공학
트렌드

소프트웨어공학센터 경영지원TF팀

Cassandra(Nosql)를 이용한 유닛 테스트 방법
Part 1 : 카산드라 소개

2014. 7. 8. [제98호]

Contents

정보통신산업진흥원 부설
nipa 소프트웨어공학센터

Ⅰ. Cassandra 소개 및 특징

구글의 BigTable 아키텍트와 AWS의 Amazon Dynamo를 기반으로 하고 있는 Cassandra는 Amazon의 Dynamo를 디자인했던 Avinash Lakshman라는 엔지니어가 Facebook으로 이직해 Facebook의 Prashant Malik 엔지니어와 함께 만들었다.

그림 1_Cassandra 마크

Cassandra는 줄임말로 C* 이라 표현하기도 한다. C* 표시가 유명한 곳은 Cassandra를 컨설팅하는 회사인 Datastax이다. 이 사이트 에서는 Planet Cassandra 라는 커뮤니티 서비스 사이트를 운영하고 있다. Planet Cassandra는 〈그림 2〉과 같이 C*이 표시된 마크로 형상화했다.

그림 2_C* 이 표시된 Planet Cassandra 사이트

출처: http://planetcassandra.org/

Cassandra는 그리스 신화에서 나온 인물 중 하나로서, Cassandra는 프리아모스 왕과 헤카베의 딸이다. 이런 얘기를 하는 것은 Cassandra에 연관된 client library 명칭이 그리스 신화와 연관되어 있기 때문이다. 예를 들어 Cassandra의 대표적인 java client중 hector나 astyanax와 같이 그리스 신화의 인물의 이름을 사용하고 있다.

표 1_위키피디아의 Cassandra 항목 중에서

아폴론은 Cassandra를 사랑하게 되었는데, 그녀를 유혹하려고 예언 능력을 주었다. 그러나 Cassandra는 예언하는 능력을 가지게 되자, 아폴론은 신이기에 늙지 않고 죽지 않는다지만, 자신은 인간이었기에 늙고 죽는다는 것을 알았다. 즉 자신이 나중에 늙거나 병이 들면 아폴론은 자신을 버리고 다른 여자와 사귈 것이라는 알게 된 것이다. 그래서 아폴론이 자기를 끌어안자 그를 밀쳐냈고, 아폴론은 크게 진노하여 그녀의 입 안에 침을 뱉었다. 그 뒤로 Cassandra가 하는 예언을 더 이상 아무도 믿지 않게 되었다. Cassandra는 트로이 군에게 목마를 도시 안으로 들여보내지 말라고 경고했지만, 트로이 군은 그녀의 말을 무시하였고 때문에 그리스 군이 들어가 숨은 목마로 인해 전쟁에서 패했다.

출처: http://ko.wikipedia.org/wiki/%EC%B9%B4%EC%82%B0%EB%93%9C%EB%9D%BC

그리스 신화의 내용과는 상관없지만, Nosql Cassandra서버, Hector, Astyanax 와 같은 라이브러리들이 어우러지는 Cassandra 이야기를 시작하고자 한다.

1.1 아키텍처 특징

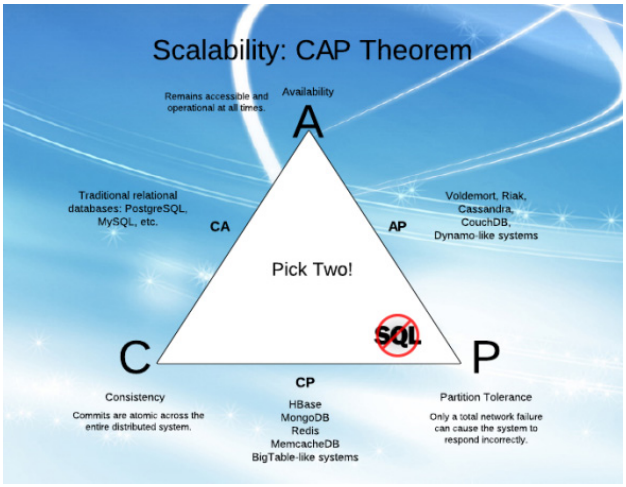
Cassandra는 BigTable Column 기반의 데이터 모델과 Dynamo 분산 모델을 기반으로 만들어졌다. 따라서 Dynamo 분산 모델의 큰 특징인 event consistency 와 BigTable Column 기반의 큰 특징인 Column 기반의 key-value 체계를 큰 틀로 하고 있다. 여기에 분산 처리(Distributed)가 되게 하여 SPOF(Single Point Of Failure)을 줄여주고 데이터 손실이 없도록 한다. 어떤 노드에 장애가 발생해도 전체 시스템은 멈추지 않도록 한다. 같은 데이터의 Replica가 동시에 장애가 발생하여 멈추는 문제가 아니라면 Read는 문제가 없다. Write는 Replica에 상관없이 항상 가능하다.

장비를 추가하고 제거하는 과정이 단순해서 새로운 장비를 추가하고 설정을 바꾼 후 Cassandra를 재 시작하면 된다. Cassandra는 노드가 추가되면 자동으로 Consistent hashing을 통해 각 노드가 가진 키의 개수를 맞춘다.

