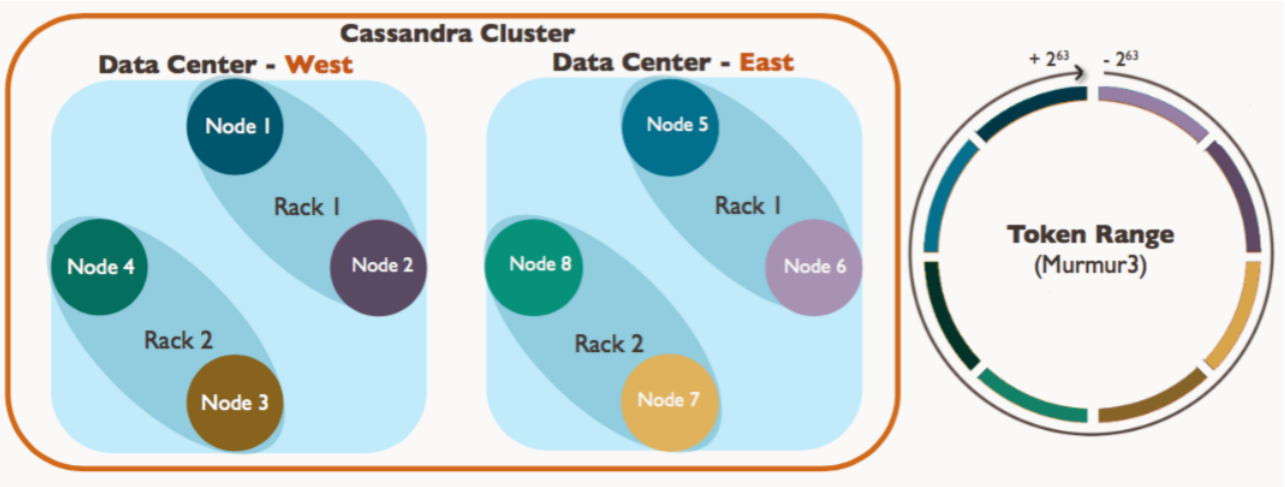
Cassandra内部架构

Cassandra是一个开源的、分布式、无中心节点、弹性可扩展、高可用、容错、一致性协调、面向列的NoSQL数据库

Cassandra集群（Cluster）

- Cluster
  - Data center(s)
    - Rack(s)
      - Server(s)
        - Node (more accurately, a vnode)

- Node（节点）：一个运行cassandra的实例
- Rack（机架）：一组nodes的集合
- DataCenter（数据中心）：一组racks的集合
- Cluster（集群）：映射到拥有一个完整令牌圆环所有节点的集合

