教程系列：如何最大化 Elasticsearch 索引性能

[**如何最大化 Elasticsearch 索引性能（第 1 部分）**](https://qbox.io/blog/maximize-guide-elasticsearch-indexing-performance-part-1/)

[亚当范德布什](https://qbox.io/blog/author/adam)发表于 2017 年 3 月 30 日

**这篇文章是关于调整Elasticsearch Indexing**的 3 部分系列文章的第 1 部分。本系列特别关注调整 Elasticsearch 以实现最大索引吞吐量并减少监控和管理负载。

作为起点，假设您启动了 Elasticsearch，创建了一个索引，并将 JSON 文档提供给它，而不包含模式。然后，Elasticsearch 将遍历 JSON 文档的每个索引字段，估计其字段，并创建相应的映射。虽然这看起来很理想，但 Elasticsearch 映射并不总是准确的。例如，如果选择了错误的字段类型，则会发生索引错误。

对于这篇文章，我们将使用 Qbox.io 上托管的 Elasticsearch。您可以 [***在此处注册或启动您的集群***](https://qbox.io/signup?utm_source=blog&utm_campaign=tutorial&utm_term=launch_your_cluster&utm_medium=article)，或单击标题导航中的“开始使用”。如果您在设置方面需要帮助，请参阅“[***配置 Qbox Elasticsearch 集群。***](https://qbox.io/blog/provisioning-a-qbox-elasticsearch-cluster?utm_source=tutorial&utm_term=provision&utm_medium=article&utm_campaign=index-attachments-files-elasticsearch-mapper)“

没有二级索引的数据存储是真正无模式的，Elasticsearch 也不例外。虽然它不需要您预先定义数据结构，但它确实从您的文档结构中派生出一个模式，并根据该结构决定如何索引它们。在这方面，Elasticsearch 使用隐式架构索引您的数据，而不是显式架构。

因此，索引决策非常重要，它们对您如何搜索数据有很大影响。如果它是一个字符串字段，是否应该对其进行标记和规范化？如果是这样，怎么做？如果是数字字段，需要什么精度？还有更多字段类型，如日期时间字段、地理空间形状和需要特别注意的父/子关系。

不要分析、存储甚至向 Elasticsearch 发送您不需要回答搜索请求的数据。特别是，仔细检查您自己没有定义的映射的内容（例如，因为像 Logstash 这样的工具会为您生成它们）。我们是否计划在 Elasticsearch 中索引大量数据？或者我们已经在尝试这样做，但事实证明吞吐量太低了？如果您的搜索要求允许，我们的索引的映射定义中有许多优化技术。本教程将列出一系列使用 Elasticsearch 提高索引吞吐量的技巧和想法。

### 自定义字段映射

对于分析的字段，使用满足字段要求的最简单的分析器。或者也许你甚至可以选择**not\_analyzed**？

默认情况下，字符串类型的字段被认为包含全文。也就是说，它们的值将在被索引之前通过分析器，并且字段上的全文查询将在搜索之前将查询字符串通过分析器。

字符串字段的两个最重要的映射属性是索引和分析器。

* **index：**索引属性控制字符串的索引方式。它可以包含三个值之一。
* **分析：**首先分析字符串，然后对其进行索引。换句话说，将此字段索引为全文。
* **not\_analyzed：**索引此字段以便它可搜索，但索引值完全按照指定。不要分析它。
* **no：**根本不索引这个字段。该字段将不可搜索。

**分析**字符串字段的默认索引值。如果我们想将该字段映射为精确值，我们需要将其设置为**not\_analyzed**：

{

“标签”：{

“类型”： “字符串”，

“索引”： “未分析”

}

}

其他简单类型（例如 long、double、date 等）也接受 index 参数，但唯一相关的值是 no 和 not\_analyzed，因为它们的值从未被分析过。

**分析器：**对于分析的字符串字段，使用分析器属性指定在搜索时和索引时应用哪个分析器。默认情况下，Elasticsearch 使用**标准**分析器，但您可以通过指定内置分析器之一来更改它，例如**whitespace**、**simple**或**english**。

{

“推文”：{

“类型”： “字符串”，

“分析器”：“英语”

}

}

### 禁用 \_source 字段

该\_source字段包含在索引时传递的原始 JSON 文档正文。该\_source字段本身没有被索引（因此不可搜索），但它被存储以便在执行获取或搜索等获取请求时可以返回。尽管使用起来非常方便，但源字段确实会在索引中产生存储开销。因此，可以按如下方式禁用它：

curl -XPUT 'http: //localhost:9200/index\_name/' -d '{

"mappings" : {

"tweet" : {

"\_source" : {

"enabled" : false

}

}

}

}'

用户经常在不考虑后果的情况下禁用该\_source字段，然后后悔。如果该\_source字段不可用，则不支持许多功能：

* [**update**](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-update.html)、update\_by\_query**[和](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-update-by-query.html)**[**reindex**](https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-reindex.html) API 。
* 在飞行[**中突出显示**](https://www.elastic.co/guide/en/elasticsearch/reference/current/search-request-highlighting.html)。
* 能够从一个 Elasticsearch 索引重新索引到另一个，以更改映射或分析，或者将索引升级到新的主要版本。
* 通过查看索引时使用的原始文档来调试查询或聚合的能力。
* 自动修复索引损坏的能力..

