Elasticsearch Service

常见问题

产品文档

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档著作权归腾讯云单独所有，未经腾讯云事先书面许可，任何主体不得以任何形式复制、修改、抄袭、传播全部或部分本文档内容。

【商标声明】

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。

【服务声明】

本文档意在向客户介绍腾讯云全部或部分产品、服务的当时的整体概况，部分产品、服务的内容可能有所调整。您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

文档目录

常见问题
产品相关问题
集群异常问题
概述
集群健康状态异常（RED、YELLOW）如何解决？
集群熔断问题如何解决？
写入拒绝或查询拒绝问题如何解决？
集群整体 CPU 使用率过高问题如何解决？
集群磁盘使用率高和 read_only 状态问题如何解决？
集群负载不均的问题如何解决？

版权所有：腾讯云计算（北京）有限责任公司

常见问题
产品相关问题

最近更新时间：2021-07-13 17:12:07

什么是 Elasticsearch？

Elasticsearch 是分布式、可扩展、实时的搜索与数据分析引擎，建立在全文搜索引擎库 Apache Lucene™ 基础之上，支持从单个节点到上百个节点的任意扩展，并支持多达 PB 级别的结构化或者非结构化数据存储和查询。

ES 适用于哪些业务场景？

腾讯云 ES 完全保持了 Elasticsearch 在海量数据检索方面的特性，拥有全文检索、准实时搜索、结构化搜索等能力，广泛应用于日志分析、站内搜索等业务场景。使用腾讯云 ES，用户可以构建其它在用云产品（如 CVM、容器等）的日志服务系统，也可以集成搜索服务到现有的业务服务框架中。

什么是 Elasticsearch Service（ES）？

腾讯云 ES 是基于开源搜索引擎 Elasticsearch 构建的云端托管 Elasticsearch 集群服务，完全保持了 Elasticsearch 本身的开放、兼容和易用特性，并提供了充足的硬件资源，便捷的图形化创建和管理工具，以及技术支持和运维支持。

有未支付的转换订单，升级了集群的配置，转换订单还有效吗？

无效，因为按量付费转换包年包月时会生成一个新购订单，如果您在订单未支付前变更了集群的配置，新购订单金额与集群不匹配，未支付订单会被禁止支付。

如果您仍然需要转换集群的付费方式，需要先在 订单中心 取消当前未支付订单，再执行新的转换操作。

在成功购买后，是否支持更换云硬盘的类型？

目前暂不支持云硬盘在不同类型之间进行切换，您可以对数据进行快照备份后，通过快照创建您需要的新类型云硬盘。

创建集群时，配置多大规格的节点和磁盘合适？

ES 服务的节点规格情况计算可参考 集群规格和容量配置评估。

版权所有：腾讯云计算（北京）有限责任公司

集群异常问题

概述

最近更新时间：2021-08-11 11:17:23

当您使用腾讯云 Elasticsearch Service 时，遇到相关错误或问题，可根据本节中的问题分类匹配问题场景和解决方案。

- 集群健康状态异常（RED、YELLOW）如何解决？
- 集群熔断问题如何解决？
- 写入拒绝或查询拒绝问题如何解决？
- 集群整体 CPU 使用率过高问题如何解决？
- 集群磁盘使用率高和 read_only 状态问题如何解决？
- 集群负载不均的问题如何解决？

版权所有：腾讯云计算（北京）有限责任公司

集群健康状态异常（RED、YELLOW）如何解决？

最近更新时间：2021-08-11 11:17:23

集群状态为什么会异常？

集群状态在什么情况下发生 RED 和 YELLOW：

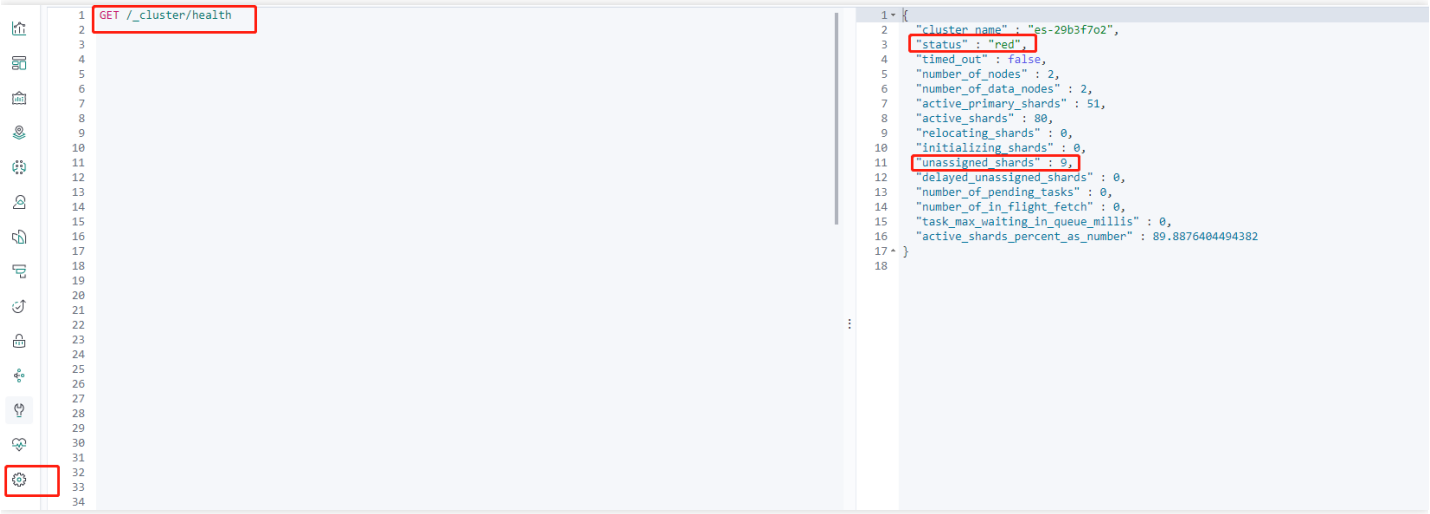
- 当集群存在未分配的**主索引分片**，集群状态会为 RED。该情况影响索引读写，需要重点关注。
- 当集群所有主索引分片都是已分配的，但是存在未分配的**副本索引分片**，集群状态则会 YELLOW。该情况不影响索引读写，一般会自动恢复。

