International Journal of Applied Information Systems (IJAIS) ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
Volume 10 - No. 8, April 2016 - www.ijais.org

Advanced SQL Query To Flink Translator

Yasien Ghallab Gouda
Full Professor
Mathematics and Computer Science Department
Aswan University, Aswan, Egypt

Hager Saleh Mohammed
Researcher
Computer Science Department
Aswan University, Asawn, Egypt

Mohamed Helmy Khafagy
Assistant Professor
Computer Science Department
Fayoum University, Egypt

ABSTRACT

Information in the digital world, data play an important role in most of Computer Engineering applications. The increasing of data has been more difficult to store and analyze data using the traditional database. Apache Flink is a framework to Big Data Analytics in the large cluster. SQL-likes Query set of rules for make an interface between the user and big database, so very need to SQL To Flink translator that allow the user to run Advanced SQL Query top Flink without need writing JAVA code to reach their request, and also, Complex SQL Query in Flink is limited scalability. 2. In this paper, the system is devolved to run top Flink without changing in Flink framework. This system calls, Advanced SQL Query To Flink Translator This proposed system receives Advanced SQL Query from the user then generate Flink Code for executing this Query. Finally, it returns the results of Query to the user.

General Terms:

SQL Query, Apache Flink

Keywords

Big data, Flink, SQL Translator, Hadoop, Hive, Advanced SQL Query

1. INTRODUCTION

The size of data in the world has been exploding, and analyzing large data sets so-called Big Data.
The Big Data is huge and complex datasets consisting of a different structured and unstructured data which becomes difficult to store and analysis using traditional techniques database [8].
Big Data requires frameworks to analyze and process datasets such as Hadoop, MapReduce, and Flink.
The Apache Hadoop is open-source software for reliable, scalable, distributed computing runs on distributed cluster. It is developed by Google MapReduce framework [2]. Hadoop consists of HDFS and MapReduce that have a good Load Balance Technique [13, 9].
MapReduce is a programming model for processing large data sets in distributed cluster implementation by Google in 2004 which provides an efficient solution to the data analysis challenge.
The MapReduce framework requires that users implement their applications by coding their map and reduce functions. While this low-level hand coding offers a high flexibility in programming applications, it increases the difficulty in program debugging [3, 12].
Apache Flink is an open source framework for distributed stream and batch data processing run on distributed cluster. Flink core is a streaming data flow engine that provides data distribution, communication, and fault tolerance for distributed computations over data streams. Flink also builds batch processing on top of the streaming engine, overlaying native iteration support, managed memory, and program optimization [1].
Apache Flink has some features the faster than Hadoop, provide input and output of Hadoop and can run Hadoop programming.
SQL-likes Query is some of the rules for makes interface between user and database and helps user for manages and retrieves data from the big database. There are some translators provide SQL Query that translator run above Hadoop such as Hive [16], YSmart[12], S2mart [7], and Qmapper [17].
So the translator is built run above Flink for executing Advanced SQL Query because Advanced SQL Query in Flink is limited scalability
The proposed system run above Flink without any change in Flink structure. The proposed system translates Advanced SQL Query to Flink Code for executing Advanced SQL Query on Flink. The proposed system handles Query that contains some keywords such as Where clauses contain (BETWEEN, AND, OR), Sub Query in Where clauses contains IN, JOIN Types, ORDER BY operation, TOP operation, COUNT Aggregation and Nested Query. Also proposed Technique facilitate many Algorithms and technique to run above Flink [15, 5, 11, 6, 10]
The rest of the paper is organized as follows: Section 2 introduces the related works of relevant systems. Section 3 describes the proposed system architecture and the proposed system methodology. Section 4 represents the results of performed experiments and comparison between the proposed system and Hive. Finally, Section 5 concludes and the brief introduction to future work.

2. RELATED WORK

In this section, an overview is introduced of related work presented so far:

2.1 Hive

Hive, is an open-source data warehousing solution built on top of Hadoop. Hive supports queries expressed in an SQL-like language called HiveQL. HiveQL transforms SQL query into MapReduce jobs that are executed using Hadoop. HiveQL allows users to create customs MapReduce scripts into queries. HiveQL has same features in SQL [16].

