Cassandra 配置

一 基本配置

1. 安装环境

Ubuntu 15.x 64位

1. 安装依赖

java jdk (具体安装不详述)

1. 下载cassandra 文件包

安装最新版本 2.2.x

wget [**http://apache.arvixe.com/cassandra/2.2.3/apache-cassandra-2.2.3-bin.tar.gz**](http://apache.arvixe.com/cassandra/2.2.3/apache-cassandra-2.2.3-bin.tar.gz)

1. 解压安装 Cassandra
   1. tar -zvxf apache-cassandra-2.2.3-bin.tar.gz
   2. mv apache-cassandra-2.2.3\* ~/cassandra
2. 启动

sudo ~/cassandra/bin/cassandra –f

如果出现 INFO  02:45:30 No gossip backlog; proceeding

表示启动成功。

二 远程连接修改

修改 cassandra.yaml 文件

$ vim cassandra.yaml

a) 启动 rpc\_start 将 false 改为 true

rpc\_start: ture

b) 将下面两个配置项的值由localhost改为IP地址,示例如下

listen\_address: 192.168.0.101

rpc\_address: 192.168.0.101

修改完成后测试：

sudo ~/cassandra/bin/cqlsh 192.168.0.101

出现如下命令行, 则表示连接成功

cqlsh>

三 集群（待更新）

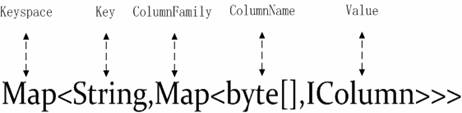
1. cassandra.yml 字段说明
2. cluster\_name : 代表一个族的标识，它通常代表一个集群。这个配置项在 Cassandra 没有存储数据时就必须指定，当 Cassandra 第一次启动后，它就会被写到 Cassandra 的系统表中，如果你要修改 Cluster Name 必须要删除 Cassandra 中数据.
3. auto\_bootstrap : 设置的主要目的是是否调整当前集群中的负载均衡。
4. Keyspaces : Cassandra 中 Keyspace 相当于关系数据库中的表空间的概念，可以理解为操作表的一个容器，它下面可以定义多个 ColumnFamily，这个 ColumnFamily 就相当于表了，它是存储数据的实体。

ColumnFamily 中几个属性的意义如下：

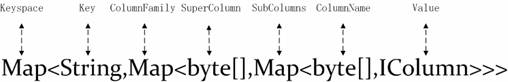
1. ColumnType。列的类型，有两种：Standard 和 Super，分别是标准列和超列，超列的含义是列还有一个父列。
2. CompareWith。表示的是列的排序规则，可以根据不同的数据类型进行排序如 TimeUUIDType，可以根据插入的时间排序
3. CompareSubcolumnsWith。子列的排序规则与 CompareWith 类似
4. RowsCached。查询时缓存的数据量，可以是多少条，也可以是百分比，如 10% 就是缓存 10% 的数据量，这个对查询性能影响很大，如果命中率高的话，可以显著提高查询效率。
5. KeysCached。缓存 ColumnFamily 中的 key，这个 key 就是对应到 Index.db 中的数据，如果没有在 RowsCached 中命中，那么就要到每个 SSTable 中查询，这时必然要查询 key，如果在 KeysCached 能命中就不需要到 Index.db 中查询了，省去了 IO 操作。

Cassandra 是一个 Key/Value 系统，从它的存储的逻辑结构来看分为：Keyspace、Key、ColumnFamily、Super Column 以及 Column 几个部分。很明显我们能看出每一对 Key/Value 都有一个寄生的容器，所以它实际上是由一个个 Map 容器构成的。

