行分布式　索

在　之前，我　将　道　一下在分布式　境中搜索是　行的。　比我　在　[distributed-docs] 章　的基本的　-　-改-　(CRUD)　求要　一些。

> **内容提示**
>
> 可以根据　趣　本章内容。　并不需要　了使用 Elasticsearch 而理解和　住所有的　。
>
> 章的　目的只　初　了解下工作原理，以便将来需要　可以及　到　些知　，　但是不要被　所困　。

一个 CRUD 操作只　个文　行　理，文　的唯一性由 `_index`, `_type`, 和 `routing values` （通常　是　文　的 `_id` ）的　合来　定。　表示我　切的知道集群中　个分片含有此文　。

搜索需要一　更加　的　行模型因　我　不知道　会命中　些文　：　些文　有可能在集群的任何分片上。　一个搜索　求必　我　注的索引（index　or indices）的所有分片的某个副本来　定它　是否含有任何匹配的文　。

但是　到所有的匹配文　完成事情的一半。　在 `search` 接口返回一个 `page`　果之前，多分片中的果必　合成　个排序列表。　此，搜索被　行成一个　段　程，我　称之　*query then fetch* 。

段

在初始　段　，　会广播到索引中　一个分片拷　（主分片或者副本分片）。　个分片在本地　行搜索并　建一个匹配文　的　先　列。

> **先　列**
>
> 一个　先　列　是一个存有 *top-n* 匹配文　的有序列表。　先　列的大小取决于分　参数 `from` 和 `size` 。例如，如下搜索　求将需要足　大的　先　列来放入100条文　。
>
> ```
> GET /_search
> {
>     "from": 90,
>     "size": 10
> }
> ```

个　段的　程如　程分布式搜索 所示。

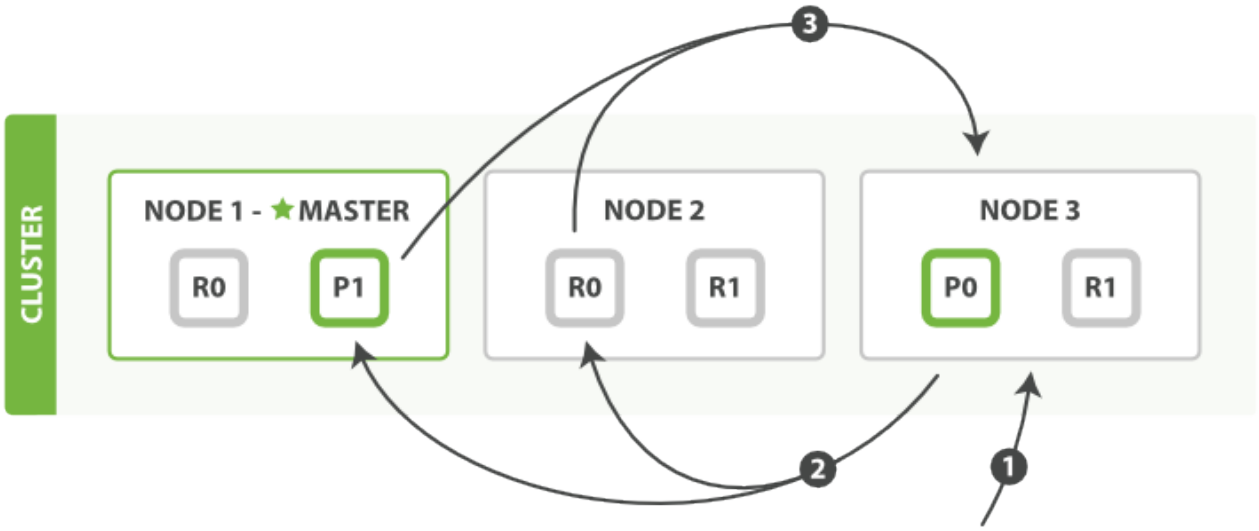

Figure 1. 程分布式搜索

段包含以下三个 :