2.2 S2MART

Smart SQL to Map-Reduce Translators, Smart transforms the SQL queries into Map-Reduce jobs besides the inclusion of intra-query correlation by building an SQL relationship tree to minimize redundant operations and computations and build a spiral modeled database to store and retrieve the recently used query results for reducing data transfer cost and network transfer cost. S2MART applies the concept of views in a database to perform parallelization of big data easy and streamlined [7].

2.3 QMAPPER

A QMapper is a tool for utilizing query rewriting rules provides a cost-based plan evaluator to choose the optimized equivalent and MapReduce flow evaluation and enhanced the performance of Hive significantly [17].

2.4 SQL TO FLINK Translator

SQL To Flink Translator is a tool built above Apache Flink without effect in Flink structure to support simple SQL Queries. SQL TO Flink Translator receives SQL Query from the user.Then generates the equivalent code for this query that it can be run on Flink. This translator has some limitations such as that SQL to Flink translator cannot translate Advanced Query and can not improve the performance for executing SQL Query [14].

3. ADVANCED SQL QUERY TO FLINK TRANSLATOR

3.1 System Architecture

The central feature of the proposed system is executing the Advanced Query on Flink without write Java Code for executing this Query on Flink. The system architecture is illustrated in Figure 1 that is divided into five phases:
The first phase, The proposed system receives SQL Query from the user. Then Query parser checks SQL Query is correct.
The second phase, the proposed system extracts tables and columns name from the input Query then recalls Java class dataset for each table has only extracted columns.
The third phase, the proposed system extract some keywords from SQL Query such as Where Clauses contain (BETWEEN, AND, OR) keywords,Sub Query in Where clauses contains IN, JOIN Types, ORDER BY operation, TOP operation, COUNT Aggregation and detects Nested Query.
The fourth phase, the proposed system generates Flink Code that executes the input Query.
The last phase, the proposed system executes the Flink Code and returns the result to the user.

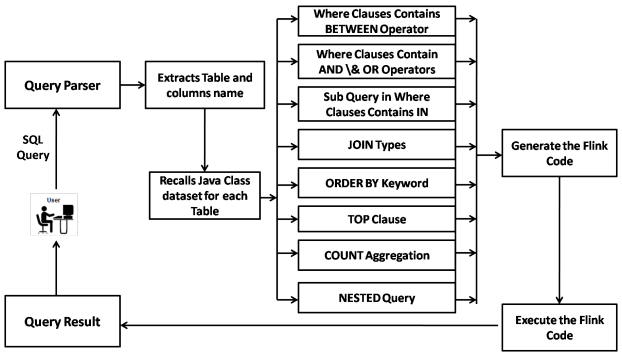

Fig. 1. System Architecture

3.2 Methodology

The proposed system translates Query from a user if Query has Where clauses contain (BETWEEN, AND, OR) operators, Sub Query in Where clauses contains IN, JOIN Types, ORDER BY, TOP Clause, COUNT Aggregation and Nested Query.
Each case is explained to view how the proposed system is handled each case.

3.2.1 Where Clauses Contains BETWEEN Operator. The BETWEEN operator filter values within range. When the Query Parser finds Where Clauses contains BETWEEN operator in input Query such as (see Figure 2). Then the proposed system generates Flink Code by calling Filter function to executing input Query and returns the result from it, (see Figure 3).

```
SELECT C_CUSTKEY, C_ADDRESS ,C_CITY FROM Customers
WHERE C_CUSTKEY BETWEEN 1 AND 50;
```

Fig. 2. SQL Query Contains BETWEEN Operator

```
DataSet<Customer3>
customer=getCustomerDataSet(env,mask);
customer = customer.filter(new FilterFunction<Customer>()
{ @Override
 public boolean filter(Customer c) {
  return  c.f0 >= 1 && c.f0 <= 50;}});
```

