

09 | 索引更新：刚发布的文章就能被搜到，这是怎么做到的？

2020-04-15 陈东

检索技术核心20讲

[进入课程 >](#)



讲述：陈东

时长 16:48 大小 15.39M



你好，我是陈东。

在前面的课程中，我们讲到，倒排索引是许多检索系统的核心实现方案。比如，搜索引擎对万亿级别网页的索引，就是使用倒排索引实现的。我们还讲到，对于超大规模的网页建立索引会非常耗时，工业界往往会使用分布式技术来并行处理。

对于发布较久的网页，搜索引擎可以有充足的时间来构建索引。但是一些新的网页和文章，往往发布了几分钟就可以被用户搜索到。这又是怎么做到的呢？今天，我们就来聊一聊这个问题。

工业界如何更新内存中的索引？


我们先来看这么一个问题：如果现在有一个小规模倒排索引，它能完全加载在内存中。当有新文章进入内存的时候，倒排索引该如何更新呢？这个问题看似简单，但是实现起来却非常复杂。

我们能想到最直接的解决思路是，只要解析新文章有哪些关键词，然后将文章 ID 加入倒排表中关键词对应的文档列表即可。没错，在没有其他用户使用的前提下，这样的方法是可行的。但如果你有过一定的工程经验，你就会知道，在实际应用中，必然会有多个用户同时访问这个索引。

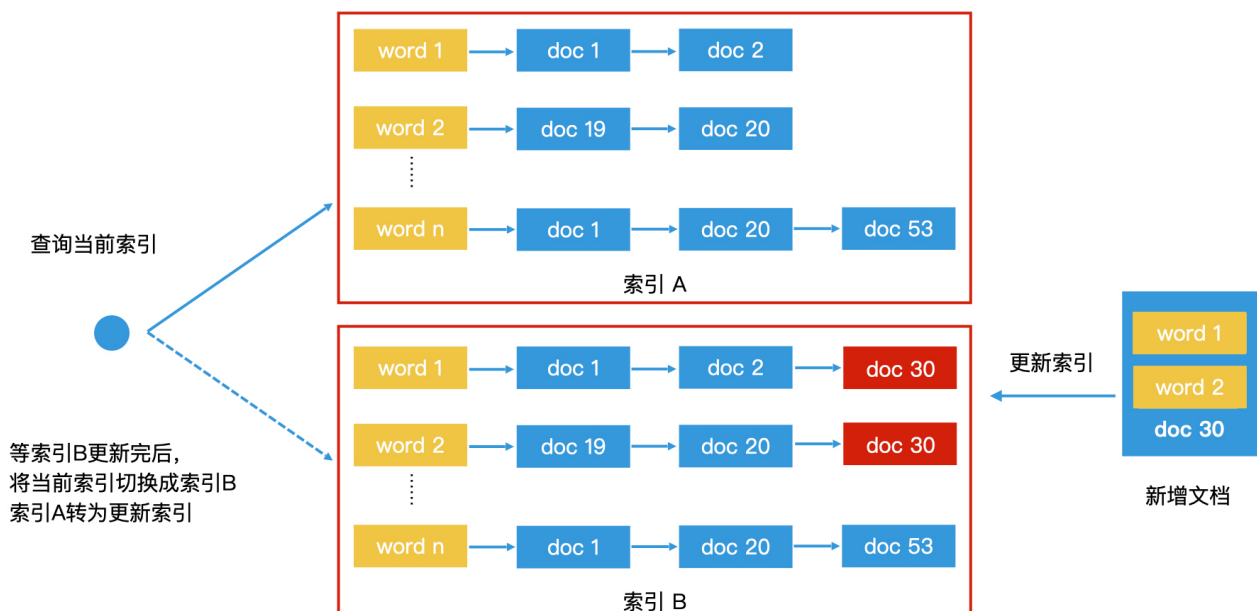
这个时候，如果我们直接更新倒排索引，就可能造成用户访问错误，甚至会引发程序崩溃。因此，一般来说，我们会对倒排表加上“读写锁”，然后再更新。但是，加上“锁”之后会带来频繁的读写锁切换，整个系统的检索效率会比无锁状态有所下降。

