【译】深入浅出 Cassandra 的删除和墓碑

2020-06-19 1411

简介： 深入浅出介绍 cassandra 作为一款分布式数据库如何支持删除操作

【原文链接】About Deletes and Tombstones in Cassandra

从Apache Cassandra这样的系统中删除分布式和复制式的数据，要比在关系型数据库中删除数据要棘手得多。当我们考虑到Cassandra将其数据存储在磁盘上的不可改变的文件中时，删除的过程就变得更加有趣。在这样的系统中，为了记录发生了删除的事实，需要写入一个叫做 "墓碑"（tombstone）的特殊值，用于标识之前的值已经被删除。尽管这可能看起来很不寻常甚至有悖直觉（特别是当你意识到删除实际上会占用磁盘空间时），我们将用这篇博文来解释实际发生的事情，同时附上例子，你可以自行试验加深理解。

Cassandra：可用性 + 一致性

在我们深入了解细节之前，我们应该先退后一步进行回顾，看看Cassandra作为一个分布式系统是如何工作的，尤其是在可用性和一致性的背景下。这对于正确理解分布式删除以及其对应的潜在问题将会有所帮助。

可用性。为了确保可用性，Cassandra会复制数据。具体来说，根据复制因子（RF），每个数据的多个副本存储在不同的节点上。RF定义了每个数据中心每个 keyspace 要保存的副本数量。根据配置，每个副本也可以由不同的机架保存，只要有足够的机架可用，并且配置对应 snitch 和 topology 策略。采用这样的方法，当任何节点或机架发生故障时，仍然可以从其他副本中读取数据。

一致性。为了确保读取的数据具有很强的一致性，我们必须遵守以下规则。

```
CL.READ  = 用于读取的一致性级别（CL）。基本上是指Cassandra认为读取成功而必须确认的节点数量。
CL.WRITE = 用于写入的一致性级别（CL）
RF       = 复制因子
```

```
CL.READ + CL.WRITE > RF
```

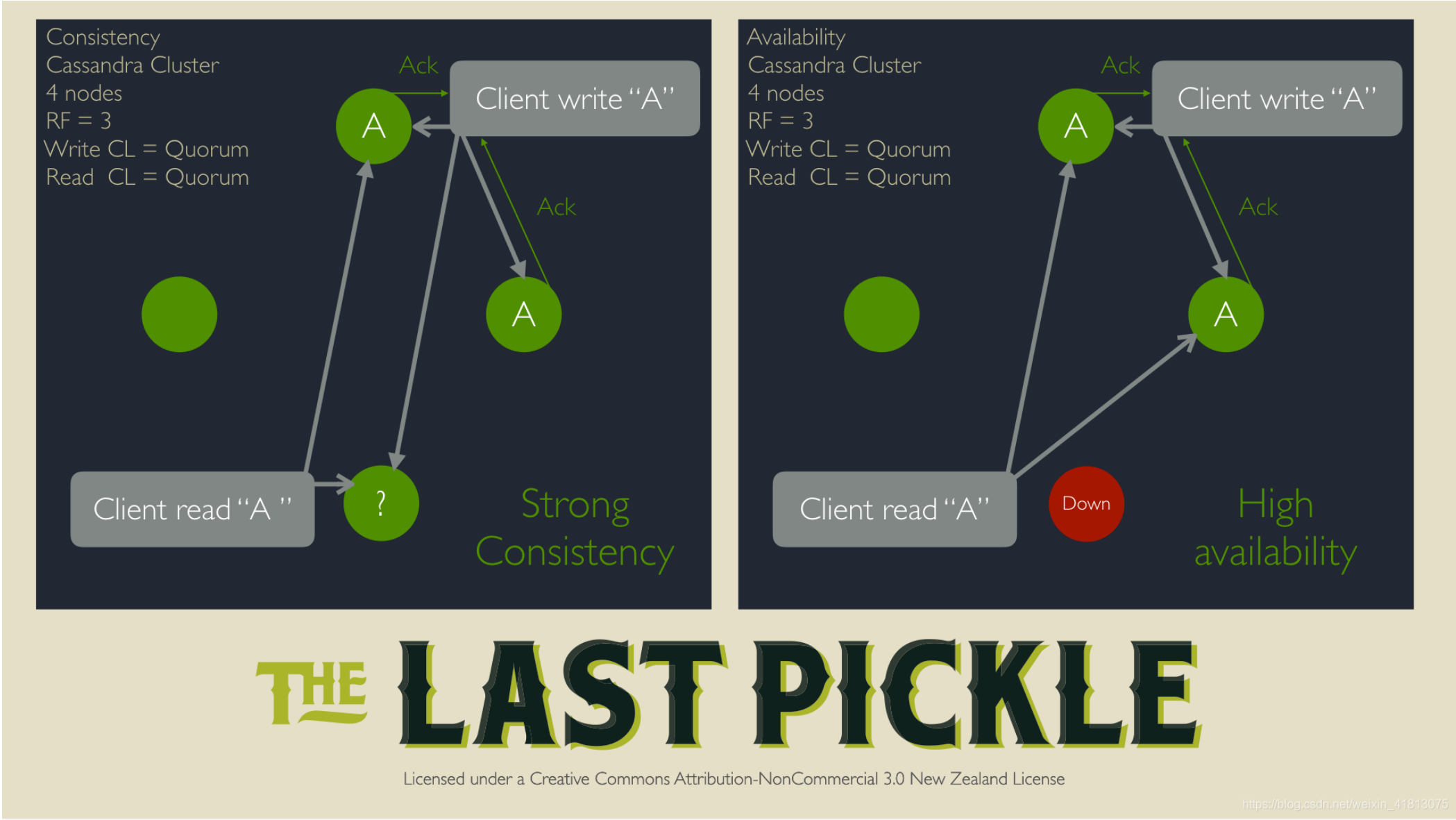
这样我们就可以确保从至少一个写入数据的节点中读取。

用例：让我们考虑以下常见的配置

```
RF        = 3
CL.READ   = QUORUM = RF/2 + 1 = 2
CL.WRITE = QUORUM = RF/2 + 1 = 2

CL.READ + CL.WRITE > RF --> 4 > 3
```