1. 客 端 送一个 search 求到 Node 3， Node 3 会 建一个大小 from + size 的空 先 列。
2. Node 3 将 求 到索引的 个主分片或副本分片中。 个分片在本地 行 并添加 果到大小 from + size 的本地有序 先 列中。
3. 个分片返回各自 先 列中所有文 的 ID 和排序 点，也就是 Node 3 ，它合并 些 到自己的 先 列中来 生一个全局排序后的 果列表。

当一个搜索 求被 送到某个 点 ， 个 点就 成了 点。 个 点的任 是广播 求到所有相 分片并将它 的 整合成全局排序后的 果集合， 个 果集合会返回 客 端。

第一 是广播 求到索引中 一个 点的分片拷 。就像 document GET requests 所描述的， 求可以被某个主分片或某个副本分片 理， 就是 什 更多的副本（当 合更多的硬件）能 加搜索 吐率。 点将在之后的 求中 所有的分片拷 来分 。

个分片在本地 行 求并且 建一个 度 <code>from + size</code> 的 先 列— 也就是 ， 个分片 建的 果集足 大，均可以 足全局的搜索 求。 分片返回一个 量 的 果列表到 点，它 包含文 ID 集合以及任何排序需要用到的 ，例如 <code>_score</code> 。

点将 些分片 的 果合并到自己的有序 先 列里，它代表了全局排序 果集合。至此 程 束。

NOTE

一个索引可以由一个或几个主分片 成， 所以一个 个索引的搜索 求需要能 把来自多个分片的 果 合起来。 *multiple* 或者 *all* 索引的搜索工作方式也是完全一致的— 是包含了更多的分片而已。

取回 段

段 些文 足搜索 求，但是我 然需要取回 些文 。 是取回 段的任 ， 正如分布式搜索的取回 段 所展示的。

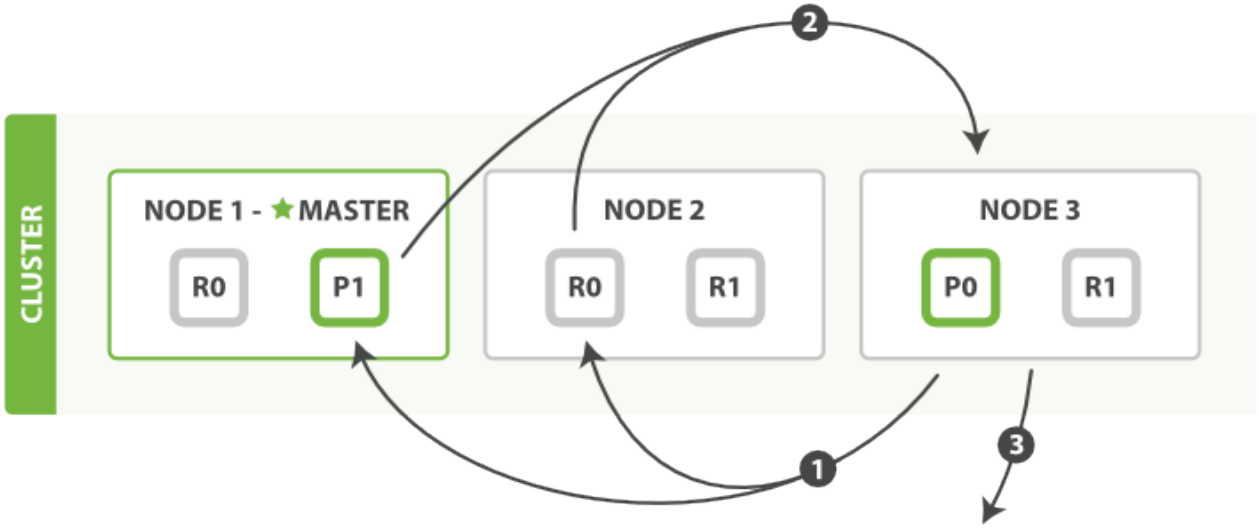

Figure 2. 分布式搜索的取回　段

分布式　段由以下　　成：

1. 　　点　　出　些文　需要被取回并向相　的分片提交多个 GET　求。
2. 　个分片加　并　富 文　，如果有需要的　，接着返回文　　点。
3. 一旦所有的文　都被取回了，　　点返回　果　客　端。