因此，为了使得系统有更好的性能，在工业界的实现中，我们会使用一种叫做“**Double Buffer（双缓冲）机制**”的解决方案，使得我们可以在无锁状态下对索引完成更新。

所谓“Double Buffer”，就是在内存中同时保存两份一样的索引，一个是索引 A，一个是索引 B。我们会使用一个指针 p 指向索引 A，表示索引 A 是当前可访问的索引。那么用户在访问时就会通过指针 p 去访问索引 A。这个时候，如果我们要更新，只更新索引 B。这样，索引 A 和索引 B 之间就不存在读写竞争的问题了。因此，在这个过程中，索引 A 和索引 B 都可以保持无锁的状态。

那更新完索引 B 之后，我们该如何告知用户应该来访问索引 B 呢？这时候，我们可以将指针 p 通过  **原子操作**（即无法被打断的最细粒度操作，在 Java 和 C++11 等语言中都有相应实现）从 A 直接切换到 B 上。接着，我们就把索引 B 当作“只读索引”，然后更新索引 A。

通过这样的机制，我们就能同时维护两个倒排索引，保持一个读、一个写，并且来回切换，最终完成高性能的索引更新。不过，为了避免切换太频繁，我们并不是每来一条新数据就更新，而是积累一批新数据以后再批量更新。这就是工业界常用的 Double Buffer 机制。



用 Double Buffer 机制更新索引

用 Double Buffer 机制更新索引是一个高效的方案，追求检索性能的应用场景常常会使用这种方案。但是对于索引到了一定量级的应用而言，使用 Double Buffer 会带来翻倍的内存资源开销。比如说，像搜索引擎这样万亿级网页的索引规模，数据大部分存储在磁盘上，更是无法直接使用 Double Buffer 机制进行更新的。因此，我们还是需要寻找其他的解决方案。

如何使用“全量索引结合增量索引”方案？

对于大规模的索引更新，工业界常用“全量索引结合增量索引”的方案来完成。下面，我们就一起来探讨一下，这个方案是如何实现索引更新的。

首先，系统会周期性地处理全部的数据，生成一份完整的索引，也就是**全量索引**。这个索引不可以被实时修改，因此为了提高检索效率，我们可以不加“锁”。那对于实时更新的数据我们应该怎样处理呢？我们会将新接收到的数据单独建立一个可以存在内存中的倒排索引，也就是**增量索引**。当查询发生的时候，我们会同时查询全量索引和增量索引，将合并的结果作为总的结果输出。这就是“**全量索引结合增量索引**”的更新方案。

其实这个方案还能结合我们上面讲的 Double Buffer 机制来优化。因为增量索引相对全量索引而言会小很多，内存资源消耗在可承受范围，所以我们可以使用 Double Buffer 机制对增量索引进行索引更新。这样一来，增量索引就可以做到无锁访问。而全量索引本身就是

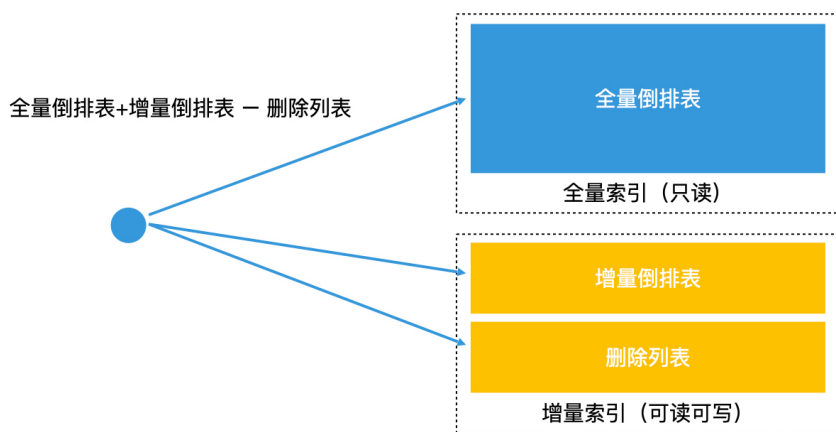
只读的，也不需要加锁。因此，整个检索过程都可以做到无锁访问，也就提高了系统的检索效率。