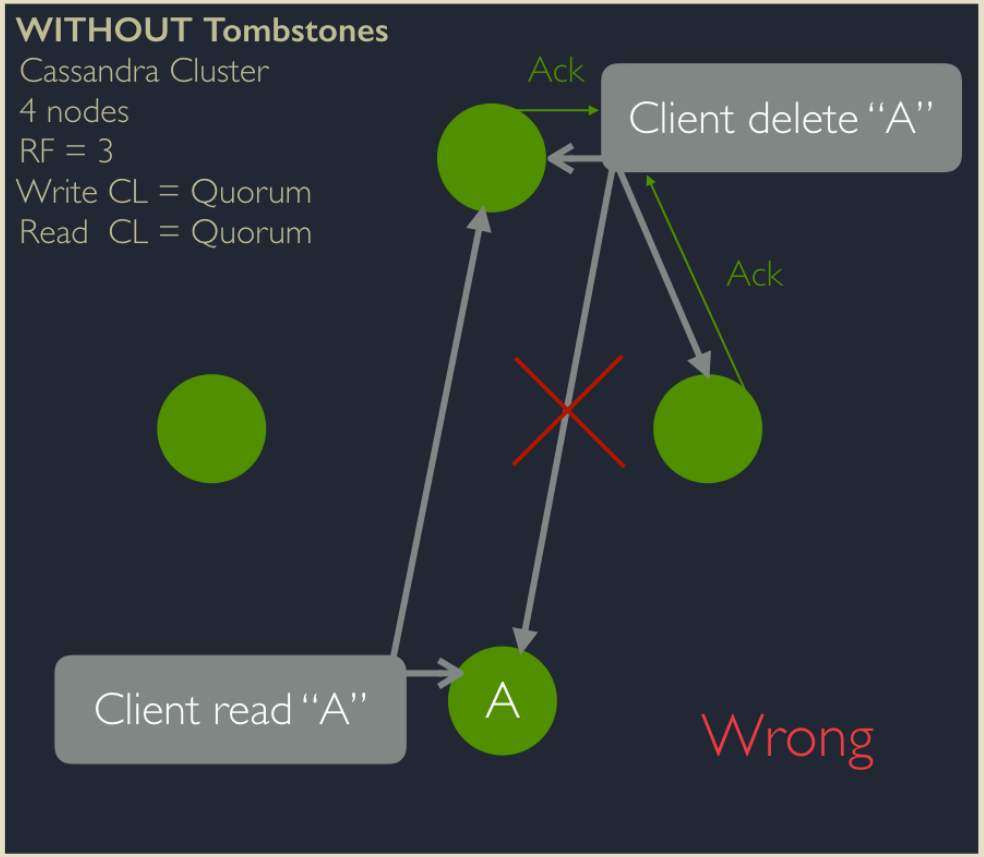
通过这个配置，避免了单点故障（SPOF），从而收获了高可用性。我们可以承受损失一个节点，因为我们确信任何读取请求都会至少在一个节点上获取写入的数据，然后应用 Last Write Wins (LWW) 算法来选择哪个节点持有这次读的正确数据。

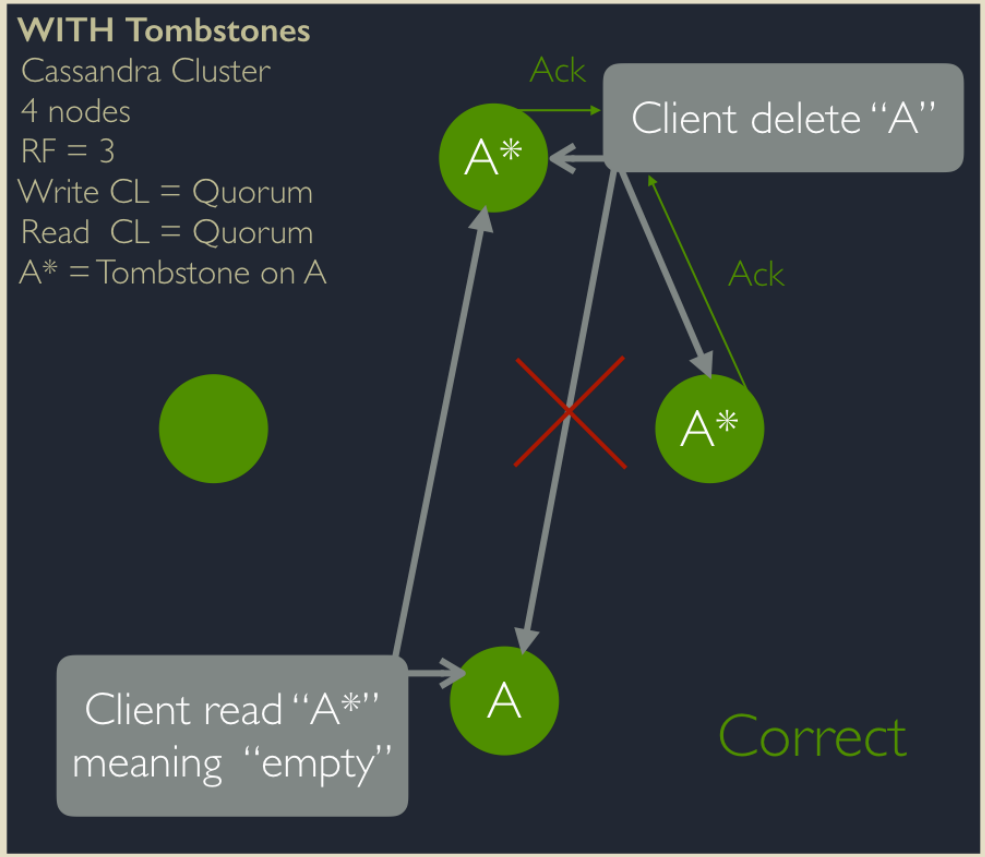


理解了上述配置和行为，下面来看一些执行删除的例子。

分布式删除的问题

从上一节的内容了解到这种配置之下应该是强一致的。让我们暂时忘掉墓碑（tombstone），考虑一下 Cassandra 没有使用墓碑删除数据的情况。让我们考虑一个成功的删除，在一个节点上失败了（三个节点中，RF=3）。这个删除仍然会被认为是成功的（两个节点承认了删除，使用CL.QUORUM）。涉及该节点的下一次读取将是一个模棱两可的读取，因为没有办法确定什么是正确的：返回一个空响应还是返回数据？Cassandra一定会认为返回数据是正确的做法，所以删除经常会导致数据重新出现，被称为 "僵尸 "或 "幽灵"，它们的行为将是不可预测的。





THE LAST PICKLE

注意：这个问题即使通过墓碑（tombstone）的方式也并没有完全解决，而是按照以下方式解决：作为 Cassandra 操作者，我们必须对任何执行删除的集群至少每 gc_grace_seconds 运行一次全面修复。参见下面的 "墓碑删除 "部分。

