ElasticSearch详解与优化设计

笔记本： 技术文章
创建时间： 2016/3/25 13:51 **更新时间：** 2016/3/25 14:04
URL： http://www.tuicool.com/articles/7fueUbb

目录

1简介

ElasticSearch（简称ES）是一个分布式、Restful的搜索及分析服务器，设计用于分布式计算；能够达到实时搜索，稳定，可靠，快速。和Apache Solr一样，它也是基于Lucence的索引服务器，而ElasticSearch对比Solr的优点在于：

- 轻量级：安装启动方便，下载文件之后一条命令就可以启动。
- Schema free：可以向服务器提交任意结构的JSON对象，Solr中使用schema.xml指定了索引结构。
- 多索引文件支持：使用不同的index参数就能创建另一个索引文件，Solr中需要另行配置。
- 分布式：Solr Cloud的配置比较复杂。

2013年初，GitHub抛弃了Solr，采取ElasticSearch 来做PB级的搜索。

近年ElasticSearch发展迅猛，已经超越了其最初的纯搜索引擎的角色，现在已经增加了数据聚合分析（aggregation）和可视化的特性，如果你有数百万的文档需要通过关键词进行定位时，ElasticSearch肯定是最佳选择。当然，如果你的文档是JSON的，你也可以把ElasticSearch当作一种“NoSQL数据库”，应用ElasticSearch数据聚合分析（aggregation）的特性，针对数据进行多维度的分析。

ElasticSearch，些国内外的优秀实例：

ElasticSearch一些国内外的优秀案例：

- Github：“GitHub使用ElasticSearch搜索20TB的数据，包括13亿文件和1300亿行代码”。
- SoundCloud：“SoundCloud使用ElasticSearch为1.8亿用户提供即时而精准的音乐搜索服务”。
- 百度：百度目前广泛使用ElasticSearch作为文本数据分析，采集百度所有服务器上的各类指标数据及用户自定义数据，通过对各种数据进行多维分析展示，辅助定位分析实例异常或业务层面异常。目前覆盖百度内部20多个业务线（包括casio、云分析、网盟、预测、文库、直达号、钱包、风控等），单集群最大100台机器，200个ES节点，每天导入30TB+数据。

2 概念

Cluster和Node

ES可以以单点或者集群方式运行，以一个整体对外提供search服务的所有节点组成cluster，组成这个cluster的各个节点叫做node。

Index

这是ES存储数据的地方，类似于关系数据库的database。

Shards

索引分片，这是ES提供分布式搜索的基础，其含义为将一个完整的index分成若干部分存储在相同或不同的节点上，这些组成index的部分就叫做shard。

Replicas

索引副本，ES可以设置多个索引的副本，副本的作用一是提高系统的容错性，当个某个节点某个分片损坏或丢失时可以从副本中恢复。二是提高ES的查询效率，ES会自动对搜索请求进行负载均衡。

Recovery

代表数据恢复或叫数据重新分布，ES在有节点加入或退出时会根据机器的负载对索引分片进行重新分配，挂掉的节点重新启动时也会进行数据恢复。

Gateway

ES索引快照的存储方式，ES默认是先把索引存放到内存中，当内存满了时再持久化到本地硬盘。gateway对索引快照进行存储，当这个ES集群关闭再重新启动时就会从gateway中读取索引备份数据。

Discovery.zen

代表ES的自动发现节点机制，ES是一个基于p2p的系统，它先通过广播寻找存在的节点，再通过多播协议来进行节点之间的通信，同时也支持点对点的

存在的节点，再通过多播协议来进行节点之间的通信，同时也支持点对点的交互。

Transport

代表ES内部节点或集群与客户端的交互方式，默认内部是使用tcp协议进行交互，同时它支持http协议（json格式）。

3 安装部署

ES由java语言实现，运行环境依赖java。ES 1.x版本，官方推荐使用jdk1.7+的环境，建议使用oracle jdk1.8；ES可以去官网下载，本文使用elasticsearch-1.6.0.tar.gz。

4 ES安装

（1）解压elasticsearch-1.6.0.tar.gz，sudo tar -zvxf elasticsearch-1.6.0.tar.gz，在当前路径生成目录：elasticsearch-1.6.0；