查看集群状态

使用 kibana 开发工具，查看集群状态：

```
GET /_cluster/health
```

这里可以看到，当前集群状态为 red，有9个未分配的分片。



ES 健康接口返回内容官方解释：

指标	含义
cluster_name	集群的名称

指标	含义
status	集群的运行状况，基于其主要和副本分片的状态。状态为： - green：所有分片均已分配 - yellow：所有主分片均已分配，但未分配一个或多个副本分片。如果群集中的某个节点发生故障，则在修复该节点之前，某些数据可能不可用 - red：未分配一个或多个主分片，因此某些数据不可用。在集群启动期间，这可能会短暂发生，因为已分配了主要分片
timed_out	如果 false 响应在 timeout 参数指定的时间段内返回（30s默认情况下）
number_of_nodes	集群中的节点数
number_of_data_nodes	作为专用数据节点的节点数
active_primary_shards	活动主分区的数量
active_shards	活动主分区和副本分区的总数
relocating_shards	正在重定位的分片的数量
initializing_shards	正在初始化的分片数
unassigned_shards	未分配的分片数
delayed_unassigned_shards	其分配因超时设置而延迟的分片数
number_of_pending_tasks	尚未执行的集群级别更改的数量
number_of_in_flight_fetch	未完成的访存数量
task_max_waiting_in_queue_millis	自最早的初始化任务等待执行以来的时间（以毫秒为单位）
active_shards_percent_as_number	群集中活动碎片的比率，以百分比表示

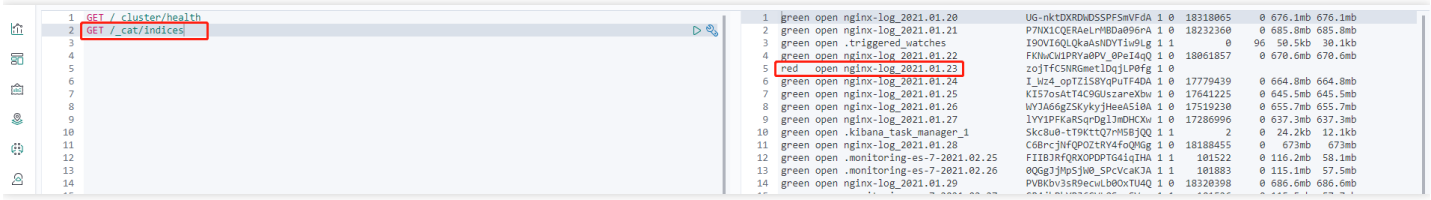
问题分析

当集群状态异常时，需要重点关注 unassigned_shards 没有正常分配的分片，这里举例说明其中一种场景。

找到异常索引

查看索引情况，并根据返回找到状态异常的索引。

```
GET /_cat/indices
```



查看详细的异常信息

```
GET /_cluster/allocation/explain
```

这里通过异常信息可以看出：

1. 主分片当前处于未分配状态（ `current_state` ），发生这个问题的原因是因为分配了该分片的节点已从集群中离开（ `unassigned_info.reason` ）。
2. 上述问题发生后，分片无法自动分配分片的原因是集群中没有该分片的可用副本（ `can_allocate` ）。
3. 同时也给出了更详细的信息（ `allocate_explanation` ）。

这种情况发生的原因是因为集群有节点下线，导致主分片已没有任何可用的分片数据，当前唯一能做的事就是等待节点恢复并重新加入集群。

> 注意：
> 某些极端场景，例如单副本集群的分片发生了损坏，或是文件系统故障导致该节点被永久移除，而此时只能接受数据丢失的事实，并通过 reroute commends 来重新分配空的主分片。为了尽量避免这种极端的场景，建议合理设计索引分片，不要给索引设置单副本。这里所谓的单副本，指的是索引有主分片，但没有副本分片，或称之为0副本。合理设计索引分片，可以将集群的总分片控制在一个很健康的规模，可以在保证高可用的情况下更加充分地利用集群分布式的特性，提高集群整体性能。

分片未分配（ `unassigned_info.reason` ）的所有可能

可通过如下分析方式初步判断集群产生未分配分片的原因，一般都可以在 allocation explain api 中得到想要的答案。

> 说明：
> 集群状态如果长时间未自动恢复，或是无法解决，则需要 提交工单，联系腾讯云技术支持。

reason	原因
INDEX_CREATED	索引创建，由于 API 创建索引而未分配的
CLUSTER_RECOVERED	集群恢复，由于整个集群恢复而未分配
INDEX_REOPENED	索引重新打开
DANGLING_INDEX_IMPORTED	导入危险的索引
NEW_INDEX_RESTORED	重新恢复一个新索引
EXISTING_INDEX_RESTORED	重新恢复一个已关闭的索引
REPLICA_ADDED	添加副本
ALLOCATION_FAILED	分配分片失败
NODE_LEFT	集群中节点丢失
REROUTE_CANCELLED	reroute 命令取消
REINITIALIZED	重新初始化
REALLOCATED_REPLICA	重新分配副本

版权所有：腾讯云计算（北京）有限责任公司

集群熔断问题如何解决？

最近更新时间：2021-08-11 11:17:23

什么是熔断

Elasticsearch Service 提供了多种官方的熔断器（circuit breaker），用于防止内存使用过高导致 ES 集群因为 OutOfMemoryError 而出现问题。Elasticsearch 设置有各种类型的子熔断器，负责特定请求处理的内存限制。此外，还有一个父熔断器，用于限制所有子熔断器上使用的内存总量。