“全量索引结合增量索引”的检索方案，可以很好地处理新增的数据。那对于删除的数据，如果我们不做特殊处理，会有什么问题呢？下面，我们一起来分析一下。

假设，一个数据存储在全量索引中，但是在最新的实时操作中，它被删除了，那么在增量索引中，这个数据并不存在。当我们检索的时候，增量索引会返回空，但全量索引会返回这个数据。如果我们直接合并这两个检索结果，这个数据就会被留下作为检索结果返回，但是这个数据明明已经被删除了，这就会造成错误。

要解决这个问题，我们就需要在增量索引中保留删除的信息。最常见的解决方案是增加一个删除列表，将被删除的数据记录在列表中，然后检索的时候，我们将全量倒排表和增量倒排表的检索结果和删除列表作对比。如果结果数据存在于删除列表中，就说明该数据是无效的，我们直接删除它即可。

因此，完整的“全量索引结合增量索引”检索方案，需要在增量索引中保存一个删除列表。



全量索引结合增量索引的检索方案

增量索引空间的持续增长如何处理？

“全量索引结合增量索引”的方案非常实用，但是内存毕竟有限。如果我们不对内存中的增量索引做任何处理，那随着时间推移，内存就会被写满。因此，我们需要在合适的时机将增量索引合并到全量索引中，释放增量索引的内存空间。

将增量索引合并到全量索引中的常见方法有 3 种，分别是：完全重建法、再合并法和滚动合并法。下面，我们——来看。

1. 完全重建法

如果增量索引的增长速度不算很快，或者全量索引重建的代价不大，那么我们完全可以在增量索引写满内存空间之前，完全重建一次全量索引，然后将系统查询切换到新的全量索引上。

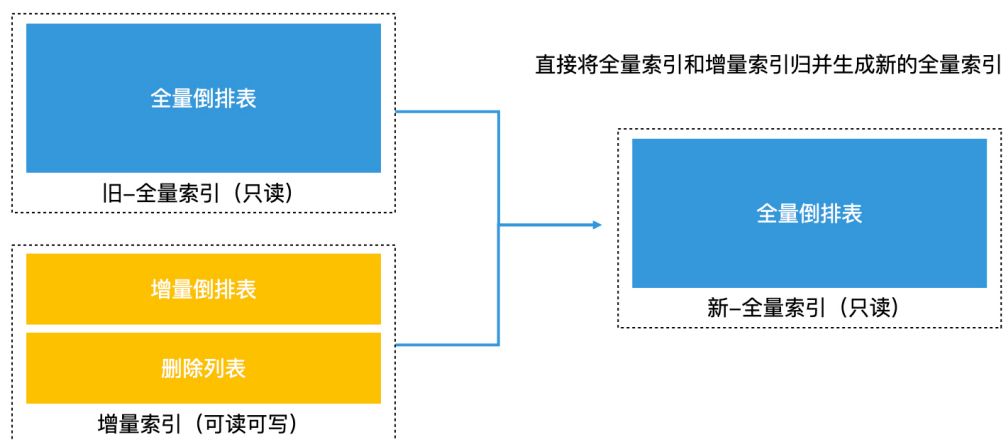
这样一来，之前旧的增量索引的空间也可以得到释放。这种方案叫作完全重建法。它对于大部分规模不大的检索系统而言，是十分简单可行的方案。

2. 再合并法

尽管完全重建法的流程很简单，但是效率并不是最优的。

在 [第 8 讲](#) 中我们讲过，对于较大规模的检索系统而言，在构建索引的时候，我们常常会将大数据集分割成多个小数据集，分别建立小索引，再把它们合并成一个大索引。

借助这样的思路，我们完全可以把全量索引想象成是一个已经将多个小索引合并好的大索引，再把增量索引想象成是一个新增的小索引。这样一来，我们完全可以直接归并全量索引和增量索引，生成一个新的全量索引，这也就避免了从头处理所有文档的重复开销。这种方法就是效率更高的再合并法。