（2）配置ES。这里只做最简单的配置，修改ES_HOME/config/elasticsearch.yml文件， 相关配置参数

```
#集群名称

cluster.name: elasticsearch

#节点名称

node.name: "node1"

#节点是否存储数据

node.data: true

#索引分片数

index.number_of_shards: 5

#索引副本数

index.number_of_replicas: 1

#数据目录存放位置

path.data: /data/elasticsearch/data

#日志数据存放位置

path.logs: /data/elasticsearch/log

#索引缓存

index.cache.field.max_size: 500000
```

index.cache.field.max_size: 50000

#索引缓存过期时间

index.cache.field.expire: 5m

（3）启动ES。进入ES安装目录，执行命令：bin/elasticsearch -d -Xms512m -Xmx512m，然后在浏览器输入 http://ip:9200/，查看页面信息，是否正常启动。status=200表示正常启动了。

5 数据索引

ES索引我们可以理解为数据入库的一个过程。我们知道ES是基于Lucene框架的一个分布式检索平台。索引的同样也是基于Lucene创建的，只不过在其上层做了一些封装。ElasticSearch客户端支持多种语言如PHP、Java、Python、Perl等，下面以Python为例：

一，安装官方提供的Python API

>> pip install elasticsearch

二，创建索引

调用create或index方法，创建索引。

三，批量录入索引数据

ElasticSearch批量索引的命令是bulk，利用Python API提交

四，数据检索查询

五，数据更新、删除

对于索引的批量删除和更新操作，对应的文档格式如下，更新文档中的doc节点是必须的。

六、常见错误

- SerializationError：JSON数据序列化出错，通常是因为不支持某个节点值的数据类型
- RequestError：提交数据格式不正确
- ConflictError：索引ID冲突
- TransportError：连接无法建立

6 索引优化

ES索引优化主要从两个方面解决问题：

一、索引数据过程

大家可能会遇到索引数据比较慢的过程。其实明白索引的原理就可以有针对性的进行优化。ES索引的过程到相对Lucene的索引过程多了分布式数据的扩展，而这ES主要是用tranlog进行各节点之间的数据平衡。所以从上我可以通过索引的settings进行第一优化：

这两个参数第一是到tranlog数据达到多少条进行平衡，默认为5000，而这个过程相对而言是比较浪费时间和资源的。所以我们可以将这个值调大一些还是设为-1关闭，进而手动进行tranlog平衡。第二参数是刷新频率，默认为120s是指索引在生命周期内定时刷新，一但有数据进来能refresh像lucene里面commit,我们知道当数据addDoucment后，还不能检索到要commit之后才能行数据的检索，所以可以将其关闭，在最初索引完后手动refresh一之，然后将索引setting里面的index.refresh_interval参数按需求进行修改，从而可以提高索引过程效率。

另外的知道ES索引过程中如果有副本存在，数据也会马上同步到副本中去。我个人建议在索引过程中将副本数设为0，待索引完成后将副本数按需量改回来，这样也可以提高索引效率。

"number_of_replicas": 0

二、检索过程

其实检索速度快度与索引质量有很大的关系。而索引质量的好坏主要与以下几方面有关：

1、分片数

分片数，与检索速度非常相关的的指标，如果分片数过少或过多都会导致检索比较慢。分片数过多会导致检索时打开比较多的文件别外也会导致多台服务器之间通讯。而分片数过少会导致单个分片索引过大，所以检索速度慢。基于索引分片数=数据总量/单分片数的计算公式，在确定分片数之前需要进行单服务单索引单分片的测试，目前我们测试的结果单个分片的内容为10G。

2、副本数

副本数与索引的稳定性有比较大的关系，如果Node在非正常挂了，经常会导致分片丢失，为了保证这些数据的完整性，可以通过副本来解决这个问题。建议在建完索引后在执行Optimize后，马上将副本数调整过来。