다른 NoSQL은 CAP이론 (Consistency, Availability, Partition Tolerance)에 대한 정책이 명확한 편이다. CAP 이론은 미국 버클리 대학의 Brewer 교수에 의해 알려졌다.[1)]

데이터 저장소는 CAP 중 하나는 만족시킬 수 없다는 내용이다.

그림 3_CAP 이론 정보



출처: http://architects.dzone.com/articles/better-explaining-cap-theorem

1) http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

〈그림 3〉을 살펴보면 Data Storage에 대해서 CAP 이론 쪽에 맞는 솔루션들을 분류한 것이다. 예를 들어 CA는 Mysql 이, AP는 CouchDB, CP는 HBase가 위치하고 있다.

Cassandra는 〈그림 3〉에 따르면 Consistency를 포기하고 AP에 집중된 것으로 보인다. 그러나 Cassandra는 Read replica count와 Write replica count의 값을 변경시켜 Consistency와 Availability 간의 균형을 개발자 또는 운영자가 설정할 수 있다. 만약 Consistency 가 주요한 부분이라는 정책을 세운다면, Read replica count와 Write replica count를 Replica 개수보다 크게 한다면 (R + W 〉 Number of replica) 강한 Consistency를 줄 수 있다.[2] 2008년 Apache 오픈 소스로 공개되어[3] 최근 소스는 Github에[4] 저장 및 공유되어 있다. 자바 언어로 개발되어 있는 특징이 있지만, 다양한 언어의 클라이언트로 개발할 수 있는 환경을 제공하고 있다.

현재 Twitter, Adobe, AOL, Rakuten, Splunk, Symantec, Hulu, Netflix, Ebay, Docomo, Expedia, Eventbrite, GE, Github, HP, IBM, Juniper, Microsoft, Onlive, Parse, Accenture, Sky방송, Instagram, Comcast 등 1500개 회사들이 Cassandra를 사용하고 있다.[5] 참고로 Facebook은 Cassandra 개발을 주도하였고, Inbox Search를 위해서 Cassandra 를 이용했으나, 2010년 HBase로 구현한 Facebook Message Platform로 바뀐 상태이다. 이후 2012년 Instagram에서 다시 쓰기 시작했다.[6]

대용량 데이터의 고성능 처리가 가능하여 용량 확장이 쉬운 데이터베이스이다. 처음에는 Relation DB(관계형 데이터베이스) 와 다르게 SQL을 사용하지 않은 NoSQL 제품 중 하나였으나 Cassandra 0.8 부터는 Cassandra Query Language (CQL) 를 제공하여 SQL과 비슷한 형태로 개발 할 수 있는 환경을 제공하고 있다. 2.0 이 추가되면서 lightweight transanction과 trigger가 추가되었고, datacenter 간 reparing을 제공하고 있다. Cassandra는 SSTable(Sorted String Table)에 Write성 데이터를 Append 하다가 일정 크기가 되면, SSTable을 통째로 디스크에 저장하도록 설계되었다. 따라서 데이터 파일 중간에 row를 끼워 넣는 저장 방식인 RDBMS보다는 성능이 좋다. SSTable의 Bloom filter를 통해 read 할 대상 데이터를 찾는 구조로 되어 있다. Bloom filter는 어떤 데이터가 어떤 곳에 저장되어 있는지 파악하는 알고리즘이다.

Cassandra는 read/write 시 한 쓰레드가 column 접근할 때 다른 쓰레드가 column 에 접

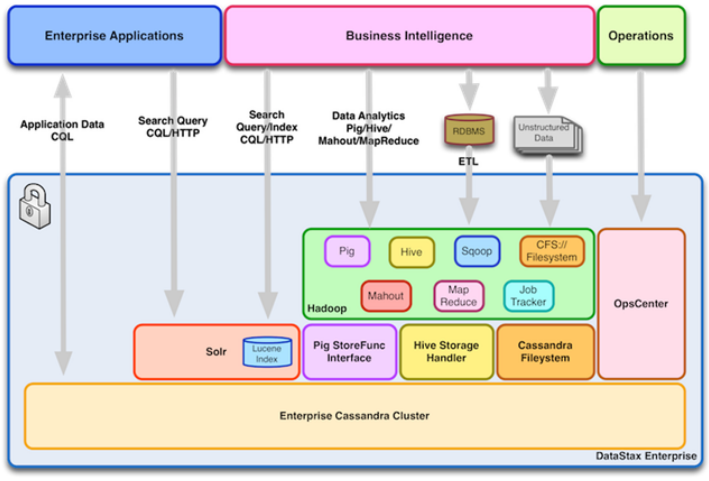
2) http://wiki.apache.org/cassandra/ArchitectureOverview
3) http://cassandra.apache.org/
4) https://github.com/apache/cassandra
5) http://planetcassandra.org/companies/
6) http://en.wikipedia.org/wiki/Apache_Cassandra