3. 滚动合并法

不过，如果全量索引和增量索引的量级差距过大，那么再合并法的效率依然不高。

为什么这么说呢？我们以搜索引擎为例来分析一下。在搜索引擎中，增量索引只有上万条记录，但全量索引可能有万亿条记录。这样的两个倒排索引合并的过程中，只有少数词典中的关键词和文档列表会被修改，其他大量的关键词和文档列表都会从旧的全量索引中被原样复制出来，再重写入到新的全量索引中，这会带来非常大的无谓的磁盘读写开销。因此，对于这种量级差距过大的全量索引和增量索引的归并来说，如何避免无谓的数据复制就是一个核心问题。

最直接的解决思路就是**原地更新法**。所谓“原地更新法”，就是不生成新的全量索引，直接在旧的全量索引上修改。

但这种方法在工程实现上其实效率并不高，原因有两点。

首先，它要求倒排文件要拆散成多个小文件，每个关键词对应的文档列表为一个小文件，这样才可以将增量索引中对应的变化直接在对应的小文件上单独修改。但这种超大规模量级的零散小文件的高效读写，许多操作系统是很难支持的。

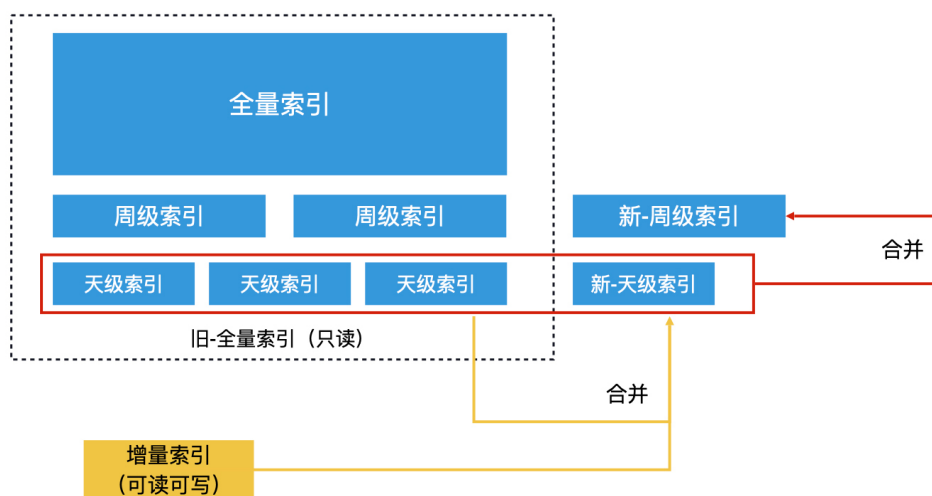
其次，由于只有一份全量索引同时支持读和写，那我们就需要“加锁”，这肯定也会影响检索效率。因此，在一些大规模工程中，我们并不会使用原地更新法。

这就又回到了我们前面要解决的核心问题，也就是如何避免无谓的数据复制，那在工业界中常用的减少无谓数据复制的方法就是**滚动合并法**。所谓滚动合并法，就是先生成多个不同层级的索引，然后逐层合并。

比如说，一个检索系统在磁盘中保存了全量索引、周级索引和天级索引。所谓**周级索引**，就是根据本周的新数据生成的一份索引，那**天级索引**就是根据每天的新数据生成的一份索引。在滚动合并法中，当内存中的增量索引增长到一定体量时，我们会用再合并法将它合并到磁盘上当天的天级索引文件中。

由于天级的索引文件条数远远没有全量索引多，因此这不会造成大量的无谓数据复制。等系统中积累了 7 天的天级索引文件后，我们就可以将这 7 个天级索引文件合并成一个新的周

级索引文件。因此，在每次合并增量索引和全量索引的时候，通过这样逐层滚动合并的方式，就不会进行大量的无谓数据复制的开销。这个过程就叫作滚动合并法。



滚动合并法

重点回顾

今天，我们介绍了工业界中，不同规模的倒排索引对应的索引更新方法。

对于内存资源足够的小规模索引，我们可以直接使用 **Double Buffer 机制** 更新内存中的索引；对于内存资源紧张的大规模索引，我们可以使用“**全量索引结合增量索引**”的方案来更新内存中的索引。