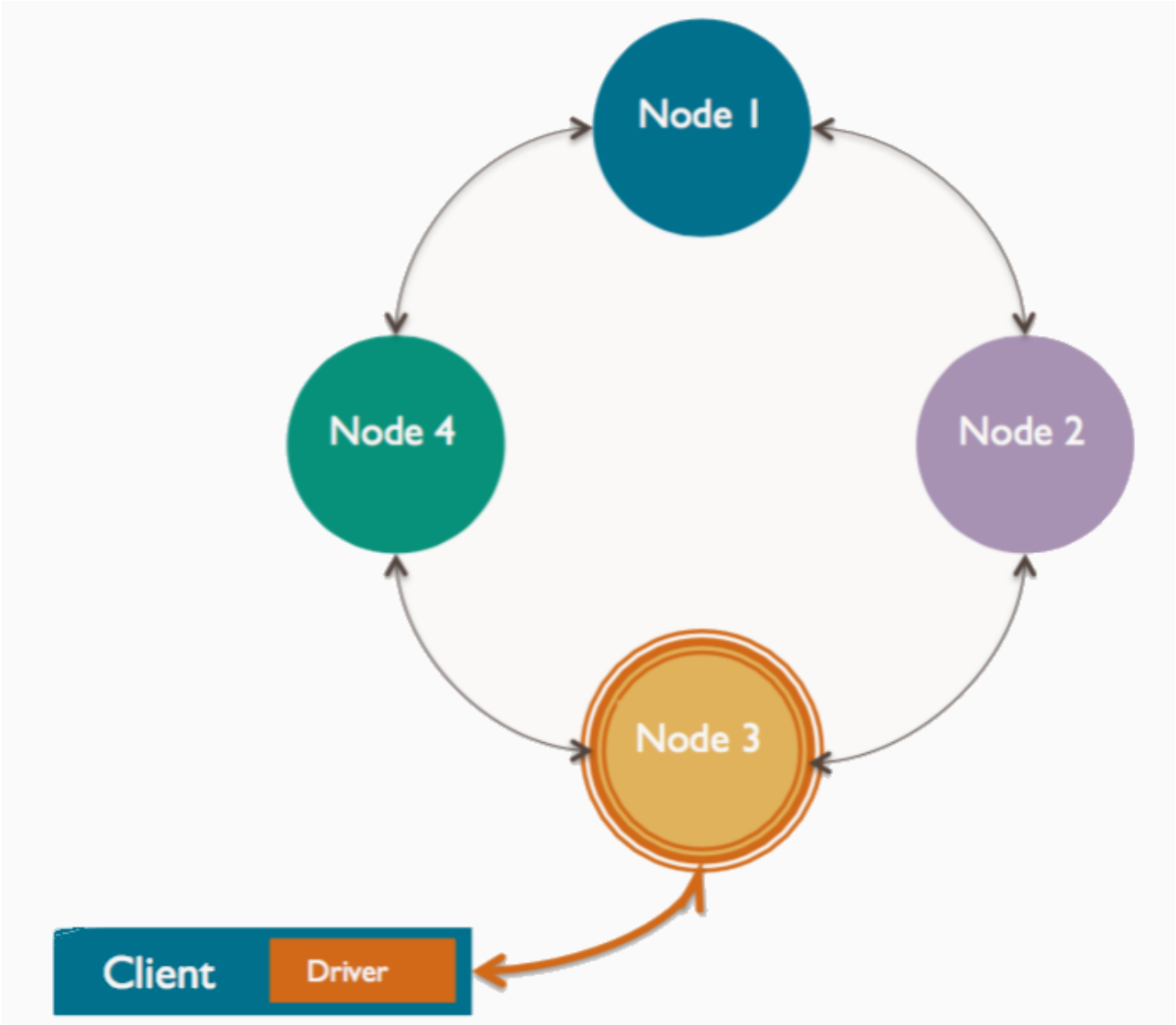


协调者（Coordinator）

客户端连接到某一节点发起读写请求时，该节点充当 客户端应用 与集群中 拥有相应数据节点 间的桥梁，称为协调者，以根据集群配置确定 环(ring)中的哪个节点 应当获取这个请求。

使用CQL连接指定的 `-h` 节点就是协调节点

- 集群中任何一个节点都可能成为协调者
- 每个客户端请求都可能由不同的节点来协调
- 由协调者管理复制因子（复制因子：一条新数据应该被复制到多少个节点）
- 协调者申请一致性级别（一致性级别：集群中有多少节点必须相应读写的请求）



分区器（Partitioner）

分区器决定了数据如何在集群内被分发。在Cassandra中，table的每行由唯一的primarykey标识，`partitioner` 实际上为 `一hash函数` 用，以 `计算primary key的token` 。Cassandra依据这个token值在集群中放置对应的行。

Cassandra提供了三种不同的分区器

- Murmur3Partitioner（默认）- 基于MurmurHash hash值将数据均匀的分布在集群
- RandomPartitioner - 基于MD5 hash值将数据均匀的分布在集群中
- ByteOrderedPartitioner - 通过键的字节来保持数据词汇的有序分布