### 在 \_source 中包含和排除字段

如果您正在使用该字段，则将任何其他字段设置为**\_stored**\_source没有额外的价值。如果您不使用该字段，请将您必须的字段设置为 \_stored。但是请注意，使用会带来某些优势，例如使用更新 API 的能力。\_source\_source

**案例研究：  [Qbox 如何使用 Supergiant 每月节省 5 个数字](https://supergiant.io/about" \t "_blank)**

仅专家功能是在\_source文档被索引之后但在\_source存储字段之前修剪字段内容的能力。从 中删除字段与\_source禁用 具有类似的缺点\_source，尤其是您无法将文档从一个 Elasticsearch 索引重新索引到另一个这一事实。

也接受通配符的包含/排除参数可以按如下方式使用：

curl -XPUT 'http: // localhost: 9200 /logs' -d '{

"mappings" : {

"event" : {

"\_source" : {

"includes" : [

"\*.count" ,

"meta.\*"

],

“排除”：[

“meta.description”，

“meta.attributes.\*”

]

}

}

}

}'

这些字段 (1,2,3,4) 将从存储的\_source字段中删除。

curl -XPUT 'http: // localhost: 9200 /logs/ event/ 1 ' -d '{

"requests" : {

"count" : 10 ,

"foo" : "bar" // 1

},

"meta" : {

"name" : "Some metric" ,

"description" : "Some metric description" , // 2

"attributes" : {

"foo" : "one" , // 3

"baz" : "two" // 4

}

}

}'

我们仍然可以在meta.attributes.foo字段中搜索，即使它不在存储的\_source.

curl -XGET 'http: // localhost: 9200 /logs/ event/\_search' -d '{

"query" : {

"match" : {

"meta.attributes.foo" : "one" //可搜索字段

}

}

}'

### 禁用 \_all 字段

该\_all字段是一个特殊的包罗万象的字段，它将所有其他字段的值连接成一个大字符串，使用空格作为分隔符，然后对其进行分析和索引但不存储。这意味着它可以被搜索，但不能被检索。

该\_all字段允许您在文档中搜索值，而无需知道哪个字段包含该值。这使得它在开始使用新数据集时成为一个有用的选项。例如：

curl -XPUT 'http: // localhost: 9200 /my\_index/user/ 1 ' -d '{

"first\_name" : "Will" ,

"last\_name" : "Smith" ,

"date\_of\_birth" : "1975-10-25"

}'

curl -XGET 'http: // localhost: 9200 /my\_index/ \_search' -d '{

"query" : {

"match" : {

"\_all" : "will smith 1975"

}

}

}'

**通过将启用**设置为\_all，可以完全禁用每个类型的字段：false

curl - XPUT 'http://localhost: 9200 /my\_index' -d '{

"mappings" : {

"type\_1" : {

"properties" : {...}

},

“type\_2”：{

“\_all”：{

“启用”：假

},

“属性”：{...}

}

}

}'

如果该\_all字段被禁用，则 URI 搜索请求以及**query\_string**和**simple\_query\_string**查询将无法将其用于查询。我们可以将它们配置为使用带有index.query.default\_field设置的不同字段：

curl -XPUT 'http: //localhost:9200/my\_index' -d '{

"mappings" : {

"my\_type" : {

"\_all" : {

"enabled" : false

},

“属性”：{

“内容”：{

“类型”：“文本”

}

}

}

},

“设置”：{

“index.query.default\_field”：“内容”

}

}'

### 禁用分析字段的规范

规范存储各种规范化因子（表示相对字段长度和索引时间提升设置的数字），稍后在查询时使用这些规范化因子，以计算文档相对于查询的分数。

**教程：**[**如何在 DigitalOcean 上运行 Supergiant 服务器**](https://supergiant.io/blog/how-to-run-supergiant-server-digitalocean)

尽管对评分很有用，规范也需要相当多的内存（通常按索引中每个字段的每个文档一个字节的顺序，即使对于没有此特定字段的文档也是如此）。因此，如果您不需要在特定领域评分，您应该禁用该领域的规范。特别是，仅用于过滤或聚合的字段就是这种情况。

