Demonstration of LogicLib: An Expressive Multi-Language Interface over Scalable Datalog System

Mingda Li
UCLA
United States
limingda@cs.ucla.edu

Jin Wang
UCLA
United States
jinwang@cs.ucla.edu

Guorui Xiao
UCLA
United States
grxiao@ucla.edu

Youfu Li
UCLA
United States
youfuli@cs.ucla.edu

Carlo Zaniolo
UCLA
United States
zaniolo@cs.ucla.edu

ABSTRACT
With the ever-increasing volume of data, there is an urgent need to provide expressive and efficient tools to support Big Data analytics. The declarative logical language Datalog has proven very effective at expressing concisely graph, machine learning, and knowledge discovery applications via recursive queries. In this demonstration, we develop Logic Library (LLib), a library of recursive algorithms written in Datalog that can be executed in BigDatalog, a Datalog engine on top of Apache Spark developed by us. LLib encapsulates complex logic-based algorithms into high-level APIs, which simplify the development and provide a unified interface akin to the one of Spark MLlib. As LLib is fully compatible with DataFrame, it enables the integrated utilization of its built-in applications and new Datalog queries with existing Spark functions, such as those provided by MLlib and Spark SQL. With a variety of examples, we will (i) show how to write programs with LLib to express a variety of applications; (ii) illustrate its user experience in Apache Spark ecosystem; and (iii) present a user-friendly interface to interact with the LLib framework and monitor the query results.

1 INTRODUCTION
In the past years, there is a resurgence of Datalog due to its ability to specify declarative data-intensive applications that execute efficiently over different systems and architectures [13]. The recent theoretical advances [6, 12, 14] enable the usage of aggregates in recursions, and this leads to considerable improvements in the expressive power of Datalog. In response to supporting Datalog queries over large-scale datasets, many parallel and distributed Datalog engines have been developed by researchers from both academia and industry. Following this line of efforts, we have proposed BigDatalog [4], a Datalog engine on top of Apache Spark [11]. Similar with the rich libraries provide by Spark such as Spark SQL, GraphX, MLlib and SparkR, BigDatalog provides the query interface of Datalog which can benefit from the efficient in-memory computation for analytical workloads with complicated recursions.

On the basis of above efforts for Datalog system, we further develop several advanced applications upon it. DatalogML [9] supports training different machine learning models with gradient descent written in Datalog queries by enabling mutual and non-linear recursion in BigDatalog. KDDlog [5] aims at developing data mining algorithms such as frequent item set mining, clustering and decision tree in Datalog. Both of them have shown superior performance than the original machine learning library MLlib in Apache Spark. Except for Datalog, we also build the RaSQL [4, 10] language and system to enable expressing aggregating in recursive SQL queries. However, it requires to develop more easy-to-use APIs as that of MLlib so as to make them compatible to the Apache Spark ecosystem and used by broader audience.

In this demonstration, we propose Logic Library (LLib), a library of applications written in Datalog programs built on top of the BigDatalog system. To be more specific, LLib not only includes popular recursive algorithms like the graph search ones [3], but also machine learning and data mining algorithms introduced in [9] and [5] that is supported by MLlib now. It provides a unified interface that is similar with MLlib so that it is rather easy for the audience of Spark community to use it. Users can utilize the built-in functions that encapsulate Datalog programs to express a variety of applications by using the DataFrame APIs. LLib also supports APIs in multiple programming languages such as Scala, Java and Python. To sum up, the LLib framework enables users to use the highly efficient Datalog applications in the similar way of those...
in Apache Spark MLlib thus significantly improve the usability of Datalog based systems.

2 THE LOGIC LIB FRAMEWORK

2.1 The BigDatalog System

As introduced above, the LLib framework is built based on BigDatalog [8], which is a distributed Datalog engine on top of Apache Spark. In this section, we first give a brief introduction of the system.

BigDatalog supports relational algebra, aggregation and recursion, as well as a host of declarative optimizations. It uses and extends Spark SQL [1] operators, and also introduces a novel recursive operator implemented in the Catalyst framework so that its planning features can be used on the recursive plans of Datalog programs. BigDatalog can resolve recursion in the compilation step by recognizing recursive tables when building the operator tree and evaluate the recursive program using the semi-naive method [7].

By enabling non-linear and mutual recursion, we can support expressing more complicated applications like machine learning [9] and data mining [5] using DATALOG on the above systems.