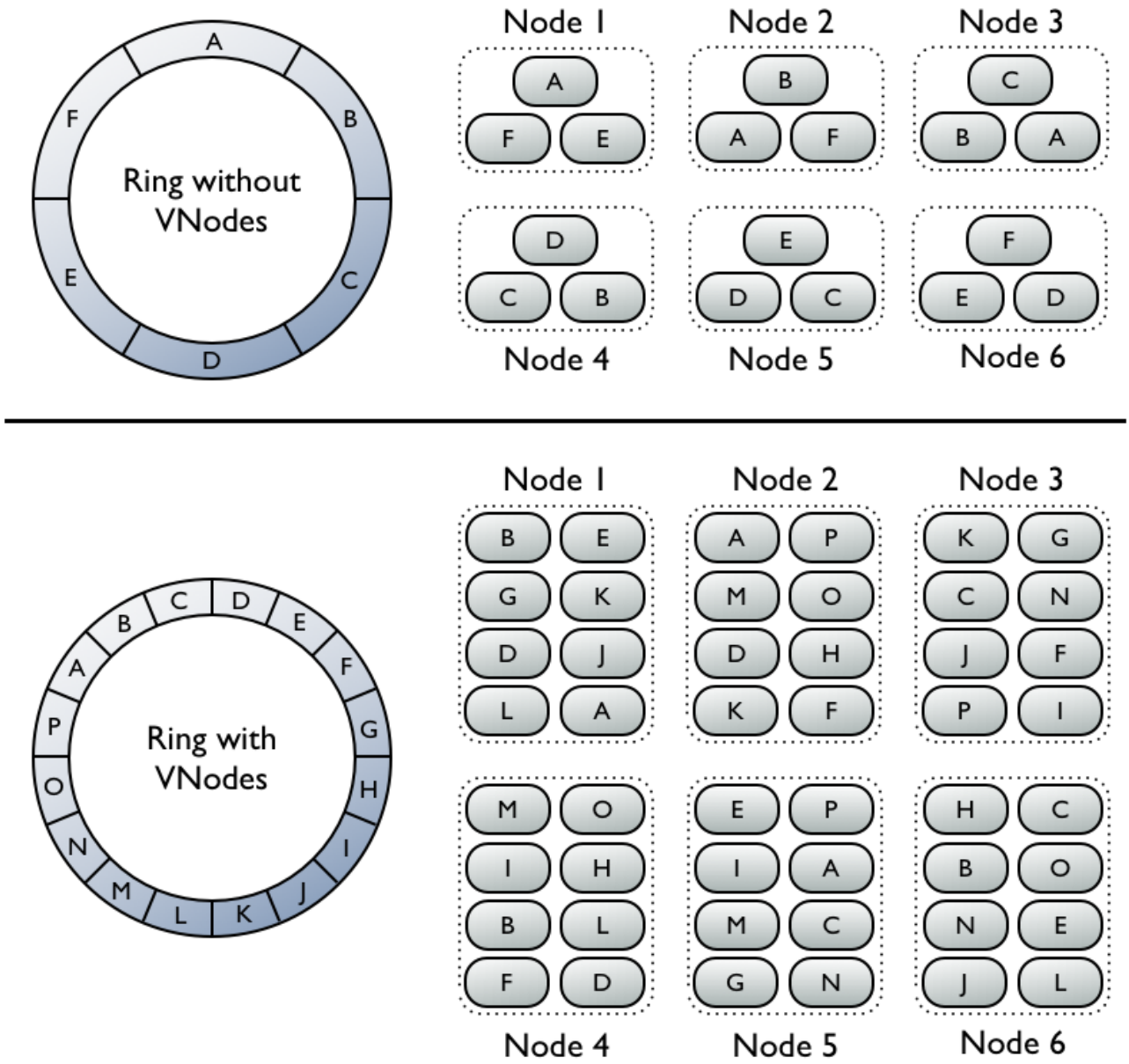
虚拟节点（Vnode）

每个虚拟节点对应一个token值，每个token决定了 `节点在环中的位置` 以及 `节点应当承担的一段连续的数据hash值的范围，` 因此每个节点都拥有一段连续的token，这一段连续的token，组成了一个封闭的圆环。

没有使用虚拟节点, Ring环的tokens数量=集群的机器数量. 比如一共有6个节点,所以token数=6.因为副本因子=3,一条记录要在集群中的三个节点存在. 简单地方式是计算rowkey的hash值,落在环中的哪个token上,第一份数据就在那个节点上,剩余两个副本落在这个节点在token环上的后两个节点.

图中的A,B,C,D,E,F是key的范围,真实的值是hash环空间,比如0~2^32区间分成10份.每一段是2^32的1/10.节点1包含A,F,E表示key范围在A,F,E的数据会存储到节点1上.以此类推.