Fig. 3. BETWEEN Operator Flink Code

3.2.2 Where Clauses Contain AND & OR Operators . The AND operator filters a dataset if all condition is true. The OR operator filters a dataset if one condition is true. When the Query Parser finds Where Clauses contains AND & OR operators in input Query such as (see Figure 4). Then the proposed system generates Flink Code by calling Filter Function to executing input Query and returns the result from it, (see Figure 5).

```
SELECT C_ADDRESS , C_CITY , C_MKTSEGMENT FROM
Customers WHERE C_MKTSEGMENT="AUTOMOBILE" OR
C_ADDRESS="MG9kdTD2WBHm" ;
```

Fig. 4. Query Contain OR operator

```
DataSet<Customer>
customer=getCustomerDataSet(env,mask);
customer = customer.filter(new FilterFunction<Customer>()
{  @Override  public boolean filter(Customer c) {
if(logicoperation.get(0).toString().equals("and")) return
c.getField(1).equals("MG9kdTD2WBHm") &&
c.getField(2).equals("AUTOMOBILE");
 else  return  c.getField(1).equals("MG9kdTD2WBHm") ||
c.getField(2).equals("AUTOMOBILE");}});
```

Fig. 5. AND & OR Operators Flink Code

3.2.3 Sub Query in Where Clauses Contains IN Keyword. The IN operator allows a user to add multi-values in Where Clauses. When the Query Parser finds Sub Query in Where Clauses contains IN in the input Query such as (see Figure 6).Then the proposed system generates Flink Code by calling coGroup Function and IN_Operator Custom Function to executing input Query and returns the result from it, (see Figure 7).

```
SELECT C_CUSTKEY,C_NAME,C_MKTSEGMENT FROM
Customers WHERE C_CUSTKEY   IN (SELECT O_CUSTKEY from
Orders);
```

Fig. 6. Sub Query in Where Clauses Contains IN Keyword

```
DataSet<Customer> customer=
getCustomerDataSet(env,maskcustomer); DataSet<Order>
order =getOrderDataSetsubquery(env,maskorder);
   DataSet<Result> result   =
customer.coGroup(order).where(0).equalTo(0) .with(new
IN_Operator());
```

Fig. 7. IN Keyword Flink Code

3.2.4 JOIN Types. SQL JOIN uses to combine rows from the multi-table. There is Types of JOIN handles in the proposed system.

—*LEFT OUTER JOIN.*
LEFT OUTER JOIN returns all rows from the left table with matching rows in the right table and returns null values in the right table if not match rows with the left table. When the Query Parser finds LEFT OUTER JOIN in the input Query such as (see Figure 8). Then the proposed system generates Flink Code by calling CoGroup function and JoinType() custom function to executing input Query and returns the result from it, (see Figure 9).

```
SELECT C_NAME, O_ORDERKEY,O_ORDERSTATUS FROM
Customers RIGHT LEFT JOIN Orders ON
Customers.C_CUSTKEY=Orders.O_CUSTKEY;
```

Fig. 8. LEFT OUTER JOIN Query

```
DataSet<Customer> customer =getCustomerDataSet(env,
maskcustomer);
DataSet<Orders> order=getOrdersDataSet( env,maskorder);
DataSet<Result> result =
order.coGroup(customer).where(0).equalTo(0)
.with(new JoinType());
```

Fig. 9. LEFT OUTER JOIN Flink Code

—*RIGHT OUTER JOIN.*
RIGHT OUTER JOIN returns all rows from the right table with matching rows in the left table and returns null values in the left table if not match rows with the right table. When the Query Parser finds RIGHT OUTER JOIN in the input Query (see Figure 10). Then the proposed system generates Flink Code by calling CoGroup function and custom function Join Type() to executing input Query and returns the result from it, (see Figure 11).

```
SELECT C_NAME, O_ORDERKEY,O_ORDERSTATUS FROM
Customers RIGHT OUTER JOIN Orders ON
Customers.C_CUSTKEY=Orders.O_CUSTKEY;
```

