Elasticsearch 分片交互过程分析

https://blog.csdn.net/u010585120/article/details/48342895

*摘要: 文章主要来源于官方文档的叙述，同时结合我自己的理解整合而成。叙述了Elasticsearch primary shard 与 replica shard 在工作过程中数据交互的全过程。*

# 一、Elasticseach如何将数据存储到分片中

问题：当我们要在ES中存储数据的时候，数据应该存储在主分片和复制分片中的哪一个中去；当我们在ES中检索数据的时候，又是怎么判断要查询的数据是属于哪一个分片。

数据存储到分片的过程是一定规则的，并不是随机发生的。

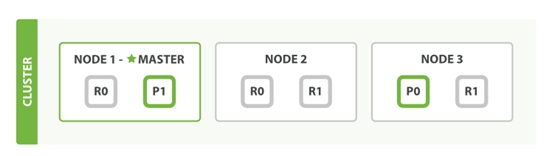
规则：shard = hash(routing) % number\_of\_primary\_shards

Routing值可以是一个任意的字符串，默认情况下，它的值为存数数据对应文档 \_id 值，也可以是用户自定义的值。Routing这个字符串通过一个hash的函数处理，并返回一个数值，然后再除以索引中主分片的数目，所得的余数作为主分片的编号，取值一般在0到number\_of\_primary\_shards - 1的这个范围中。通过这种方法计算出该数据是存储到哪个分片中。

正是这种路由机制，导致了主分片的个数为什么在索引建立之后不能修改。对已有索引主分片数目的修改直接会导致路由规则出现严重问题，部分数据将无法被检索。

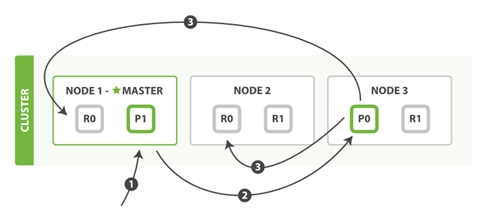
# 二、主分片与复制分片如何交互

为了说明这个问题，我用一个例子来说明。



在上面这个例子中，有三个ES的node，其中每一个index中包含两个primary shard，每个primary shard拥有一个replica shard。下面从几种常见的数据操作来说明二者之间的交互情况。

## 1、索引与删除一个文档



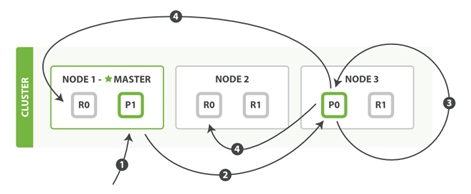
这两种过程均可以分为三个过程来描述：

阶段1：客户端发送了一个索引或者删除的请求给node 1。

阶段2：node 1通过请求中文档的 \_id 值判断出该文档应该被存储在shard 0 这个分片中，并且node 1知道shard 0的primary shard位于node 3这个节点上。因此node 1会把这个请求转发到node 3。

阶段3：node 3在shard 0 的primary shard上执行请求。如果请求执行成功，它node 3将并行地将该请求发给shard 0的其余所有replica shard上，也就是存在于node 1和node 2中的replica shard。如果所有的replica shard都成功地执行了请求，那么将会向node 3回复一个成功确认，当node 3收到了所有replica shard的确认信息后，则最后向用户返回一个Success的消息。

## 2、更新一个文档



该过程可以分为四个阶段来描述：

阶段1：客户端向node 1发送一个文档更新的请求。

阶段2：同样的node 1通过请求中文档的 \_id 值判断出该文档应该被存储在shard 0 这个分片中，并且node 1知道shard 0的primary shard位于node 3这个节点上。因此node 1会把这个请求转发到node 3。

阶段3：node 3从文档所在的primary shard中获取到它的JSON文件，并修改其中的\_source中的内容，之后再重新索引该文档到其primary shard中。

阶段4：如果node 3成功地更新了文档，node 3将会把文档新的版本并行地发给其余所有的replica shard所在node中。这些node也同样重新索引新版本的文档，执行后则向node 3确认成功，当node 3接收到所有的成功确认之后，再向客户端发送一个更新成功的信息。

## 3、检索文档

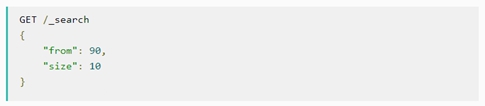
CRUD这些操作的过程中一般都是结合一些唯一的标记例如：\_index，\_type，以及routing的值，这就意味在执行操作的时候都是确切的知道文档在集群中的哪个node中，哪个shard中。

而检索过程往往需要更多的执行模式，因为我们并不清楚所要检索的文档具体位置所在， 它们可能存在于ES集群中个任何位置。因此，一般情况下，检索的执行不得不去询问index中的每一个shard。

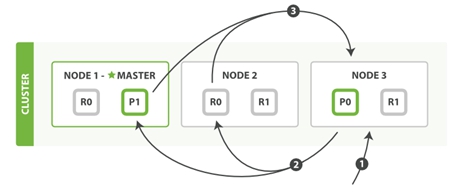
但是，找到所有匹配检索的文档仅仅只是检索过程的一半，在向客户端返回一个结果列表之前，必须将各个shard发回的小片的检索结果，拼接成一个大的已排好序的汇总结果列表。正因为这个原因，检索的过程将分为查询阶段与获取阶段（Query Phase and Fetch Phase）。

Query Phase

在最初的查询过程中，查询请求会广播到index中的每一个primary shard和replica shard中，每一个shard会在本地执行检索，并建立一个优先级队列（priority queue）。这个优先级队列是一个根据文档匹配度这个指标所排序列表，列表的长度由分页参数from和size两个参数所决定。例如：



下面从一个例子中说明这个过程：



Query Phase阶段可以再细分成3个小的子阶段：

子阶段1：客户端发送一个检索的请求给node 3，此时node 3会创建一个空的优先级队列并且配置好分页参数from与size。

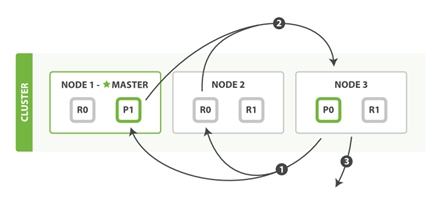
子阶段2：node 3将检索请求发送给该index中个每一个shard（这里的每一个意思是无论它是primary还是replica，它们的组合可以构成一个完整的index数据）。每个shard在本地执行检索，并将结果添加到本地优先级队列中。

子阶段3：每个shard返回本地优先级序列中所记录的\_id与sort值，并发送node 3。Node 3将这些值合并到自己的本地的优先级队列中，并做全局的排序。

Fetch Phase

Query Phase主要定位了所要检索数据的具体位置，但是我们还必须取回它们才能完成整个检索过程。而Fetch Phase阶段的任务就是将这些定位好的数据内容取回并返回给客户端。

同样也用一个例子来说明这个过程：



Fetch Phase过程可以分为三个子过程来描述：

子阶段1：node 3获取了所有待检索数据的定位之后，发送一个mget的请求给与数据相关的shard。

子阶段2：每个收到node 3的get请求的shard将读取相关文档\_source中的内容，并将它们返回给node 3。

子阶段3：当node 3获取到了所有shard返回的文档后，node 3将它们合并成一条汇总的结果，返回给客户端。