Why and how we wrote a Python driver for Scylla

A deep dive and comparison of Python drivers for Cassandra and Scylla

Why and how we wrote a Python driver for Scylla

EuroPython 2020
Bonjour !

Alexys Jacob

**Gentoo Linux developer**
- dev-db / mongodb / redis / scylla
- sys-cluster / keepalived / ipvsadm / consul
- dev-python / pymongo
- cluster + containers team member

**Open Source contributor**
- MongoDB
- Scylla
- Apache Airflow
- Python Software Foundation contributing member
EuroPython uses Discord… Discord uses Scylla!

Check out the talk of Mark Smith, Director of Engineering at Discord
Leveraging **Consistent Hashing** in **Python** applications

Check out my talk from EuroPython 2017 to get deeper into consistent hashing
Deep dive
Cassandra & Scylla token ring architectures
A cluster is a collection of nodes

Cassandra ring

= 

Scylla ring
Each **node** is responsible for a **partition** on the **token ring**.
Replication Factor provides higher data availability

Replication Factor = 2
Virtual Nodes = better partition distribution between nodes

Replication Factor = 2
Scylla’s **Virtual Nodes** are split into **shards** bound to cores!
Rows are located on nodes by hashing their partition key.
Take away: shard-per-node vs shard-per-core architecture

**Cassandra**
hash(Partition Key) token leads to RF*nodes

**Scylla**
hash(Partition Key) token leads to RF*nodes cores
Client drivers should leverage the token ring architecture!
Naive clients route queries to any node (coordinator)

The coordinator may not be a replica for the queried data!

SELECT * FROM motorbikes WHERE code = 'R1250GS'
Deep dive
Python cassandra-driver TokenAwarePolicy
Token Aware clients route queries to the right node(s)!

SELECT * FROM motorbikes WHERE code = 'R1250GS'
murmur3hash('R1250GS') → node X + node Y

RF=2
TokenAwarePolicy: **Statement + routing key = node(s)**
TokenAwarePolicy: Statement + routing key = node(s)

```
import logging
from cassandra.cluster import Cluster
from cassandra.policies import DCAwareRoundRobinPolicy, TokenAwarePolicy
from cassandra.query import dict_factory

logging.basicConfig(level=logging.DEBUG)

cluster = Cluster(
    contact_points=['nodeX', 'nodeY', 'nodeZ'],
    compression=True,
    load_balancing_policy=TokenAwarePolicy(DCAwareRoundRobinPolicy()),  # default
)

session = cluster.connect()
session.row_factory = dict_factory
session.set_keyspace('test')

statement = session.prepare("SELECT * FROM motorbikes WHERE code = ?")

for row in session.execute(statement, ('R125GS',)):
    print(row)

cluster.shutdown()
```
Default TokenAwarePolicy (DCAwareRoundRobinPolicy)

SELECT * FROM motorbikes WHERE code = 'R1250GS'

murmur3hash('R1250GS') = partition 1 = node X + node Y

load balanced (round-robin)

DC local nodes
Can’t beat my Cassandra’s TokenAwarePolicy (DCAwareRoundRobinPolicy)!
Yes you can. Use Scylla and a shard-per-core aware driver!
Shard Aware clients route queries to the right node(s) + core!

Coordinator + Data replica

SELECT * FROM motorbikes WHERE code = 'R1250GS'
murmur3hash('R1250GS') → node X + node Y

RF=2

Data replica

Shard Aware Client

Shard Aware clients route queries to the right node(s) + core!
Scylla shard aware drivers: Python was missing!

Forks of DataStax drivers to retain maximal compatibility and foster fast iteration

- **Java**
  - First one officially released in 2019

- **Go (gocql, gocqlx)**
  - Used in scylla-manager and other Go based tooling

- **C++**
  - WIP
Let’s make a **Python shard-aware driver!**
cassandra-driver / scylla-driver structural differences

- 1 control connection (cluster metadata, topology)
- 1 connection per node
- Token calculation selects the right connection to node to route queries

- 1 control connection (cluster metadata, topology)
- 1 connection per core per node
- Token calculation selects the right node
- Shard id calculation selects the connection to the right core to route queries
1 control connection (cluster metadata, topology)
  ○ Use as-is

1 connection per core per node
  ○ Connection needs to detect Scylla shard aware clusters (while retaining compatibility with Cassandra clusters)
  ○ HostConnection pool should open a Connection to every core of its host/node

Token calculation selects the right node
  ○ Use TokenAwarePolicy as-is

Shard id calculation selects the right connection to core to route queries
  ○ Cluster should pass down the query routing_key to the pool to allow connection selection
  ○ Implement shard id calculation based on the query routing_key token
  ○ HostConnection pool should select the connection to the right core to route the query

TODO: from cassandra-driver to scylla-driver
Inspired by Java driver’s shard aware implementation, Israel Fruchter paved the path and made the first PR for Python shard-awareness!