若不使用虚拟节点则需手工为集群中每个节点计算和分配一个token。每个token决定了节点在环中的位置以及节点应当承担的一段连续的数据hash值的范围。如图上半部分，每个节点分配了一个单独的token代表环中的一个位置，每个节点存储将row key映射为hash值之后落在该节点应当承担的唯一的一段连续的hash值范围内的数据。每个节点也包含来自其他节点的row的副本。



而使用虚拟节点允许每个节点拥有多个较小的不连续的hash值范围。如图中下半部分，集群中的节点使用了虚拟节点，虚拟节点随机选择且不连续。数据的存放位置也由row key映射而得的hash值确定，但是是落在更小的分区范围内。

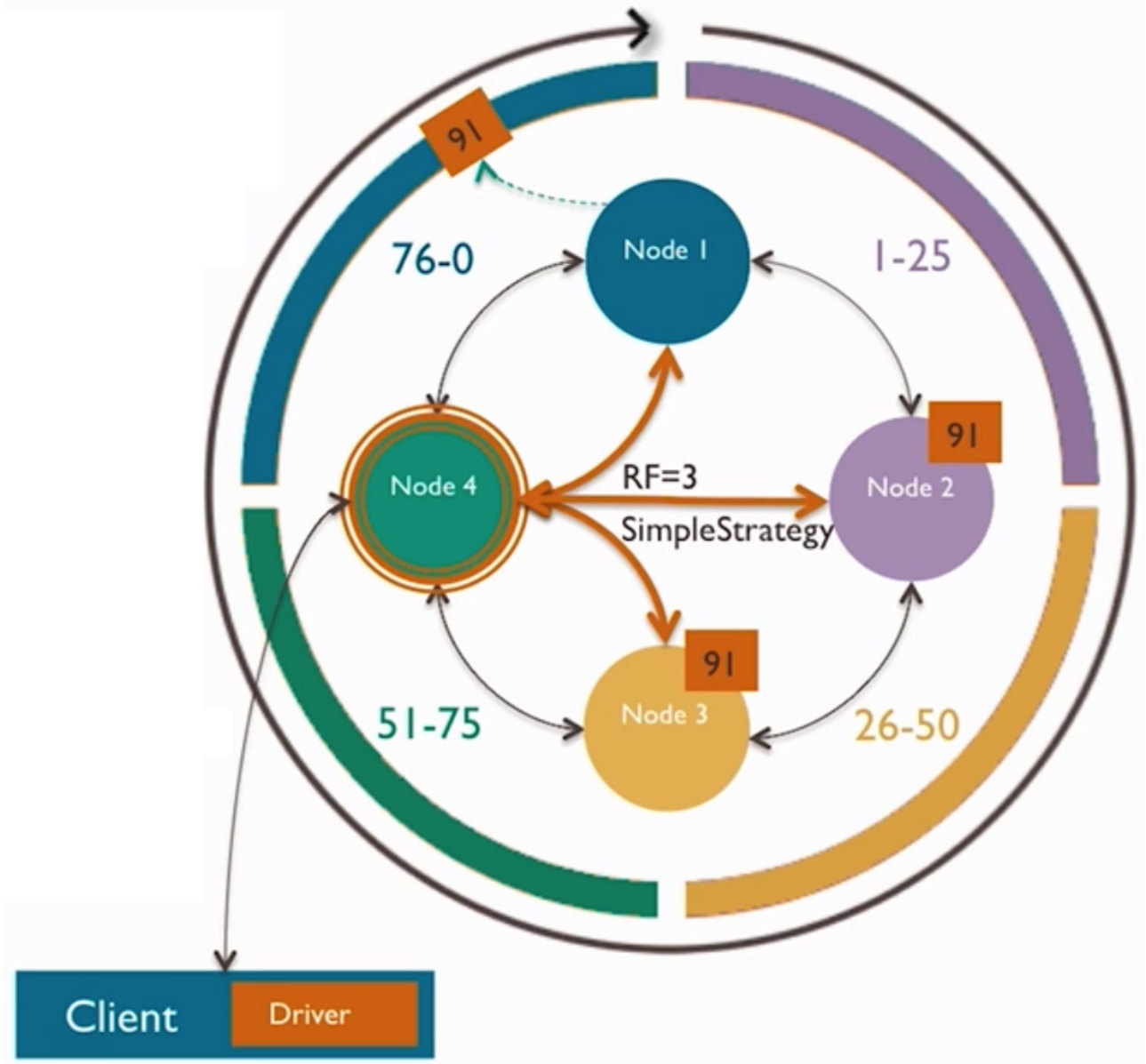
使用virtual nodes的好处

- 无需为每个节点计算、分配token
- 添加移除节点后无需重新平衡集群负载
- 重建异常的节点更快

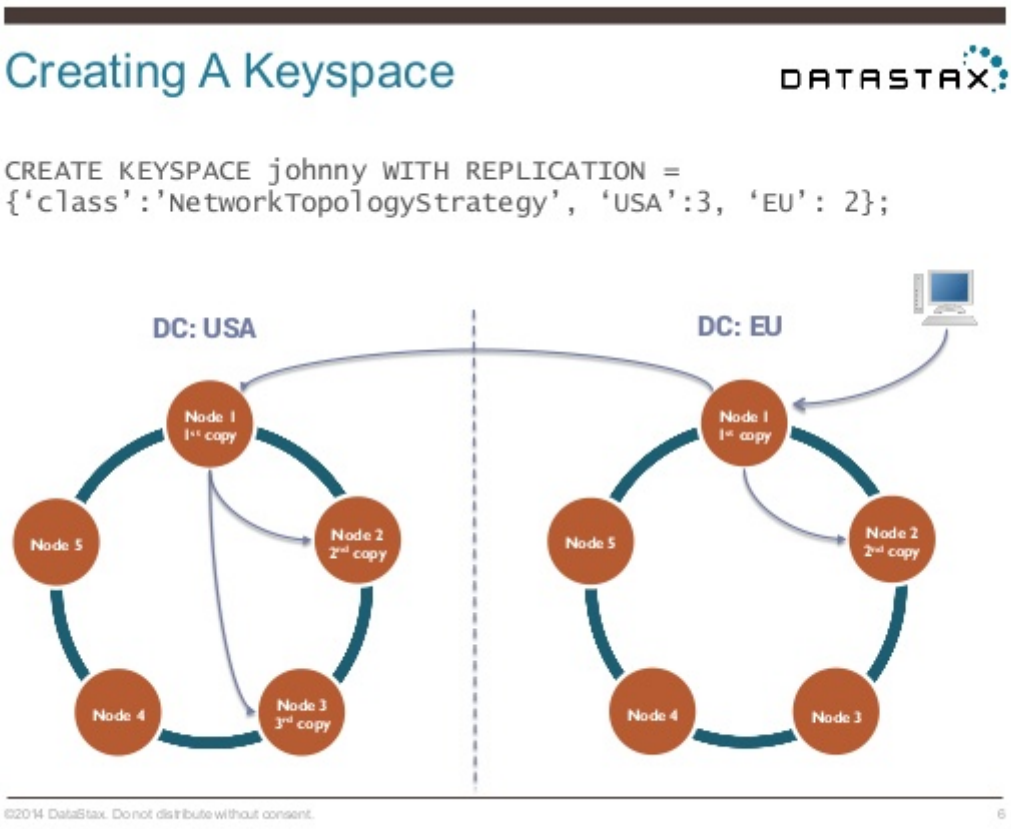
