Elasticsearch开发进阶指南

Lex Li
Elastic 解决方案架构师

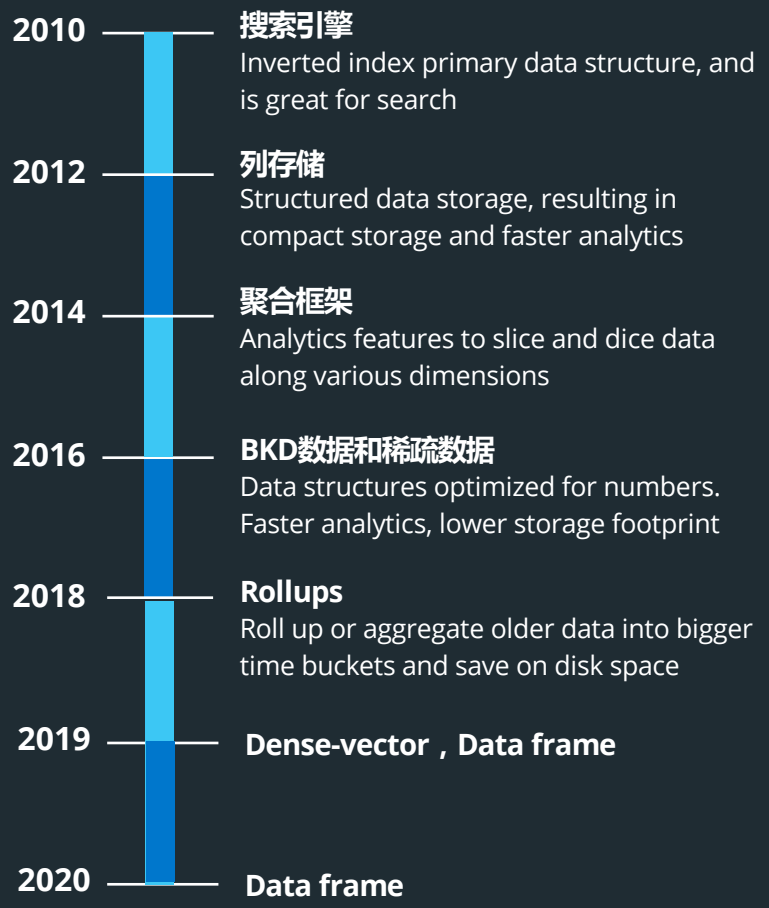
Elasticsearch开发进阶指南

本讲座，我们不谈运维，只谈开发

- 选择合适的ES版本
- 设计进阶
- 性能调优进阶

选择合适的ES版本

Lucene & Elasticsearch 付出的努力



应该选择哪个版本的Elasticsearch

选择最新的版本不会是一个错误的

- 仔细审查自己的需求，看看会用到哪些特性
  - 是否有大量的聚合需求，需对各种指标进行聚合？ Block-KD tree (> 5.0)
  - 随着索引数据持续增加，是否需要自动新建索引？ Roll over API(> 5.0)
  - 随着数据量的增加，是否需要动态的调整shard的数量？ split API (> 6.2)
  - 哪些用户需要使用Elasticsearch，是否需要支持SQL查询？ Elasticsearch SQL (> 6.3)
  - OOM问题？需要更精确计算的基于内存的断路器？ real-memory **circuit** breaker（> 7.0）
  - 更多的数据分析能力和搜索能力？Dataframe, Dense-Vector?（> 7.0）