　　点首先决定　些文　　需要被取回。例如，如果我　的　　指定了 `{ "from": 90, "size": 10 }`，最初的90个　果会被　　，只有从第91个　始的10个　果需要被取回。　些文　可能来自和最初搜索　求有　的一个、多个甚至全部分片。

　　点　持有相　文　的　个分片　建一个　multi-get　request　，并　送　求　同　理段的分片副本。

分片加　文　体-- `_source` 字段—如果有需要，用元数据和 search snippet highlighting　富　果文　。一旦　　点接收到所有的　果文　，它就　装　些　果　　个　　返回　客　端。

深分 （Deep Pagination）

先 后取的 程支持用 from 和 size 参数分 ，但是 是 有限制的 。 要 住需要 信息 点的 个分片必 先 建一个 from + size 度的 列， 点需要根据 number_of_shards * (from + size) 排序文 ，来 到被包含在 size 里的文 。

取决于 的文 的大小，分片的数量和 使用的硬件， 10,000 到 50,000 的 果文 深分 （1,000 到 5,000 ）是完全可行的。但是使用足 大的 from ，排序 程可能会 得非常 重，使用大量的CPU、内存和 。因 个原因，我 烈建 不要使用深分 。

上， ``深分 '' 很少符合人的行 。当2到3 去以后，人会停止翻 ，并且改 搜索 准。会不知疲倦地一 一 的 取 直到 的服 崩 的罪魁 首一般是机器人或者web spider。

如果 需要从 的集群取回大量的文 ， 可以通 用 scroll 禁用排序使 个取回行 更有效率，我 会在 later in this chapter 行 。

搜索

有几个 参数可以影 搜索 程。

偏好

偏好 个参数 preference 允 用来控制由 些分片或 点来 理搜索 求。 它接受像 _primary, _primary_first, _local, _only_node:xyz, _prefer_node:xyz, 和 _shards:2,3 的 ， 些 在 {ref}/search-request-preference.html[search preference] 文 面被 解 。

但是最有用的 是某些随机字符串，它可以避免 *bouncing results* 。

Bouncing Results

想象一下有 个文 有同 的 字段，搜索 果用 timestamp 字段来排序。 由于搜索 求是在所有有效的分片副本 的，那就有可能 生主分片 理 求 ， 个文 是一 序 ， 而副本分片 理 求 又是 一 序。

就是所 的 *bouncing results* ： 次用 刷新 面，搜索 果表 是不同的 序。同一个用 始 使用同一个分片， 可以避免 ， 可以 置 preference 参数 一个特定的任意 比如用 会 ID来解决。

超

通常分片 理完它所有的数据后再把 果返回 同 点， 同 点把收到的所有 果合并 最 果。

意味着花 的 是最慢分片的 理 加 果合并的 。如果有一个 点有 ，就会 致所有的 慢。

参数 timeout 告 分片允 理数据的最大 。如果没有足 的 理所有数据， 个分片的

果可以是部分的，甚至是空数据。

搜索的返回 果会用属性 `timed_out` 明分片是否返回的是部分 果：

```
...
"timed_out":     true,  ①
...
```

① 个搜索 求超 了。

WARNING

超 然是一个最有效的操作，知道 一点很重要； 很可能 会超 定的超 。 行 有 个原因：

1. 超 是基于 文 做的。 但是某些 型有大量的工作在文 估之前需要完成。 "setup" 段并不考 超 置，所以太 的建立 会 致超 超 的整体延 。
2. 因 是基于 个文 的，一次 在 个文 上 行并且在下个文 被 估之前不会超 。 也意味着差的脚本（比如 无限循 的脚本）将会永 行下去。