2.2 Overview of LLib

The overall architecture of LLib is shown in Figure 1. Similar with that of MLlib, we encapsulate applications written in Datalog programs into functions which can be imported and called in programs written in high-level programming languages. As a result, these can be seamlessly integrated with other features of the Apache Spark ecosystem. For example, in an LLib application, there can be steps consisting of regular DataFrame operations, the data transformations from MLlib and functions written in the Datalog provided by LLib. Within a pipeline illustrated in Figure 1, all steps from preparing Datalog input data to persisting and loading Datalog execution results for subsequent processing can be sequentially executed without extra programming efforts.

Similar with that of MLlib, LLib also supports the interfaces for multiple programming languages, i.e., Java, Scala and Python. Since Java is interpretable with Scala, it is relatively simple to support Java interface. The remaining gaps between Scala and Java version are mainly the conversions of data collections. We bridge these gaps using collections known by both languages in Scala implementation and using the implicit converting mechanism in Scala. To support Python interface, we utilize Py4J 1, a bridge between Java and Python language, and design a Removal and Recovery mechanism where the transferred DataFrame in PySpark will be converted to a common data collection type acceptable by both Java and Python languages. Then during execution in Java, we could infer the schema or directly get the schema from users.

2.3 Programming with LLib

Next we will describe how to write a program with built-in functions in LLib through the example of Transitive Closure.

2.3.1 Working Session and Acquiring Data. In LLib, the first step is to construct a working environment for the Datalog queries and libraries. We follow the paradigm of building Spark Session in a similar way as that shown Figure 2.

[1] https://www.py4j.org/index.html

2.3.2 Initializing an Executable Object and Schema Mapping. In LLib, the data processing pipeline for a typical application is wrapped into an executable object, for which users control initialization and parameter setting. We initialize a transitive closure object tc with the TC function. Then, we can set the property in the way shown in Figure 4.

Among these built-in functions, all the libraries need the function setDirection, an extension to existing DataFrame made by us, for schema mapping. One attribute may have different names in different DataFrames. With schema mapping, we could know the corresponding relationship between input DataFrames and the attributes in LLib. For example, in Query 1 shown in Figure 3 there are two attributes From and To in the arc table. They can be called in different ways such as (Node1, Node2) as shown in Figure 4. Then the mapping between (From, To) and (Node1, Node2) is provided by the setDirection function.

2.3.3 Execution and Persistence. With the executable object and imported data, the execution and persistence can be merely a one-line execution with a pre-defined function (run, genDF or genRDD) defined by us. These three functions are implemented for each library in LLib. They expect the input data (DataFrame) and the environment (Session) as inputs. Their functionality satisfies all basic requirements needed to operate on data and preserve the result as a variable or a file. As shown in Figure 5, the function run executes the logical programs and persist the result directly to the target address. It can also output the data into a DataFrame or RDD as shown in second and third line of Figure 5. In this way, users can also apply other APIs provided by Apache Spark for further post-processing or execution.

3 APPLICATIONS

In this section, we introduce two typical applications of LLib. Due to the space limitation, we omit the full Datalog programs here and just provide the references for them, respectively.

3.1 Recursive Algorithm

We first show how to support a typical recursive algorithm in LLib.

Here TC is a built-in function of LLib for Transitive Closure written in Datalog shown in Figure 3. LogiclibSession synthesizes the Spark environment with required designs for logical programs. Within the same session, users can use both Spark libraries like data loading functions to create a DataFrame df (line 4) and the functions in our LLib framework.

3.2.2 Initializing an Executable Object and Schema Mapping. In LLib, the data processing pipeline for a typical application is wrapped into an executable object, for which users control initialization and parameter setting. We initialize a transitive closure object tc with the TC function. Then, we can set the property in the way shown in Figure 4.

Among these built-in functions, all the libraries need the function setDirection, an extension to existing DataFrame made by us, for schema mapping. One attribute may have different names in different DataFrames. With schema mapping, we could know the corresponding relationship between input DataFrames and the attributes in LLib. For example, in Query 1 shown in Figure 3 there are two attributes From and To in the arc table. They can be called in different ways such as (Node1, Node2) as shown in Figure 4. Then the mapping between (From, To) and (Node1, Node2) is provided by the setDirection function.

3.3 Execution and Persistence. With the executable object and imported data, the execution and persistence can be merely a one-line execution with a pre-defined function (run, genDF or genRDD) defined by us. These three functions are implemented for each library in LLib. They expect the input data (DataFrame) and the environment (Session) as inputs. Their functionality satisfies all basic requirements needed to operate on data and preserve the result as a variable or a file. As shown in Figure 5, the function run executes the logical programs and persist the result directly to the target address. It can also output the data into a DataFrame or RDD as shown in second and third line of Figure 5. In this way, users can also apply other APIs provided by Apache Spark for further post-processing or execution.