复制（Replication）

当前有两种可用的复制策略：

- SimpleStrategy：仅用于单数据中心，将第一个replica放在由partitioner确定的节点中，其余的replicas放在上述节点顺时针方向的后续节点中。



- NetworkTopologyStrategy：可用于较复杂的多数据中心。可以指定在每个数据中心分别存储多少份replicas。



复制策略在创建keyspace时指定，如

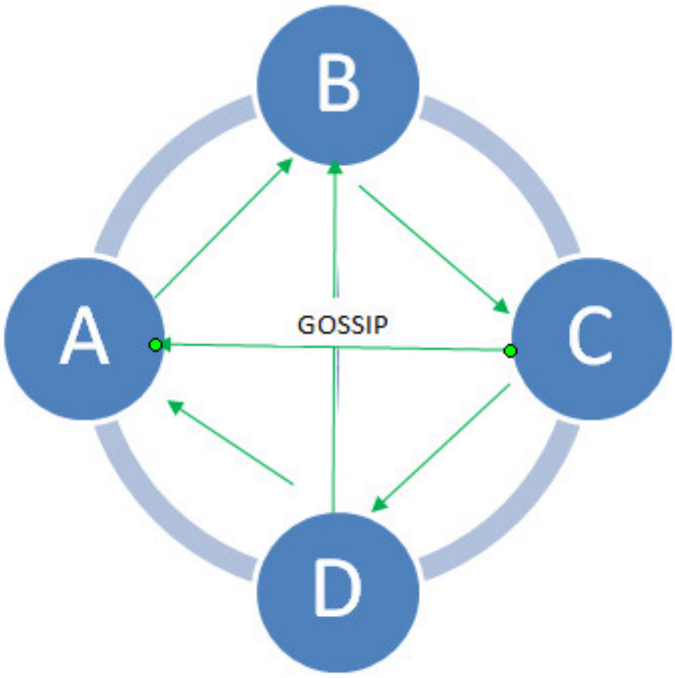
```
CREATE KEYSPACE Excelsior WITH REPLICATION = { 'class' : 'SimpleStrategy','replication_f
CREATE KEYSPACE Excalibur WITH REPLICATION = {'class' :'NetworkTopologyStrategy', 'dc1'
```