Fig. 10. RIGHT OUTER JOIN Query

```
DataSet<Customer> customer =getCustomerDataSet(env,
maskcustomer);
DataSet<Orders> order=getOrdersDataSet( env,maskorder);
DataSet<Result> result =
order.coGroup(customer).where(0).equalTo(0)
.with(new JoinType());
```

Fig. 11. RIGHT OUTER JOIN Flink Code

3.2.5 ORDER BY Keyword. ORDER BY is used to sort results by one column or multi-column, it sorts results in ascending or descending order. When the Query Parser finds ORDER BY in the input Query such as (see Figure 12). Then the proposed system generates Flink Code by calling sortPartion(Fileds number, Order type) to executing input Query and returns the result from it, (see Figure 13).

```
SELECT  C_CUSTKEY , C_ADDRESS , C_MKTSEGMENT FROM
Customers WHERE C_MKTSEGMENT="AUTOMOBILE" ORDER
BY C_CUSTKEY DESC;
```

Fig. 12. Query Contains ORDER BY Keyword

```
DataSet<Customer>
customer=getCustomerDataSet(env,mask);
customer = customer.filter(new
FilterFunction<Customer3>()
{  @Override  public boolean filter(Customer c) {
return   c.f2.equals("AUTOMOBILE")  ;}});
customer= customer.sortPartition(0,
Order.DESCENDING).setParallelism(1);
```

Fig. 13. ORDER BY Keyword Flink Code

3.2.6 TOP Clause. TOP Clause is used to return the specified number of rows. When the Query Parser finds TOP Clause in the input Query such as (see Figure 14). Then the proposed system generates Flink Code by calling the first() function to executing input Query and returns the result from it, (see Figure 15).

```
SELECT  TOP 10 C_CUSTKEY , C_ADDRESS , C_MKTSEGMENT
FROM Customers WHERE C_MKTSEGMENT="AUTOMOBILE";
```

Fig. 14. Query Contains Top Clause

```
DataSet<Customer3>
customer=getCustomerDataSet(env,mask);
customer = customer.filter(new
FilterFunction<Customer3>()
{ @Override public boolean filter(Customer3 c) {
   return c.f2.equals("AUTOMOBILE") ;}});
customer=customer.first(10);
```

Fig. 15. Top Clause Flink Code

3.2.7 COUNT Aggregation. COUNT Aggregation used to return the number of rows in the result. When the Query Parser finds COUNT Aggregation in the input Query such as (see Figure 16). Then the proposed system generates Flink Code by calling the count() function to executing input Query and returns the result from it,(see Figure 17).

```
SELECT COUNT C_CUSTKEY , C_ADDRESS , C_MKTSEGMENT
FROM Customers WHERE C_MKTSEGMENT="AUTOMOBILE;
```

Fig. 16. Query Contains COUNT Aggregation

```
DataSet<Customer>
customer=getCustomerDataSet(env,mask);
customer = customer.filter(new
FilterFunction<Customer3>()
 {  @Override public boolean filter(Customer c) {
return  c.f2.equals("AUTOMOBILE") ;} });
Long number_of_coulmn=customer.count();
 System.out.println(number_of_coulmn);
```

Fig. 17. Count Aggregation Flink Code

3.2.8 NESTED Query. When Query Parser finds sub-select in the input query such as (see Figure 18).Then the proposed system generates Flink code to executing sub-select (see Figure 19) and then the proposed system generates Flink code to executing top select depends on returns values from sub-select (see Figure 20).

```
SELECT C_CUSTKEY,C_NAME
FROM Customers WHERE C_MKTSEGMENT=( SELECT
C_MKTSEGMENT from Customers WHERE C_CUSTKEY<=20);
```

Fig. 18. Query Contain Sub Query

```
DataSet<Customer2> customer2
=getCustomerDataSetsubquery(env,mask1);
customer2 = customer2.filter(new
FilterFunction<Customer2>()
{@Override  public boolean filter(Customer2 c) {
return c.f0 <= 20 ;}});
```