근하지 못하는 경우가 없도록 한다. 이를 lockless 라고 한다. 이 때문에 동시성 문제는 발생하지 않는다. 2.0에서 lightweight transaction을 제공하였다. 이는 compare and set 의 개념으로 존재 여부(if)를 확인하고 read/write를 할 수 있도록 하였다. Cassandra 2.0 부터는 java 7 이상 사용할 수 있는 환경이었는데, 최근에는 3.0이 개발이 진행되고 있다. 아직 정식으로 Release 하지 않았지만, 잠깐 내용을 살펴보면 Java 8에서 동작할 수 있는 구조이다.[7] 미국의 Netflix사는 아마존 AWS에서 Cassandra를 잘 사용하여 서비스 운영한 발표한 자료가 있으니, 자세한 내용은 아래 URL을 참조하도록 한다.[8]

OPS Center, DEV Center와 같은 운영 UI 툴이 제공되어 Cassandra 모니터링이나 데이터 조회가 가능하다. 서비스를 개발하고 운영하는 입장에서는 단순히 Cassandra 만 가지고 서비스 할 수 없다. 실시간 처리 및 분석을 지원하는 것은 물론 대용량을 지원하며 Cassandra와 integration 할 수 있는 오픈 소스들을 설치 운영의 조합이 필요하다. Hadoop / Pig나 Mahout과 같은 Hadoop 연동 시스템이나 Solr와 같은 검색 오픈 소스들과 Cassandra 모니터링을 하나의 군으로 패키징 할 수 있다. 이를 Consulting하는 회사는 DataStax 이다. DataStax는 DSE(DataStax Entreprise Edition)[9]라는 제품군을 참조하면 Cassandra를 기반으로 하는 솔루션군 배포 및 컨설팅을 지원하고 있다.

〈그림 4〉는 DataStax의 DSE는 Cassandra 기반으로 Hadoop, Hadoop 기반의 오픈 소스, Solr 검색, OpsCenter의 운영 도구가 어떻게 하나의 제품군으로 패키징 될 수 있는지 보여준다.

그림 4_DataStax의 DSE(DataStax Entreprise Edition)



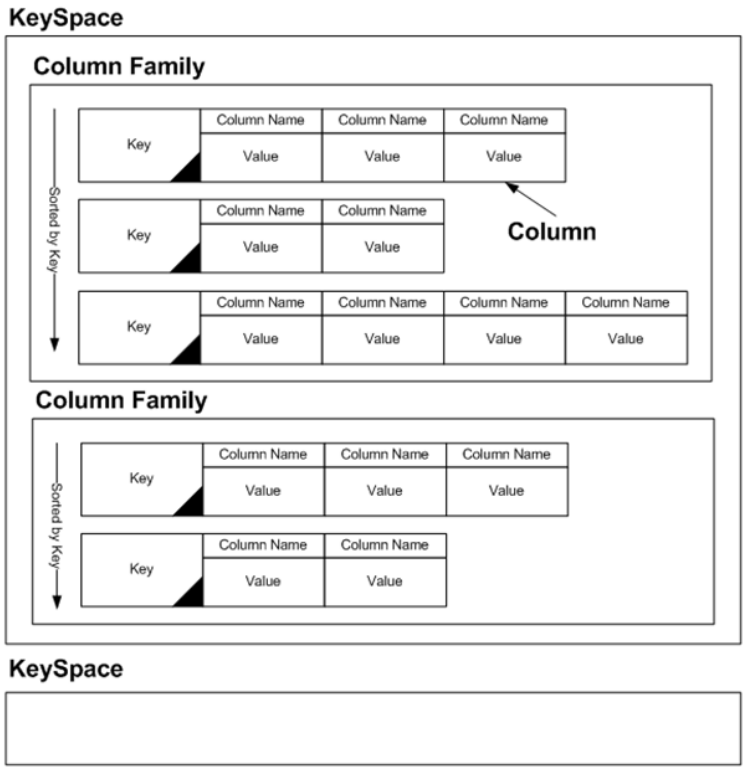
출처: http://blog.xebia.fr/2013/07/18/retours-sur-le-cassandra-breakfast/

7) https://github.com/apache/cassandra/blob/trunk/CHANGES.txt
8) http://www.slideshare.net/acunu/cassandra-eu-2012-netflixs-cassandra-architecture-and-open-source-efforts
9) http://www.datastax.com/what-we-offer/products-services/datastax-enterprise

1.2 데이터 저장 방식

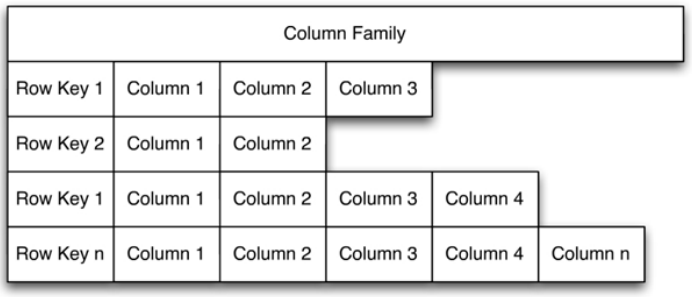
RDBMS는 row를 레코드 형태로 저장한다. 그러나 Cassandra는 〈그림 5〉처럼 Column Family 형태로 저장된다. Key-value 형태보다는 Key에 조금 더 많은 정보를 저장할 수 있다. 또한 Keyspace를 두어 multi-tenant 를 일부 지원하고 있다.