其中dc1、dc2这些数据中心名称要与snitch中配置的名称一致.上面的拓扑策略表示在dc1配置3个副本,在dc2配置2个副本

八卦（gossip）

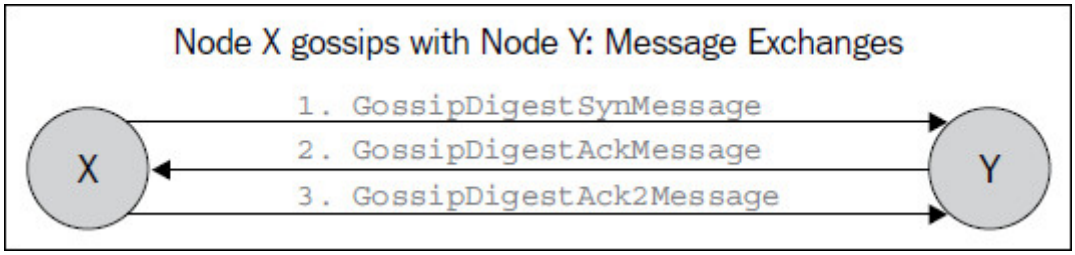
Gossip协议是群集中节点相互通信的内部通信技术。 Gossip是一种高效，轻量，可靠的节点间用于交换数据的广播协议，。 它是分散式的、容错和点对点的通信协议。 Cassandra使用gossipibing来进行对等发现和元数据传播。

gossip协议的具体表现形式就是配置文件中的seeds种子节点. 一个注意点是同一个集群的所有节点的种子节点应该一致.否则如果种子节点不一致, 有时候会出现集群分裂, 即会出现两个集群. 一般先启动种子节点,尽早发现集群中的其他节点.每个节点都和其他节点交换信息, 由于随机和概率,一定会穷举出集群的所有节点.同时每个节点都会保存集群中的所有其他节点.这样随便连到哪一个节点,都能知道集群中的所有其他节点. 比如cql随便连接集群的一个节点,都能获取集群所有节点的状态.也就是说任何一个节点关于集群中的节点信息的状态都应该是一致的!



PEER TO PEER DISTRIBUTION
MODEL OF CASSANDRA

gossip进程每秒运行一次，与至多3个其他节点交换信息，这样所有节点可很快了解集群中的其他节点信息。由于整个过程是分散的，并没有一个节点去协调每个节点的gossip信息，每个节点都会独立地选择一个到三个节点进行gossip通信，它始终选择集群中活动着的对等节点，概率性的选择seed节点和不可达的节点。



gossip协议和tcp三次握手比较相似，使用常规的广播协议，每轮只能有一条消息，并且允许消息在集群内部逐渐传播，但随着gossip协议，每条gossip消息都只会包含三条消息，增加了一定程度的逆熵。这个过程允许相互进行数据交换的节点之间快速的收敛。

首先系统需要配置几个种子节点，比如说A、B, 每个参与的节点都会维护所有节点的状态，node->(Key,Value,Version),版本号较大的说明其数据较新，节点P只能直接更新它自己的状态，节点P只能间接的通过gossip协议来更新本机维护的其他节点的数据。

大致的过程如下，

SYN:节点A向随机选择一些节点，这里可以只选择发送摘要，即不发送valus,避免消息过大

ACK:节点B接收到消息后，会将其与本地的合并，这里合并采用的是对比版本，版本较大的说明数据较新. 比如节点A向节点B发送数据C(key,valuea,2),而节点B本机存储的是C(key,valueb,3),那么因为B的版本比较新，合并之后的数据就是B本机存储的数据，然后会发回A节点。

ACK2:节点A接收到ACK消息，将其应用到本机的数据中.