Fig. 19. Sub-select Flink Code

```
DataSet<Customer1>
customer1=getCustomerDataSet(env,mask);
DataSet<Result> result1 =
customer1.join(customer2).where(2).equalTo(1)
.with( new JoinFunction<Customer1, Customer2,Result>() {
@Override
public Result join(Customer1 c1, Customer2 c2) {
return new Result(c1.f0,c1.f1,c1.f2,c2.f0,c2.f1);}});
```

Fig. 20. Top-select Flink Code

4. EXPERIMENTAL RESULTS

4.1 DATA SET AND QUERIES

Using dataset and Queries from TPC-H Benchmark. This benchmark illustrates decision support systems that provides large volumes of data, execute complexity queries, and give answers to critical business questions [4].
Every dataset is split to a different size for executing TPC-H Queries on this dataset.

4.2 ENVIRONMENT SETUP

—A Hadoop Single Node, Ubuntu 9.0.3 virtual machines, and each one running Java(TM) SE Runtime Environment on Netbeans IDE. Hadoop version1.2.1 is installed, and one Namenode, and 2 Datanodes are configured. The Namenode and Datanodes have 20 GB of RAM, seven cores, and 100GB disk. Also, Hive 1.2.1 is installed on the Hadoop Namenode and Datanodes.

—Flink 9 is used, Flink cluster is installed, and one Master Node and two Work Nodes are configured. The Master Node and Worker Node have 20 GB of RAM, seven cores, and 100GB disk.

4.3 Result

Comparison between Advanced SQL Query To Flink Translator and HiveQl when run TCP-H Query 4 and TCP-H Query 13 on different data size.

4.3.1 TCP-H Query 4. In this system, TCP-H Query 4 (see Figure 21) is used because it contains cases that handle in the proposed system.

```
select o_orderpriority, count(*) from orders
where o_orderdate >= '12/1/1996' and o_orderdate <
10/23/1995'  and IN ( select * from lineitem where l_orderkey
= o_orderkey and l_commitdate < l_receiptdate )
group by o_orderpriority
order by o_orderpriority;
```

Fig. 21. TCP-H Query 4

IN Figure 22 show a comparison between Advanced SQL Query to Flink translator and HiveQl when run TCP-H Query 4. Also, Advanced SQL Query To Flink translator enhances performance by average 21%.

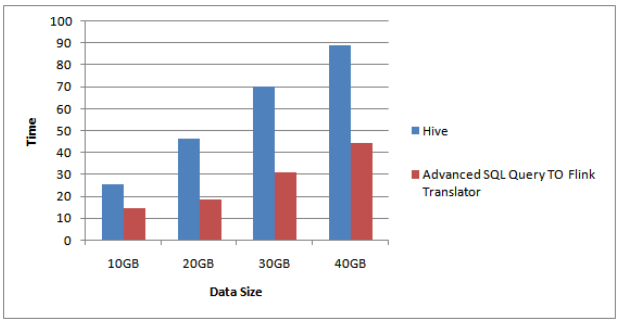

Fig. 22. Compare Between Advanced SQL Query To Flink and HiveQl When run TCP-H Query 4 Using Different Size of Data

4.3.2 TCP-H Query 13. In this system, TCP-H Query 13 (see Figure 23) is used because it contains cases that handle in the proposed system.

```
select c_name
from ( select c_name,c_custkey,count(o_orderkey) from
customer left outer join orders on c_custkey = o_custkey)
group by c_name
order by c_name;
```

Fig. 23. TCP-H Query 13

IN Figure 24 show a comparison between Advanced SQL Query to Flink translator and HiveQl when run TCP-H Query 13. Also, Advanced SQL Query To Flink translator enhances performance by average 29%.

