Barista: A Distributed, Synchronously Replicated, Fault Tolerant, Relational Data Store

Anant Bhardwaj
anantb@csail.mit.edu

Rebecca Yale Taft
rytaft@mit.edu

Manasi Vartak
mvartak@mit.edu

MIT Computer Science & Artificial Intelligence Laboratory
6.824: Distributed Systems (Spring 2014)

Abstract

Barista is a distributed, synchronously replicated, fault tolerant, relational data store. It is a layer written over postgres that manages realtime replication of data in a distributed infrastructure to provide fault-tolerance and load balancing. All writes are propagated synchronously using paxos. Barista exposes SQL for data management. Client applications can use the same SQL with Barista and under the hood it takes care of load balancing, consistency, and fault-tolerance seamlessly.

1 Introduction

2 Client APIs

Barista APIs are exposed as Thrift IDL. Thrift is a framework, for scalable cross-language services development, combines a software stack with a code generation engine to build RPC services that work efficiently and seamlessly between C++, Java, Go, Python, Ruby, JavaScript, and various other languages. Although we are implementing Barista in Go on the server side, we expect the client call to be in any language.

2.1 A Sample Python Client

Barista RPC stubs for Python can be generated as the following:

```
thrift --gen py barista.thrift
```

Once RPC stubs are generated, a python code can call Barista APIs. Below is a sample client code:

```
transport = TSocket('any_replica_host', 'port')
transport = TTransport.TBufferedTransport(transport)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = Barista.Client(protocol)

transport.open()

con_params = ConnectionParams(user='postgres', password='postgres')
con = client.connect(con_params=con_params)

res  = client.execute_sql(
  con=con,
  query='select * from db',
  query_params=None)

for tuple in res.tuples:
  for cell in tuple.cells:
    print cell

transport.close()
```

2.2 Barista Client APIs

Below is the list of data types and methods available to client applications:

```
/* Database Connection */
// connection parameters
struct ConnectionParams {
  1: optional string user,
  2: optional string password,
  3: optional string database
}

// connection info -- must be passed in every execute_sql call
struct Connection {
  1: optional string client_id,
  2: optional string seq_id,
  3: optional string user,
  4: optional string database
}


/* ResultSet */
// A cell in a table
struct Cell {
  1: optional binary value
}

// A tuple
struct Tuple {
  1: optional list <Cell> cells
}

// A result set (list of tuples)
struct ResultSet {
  1: required bool status,
  2: Connection con,
  3: optional i32 row_count,
  4: optional list <Tuple> tuples,
  5: optional list <string> field_names,
  6: optional list <string> field_types
}


/* Barista Exceptions */
// Database Exception
exception DBException {
  1: optional i32 errorCode,
  2: optional string message,
  3: optional string details
}


/* Barista RPC APIs */
service Barista {
  double get_version()

  Connection connect (
     1: ConnectionParams con_params) throws (1: DBException ex)

  ResultSet execute_sql (
     1: Connection con,
     2: string query,
     3: list <binary> query_params) throws (1: DBException ex)
}
```