事后可以禁用（但不能重新启用）规范，使用[**PUT 映射 API**](https://www.elastic.co/guide/en/elasticsearch/reference/2.0/indices-put-mapping.html)，如下所示：

curl -XPUT 'http: // localhost: 9200 /my\_index/ \_mapping/my\_type' -d '{

"properties" : {

"title" : {

"type" : "string" ,

"norms" : {

"enabled" : false

}

}

}

}'

规范不会立即被删除，但是当您继续索引新文档时，它们会随着旧段合并到新段中而被删除。对已删除规范的字段进行任何分数计算都可能返回不一致的结果，因为某些文档将不再具有规范，而其他文档可能仍具有规范。

### 注意默认 index\_options

您是否需要存储术语频率和位置，默认情况下，或者您可以使用更少的，也许只有文档编号？将**index\_options**设置为您真正需要的，如字符串核心类型描述中所述。

index\_options 参数控制将哪些信息添加到倒排索引，用于搜索和突出显示。它接受以下设置：

* **docs：**仅索引文档编号。可以回答这个问题这个领域是否存在这个术语？
* **freqs：**索引文档编号和术语频率。词频用于对重复词的评分高于单个词。
* **位置：**索引文档编号、术语频率和术语位置（或顺序）。位置可用于邻近或短语查询。
* **偏移量：**索引文档编号、术语频率、位置以及开始和结束字符偏移量（将术语映射回原始字符串）。帖子突出显示使用偏移量。

分析的字符串字段使用位置作为默认值，所有其他字段都使用文档作为默认值。

curl -XPUT 'http: // localhost: 9200 /my\_index' -d '{

"mappings" : {

"my\_type" : {

"properties" : {

"text" : {

"type" : "text" ,

"index\_options" : “抵消”

}

}

}

}

}'

curl -XPUT 'http: // localhost: 9200 /my\_index/my \_type/ 1 ' -d '{

"text" : "Quick brown fox"

}'

默认情况下，文本字段将使用**帖子突出显示**，因为**偏移量**是索引的。

curl -XGET 'http: // localhost: 9200 /my\_index/ \_search' -d '{

"query" : {

"match" : {

"text" : "brown fox"

}

},

“突出显示”：{

“字段”：{

“文本”：{}

}

}

}'

### 使用自动 ID 功能

如果您没有每个文档的自然 ID，请使用 Elasticsearch 自动 ID 功能。它经过优化以避免版本查找，因为自动生成的 ID 是唯一的。

如果您使用自己的 ID，请尝试选择对 Lucene 友好的 ID。示例包括**零填充的顺序 ID**、**UUID-1**和**nanotime**；这些 ID 具有一致的顺序模式，可以很好地压缩。相比之下，像 UUID-4 这样的 ID 本质上是随机的，并且提供的压缩效果很差，并且会降低 Lucene 的速度。

[**继续阅读“如何最大化 Elasticsearch 索引性能”的第 2 部分。**](https://qbox.io/blog/maximize-guide-elasticsearch-indexing-performance-part-2)

# [如何最大化 Elasticsearch 索引性能（第 2 部分）](https://qbox.io/blog/maximize-guide-elasticsearch-indexing-performance-part-2/)

亚当·范德布什( [Adam Vanderbush](https://qbox.io/blog/author/adam)) 于 2017 年 3 月 31 日发布

这篇文章是关于调整 Elasticsearch 索引的 3 部分系列文章的第 2 部分。[**第 1 部分可在此处找到**](https://qbox.io/blog/maximize-guide-elasticsearch-indexing-performance-part-1)。

本教程系列特别关注调整 elasticsearch 以实现最大索引吞吐量并减少监控和管理负载。Elasticsearch 是近乎实时的，从某种意义上说，当您索引文档时，您需要等待下一次刷新该文档才能出现在搜索中。

刷新是一项昂贵的操作，这就是为什么它会定期（默认）进行，而不是在每次索引操作之后进行。如果您计划为大量文档编制索引并且不需要立即可用于搜索的新信息，则可以通过降低刷新频率直到完成索引来优化索引性能而不是搜索性能。

对于这篇文章，我们将使用 Qbox.io 上托管的 Elasticsearch。您可以 [***在此处注册或启动您的集群***](https://qbox.io/signup?utm_source=blog&utm_campaign=tutorial&utm_term=launch_your_cluster&utm_medium=article)，或单击标题导航中的“开始使用”。如果您在设置方面需要帮助，请参阅“[***配置 Qbox Elasticsearch 集群。***](https://qbox.io/blog/provisioning-a-qbox-elasticsearch-cluster?utm_source=tutorial&utm_term=provision&utm_medium=article&utm_campaign=index-attachments-files-elasticsearch-mapper)“