3、分词

分词对于索引的影响可大可小，看自己把握。大家或许认为词库越多，分词效果越好，索引质量越好，其实不然。分词有很多算法，大部分基于词表进行分词。也就是说词表的大小决定索引大小。所以分词与索引膨涨率有直接关系。词表不应很多，而对文档相关特征性较强的即可。比如论文的数据进行建索引，分词的词表与论文的特征越相似，词表数量越小，在保证查全查准的情况下，索引的大小可以减少很多。索引大小减少了，那么检索速度也就提高了。

4、索引段

索引段即lucene中的segments概念，我们知道ES索引过程中会refresh和tranlog也就是说我们在索引过程中segments number不只一个。而segments number与检索是有直接联系的，segments number越多检索越慢，而将segments numbers 有可能的情况下保证为1，这将可以提高将近一半的检索速度。

7 内存优化

ES对于内存的消耗，和很多因素相关，诸如数据总量、mapping设置、查询方式、查询频度等等。默认的设置虽开箱即用，但不能适用每一种使用场景。作为ES的开发、运维人员，如果不了解ES对内存使用的一些基本原理，就很难针对特有的应用场景，有效的测试、规划和管理集群，从而踩到各种坑，被各种问题挫败。

1要理解ES如何使用内存，先要尊重下面两个基本事实：

ES是JAVA应用

首先，作为一个JAVA应用，就脱离不开JVM和GC。很多人上手ES的时候，对GC一点概念都没有就去网上抄各种JVM“优化”参数，却仍然被heap不够用，内存溢出这样的问题搞得焦头烂额。即使对于JVM GC机制不够熟悉，头脑里还是需要有这么一个基本概念: 应用层面生成大量长生命周期的对象，是给heap造成压力的主要原因，例如读取一大片数据在内存中进行排序，或者在heap内部建cache缓存大量数据。如果GC释放的空间有限，而应用层面持续大量申请新对象，GC频度就开始上升，同时会消耗掉很多CPU时间。严重时可能恶性循环，导致整个集群停工。因此在使用ES的过程中，要知道哪些设置和操作容易造成以上问题，有针对性的予以规避。

底层存储引擎是基于Lucene的

Lucene的倒排索引(Inverted Index)是先在内存里生成，然后定期以段文件(segment file)的形式刷到磁盘的。每个段实际就是一个完整的倒排索引，并且一旦写到磁盘上就不会做修改。 API层面的文档更新和删除实际上是增量写入的一种特殊文档，会保存在新的段里。不变的段文件易于被操作系统cache，热数据几乎等效于内存访问。

基于以上2个基本事实，我们不难理解，为何官方建议的heap size不要超过系统可用内存的一半。heap以外的内存并不会被浪费，操作系统会很开心的利用他们来cache被用读取过的段文件。

Heap分配多少合适？遵从官方建议就没错。 不要超过系统可用内存的一半，并且不要超过32GB。JVM参数呢？对于初级用户来说，并不需要做特别调整，仍然遵从官方的建议，将xms和xmx设置成和heap一样大小，避免动态分配heap size就好了。虽然有针对性的调整JVM参数可以带来些许GC效率的提升，当有