그림 5_Column Family 구조(1)



출처: http://javamaster.wordpress.com/2010/03/22/apache-cassandra-quick-tour/

그림 6_Column Family 구조(2)

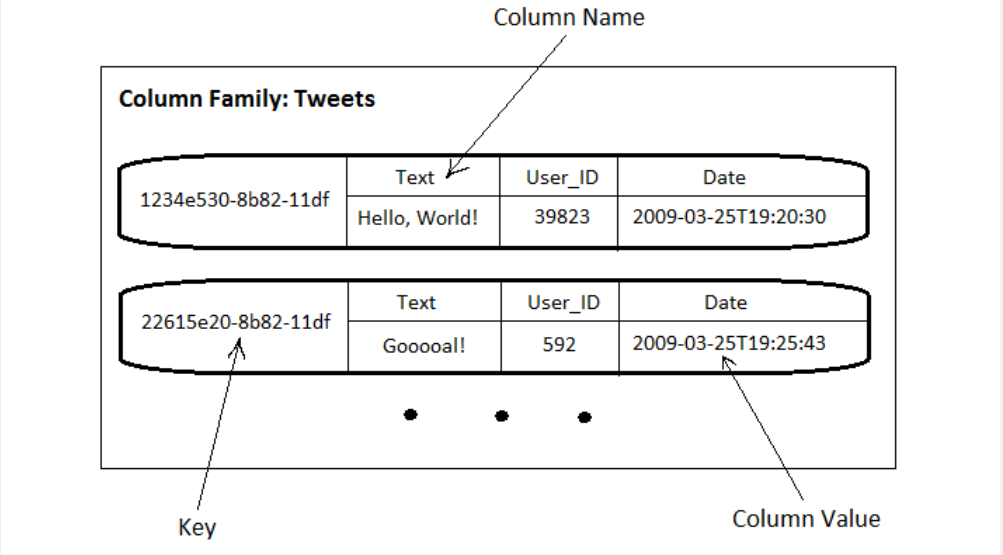


출처: http://javamaster.wordpress.com/2010/03/22/apache-cassandra-quick-tour/

각 Column은 name, value, timestamp를 가지고 있다. timestamp은 명시적으로 보여주고 있지 않다. 각각 따로 보려고 해야 볼 수 있는 시간 값이다. RDMS 관점으로 쉽게 표현하자면 KeySpace는 논리 Database와 비슷하고, Column Family는 Table 과 비슷하다. 그리고 Key와 Column 은 마치 Row의 Key와 Value와 비슷한 특징이 있다. RDBMS의 관점으로 <그림 6>을 보면 이해하는데 도움이 될 것이다.

Column Family를 사용한 예제를 통해 살펴보도록 한다. Tweeter의 예제를 들어보면, 사용자마다 TimeLine이 따로 존재한다. 이 TimeLine에 대한 글을 추상화하면 <그림 7>과 같을 것이다. Column Family Tweets는 Time UUID의 레코드 타입을 Key로 하고 User_ID(글 작성자), Text(글 내용), Date(사람이 알아볼 수 있는 Date 정보) 정보를 저장하도록 되어 있다.

그림 7_Tweets 예제



출처: http://maxgrinev.com/2010/07/09/a-quick-introduction-to-the-cassandra-data-model/

Cassandra 에는 Super Column 이라는 개념(Column 의 Map)이 초기에는 있었다. 그러나 성능 이슈가 생기면서 개수 제한이 주는 경우가 많아지면서, 최대한 쓰지 않도록 하고 있다.[10]

1.3 CQL3

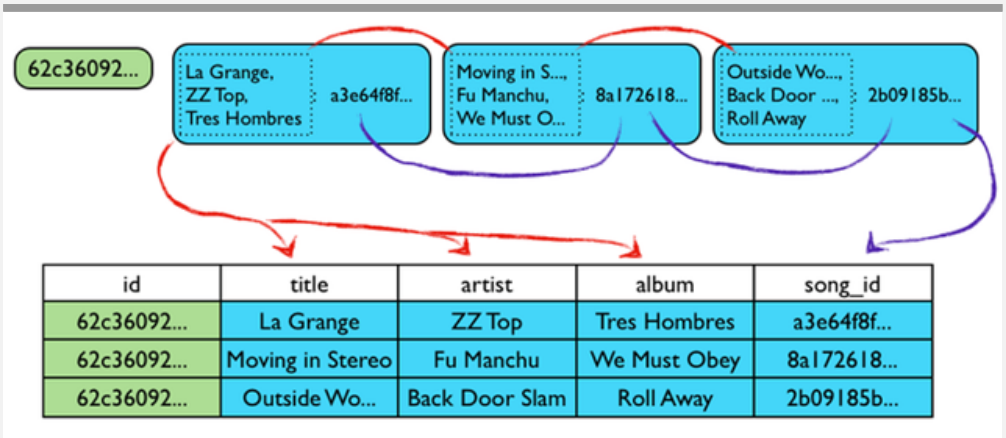
Cassandra는 Column Family라는 개념이 계속 쓰이고 있다. 그러나 CQL 이라는 Query Language 로 사용할 때는 상황이 다르다. CQL 초기 Specification에서는 Column Family를 사용했으나, 점차 Column Family 대신 Table로 사용을 확대했고, Column Family 라는 단

10) http://www.wentnet.com/blog/?p=38

어는 쓰이지 않고 있다. 대신 DB에서 사용하는 Table이라는 이름으로 변경되었다.

