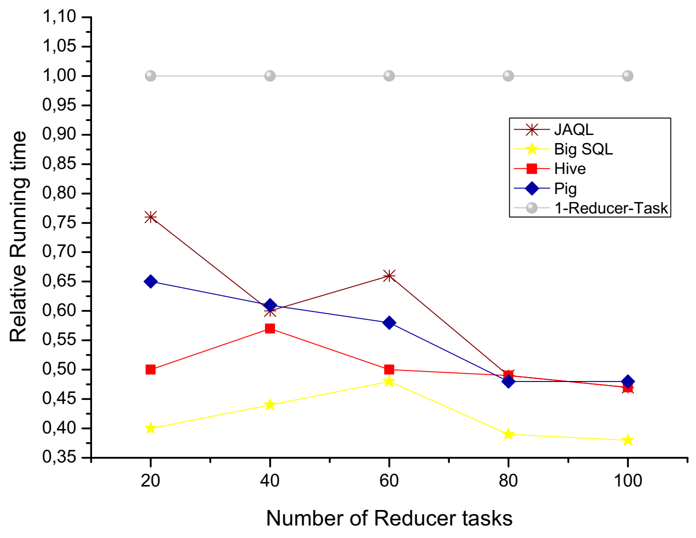

Fig. 14 Controlling the number of reducers: Join benchmark

developers to provide more custom data formats and functions. Moreover, MapReduce-based HLQL are very important components in the Hadoop environment because they work on the top of MR as Big Data processing model. Further, MapReduce-based HLQL provide the interface to process large datasets stocked in HDFS. However, the following table lists the important functions, comments and different perspectives of the four MapReduce-based HLQL presented in this paper.

JAQL is a weak high-level scripting language in term of running time and Turing Complete, while the other MapReduce-based HLQL are not (Fig. 6). Based on the results achieved in this paper, JAQL does not competitive with the running time performance of Big SQL, Hive and Pig as shown in "Increasing input size metric" and "Scale-out number of nodes" sections. However, it does not change much performance by adding additional nodes, as shown in Fig. 12. JAQL shows similar query runtime performance to Hive, but the flexibility is still needed. Furthermore, JAQL is designed to enable developers to use MR framework when it is needed, such low-level characteristics with declarative data-flows.

Big SQL is the most concise language, with an average ratio with MR Java of just 4.9%, as shown in Table 1. It presents the new-generation of SQL on Hadoop framework. In fact, for the previously mentioned constructs, it is exactly native SQL. The Big SQL engine supports joins, grouping, unions, common table expressions, and other familiar SQL expressions. The Big SQL LOAD command simply read and moves data directly from several relational DBMS systems as well as from files stored locally or in HDFS. Big SQL can use Hadoop's MR framework to process various query tasks in parallel or execute the queries locally within the Big SQL server whichever may be most appropriate for these queries. Big SQL has shown up a challenge for comparative studies like this report. In addition to the fact that these languages are comparatively new, they are moving targets when trying to build fair and relevant performance and feature comparisons.

Table 1 Comparative study between Big SQL, Pig, Hive and JAQL

Characteristic	MR	JAQL	Big SQL	Hive	Pig
Description	A programming model for parallel processing and generating large data sets	A data-flow processing and querying language	A HLQL designed for providing native SQL access for Hadoop	A data warehouse infrastructure for Hadoop	A high-level data flow interface for Hadoop
Language name	MapReduce	Jaql	Ansi-SQL	HiveQL	Pig Latin
Developed by	Google	IBM	IBM	Facebook	Yahoo
Type of language	Data processing paradigm	Data flow	SQL	SQL-like (presenting a declarative language)	Data flow
Evaluation	At runtime	At runtime	At runtime	During compilation	During compilation
Supported data	Structured and unstructured data (for structured data, MR may not be as efficient as Big SQL, Hive and Pig)	JSON and semi-structured	Mostly structured	Mostly structured	Complex
Process category	Batch processing	Dataflow for JSON/batch	Dataflow system OLAP/batch	Data warehouse OLAP/batch	Dataflow/batch
User defined functions	Extendable	Extendable	Extendable	Extendable	Extendable
Schema optional?	Without schema	Yes	No, mandatory	No, mandatory	Yes
Relational complete?	No	No	Yes	Yes	Yes
Turing complete?	Yes	Yes	Yes, when extended UDF	Yes, when extended UDF	Yes, when extended UDF
Source lines of code (mean ratio with MR Java)	–	7.1%	4.9%	25.4%	21.1%
Join operation	Difficult (it is quite hard to perform a join operation between data sets, and very hard with multiple data sources)	Simple	Simple	Simple	Simple