从 Cassandra 单节点的角度看删除问题

如前所述，墓碑解决了使用不可变文件（sstable）存储数据的系统中删除数据的问题。

Cassandra 的特点之一是它使用了一个日志结构的合并树（LSM树），而大多数 RDBMS 使用的是 B 树。理解这一点的最好方法是记住 Cassandra 总是将写的数据追加更新，读的时候负责将一行的数据碎片合并在一起，挑选每个列的最新版本返回。

LSM树的另一个属性是数据写在不可变的文件中（在Cassandra中称为SSTables）。正如最初讨论的那样，那么很明显，通过这样的系统，删除只能通过一种特殊的写来完成。读取将获取墓碑，而不考虑墓碑时间戳之前的任何数据。

墓碑

在Cassandra的上下文中，墓碑是与标准数据一起存储的特定数据。删除只不过是插入一个墓碑。当Cassandra读取数据时，它将从 memtable 和SSTables 中合并所有请求行的碎片。然后，它应用 Last Write Wins（LWW）算法来选择什么是正确的数据，不管它是标准值还是墓碑。

举个例子。

让我们考虑以下例子，在一个有3个节点的Cassandra 3.7集群上（使用CCM）。

```
CREATE KEYSPACE tlp_lab WITH replication = {'class': 'NetworkTopologyStrategy', 'datacenter1' : 3};
CREATE TABLE tlp_lab.tombstones (fruit text, date text, crates set<int>, PRIMARY KEY (fruit, date));
```

并添加一些数据

```
INSERT INTO tlp_lab.tombstones (fruit, date, crates) VALUES ('apple', '20160616', {1,2,3,4,5});
INSERT INTO tlp_lab.tombstones (fruit, date, crates) VALUES ('apple', '20160617', {1,2,3});
```

```
INSERT INTO tlp_lab.tombstones (fruit, date, crates) VALUES ( 'pickles' ,  '20160616' , {6,7,8}) USING TTL 2592000;
```

这是刚才存储的数据。

```
alain$ echo "SELECT * FROM tlp_lab.tombstones LIMIT 100;" | cqlsh

fruit   | date     | crates
---------+----------+-----------------
apple | 20160616 | {1, 2, 3, 4, 5}
apple | 20160617 |       {1, 2, 3}
pickles | 20160616 |       {6, 7, 8}
```

现在我们需要手动刷新数据（即在磁盘上写一个新的SSTable，释放内存），因为内存不像磁盘上的SSTable，其内容是支持变更的，所以内存中的墓碑，更准确的说是memtable中的墓碑，会覆盖memtable中现有的任何值，这一行为与应用到磁盘的 sstable 上时存在区别。

```
nodetool -p 7100 flush
```

我们现在可以看到磁盘上的数据。

```
alain$ ll /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/
total 72
drwxr-xr-x  11 alain  staff   374 Jun 16 20:53 .
drwxr-xr-x   3 alain  staff   102 Jun 16 20:25 ..
drwxr-xr-x   2 alain  staff    68 Jun 16 17:05 backups
-rw-r--r--   1 alain  staff    43 Jun 16 20:53 mb-5-big-CompressionInfo.db
-rw-r--r--   1 alain  staff   127 Jun 16 20:53 mb-5-big-Data.db
-rw-r--r--   1 alain  staff    10 Jun 16 20:53 mb-5-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 16 20:53 mb-5-big-Filter.db
-rw-r--r--   1 alain  staff    20 Jun 16 20:53 mb-5-big-Index.db
-rw-r--r--   1 alain  staff  4740 Jun 16 20:53 mb-5-big-Statistics.db
```

```
-rw-r--r--   1 alain  staff    61 Jun 16 20:53 mb-5-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 16 20:53 mb-5-big-TOC.txt
```

我们可以使用SSTabledump工具将 sstable 中的内容以可读的方式呈现

```
alain$ SSTabledump /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/mb-5-big-Data.db
[
  {
    "partition" : {
      "key" : [ "apple" ],
      "position" : 0
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 19,
        "clustering" : [ "20160616" ],
        "liveness_info" : { "tstamp" : "2016-06-16T18:52:41.900451Z" },
        "cells" : [
          { "name" : "crates", "deletion_info" : { "marked_deleted" : "2016-06-16T18:52:41.900450Z", "local_delete_time" : "2016-06-16T18:
          { "name" : "crates", "path" : [ "1" ], "value" : "" },
          { "name" : "crates", "path" : [ "2" ], "value" : "" },
          { "name" : "crates", "path" : [ "3" ], "value" : "" },
          { "name" : "crates", "path" : [ "4" ], "value" : "" },
          { "name" : "crates", "path" : [ "5" ], "value" : "" }
        ]
      },
      {
        "type" : "row",
        "position" : 66,
        "clustering" : [ "20160617" ],
        "liveness_info" : { "tstamp" : "2016-06-16T18:52:41.902093Z" },
        "cells" : [
          { "name" : "crates", "deletion_info" : { "marked_deleted" : "2016-06-16T18:52:41.902092Z", "local_delete_time" : "2016-06-16T18:
```

```
          { "name" : "crates", "path" : [ "1" ], "value" : "" },
          { "name" : "crates", "path" : [ "2" ], "value" : "" },
          { "name" : "crates", "path" : [ "3" ], "value" : "" }
        ]
      }
    ]
  },
  {
    "partition" : {
      "key" : [ "pickles" ],
      "position" : 104
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 125,
        "clustering" : [ "20160616" ],
        "liveness_info" : { "tstamp" : "2016-06-16T18:52:41.903751Z", "ttl" : 2592000, "expires_at" : "2016-07-16T18:52:41Z", "expired" :
        "cells" : [
          { "name" : "crates", "deletion_info" : { "marked_deleted" : "2016-06-16T18:52:41.903750Z", "local_delete_time" : "2016-06-16T18:
          { "name" : "crates", "path" : [ "6" ], "value" : "" },
          { "name" : "crates", "path" : [ "7" ], "value" : "" },
          { "name" : "crates", "path" : [ "8" ], "value" : "" }
        ]
      }
    ]
  }
]
```


现在磁盘上存储了两个分区（3行，2行共享同一个分区）。现在让我们考虑不同类型的删除。

Cell 删除

在Cassandra存储引擎中，来自特定行的一列称为 "Cell"。

从行中删除一个单元格

```
DELETE crates FROM tlp_lab.tombstones WHERE fruit='apple' AND date ='20160617';
```

在对应的行中，crates 列显示为 null。

```
alain$ echo "SELECT * FROM tlp_lab.tombstones LIMIT 100;" | cqlsh

fruit   | date     | crates
---------+----------+-----------------
apple | 20160616 | {1, 2, 3, 4, 5}
apple | 20160617 |            null
pickles | 20160616 |       {6, 7, 8}

(3 rows)
```