一致性（Consistency Level）

一致性指一行数据的所有副本集是否最新和同步。Cassandra扩展了最终一致性的概念，对一个读或者写操作，所谓可调节的一致性的概念，指发起请求的客户端，可以通过consistency level参数，指定本次请求，需要的一致性。

写操作的一致性

级别	描述	用法
ANY	任意一个节点写操作已经成功。如果所有的replica节点都挂了，写操作还是可以在记录一个hinted handoff事件之后，返回成功。如果所有的replica节点都挂了，写入的数据，在挂掉的replica节点恢复之前，读不到。	最小的延时等待，并且确保写请求不会失败。相对于其他级别提供最低的一致性和最高的可用性。
ALL	写操作必须将指定行的数据写到所有replica节点的commit log和memtable。	相对于其他级别提供最高的一致性和最低的可用性。
EACH_QUORUM	写操作必须将指定行的数据写到每个数据中心的quorum数量的replica节点的commit log和memtable。	用于多数据中心集群严格的保证相同级别的一致性。例如，如果你希望，当一个数据中心挂掉了，或者不能满足quorum数量的replica节点写操作成功时，写请求返回失败。
LOCAL_ONE	任何一个本地数据中心内的replica节点写操作成功。	对于多数据中心的情况，往往期望至少一个replica节点写成功，但是，又不希望有任何跨数据中心的通信。LOCAL_ONE正好能满足这样的需求。
LOCAL_QUORUM	本地数据中心内quorum数量的replica节点写操作成功。避免跨数据中心的通信。	不能和SimpleStrategy一起使用。用于保证本地数据中心的数据一致性。
LOCAL_SERIAL	本地数据中心内quorum数量的replica节点有条件地（conditionally）写成功。	用于轻量级事务（lightweight transaction）下实现linearizable consistency，避免发生无条件的（unconditional）更新。。

级别	描述	用法
ONE	任意一个replica节点写操作已经成功。	满足大多数用户的需求。一般离coordinator节点具体最近的replica节点优先执行。

注意：

即使指定了consistency level ON或LOCAL_QUORUM，写操作还是会被发送给所有的replica节点，包括其他数据中心的里replica节点。consistency level只是决定了，通知客户端请求成功之前，需要确保写操作成功的replica节点的数量。

读一致性

级别	描述	用法
ALL	向所有replica节点查询数据，返回所有的replica返回的数据中，timestamp最新的数据。如果某个replica节点没有响应，读操作会失败。	相对于其他级别，提供最高的一致性和最低的可用性。
EACH_QUORUM	向每个数据中心内quorum数量的replica节点查询数据，返回时间戳最新的数据。	同LOCAL_QUORUM。
LOCAL_SERIAL	同SERIAL，但是只限制为本地数据中心。	同SERIAL。
LOCAL_QUORUM	向每个数据中心内quorum数量的replica节点查询数据，返回时间戳最新的数据。避免跨数据中心的通信。	使用SimpleStrategy时会失败。
LOCAL_ONE	返回本地数据中心内离coordinator节点最近的replica节点的数据。	同写操作Consistency level中该级别的用法。
ONE	返回由snitch决定的最近的replica返回的结果。默认情况下，后台会触发read repair确保其他replica的数据一致。	提供最高级别的可用性，但是返回的结果不一定最新。
QUORUM	读取所有数据中心中quorum数量的节点的结果，返回合并后timestamp最新的结果。	保证很强的一致性，虽然有可能读取失败。
SERIAL	允许读取当前的（包括uncommitted的）数据，如果读的过程中发现uncommitted的事务，则commit它。	轻量级事务。
TWO	返回两个最近的replica的最新数据。	和ONE类似。

级别	描述	用法
THREE	返回三个最近的replica的最新数据。	和TWO类似。

关于QUORUM级别

QUORUM级别确保数据写到指定quorum数量的节点。一个quorum的值由下面的公式四舍五入计算而得：

(sum_of_replication_factors / 2) + 1

sum_of_replication_factors指每个数据中心的所有replication_factor设置的总和。

例如，如果某个单数据中心的replication factor是3，quorum值为2-表示集群可以最多容忍1个节点down。如果replication factor是6，quorum值为4-表示集群可以最多容忍2个节点down。如果是双数据中心，每个数据中心的replication factor是3，quorum值为4-表示集群可以最多容忍2个节点down。如果是5数据中心，每个数据中心的replication factor是3，quorum值为8 。

如果想确保读写一致性可以使用下面的公式：

(nodes_written + nodes_read) > replication_factor

例如，如果应用程序使用QUORUM级别来读和写，replication factor 值为3，那么，该设置能够确保2个节点一定会被写入和读取。读节点数加上写节点数（4）个节点比replication factor （3）大，这样就能确保一致性。