> 说明：
> 出现熔断说明当前节点 JVM 使用率过高，通过熔断保护进程不会 OOM。此时可以通过适当降低读写、清理内存等方法降低节点负载，也可以通过升级节点内存规格来提高 JVM 大小。

常用的内存清理方法

- 清理 fielddata cache：在 text 类型的字段上进行聚合和排序时会使用 fileddata 数据结构，可能占用较大内存。可以在 Kibana 界面的【Dev Tools】中使用如下命令查看索引的 fielddata 内存占用：

  ```
  GET /_cat/indices?v&h=index,fielddata.memory_size&s=fielddata.memory_size:desc
  ```

  若 fielddata 占用内存过高，可以在 Kibana 界面的【Dev Tools】中使用如下命令清理 fielddata：

  ```
  POST /${fielddata占用内存较高的索引}/_cache/clear?fielddata=true
  ```

- 清理 segment：每个 segment 的 FST 结构都会被加载到内存中，并且这些内存是不会被 GC 回收的。因此如果索引的 segment 数量过大，也会导致内存使用率较高。可以在 Kibana 界面的【Dev Tools】中使用如下命令查看各节点的 segment 数量和占用内存大小：

  ```
  GET /_cat/nodes?v&h=segments.count,segments.memory&s=segments.memory:desc
  ```

  若 segment 占用内存过高，可以通过删除部分不用的索引、关闭索引，或定期合并不再更新的索引等方式缓解。
- 扩容集群：如果您清理内存后，仍频繁触发熔断，说明您的集群规模已经不匹配于您的业务负载，最好的方式是扩大集群规模，具体可参考 扩容集群。

熔断器介绍

Elasticsearch 官方熔断器

- 父熔断器（Parent circuit breaker）
  父熔断器限制所有子熔断器上使用的内存总量，当触发父熔断器熔断时，可能的日志信息如下：

```
Caused by: org.elasticsearch.common.breaker.CircuitBreakingException: [parent]
Data too large, data for [<transport_request>] would be [1749436147/1.6gb], whi
ch is larger than the limit of [1622605824/1.5gb], real usage: [1749435872/1.6g
b], new bytes reserved: [275/275b]
```

- Field data 熔断器（Field data breaker）
  当对 text 字段聚合或排序时，会产生 Field data 数据结构。Field data 熔断器会预估有多少数据被加载到内存中。当预估的数据占用内存到达 Field data 熔断器阈值时，会触发 Field data 熔断器熔断。此时可能的日志信息如下：

```
org.elasticsearch.common.breaker.CircuitBreakingException: [fielddata] Data too
large, data for [_id] would be [943928680/900.2mb], which is larger than the li
mit of [255606128/243.7mb]
```

- In flight 请求熔断器（In flight requests circuit breaker）
  In flight 请求熔断器限制了在 transport 和 HTTP 层的所有当前传入的请求所使用的内存。当触发 In flight 请求熔断器时，可能的日志信息如下：

```
[o.e.x.m.e.l.LocalExporter] [1611816935001404932] unexpected error while indexi
ng monitoring document
org.elasticsearch.xpack.monitoring.exporter.ExportException: RemoteTransportExc
eption[[1611816935001404732][9.10.153.16:9300][indices:data/write/bulk[s]]]; ne
sted: CircuitBreakingException[[in_flight_requests] Data too large, data for [<
transport_request>] would be [19491363612/18.1gb], which is larger than the lim
it of [17066491904/15.8gb]];
```

腾讯云 ES 自研熔断器

官方熔断机制的一个不足是仅跟踪那些经常会出问题的请求来预估内存的使用，而无法根据当前节点的实际内存使用状态，来限制请求的内存使用或触发熔断。在腾讯云 ES 中，开发了针对 JVM OLD 区内存使用率的自研熔断器来解决这个问题。

腾讯云 ES 的自研熔断器监控 JVM OLD 区的使用率，当使用率超过 `85%` 时开始拒绝写入请求，若 GC 仍无法回收 JVM OLD 区中的内存，在使用率到达 `90%` 时将拒绝查询请求。当请求被拒绝时，客户端将收到如下的响应：

```
{
"status": 403,
"error": {
"root_cause": [{
"reason": "pressure too high, (smooth) bulk request circuit break",
"type": "status_exception"
```

```
}],
"type": "status_exception",
"reason": "pressure too high, (smooth) bulk request circuit break"
}
}
```

写入拒绝或查询拒绝问题如何解决？

最近更新时间：2021-08-11 11:17:23

问题现象

集群在某些情况下会出现写入拒绝率（bulk reject）/查询拒绝率（search reject）增大的现象，具体表现为 bulk 写入/查询时，会有类似以下报错：
报错一：

```
[2019-03-01 10:09:58][ERROR]rspItemError: {"reason":"rejected execution of org.elasticsearch.transport.TransportService$7@5436e129 on EsThreadPoolExecutor[bulk, queue capacity = 1024, org.elasticsearch.common.util.concurrent.EsThreadPoolExecutor@6bd77359[Running, pool size = 12, active threads = 12, queued tasks = 2390, completed tasks = 20018208656]]","type":"es_rejected_execution_exception"}
```