Column Family 데이터 모델링이 사실상 DB의 데이터 모델링과 비슷한 형태를 취하고 있다. 그러나 constraint가 없고 명시적인 transaction이 없는 유연한 DB의 테이블 형태와 비슷한 모델이 되고 있다. 1.2에서 얘기한 Column Family 구조를 RDBMS의 row와 같은 구조로 개념화 할 수 있다. 〈그림 8〉과 같이 바뀐 2차원 테이블 형태의 정보는 Query Language 질의가 가능하다. 간단하게 내용을 설명하면 다음과 같다.

그림 8_Column Family에서 RDBMS의 row로 개념화



출처: http://www.datastax.com/dev/blog/cql3-for-cassandra-experts

CQL3는 Cassandra에서 사용할 수 있는 Query Language로서 SQL과 비슷한 형태를 가지고 있다. playlists 라는 Table을 생성한다. Table은 사실상 Column Family 이다.

```
CREATE TABLE playlists (
id uuid,
title text,
album text,
artist text,
song_id uuid,
PRIMARY KEY (id, title, album, artist)
);
```

CQL3 에 대한 정보는 Cassandra 홈페이지[11]를 참조한다. 예를 들어 Select 문은 order by, limit이 제공된다. join와 sub query, group by, cursor는 제공이 안 된다. 간단한 종류의 쿼리를 사용할 수 있다.

11) http://cassandra.apache.org/doc/cql3/CQL.html

1.4 Cassandra 서버 설치 및 테스트

Cassandra 서버를 동작하기 위해서는 Cassandra 홈페이지[12]에서 Cassandra 압축파일을 다운로드 한다. Java 1.7을 설치한 후 다음의 작업을 진행한다.

```
$ tar -zxvf apache-cassandra-$VERSION.tar.gz
$ cd apache-cassandra-$VERSION
$ sudo mkdir -p /var/log/cassandra
$ sudo chown -R `whoami` /var/log/cassandra
$ sudo mkdir -p /var/lib/cassandra
$ sudo chown -R `whoami` /var/lib/cassandra
```

Cassandra 를 forground(-f)로 실행한다.

```
$ bin/cassandra -f
```

이제 Cassandra 설치 디렉토리에 접근해서 CQL을 이용해서 데이터 저장/조회를 해본다.

```
$ bin/cqlsh
cqlsh>
```

먼저 SCHEMA를 생성한 후, TABLE을 생성한다. INSERT 와 SELECT를 통해서 데이터 확인이 가능하다.

```
cqlsh> CREATE SCHEMA schema1
       WITH replication = { 'class' : 'SimpleStrategy', 'replication_factor' : 1 };
cqlsh> USE schema1;
cqlsh:schema1> CREATE TABLE users (
                    user_id varchar PRIMARY KEY,
                    first varchar,
                    last varchar,
                    age int
                  );
cqlsh:schema1> INSERT INTO users (user_id, first, last, age)
                  VALUES ('jsmith', 'John', 'Smith', 42);
cqlsh:schema1> SELECT * FROM users;
  user_id | age | first | last
 ---------+-----+-------+-------
   jsmith |  42 |  john | smith
```

그리고, TTL(Time To Live) 을 이용해서 특정 시간이 지나면 데이터를 자동으로 삭제 시킬 수 있다.

12) http://cassandra.apache.org/

```
cqlsh:schema1> INSERT INTO users (user_id, first, last, age) VALUES ('michael', 'Jack',
'Michael', 20) using ttl 10 ;

cqlsh:schema1> select * from users where user_id='michael';

user_id | age | first | last
---------+-----+-------+---------
michael |  20 |  Jack | Michael

(10 초 후)

cqlsh:schema1> select * from users where user_id='michael';

(0 rows)
```

그리고, DB와 같이 primary key가 아닌 필드에 인덱싱(indexing)을 할 수 있다.

```
cqlsh:schema1> create index on users(age);
cqlsh:schema1> create index on users(last);
cqlsh:schema1> create index on users(first);
```

II. Cassandra Java Client

Cassandra에 접근할 수 있는 언어별 클라이언트는 Planet Cassandra 사이트[13]에서 정리되었다. 지원하는 언어는 .net/c#, c++, closure, go, haskell, java, node.js, odbc, perl, php, python, R, ruby, scala 이다. 이 중에 잘 살펴봐야 하는 것은 Cassandra 1.0은 지원하지만 Cassandra 2.0은 지원하지 않은 부분들이 있으니 잘 확인하고 사용할 필요가 있다. Java 클라이언트의 경우는 Thrift API 을 이용하는 경우가 많다. 그러나 Thrift 가 쓰기에 쉽지 않기 때문에 Hector 가 가장 많이 사용되고 있으며, Netflix에서 개발한 Astyanax도 많이 쓰이고 있다.