**索引**的分片由多个**段组成**。来自 Lucene 的核心数据结构，一个段，本质上是一个索引的变更集。这些段在每次刷新时创建，随后在后台随着时间的推移合并在一起，以确保资源的有效利用；每个段都使用文件句柄、内存和 CPU。在幕后工作的 lucene 负责段合并，但如果不小心处理，它的计算成本可能会非常高，并且可能导致 Elasticsearch 自动将索引请求限制到单个线程。

在这里，我们将继续我们的弹性搜索索引调优策略，特别关注集群和索引级别的各种索引配置设置。

### 注意 refresh\_interval

此间隔由index.refresh\_interval设置定义，可以在 Elasticsearch 配置中或在每个索引的设置中进行。如果您同时使用两者，则索引设置会覆盖配置。默认值为**1s**，因此新索引的文档最多会在 1 秒后出现在搜索中。

因为刷新很昂贵，所以提高索引吞吐量的一种方法是增加refresh\_interval. 更少的刷新意味着更少的负载，更多的资源可以进入索引线程。因此，根据您的搜索要求，您可以考虑将刷新间隔设置为高于一秒。甚至可以暂时关闭完全刷新索引（通过将间隔设置为**-1**），例如，在批量索引运行期间，并在最后手动触发它。

更新设置 API 可用于动态更改索引，使其对批量索引的性能更高，然后将其移动到更实时的索引状态。在开始批量索引之前，请使用：

curl -XPUT 'localhost : 9200 /test/\_settings' -d '{

"index" : {

"refresh\_interval" : "-1"

}

}'

如果您正在执行大批量导入，请考虑通过设置禁用副本index.number\_of\_replicas: 0。当文档被复制时，整个文档被发送到副本节点并重复索引过程。这意味着每个副本都将执行分析、索引和潜在的合并过程。相反，如果您使用零副本进行索引，然后在摄取完成时启用副本，则恢复过程本质上是逐字节的网络传输。这比复制索引过程要有效得多。

curl -XPUT'localhost ： 9200 / my\_index/\_settings'-d'{

“index”：{

“number\_of\_replicas”：0

}

}'

然后，一旦完成批量索引，就可以更新设置（例如恢复默认值）：

curl -XPUT 'localhost : 9200 /my\_index/\_settings' -d '{

"index" : {

"refresh\_interval" : "1 s "

}

}'

并且，应该调用强制合并：

curl -XPOST 'localhost: 9200 /my\_index/\_forcemerge?max\_num\_segments=5'

刷新 API 允许显式刷新一个或多个索引，使自上次刷新以来执行的所有操作都可用于搜索。（接近）实时功能取决于所使用的索引引擎。例如，内部需要调用刷新，但默认情况下会定期安排刷新。

curl -XPOST '本地主机：9200 /my\_index/\_refresh'

### 细分和合并

段合并的计算成本很高，并且会占用大量磁盘 I/O。合并计划在后台运行，因为它们可能需要很长时间才能完成，尤其是大型部分。这通常很好，因为大段合并的比率相对较少。

**Kubernetes 系列：**[**了解为什么容器架构对您的业务的未来很重要**](https://supergiant.io/blog/how-to-make-containers-that-scale)

但有时合并会落后于摄取率。如果发生这种情况，Elasticsearch 会自动将索引请求限制到单个线程。这可以防止分段爆炸问题，即在合并之前生成数百个分段。

这里的 Elasticsearch 默认值是保守的：您不希望搜索性能受到后台合并的影响。但有时（尤其是在 SSD 或日志记录场景中），油门限制太低。

默认值为**20 MB/s**，这对于旋转磁盘来说是一个很好的设置。如果您有 SSD，您可能会考虑将其增加到**100–200 MB/s。**

curl -XPUT 'localhost : 9200 /\_cluster/settings' -d '{

"persistent" : {

"indices.store.throttle.max\_bytes\_per\_sec" : "100 mb "

}

}'

如果您正在进行批量导入并且根本不关心搜索，则可以完全禁用合并限制。这将允许索引以您的磁盘允许的速度运行：

curl -XPUT 'localhost:9200/\_cluster/settings' -d '{

“瞬态”：{

“ index.store.throttle.type ”： “无”

}

}'

将节流类型设置为 none 会完全禁用合并节流。完成导入后，将其重新设置为合并以重新启用限制。

curl -XPUT 'localhost:9200/\_cluster/settings' -d '{

“瞬态”：{

“ index.store.throttle.type ”： “合并”

}

}'

**注意**：以上设置仅适用于**Elasticsearch 1.X**版本。Elasticsearch 2.X 删除了所有索引/索引存储级别的速率限制 ( indices.store.throttle.type,   indices.store.throttle.max\_bytes\_per\_sec, index.store.throttle.type, index.store.throttle.max\_bytes\_per\_sec) 并切换到 Lucene 的**ConcurrentMergeScheduler**以自动管理节流。