一些“坏”用例的时候，这些调整并不会有什么魔法效果帮你减轻heap压力，甚至可能让问题更糟糕。

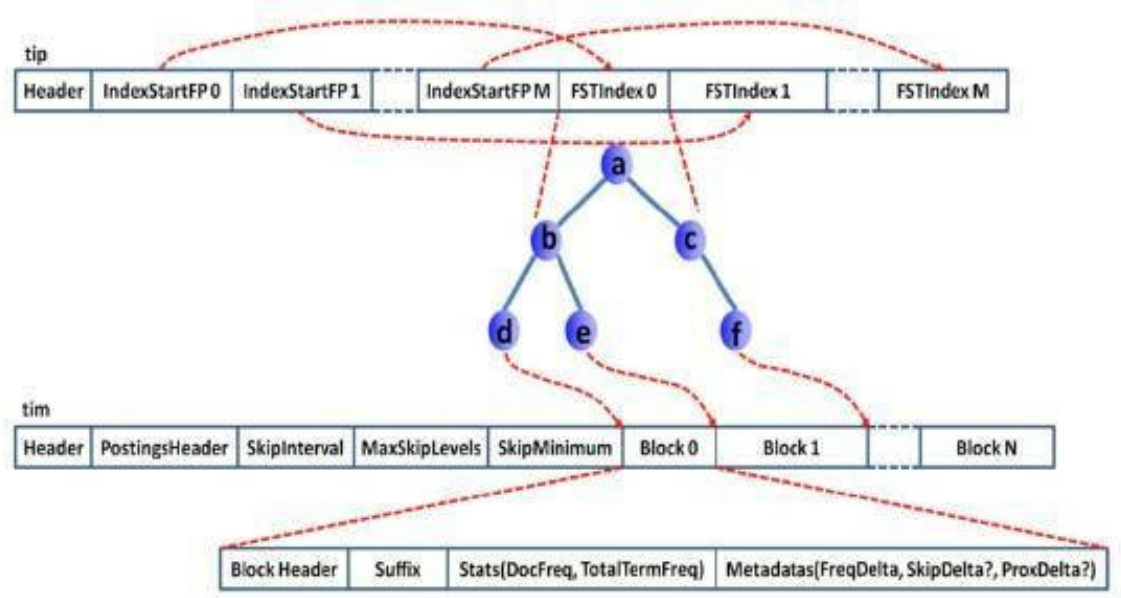
2ES的heap是如何被瓜分掉的?

以下分别做解读几个我知道的内存消耗大户：

Segment Memory

Segment不是file吗？segment memory又是什么？前面提到过，一个segment是一个完备的lucene倒排索引，而倒排索引是通过词典 (Term Dictionary)到文档列表(Postings List)的映射关系，快速做查询的。 由于词典的size会很大，全部装载到heap里不现实，因此Lucene为词典做了一层前缀索引(Term Index)，这个索引在Lucene4.0以后采用的数据结构是FST (Finite State Transducer)。 这种数据结构占用空间很小，Lucene打开索引的时候将其全量装载到内存中，加快磁盘上词典查询速度的同时减少随机磁盘访问次数。

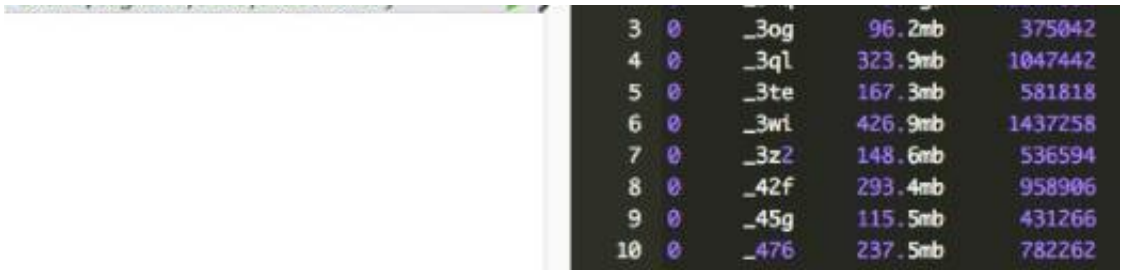
下面是词典索引和词典主存储之间的一个对应关系图:



说了这么多，要传达的一个意思就是，ES的data node存储数据并非只是耗费磁盘空间的，为了加速数据的访问，每个segment都有会一些索引数据驻留在heap里。因此segment越多，瓜分掉的heap也越多，并且这部分heap是无法被GC掉的！ 理解这点对于监控和管理集群容量很重要，当一个node的segment memory占用过多的时候，就需要考虑删除、归档数据，或者扩容了。