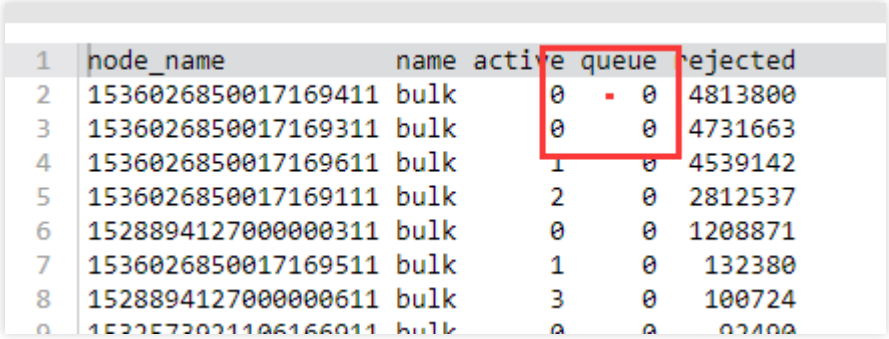
报错二：

```
[o.e.a.s.TransportSearchAction] [1590724712002574732] [31691361796] Failed to execute fetch phase
org.elasticsearch.transport.RemoteTransportException: [1585899116000088832][10.0.134.65:22624][indices:data/read/search[phase/fetch/id]]
Caused by: org.elasticsearch.common.util.concurrent.EsRejectedExecutionException: rejected execution of org.elasticsearch.common.util.concurrent.TimedRunnable@63779fac on QueueResizingEsThreadPoolExecutor[name = 1585899116000088832/search, queue capacity = 1000, min queue capacity = 1000, max queue capacity = 1000, frame size = 2000, targeted response rate = 1s, task execution EWMA = 2.8ms, adjustment amount = 50, org.elasticsearch.common.util.concurrent.QueueResizingEsThreadPoolExecutor@350da023[Running, pool size = 49, active threads = 49, queued tasks = 1000, completed tasks = 57087199564]]
```

- 可在云监控中看到集群写入/查询拒绝率增大。
- 也可在 kibana 控制台，通过命令查看正在拒绝或者历史拒绝的个数。

```
GET _cat/thread_pool/bulk?s=queue:desc&v
GET _cat/thread_pool/search?s=queue:desc&v
```

一般默认队列是1024，如果 queue 下有1024，说明这个节点有 reject 的现象。

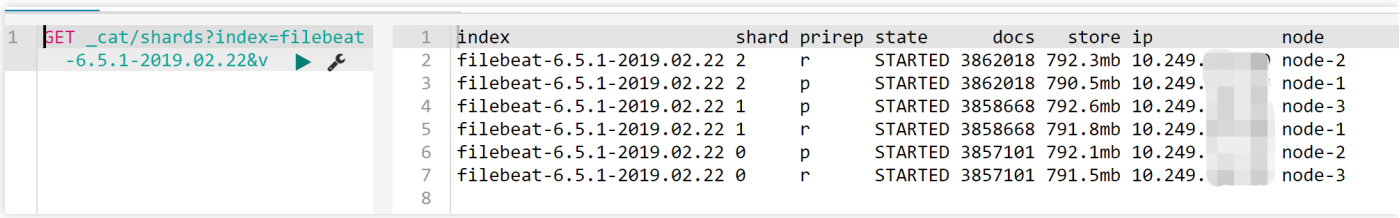


node_name	name	active	queue	rejected
1536026850017169411	bulk	0	0	4813800
1536026850017169311	bulk	0	0	4731663
1536026850017169611	bulk	1	0	4539142
1536026850017169111	bulk	2	0	2812537
1528894127000000311	bulk	0	0	1208871
1536026850017169511	bulk	1	0	132380
1528894127000000611	bulk	3	0	100724

问题定位

1. 检查 bulk 请求的 body 大小是否不合理。单个 bulk 请求的大小在10MB以内比较合适，如过大，则会导致单个 bulk 请求处理时间过长，导致队列排满；如过小，则会导致 bulk 请求数过多，导致队列排满。
2. 检查写入 QPS 和集群配置是否匹配，经验值为在4C16G 3节点集群上分片分布均衡时可以承担约2W - 3W QPS 的写入，但如果还有较多的查询请求时 QPS 会更低，具体可以通过压测确定集群最高能承受的 QPS 写入量，选择合适的配置。
3. 检查分片（shard）数据量是否过大。分片数据量过大，有可能引起 Bulk Reject，建议单个分片大小控制在20GB - 50GB左右。可在 kibana 控制台，通过命令查看索引各个分片的大小。

```
GET _cat/shards?index=index_name&v
```



```
GET _cat/shards?index=filebeat-6.5.1-2019.02.22&v
```

index	shard	prirep	state	docs	store	ip	node
filebeat-6.5.1-2019.02.22	2	r	STARTED	3862018	792.3mb	10.249.	node-2
filebeat-6.5.1-2019.02.22	2	p	STARTED	3862018	790.5mb	10.249.	node-1
filebeat-6.5.1-2019.02.22	1	p	STARTED	3858668	792.6mb	10.249.	node-3
filebeat-6.5.1-2019.02.22	1	r	STARTED	3858668	791.8mb	10.249.	node-1
filebeat-6.5.1-2019.02.22	0	p	STARTED	3857101	792.1mb	10.249.	node-2
filebeat-6.5.1-2019.02.22	0	r	STARTED	3857101	791.5mb	10.249.	node-3

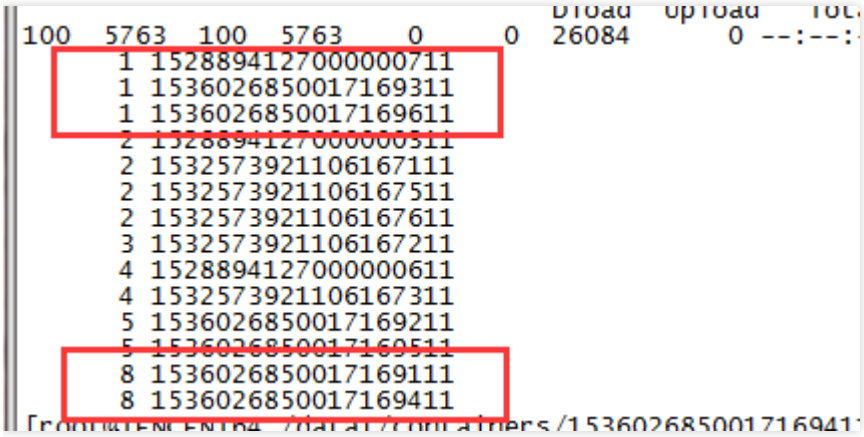
检查分片数是否分布不均匀

集群中的节点分片分布不均匀，有的节点分配的 shard 过多，有的分配的 shard 少。

- 在 ES 控制台 集群详情页的【集群监控】>【节点状态】查看，具体可参考 查看监控。
- 通过 ES API，查看集群各个节点的分片个数。

```
GET /_cat/shards?index={index_name}&s=node,store:desc
```