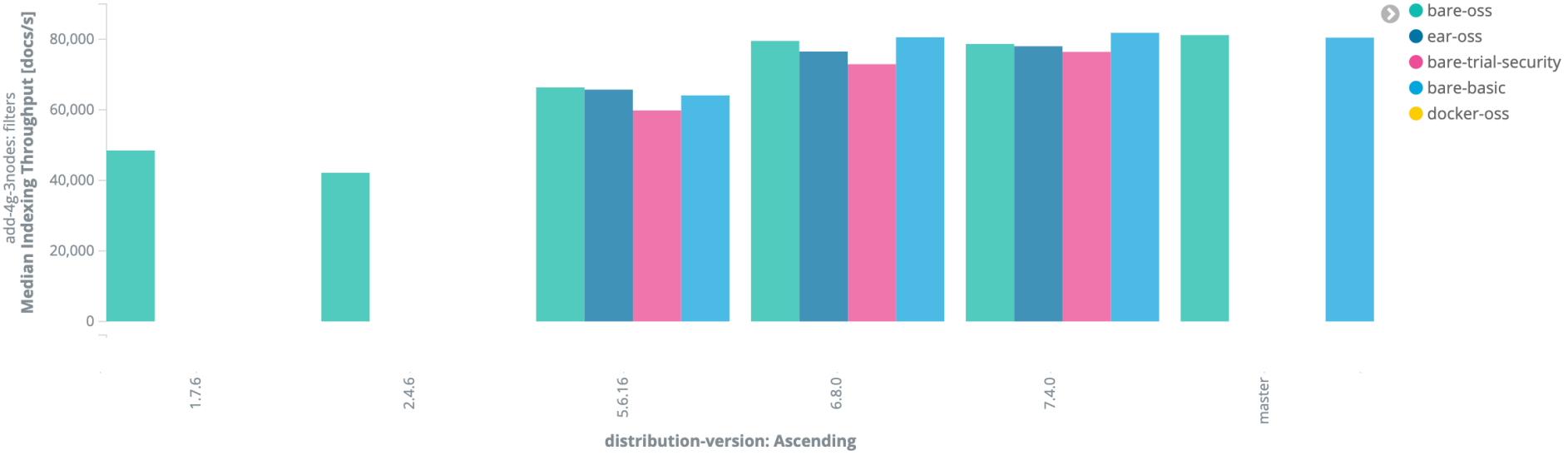
应该选择哪个版本的Elasticsearch

数据类型的偏好？Benchmark上查看各个版本的性能表现

- 用于评估结构化数据的性能
- 用于评估地理查询的性能。
- 用于评估渗透查询的性能。
- 用于评估全文搜索的性能。
- 用于评估高度结构化数据的性能。
- 用于评估嵌套文档的性能。
- 用于评估(Web)服务器日志的性能。
- 用于评估range field的性能。

elastic

- `bare` : Elasticsearch on an unencrypted drive
- `ear` : Elasticsearch on a drive that is encrypted with dm-crypt to benchmark the performance impact of encryption-at-rest.
- `docker` : Official Elasticsearch Docker image
- `oss` : Elasticsearch with Apache 2.0 license
- `basic` : Elasticsearch with commercial Elastic license; see x-pack/open.
- `trial-security` : Elasticsearch with X-Pack Security and TLS enabled.