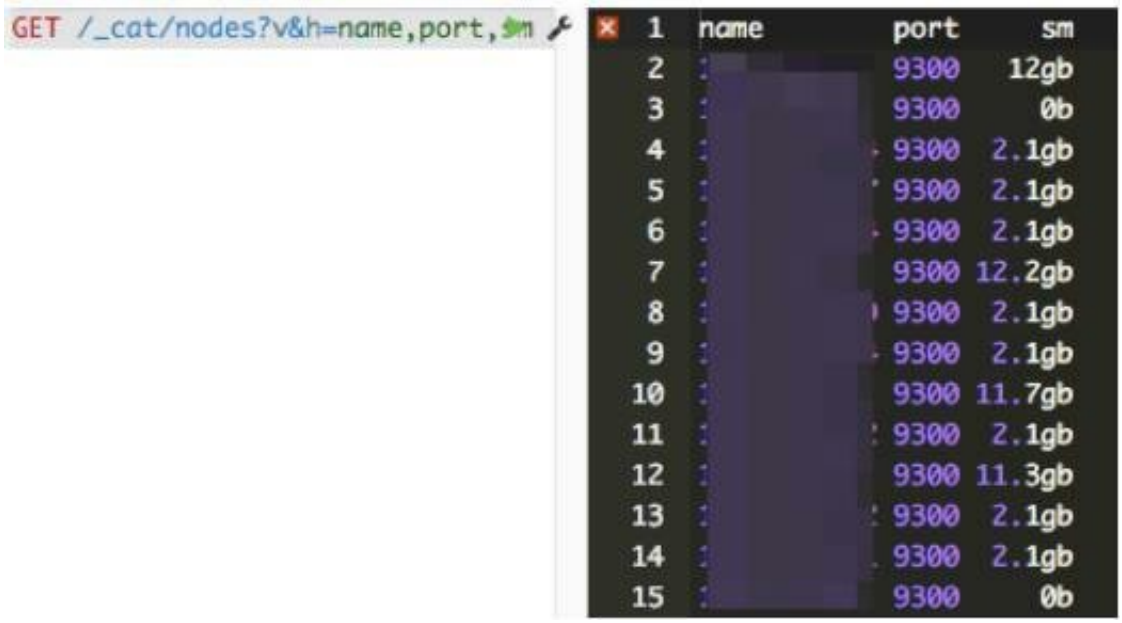
怎么知道segment memory占用情况呢？ CAT API可以给出答案。

1. 查看一个索引所有segment的memory占用情况：



3	0	_3og	96.2mb	375042
4	0	_3ql	323.9mb	1047442
5	0	_3te	167.3mb	581818
6	0	_3wi	426.9mb	1437258
7	0	_3z2	148.6mb	536594
8	0	_42f	293.4mb	958906
9	0	_45g	115.5mb	431266
10	0	_476	237.5mb	782262

2. 查看一个node上所有segment占用的memory总和:



```
GET /_cat/nodes?v&h=name,port,sm
```

	name	port	sm
2	[redacted]	9300	12gb
3	[redacted]	9300	0b
4	[redacted]	9300	2.1gb
5	[redacted]	9300	2.1gb
6	[redacted]	9300	2.1gb
7	[redacted]	9300	12.2gb
8	[redacted]	9300	2.1gb
9	[redacted]	9300	2.1gb
10	[redacted]	9300	11.7gb
11	[redacted]	9300	2.1gb
12	[redacted]	9300	11.3gb
13	[redacted]	9300	2.1gb
14	[redacted]	9300	2.1gb
15	[redacted]	9300	0b

那么有哪些途径减少data node上的segment memory占用呢？ 总结起来有三种方法：

1. 删除不用的索引。
2. 关闭索引（文件仍然存在于磁盘，只是释放掉内存）。需要的时候可以重新打开。
3. 定期对不再更新的索引做optimize (ES2.0以后更改为force merge api)。这Optimze的实质是对segment file强制做合并，可以节省大量的segment memory。

Filter Cache

Filter cache是用来缓存使用过的filter的结果集的，需要注意的是这个缓存也是常驻heap，无法GC的。默认的10% heap size设置工作得够好了，如果实际使用中heap没什么压力的情况下，才考虑加大这个设置。

Field Data cache

对搜索结果做排序或者聚合操作，需要将倒排索引里的数据进行解析，然后进行一次倒排。在有大量排序、数据聚合的应用场景，可以说field data cache是性能和稳定性的杀手。这个过程非常耗费时间，因此ES 2.0以前的版本主要依赖这个cache缓存已经计算过的数据，提升性能。但是由于heap空间有限，当遇到用户对海量数据做计算的时候，就很容易导致heap吃紧，集群频繁GC，根本无法完成计算过程。 ES2.0以后，正式默认启用Doc Values特性(1.x需要手动更改mapping开启)，将field data在indexing time构建在磁盘上，经过一系列优化，可以达到比之前采用field data cache机制更好的性能。因此需要限制对field data cache的使用，最好是完全不用，可以极大释放heap压力。这里需要注意的是，排序、聚合字段必须为not analyzed。 设想如果有一个字段是analyzed过的，排序的实际对象其实是词典，在数据量很大情况下这种情况非常致命。