3 APPLICATIONS

In this section, we introduce two typical applications of LLib. Due to the space limitation, we omit the full Datalog programs here and just provide the references for them, respectively.

3.1 Recursive Algorithm

We first show how to support a typical recursive algorithm in LLib.

Here TC is a built-in function of LLib for Transitive Closure written in Datalog shown in Figure 3. LogiclibSession synthesizes the Spark environment with required designs for logical programs. Within the same session, users can use both Spark libraries like data loading functions to create a DataFrame df (line 4) and the functions in our LLib framework.

3.2.2 Initializing an Executable Object and Schema Mapping. In LLib, the data processing pipeline for a typical application is wrapped into an executable object, for which users control initialization and parameter setting. We initialize a transitive closure object tc with the TC function. Then, we can set the property in the way shown in Figure 4.

Among these built-in functions, all the libraries need the function setDirection, an extension to existing DataFrame made by us, for schema mapping. One attribute may have different names in different DataFrames. With schema mapping, we could know the corresponding relationship between input DataFrames and the attributes in LLib. For example, in Query 1 shown in Figure 3 there are two attributes From and To in the arc table. They can be called in different ways such as (Node1, Node2) as shown in Figure 4. Then the mapping between (From, To) and (Node1, Node2) is provided by the setDirection function.

3.3 Execution and Persistence. With the executable object and imported data, the execution and persistence can be merely a one-line execution with a pre-defined function (run, genDF or genRDD) defined by us. These three functions are implemented for each library in LLib. They expect the input data (DataFrame) and the environment (Session) as inputs. Their functionality satisfies all basic requirements needed to operate on data and preserve the result as a variable or a file. As shown in Figure 5, the function run executes the logical programs and persist the result directly to the target address. It can also output the data into a DataFrame or RDD as shown in second and third line of Figure 5. In this way, users can also apply other APIs provided by Apache Spark for further post-processing or execution.

3 APPLICATIONS

In this section, we introduce two typical applications of LLib. Due to the space limitation, we omit the full Datalog programs here and just provide the references for them, respectively.

3.1 Recursive Algorithm

We first show how to support a typical recursive algorithm in LLib.

Example 1 (Multi-level Market (MLM) Bonus). In the organization, new members are recruited by and get products from old members (sponsors). One member’s bonus is based on his own sales and a proportion of the sales from the people directly or indirectly recruited by him. The scale of the proportion is user-defined. There are two relations in MLM Bonus, including the sponsor and sales. The sponsor relation describes the recruiting relationship among members, while the sales relation records the profits for each member. In Datalog syntax, the base case should be calculating the member’s bonus by the sales table. And the recursive rule is to calculate the bonus based on the basic profits and the profits derived from the downstream members.
Demonstration of LogicLib

Under the LLib framework, users could implement this program by a built-in function MLM. The full program of MLM can be found in Program 10 of [8]. The program using LLib is detailed in Figure 6.

Here we start with two relations Sales and Sponsor stored in DataFrame. We first build an executable object of MLM and then set the schema mappings for two relations with functions setDirection and setSecDirection, which is a DataFrame function defined by us that is similar with setDirection. To operate the data and persist to resMLM file, we use the run function in the 4-th line.

Since most graph algorithms can be also easily expressed with Datalog in a similar way [2], the LLib framework can support them in a similar way.

3.2 Machine Learning and Data Mining

Then we show how to express machine learning applications with LLib. The example in Figure 7 expresses the process of training a Logistic Regression classifier on the training data dataDTrain, and making prediction on the test data, dataDTest. Here DatalogLR is the Datalog program for logistic regression which can be found in Query 3 of [9]. To use Datalog programs for machine learning, we first construct a working environment, i.e. LogiclibSession, for our library of machine learning algorithm (line: 1 to 3). After importing the required training and predicting functions for Logistic Regression (line: 14 to 15), we can build executable objects for training lr (line 16) and predicting lrPredict (line: 22). The lr object wraps all the logical rules and required relations (e.g. parameters with default value 0) of the Datalog implementation for Logistic Regression. When initializing lr, users can exploit the built-in functions to set the hyper-parameters that control the maximum number of iterations, the method used for parameter initialization, and many others. After fitting the model to df, the lrPredict object could make predictions on the testing instances with the pre-trained model, lrModel. In both the fitting and predicting processes, the information of Datalog execution runtime can be obtained by using session as an input argument, which is same as the practice of Apache Spark.