- **Connection** needs to detect Scylla shard aware clusters (while retaining compatibility with Cassandra clusters)

```python
class Connection(object):

    _check_hostname = False
    _product_type = None

    shard_id = 0
    sharding_info = None

    def _init_(self, host='127.0.0.1', port=9042, authenticator=None,
        ssl_options=None, socket=Socket(), compression=True,
        cql_version=None, protocol_version=ProtocolVersion.MAX_SUPPORTED, is_control_connection=False,

    def _send_options_message(self):
        def _handle_options_response(self, options_response):
            self.shard_id, self.sharding_info = ShardingInfo.parse_sharding_info(options_response)
```

```
scylla-driver **shard-awareness** detection

- **Connection** detects Scylla shard aware clusters thanks to response message options:

```python
+ class ShardingInfo(object):
+     
+     + def __init__(self, shard_id, shards_count, partitioner, sharding_algorithm, shardingIgnore_MSB):
+         self.shards_count = int(shards_count)
+         self.partitioner = partitioner
+         self.sharding_algorithm = sharding_algorithm
+         self.shardingIgnore_MSB = int(shardingIgnore_MSB)
+     
+     @staticmethod
+     + def parse_sharding_info(message):
+         shard_id = message.options.get('SCYLLA_SHARD', ['', '])[0] or None
+         shards_count = message.options.get('SCYLLA_NR_SHARDS', ['', '])[0] or None
+         partitioner = message.options.get('SCYLLA_PARTITIONER', ['', '])[0] or None
+         sharding_algorithm = message.options.get('SCYLLA_SHARDING_ALGORITHM', ['', '])[0] or None
+         shardingIgnore_MSB = message.options.get('SCYLLA_SHARDING_IGNORE_MSB', ['', '])[0] or None
```
scylla-driver connections to **shards/cores**

- **HostConnection** pool should open a **Connection** to **every core** of its host/node

  self._connections keys = shard id, values = connection obj

  first connection detects shard support on the node

  synchronous and optimistic way to get a connection to all cores... we try at max 2*number of cores on the node...

  ...and fail if not fully connected!
There is no way for a client to specify which shard/core it wants to connect to!

- Would require Scylla protocol to diverge from Cassandra’s
- This means that all other Scylla drivers are affected!
- Sent an RFC on the mailing-list to raise the problem

- Current status looking good
  - Client source port based shard attribution logic
  - Currently being implemented!

**TODO: connection to cores optimization**

- Fix startup time with asynchronous connection logic
- On startup try to connect to every shard only once
- A connection to all shard should **not** be mandatory
scylla-driver enhanced connections to shards/cores

- HostConnection pool should open a Connection to every core of its host/node

```python
from scylla.driver import HostConnection

# Example code snippet

def open_connection_to_missing_shard(shard_id):
    # Code for opening a connection to a missing shard

    # Code for creating a new connection and populating shard
    connection = self.session.cluster.connection_factory(self.host)
    connection.connect(shard_id, self.host)
    connection.set_keyspace(blocking=True)

    return connection

+ def open_connections_for_all_shards():
+     # Code for opening connections for all shards
+     for shard_id in range(self.sharding_info.shards_count):
+         connection = self.open_connection_to_missing_shard(shard_id)
+         connection = self.session.submit(self.open_connection_to_missing_shard, shard_id)
```

asynchronous!
Cluster should pass down the query `routing_key` to the pool to allow connection selection.

```python
def _query(self, host, message=None, cb=None):
    if message is None:
        message = self.message

    pool = self.session._pools.get(host)
    if not pool:
        self._errors[host] = ConnectionException("Host has been marked down or removed")
        return None
    elif pool.is_shutdown:
        self._ errors[host] = ConnectionException("Pool is shutdown")
        return None

    self._current_host = host
    connection = None
    try:
        connection, request_id = pool.borrow_connection(
            timeout=2.0,
            routing_key=self.query.routing_key if self.query else None
        )
```
Performance concern: move **shard id calculation** to Cython

- **cassandra.shard_info**: Cython shard id calculation used by **HostConnection** to route queries

```python
cdef class ShardingInfo:
    @staticmethod
    def parse_sharding_info(message):
        ...
        shard_id = message.options.get('SCYLIA_SHARD', ['', ''])[0] or None
        shards_count = message.options.get('SCYLIA_NR_SHARDS', ['', ''])[0] or None
        partitioner = message.options.get('SCYLIA_PARTITIONER', ['', ''])[0] or None
        sharding_algorithm = message.options.get('SCYLIA_SHARINH_ALGORITHM', ['', ''])[0] or None
        sharding_ignore_msb = message.options.get('SCYLIA_SHARDING_IGNORE_MSB', ['', ''])[0] or None

        if not (shard_id or shards_count or partitioner == "org.apache.cassandra.dht.Murmur3Partitioner" or
                sharding_algorithm == "biased-token-round-robin" or sharding_ignore_msb):
            return 0, None

        return int(shard_id), ShardingInfo(shard_id, shards_count, partitioner, sharding_algorithm, sharding_ignore_msb)

    def shard_id_from_token(self, uint64_t token_input):
        ...
        Find the right shard id (core) from the given routing_key's token
        This is how we route queries to the right core!