结果如下图（第一列为分片个数，第二列为节点 ID），有的节点分片为1，有的为8，分布极不均匀。



解决方案

1. 设置分片大小

   分片大小可以通过 index 模版下的 number_of_shards 参数进行配置（模板创建完成后，再次新创建索引时生效，老的索引不能调整）。
2. 调整分片数据不均匀

临时解决方案

如果发现集群有分片分配不均的现象，可通过设置 routing.allocation.total_shards_per_node 参数，动态调整某个 index 解决，详情可参考 Total Shards Per Node。

> 注意：
> total_shards_per_node 要留有一定的 buffer，防止机器故障导致分片无法分配（例如10台机器，索引有20个分片，则 total_shards_per_node 设置要大于2，可以取3）。

```
PUT {index_name}/_settings
{
"settings": {
"index": {
"routing": {
"allocation": {
"total_shards_per_node": "3"
}
}
}
```

```
}
}
```

索引生产前设置：通过索引模板，设置其在每个节点上的分片个数。

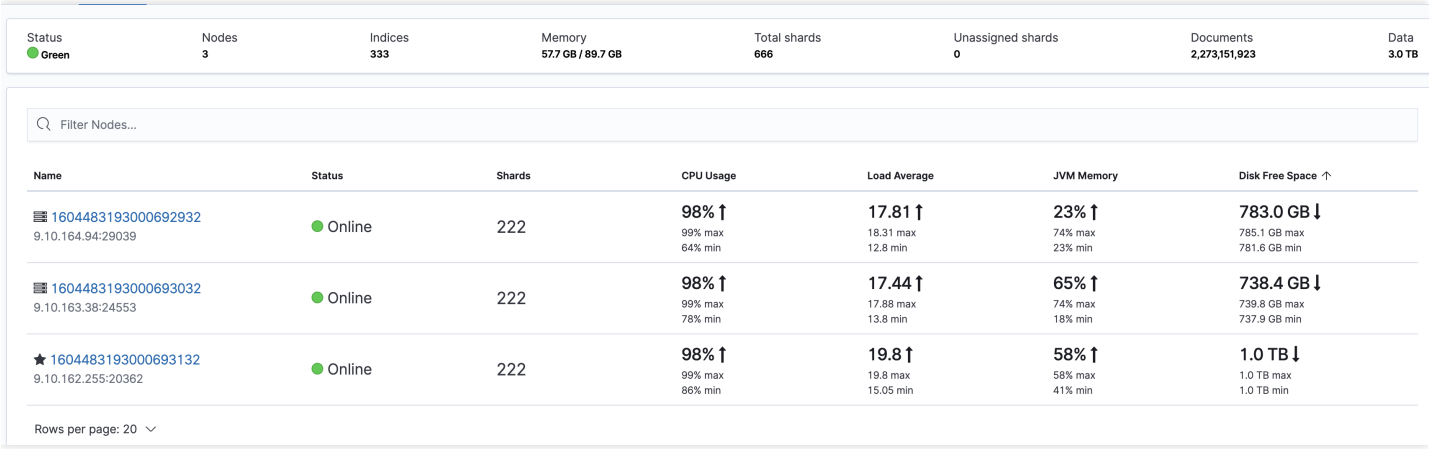
```
PUT _template/ {template_name}
{
"order": 0,
"template": " {index_prefix@} *", //要调整的 index 前缀
"settings": {
"index": {
"number_of_shards": "30", //指定 index 分配的 shard 数，可以根据一个 shard 30GB左右的空间来分配
"routing.allocation.total_shards_per_node":3 //指定一个节点最多容纳的 shards 数
}
},
"aliases": {}
}
```