合并调度程序 (ConcurrentMergeScheduler) 在需要时控制合并操作的执行。合并在单独的线程中运行，当达到最大线程数时，将等待进一步的合并，直到合并线程可用。

合并调度程序支持以下动态设置：  
index.merge.scheduler.max\_thread\_count

一次可以合并的最大线程数默认为Math.max(1, Math.min(4, Runtime.getRuntime().availableProcessors() / 2))适用于良好的固态磁盘 (SSD)。如果您的索引位于旋转盘片驱动器上，请将其减少到**1**。

旋转媒体在并发 I/O 上比较困难，因此我们需要减少每个索引可以同时访问磁盘的线程数。此设置将允许**max\_thread\_count + 2 个**线程一次在磁盘上操作，因此设置为 1 将允许**三个**线程。

如果您使用的是旋转媒体而不是 SSD，则需要将其添加到您的elasticsearch.yml：

索引.merge .scheduler .max\_thread\_count : 1

我们也可以在索引设置中进行设置：

curl -XPUT 'localhost: 9200 /my\_index/\_settings' -d '{

“index.merge.scheduler.max\_thread\_count”：1

}'

要为所有现有索引设置它，请使用：

curl -XPUT 'localhost: 9200 /\_settings' -d '{

“index.merge.scheduler.max\_thread\_count”：1

}'

### 事务日志的刷新

Translog 有助于防止在节点发生故障时丢失数据。它旨在帮助分片恢复可能在刷新之间丢失的操作。日志每 5 秒提交到磁盘，或者在每个成功的索引、删除、更新或批量请求（以先发生者为准）时提交。对 Lucene 的更改仅在 Lucene 提交期间持久化到磁盘，这是一个相对繁重的操作，因此无法在每次索引或删除操作后执行。在进程退出或硬件故障的情况下，在一次提交之后和另一次提交之前发生的更改将丢失。

为了防止这种数据丢失，每个分片都有一个与之关联的事务日志或预写日志。任何索引或删除操作都经过内部 Lucene 索引处理后写入 translog。如果发生崩溃，当分片恢复时，可以从事务日志中重播最近的事务。

**博客文章：**[**为什么 Supergiant 打包算法是独一无二的？它如何为我省钱？**](https://supergiant.io/blog/supergiant-packing-algorithm-unique-save-money)

Elasticsearch 刷新是执行 Lucene 提交并启动新的 translog 的过程。它是在后台自动完成的，以确保事务日志不会变得太大，这将使得在恢复过程中重放其操作需要相当长的时间。它也通过 API 公开，尽管很少需要手动执行。

与刷新索引分片相比，真正昂贵的操作是刷新其事务日志（这涉及 Lucene 提交）。Elasticsearch 根据可能在运行时更改的许多触发器执行刷新。通过延迟刷新或完全禁用它们，您可以提高索引吞吐量。请注意，没有什么是免费的，延迟刷新在最终发生时当然会花费更长的时间。

以下动态可更新设置控制内存缓冲区刷新到磁盘的频率：

* index.translog.flush\_threshold\_size– 一旦 translog 达到这个大小，就会发生刷新。默认为**512mb**。
* index.translog.flush\_threshold\_ops– 经过多少次操作后刷新。默认为**无限制**。
* index.translog.flush\_threshold\_period– 无论 translog 大小如何，在触发刷新之前等待多长时间。默认为**30m**。
* index.translog.interval– 检查是否需要刷新的频率，在间隔值和 2 倍间隔值之间随机化。默认为**5s**。

我们可以index.translog.flush\_threshold\_size从默认的**512** MB 增加到更大的值，例如 1 GB。这允许较大的段在刷新发生之前在 translog 中累积。通过让更大的段构建，您可以减少刷新的频率，并且更大的段合并的频率更低。所有这些加起来可以减少磁盘 I/O 开销并提高索引率。当然，您将需要相应数量的可用堆内存来积累额外的缓冲空间，因此在调整此设置时请记住这一点。

### 索引缓冲区的容量规划

索引缓冲区用于存储新索引的文档。当它填满时，缓冲区中的文档被写入磁盘上的一个段。它在节点上的所有分片之间划分。

以下设置是静态的，必须在集群中的每个数据节点上进行配置：

* indices.memory.index\_buffer\_size– 接受百分比或字节大小值。它默认为**10%，**这意味着分配给节点的总堆的 10% 将用作所有分片共享的索引缓冲区大小。
* indices.memory.min\_index\_buffer\_size– 如果 index\_buffer\_size 指定为百分比，则此设置可用于指定绝对最小值。默认为**48mb**。
* indices.memory.max\_index\_buffer\_size– 如果 index\_buffer\_size 指定为百分比，则此设置可用于指定绝对最大值。默认为无界。