在“全量索引结合增量索引”的方案中，全量索引根据内存资源的使用情况不同，它既可以存在内存中，也可以存在磁盘上。而增量索引则需要全部存在内存中。

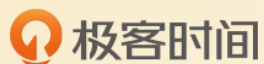
当增量索引增长到上限时，我们需要合并增量索引和全量索引，根据索引的规模和增长速度，我们可以使用的合并方法有完全重建法、再合并法和滚动合并法。

除此之外，我们还讲了一个很重要的工业设计思想，就是读写分离。实际上，高效的索引更新方案都应用了读写分离的思想，将主要的数据检索放在一个只读的组件上。这样，检索时就不会有读写同时发生的竞争状态了，也就避免了加锁。事实上，无论是 Double Buffer 机制，还是全量索引结合增量索引，都是读写分离的典型例子。

课堂讨论

为什么在增量索引的方案中，对于删除的数据，我们不是像 LSM 树一样在索引中直接做删除标记，而是额外增加一个删除列表？

欢迎在留言区畅所欲言，说出你的思考过程。如果有收获，也欢迎把这篇文章分享给你的朋友。



检索技术核心 20 讲

从搜索引擎到推荐引擎，带你吃透检索

陈东

奇虎 360 商业产品事业部
资深总监



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 08 | 索引构建：搜索引擎如何为万亿级别网站生成索引？

下一篇 10 | 索引拆分：大规模检索系统如何使用分布式技术加速检索？

精选留言 (12)

写留言



一步

2020-04-15

对于结尾的问题：我在补偿一下，除了上面说的原因还有就是，一个文档会有多个 key，也不可能对文档包含的每个 key 进行文档标记

展开 ∨

作者回复: 综合你前一条一起回复, 你说到了两个点上:

- 1.倒排索引和kv不一样, posting list元素很多, 每个元素都加标记代价太大。
- 2.一个文档可能会影响多个key, 因此每个文档都要修改标记的话, 读写操作会很频繁, 加锁性能下降。

此外, 还有一点是, 加上标记也没啥用, 在posting list求交并的过程中, 依然要全部留下来, 等着最后和全量索引合并时才能真正删除。这样的话不如直接用一个delete list存着, 最后求交集更高效。



2



青生先森

2020-04-21

双缓存机制有个疑问, 假如A更新了一个数据1, B也需要更新数据1, 这个如何保证呢?

作者回复: 是这样, 双缓存机制的设计理念是读写分离, 一个索引只负责写, 另一个索引只负责读。因此, 不会存在你说的两个索引同时被写的情况。



1



Mq

2020-04-15

看不懂滚动合并机制, 老师能结合具体数据分析下, 例如我今天增加了几个网页, 有倒排索引关键字的value都要加上这个网页, 这个滚动合并的流程是咋样的。

展开 ∨

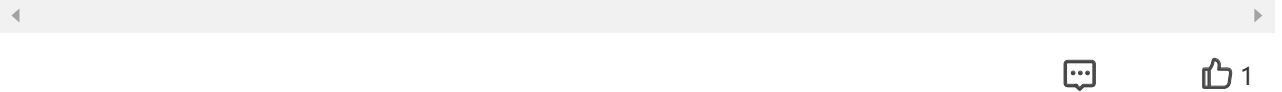
作者回复: 滚动合并机制的确是最复杂的一种。它的核心思想是“解决小索引和大索引合并的效率问题, 避免大索引产生大量无谓的复制操作”。而解决方案则是“在小索引和大索引中间加入中索引进行过渡”。

这个设计方案其实会很常见。比如说在lsm树那一课, 我说了“假设只有c0树和c1树”, 而实际情况是c1树会非常大, 合并效率会很低, 因此lsm树的设计中就有着多棵不同大小的树。包括level db的实现, 也会有着多层索引。因此, 这是一个值得我们学习和掌握的方法。

至于你举的这个例子, 结合文中的内容, 使用滚动合并的流程是这样的:

- 1.今天增加的网页会先存在内存的增量索引中。
- 2.增量索引满了, 要开始合并。
- 3.增量索引和当天的天级索引合并(天级索引不大, 所以合并代价小)。
- 4.当天级索引达到了7天时, 可以将多个天级索引合并, 变成一个新的周级索引。
- 5.当有多个周级索引的时候, 全量索引会和多个周级索引合并, 生成一份新的全量索引。(不过,