**标准的 Column 结构图**



##### 含有 Super Column 的结构图



1. ReplicaPlacementStrategy : 定义数据复制策略，默认是 org.apache.cassandra.locator.RackUnawareStrategy，数据复制到其它节点没有特别的规定。org.apache.cassandra.locator.RackAwareStrategy 是将节点分为不同的 Rack，这种方式不管是存数据还是查数据，都从不同的 Rack 的节点取数据或写数据。org.apache.cassandra.locator.DatacenterShardStategy 又将节点划分为不同的 Data Center，让数据放在不同数据中心，从而保证数据的安全性，例如可以按机房划分 Data Center，从而避免一个机房出现故障，会影响整个集群。
2. ReplicationFactor : 定义数据要保存几个备份，结合 ReplicaPlacementStrategy 可以把数据放在不同的地方。
3. EndPointSnitch : org.apache.cassandra.locator.EndPointSnitch 可以根据当前的网络情况选择更好的节点路由，一般默认即可。
4. Authenticator : 这个配置项可以控制数据访问的安全性，可以在 access.properties 和 passwd.properties 设置用户和密码。
5. Partitioner : 控制数据的分布规则，org.apache.cassandra.dht.RandomPartitioner 是随机分布，Cassandra 控制数据在不同的节点是通过 key 的来划分的，这个方式是将 key 进行 MD5 Hash，从而形成随机分布的 Token，然后根据这个 Token 将数据分布到不同的节点上。org.apache.cassandra.dht.OrderPreservingPartitioner 是取 key 的 Ascii 字符来划分的，因此我们可以根据 key 来主动控制数据的分布，例如我们可以给 key 加一个前缀，相同前缀的 key 分布在同一个节点中。
6. InitialToken : 给节点分配一个初始 Token，当节点第一次启动后这个 Token 就被写在系统表中。结合 Partitioner 就可以控制数据的分布。这个配置项可以让我们能调整集群的负载均衡。CommitLogDirectory、DataFileDirectories 这两个配置项是设置 CommitLog 和 SSTable 存储的目录。
7. Seeds:

关于 Seeds 节点的配置有这样几个疑问：

是不是集群中的所有节点都要配置在 seed 中。

本机需不需要配置在 seed 中。

关于第二个问题在前面中已经说明了，是否配置就决定是否作为 seed 节点来启动。关于第一个问题，答案是否定的，因为即使你把集群中的所有节点都配置在 seed 中，当 Cassandra 在启动时它也不会往每个 seed 发送心跳信息，而是随机选择一个节点与其同步集群中的其他所有节点状态。几个回合后这个节点同样能够获取集群中所有的节点的列表。这就是集群自治理的优点，只要能发现其中一个节点就能发现全部节点。

1. ListenAddress : 这个配置是用来监听集群中其它节点与本节点交换状态信息和数据的地址。需要注意的是当你配置为本机的 ip 地址没有问题，不配置通常也没问题，但是如果你没有配置或者配置成主机名，而你又把你的主机名绑定到 127.0.0.1 时，这时将会导致本节点不能加入到集群中，因为它接受不到其他节点过来的任何信息，防止出错直接绑定本机 ip 最好。
2. ThriftAddress : 监听 Client 的连接请求，不设或者配置成 0.0.0.0，监听所有地址的请求。
3. RowWarningThresholdInMB : 当 Cassandra 压缩时，如果一个 row 超出了配置的大小时打印 warn 日志，没有任何其它作用。
4. SlicedBufferSizeInKB 和 ColumnIndexSizeInKB : 分别是用来配置，根据 Slice 和 Column Name 来查询时 Cassandra 缓存数据的大小，当查询范围较小时可以适当设置大一点以提高命中率。
5. FlushDataBufferSizeInMB 和 FlushIndexBufferSizeInMB : 这两个配置项是设置 Cassandra 在将内存中的数据写到磁盘时一次写入的缓存量，适当提高这个两个值可以提高 Cassandra 的写性能。
6. MemtableThroughputInMB、MemtableOperationsInMillions 和 MemtableFlushAfterMinutes :

MemtableOperationsInMillions 是定义当前 Keyspace 对应的数据在内存中的缓存大小，Cassandra 默认是 64M，也就是当写到 Cassandra 的数据达到 64M 时，Cassandra 会将内存的数据写到本地磁盘中。