该设置indices.memory.index\_buffer\_size定义了可用于索引操作的可用堆内存的百分比（剩余的堆内存将主要用于搜索操作）。如果您有大量数据要索引，则默认值 10% 可能太低，将其设置为更高的值可能是有意义的。

### 索引和批量操作线程池大小

考虑增加索引和批量操作的节点级线程池大小（并衡量它是否真的带来了改进）。

* index– 用于索引/删除操作。线程池类型固定为可用处理器数量，queue\_size 为 200。此池的最大大小为(1 + No. of available processors).
* bulk– 对于批量操作。线程池类型固定为可用处理器数量，queue\_size 为 50。此池的最大大小为(1 + No. of available processors).

单个分片，即 Lucene 级别，对允许同时执行索引的并发线程数有限制。在 Lucene 中默认为**8**，但在 ES 中，允许使用index.index\_concurrency.

我们应该更聪明地为其设置默认值，特别是在索引单个索引的情况下，一个节点上有一个分片。我们已经有了索引/批量线程池来保护和控制并发，所以我们可以在大多数情况下将默认值增加到更宽松的值。考虑增加该值，尤其是当节点上没有其他分片时（并衡量它是否有回报）。

# [如何最大化 Elasticsearch 索引性能（第 3 部分）](https://qbox.io/blog/maximize-guide-elasticsearch-indexing-performance-part-3/)

[亚当·范德布什 ( Adam Vanderbush](https://qbox.io/blog/author/adam)) 发表于 2017 年 4 月 5 日

这篇文章是关于调整 Elasticsearch 索引的 3 部分系列文章的第 3 部分。 [**第 1 部分可以在这里找到**](https://qbox.io/blog/maximize-guide-elasticsearch-indexing-performance-part-1) ， [**第 2 部分可以在这里找到**](https://qbox.io/blog/maximize-guide-elasticsearch-indexing-performance-part-2)。

本教程系列特别关注调整 elasticsearch 以实现最大索引吞吐量并减少监控和管理负载。

Elasticsearch 提供分片和复制作为扩展和提高索引可用性的推荐方式。过度分配是好的，但大量的分片是不好的。很难定义什么构成了太多的分片，因为这取决于它们的大小以及它们的使用方式。一百个很少使用的分片可能很好，而两个使用量非常大的分片可能太多了。监控您的节点以确保它们有足够的备用容量来处理异常情况。

对于这篇文章，我们将使用 Qbox.io 上托管的 Elasticsearch。您可以 [***在此处注册或启动您的集群***](https://qbox.io/signup?utm_source=blog&utm_campaign=tutorial&utm_term=launch_your_cluster&utm_medium=article)，或单击标题导航中的“开始使用”。如果您在设置方面需要帮助，请参阅“[***配置 Qbox Elasticsearch 集群。***](https://qbox.io/blog/provisioning-a-qbox-elasticsearch-cluster?utm_source=tutorial&utm_term=provision&utm_medium=article&utm_campaign=index-attachments-files-elasticsearch-mapper)“

横向扩展应该分阶段进行。建立足够的能力以进入下一阶段。一旦你进入下一个阶段，你就有时间考虑你需要做出哪些改变才能进入下一个阶段。优化将索引请求传输到 Elasticsearch 的方式也可以获得很多好处，例如 - 您是否必须为每个文档发送单独的请求？或者您可以缓冲文档以便使用批量 API 通过单个请求来索引多个文档？

我们之前研究过索引性能指标和设置，例如刷新、刷新、段合并和自动限制。本教程将列出一组想法，以参考分片和复制、请求、客户端和存储来提高 Elasticsearch 的索引吞吐量。

### 横向扩展 Elasticsearch 集群

Elasticsearch 是为扩展而构建的。它将在您的机器上或包含数百个节点的集群中非常愉快地运行，并且体验几乎相同。从小集群成长为大集群几乎是完全自动且无痛的。从一个大集群发展到一个非常大的集群需要更多的规划和设计，但它仍然相对轻松。

**教程：**[**使用 Supergiant 在 DigitalOcean 上自动扩展 Kubernetes**](https://supergiant.io/blog/how-to-auto-scaling-kubernetes-cluster-digitalocean)

Elasticsearch 中的默认设置将带您走很长一段路，但要获得最大的收益，您需要考虑数据如何在您的系统中流动。它可以是基于时间的数据（例如日志事件或社交网络流，其中相关性由新近度驱动）或基于用户的数据（其中可以按用户或客户细分大型文档集合）。