Pig is a high-level dataflow language for Hadoop that allows writing MR operations by the scripting language of Pig, Pig Latin. When writing a Pig Latin script, it is not necessary for developers to think about the MR paradigm since Pig handles the transformation of the script to the particular MR jobs. Pig has a concise language, with an average ratio with MR Java of 21.1%, while Hive has 25.4%, as shown in Table 1. Figure 10 shows that Pig and JAQL have almost the same runtime performance when scaling input size for the Join benchmark. In "Scale-out number of nodes" section, by adding nodes, the results show that Pig takes advantage from adding nodes with 66.3% of decreasing in runtime performance (Fig. 12). Hive knows a very low change, as shown in WordCount benchmark presented in Fig. 11. Besides the above observations presented in Join benchmark and illustrated in Fig. 12, Hive is not designed for online transaction processing and offer neither real-time queries nor row level updates. It is best applicable for query batch jobs to process large sets of immutable data, like web log processing.

Conclusion

This paper is a comparative study of the four MapReduce-based HLQL built on the top of MR processing model. The languages under investigation are JAQL, Big SQL, Hive and Pig designed to translate their queries into native MR jobs and provide more abstract query facilities instead of using low-level MR. The baseline numerical metrics reported in this paper are: increasing input size, scale-out number of nodes, controlling number of reducers. Moreover, the language conciseness of each MapReduce-based HLQL gives insight from programming language perspectives, such as: ease of programming and configuration to link with Hadoop, execution environment. This conciseness settles a ground for comparison that to see how expressive are the MapReduce-based HLQL, in order to determine whether or not MapReduce-based HLQL pay a performance penalty for providing more abstract languages. In fact, Big SQL has proved to be the best solution for the problem of integrating native SQL query processing on the top of MR. Whilst Pig Latin shows its limitation that lies in its expressiveness. In most benchmarks, Pig and JAQL have almost shown the same performance when increasing input size. Even though, it has the biggest percentage in source lines of code metric comparing to others, Hive provides performance closest to Big SQL.

Finally, this report also highlights the concise nature of the presented MapReduce-based HLQL. These languages provide an abstract layer to remove the burden away from developers. The paper provides also a summary comparison for developers to choose the MapReduce-based HLQL which fulfill their needs and interests. The findings have also proved that the presented MapReduce-based HLQL make soft migration of existing competences with SQL skills and systems to not cost-effective Big Data environment.

Abbreviations

BSP: bulk synchronous processing; HDFS: Hadoop Distributed File System; HLQL: high-level query languages; JSON: JavaScript Object Notation Language; MPP: massively parallel processing; MR: MapReduce; SQL: Structured Query Language; RDD: Resilient Distributed Dataset; UDF: user defined functions.

Authors' contributions

All authors read and approved the final manuscript.

Author details

[1] LAROSERI Laboratory, Department of Computer Science, Faculty of Sciences, University of Chouaib Doukkali, El Jadida, Morocco. [2] TIAD Laboratory, Department of Computer Science, Faculty of Sciences and Technologies, University of Sultan Moulay Slimane, Béni Mellal, Morocco.

Acknowledgements

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Availability of data and materials

Data will not be shared at this moment, as the datasets are for use in extension for my research work. Complete results and datasets of my research work will be shared in github.

Consent for publication

Not applicable.

Ethics approval and consent to participate
Not applicable.

