Business-oriented customized big data query system and its SQL parser design and implementation

Ge Wenshuai[1,a], He Gang[2] and Liu Xinwen[3]

[1]Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]Beijing University of Posts and Telecommunications, Beijing 100876, China
[3]Wisetone Technologies, Beijing 100876, China

Abstract. This paper proposes a big data query system for customized queries based on specific business needs. This paper introduces the components and structure of the query system. ANTLR tools are used as language recognizer to design and implement a customized SQL dialect. The system builds a simpler and easier query interface on Spark SQL, which satisfies the query requirements of the Internet user behavior analysis platform.

1 Introduction

In recent years, with the increase in the number of mobile Internet users, user data has gradually accumulated, and the application of big data technology has become more and more extensive. Internet user behavior analysis is one of the important applications.

The analysis of user behavior based on big data platform refers to the use of distributed data and analysis technology to efficiently collect, filter, and analyze those data and get the relationship between data and user behavior from the analysis results, thus summing up the user's behavior model. User data typically reaches the order of GB or even TB. So it is reasonable to process user data using big data query platform.

Apache Spark is an in-memory cluster computing engine, and is the most active open source project for big data processing. Spark offers a programming abstraction called Resilient Distributed Datasets (RDDs), which enables efficient data reuse compared to existing models (MapReduce). Recently, a native SQL layer is introduced on top of spark. This module is called Spark SQL. Spark SQL uses a new component called Catalyst and it is the key component of Spark SQL and we will dive deeper into Catalyst in future work[9].

The emergence of big data technologies such as Spark has provided strong technical support for large scale data processing. The query demand is the basic requirement of the Internet user behavior analysis system. Different business fields have different requirements for user data queries. This paper focuses on several common query scenarios of the Internet user behavior analysis system. ANTLR is used as the language recognizer to build a customized SQL query syntax based on Spark SQL. This paper implements a customized query system, which provides a quick and convenient query platform for data analysts to meet the burst of query requirements.

2 Domain specific language on top of Spark SQL

Catalyst is an extensible optimizer for Spark SQL. It provides the convenient inject point for external developers to add customized analyzation and optimization rules. A query in is first parsed into an Abstract Syntax Tree (AST), then this AST is transformed into a logical plan. The logical plan is then optimized by optimizer with some predefined optimization rules, such as constant folding, combine filters , column pruning and predicate pushdown. Spark SQL generates one or more physical plans from the optimized logical plan. Cost model is used to select one for physical execution. Spark SQL also adopts code generation to optimize the final plan for better performance. Figure 1 shows the work flow of Catalyst in Spark SQL as explained above[11]:

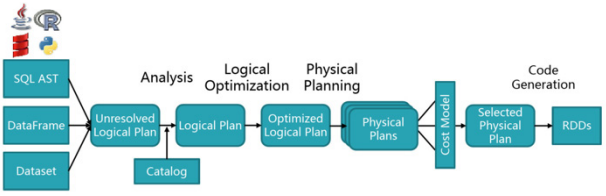

Figure 1 workflow of Catalyst in Spark SQL

The general step of using Spark to process big data problems is to confirm the business demand first, then write specific java or scala code according to the query logic, then submit the query job for execution at the end.

[a] Corresponding author: gws@bupt.edu.cn

Once the query logic has changed, or a query condition changes, the java or scala code must be rewritten. In this case, writing such business code on changing demands is very inefficient. On the contrary, defining a Domain Specific Language (DSL) can be a good solution because its abstraction and generalization of these similar business logic. It frees up developers from boring and repetitive business logic code, and improves efficiency.

In software development, domain specific language is a type of programming language dedicated to a particular problem domain, a specific problem representation technique or a specific solution technology. It is designed specifically for a particular problem of computer language.

This system defines such a dedicated query syntax on top of Spark SQL. We abstract the general query scenario in the user behavior analysis system according to the specific requirements. ANTLR is used as the syntax analysis tool to define such kind of syntax. The query written in this syntax is then parsed to build query job based on Spark SQL. This system is generally an outer application on top of Spark. The following sections will focus on the structure of the system and the design and implementation of the SQL parser.

3 The structure of the system and its components

Spark's in-memory computation gives 100 times faster performance and 10 times faster when running on disk compared to the MapReduce engine. Thus, this paper selects Spark SQL as the query engine.