创建索引 API 允许实例化索引。Elasticsearch 提供对多个索引的支持，包括跨多个索引执行操作。创建的每个索引都可以具有与之关联的特定设置。索引的分片数需要在创建索引时设置，以后不能更改。如果您不确切知道期望有多少数据，您可以考虑过度分配一些分片（但不要太多，它们不是免费的！）以提供一些可用的备用容量。但是，副本的数量可以稍后更改。

curl -XPUT 'localhost : 9200 /my\_index -d '{

"settings" : {

"index" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 2

}

}

}'

索引别名也可以提供一种在以后的时间点扩展索引的方法（有限制）。索引别名 API 允许使用名称为索引设置别名，所有 API 都会自动将别名转换为实际的索引名称。一个别名也可以映射到多个索引，当指定它时，别名会自动扩展为别名索引。别名也可以与搜索时自动应用的过滤器和路由值相关联。别名不能与索引同名。

以下是将别名 alias1 与索引 test1 关联的示例：

curl -XPOST 'localhost : 9200 /\_aliases' -d '{

"actions" : [

{ “添加”：{“索引”：“ test1 ”，“别名”：“别名1 ”}}

]

}'

这里是删除相同的别名：

curl -XPOST 'localhost : 9200 /\_aliases' -d '{

"actions" : [

{ “删除”：{“索引”：“ test1 ”，“别名”：“别名1 ”}}

]

}'

重命名别名是在同一个 API 中简单的删除然后添加操作。这个操作是原子的，不用担心别名不指向索引的短时间内：

curl -XPOST 'localhost : 9200 /\_aliases' -d '{

"actions" : [

{ “删除”：{“索引”：“ test1 ”，“别名”：“别名1 ”}}，

{“添加”：{“索引”：“ test2 ”，“别名”：“别名1 ”}}

]

}'

### 复制

复制之所以重要，主要有两个原因：

* 它在分片或节点发生故障时提供高可用性。出于这个原因，重要的是要注意，副本分片永远不会与从中复制它的原始/主分片分配在同一节点上。
* 它允许您扩展搜索量/吞吐量，因为搜索可以在所有副本上并行执行。

复制是能够应对故障的一项重要功能，但是您拥有的副本越多，索引所需的时间就越长。因此，对于原始索引吞吐量，最好根本没有副本。幸运的是，与分片数量相比，您可以随时更改索引的副本数量，这为我们提供了一些额外的选择。

在某些情况下，例如最初填充新索引，或将数据从一个索引迁移到另一个索引，在时间关键的初始索引完成后，开始时不进行复制并仅在稍后添加副本可能会证明是有益的。

可以通过更新索引 API 更新副本数：

curl -XPUT'localhost ： 9200 / my\_index/\_settings'-d'{

“index”：{

“number\_of\_replicas”：0

}

}'

完成索引操作后，可以将副本数设置回相关值。

### 使用专用数据节点

数据节点保存包含您已索引的文档的分片。数据节点处理数据相关的操作，如 CRUD、搜索和聚合。这些操作是 I/O、内存和 CPU 密集型的。监控这些资源并在它们过载时添加更多数据节点非常重要。

拥有专用数据节点的主要好处是主角色和数据角色的分离。

要创建专用数据节点，请设置：

node.master： 假

node.data： 真

node.ingest： 假

当聚合器节点处理搜索查询并且仅根据需要联系数据节点时，它们会减轻数据节点的负载，从而具有更多处理索引请求的能力。

如果所有数据节点的磁盘空间不足，则需要向集群添加更多数据节点。您还需要确保您的索引有足够的主分片，以便能够在所有这些节点之间平衡它们的数据。[**在将分片分配给节点时，基于**](https://www.elastic.co/guide/en/elasticsearch/reference/current/disk-allocator.html)Elasticsearch磁盘的分片分配会考虑可用磁盘空间。默认情况下，它不会将分片分配给磁盘使用率超过 85% 的节点。

磁盘空间不足有两种补救措施。一种是删除过时的数据并将其存储在集群之外。对于所有用户来说，这可能不是一个可行的选择，但是，如果您要存储基于时间的数据，您可以在集群外存储旧索引数据的快照以进行备份，并更新索引设置以关闭这些索引的复制.

**教程：**[**使用 Supergiant 在 AWS EC2 上自动扩展 Kubernetes 集群**](https://supergiant.io/blog/auto-scaling-kubernetes-cluster-on-aws-ec2-with-supergiant)

如果您需要继续将所有数据存储在集群上，则第二种方法是您的唯一选择：垂直或水平扩展。如果您选择垂直扩展，这意味着升级您的硬件。但是，为了避免不得不再次升级，您应该利用 Elasticsearch 设计为水平扩展的事实。为了更好地适应未来的增长，您最好重新索引数据并在新创建的索引中指定更多主分片。