For the sake of comparison, we also show how Apache Spark MLlib will be used to implement the above example. The snippet

```scala
1 val appMLM = new MLM()
2 appMLM.setDirection(MCol = "Member", ProfitCol = "Bonus")
3 appMLM.setSecDirection(MCol = "Mem1", M2Col = "Mem2")
4 appMLM.run(Array(Sales, Sponsor), output = "result", session)
```
are welcome to come up with the program by encapsulating new Datalog programs into user defined functions. We have imported all supported functions in the second line of the upper text box shown in Figure 9. Users can feel free to use other functionalities of DataFrame provided by Apache Spark. We also support three kinds of programming languages in this demonstration: Python, Scala and Java. Users can switch the language by clicking the options in the bottom left.

Our plan of demonstration consists of two steps:

Firstly, we will give a brief introduction to the functionalities of LLib, in which we will exhibit the key components and features of our framework. To this end, we will explain the detailed steps of initializing the session and schema for some simple graph queries, such as transitive closure and connected components.

Secondly, we will demo how to interact with the system and test its capabilities. We will show users how to write the programs with algorithms provided by LLib freely on different kinds of applications, including graph algorithm, machine learning and data mining algorithms. We also demonstrate different programming languages to implement the same program and how system works with libraries like MLlib so as to gain similar user experience with the official library in Apache Spark.

REFERENCES

[1] Michael Armbrust, Reynold S. Xin, and Cheng Lian et al. 2015. Spark SQL: Relational Data Processing in Spark. In SIGMOD: 1383–1394.

[2] Tyson Condie, Ariyam Das, Matteo Interlandi, Alexander Shkapsky, Mohan Yang, and Carlo Zaniolo. 2018. Scaling-up reasoning and advanced analytics on BigData. TPLP 18, 5-6 (2018), 806–845.

[3] Ariyam Das, Youfu Li, Jin Wang, Mingda Li, and Carlo Zaniolo. 2019. Big Data Applications from Graph Analytics to Machine Learning by Aggregates in Recursion. In ICLP. 273–279.

[4] Jiaqi Gu, Yugo Watanabe, William Maoza, Alexander Shkapsky, Mohan Yang, Ling Ding, and Carlo Zaniolo. 2019. RaSQL: Greater Power and Performance for Big Data Analytics with Recursive-aggregate-SQL on Spark. In SIGMOD. 467–484.

[5] Youfu Li, Jin Wang, Mingda Li, Ariyam Das, Jiaqi Gu, and Carlo Zaniolo. 2021. KDDLog: Performance and Scalability in Knowledge Discovery by Declarative Queries with Aggregates. In ICDE. 1260–1271.

[6] Mirjana Mazuran, Edoardo Serra, and Carlo Zaniolo. 2013. Extending the power of datalog recursion. VLDB J. 22, 4 (2013), 471–493.

[7] Marianne Shaw, Paraschos Koutris, Bill Howe, and Dan Suciu. 2012. Optimizing Large-Scale Semi-Naïve Datalog Evaluation in Hadoop. In Datalog in Academia and Industry. 165–176.

[8] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. 2016. Big Data Analytics with Datalog Queries on Spark. In SIGMOD. 1135–1149.

[9] Jin Wang, Jiaqi Gu, Ariyam Das, and Carlo Zaniolo. 2021. Formal semantics and high performance in declarative machine learning using Datalog. VLDB J. 30, 5 (2021), 859–881.

[10] Jin Wang, Guorui Guo, Jiaqi Gu, and Carlo Zaniolo. 2020. RASQL: A Powerful Language and its System for Big Data Applications. In SIGMOD. 2673–2676.

[11] Matei Zaharia, Moshafar Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In NSDI. 15–28.

[12] Carlo Zaniolo, Ariyam Das, Jiaqi Gu, Youfu Li, Mingda Li, and Jin Wang. 2019. Monotonic Properties of Completed Aggregates in Recursive Queries. CoRR abs/1910.08888 (2019).

[13] Carlo Zaniolo, Ariyam Das, Jiaqi Gu, Youfu Li, Mingda Li, and Jin Wang. 2021. Developing Big-Data Application as Queries: an Aggregate-Based approach. IEEE Data Eng. Bull. 44, 2 (2021), 3–13.

[14] Carlo Zaniolo, Mohan Yang, Ariyam Das, Alexander Shkapsky, Tyson Condie, and Matteo Interlandi. 2017. Fixpoint semantics and optimization of recursive Datalog programs with aggregates. TPLP 17, 5-6 (2017), 1048–1065.