Bulk Queue

Bulk Queue是做什么用的？当所有的bulk thread都在忙，无法响应新的bulk request的时候，将request在内存里排列起来，然后慢慢清掉。一般来说，Bulk queue不会消耗很多的heap，但是见过一些用户为了提高bulk的速度，客户端设置了很大的并发量，并且将bulk Queue设置到不可思议的大，比如好几千。这在应对短暂的请求爆发的时候有用，但是如果集群本身索引速度一直跟不上，设置的好几千的queue都满了会是什么状况呢？ 取决于一个bulk的数据量大小，乘上queue的大小，heap很有可能就不够用，内存溢出了。一般来说官方默认的thread pool设置已经能很好的工作了，建议不要随意去“调优”相关的设置，很多时候都是适得其反的效果。

Indexing Buffer

Indexing Buffer是用来缓存新数据，当其满了或者refresh/flush interval到了，就会以segment file的形式写入到磁盘。 这个参数的默认值是10% heap size。根据经验，这个默认值也能够很好的工作，应对很大的索引吞吐量。 但有些用户认为这个buffer越大吞吐量越高，因此见过有用户将其设置为40%的。到了极端的情况，写入速度很高的时候，40%都被占用，导致OOM。

Cluster State Buffer

ES被设计成每个Node都可以响应用户的api请求，因此每个Node的内存里都包含有一份集群状态的拷贝。这个Cluster state包含诸如集群有多少个Node，多少个index，每个index的mapping是什么？有少shard，每个shard的分配情况等等 (ES有各类stats api获取这类数据)。 在一个规模很大的集群，这个状态信息可能会非常大的，耗用的内存空间就不可忽视了。并且在ES2.0之前的版本，state的更新是由Master Node做完以后全量散播到其他结点的。 频繁的状态更新都有可能给heap带来压力。 在超大规模集群的情况下，可以考虑分集群并通过tribe node连接做到对用户api的透明，这样可以保证每个集群里的state信息不会膨胀得过大。

超大搜索聚合结果集的fetch

ES是分布式搜索引擎，搜索和聚合计算除了在各个data node并行计算以外，还需要将结果返回给汇总节点进行汇总和排序后再返回。无论是搜索，还是聚合，如果返回结果的size设置过大，都会给heap造成很大的压力，特别是数据汇聚节点。超大的size多数情况下都是用户用例不对，比如本来是想计算cardinality，却用了terms aggregation + size:0这样的方式; 对大结果集做深度分页；一次性拉取全量数据等等。

在开发与维护过程中我们总结出以下优化建议：

1. 尽量运行在Sun/Oracle JDK1.7以上环境中，低版本的jdk容易出现莫名的bug，ES性能体现在在分布式计算中，一个节点是不足以测试出其性能，一个生产系统至少在三个节点以上。
2. 倒排词典的索引需要常驻内存，无法GC，需要监控data node上segment memory增长趋势。
3. 根据机器数，磁盘数，索引大小等硬件环境，根据测试结果，设置最优的分片数和备份数，单个分片最好不超过10GB，定期删除不用的索引，做好冷数据的迁移。
4. 保守配置内存限制参数，尽量使用doc value存储以减少内存消耗，查询时限制size、from参数。
5. 如果不使用_all字段最好关闭这个属性，否则在创建索引和增大索引大小的时候会使用额外更多的CPU，如果你不受限CPU计算能力可以选择压缩文档的_source。这实际上就是整行日志，所以开启压缩可以减小索引大小。
6. 避免返回大量结果集的搜索与聚合。缺失需要大量拉取数据可以采用scan & scroll api来实现。
7. 熟悉各类缓存作用，如field cache, filter cache, indexing cache, bulk queue等等，要设置合理的大小，并且要应该根据最坏的情况来看heap是否够用。
8. 必须结合实际应用场景，并对集群使用情况做持续的监控。

来源：开源技术社区