MemtableOperationsInMillions 是定义当前这个 Memtable 中所持有数据对象的个数，真实的个数是 MemtableOperationsInMillions\*1024\*1024。当超出这个数值时 Memtable 同样会被写到磁盘中。

MemtableFlushAfterMinutes 的作用是，当前两个条件都长时间不满足时，Memtable 中数据会一直不会写到磁盘，这也不合适，所以设置了一个时间限制，当超过这个时间长度时 Memtable 中的数据也会被写到磁盘中。

所以 Memtable 中的数据何时被写到写到磁盘是由这三个值决定，任何一个条件满足都会写到磁盘。

1. ConcurrentReads 和 ConcurrentWrites : 这两个是定义 Cassandra 用来处理 read 和 write 的线程池中线程的个数，根据当前的测试结果，读写的性能大慨是 1:10，适当的设置这两个值不仅要根据读写的性能，还要参考当前机器的处理性能。当机器的 load 很高，但是 cpu 的利用率却很低时，很明显是由于连接数过多，Cassandra 的已经处理不过来都处于等待状态。这样就可以适当增加读写的线程数，同样如果当读的请求大于写的请求时，也应该适当增加读的线程数，反之亦然。
2. CommitLogSync、CommitLogSyncPeriodInMS 和 CommitLogSyncBatchWindowInMS : 我们知道 Cassandra 是先写到 CommitLog 中再写到 Memtable 和磁盘中。如果每写一条数据都要写一次到磁盘那样性能将会大打折扣。Cassandra 为了提高写 CommitLog 的性能提供了两种写的方式。

Periodic : 周期性的把 CommitLog 数据写到磁盘中，这个时间周期由 CommitLogSyncPeriodInMS 指定，默认是 10000MS, 如果是这种方式，可想而知 Cassandra 并不能完全保证写到 Cassandra 的数据不会丢失，最坏的情况就是在这个时间段的数据会被丢失，但是 Cassandra 的解释是通过数据的多个备份，来能提高安全性。但是如果是单机存储数据，最坏的情况仍然会丢失 10000MS 时间段写入的数据。可以说这种方式写 CommitLog 是完全的异步的方式。

Batch : 这种方式是等待数据被写到磁盘中才会返回，与前面相比安全性会得到保证，它能保证 100% 数据的正确性。但也并不是每写一条数据都立即写到磁盘中，而是有一个延迟时间，这个延迟时间就是由 CommitLogSyncBatchWindowInMS 指定的，也就是写一条数据到 CommitLog 的最大时间是 CommitLogSyncBatchWindowInMS 指定的时间，理想的时间范围是 0.1~10MS 之间。这个时间既要平衡客户端的相应时间也要考虑服务器写数据到磁盘的性能。

这两种方式各有好处，如果数据是存储在有多个备份的集群中，第一种情况下，丢数据的情况几乎为零，但是性能肯定会比第二种要好很多。如果是单机情况下，要保证数据的安全性第二种较合适。

1. GCGraceSeconds : 这个配置项不是 Java 中的 gc 回收内存，但是其功能类似于 jvm 中 gc，它也是回收已经没有被关联的数据，例如已经被标识为删除的数据，Cassandra 处理数据有点奇怪，即使数据被标识为删除，但是只要是没有超过 GCGraceSeconds 的时间这个数据仍然是存在的，也就是可以定制数据的实效时间，超出这个时间数据将会被回收。

其余说明：

最小化配置集群

cluster\_name

集群的名字，默认情况下是TestCluster。对于这个属性的配置可以防止某个节点加入到其他集群中去，所以一个集群中的节点必须有相同的cluster\_name属性。

listen\_address

Cassandra需要监听的IP或主机名，默认是localhost。建议配置私有IP，不要用0.0.0.0。

commitlog\_directory

commit log的保存目录，压缩包安装方式默认是/var/lib/cassandra/commitlog。通过前面的了解，我们可以知道，把这个目录和数据目录分开存放到不同的物理磁盘可以提高性能。