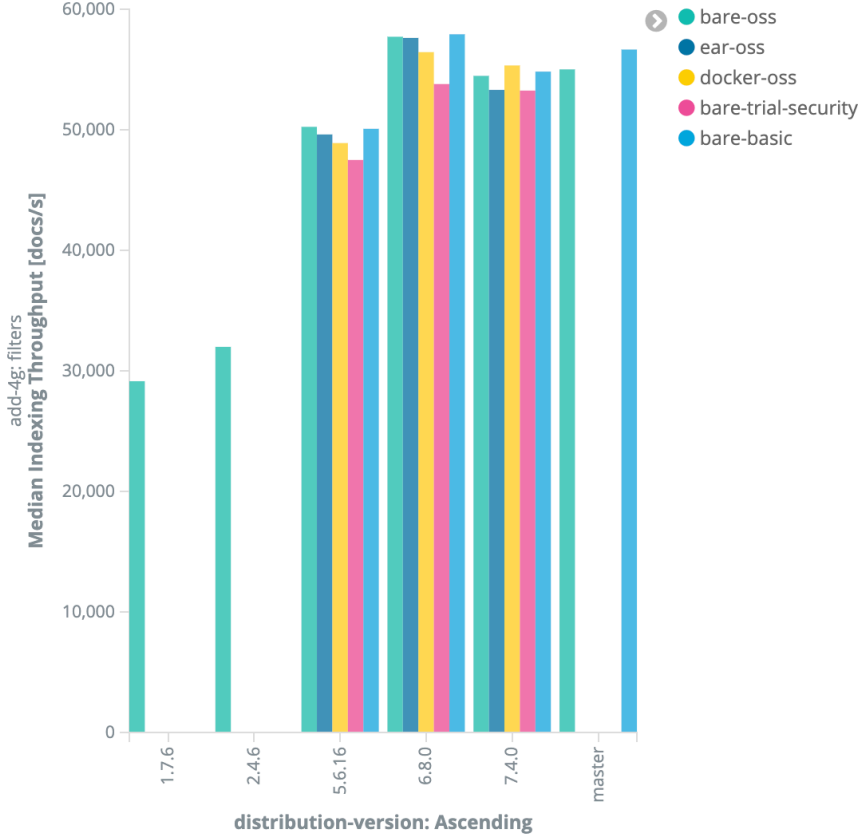


https://elasticsearch-

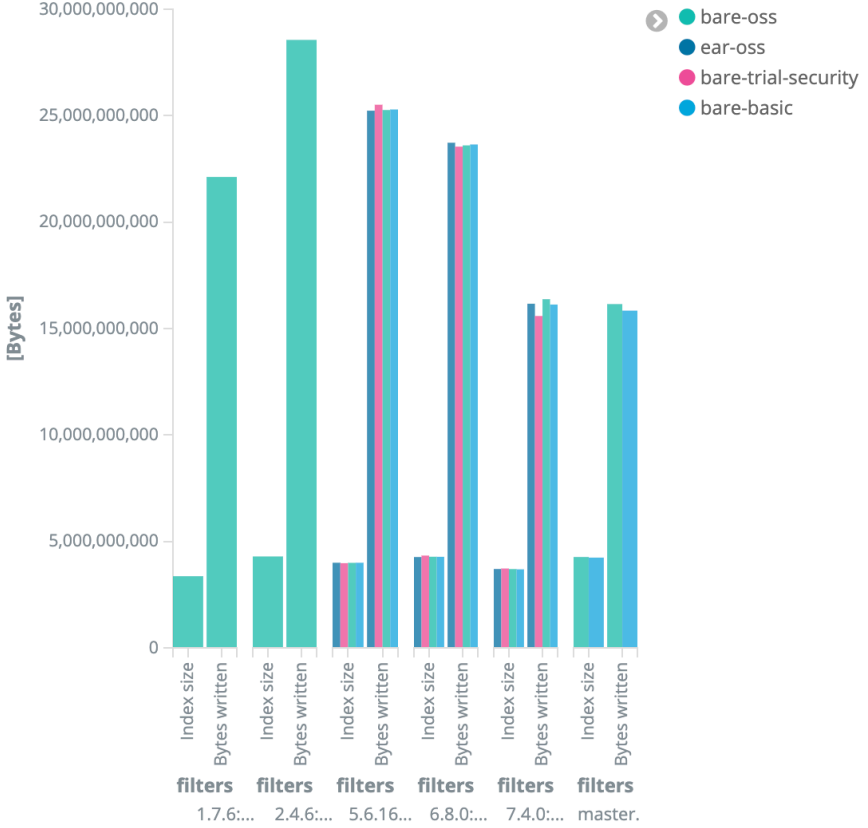
elastic

Nested Data Type

release-nested-indexing-throughput

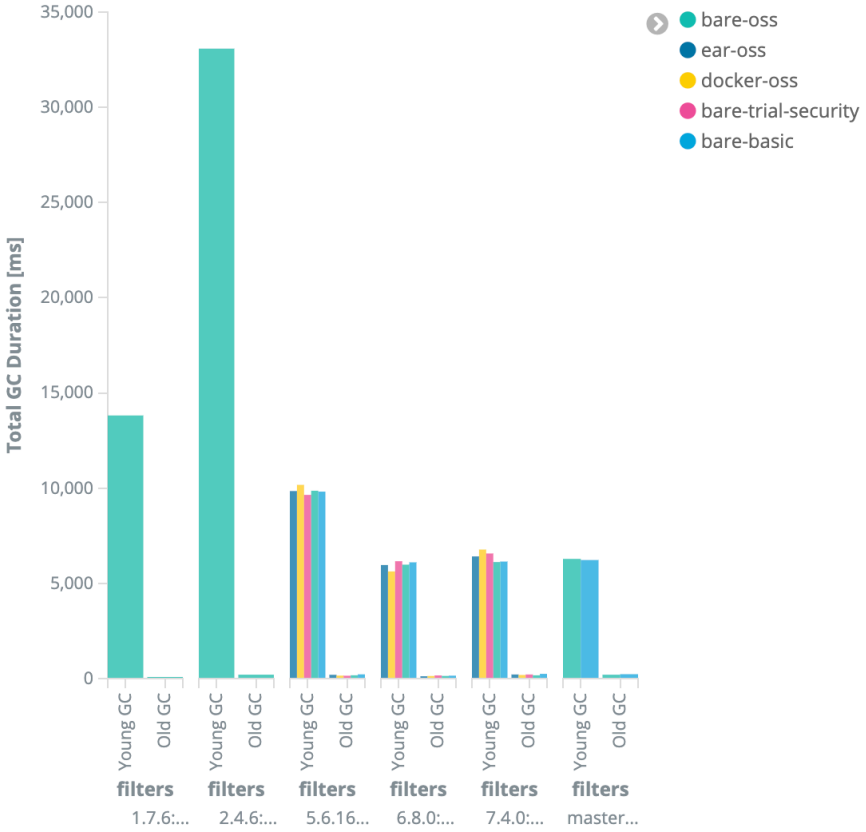