集群整体 CPU 使用率过高问题如何解决？

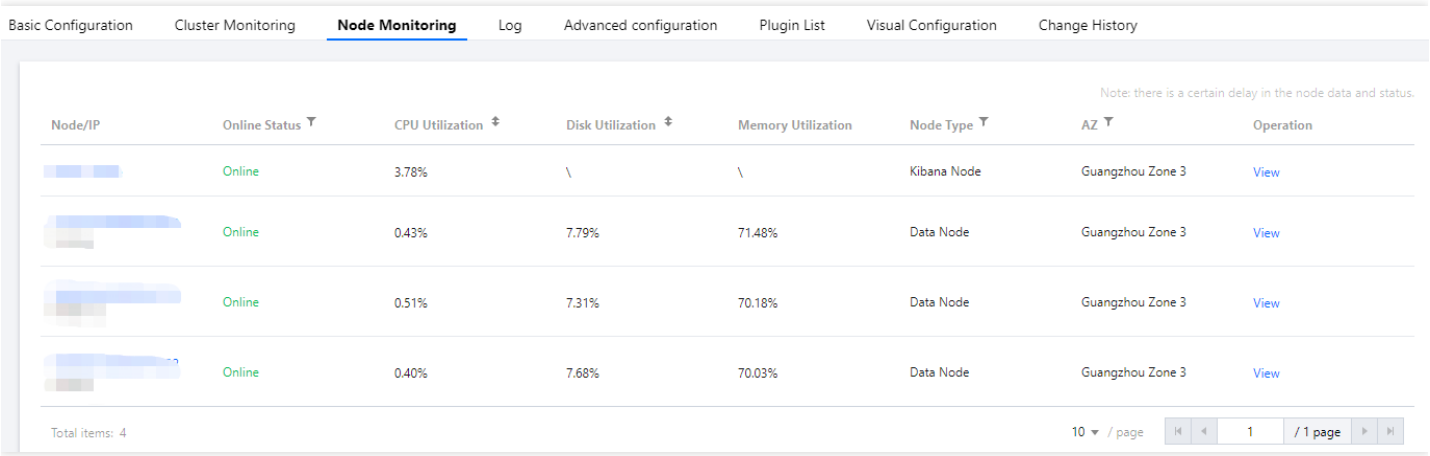
最近更新时间：2021-08-11 11:17:23

问题现象

集群所有节点 CPU 都很高，但读写都不是很高。具体表现可以从 kibana 端 Stack Monitoring 监控页面看到：



另外也可以从 ES 控制台 UI 的节点监控页面看到各节点的 CPU 使用率情况：



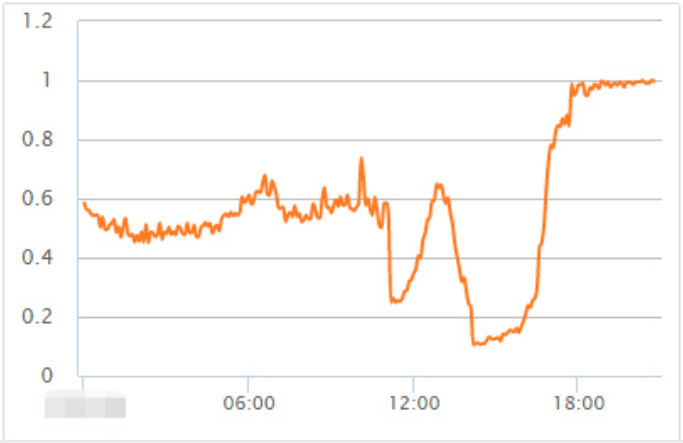
出现这种情况，由于表面上看集群读写都不高，导致很难快速从监控上找到根因。所以需要细心观察，从细节中找答案，下面我们介绍几种可能出现的场景以及排查思路。

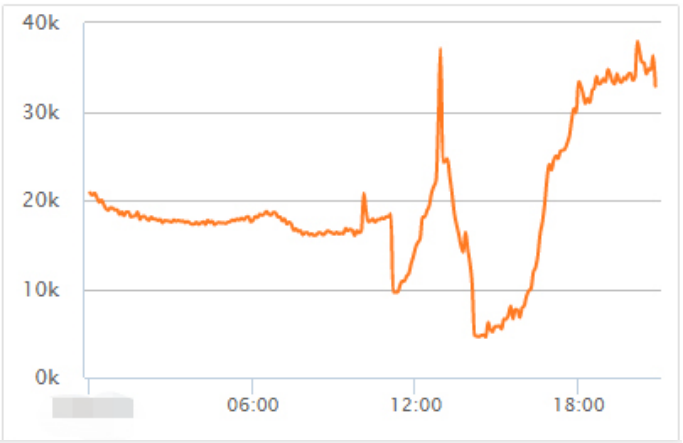
> 说明：
> 对于个别节点 CPU 使用率远高于其他节点，这种情况较为常见，多数是因为集群使用不当导致的负载不均，可参考 出现集群负载不均的问题如何解决。

问题定位和解决方案

查询请求较大导致 CPU 飙高

这种情况比较常见，可以从监控上找到线索。通过监控可以发现，查询请求量的波动与集群最大 CPU 使用率是基本吻合的。





进一步确认问题需要开启集群的慢日志收集，具体可参考 查询集群日志。从慢日志中，可以得到更多信息，例如引起慢查询的索引、查询参数以及内容。

解决方案

- 尽量避免大段文本搜索，优化查询。
- 通过慢日志确认查询慢的索引，对于一些数据量不大的索引，设置少量分片多副本，例如1分片多副本，以此来提高查询性能。

写入请求导致 CPU 飙高

通过监控来观察到 CPU 飙高与写入相关，然后开启集群的慢日志收集，确认写入慢的请求，进行优化。也可以通过获取 `hot_threads` 信息来确认什么线程在消耗 CPU：

```
curl http://9.15.49.78:9200/_nodes/hot_threads
```

```
   100.5% (502.4ms out of 500ms) cpu usage by thread 'elasticsearch[16044831930006929 32][write][T#14]'
     4/10 snapshots sharing following 35 elements
       java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
       java.util.regex.Pattern$CharProperty.match(Unknown Source)
       java.util.regex.Pattern$Curly.match0(Unknown Source)
       java.util.regex.Pattern$Curly.match(Unknown Source)
--
   99.9% (499.7ms out of 500ms) cpu usage by thread 'elasticsearch[16044831930006929 32][write][T#9]'
     10/10 snapshots sharing following 33 elements
       java.util.regex.Pattern$Curly.match0(Unknown Source)
       java.util.regex.Pattern$Curly.match(Unknown Source)
       java.util.regex.Pattern$GroupHead.match(Unknown Source)
       java.util.regex.Pattern$Start.match(Unknown Source)
--
   98.8% (493.9ms out of 500ms) cpu usage by thread 'elasticsearch[16044831930006929 32][write][T#16]'
     3/10 snapshots sharing following 34 elements
       java.util.regex.Pattern$BmpCharProperty.match(Unknown Source)
       java.util.regex.Pattern$Curly.match0(Unknown Source)
       java.util.regex.Pattern$Curly.match(Unknown Source)
       java.util.regex.Pattern$GroupHead.match(Unknown Source)
--
   99.9% (499.2ms out of 500ms) cpu usage by thread 'elasticsearch[16044831930006931 32][write][T#4]'
     7/10 snapshots sharing following 35 elements
       java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
       java.util.regex.Pattern$CharProperty.match(Unknown Source)
       java.util.regex.Pattern$Curly.match0(Unknown Source)
       java.util.regex.Pattern$Curly.match(Unknown Source)
--
   99.8% (499ms out of 500ms) cpu usage by thread 'elasticsearch[16044831930006931 32][write][T#2]'
     6/10 snapshots sharing following 35 elements
       java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
       java.util.regex.Pattern$CharProperty.match(Unknown Source)
       java.util.regex.Pattern$Curly.match0(Unknown Source)
       java.util.regex.Pattern$Curly.match(Unknown Source)
--
   99.7% (498.3ms out of 500ms) cpu usage by thread 'elasticsearch[16044831930006931 32][write][T#7]'
     3/10 snapshots sharing following 35 elements
       java.util.regex.Pattern$CharProperty$1.isSatisfiedBy(Unknown Source)
       java.util.regex.Pattern$CharProperty.match(Unknown Source)
:
```