data\_file\_directories

数据文件的存放目录，压缩包安装方式默认是/var/lib/cassandra/data。为了更好的效果，建议使用RAID 0或SSD。

save\_caches\_directory

保存表和行的缓存，压缩包安装方式默认是/var/lib/cassandra/saved\_caches。

通常使用：用得比较频繁的属性

在启动节点前，需要仔细评估你的需求。

commit\_failure\_policy

提交失败时的策略（默认stop）：

stop：关闭gossip和Thrift，让节点挂起，但是可以通过JMX进行检测。

stop\_commit：关闭commit log，整理需要写入的数据，但是提供读数据服务。

ignore：忽略错误，使得该处理失败。

disk\_failure\_policy

设置Cassandra如何处理磁盘故障（默认stop）。

stop：关闭gossip和Thrift，让节点挂起，但是可以通过JMX进行检测。

stop\_paranoid：在任何SSTable错误时就闭gossip和Thrift。

best\_effort：这是Cassandra处理磁盘错误最好的目标。如果Cassandra不能读取磁盘，那么它就标记该磁盘为黑名单，可以继续在其他磁盘进行写入数据。如果Cassandra不能从磁盘读取数据，那个这些SSTable就标记为不可读，其他可用的继续堆外提供服务。所以就有可能在一致性水平为ONE时会读取到过期的数据。

ignore：用于升级情况。

endpoint\_snitch

用于设置Cassandra定位节点和路由请求的snitch（默认org.apache.cassandra.locator.SimpleSnitch），必须设置为实现了IEndpointSnitch的类。

rpc\_address

用于监听客户端连接的地址。可用的包括：

0.0.0.0监听所有地址

IP地址

主机名

不设置：使用hosts文件或DNS

seed\_provider

需要联系的节点地址。Cassandra使用-seeds集合找到其他节点并学习其整个环中的网络拓扑。

– class\_name：（默认org.apache.cassandra.locator.SimpleSeedProvider），可用自定义，但通常不必要。

– seeds：（默认127.0.0.1）逗号分隔的IP列表。

compaction\_throughput\_mb\_per\_sec

限制特定吞吐量下的压缩速率。如果插入数据的速度越快，越应该压缩SSTable减少其数量。推荐16-32倍于写入速度（MB/s）。如果是0表示不限制。

memtable\_total\_space\_in\_mb

指定节点中memables最大使用的内存数（默认1/4heap）。

concurrent\_reads

(默认32）读取数据的瓶颈是在磁盘上，设置16倍于磁盘数量可以减少操作队列。

concurrent\_writes

（默认32）在Cassandra里写很少出现I/O不稳定，所以并发写取决于CPU的核心数量。推荐8倍于CPU数。

incremental\_backups

（默认false）最后一次快照发生时备份更新的数据（增量备份）。当增量备份可用时，Cassandra创建一个到SSTable的的硬链接或者流式存储到本地的备份/子目录。删除这些硬链接是操作员的责任。

snapshot\_before\_compaction

（默认false）启用或禁用在压缩前执行快照。这个选项在数据格式改变的时候来备份数据是很有用的。注意使用这个选项，因为Cassandra不会自动删除过期的快照。

phi\_convict\_threshold

（默认8）调整失效检测器的敏感度。较小的值增加了把未响应的节点标注为挂掉的可能性，反之就会降低其可能性。在不稳定的网络环境下（比如EC2），把这个值调整为10或12有助于防止错误的失效判断。大于12或小于5的值不推荐！

性能调优

commit\_sync

（默认：periodic）Cassandra用来确认每毫秒写操作的方法。

periodic：和commitlog\_sync\_period\_in\_ms（默认10000 – 10 秒）一起控制把commit log同步到磁盘的频繁度。周期性的同步会立即确认。

batch：和commitlog\_sync\_batch\_window\_in\_ms（默认disabled）一起控制Cassandra在执行同步前要等待其他写操作多久时间。当使用该方法时，写操作在同步数据到磁盘前不会被确认。

commitlog\_periodic\_queue\_size

（默认1024\*CPU的数量）commit log队列上的等待条目。当写入非常大的blob时，请减少这个数值。比如，16倍于CPU对于1MB的Blob工作得很好。这个设置应该至少和concurrent\_writes一样大。