刷新后我们在磁盘上多了一个SSTable，mb-6-big。

```
alain$ ll /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/
total 144
drwxr-xr-x  19 alain  staff   646 Jun 16 21:12 .
drwxr-xr-x   3 alain  staff   102 Jun 16 20:25 ..
drwxr-xr-x   2 alain  staff    68 Jun 16 17:05 backups
-rw-r--r--   1 alain  staff    43 Jun 16 20:53 mb-5-big-CompressionInfo.db
-rw-r--r--   1 alain  staff   127 Jun 16 20:53 mb-5-big-Data.db
-rw-r--r--   1 alain  staff    10 Jun 16 20:53 mb-5-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 16 20:53 mb-5-big-Filter.db
-rw-r--r--   1 alain  staff    20 Jun 16 20:53 mb-5-big-Index.db
-rw-r--r--   1 alain  staff  4740 Jun 16 20:53 mb-5-big-Statistics.db
-rw-r--r--   1 alain  staff    61 Jun 16 20:53 mb-5-big-Summary.db
```

```
-rw-r--r--   1 alain  staff    92 Jun 16 20:53 mb-5-big-TOC.txt
-rw-r--r--   1 alain  staff    43 Jun 16 21:12 mb-6-big-CompressionInfo.db
-rw-r--r--   1 alain  staff    43 Jun 16 21:12 mb-6-big-Data.db
-rw-r--r--   1 alain  staff    10 Jun 16 21:12 mb-6-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 16 21:12 mb-6-big-Filter.db
-rw-r--r--   1 alain  staff     9 Jun 16 21:12 mb-6-big-Index.db
-rw-r--r--   1 alain  staff  4701 Jun 16 21:12 mb-6-big-Statistics.db
-rw-r--r--   1 alain  staff    59 Jun 16 21:12 mb-6-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 16 21:12 mb-6-big-TOC.txt
```

而这里是mb-6-big的内容。

```
alain$ SSTabledump /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/mb-6-big-Data.db
[
  {
    "partition" : {
      "key" : [ "apple" ],
      "position" : 0
    },
    "rows" : [
      {
        "type" : "row",
        "position" : 19,
        "clustering" : [ "20160617" ],
        "cells" : [
          { "name" : "crates", "deletion_info" : { "marked_deleted" : "2016-06-16T19:10:53.267240Z", "local_delete_time" : "2016-06-16T19:
        ]
      }
    ]
  }
]
```

看看这个墓碑删除的 cell 和插入的行 cell 相比有多相似。partition、row 和cell 都还在，只是在列的层面上没有liveness_info了。deletion_info 字段也相应地更新了。这就是一个 Cell 墓碑。

Row 删除

从分区中删除一条记录

```
DELETE FROM tlp_lab.tombstones WHERE fruit='apple' AND date ='20160617';
```

删除后，该行不再显示

```
alain$ echo "SELECT * FROM tlp_lab.tombstones LIMIT 100;" | cqlsh

 fruit   | date     | crates
---------+----------+-----------------
   apple | 20160616 | {1, 2, 3, 4, 5}
 pickles | 20160616 |       {6, 7, 8}

(2 rows)
```

刷新后，磁盘上多了一个SSTable，'mb-7-big'，它的样子如下。

```
alain$ SSTabledump /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/mb-7-big-Data.db
[
  {
    "partition" : {
      "key" : [ "apple" ],
      "position" : 0
    },
    "rows" : [
      {
```

```
      {
        "type" : "row",
        "position" : 19,
        "clustering" : [ "20160617" ],
        "deletion_info" : { "marked_deleted" : "2016-06-16T19:31:41.142454Z", "local_delete_time" : "2016-06-16T19:31:41Z" },
        "cells" : [ ]
      }
    ]
  }
]
```

可以看到 cell 的值是一个空数组。row 墓碑是指没有liveness_info和没有cell的行。deletion_info 字段存在于行级别

Range 删除

从单个分区中删除一个范围（即许多行）。

```
DELETE FROM tlp_lab.tombstones WHERE fruit='apple' AND date > '20160615';
```

apple 分区不再返回数据，因为它已经没有行了。除非我们有比20160616小的数据

```
echo "SELECT * FROM tlp_lab.tombstones LIMIT 100;" | cqlsh

fruit   | date     | crates
---------+----------+-----------
pickles | 20160616 | {6, 7, 8}

(1 rows)
```

刷新后，磁盘上多了一个SSTable，'mb-8-big'，内容如下。

```
alain$ SSTabledump /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/mb-8-big-Data.db
[
  {
    "partition" : {
      "key" : [ "apple" ],
      "position" : 0
    },
    "rows" : [
      {

        "type" : "range_tombstone_bound",
        "start" : {
          "type" : "exclusive",
          "clustering" : [ "20160615" ],
          "deletion_info" : { "marked_deleted" : "2016-06-16T19:53:21.133300Z", "local_delete_time" : "2016-06-16T19:53:21Z" }
        }
      },
      {
        "type" : "range_tombstone_bound",
        "end" : {
          "type" : "inclusive",
          "deletion_info" : { "marked_deleted" : "2016-06-16T19:53:21.133300Z", "local_delete_time" : "2016-06-16T19:53:21Z" }
        }
      }
    ]
  }
]
```

我们可以看到，我们现在有一个新的特殊插入，它的类型字段 type 不是 row，而是range_tombstone_bound。并且伴随 start 和 end 字段：clustering key 从20160615排除到无穷大。那些带有range_tombstone_bound类型的条目按照预期嵌套在 apple 对应的分区中。所以从磁盘空间的角度来看，删除整个范围是相当高效的，我们并不是每个单元格写一个信息，只是存储对应的删除边界。

Partition 删除

删除整个分区

```
DELETE FROM tlp_lab.tombstones WHERE fruit='pickles';
```

在删除后，分区和所有嵌套的行都不再显示，正如预期的那样。该表现在是空的。

```
alain$ echo "SELECT * FROM tlp_lab.tombstones LIMIT 100;" | cqlsh

 fruit | date | crates
-------+------+--------

(0 rows)
```

刷新后磁盘上多了一个SSTable，mb-9-big，内容如下。

```
alain$ SSTabledump /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/mb-9-big-Data.db
[
  {
    "partition" : {
      "key" : [ "pickles" ],
      "position" : 0,
      "deletion_info" : { "marked_deleted" : "2016-06-17T09:38:52.550841Z", "local_delete_time" : "2016-06-17T09:38:52Z" }
    }

  }
]
```

