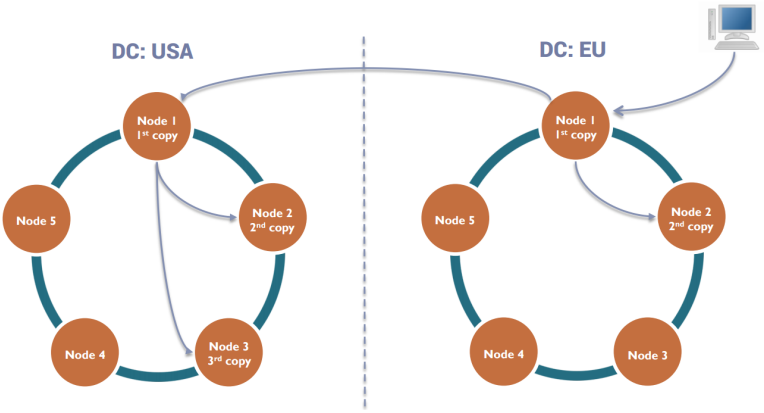
Cassandra Note

Chen Hongming <chenhm@gmail.com[1]>

1. 基本概念

Cassandra(社区也称为C*)在设计的时候就考虑了多数据中心多副本的情况，这使得它在跨数据中心的数据复制上相比其他数据库更具优势，一个典型的Cassandra集群含有多个数据中心，每个数据中心又由多个对等的节点组成。



1.1. Node

提供数据存储服务的基本组件。所有节点都是完全对等的，没有中心协调者，每个节点都知道数据在整个集群是如何分片的，都可以充当协调者。

1.2. Data center

由一组node组成的虚拟集合，可以设置数据在不同数据中心上的复制策略，同一数据中心内的node不应跨越物理上的数据中心。

1.3. Cluster

集群可以包含多个数据中心。

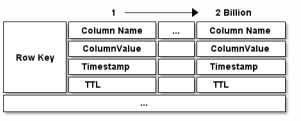
[1] mailto:chenhm@gmail.com

1.4. Keyspace

Keyspace是Table的外层容器，主要是为了定义一组表在集群内的复制策略。

1.5. Table

在早期 Thrift API[2] 时代Table叫Column Family，所以在一些文档会看到CF的简称，到 CQL API[3] 时代就改名叫Table了。C*的Table参考了 Google Bigtable[4] 的设计



C*使用Timestamp来解决记录冲突，而TTL过期的记录会自动从C*删除，通常我们不用管这两个属性，可以简单的将Table理解为Map，其中RowKey是主键。

```
Map<RowKey, SortedMap<ColumnKey, ColumnValue>>
```

如果是复合主键, C*的存储方式会有点特别，假设我们有下面这个表和数据

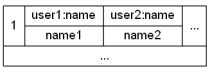
```
CREATE TABLE Note (
    key int,
    user text,
    name text,
    PRIMARY KEY (key, user)
);
insert into note(key,user,name) values(1,'user1','name1');
insert into note(key,user,name) values(1,'user2','name2');
```

```
 key | user  | name
-----+-------+-------
   1 | user1 | name1
   1 | user2 | name2
```

通过 `cassandra-cli` list命令查看会发现复合主键的第一个建仍然是RowKey，其余的部分则以column的形式被存储了。

[2] http://wiki.apache.org/cassandra/API10
[3] http://cassandra.apache.org/doc/cql3/CQL.html
[4] http://research.google.com/archive/bigtable.html

```
./cassandra-cli
list note;
-------------------
RowKey: 1
=> (name=user1:, value=, timestamp=1449473821587616)
=> (name=user1:name, value=6e616d6531, timestamp=1449473821587616)
=> (name=user2:, value=, timestamp=1449473822481729)
=> (name=user2:name, value=6e616d6532, timestamp=1449473822481729)
```



