Dask Cassandra Loader Documentation

Release 0.0.0.dev0

Romulo Goncalves

Apr 28, 2020

Contents:

A data loader which loads data from a Cassandra table into a Dask dataframe. It allows partition elimination, selection and projections pushdown.

CHAPTER 1

Tutorial

Welcome to the Dask cassandra loader tutorial. This tutorial demonstrates the basics of using Dask cassandra loader using either a local or a remote Cassandra database.

To install Dask cassandra loader, use

```
pip install dask-cassandra-loader
```

If you're using dask cassandra loader in a program, you will probably want to use a virtualenv and install Cerulean into that, together with your other dependencies.

1.1 Setup

The tutorial requires the creation of a keyspace in an existent Cassandra cluster. For this tutorial it is used the keyspace called **tutorial**. In this example it is assume the local client cqlsh is installed and configured accordingly.

```
cqlsh -e "create keyspace tutorial with replication = {'class': 'SimpleStrategy',
↪'replication_factor': 1};"
```

Once the keyspace is created the user needs to create a table and load it. To do that the user needs to run the `tutorial.cql` file as follow:

```
cqlsh --keyspace=tutorial -f tutorial.cql
```

Once the table is loaded, the user will have a table called **tab1** with the following schema:

```
create table tab1(id int, year int, month int, day int, timest timestamp, lat float,␣
↪lon float, PRIMARY KEY((id, year, month)));
```

The loaded data has two partitions due two distinct months.

1.2 Dask cassandra loader

The first step to load a table from Cassandra into a Dask data-frame is to create *dask_cassandra_loader.loader.Loader*. To do that the user should do the following:

```
from dask_cassandra_loader import Loader

dask_cassandra_loader = Loader()
```

1.2.1 Connect to Cassandra

With the loader the user is then able to set a connection to an existent Cassandra cluster. In this example we assume the user is connecting to local cluster using the default credentials.

```
keyspace = 'tutorial'
cluster = ['127.0.0.1']

dask_cassandra_loader.connect_to_cassandra(cluster,
                                           keyspace,
                                           username='cassandra',
                                           password='cassandra')
```

1.2.2 Connect to Dask

Before a table is loaded it is necessary to connect to a Dask Cluster. For testing proposes it might be handy to have the option to create a **LocalCluster**. Both options are supported as the following examples will show.

To create and connect to a local Dask cluster you use the following code:

```
dask_cassandra_loader.connect_to_local_dask()
```

To connect to a remote cluster you use the following code:

```
cluster = "host1.domain.nl:9091"
dask_cassandra_loader.connect_to_dask(cluster)
```

1.2.3 Read Table

In this example the user will load table *tab1*, project columns *id*, *year*, *month*, *day*, have a predicate on column *day* (*day = 18*) and only select the partitions for which *id in [18]*, *year in [2018]* and *month in [11]*. In this example, it is requested to not load all partitions in case the query qualifies all of them for loading. For more details about the function, the user should read *dask_cassandra_loader.loader.Loader.load_cassandra_table()*.

```
table = dask_cassandra_loader.load_cassandra_table('tab1',
                                           ['id', 'year', 'month', 'day'],
                                           [('day', 'equal', [8])],
                                           [('id', [18]), ('year', [2018]),
                                            ('month', [11])],
                                           force=False)
if table is None:
    raise AssertionError("Table is not supposed to be None!!!")
```

(continues on next page)

(continued from previous page)

```
if table.data is None:
    raise AssertionError("Table.data is not supposed to be None!!!")

# Compute the Dask DataFrame and collect it as a Pandas DataFrame
local_table = table.data.compute()

# Inspect table information
print(local_table.head())
```

1.3 More information

To find all the details of what dask cassandra loader can do and how to do it, please refer to the *API documentation*.

CHAPTER 2

API Reference

2.1 dask_cassandra_loader.loader module

class dask_cassandra_loader.loader.**Connector**(*cassandra_clusters*, *cassandra_keyspace*, *username*, *password*)

Bases: object

It sets and manages a connection to a Cassandra Cluster.

shutdown()

Shutdowns the existing connection with a Cassandra cluster.

> shutdown()

exception dask_cassandra_loader.loader.**DaskCassandraLoaderException**

Bases: Exception

Raise when the DaskCassandraLoader fails.

class dask_cassandra_loader.loader.**Loader**

Bases: object

A loader to populate a Dask Dataframe with data from a Cassandra table.

connect_to_cassandra(*cassandra_clusters*, *cassandra_keyspace*, *username*, *password*)

Connects to a Cassandra cluster specified by a list of IPs.

> connect_to_cassandra('test', ['10.0.1.1', '10.0.1.2'])

Parameters

- **cassandra_keyspace** – It is a string which contains an existent Cassandra keyspace.
- **cassandra_clusters** – It is a list of IPs with each IP represented as a string.
- **username** – It is a string.
- **password** – It is a string.

connect_to_dask(*dask_cluster*)
Connect to a Dask Cluster

> connect_to_Dask('127.0.0.1:8786') or > connect_to_Dask(cluser)

Parameters **dask_cluster** – String with format url:port or an instance of Cluster

connect_to_local_dask()
Connects to a local Dask cluster.