所以，我们又插入了一个特定的标记。Partition 类型的墓碑是指插入的分区有deletion_info，没有行。

注意：当使用集合类型时，每次使用整个集合进行复制操作的时候，range墓碑都会由INSERT和UPDATE操作生成，而并非更新原有集合。插入一个新集合，而不是追加或只更新原集合中的元素，也会导致先插入一个 range 墓碑 + 插入集合的新值。这样隐藏式进行的DELETE操作，往往导致一些奇怪的问题。

仔细看一下第一个SSTabledump的输出，就在上面的数据插入之后，在任何删除之前，你会发现一个墓碑已经存在了。

```
"cells" : [
  { "name" : "crates", "deletion_info" : { "marked_deleted" : "2016-06-16T18:52:41.900450Z", "local_delete_time" : "2016-06-16T18:52:41Z"
  { "name" : "crates", "path" : [ "1" ], "value" : "" },
  { "name" : "crates", "path" : [ "2" ], "value" : "" },
  { "name" : "crates", "path" : [ "3" ], "value" : "" },
  { "name" : "crates", "path" : [ "4" ], "value" : "" },
  { "name" : "crates", "path" : [ "5" ], "value" : "" }
]
```

从邮件列表中，我发现James Ravn用 list 举例讨论了这个话题，但对所有的集合类型都适用，这里不再展开更多的细节，我只是想指出这一点，因为乍看之下比较意外，见：http://www.jsravn.com/2015/05/13/cassandra-tombstones-collections.html#lists。

墓碑可能产生的问题

好了，现在我们明白了为什么我们要使用墓碑，并且对墓碑的内容有了大致的了解，让我们看看墓碑可能引起的潜在问题，以及我们可以采取哪些措施来缓解这些问题。

第一件显而易见的事情是，我们不仅没有删除数据，而是存储了更多的数据。在某些时候，我们需要删除这些墓碑，以释放一些磁盘空间，并限制不必要的数据读取量，改善延迟和资源利用率。正如我们很快就会看到的那样，这是通过 Compaction 的过程来实现的。

Compactions

当我们读取一条特定的行时，需要查阅的SSTables越多，读取速度就越慢。因此，为了保持较低的读取延迟，有必要通过 Compaction 的过程来合并 sstable 文件。同时因为我们要继续尽可能地释放磁盘空间，所以这一过程也包含删除符合条件的墓碑。

Compaction 的工作方式是合并来自多个SSTables的行片段，在条件满足的情况下删除墓碑。这些条件部分是在创建表的时候指定的，从而可以调整，比如gc_grace_seconds，部分条件由于Cassandra内部实现的原因，是硬编码的，以确保数据的持久性和一致性。确保所有数据片段所在的 sstable 都参与当前的 compaction 中（通常被称为 "重叠SSTables"）是必要的，以避免不一致，因为一旦墓碑被驱逐，这些数据就会重新出现，形成这种 "僵尸 "数据。

考虑到上面的例子。在所有的删除和刷新之后，表文件夹的样子是这样的。

```
alain$ ll /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/
total 360
drwxr-xr-x  43 alain  staff  1462 Jun 17 11:39 .
drwxr-xr-x   3 alain  staff   102 Jun 16 20:25 ..

drwxr-xr-x   2 alain  staff    68 Jun 16 17:05 backups
-rw-r--r--   1 alain  staff    43 Jun 17 11:13 mb-10-big-CompressionInfo.db
-rw-r--r--   1 alain  staff    43 Jun 17 11:13 mb-10-big-Data.db
-rw-r--r--   1 alain  staff    10 Jun 17 11:13 mb-10-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 17 11:13 mb-10-big-Filter.db
-rw-r--r--   1 alain  staff     9 Jun 17 11:13 mb-10-big-Index.db
-rw-r--r--   1 alain  staff  4701 Jun 17 11:13 mb-10-big-Statistics.db
-rw-r--r--   1 alain  staff    59 Jun 17 11:13 mb-10-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 17 11:13 mb-10-big-TOC.txt
-rw-r--r--   1 alain  staff    43 Jun 17 11:33 mb-11-big-CompressionInfo.db
-rw-r--r--   1 alain  staff    53 Jun 17 11:33 mb-11-big-Data.db
-rw-r--r--   1 alain  staff     9 Jun 17 11:33 mb-11-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 17 11:33 mb-11-big-Filter.db
-rw-r--r--   1 alain  staff     9 Jun 17 11:33 mb-11-big-Index.db
-rw-r--r--   1 alain  staff  4611 Jun 17 11:33 mb-11-big-Statistics.db
-rw-r--r--   1 alain  staff    59 Jun 17 11:33 mb-11-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 17 11:33 mb-11-big-TOC.txt
-rw-r--r--   1 alain  staff    43 Jun 17 11:33 mb-12-big-CompressionInfo.db
-rw-r--r--   1 alain  staff    42 Jun 17 11:33 mb-12-big-Data.db
-rw-r--r--   1 alain  staff    10 Jun 17 11:33 mb-12-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 17 11:33 mb-12-big-Filter.db
-rw-r--r--   1 alain  staff     9 Jun 17 11:33 mb-12-big-Index.db
-rw-r--r--   1 alain  staff  4611 Jun 17 11:33 mb-12-big-Statistics.db
-rw-r--r--   1 alain  staff    59 Jun 17 11:33 mb-12-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 17 11:33 mb-12-big-TOC.txt
-rw-r--r--   1 alain  staff    43 Jun 17 11:39 mb-13-big-CompressionInfo.db
-rw-r--r--   1 alain  staff    32 Jun 17 11:39 mb-13-big-Data.db
-rw-r--r--   1 alain  staff     9 Jun 17 11:39 mb-13-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 17 11:39 mb-13-big-Filter.db
```

```
-rw-r--r--   1 alain  staff    11 Jun 17 11:39 mb-13-big-Index.db
-rw-r--r--   1 alain  staff  4591 Jun 17 11:39 mb-13-big-Statistics.db
-rw-r--r--   1 alain  staff    65 Jun 17 11:39 mb-13-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 17 11:39 mb-13-big-TOC.txt
-rw-r--r--   1 alain  staff    43 Jun 17 11:12 mb-9-big-CompressionInfo.db
-rw-r--r--   1 alain  staff   127 Jun 17 11:12 mb-9-big-Data.db
-rw-r--r--   1 alain  staff    10 Jun 17 11:12 mb-9-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 17 11:12 mb-9-big-Filter.db
-rw-r--r--   1 alain  staff    20 Jun 17 11:12 mb-9-big-Index.db
-rw-r--r--   1 alain  staff  4740 Jun 17 11:12 mb-9-big-Statistics.db
-rw-r--r--   1 alain  staff    61 Jun 17 11:12 mb-9-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 17 11:12 mb-9-big-TOC.txt
```