2.1 Thrift API

Thrift 는 일종의 IDL(interface specification language)이기 때문에 다양한 언어에서 구현한

13) http://planetcassandra.org/client-drivers-tools/

stub코드를 이용하여 개발할 수 있다. Cassandra 뿐 아니라 Perl, Ruby, C, C++ 뿐 아니라 Java에서도 쉽게 사용할 수 있다. 아래는 Thift를 이용해서 'schema1' Keyspace의 'cf11'이라는 table(또는 column family)를 생성한 후, 한 건의 데이터를 insert 하는 java 코딩이다. 먼저 Cassandra 서버에 연결 후, table을 생성 후 insert 를 하는 코드이다.

```
public class ThriftTest {
 private static final String KEYSPACE_NAME = "schema1";
 private static final String ROW_KEY_NAME = "row1";
 private static final String COLUMN_FAMILY_NAME = "cf11";

 private Cassandra.Client client;
 private TTransport tr;

 public static void main(String[] args) throws Exception {
   ThriftTest sample = new ThriftTest();
   sample.getConfig();
   sample.create();
   sample.insert();
 }

 public void getConfig() {
   tr = new TFramedTransport(new TSocket("localhost", 9160));
   TProtocol proto = new TBinaryProtocol(tr);
   client = new Cassandra.Client(proto);
 }

 public void create() throws Exception {
   tr.open();
   String cql = "use " + KEYSPACE_NAME + ";";
   client.execute_cql_query(ByteBuffer.wrap(cql.getBytes()), Compression.NONE);
   cql = "create columnfamily " + COLUMN_FAMILY_NAME + "  (key text primary key,
name text, number text);";
   client.execute_cql_query(ByteBuffer.wrap(cql.getBytes()), Compression.NONE);
   tr.close();
 }

 public void insert() throws Exception {
   tr.open();
   client.set_keyspace(KEYSPACE_NAME);

   // insert data
   long timestamp = System.currentTimeMillis();
   Column nameColumn = new Column(ByteBuffer.wrap("name".getBytes()));
   nameColumn.setValue(Long.toHexString(1).getBytes());
   nameColumn.setTimestamp(timestamp);

   Column numberColumn = new Column(ByteBuffer.wrap("number".getBytes()));
   numberColumn.setValue(Long.toHexString(2).getBytes());
   numberColumn.setTimestamp(timestamp);

   ColumnParent columnParent = new ColumnParent(COLUMN_FAMILY_NAME);
```

```
    client.insert(ByteBuffer.wrap(ROW_KEY_NAME.getBytes()),
columnParent,nameColumn,ConsistencyLevel.ALL) ;
    client.insert(ByteBuffer.wrap(ROW_KEY_NAME.getBytes()),
columnParent,numberColumn,ConsistencyLevel.ALL);
    tr.close();
}
```

cqlsh를 이용하면 cf11 이라는 Table이 생성되었음을 확인할 수 있고, Insert 한 데이터도 확인 할 수 있다.

```
cqlsh:schema1> describe table cf11

CREATE TABLE cf11 (
  key text,
  name text,
  number text,
  PRIMARY KEY (key)
) WITH COMPACT STORAGE AND
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=864000 AND
  index_interval=128 AND
  read_repair_chance=0.100000 AND
  replicate_on_write='true' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='99.0PERCENTILE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

cqlsh:schema1> select * from cf11;

key   | name | number
------+------+--------
row1 |    1 |      2

(1 rows)
```

Thrift API 코드는 범용 프레임워크이기 때문에 Cassandra와 통신하는 코드가 깔끔하지 않다고 느껴질 수 있다. 또한 한 대의 노드가 아닌 클러스터 단위로 통신 작업을 해야하다 보니, 이에 대한 Cassnadra 노드의 통신 불능에 대한 처리 방식, Connection Pooling이 지원되고 있지 않다. 즉, DB와의 연결 및 DB 문제로 인한 Error/Exception 처리가 필요한 것이다. 따라서 Thrift API만 가지고 개발하고 있는 경우는 그리 많지 않으며, 위의 언급한 고급 기능을 포함시키고, Thrift API 기반으로 개발된 Hector나 Astyanax를 많이 사용하고 있다.