The system mainly included modules are web user interface, web server, metadata storage, SQL parsing, and job submission modules. The overall structure of the system is shown in the figure 2:

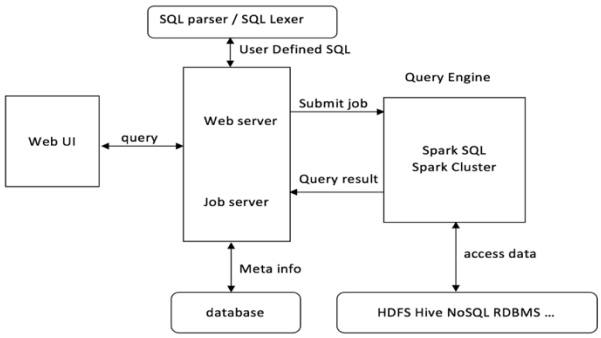

Figure 2. structure of the customized query system

3.1.Web UI and web server

Many open source big data systems such as Hadoop and Spark provide users with a friendly web interface, which is easy to user. We choose Spring, Spring MVC and MyBatis as the web framework.

The web UI provides two kinds of query methods. One is to input customized SQL-like query strings directly in the text box[8][10]. This method directly submits query commands to the backend server. The other is to build query commands through a combination of web components, such as drop-down boxes, check boxes, etc. The frontend web page communicates with the backend server mainly through Ajax. The frontend of the system is a single-page application, using Ajax to avoids frequently reloading the page and thus providing users with better experience. The backend uses Spring MVC's RequestMapping annotation to provide RESTful API interfaces. These interfaces include '/meta/*', '/user/*', '/job/*', etc. API '/meta/*' provides Metadata information, including job submission history, query result downloading, HDFS directory structure, and file metadata access interface; API '/user/*' provides interfaces of user login, registration, verification; And API '/job/* is used to submit query jobs to the backend.

3.2.Metadata storage

In Internet user behavior analysis platform, HDFS is used as the file storage system. Data is collected and filtered and stored in HDFS as structured text file. The storage location of data has a certain regularity. Structured files of different structure types are stored in different locations. Different structure files correspond to different data tables in the system. Therefore, it is necessary to use a metabase to store the meta information of the structured files in HDFS.

Metadata stores the data describing the structured files in HDFS, including the file storage location, table name, column name, column data type, index, description information, etc. At the same time, the metabase also serves as the backend database of the web server, and stores registered user information, query result information and so on. In addition, metabase also provides a database-like namespace to support table with the same name in different databases.

This metabase refers to the design of Hive's metadata table, and we use MySQL as the database. The design of the metabase is as shown in the figure 3. The `store_tables` is the schema table of the text files in different structures in HDFS, including the file location in HDFS, the table name, the creation time, the owner. Schema `columns_details` is the detail of columns in these structured text files, including column name, column type, column index, column label name, etc. Schema `cds` corresponds to all columns of each `store_tables`, and the `exec_history` table stores detail of all queries, including query command, query result location, query status, acceptance time and other information. It also includes an `auth_user` table, which stores user information for the query system, including user names, user passwords, and other information.

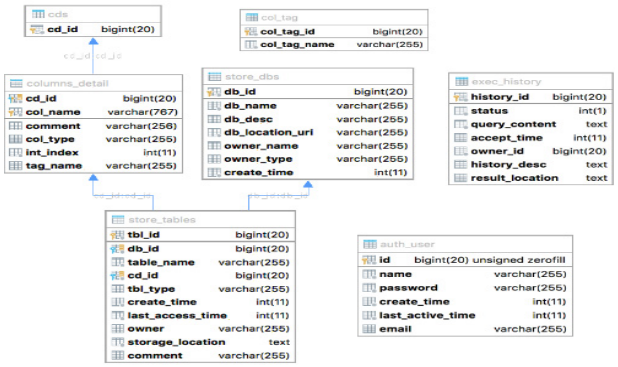

Figure 3 Metadata table

3.3.SQL parsing

This system uses ANTLR (Another Tool for Language Recognition) as the tool for language recognition. ANTLR is a widely used syntax recognizer. Many excellent open source software use ANTLR as a parsing tool, such as Spark SQL, Hive, Presto, etc. ANTLR defines the SQL grammar rules with an extended Backus-Naur Form (EBNF), then ANTLR generates parsers based on the grammar rules, including lexer, parsers, and tree visitors. Figure 4 illustrates the basic data flow of a language recognizer[1][2][3]:

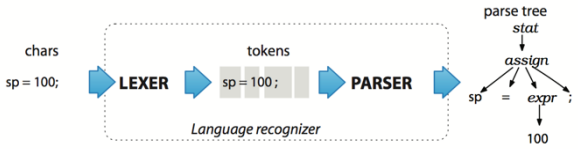

Figure 4 data flow of a language recognizer

The input query text is parsed into a series of tokens by the lexer. If the user's input text does not match the pre-defined lexical rules, errors can be detected. ANTLR supports embedding the target language code in the lexical file for detecting and processing these errors. If no lexical error is detected, the token stream is then passed to the parser and a corresponding syntax tree is generated. ANTLR provides two ways to traverse the syntax tree. One is the listener mode and the other is the visitor mode. Compared to the listener mode, using the visitor mode is more flexible, Figure 5 describes the process of accessing the syntax tree by visitor[6]. We can control the traversal path and explicitly call the methods to access the subtree. This system uses visitor mode to access the syntax tree. The SQL parsing module retrieves the various components of the SQL string by calling the tree visitor, and stores the result in an auxiliary data structure. The object-oriented encapsulating module encapsulates SQL into a java object by accessing an auxiliary data structure generated by the SQL analysis module. For example, a select statement query string may be separately packaged into a Java language objects, then we build actual query job on top of Spark SQL query engine with this object[7].

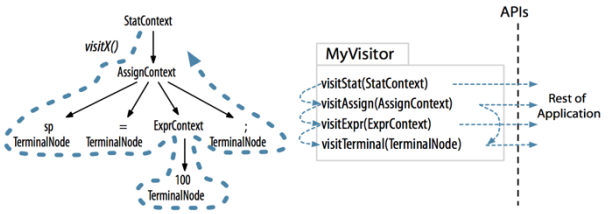

Figure 5 control flow of Parse-Tree Visitors

3.4.Job submission

After the input query text is parsed into a syntax tree, we use a visitor to traverse this syntax tree. All query information we need to build the query task will be packaged into Java objects. Using these Java objects as parameters, we can write the code for the actual query task, then package the code and submit it to Spark cluster. We use a separate open source project called Spark-Jobserver for job submission, which provides RESTful interface for submitting and managing Apache Spark jobs. Spark can be offered as a service to anyone in a simple way. End user can manipulate the Spark jobs at his own convenience through the REST API. Spark-jobserver supports Spark SQL, Hive, Streaming contexts/jobs, and customized job contexts. Besides, it provides asynchronous and synchronous job APIs for job submission. The synchronous API is very effective for low latency jobs. Spark jobserver works with standalone spark as well on cluster, mesos, yarn client mode[4].

Since the data scale of Internet user behavior analysis system is very large, it is impossible to return the query results immediately after job submission. Therefore, we choose the asynchronous job API. After submitting a job in this way, Spark-jobserver will immediately return a jobId, so we need to check the execution result of the job accroding to this jobId. The job status includes STARTED, RUNNING, FINISHED, and KILLED. When the job status becomes FINISHED, the job execution is completed, so we can get the result of the query job and statistics of the job execution.

In order to obtain the job execution state, we need to define a thread pool in the Spring container. Whenever a job is submitted, we take a thread from the thread pool to check the execution result of the job periodically. When the job is completed, this thread will store the result information of this job to the MySQL database.

4 An example of customized query and SQL parser implementation

Data of this system is stored in HDFS. Figure 6 shows the directory structure of `aWifi` data on HDFS. `aWifi` is a namespace, which contains different structured data such as `authLog`, `netLog`, and so on. There are different namespaces such as `wjpt` and `wfpt` in the system, which is equivalent to the concept of database in a relational database. A folder is generated every day for every structured file that belongs to `aWifi`. The data of different structures are kept under the directory named after the date. Each file ending with ".ok" is tab-

delimited structured data, and its schema information is stored in the MySQL database. Each structured file is similar to a table in database.

```
/user/aWifi/20171117
/user/aWifi/20171118   /user/aWifi/20171117/authLog.ok
/user/aWifi/20171119   /user/aWifi/20171117/authLog2016.ok
/user/aWifi/20171120   /user/aWifi/20171117/netIdLog.ok
/user/aWifi/20171121   /user/aWifi/20171117/netLog.ok
/user/aWifi/20171122   /user/aWifi/20171117/netLog2016.ok
```

Figure 6 structure of data storage directory

Now we choose a common query scenario in the Internet user behavior analysis system as an example: to extract related records during a period of time based on the phone number, Mac, or accounts. A period of time refers to the date in the storage path, for example, 20171117 in the figure 6.