Funding
I certify that no funding has been received for the conduct of this study and/or preparation of this manuscript.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 3 August 2018 Accepted: 28 September 2018
Published online: 06 October 2018

References

1. Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters. In: Proceedings of 6th USENIX symposium on operating systems design and implementation, OSDI 2004, San Francisco, USA. 2004.
2. Fegaras L, Li C, Gupta U. An optimization framework for map-reduce queries. In: Proceedings of the 15th international conference on extending database technology—EDBT'12. 2012. p. 26–37.
3. Hashem IAT, et al. Multi-objective scheduling of MapReduce jobs in big data processing. Multimed Tools Appl. 2017;77(8):9979–94.
4. Floratou A, Minhas UF, Ozcan F. SQL-onHadoop: full circle back to shared-nothing database architectures. Proc VLDB Endow. 2014;7(12):1295–306.
5. Stewart RJ, Trinder PW, Loidl HW. Comparing high level MapReduce query languages. In: Advanced parallel processing technologies, lecture notes in computer science, vol. 6965; 2011. p. 58–72.
6. Vasiliki K, Vladimir V. MapReduce: limitations, optimizations and open issues. In: 12th IEEE international conference on trust, security and privacy in computing and communications. 2013. p. 1031–8.
7. Bunjamin M, María SP, Gabriel A. Failure detector abstractions for MapReduce-based systems. Inf Sci. 2017;379:112–27.
8. Jaeseok M, Junho S, Jongheum Y, Sang-goo L. Handling data skew in join algorithms using MapReduce. Expert Syst Appl. 2016;51:286–99.
9. Chen Y et al. A study of sql-on-hadoop systems. In: workshop on Big Data benchmarks, performance optimization, and emerging hardware, lecture notes in computer science, vol. 8807; 2014. p. 154–66.
10. Tajdanowicz T, Indyk W, Kazienko P, Kukul J. Comparison of the efficiency of mapreduce and bulk synchronous parallel approaches to large network processing. In: Proceedings of IEEE 12th international conference on data mining workshops. 2012. p. 218–25.
11. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, Mccauley M, et al. 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. p. 15–28.
12. Dobre C, Xhafa F. Parallel programming paradigms and frameworks in Big Data era. Int J Parallel Prog. 2013;42(5):710–38.
13. Liang F, Lu X. Accelerating iterative Big Data computing through MPI. J Comput Sci Technol. 2015;30(2):283–94.
14. Mavridis I, Karatza H. Performance evaluation of cloud-based log file analysis with Apache Hadoop and Apache Spark. J Syst Softw. 2017;125:133–51.
15. Jing W, et al. MaMR: high-performance MapReduce programming model for material cloud applications. Comput Phys Commun. 2017;211:79–87.
16. Jakobsson A, et al. Replicated synchronization for imperative BSP programs. Proc Comput Sci. 2017;108:535–44.
17. Birjali M, Beni-Hssane A, Erritali M. Analyzing social media through Big Data using InfoSphere BigInsights and Apache Flume. Proc Comput Sci. 2017;113:280–5.
18. Kevin SB, et al. JAQL: a scripting language for large scale semistructured data analysis. Proc VLDB Endow. 2011;4(12):1272–83.
19. Nick RK, et al. A generic solution to integrate SQL and analytics for Big Data. In: 18th international conference on extending database technology (EDBT). 2015. p. 671–6.
20. Ashish T, et al. Hive: a warehousing solution over a map-reduce framework. Proc VLDB Endow. 2009;2(2):1626–9.
21. Christopher O, Benjamin R, Utkarsh S, Ravi K, Andrew T. Pig latin: a not-so-foreign language for data processing. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. 2008. p. 1099–110.
22. Namrata S, Sanjay A. A performance analysis of high-level MapReduce query languages in Big Data. In: Proceedings of the international congress on information and communication technology, advances in intelligent systems and computing, vol. 438; 2016. p. 551–8 **(only in RW)**.
23. Xin C, Liting H, Liangqi L, Jing C. Breaking down Hadoop distributed file systems data analytics tools: Apache Hive vs. Apache Pig vs. pivotal HWAQ. In: 10th international conference on cloud computing (CLOUD), IEEE. 2017. p. 794–7.
24. Katsogridakis P, Papagiannaki S, Pratikakis P. Execution of recursive queries in Apache Spark. In: Parallel processing euro-par, lecture notes in computer science, vol. 10417; 2017. p. 289–302.
25. Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters. Commun ACM. 2008;51(1):107–13 **(50th anniversary issue: 1958–2008)**.
26. Fegaras L, Li C, Gupta U. An optimization framework for map-reduce queries. In: Proceedings of the 15th international conference on extending database technology—EDBT. 2012. p. 26–37.
27. Apache MRQL, the Apache Software Foundation. https://mrql.incubator.apache.org. Accessed 22 Apr 2017.