2.2 Hector

Hector는 Cassandra의 형제이다. Cassandra java client 중의 가장 유명한 것으로 알려진 Hector[14]를 이용하여 thrift API를 사용하는 것보다 쉽게 개발이 가능하다. 2.1의 예제와 비슷하게 Hector 코드를 가지고 개발한 코드이다. Cassandra 서버에 연결 후, table을 생성 후 insert 를 하는 코드이다. Table의 name_idx, number_idx 인덱스도 확인 가능하다.

```
public class HectorTest {
 private Cluster cluster = null;
 private Keyspace keySpace = null;
 private BasicColumnFamilyDefinition columnFamilyDefinition = null;
 private static final String KEY_SPACE_NAME = "schema1";
 private static final String COLUMN_FAMILY_NAME = "cf6";

 public static void main(String[] args) {
   HectorTest sample = new HectorTest();
   sample.getConfig();
   sample.create();
   sample.insert();
 }

 public void getConfig() {
   String hosts = "localhost:9160";

   CassandraHostConfigurator cassandraHostConfigurator = new
CassandraHostConfigurator(hosts);
   cassandraHostConfigurator.setMaxActive(1);
   cassandraHostConfigurator.setCassandraThriftSocketTimeout(3000);
   cassandraHostConfigurator.setMaxWaitTimeWhenExhausted(4000);
   cluster = HFactory.getOrCreateCluster("Test Cluster", cassandraHostConfigurator);

   ConfigurableConsistencyLevel configurableConsistencyLevel = new
ConfigurableConsistencyLevel();
   Map<String, HConsistencyLevel> clmap = new HashMap<String, HConsistencyLevel>();

   clmap.put("MyColumnFamily", HConsistencyLevel.ONE);
   configurableConsistencyLevel.setReadCfConsistencyLevels(clmap);
   configurableConsistencyLevel.setWriteCfConsistencyLevels(clmap);
   HFactory.createKeyspace(KEY_SPACE_NAME, cluster, configurableConsistencyLevel);

   keySpace = HFactory.createKeyspace(KEY_SPACE_NAME, cluster);
 }

 public void create() {
```

14) http://1and1.github.io/hector/build/html/index.html

```
    StringSerializer stringSerializer = StringSerializer.get();

    BasicColumnDefinition numberColumnDefinition = new BasicColumnDefinition();
    numberColumnDefinition.setName(stringSerializer.toByteBuffer("number"));
    numberColumnDefinition.setIndexName("number_idx");
    numberColumnDefinition.setIndexType(ColumnIndexType.KEYS);
    numberColumnDefinition.setValidationClass(ComparatorType.UTF8TYPE.getClassName());

    BasicColumnDefinition nameColumnDefinition = new BasicColumnDefinition();
    nameColumnDefinition.setName(stringSerializer.toByteBuffer("name"));
    nameColumnDefinition.setIndexName("name_idx");
    nameColumnDefinition.setIndexType(ColumnIndexType.KEYS);
    nameColumnDefinition.setValidationClass(ComparatorType.UTF8TYPE.getClassName());

    columnFamilyDefinition = new BasicColumnFamilyDefinition();
    columnFamilyDefinition.setKeyspaceName(KEY_SPACE_NAME);
    columnFamilyDefinition.setName(COLUMN_FAMILY_NAME);

columnFamilyDefinition.setKeyValidationClass(ComparatorType.UTF8TYPE.getClassName());
    columnFamilyDefinition.setComparatorType(ComparatorType.UTF8TYPE);

    columnFamilyDefinition.addColumnDefinition(numberColumnDefinition);
    columnFamilyDefinition.addColumnDefinition(nameColumnDefinition);

    cluster.addColumnFamily(columnFamilyDefinition);
    assert columnFamilyDefinition != null;
 }

 public void insert() {
    StringSerializer stringSerializer = StringSerializer.get();
    Mutator<String> mutator = HFactory.createMutator(keySpace, stringSerializer);

    mutator.insert("row1", columnFamilyDefinition.getName(),
HFactory.createStringColumn("1", "Kim"));
    mutator.insert("row1", columnFamilyDefinition.getName(),
HFactory.createStringColumn("2", "Park"));
    mutator.insert("row1", columnFamilyDefinition.getName(),
HFactory.createStringColumn("3", "Yun"));

    mutator.addInsertion("row2", columnFamilyDefinition.getName(),
HFactory.createStringColumn("1", "A"))
       .addInsertion("row2", columnFamilyDefinition.getName(),
HFactory.createStringColumn("2", "B"))
       .addInsertion("row2", columnFamilyDefinition.getName(),
HFactory.createStringColumn("3", "C"));
    mutator.execute();
 }

}
```

코드를 실행 후, Table이 실제로 생성되었는지 확인한다. Index 2개까지 생성된 것을

확인할 수 있다.

```
cqlsh:schema1> describe table cf6

CREATE TABLE cf6 (
  key text,
  name text,
  number text,
  PRIMARY KEY (key)
) WITH COMPACT STORAGE AND
  bloom_filter_fp_chance=0.010000 AND
  caching='KEYS_ONLY' AND
  comment='' AND
  dclocal_read_repair_chance=0.000000 AND
  gc_grace_seconds=0 AND
  index_interval=128 AND
  read_repair_chance=0.000000 AND
  replicate_on_write='false' AND
  populate_io_cache_on_flush='false' AND
  default_time_to_live=0 AND
  speculative_retry='NONE' AND
  memtable_flush_period_in_ms=0 AND
  compaction={'class': 'SizeTieredCompactionStrategy'} AND
  compression={'sstable_compression': 'LZ4Compressor'};

CREATE INDEX name_idx ON cf6 (name);

CREATE INDEX number_idx ON cf6 (number);
```

가장 많이 사용한다는 Hector 역시 Thrift보다는 덜 하지만 테이블 생성 코드나 데이터 추가 시 중복되는 객체가 많다.