你的SSTable很可能与上面的例子不一致，但是插入和删除的内容是完全一样的。我们可以看到这个表实际上是空的。存储在磁盘上的文件只包含墓碑和他们专门删除的条目。从读取的角度来看，没有任何结果返回。

```
echo "SELECT * FROM tlp_lab.tombstones LIMIT 100;" | cqlsh

fruit | date | crates
-------+------+--------

(0 rows)
```

此时，让我们触发一个 major compaction 来合并所有的SSTables。当 compaction 是自动触发的时候，通常会运行得更好。禁用自动 compaction 手动触发 major compaction 很少是一个好主意。这里是为了教学目的而做的。

```
nodetool -p 7100 compact
```

现在，所有的SSTable已经合并成一个SSTable

```
alain$ ll /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/
```

```
total 72
drwxr-xr-x  11 alain  staff   374 Jun 17 14:50 .
drwxr-xr-x   3 alain  staff   102 Jun 16 20:25 ..
drwxr-xr-x   2 alain  staff    68 Jun 16 17:05 backups
-rw-r--r--   1 alain  staff    51 Jun 17 14:50 mb-14-big-CompressionInfo.db
-rw-r--r--   1 alain  staff   105 Jun 17 14:50 mb-14-big-Data.db
-rw-r--r--   1 alain  staff    10 Jun 17 14:50 mb-14-big-Digest.crc32
-rw-r--r--   1 alain  staff    16 Jun 17 14:50 mb-14-big-Filter.db
-rw-r--r--   1 alain  staff    20 Jun 17 14:50 mb-14-big-Index.db
-rw-r--r--   1 alain  staff  4737 Jun 17 14:50 mb-14-big-Statistics.db

-rw-r--r--   1 alain  staff    61 Jun 17 14:50 mb-14-big-Summary.db
-rw-r--r--   1 alain  staff    92 Jun 17 14:50 mb-14-big-TOC.txt
```

这里是SSTable的内容，包含所有的墓碑，合并到同一个结构中，在同一个文件中。

```
alain$ SSTabledump /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/mb-14-big-Data.db
[
  {
    "partition" : {
      "key" : [ "apple" ],
      "position" : 0
    },
    "rows" : [
      {
        "type" : "range_tombstone_bound",
        "start" : {
          "type" : "exclusive",
          "clustering" : [ "20160615" ],
          "deletion_info" : { "marked_deleted" : "2016-06-17T09:14:11.697040Z", "local_delete_time" : "2016-06-17T09:14:11Z" }
        }
      },
      {
        "type" : "row",
```

```
        "position" : 40,
        "clustering" : [ "20160617" ],
        "deletion_info" : { "marked_deleted" : "2016-06-17T09:33:56.367859Z", "local_delete_time" : "2016-06-17T09:33:56Z" },
        "cells" : [ ]
      },
      {
        "type" : "range_tombstone_bound",
        "end" : {
          "type" : "inclusive",
          "deletion_info" : { "marked_deleted" : "2016-06-17T09:14:11.697040Z", "local_delete_time" : "2016-06-17T09:14:11Z" }
        }
      }
    ]
  },
  {
    "partition" : {
      "key" : [ "pickles" ],
      "position" : 73,
      "deletion_info" : { "marked_deleted" : "2016-06-17T09:38:52.550841Z", "local_delete_time" : "2016-06-17T09:38:52Z" }
    }
  }
]
```


请注意，此时被删除的数据在 compaction 过程中被直接删除。然而，正如前面所讨论的那样，我们仍然会在磁盘上存储一个墓碑标记，因为我们需要保留删除本身的记录，以便有效地将删除操作传达给集群的其他成员。我们不需要保留实际值，因为为了保持一致性，不需要这样做。

墓碑标记清除

Cassandra会在compaction触发时，只有在数据所属表上定义的local_delete_time + gc_grace_seconds之后，才会完全清理这些墓碑标记。请记住，所有的节点都应该在gc_grace_seconds内被修复，以确保墓碑标记的正确分布，并防止被删除的数据再次出现，如上所述。

这个gc_grace_seconds参数是数据被删除后，墓碑在磁盘上保留的最小时间。我们需要确保所有的副本也收到了删除，并且有墓碑存储，避免出现一些僵尸数据的问题。我们唯一的办法就是全面修复。在gc_grace_seconds之后，墓碑最终会被驱逐，如果有节点错过了墓碑，我们就会出现上面所说的情况，数据会重新出现。TTL不受此影

响，因为没有一个节点可以拥有数据而错过相关的TTL，它是原子的，是同一个记录。任何拥有数据的节点也会知道数据什么时候要被删除。

另外，为了从磁盘中删除数据和墓碑，Cassandra代码还需要遵循其他安全规则。我们需要一行或一个分区的所有碎片都在同一个 Compaction 中，墓碑才能被删除。比如一个 Compaction 处理文件1到4，如果一些数据在文件5上，墓碑不会被驱逐，因为我们仍然需要它将SSTable 5上的数据标记为被删除，否则SSTable 5上的数据会回来（僵尸）。

这些条件有时会使删除墓碑成为一件非常复杂的事情。它常常给Cassandra用户带来很多麻烦。墓碑不被移除可能意味着使用了大量的磁盘空间，读取速度较慢，维修工作较多，GC压力较大，需要更多的资源等等。当你的一张表的大部分SSTables的墓碑比例很高的时候（90%的数据都是墓碑），就很难读到一个合适的值，一个相关的数据，存储成本就会比较高。这类问题甚至会导致磁盘空间的耗尽。

很多用法都会导致数据删除（TTL或deletes），这是我们作为Cassandra运维人员需要关注和处理的。

最后一次回到我们的例子。几天后我重启了节点（几天>10天，默认的gc_grace_seconds）。Cassandra打开了我们在mb-14-big上面建立的压缩SSTable，它马上就被压缩了。

```
MacBook-Pro:tombstones alain$ grep 'mb-14-big' /Users/alain/.ccm/Cassa-3.7/node1/logs/system.log

DEBUG [SSTableBatchOpen:1] 2016-06-28 15:56:17,947 SSTableReader.java:482 - Opening /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombsto
DEBUG [CompactionExecutor:2] 2016-06-28 15:56:18,525 CompactionTask.java:150 - Compacting (166f61c0-3d38-11e6-bfe3-e9e451310a18) [/Users/a
```