        cdef uint64_t biased_token = token_input + (uint64_t(1) << 63);
        biased_token <<= self.sharding_ignore_msb;
        cdef int shardId = (uint128_t(biased_token * self.shards_count) >> 64);
        return shardId
```

**Pure Python**
429.0309897623956 nsec per call

**Cython**
63.073349883779876 nsec per call

**Almost 7x faster!**
At the heart of scylla-driver’s **shard-awareness** logic

- **HostConnection** pool selects the **connection to the right core** to route the query

```python
+ shard_id = None
+ if self.host.sharding_info and routing_key:
+     t = self.session.cluster.metadata.token_map.token_class.from_key(routing_key)
+     shard_id = self.host.sharding_info.shard_id_from_token(t)
+ conn = self._connections.get(shard_id)
+ # missing shard aware connection to shard_id, let's schedule an
+ # optimistic try to connect to it
+ if shard_id is not None:
+     if conn:
+         log.debug("Using connection to shard_id\%d on host %s for routing_key\%s", shard_id, self.host, routing_key"
+     elif shard_id not in self._connecting:
+         # a rate controlled optimistic attempt to connect to a missing shard
+         self._connecting.add(shard_id)
+         self._session.submit(self._open_connection_to_missing_shard, shard_id)
+         log.debug("Trying to connect to missing shard_id\%d on host %s (%s/%s)\", shard_id, self.host, \"random.choice(self._connections.keys()\}), self.host.sharding_info.shARDS_COUNT"
+     # we couldn’t find a shard aware connection, let’s pick a random one
+     # from our pool
+     if not conn:
+         conn = self._connections.get(random.choice(self._connections.keys()))
```

- **Calculate shard id from query routing_key token**
- **Try to find a connection to the right shard id/core**
- **Use our direct connection to the right core** to route the query!

- **No connection to the right core yet, asynchronously try to get one**

- **There was no connection to the right core, pick a random one #legacy**
Python shard-aware driver
expectations & production results
scylla-driver **expectations** checks

- **1 connection per core per node**
  - Number of cores on node times more connections open to each cluster node
    - Production real-time processing rolling update effect:
      - More CPU requirements to handle/keepalive more connections
        - Production Kubernetes resources adjustment to avoid pod CPU saturation / throttling

- **Routing queries to the right core of the right node**
  - Reduced query latency...
15% to 25% performance boost!
Scylla-driver shard-aware latency impact: 15% to 25% performance boost!

This is a max() worst case scenario graph.

All shards are not connected yet.

More shards connected = Better latency.

Same analytics job peak.
**scylla-driver shard-awareness** is awesome!

- **movingMedian**(max(processing_time), “15min”)

- **Unexpected (and cool) side effect**
  - Reduced Scylla cluster load + reduced client latency = reduced resources on Kubernetes for the same workload!
Recent additions: shard-aware capability and connection statistics helpers

```python
from cassandra.cluster import Cluster

cluster = Cluster()
session = cluster.connect()

if cluster.is_shard_aware():
    print("connected to a Scylla cluster")

stats = cluster.shard_aware_stats()
if all([v["shards_count"] == v["connected"] for v in stats.values()]):
    print("Successfully connected to all shards of all Scylla nodes")
```

Use **shard capable ports** on Scylla when available

- [scylla/pull/6781](scylla/pull/6781)
- [scylladb/python-driver/pull/54](scylladb/python-driver/pull/54)

Improve Scylla specific **documentation**

Merge & rebase **latest cassandra-driver** improvements
$ pip install scylla-driver

Repository
https://github.com/scylladb/python-driver

PyPi
https://pypi.org/project/scylla-driver/

Documentation
https://scylladb.github.io/python-driver/master/index.html

Chat with us on ScyllaDB users Slack #pythonistas
https://slack.scylladb.com/
Thanks for attending and making this EuroPython a success!

Catch me online: @ultrabug

Discord talk channel
Late questions, deep-dive remarks? Let’s keep in touch :)

BRIAN BREAKOUTS
#talk-cassandra-scylla-drivers

Discord Numberly channel
Sponsor talk session tomorrow, Friday July 24th at 12:00 CEST
- Real-world experience sharing
- Open Source creations & contributions overview
- Conference talks experience, updates and feedbacks

SPONSOR EXHIBIT
#numberly