2.3 Astyanax

그리스 신화의 Astyanax는 Hector의 동생이다. 2.2의 Hector 클라이언트와는 상관이 없지만 Netflix[15]에서 Cassandra를 적용하면서 Cassandra 클라이언트인 Astyanax[16]를 개발했고 오픈소스로 전향했다. 위 예제들과 같이 Table을 생성하고 한 건의 데이터를 저장하는 예제이다.

```
public class AstyanaxTest {

 private static final String CLUSTER_NAME = "Test Cluster";
 private static final String KEY_SPACE_NAME = "schema1";
```

15) https://www.netflix.com/global
16) https://github.com/Netflix/astyanax

```
 private static final String COLUMN_FAMILY_NAME = "cf21";

 private AstyanaxContext<Keyspace> context;
 private Keyspace keyspace;
 private ColumnFamily<String, String> COLUMN_FAMILY;

 public static void main(String[] args) {
   AstyanaxTest sample = new AstyanaxTest();
   sample.getConfig();
   sample.create();
   sample.insert();
   sample.list();
 }

 public void getConfig() {
   context = new AstyanaxContext.Builder()
        .forCluster(CLUSTER_NAME)
        .forKeyspace(KEY_SPACE_NAME)
        .withAstyanaxConfiguration(new AstyanaxConfigurationImpl()
        .setDiscoveryType(NodeDiscoveryType.RING_DESCRIBE))
        .withConnectionPoolConfiguration(
          new ConnectionPoolConfigurationImpl("MyConnectionPool")
        .setPort(9160).setMaxConnsPerHost(1)
        .setSeeds("127.0.0.1:9160"))
        .withAstyanaxConfiguration(new AstyanaxConfigurationImpl().setCqlVersion("3.0.0")
        .setTargetCassandraVersion("2.0.1"))
        .withConnectionPoolMonitor(new CountingConnectionPoolMonitor())
        .buildKeyspace(ThriftFamilyFactory.getInstance());

   context.start();
   keyspace = context.getClient();

   COLUMN_FAMILY = ColumnFamily.newColumnFamily(COLUMN_FAMILY_NAME,
        StringSerializer.get(), StringSerializer.get());
}

public void insert() {
 try {
   String statement = String.format("insert into " + COLUMN_FAMILY_NAME + " (key,
number, name) values (?, ?, ?) ₩n");
   keyspace.prepareQuery(COLUMN_FAMILY)
        .withCql(statement)
        .asPreparedStatement()
        .withStringValue("row1")
        .withStringValue("1")
        .withStringValue("Kim")
        .execute();

 } catch (ConnectionException e) {
     throw new RuntimeException("failed to write data to cql", e);
 }
```

```
}

public void create() {
 drop();
 String statement = "create table IF NOT EXISTS " + COLUMN_FAMILY_NAME
+ " (KEY text, number text, name text, PRIMARY KEY (KEY, number)) WITH COMPACT
STORAGE;";
 try {
    keyspace.prepareQuery(COLUMN_FAMILY)
       .withCql(statement)
       .execute();
    } catch (ConnectionException e) {
       throw new RuntimeException("failed to create table", e);
    }
 }
}
```

코드를 실행 후, Table 정보를 확인하고, 데이터가 저장되었는지 확인 가능하다.

```
cqlsh:schema1> describe table cf21

CREATE TABLE cf21 (
 key text,
 number text,
 name text,
 PRIMARY KEY (key, number)
) WITH COMPACT STORAGE AND
 bloom_filter_fp_chance=0.010000 AND
 caching='KEYS_ONLY' AND
 comment='' AND
 dclocal_read_repair_chance=0.000000 AND
 gc_grace_seconds=864000 AND
 index_interval=128 AND
 read_repair_chance=0.100000 AND
 replicate_on_write='true' AND
 populate_io_cache_on_flush='false' AND
 default_time_to_live=0 AND
 speculative_retry='99.0PERCENTILE' AND
 memtable_flush_period_in_ms=0 AND
 compaction={'class': 'SizeTieredCompactionStrategy'} AND
 compression={'sstable_compression': 'LZ4Compressor'};

cqlsh:schema1> select * from cf21;

key   | number | name
------+--------+------
row1 |       1 |  Kim

(1 rows)
```

저자는 Hector 보다는 Astyanax 가 훨씬 깔끔하게 사용할 수 있는 소지들이 많다. 그래서 저자는 저자가 개발한 서비스 모니터링 프로젝트의 데이터 저장소에 Astyanax 를 사용하

여 개발하였다. 잘 다듬어서 DB와 연결하는 코드처럼 쉽게 쓸 수 있도록 처리하였다.

Ⅲ. 정리

Nosql 중 하나인 Cassandra 에 대한 소개와 특징을 설명했고, 자바 클라이언트 중 대표적인 Thrift, Hector, Astyanax 예제를 통해 어떻게 사용하는지 설명했다. Part 2에서는 Cassandra를 어떻게 테스트할 수 있는지 설명하고 Unit Test 방법을 설명할 예정이다.

참고 자료

1. http://cassandra.apache.org/
2. http://maxgrinev.com/2010/07/09/a-quick-introduction-to-the-cassandra-data-model/
3. http://planetcassandra.org/
4. http://www.datastax.com/