此时由于gc_grace_seconds已经过了，墓碑是符合清理条件的。所以所有的墓碑都被删除了，由于这个表里已经没有数据了，所以数据文件夹现在终于空了。

```
MacBook-Pro:tombstones alain$ ll /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/
total 0
drwxr-xr-x  3 alain  staff  102 Jun 28 15:56 .
drwxr-xr-x  3 alain  staff  102 Jun 16 20:25 ..
MacBook-Pro:tombstones alain$
```

如果墓碑在所有的副本上都能正确复制，我们就会有一个完全一致的删除，不会再出现数据。这时我们还可以多释放一些磁盘空间，让我们更容易读取其他值，即使我的例子对于演示这个目的有点傻，因为现在表已经完全空了。

监控墓碑占比和过期

由于Cassandra设计的原因，在删除数据或使用TTL时出现墓碑是正常的，然而这是我们需要控制的。

要想知道一个表的墓碑率，或者想知道任何SSTable上墓碑清理时间（即删除时间戳）的估计分布，可以使用SSTablemetadata。

```
alain$ SSTablemetadata /Users/alain/.ccm/Cassa-3.7/node1/data/tlp_lab/tombstones-c379952033d311e6aa4261d6a7221ccb/mb-14-big-Data.db

--
Estimated droppable tombstones: 2.0
--
Estimated tombstone drop times:
1466154851:         2
1466156036:         1
1466156332:         1
--
```

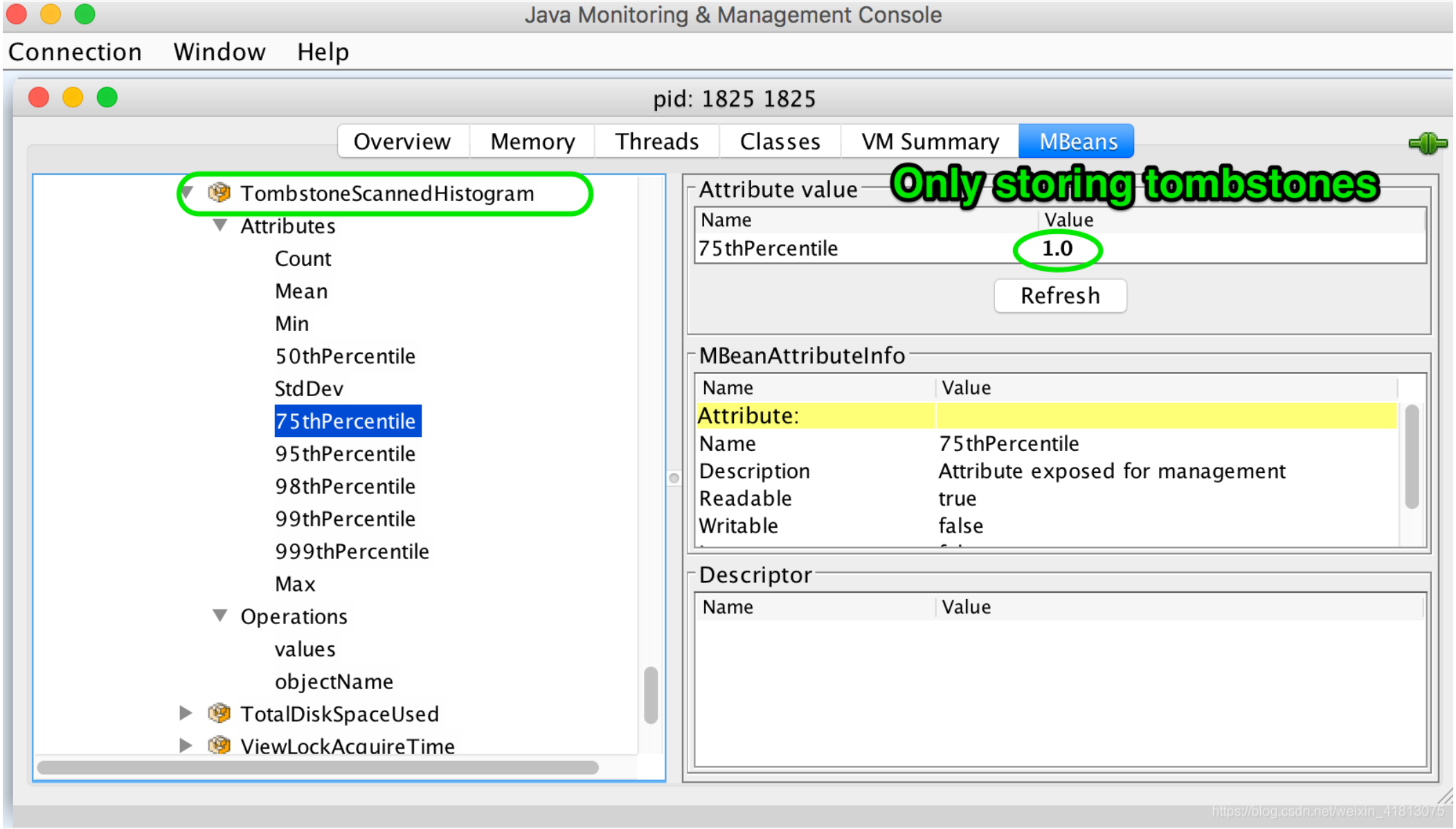
上面展示的比例可能是错误的，因为在这个mb-14-big文件中，我有0个有用的行，只有墓碑的特定状态，但它通常是一个关于墓碑的SSTable状态的好指标。

另外，Cassandra通过JMX为所有表暴露了这个名为TombstoneScannedHistogram的监控指标。scope=tombstones 这样就是指定表：tombstones

```
org.apache.cassandra.metrics:type=Table,keyspace=tlp_lab,scope=tombstones,name=TombstoneScannedHistogram
```

这是值得纳入进监控工具的一个指标，如Graphite / Grafana，Datadog，New Relic等。

下面是我上面的例子在 Compaction 清理墓碑之前的jconsole输出。



单文件 compaction

在Jonathan Ellis在CASSANDRA-3442中报告了以下内容后，并在Cassandra 1.2中引入了单文件SSTable compaction。

> 在 sized tired compaction 的情况下，往往出现不经常执行 compaction 的大型SSTable文件。在这种情况下，如果混入了过期的数据，我们可能会浪费很多空间。

如上所述，compaction 负责墓碑清理工作。在某些情况下，compaction 无法做好墓碑的清理工作。不仅上述提到的 STCS 策略，其实目前所有的 compaction 策略都是如此。有些SSTables的 compaction 频率可能很低，或者有重叠的SSTables的时间很长。这就是为什么，现在所有的 compaction 策略都带有一套参数来帮助墓碑清理。

tombstone_threshold。这个参数的行为和Jonathan Ellis 在 2011 年所提交的 ticket 的描述的一模一样。

> 如果我们在SSTable元数据中保留一个TTL EstimatedHistogram，我们就可以对过期数据超过20%的SSTable进行单SSTable压实。