例如，这里发现的是有大量 ingest pipeline 操作，而 ingest 操作是十分消耗资源的。

```
  java.util.regex.Matcher.find(Unknown Source)
  java.util.regex.Matcher.replaceAll(Unknown Source)
  org.elasticsearch.ingest.common.GsubProcessor.process(GsubProcessor.java:55)
  org.elasticsearch.ingest.common.GsubProcessor.process(GsubProcessor.java:32)
  org.elasticsearch.ingest.common.AbstractStringProcessor.execute(AbstractStringProcessor.java:69)
  org.elasticsearch.ingest.Processor.execute(Processor.java:50)
```

解决方案

如遇到上面这种问题，则需要业务方根据实际情况来优化。排查该类问题的关键点，在于善用集群的监控指标来快速判断问题的方向，再配合集群日志来定位问题的根因，才能快速地解决问题。

集群磁盘使用率高和 read_only 状态问题如何解决？

最近更新时间：2021-08-11 11:17:23

问题现象

当磁盘使用率超过85%，或者达到100%，会导致 Elasticsearch 集群或 Kibana 无法正常提供服务，可能会出现以下几种问题场景：

- 在进行索引请求时，返回类似 `{[FORBIDDEN/12/index read-only/allow delete(api)];","type":"cluster_block_exception"}` 的报错。

2. 在对集群进行操作时，返回类似 `[FORBIDDEN/13/cluster read-only / allow delete (api)]` 的报错。
3. 集群处于 Red 状态，严重情况下存在节点未加入集群的情况（可通过 `GET _cat/allocation?v` 命令查看），并且存在未分配的分片（可通过 `GET _cat/allocation?v` 命令查看）。
4. 通过 Elasticsearch 控制台的节点监控页面，集群节点磁盘使用率曾达到或者接近100%。

问题分析

上述问题是由于磁盘使用率过高所导致。数据节点的磁盘使用率存在以下三个水位线，超过水位线可能会影响 Elasticsearch 或 Kibana 服务。

- 当集群磁盘使用率超过85%：会导致新的分片无法分配。
- 当集群磁盘使用率超过90%：Elasticsearch 会尝试将对应节点中的分片迁移到其他磁盘使用率比较低的数据节点中。
- 当集群磁盘使用率超过95%：系统会对 Elasticsearch 集群中对应节点里每个索引强制设置 read_only_allow_delete 属性，此时该节点上的所有索引将无法写入数据，只能读取和删除对应索引。

解决方案

清理集群过期数据

1. 用户可以通过访问【Kibana】>【Dev Tools】删除过期索引释放磁盘空间。步骤如下：

> 警告：
> 数据删除后将无法恢复，请谨慎操作。您也可以选择保留数据，但需进行磁盘扩容。

第一步：开启集群索引批量操作权限。

```
PUT _cluster/settings
{
"persistent": {
"action.destructive_requires_name": "false"
}
}
```

第二步：删除数据，例如 `DELETE NginxLog-12*` 。

```
DELETE index-name-*
```

2. 执行完上述步骤后，如果用户腾讯云 Elasticsearch 的版本是7.5.1以前的版本，还需要在 Kibana 界面的【Dev Tools】中执行如下命令：
   - 关闭索引只读状态，执行如下命令：

     ```
     PUT _all/_settings
     {
     "index.blocks.read_only_allow_delete": null
     }
     ```

   - 关闭集群只读状态，执行如下命令：

     ```
     PUT _cluster/settings
     {
     "persistent": {
     "cluster.blocks.read_only_allow_delete": null
     }
     }
     ```

3. 查看集群索引是否依然为 `read_only` 状态，索引写入是否恢复正常。
4. 若集群是否依然为 Red 状态，执行以下命令，查看集群中是否存在未分配的分片。

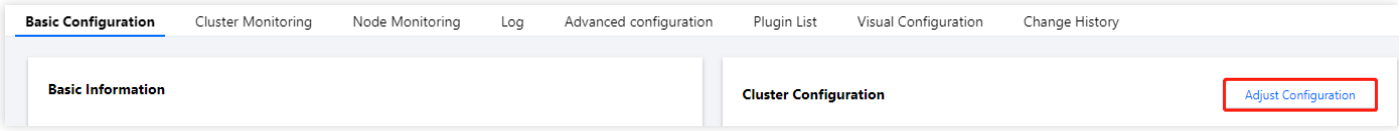
   ```
   GET /_cluster/allocation/explain
   ```

5. 等待分片下发完成后，查看集群状态。如果集群状态依然为 Red，请 提交工单，联系腾讯云技术支持。
6. 为避免磁盘使用率过高影响 Elasticsearch 服务，建议开启磁盘使用率监控报警，及时查收报警短信，提前做好防御措施，具体可参考 监控告警配置建议。