### 优化批量请求

[**Bulk API**](https://www.elastic.co/guide/en/elasticsearch/reference/2.4/docs-bulk.html)可以在单个 API 调用中执行许多索引/删除操作。这可以大大提高索引速度。每个子请求都是独立执行的，因此一个子请求的失败不会影响其他子请求的成功。如果任何请求失败，顶级错误标志将设置为 true，并将在相关请求下报告错误详细信息。

可能的操作是**索引、创建、删除**和**更新**。index 和 create 期望下一行有一个源，并且与标准索引 API 的 op\_type 参数具有相同的语义（即 index 将添加或替换文档，而如果具有相同索引和类型的文档已经存在，则 create 将失败） . delete 不期望以下行中的源，并且具有与标准删除 API 相同的语义。update 期望在下一行指定部分 doc、upsert 和 script 及其选项。

整个批量请求需要被接收到我们请求的节点加载到内存中，所以请求越大，其他请求可用的内存就越少。有一个批量请求的最佳大小。超过该大小，性能不再提高，甚至可能下降。然而，最佳尺寸并不是一个固定的数字。这完全取决于您的硬件、文档大小和复杂性，以及您的索引和搜索负载。

幸运的是，很容易找到这个最佳点：尝试以越来越大的批量索引典型文档。当性能开始下降时，你的批量太大了。一个好的起点是批量处理 1,000 到 5,000 个文档，或者，如果您的文档非常大，可以使用更小的批量。批量大小取决于您的数据、分析和集群配置，但一个好的起点是每个批量 5–15 MB。请注意，这是物理尺寸。文档数量不是批量大小的良好指标。例如，如果您要为每批 1,000 个文档编制索引，请记住以下几点：

* 1,000 个 1 KB 的文档是 1 MB。
* 1,000 个 100 KB 的文档是 100 MB。

这些是完全不同的散装尺寸。批量需要在协调节点加载到内存中，因此批量的物理大小比文档数量更重要。

从大约 5-15 MB 的块大小开始，然后慢慢增加它，直到您不再看到性能提升。然后开始增加批量摄取的并发性（多线程等）。

**使用Marvel**和/或**iostat**、**top**和**ps**等工具监控您的节点，以查看资源何时开始出现瓶颈。如果您开始收到**EsRejectedExecutionException**，则您的集群无法再跟上：至少一个资源已达到容量。要么减少并发，提供更多有限资源（例如从旋转磁盘切换到 SSD），要么添加更多节点。

摄取数据时，请确保批量请求在所有数据节点上循环。不要将所有请求发送到单个节点，因为该单个节点在处理时需要将所有批量存储在内存中。

### 贮存

如果您一直在遵循正常的开发路径，那么您可能一直在您的机器上或周围的一个小型机器集群上使用 Elasticsearch。但是在将 Elasticsearch 部署到生产环境时，您应该考虑一些建议。没有什么是一成不变的规则；Elasticsearch 用于广泛的任务和令人眼花缭乱的机器阵列。但这些建议基于我们在生产集群方面的经验提供了良好的起点。

磁盘通常是任何现代服务器的瓶颈。Elasticsearch 大量使用磁盘，您的磁盘可以处理的吞吐量越大，您的节点就越稳定。以下是一些优化磁盘 I/O 的技巧：

* 如果您买得起**SSD**，那么它们远远优于任何旋转介质。SSD 支持的节点在查询和索引性能方面都有所提升。或者，如果您使用旋转介质，请尽量获得最快的磁盘（高性能服务器磁盘、15k RPM 驱动器）。
* 使用**RAID 0**。条带化 RAID 将增加磁盘 I/O，但如果驱动器死机，则以潜在故障为代价。不需要使用 RAID 的镜像或奇偶校验变体，因为高可用性是通过副本内置到 Elasticsearch 中的。对于旋转磁盘和 SSD，使用 RAID 0 是提高磁盘速度的有效方法。
* 避免使用**EFS**，因为为了提供持久性、共享存储和增长/收缩而做出的牺牲是以性能为代价的。众所周知，此类文件系统会导致索引损坏，并且由于 Elasticsearch 是分布式的并且具有内置复制功能，因此不需要 EFS 提供的好处。
* 不要使用远程安装的存储，例如**NFS**或**SMB/CIFS**。这里引入的延迟与性能直接相反。
* 如果您在 EC2 上，请注意**EBS**。即使是 SSD 支持的 EBS 选项也通常比本地实例存储慢。EBS 非常适合运行小型集群（1-2 个节点），但无法承受更大的搜索和索引基础架构的负载。如果使用 EBS，则利用预置的 IOPS 来确保性能。
* 最后，避免使用网络附加存储 ( **NAS** )。人们经常声称他们的 NAS 解决方案比本地驱动器更快、更可靠。尽管有这些说法，但我们从未见过 NAS 辜负了它的炒作。NAS 通常较慢，显示较大的延迟和较大的平均延迟偏差，并且是单点故障。