一般这一步会用重新生成全量索引来代替，你可以理解为为了保证系统的稳定性，需要定期进行索引重建。就像系统要进行定期重启一样)。



兰柯一梦

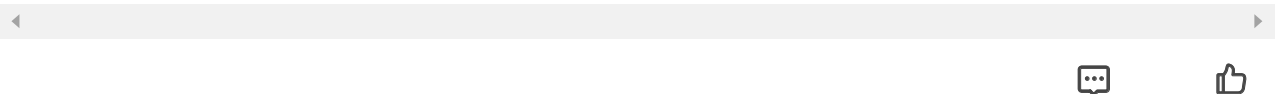
2020-04-16

如果在doc的正排字段中做标记删除是不是也可以呢？这样等各个索引进行合并的时候，看doc对应的正排的删除标记，如果是删除状态那边直接丢掉

展开 ∨

作者回复: 你的想法很好，其实是有可能的。本质上，你是复用了正排表，让它承载了删除列表的功能。在最后posting list合并的时候，通过查正排表完成过滤(其实就是加餐一中说的哈希表法:将删除列表变成了哈希表)。

在系统比较简单的时候，这样使用是OK的。不过当系统足够复杂的时候，我们需要将不同功能和数据进行合理的划分，倒排检索和正排查询有可能是两个不同的环节和模块(包括中间可能还有其他环节，比如抽取特征，打分计算等)。因此从这个角度出发，复杂系统才会抽象出删除列表这个对象，这样就可以不依赖于正排表，从而完成了系统架构的解耦设计。



每天晒白牙

2020-04-15

第一时间看到这个思考题，我没啥思路，看了大家的留言和老师的回复，学到了，把老师的回复总结了起来

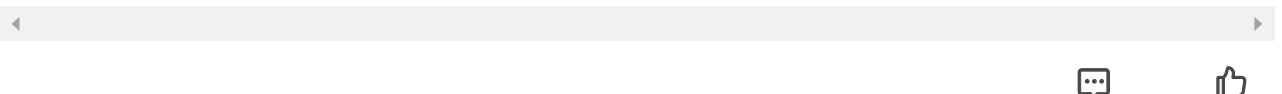
为什么在增量索引中，对于要删除的数据没有像 LSM 树那样一样在索引中直接做删除标记，而是额外增加一个删除列表？

1.倒排所以和 kv 存储还是有不一样的地方，倒排索引的 posting list 元素有很多，每个...

展开 ∨

作者回复: 总结得很认真。相信你学完这一课后，再去看es中的segment的处理就会很轻松了，比如segment的生成和合并，还有.del文件存储删除列表等。

此外，你还可以思考索引更新这一块，你们当前系统的实现方案是否合理，是否有优化空间等。



一步

2020-04-15

在滚动合并方案中，查询也要一级一级的进行查询，先查增量索引---> 天级索引----> 周

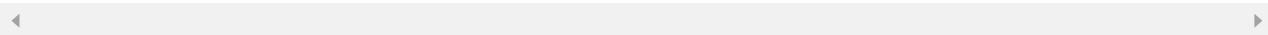
级索引---> 最后是权重索引。这个的话查询的链路增加了好多，查询的效率会降低多少？

作者回复: 这些是可以并行查找的。而不是串行。

而且一般来说，以现在的机器处理能力，周级索引其实也可以不用的，这样也能减少系统复杂度。

因此一般系统实现就是：

增量索引+ 天级索引+全量索引 三个索引并行检索，再合并结果。



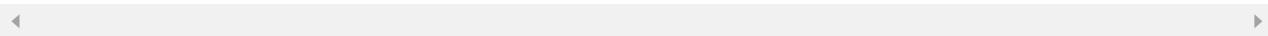
一步

2020-04-15

为什么在增量索引的方案中，对于删除的数据，我们不是像 LSM 树一样在索引中直接做删除标记，而是额外增加一个删除列表？

这个我认为，删除的数据相对全量的数据是非常少的，如果用删除标记，那么全部的数据都要进行标记，这样大量的没有删除的数据都会有个未删除的标志，极大的浪费空间资源
展开 ∨