We add a tag to each field in each table. Different tags have different meanings, for example, mobile phone number, Mac address, or timestamp. We can select the desired record based on these tags. For example, we need to select all records from the table `wj_post` that contain Mac address dc-32-83-a6-d4-43 from November 17, 2017 to November 22, 2017. We can naturally think of using a select clause, such as `select Mac = `dc-32-83-a6-d4-43` from wj_post`. `Mac` is the tag name, and `dc-32-83-a6-d4-43` is the desired value. We don't have to care about which field in the `wj_post` data table represents the Mac address, because such information is stored in the metastore, and the backend program will take care of this for us. Users can easily select the data they need. In addition, we also need to take the time into account, so we can add a time clause: `time between 20171117 and 20171122` in the query statement. Thus, we have located the data source we are looking for. We could also define where the query results should be saved. The `saveto /user/result` clause can specify where we want to save the query results on HDFS. Thus, the complete SQL statement is `select Mac = dc-32-83-a6-d4-43 from wj_post time between 20171117 and 20171122 saveto /user/result`. This statement is parsed into a syntax tree as is shown in figure 7.

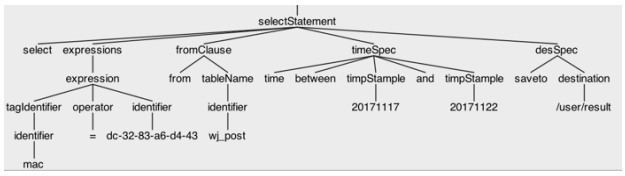

Figure 7 syntax tree built from input query text

Then we need to define the visitor. The java class of `MainAntlrVisitor` in Figure 9 is a visitor that was written according to the syntax parsing rules in Figure 8. We use this visitor to traverse the syntax tree to get the desired values, time ranges, and tag names. Then pass these fields as arguments to Spark–jobserver. With those arguments, Spark-jobserver packages the query job and submits it to the spark cluster to execute the query. The metadata needs to be accessed to package the query task, so we need to access MySQL database using MyBatis. The figure 8 shows the syntax analysis rules designed for this query scenario. The lexical rules are omitted for a simplified layout. The figure 9 shows the lexer, parser, and base visitor that ANTLR automatically generated for us according to the grammar rules in file SSQL.g4.

```
grammar SSQL;

root : statements;

statements : statement+;
statement  :   selectStatement;
selectStatement :   SELECT expressions fromClause timeSpec desSpec;
expressions   :   expression (COMMA  expression)*;
expression    :   tagIdentifier operator  identifier;
operator    :   EQ   |   LIKE  |   NE   | GT   | LT ;
fromClause  :   FROM name=tableName ;
timeSpec    :   TIME BETWEEN begindate=timpStample AND   enddate=timpStample ;
timpStample :   TIMESTAMPLE ;
desSpec     :   SAVETO path=destination;
destination :   IDENTIFIER | SLASH;
tableName :   (db=identifier DOT)? table=identifier   ;
tagIdentifier   :   tag=identifier ;
identifier  :  IDENTIFIER   | QUOTE_IDENTIFIER;
```

Figure 8 parser rules

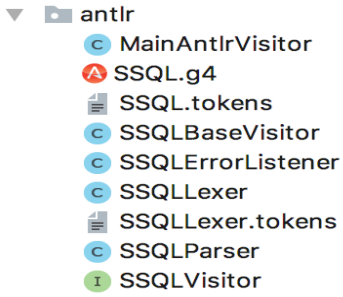

Figure 9 parser, lexer and visitor

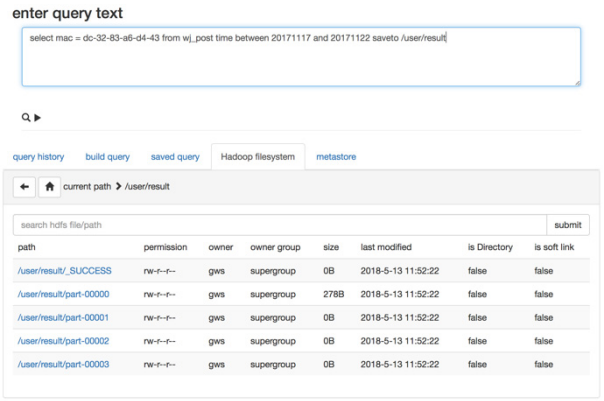

Figure 10 web UI of the query system

Figure 10 shows the web interface of the query system. Users enters a query from the text box, and the frontend submits the query to the backend through Ajax. The query result is saved to HDFS. Besides, the web interface also provides a simple HDFS file browsing module, which provides file preview and download functions.