28. Siddique K, Akhtar Z, Kim Y, Jeong YS, Yoon EJ. Investigating Apache Hama: a bulk synchronous parallel computing framework. J Supercomput. 2017;73(9):4190–205.
29. Katsifodimos A, Schelter S. Apache Flink: stream analytics at scale. 2016 IEEE international conference on cloud engineering workshop (IC2EW). 2016.
30. Shvachko K, Kuang H, Radia S, Chansler R. The Hadoop distributed file system. In: 2010 IEEE 26th symposium on mass storage systems and technologies (MSST); 2010. p. 1–10. http://doi.ieeecomputersociety.org/10.1109/MSST.2010.5496972.
31. Hausenblas M, Nadeau J. Apache drill: interactive ad-hoc analysis at scale. Big Data. 2013;1(2):100–4. https://doi.org/10.1089/big.2013.0011.
32. Apache Drill, the Apache Software Foundation. https://drill.apache.org/.
33. Apache Phoenix, the Apache Software Foundation. https://phoenix.apache.org/. Accessed 01 Oct 2018.
34. Chang L, et al. HAWQ: a massively parallel processing SQL engine in Hadoop. In: Proceedings of the ACM SIGMOD international conference on management of data—SIGMOD'14. 2014. p. 794–7.
35. Kornacker M, et al. Impala: a modern, open-source SQL engine for Hadoop. In: 7th biennial conference on innovative data systems research (CIDR'15). 2015.
36. Llama Installation, documentation for CDH 5.0.x. https://www.cloudera.com/documentation/cdh/5-0-x/CDH5-Installation-Guide/cdh5ig_llama_installation.html.
37. Michael A, et al. Spark SQL: relational data processing in spark. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data. 2015.
38. Zhang X, Khanal U, Zhao X, Ficklin S. Making sense of performance in in-memory computing frameworks for scientific data analysis: a case study of the spark system. J Parallel Distrib Comput. 2017;120:369–82.
39. Cassales GW, Schwertner Charão A, Kirsch-Pinheiro M, Souveyet C, Steffenel L-A. Improving the performance of Apache Hadoop on pervasive environments through context-aware scheduling. J Ambient Intell Humaniz Comput. 2016;7(3):33–345.
40. Robert JS. Performance and programmability comparison of mapreduce query languages: Pig, Hive, JAQL & Java. Master's thesis, Heriot Watt University, Edinburgh, United Kingdom. 2010.
41. Johan U, Konstantin H. Hadoop scripting languages domain specific languages Pig and JAQL. Seminar "Map/Reduce algorithms on Hadoop. 2009.
42. Edward C, Dean W, Jason R. Programming Hive: data warehouse and query language for Hadoop. Sebastopol: O'Reilly Media Inc.; 2012.
43. Query Language for JavaScript(r) Object Notation (JSON). https://code.google.com/archive/p/jaql/.
44. Kabáč M, Consel C, Volanschi N. Designing parallel data processing for enabling large-scale sensor applications. Pers Ubiquit Comput. 2017;21(3):457–73.
45. Cynthia MS, Uttam J. What's the big deal about Big SQL? Introducing relational DBMS users to IBM's SQL technology for Hadoop. https://www.ibm.com/developerworks/library/bd-bigsql/bd-bigsql-pdf.pdf.
46. Hung CY, Dasdan A, Ruey LH, Parker DS. Map-reduce-merge: simplified relational data processing on large clusters. In: SIGMOD'07: proceedings of the 2007 ACM SIGMOD international conference on management of data. 2007. p. 1029–40.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ▶ Convenient online submission
- ▶ Rigorous peer review
- ▶ Open access: articles freely available online
- ▶ High visibility within the field
- ▶ Retaining the copyright to your article

Submit your next manuscript at ▶ springeropen.com