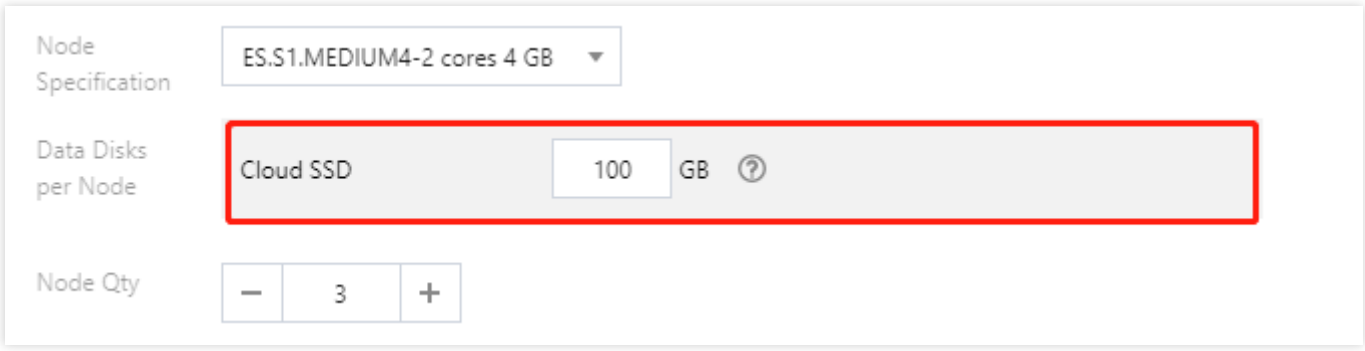
扩容云盘空间

若用户不想清理集群数据，也可以在腾讯云 ES 控制台 的集群配置界面，扩容磁盘空间。步骤如下：

1. 在集群列表中单击要配置的实例【ID/名称】，进入实例详情页，在【基础配置】页签中单击【调整配置】。



2. 进入调整配置页面，【单节点数据盘】选择您需要扩容的磁盘空间，单击【下一步】，提交任务即可。



3. 如果用户腾讯云 Elasticsearch 的版本是7.5.1以前的版本，还需要在 Kibana 界面的【Dev Tools】中执行如下命令：
   - 关闭索引只读状态，执行如下命令：

```
PUT _all/_settings
{
"index.blocks.read_only_allow_delete": null
}
```

   - 关闭集群只读状态，执行如下命令：

```
PUT _cluster/settings
{
"persistent": {
"cluster.blocks.read_only_allow_delete": null
}
}
```

集群负载不均的问题如何解决？

最近更新时间：2021-08-11 11:17:23

问题现象

集群在某些情况下会个别节点 CPU 使用率远高于其他节点的现象，具体表现为 ES 集群控制台节点监控上可以明显看到某些节点 CPU 使用率很高。

问题原因

- 索引分片设计不合理
- Segment 大小不均
- 存在典型的冷热数据需求场景

原因分析及解决方案

Shard 设置不合理

1. 登录 Kibana 控制台，在开发工具中执行以下命令，查看索引的 shard 信息，确认索引的 shard 在负载高的节点上呈现的数量较多，说明 shard 分配不均。

```
GET _cat/shards?v
```

2. 登录 Kibana 控制台，在开发工具中执行以下命令，查看索引信息。结合集群配置，确认存在节点 shard 分配不均的现象。

```
GET _cat/indices?v
```

3. 重新分配分片，合理规划 shard，确保主 shard 数与副 shard 数之和是集群数据节点的整数倍。

> 注意：
> Elasticsearch 在检索过程中也会检索 `.del` 文件，然后过滤标记有 `.del` 的文档，这会降低检索效率，耗费规格资源，建议在业务低峰期进行强制合并操作，具体请参见 force merge。

shard 规划建议

Shard 大小和数量是影响 Elasticsearch 集群稳定性和性能的重要因素之一。Elasticsearch 集群中任何一个索引都需要有一个合理的 shard 规划。合理的 shard 规划能够防止因业务不明确，导致分片庞大消耗 Elasticsearch 本身性能的问题。以下是 shard 规划时的几个建议：

- 尽量遵循索引单分片20g - 50g的设计原则。

2. 索引尽量增加时间后缀，按时间滚动，方便管理。
3. 在遵循单分片设计原则的前提下，预测出索引最终大小，并根据集群节点数设计索引分片数量，使分片尽量平均分布在各个节点。

> 注意：
> 主分片不是越多越好，因为主分片越多，Elasticsearch 性能开销也会越大。建议单节点 shard 总数按照单节点内存×30进行评估，如果 shard 数量太多，极易引起文件句柄耗尽，导致集群故障。

Segment 大小不均

1. 在查询 body 中添加 `"profile": true`，检查 test 索引是否存在某个 shard 查询时间比其他 shard 长。
2. 查询中分别指定 `preference=_primary` 和 `preference=_replica`，在 body 中添加 `"profile": true`，分别查看主副 shard 查询消耗的时间。检查较耗时的 shard 主要体现在主 shard 上还是副 shard 上。
3. 登录 Kibana 控制台，在开发工具中执行以下命令，查看 shard，并根据其中 segment 信息分析问题所在，确认负载不均与 segment 大小不均有关。

```
GET _cat/segments/index?v&amp;h=shard,segment,size,size.momery,ip
GET _cat/shards?v
```

4. 参考以下两种方法其中一种解决问题：
   - 在业务低峰期进行强制合并操作，具体请参见 force merge，将缓存中的 delete.doc 彻底删除，将小 segment 合并成大 segment。
   - 重启主 shard 所在节点，触发副 shard 升级为主 shard。并且重新生成副 shard，副 shard 复制新的主 shard 中的数据，保持主副 shard 的 segment 一致。

存在典型的冷热数据需求场景

如果查询中添加了 routing 或查询频率较高的热点数据，则必然导致数据出现负载不均。