作者回复: 在另一条下面统一回复你了。



峰

2020-04-15

对这个问题，老师故意在文章里说的很含糊，只说了个记录删除列表的思路 😊。

lsm之所以可以和删除标记一起存，核心在于类似kv存，删除标记和对应的v是共享k的，所以要拿是会一起拿出来，就可以判断数据这个时候存在不存在，相当于拿到了值的变迁历史。

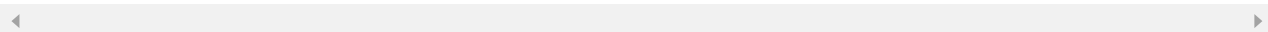
而这个场景，删除文档，对文档集合而言，也可以添加个删除标记，但对于索引而言，...
展开 ∨

作者回复: 哈哈，的确说得没那么透，只说了“怎么做”，但没说“为什么”。因此才在课后讨论题让大家想想为什么。

你说到了很重要的一点，kv只有一个值，但倒排表是一整个posting list，所以修改代价会大。

另一方面，即便是使用double buffer技术对增量索引做无锁更新，但增量索引检索过程中，依然要把所有被删除的文档保留到最后，再和全量索引做合并。

那既然所有的标记都要保留到最后一步，不如直接在最后一步用一个delete list来求交集更快，逻辑也清晰。





范闲

2020-04-15

- 1.如果增加一个删除标记，相当于增量索引的每个内容都有这样一个标记，随着增量的数量变大，内存占用会更高。
- 2.利用删除列表就不会有这样的问題，同样可以避免加锁。

展开 ▾

作者回复: 这两点都很好。

的确posting list里每个元素都加标记，这个代价会远大于lsm这种只存一个元素kv的场景。

此外，一个文档被删除，它可能会影响很多key和posting list，这个读写加锁代价不小。

还有，即使使用double buffer实现增量索引，但是这个标记也没什么用。我们在增量索引中求交集和并集时，依然要保留所有的元素，这样和全量索引的结果合并时才不会出错。因此提前打上标记并不能加快检索效率。不如最后记录一个delete list，然后快速求交集处理掉。



pedro

2020-04-15

按照索引的高性能选择，全量索引是只读的，而增量索引和删除项是可读可写的，所以不会选择在索引上添加删除项，会拉低系统效率。

展开 ▾

作者回复: 你很好地吸收了读写分离的思想。全量索引上肯定是不能加删除项的。不过可读写的增量索引上面能否加上删除标记呢？你可以想一想。

提示:

1.加的话是否有性能损失。

2.不管有没有性能损失，加上后，求检索结果的过程是怎么样的？加上这个标记和单独记录一个删除列表相比有帮助么？

你在思考以后，可以再看看我的回复。



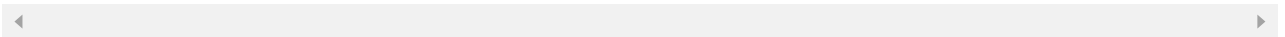
那一刻

2020-04-15

在滚动更新中，周索引往全量索引更新的时候，需要加锁操作么？

作者回复: 这一步我没有画，不过你可以沿着前面天级索引和周级索引合并的思路思考一下，包括总结时我强调的“读写分离”去想想，我相信，你应该可以得到“不要加锁”这个结论的。

这里我也补充一点，就是周级到全量索引的合并，其实由于隔的时间已经很久了，因此，很多时候我们会直接完全重建全量索引，一方面，重建时间是足够的，另一方面，也等于定期给系统“重启”，保证系统的稳定性和正确性。



Kăfkă²⁰²⁰

2020-04-15

用删除列表而不是打删除标记，可以避免对增量索引加锁

作者回复: 避免加锁操作的确是一个考虑因素。新删除一个文档，这个文档里可能有很多key，如果要打删除标记，就意味着这些key和posting list都要执行加锁操作，这个代价的确会比较大。

而且，即使我们使用double buffer，对于增量索引不加锁，那么你可以想想处理过程，如果对于增量索引的posting list中的文档打上删除标记，在进行交并操作的时候，所有的文档都依然要被留下！因为增量索引的结果需要和全量索引结果合并，如果增量索引的结果没有保留删除标记，那么合并时会出错。

既然删除标记要在增量索引处理过程中一直保留，那不如单独记录来得方便。