> connect_to_local_dask()

disconnect_from_cassandra()
Ends the established Cassandra connection.

> disconnect_from_cassandra()

disconnect_from_dask()
Ends the established Dask connection.

> disconnect_from_dask()

load_cassandra_table(*table_name*, *projections*, *and_predicates*, *partitions_to_load*, *force=False*)
It loads a Cassandra table into a Dask dataframe.

> load_cassandra_table('tab1', ['id', 'year', 'month', 'day'], [('month', 'less_than', [1]), ('day', 'in_', [1,2,3,8,12,30])], [('id', [1, 2, 3, 4, 5, 6]), ('year',[2019])])

Parameters

- **table_name** – It is a String.
- **projections** – A list of columns names. Each column name is a String.
- **and_predicates** – List of triples. Each triple contains column name as String, operator name as String, and a list of values depending on the operator. CassandraOperators.print_operators() prints all available operators. It should only contain columns which are not partition columns.
- **partitions_to_load** – List of tuples. Each tuple as a column name as String. and a list of keys which should be selected. It should only contain columns which are partition columns.
- **force** – It is a boolean. In case all the partitions need to be loaded, which is not recommended, it should be set to 'True'. By Default it is set to 'False'.

class dask_cassandra_loader.loader.**LoadingQuery**
Bases: object

Class to define a SQL select statement over a Cassandra table.

build_query(*table*)
It builds and compiles the query which will be used to load data from a Cassandra table into a Dask Dataframe.

> build_query(table)

Parameters **table** – Instance of CassandraTable.

drop_projections()
It drops the list of columns to be projected, i.e., selected.

> drop_projections()

static partition_elimination(*table*, *partitions_to_load*, *force*)
It does partition elimination when by selecting only a range of partition key values.

> partition_elimination(table, [(id, [1, 2, 3, 4, 5, 6]), ('year',[2019])])

Parameters

- **table** – Instance of a CassandraTable
- **partitions_to_eliminate** – List of tuples. Each tuple as a column name as String and a list of keys which should be selected. It should only contain columns which are partition columns.
- **force** – It is a boolean. In case all the partitions need to be loaded, which is not recommended, it should be set to 'True'.

print_query()
It prints the query which will be used to load data from a Cassandra table into a Dask Dataframe.

> print_query()

remove_and_predicates()
It drops the list of predicates with 'and' clause over the non partition columns of a Cassandra's table.

> remove_and_predicates()

set_and_predicates(*table*, *predicates*)
It sets a list of predicates with 'and' clause over the non partition columns of a Cassandra's table.

> set_and_predicates(table, [('month', 'less_than', 1), ('day', 'in_', [1,2,3,8,12,30])])

Parameters

- **table** – Instance of class CassandraTable.
- **predicates** – List of triples. Each triple contains column name as String, operator name as String, and a list of values depending on the operator. CassandraOperators.print_operators() prints all available operators. It should only contain columns which are not partition columns.

set_projections(*table*, *projections*)
It set the list of columns to be projected, i.e., selected.

> set_projections(table, ['id', 'year', 'month', 'day'])

Parameters

- **table** – Instance of class CassandraTable
- **projections** – A list of columns names. Each column name is a String.

class dask_cassandra_loader.loader.**Operators**
Bases: object

Operators for a valida SQL select statement over a Cassandra Table.

static create_predicate(*table*, *col_name*, *op_name*, *values*)

It creates a single predicate over a table's column using an operator. Call CassandraOperators.print_operators() to print all available operators.

> create_predicate(table, 'month', 'les_than', 1)

Parameters

- **table** – Instance of CassandraTable.

- **col_name** – Table's column name as string.
- **op_name** – Operators name as string.
- **values** – List of values. The number of values depends on the operator.

print_operators()
Print all the operators that can be used in a SQL select statement over a Cassandra's table.

> print_operators()

class dask_cassandra_loader.loader.**PagedResultHandler**(*future*)
Bases: object

An handler for paged loading of a Cassandra's query result.

handle_error(*exc*)
It handles and exception. > handle_error(exc) :param exc: It is a Python Exception. :return:

handle_page(*rows*)
It pages the result of a Cassandra query. > handle_page(rows) :param rows: Cassandra's query result. :return:

class dask_cassandra_loader.loader.**Table**(*keyspace*, *name*)
Bases: object

It stores and manages metadata and data from a Cassandra table loaded into a Dask DataFrame.

load_data(*cassandra_connection*, *ca_loading_query*)
It defines a set of SQL queries to load partitions of a Cassandra table in parallel into a Dask DataFrame.

> load_data(cassandra_con, ca_loading_query)

Parameters

- **cassandra_connection** – Instance of CassandraConnector.
- **ca_loading_query** – Instance of CassandraLoadingQuery.

load_metadata(*cassandra_connection*)
It loads metadata from a Cassandra Table. It loads the columns names, partition columns, and partition columns keys.

> load_metadata(cassandra_con)

Parameters **cassandra_connection** – It is an instance from a CassandraConnector

print_metadata()
It prints the metadata of a CassandraTable.

> print_metadata()

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index

T