数据在C*内做数据分片的时候只能基于RowKey进行，所以复合键中的第一个键也被称为“Partition Key”。

显然我们查询的时候只能按主键定义的顺序查找记录：

```
SELECT * FROM note WHERE key = 1;
SELECT * FROM note WHERE key = 1 AND user = 'user1';
SELECT * FROM note WHERE user = 'user1'; --非法查询
```

同样在数据结构上也可以简单理解为

```
Map<RowKey, SortedMap<PrimaryKey2, SortedMap<ColumnKey, ColumnValue>>>
```

1.6. Commit log

所有数据会先写到commit log并持久化，之后会被刷新到SSTable，跟RDBMS的redo log是一样的。

1.7. Memtable

数据在写到SSTable之前先缓存在Memtable，达到一定数据量后再一次性写到SSTable，可以有效提高写性能。Memtable本质上就是`ConcurrentSkipListMap`.

1.8. SSTable

SSTable(sorted string table)上的数据是不可变的，通过追加数据来实现数据修改和删除，这也导致了C*的写性能优秀而读性能不是很好。每个Table会有一个或多个SSTable，每个

SSTable都包括三个子文件 bloomfilter文件，index文件和数据文件。bloomfilter可以高效标记某个key是否存在于这份sstable文件中； index文件记录key在对应数据文件中的位置。C*还会根据规则合并多个SSTable文件。

1.9. 一致性(CONSISTENCY)

分布式系统绕不开CAP定理，即CAP不可同时满足。

- Consistency(一致性), 数据一致更新，所有数据变动都是同步的
- Availability(可用性), 好的响应性能
- Partition tolerance(分区容错性), 允许节点之间丢失消息

显然P是必须的，否则一个节点故障就导致集群不可用，分布式系统意义就变小了，于是我们只能在AP和CP中选择。C*的一个优势就是可以通过设置ConsistencyLevel实现CP或AP的切换。

为了保证P，数据必须有replication，通常我们设置replication factors为3，即一份数据存3份。于是我们存取数据的策略有以下几种：

- 每次写都写3份(`ConsistencyLevel.ALL`)，无疑系统肯定是一致的，这个时候从任何一个节点读取都可以获得最新的数据(`ConsistencyLevel.ONE`)
- 每次只写1份(`ConsistencyLevel.ONE`)，系统出现了不一致，但只要读取了所有节点(`ConsistencyLevel.ALL`)，我们还是可以获得最新的数据。注意，要判断哪个节点上的数据是最新的，显然依赖时钟同步，所以C*各个节点必须配置ntp同步，但即使这样也无法精确同步时钟，所以C*在理论上无法保证完美的一致性。不过业务上很少会发生在非常小的时间内（同一机房一般小于1ms）多个客户端从不同节点更新了同一条记录的情况，所以从业务上来看一致性还是有保证的。关于记录选择参考 What happens if two updates are made with the same timestamp?[5]
  除了时钟同步，我们还可以通过每次都写同一个node的方法保证一致性，这样集群内记录的数据时间就以这个node为准了。对应`LoadBalancingPolicy`策略`new TokenAwarePolicy(new DCAwareRoundRobinPolicy(), false) //一个数据中心内使用同一个node`或`new TokenAwarePolicy(new RoundRobinPolicy(), false) //一个集群内使用同一个node`
- 上面第一种情况对于读的可用性很高，但对于写的可用性很低，第二种情况正好相反，如果我们每次写2份，读2份，那么既保证了一致性，同时读写也都有一定的可用性。这也是C*默认提供的一致性`ConsistencyLevel.QUORUM`， quorum定义为`quorum =`

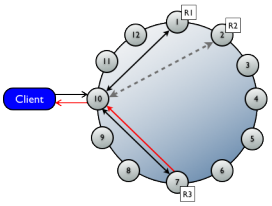
[5] http://wiki.apache.org/cassandra/FAQ#clocktie