所以当认为可清理的墓石比例高于X（X=0.2，默认为20%）时，这个选项会触发一次单文件 compaction。因为计算出来的墓石比例没有考虑gc_grace_seconds所以往往实际能清理的墓碑会小于估计值。

tombstone_compaction_interval.这个选项在CASSANDRA-4781中被引入，当墓碑比例高到足以触发单文件 compaction 时，可能由于重叠 sstable 的因素导致墓碑实际上无法被清理，进而引起无限循环，这个选项就是为了解决这个问题引入的。我们必须确保删除所有的数据碎片以避免僵尸。在这种情况下，一些SSTable上会持续不断的进行 compaction。因为单文件的 compaction 触发条件仅仅基于墓碑率，而墓碑率是一个估计值。这个选项可以调整2个单文件 compaction 之间的最小间隔，默认是1天。

unchecked_tombstone_compaction。由Paulo Motta在CASSANDRA -6563中引入。这里是他对单SSTable历史的介绍，以及他引入这个参数的原因，非常有趣，我也无法解释得更好。

但是要注意 trade-off：把这个选项设置为true，只要墓碑比（估计）高于0.2（20%的数据是墓碑，墓碑_阈值默认），就会每天触发单文件 compaction（默认间隔：tombstone_compaction_interval），即使实际上没有一个墓碑是可以清理的。这将是最坏的情况。

所以，trade-off 是消耗更多的资源，期望能更好的完成墓碑清理。

建议：在一些数据中心遇到墓碑清理麻烦的时候，尽快给这个选项一个尝试应该是值得的。我有一些使用这个选项的成功经验，没有真正的坏经验，而是有一些情况下，这个改变没有真正的影响。我甚至在一些磁盘使用率达到100%的节点上将这个选项设置为true并且手动 compaction 正确的SSTables后，观测到磁盘使用率下降到一个合理的水位

要改变这些参数中的任何一个，先对你想要改的表使用 describe 命令，然后填写完整的 compaction 参数以避免任何问题。要改变tlp_lab键空间中表墓碑的 compaction 选项，我会这样做。

```
MacBook-Pro:~ alain$ echo "DESCRIBE TABLE tlp_lab.tombstones;" | cqlsh
CREATE TABLE tlp_lab.tombstones (
fruit text,
date text,
crates set<int>,
PRIMARY KEY (fruit, date)
) WITH CLUSTERING ORDER BY (date ASC)
AND bloom_filter_fp_chance = 0.01
AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
AND comment = ''
AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
AND crc_check_chance = 1.0
AND dclocal_read_repair_chance = 0.1
AND default_time_to_live = 0
AND gc_grace_seconds = 864000
AND max_index_interval = 2048
AND memtable_flush_period_in_ms = 0
AND min_index_interval = 128
AND read_repair_chance = 0.0
AND speculative_retry = '99PERCENTILE';
```

然后，我复制 compaction 参数，并修改表格如下。

```
echo "ALTER TABLE tlp_lab.tombstones WITH compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_t
```

或者把以下内容写入脚本文件中执行。

```
ALTER TABLE tlp_lab.tombstones WITH compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_thresho
```

```
cqlsh -f myfile.cql
```

也可以使用-e选项

```
cqlsh -e "ALTER TABLE tlp_lab.tombstones WITH compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'm
```

注意：我使用其中一个选项，而不是进入cqlsh控制台，因为它更容易和 pipe 相结合。在选择数据或描述表时，它更有意义。想象一下，你需要检查你所有表的read_repair_chance情况

```
MacBook-Pro:~ alain$ echo "DESCRIBE TABLE tlp_lab.tombstones;" | cqlsh | grep -e TABLE -e read_repair_chance
    CREATE TABLE tlp_lab.tombstones (
        AND dclocal_read_repair_chance = 0.1
        AND read_repair_chance = 0.0
    CREATE TABLE tlp_lab.foo (
        AND dclocal_read_repair_chance = 0.0
        AND read_repair_chance = 0.1
    CREATE TABLE tlp_lab.bar (
        AND dclocal_read_repair_chance = 0.0
        AND read_repair_chance = 0.0
```

手动清理墓碑

有时候一些SSTables包含95%的墓碑，但由于 compaction 参数设置、SSTables重叠，或者仅仅是单SSTable compaction 的优先级低于常规 compaction，从而一直未能执行 compaction 操作。要知道，我们可以手动强制Cassandra运行用户定义的 compaction，这一点很重要。要做到这一点，我们需要能够通过JMX发送命令。我这里会考虑使用jmxterm。你可以选择自己喜欢的工具。

下载 jmxterm。

```
wget http://sourceforge.net/projects/cyclops-group/files/jmxterm/1.0-alpha-4/jmxterm-1.0-alpha-4-uber.jar
```

然后运行JMX命令，比如强制 compaction

```
echo "run -b org.apache.cassandra.db:type=CompactionManager forceUserDefinedCompaction myks-mytable-marker-sstablenumber-Data.db" | java -
```

在某些情况下，我使用了基于这种命令的脚本，结合sstablemetadata工具给出的墓碑比例来搜索最差的文件，并将它们压缩，效果非常成功。

总结

针对分布式系统执行删除一直是一个棘手的操作，特别是当试图同时兼顾可用性、一致性和持久性时。

在像Apache Cassandra这样的分布式系统中，使用墓碑是一种执行删除的明智方式，但它也有一些注意事项。我们需要用一种非直观的方式来思考，因为在删除的时候，添加这块数据并不是一件自然的事情。然后了解墓碑的生命周期，这也不是小事。然而，当我们了解了墓碑的行为，并使用相应的工具来帮助我们解决墓碑问题时，墓碑也就变得容易理解了。

由于Cassandra是一个快速发展的系统，这里有一些正在进行的关于墓碑的提议，你可能有兴趣了解一下。

进行中提议：。

CASSANDRA-7019：改进墓碑 compaction（主要是解决SSTable墓碑重叠的问题，通过让单SSTable compaction 实际运行在多个智能选择的SSTables上）。

CASSANDRA-8527：凡是我们统计墓碑的地方都要考虑 range 类型的墓碑。看起来 range 类型的墓碑可能在代码的多个部分没有被很好地考虑。CASSANDRA-11166 和 CASSANDRA-9617 也指出了这个问题。