Compared to the traditional big data query development process, using this platform for big data query has the advantage of rapid development. The system's web interface also allows developers to perform query tasks at anytime, anywhere. The system has been deployed in production environment and has a certain practical value.

5 Correctness and performance test

5.1.Correctness Verification

We design test data and SQL statements according to grammar rules in Figure 8. To verify the correctness of the query system, we also write the corresponding MapReduce code. We prepare a table 'wj_post' as described in section 4 with three data volumes.

Table 1 exhibits an example of verifying the correctness of the system. The system is called SSQL as below.

Table 1 Correctness verification

NO	SSQL	MapReduce	Expected result	Experiment result
1	select mac = dc-32-83-a6-d4-43from wj_post time between 20171117 and 20171122 saveto /user/result	Mapper And Reducer	1	1

5.2.Performance Test

Since SSQL is based on Spark SQL and conforms to the workflow of Spark, we hold that SSQL should maintain the high performance of Spark. Thus we adopt the same way to test performance of Hadoop and SSQL. We take MapReduce and SSQL to execute the same query, and compare the elapsed time of the execution. The experiment steps are as follows:

1.The program reads input data from HDFS.

2.The program computes queries.

3.The program outputs results to HDFS.

The computing scale of the test data increases progressively. The experiment result is shown in Table 2.

Table 2 Performance test

Data scale(Lines)	MapReduce(Time)	SSQL(Time)
960,000	40.0s	18.0s
4,800,000	65.0s	45.0s
24,000,000	210.0s	157.0s

The reason why SSQL does not display great computing advantage is that the calculation process lasts for a small amount of time. Spark has a great advantage in memory computing optimization and data scheduling over Hadoop. However, a large portion of time is spent on distributed task deployment and read/write the file. But with the increase in the computing scale, the portion of the calculation process grows and the performance advantage of Spark is growing prominent. In conclusion, this system maintains the high performance of Spark system[11].

6 Conclusion and future work

This paper proposes a customized big data query system for Internet user behavior analysis platform. The query system includes frontend interface, backend Spring server, metabase, job submission module, SQL parser module.

We research the mechanism of applying domain specific language on top of Spark SQL. Then we implement a prototype system and test the correctness and performance of it. Our work lay the ground for a more comprehensive extension on Spark SQL in the future.

In addition, SSQL has been deployed in production environment, which reduces the time cost of building the query task and improves the efficiency. The future work will be implementing more features for it. Besides the data manipulation language (DML), we will also define data definition language (DDL), data control language (DCL) for SSQL in the future.

We will dive deeper into Spark SQL to combine SQL parser with Catalyst in future work. With Spark extension injection mechanism provided by the release in the Spark 2.2.0 version, we can inject customized SQL parser, optimizer rules and planner strategies into SparkSession. The injection mechanism will give us a more flexible way to customize Spark SQL.

References

1. W. Haiyan, Y. Hebiao. *SQL parser strategy and implementation based on ANTLR* [J]. Computer Applications and Software, **11**: 68-70, 101, (2013)
2. C. Danyang, B. Donghui. *Design and implementation for SQL parser based on ANTLR[C]*, ICCET 2010, 2381-2384.
3. Terence Parr. *The Definitive ANTLR 4 Reference, 2nd Edition.* https://pragprog.com/book/tpantlr2/the-definitive-antlr-4-reference
4. *Spark Jobserver: REST Job Server for Spar*k. https://github.com/ooyala/spark-jobserver
5. Li Changqing, Gu Jianhua. *An ANTLR based SQL transformation model of MongoDB database[J]*. Journal of Northwestern Polytechnical University, 2017, **35**(01):143-147.
6. ANTLR offical website: http://www.antlr.org/
7. Sunkle, Sagar. *Generating highly customizable SQL parsers*. Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management. ACM, 2008
8. Fan, Ju, Guoliang Li, and Lizhu Zhou. *Interactive SQL query suggestion: Making databases user-friendly*. Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011
9. Armbrust, Michael. *Spark sql: Relational data processing in spark*. Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, 2015

10. E. Altuncu, B. K. Bilgehan, Y. S. Kartal, S. Kızılgüneş, M. S. Nikoo and H. Oğuztüzün, *Blocks and text integration in a language-based editor for a domain-specific language*, 2017 ICCSM (UBMK), Antalya, 2017, pp. 1084-1089
11. Meng Q., Ma X., Lu W., Yao Z. *A Spatial SQL Based on SparkSQL*. Geo-Spatial Knowledge and Intelligence. GRMSE 2016. CCIS, vol **698**. Springer, Singapore, (2017)