`(sum_of_replication_factors / 2) + 1`。这也是我们常说的 W + R > N 即可保证一致性。

- 牺牲强一致性，读写都为1，获得最好的可用性，由C*通过算法在一段时间后实现最终一致性。

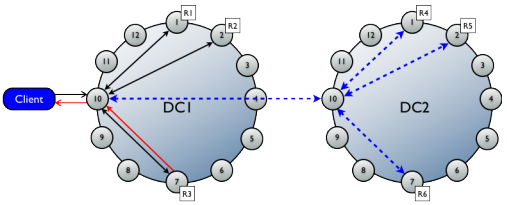
关于C*的隔离级请参考 http://docs.datastax.com/en/cassandra/2.0/cassandra/dml/dml_config_consistency_c.html

1.10. 读请求



上图显示了12个节点，复制因子为3，一致性级别为QUORUM的读情况，其中node 10是协调者。协调者会对比R1和R3返回的数据，将最新的数据返回给client。如果发现了数据不一致，后台进程还会尝试修复。可以通过表属性 `read_repair_chance` 设定读修复的比率，但Cassandra 2.0.9及以后的版本不需要读修复。

1.11. 写请求



上图显示了两个数据中心DC1和DC2，复制因子在两个数据中心都是3。当需要写数据的时候，协调者将写请求发到所有的复制节点，但跨数据中心的节点只发送一份，数据中心内部再做同步。写一致性级别为ONE，所以只需要有一个节点R3返回了请求，协调者就可以将数据返回给Client了。

查看数据如何在各个节点间同步，可以在CQL中启用trace或通过编码实现。参考 Request tracing[6] 和 Enabling tracing[7]

[6] http://www.datastax.com/dev/blog/tracing-in-cassandra-1-2
[7] https://docs.datastax.com/en/developer/java-driver/2.0/java-driver/tracing_t.html

2. 代码样例

- 以下代码实现了从一张表读取数据然后用多线程并行插入另一张表的过程，对C*的读写操作可以参考此代码。

```
import static com.datastax.driver.core.querybuilder.QueryBuilder.*;

final Cluster cluster = Cluster.builder()
        .addContactPoints("10.175.189.66", "10.175.189.67")
        .withLoadBalancingPolicy(new TokenAwarePolicy(new
 DCAwareRoundRobinPolicy(), false)) ❶
        .withQueryOptions(new QueryOptions()
            .setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM)) ❷
        .withRetryPolicy(DefaultRetryPolicy.INSTANCE)
        // .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE) ❸
        .withProtocolVersion(ProtocolVersion.NEWEST_SUPPORTED)
        .build();
final Session session = cluster.connect(); ❹

//通过CQL构建PreparedStatement
//final PreparedStatement ps = session
//        .prepare("INSERT INTO facade.service_profile(identity, serviceid,
 createtime, extensions, refencetid, status) values(?,?,?,?,?,?);")
//        .setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM);

TableMetadata metaData =
 cluster.getMetadata().getKeyspace("facade").getTable("service_profile");
final PreparedStatement ps = session.prepare(insertInto(metaData) ❺
                .value("identity", bindMarker())
                .value("serviceid", bindMarker())
                .value("createtime", bindMarker())
                .value("extensions", bindMarker())
                .value("refencetid", bindMarker())
                .value("status", bindMarker()))
        .setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM); ❻

Statement statement = select().all()
        .from("iam", "service_profile").limit(100000)
        .setConsistencyLevel(ConsistencyLevel.ONE); ❼
ResultSet results = session.execute(statement);
final long startTime = System.nanoTime();
System.err.println("start.");
final List<Row> res = results.all();
final long readTime = System.nanoTime();
System.err.println("Read Time:"+ (readTime - startTime)/(1000*1000) +"ms");
```

```
final AtomicInteger inProcess = new AtomicInteger();
final int poolSize = 4;
final int subSize = res.size() / poolSize;
ExecutorService executor = Executors.newFixedThreadPool(poolSize);

for (int i = 0; i < poolSize;i++) {
    final int num = i;
    executor.submit(new Runnable() {
        @Override
        public void run() {
            inProcess.incrementAndGet();
            List<Row> subList = res.subList(num * subSize, subSize * (num + 1));
            System.out.println(num + ":" + subList.size());

            for (Row row : subList) {
                session.executeAsync(ps.bind(row.getString(0), row.getString(1),
                                row.getLong(2), row.getObject(3),
                                row.getString(4), row.getString(5)));
            }

            if (inProcess.decrementAndGet() == 0) {
                session.close();  ❽
                long end = System.nanoTime();
                System.err.println("Write Time:" + (end - readTime)/(1000*1000)
 +"ms");
                cluster.close();
            }
        }
    });
}
executor.shutdown();
```