路由

在 [routing-value] 中， 我 解 如何定制参数 `routing` ，它能 在索引 提供来 保相 的文 ，比如属于某个用 的文 被存 在某个分片上。 在搜索的 候，不用搜索索引的所有分片，而是通 指定几个 `routing` 来限定只搜索几个相 的分片：

```
GET /_search?routing=user_1,user2
```

个技 在 大 模搜索系 就会派上用 ，我 在 [scale] 中 它。

搜索 型

省的搜索 型是 `query_then_fetch` 。 在某些情况下， 可能想明 置 `search_type` `dfs_query_then_fetch` 来改善相 性精 度：

```
GET /_search?search_type=dfs_query_then_fetch
```

搜索 型 `dfs_query_then_fetch` 有 段， 个 段可以从所有相 分片 取 来 算全局 。 我 在 [relevance-is-broken] 会再 它。

游 'Scroll'

`scroll` 可以用来 Elasticsearch 有效地 行大批量的文 ，而又不用付出深度分 那 代 。

游 允 我 先做 初始化，然后再批量地拉取 果。 有点儿像 数据 中的 *cursor* 。

游　　会取某个　　点的快照数据。　　初始化之后索引上的任何　化会被它忽略。　　它通　保存旧的数据文件来　　个特性，　果就像保留初始化　的索引'　'一　。

深度分　的代　根源是　果集全局排序，如果去掉全局排序的特性的　　果的成本就会很低。　　游　　用字段 _doc 来排序。　个指令　Elasticsearch　　从　有　果的分片返回下一批　果。

　用游　　可以通　在　　的　候　置参数 scroll 的　　我　期望的游　　的　期　　。　游　的　期　　会在　次做　　的　候刷新，所以　个　　只需要足　　理当前批的　果就可以了，而不是　理　　果的所有文　的所需　　。　　个　期　　的参数很重要，因　保持　个游　　口需要消耗　源，所以我　期望如果不再需要　　源就　早点儿　放掉。　　置　个超　能　　Elasticsearch 在　后空　的　候自　　放　部分　源。

```
GET /old_index/_search?scroll=1m ①
{
    "query": { "match_all": {}},
    "sort" : ["_doc"], ②
    "size":  1000
}
```

① 保持游　　口一分　。

②　　字 _doc 是最有效的排序　序。

　个　　的返回　果包括一个字段 _scroll_id，　它是一个base64　　的　字符串 。　　在我　能　　字段 _scroll_id 到 _search/scroll　　接口　取下一批　果：

```
GET /_search/scroll
{
    "scroll": "1m", ①
    "scroll_id" :
"cXVlcnlUaGVuRmV0Y2g7NTsxMDk5NDpkUmpiR2FjOFNhNnlCM1ZDMWpWYnRROzEwOTk1OmRSamJHYWM4U2E2e
UIzVkMxalZidFE7MTA5OTM6ZFJqYkdhYzhTYTZ5QjNWQzFqVmJ0UTsxMTE5MDpBVUtwN2lxc1FLZV8yRGVjWlI
2QUVBOzEwOTk2OmRSamJHYWM4U2E2eUIzVkMxalZidFE7MDs="
}
```

① 注意再次　置游　　期　　一分　。

　个游　　返回的下一批　果。　　尽管我　指定字段　size　的　　1000，我　有可能取到超　　个　数量的文　。当　　的　候，字段 size 作用于　个分片，所以　个批次　　返回的文　数量最大　size * number_of_primary_shards 。

NOTE	注意游　　次返回一个新字段　_scroll_id。　次我　做下一次游　　，　我　必　把前一次　返回的字段 _scroll_id　　去。　当没有更多的　果返回的　候，我　就　理完所有匹配的文　了。

TIP	提示：某些官方的 Elasticsearch　客　端比如 Python　客　端　和 Perl　客　端　提供了　个功能易用的封装。