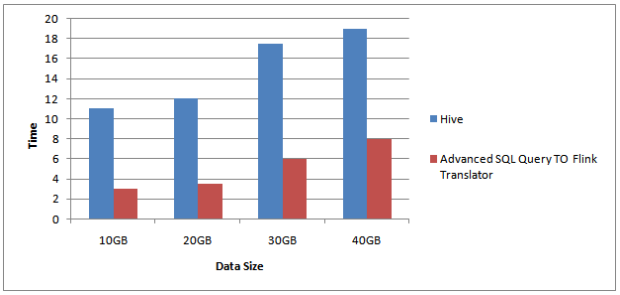

Fig. 24. Compare between Advanced SQL Query To Flink and HiveQl When run TCP-H Query 13 Using Different Size of Data

5. CONCLUSIONS AND FUTURE RESEARCH

Execution of Advanced SQL Query on Flink without write Java Code is very necessary, so developed system that can execute Advanced SQL Query on Flink that has three main stages. First, receiving Advanced SQL Query from the user. Second, generates Flink Code for executing this Query. Third, verified the correctness of system by performing various experimental results using different Queries. Finally, achieve efficiency in all the experimental results. In the future work, build system to executing Advanced SQL Query on Flink in the short time.

6. REFERENCES

[1] Apache flink, 2 2016. https://ci.apache.org/projects/flink/flink-docs-master/.

[2] Apache hadoop, 2 2016. http://hadoop.apache.org/.

[3] Mapreduce, 2 2016. https://cloud.google.com/appengine/docs/python/data

[4] Tpc-h, 2 2016. http://www.tpc.org/tpch/.

[5] Marwah N Abdullah, Mohamed H Khafagy, and Fatma A Omara. Home: Hiveql optimization in multi-session environment. In *5th European Conference of Computer Science (ECCS'14)*, volume 80, page 89, 2014.

[6] Hussien SH. Abdel Azez, Mohamed H. Khafagy, and Fatma A. Omara. Joum: An indexing methodology for improving join in hive star schema. *International Journal of Scientific & Engineering Research*, 6:111–119, 2015.

[7] Narayan Gowraj, Prasanna Venkatesh Ravi, V Mouniga, and MR Sumalatha. S2mart: smart sql to map-reduce translators. In *Web Technologies and Applications*, pages 571–582. Springer, 2013.

[8] Katarina Grolinger, Michael Hayes, Wilson Akio Higashino, Alexandra L'Heureux, David S Allison, Miriam Capretz, et al. Challenges for mapreduce in big data. In *Services (SERVICES), 2014 IEEE World Congress on*, pages 182–189. IEEE, 2014.

[9] Hesham A Hefny, Mohamed Helmy Khafagy, and M Wahdan Ahmed. Comparative study load balance algorithms for map reduce environment. *International Journal of Computer Applications*, 106(18):41–50, 2014.

[10] Mohamed Helmy Khafagy. Index to index two-way join algorithm. *International Journal of Digital Content Technology and its Applications*, 9(4):25, 2015.

[11] Mohamed Helmy Khafagy. Indexed map-reduce join algorithm. *International Journal of Scientific & Engineering Research*, 6(5):705–711, 2015.

[12] Rubao Lee, Tian Luo, Yin Huai, Fusheng Wang, Yongqiang He, and Xiaodong Zhang. Ysmart: Yet another sql-to-mapreduce translator. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 25–36. IEEE, 2011.

[13] Ebada Sarhan, Atif Ghalwash, and Mohamed Khafagy. Queue weighting load-balancing technique for database replication in dynamic content web sites. In *Proceedings of the 9th WSEAS International Conference on APPLIED COMPUTER SCIENCE*, pages 50–55, 2009.

[14] Fawzya Ramadan Sayed and Mohamed Helmy Khafagy. Sql to flink translator. *IJCSI International Journal of Computer Science Issues*, 12(1):169, 2015.

[15] Mina Samir Shanoda, Samah Ahmed Senbel, and Mohamed Helmy Khafagy. Jomr: Multi-join optimizer technique to enhance map-reduce job. In *Informatics and Systems (INFOS), 2014 9th International Conference on*, pages PDC–80. IEEE, 2014.

[16] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[17] Yingzhong Xu and Songlin Hu. Qmapper: a tool for sql optimization on hive using query rewriting. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 211–212. International World Wide Web Conferences Steering Committee, 2013.