commitlog\_segment\_size\_in\_mb

（默认32）设置每个commit log文件段的大小。一个commit log段在其所有数据刷新到SSTable后可能会被归档、删除或回收。数据的总数可以潜在的包含系统中所有表的commit log段。默认值适合大多数情况，当然你也可以修改，比如8或16MB。

commitlog\_total\_space\_in\_mb

（默认32位JVM为32,64位JVM为1024）commit log使用的总空间。如果使用的空间达到以上指定的值，Cassandra进入下一个临近的部分，或者把旧的commit log刷新到磁盘，删除这些日志段。该个操作减少了在启动时加载过多数据引起的延迟，防止了把无限更新的表保存到有限的commit log段中。

compaction\_preheat\_key\_cache

（默认true）当设置为true的时候，缓存的row key在压缩期间被跟踪，并且重新缓存其在新压缩的SSTable中的位置。如果有及其大的key要缓存，把这个值设为false。

concurrent\_compactors

（默认每个CPU一个）设置每个节点并发压缩处理的值，不包含验证修复逆商。并发压缩可以在混合读写工作下帮助保持读的性能——通过减缓把一堆小的SSTable压缩而进行的长时间压缩。如果压缩运行得太慢或太快，请首先修改compaction\_throughput\_mb\_per\_sec的值。

in\_memory\_compaction\_limit\_in\_mb

（默认64）针对数据行在内存中的压缩限制。超大的行会溢出磁盘并且使用更慢的二次压缩。当这个情况发生时，会对特定的行的key记录一个消息。推荐5-10%的Java对内存大小。

multithreaded\_compaction

（默认false）当设置为true的时候，每个压缩操作使用一个线程，一个线程用于合并SSTable。典型的，这个只在使用SSD的时候有作用。使用HDD的时候，受限于磁盘I/O（可参考compaction\_throughput\_mb\_per\_sec）。

preheat\_kernel\_page\_cache

（默认false） 启用或禁用内核页面缓存预热压缩后的key缓存。当启用的时候会预热第一个页面（4K）用于优每个数据行的顺序访问。对于大的数据行通常是有危害的。

file\_cache\_size\_in\_mb

（小于1/4堆内存或512）用于SSTable读取的缓存内存大小。

memtable\_flush\_queue\_size

（默认4）等待刷新的满的memtable的数量（等待写线程的memtable）。最小是设置一个table上索引的最大数量。

memtable\_flush\_writers

（默认每数据目录一个）设置用于刷新memtable的线程数量。这些线程是磁盘I/O阻塞的，每个线程在阻塞的情况下都保持了memtable。如果有大的堆内存和很多数据目录，可以增加该值提升刷新性能。

column\_index\_size\_in\_kb

（默认64）当数据到达这个值的时候添加列索引到行上。这个值定义了多少数据行必须被反序列化来读取列。如果列的值很大或有很多列，那么就需要增加这个值。

populate\_io\_cache\_on\_flush

（默认false）添加新刷新或压缩的SSTable到操作系统的页面缓存

reduce\_cache\_capacity\_to

（默认0.6）设置由reduce\_cache\_sizes\_at定义的Java对内存达到限制时的最大缓存容量百分比。

reduce\_cache\_sizes\_at

（默认0.85）当Java对内存使用率达到这个百分比，Cassandra减少通过reduce\_cache\_capacity\_to定义的缓存容量。禁用请使用1.0。

stream\_throughput\_outbound\_megabits\_per\_sec

（默认200）限制所有外出的流文件吞吐量。Cassandra在启动或修复时使用很多顺序I/O来流化数据，这些可以导致网络饱和以及降低RPC的性能。

trickle\_fsync

（默认false）当使用顺序写的时候，启用该选项就告诉fsync强制操作系统在trickle\_fsync\_interval\_in\_kb设定的间隔刷新脏缓存。建议在SSD启用。

trickle\_fsync\_interval\_in\_kb

（默认10240）设置fsync的大小