❶ TokenAwarePolicy会将同一个key的操作指向同一台机，避免集群时钟不同步的一致性问题，注意shuffleReplicas要设成false

❷ 设定默认的读一致性级别

❸ DowngradingConsistencyRetryPolicy允许默认的一致性级别失败后，用更低的一致性级别重试，比如在replication factors为3的环境设置了QUORUM，那么默认会尝试从2个节点读取数据，如果失败了，会再次用`ConsistencyLevel.ONE`尝试，在提高可用性的同时也可能导致出现数据不一致，请小心使用。

❹ Session管理了到Cluster多个接入点的网络连接，并且是线程安全的，一个应用有一个Session即可

❺ 设置TableMetadata以便TokenAwarePolicy生效

❻ 设定Statement上的写一致性级别

❼ 设定Statement的读一致性级别，会覆盖Cluster的设置

❽ 需要小心Session只能关闭一次

- 通过CQL使用Batch

```
String cql =  "BEGIN BATCH "
       cql += "INSERT INTO test.prepared (id, col_1) VALUES (?,?); ";
       cql += "INSERT INTO test.prepared (id, col_1) VALUES (?,?); ";
       cql += "APPLY BATCH; "

DatastaxConnection.getInstance();
PreparedStatement prepStatement = DatastaxConnection.getSession().prepare(cql);
prepStatement.setConsistencyLevel(ConsistencyLevel.ONE);

// this is where you need to be careful
// bind expects a comma separated list of values for all the params (?) above
// so for the above batch we need to supply 4 params:
BoundStatement query = prepStatement.bind(userId, "col1_val",
 userId_2, "col1_val_2");

DatastaxConnection.getSession().execute(query);
```

3. CQL

3.1. 使用系统时间

```
INSERT INTO TEST (ID, NAME, VALUE, LAST_MODIFIED_DATE) VALUES ('2', 'elephant',
  'SOME_VALUE', dateof(now()));
```

The `now` function takes no arguments and generates a new unique timeuuid (at the time where the statement using it is executed). The `dateOf` function takes a timeuuid argument and extracts the embedded timestamp. (Taken from the CQL documentation on timeuuid functions[8])

3.2. 轻量级事务

使用IF从句实现

```
INSERT INTO emp(empid,deptid,address,first_name,last_name)
 VALUES(102,14,'luoyang','Jane Doe','li') IF NOT EXISTS;
```

[8] http://cassandra.apache.org/doc/cql3/CQL.html#timeuuidFun

```
UPDATE emp SET address = 'luoyang' WHERE empid = 103 and deptid = 16 IF
 last_name='zhang';
```

3.3. 指定column的过期时间

```
INSERT INTO emp(empID, deptID, first_name, last_name) VALUES(105, 17, 'jane',
 'smith') USING TTL 60;
```

其中USING TTL 60指明该条数据60秒后过期，届时会被自动删除。另外指定了TTL的数据columns会在compaction和repair操作中被自动删除。指定TTL会有8字节额外开销。

3.4. 查询过期时间

```
SELECT TTL(last_name)from emp;
```

3.5. 更新过期时间

```
INSERT INTO emp (empID, deptID, first_name, last_name) VALUES (105, 17, 'miaomiao',
 'han') USING TTL 3600;
```

也即，以新的TTL重插一遍数据即可。（指定插入的整条数据的过期时间）

或者`UPDATA emp USING TTL 3600 SET last_name='han' where empid=105 and deptid=17;`（指定set指明的数据的过期时间）

3.6. 查询写入时间

```
SELECT WRITETIME(first_name) from emp;
```

可查的该数据何时被插入。

3.7. 添加columns

```
ALTER TABLE emp ADD address varchar;
```

3.8. 更改column数据类型

```
ALTER TABLE emp ALTER address TYPE text;
```

4. Tips

4.1. 关于row cache

Datastax About the row cache[9] 有一句很重要的话：

> Cassandra caches all rows in a partition when reading the partition. While storing the row cache *off-heap*, Cassandra has to deserialize a partition into heap to read from it.

Cache基于partition从off-heap向heap复制，这一过程几乎是不可控的，很容易导致heap溢出，所以也不推荐使用。

4.2. 批处理BoundStatement

BoundStatement默认类型是`Type.LOGGED`,此模式下可实现原子提交，这也是BoundStatement最主要的作用。

BatchStatement减少了网络交互，但也增加了日志在多节点复制的过程，性能可能提升也可能下降，应使用`executeAsync`获得性能优化。参考 http://wiki.apache.org/cassandra/FAQ#batch_bulkload

BatchStatement最多只允许65535(0xFFFF)条记录一次提交。

4.3. 数据估算

Cassandra没有索引，所以无法通过扫描索引获得count，只能全表扫描，性能较差，所以限制了返回的记录数量。 当记录数较小的时候可以用

```
select count(*) from cf;
```

当记录数很大的时候，会返回`OperationTimedOut:  errors={}`错误，这时可以通过nodetool获得一个估算值

```
nodetool cfstats [<keyspace.cfname>...]
```

Number of keys (estimate) 一行显示的就是估算值。

[9] http://docs.datastax.com/en/cassandra/2.0/cassandra/operations/ops_configuring_caches_c.html?scroll=concept_ds_n35_nnr_ck__about-the-row-cache

如果是多重主键，这里只是第一个键的数量，跟记录数无关

4.4. 删除所有数据

Cassandra只能按where条件删除记录，如果想删除所有记录需要用 `TRUNCATE`

```
TRUNCATE keyspace_name.table_name;
```

Or if you are already using the keyspace that contains your target table:

```
TRUNCATE table_name;
```

5. References:

- Things You Should Be Doing When Using Cassandra Drivers[10]
- Cassandra: Datastax Java driver retry policy[11]
- Composite Keys in Apache Cassandra[12]
- Cassandra SSTable, Memtable inside[13]
- The data model is dead, long live the data model[14]
- Real data models of silicon valley[15]
- Become a super modeler[16]
- Going native with Apache Cassandra[17]
- http://dongxicheng.org/nosql/cassandra-strategy/
- http://blog.csdn.net/zyz511919766/article/details/38683219

[10] https://ahappyknockoutmouse.wordpress.com/2014/11/12/246/
[11] http://christopher-batey.blogspot.jp/2013/10/cassandra-datastax-java-driver-retry.html
[12] http://www.planetcassandra.org/blog/composite-keys-in-apache-cassandra/
[13] http://codrspace.com/b441berith/cassandra-sstable-memtable-inside/
[14] http://www.slideshare.net/patrickmcfadin/the-data-model-is-dead-long-live-the-data-model
[15] http://www.slideshare.net/patrickmcfadin/real-data-models-of-silicon-valley
[16] http://www.slideshare.net/patrickmcfadin/become-a-super-modeler
[17] http://www.slideshare.net/johnny15676/going-native-with-apache-cassandra