release-nested-add-4g-io

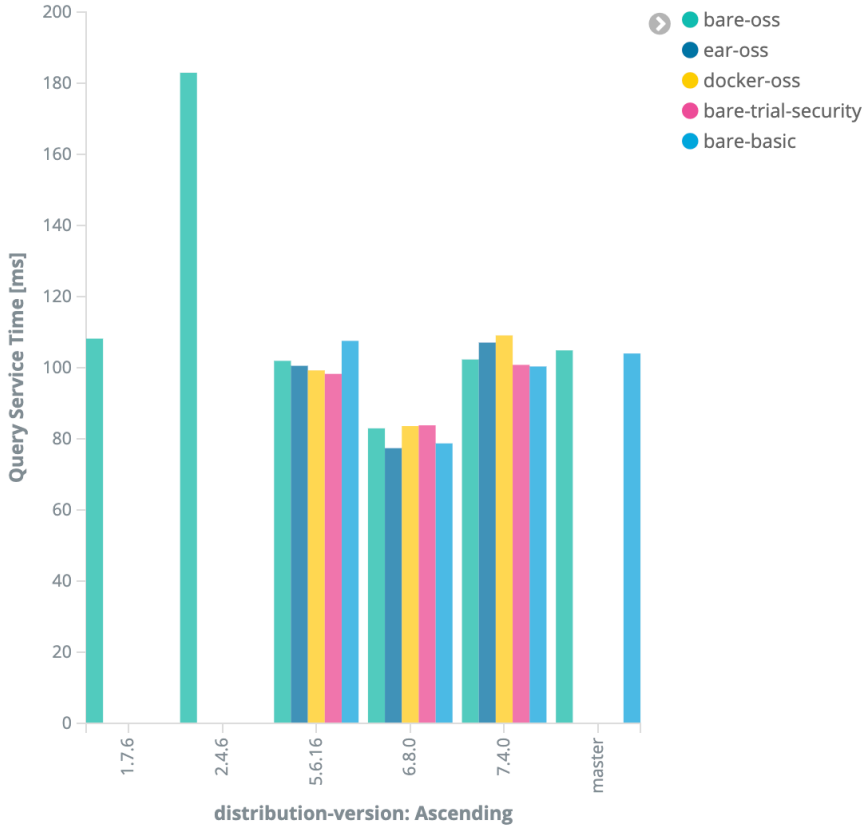


Nested Data Type

release-nested-add-4g-gc

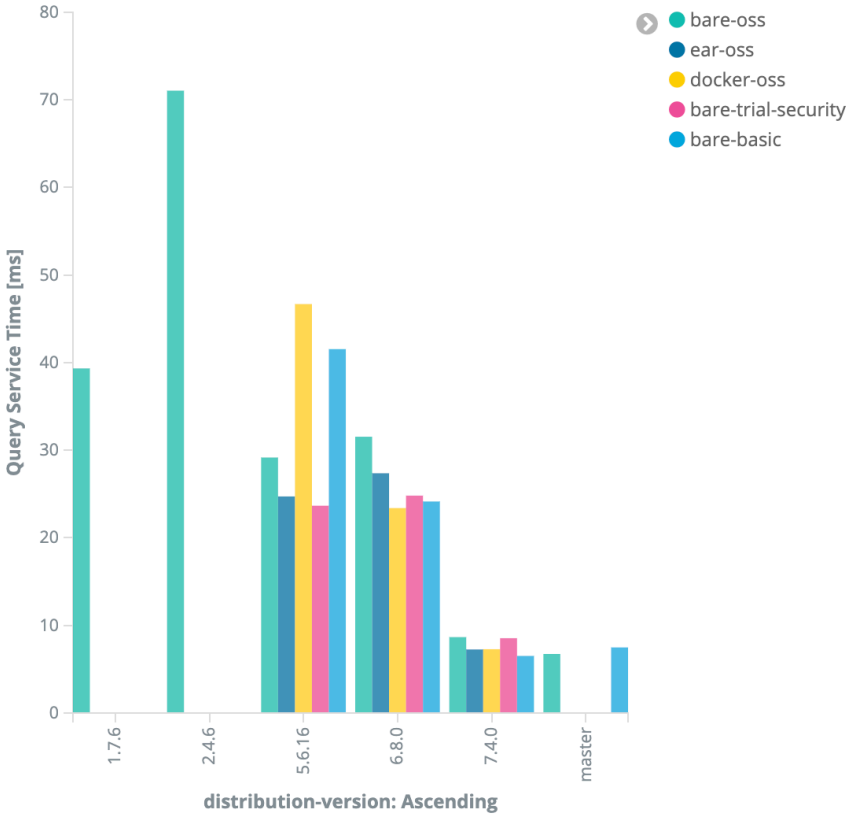


release-nested-add-4g-randomized-nested-queries-p99-service_time

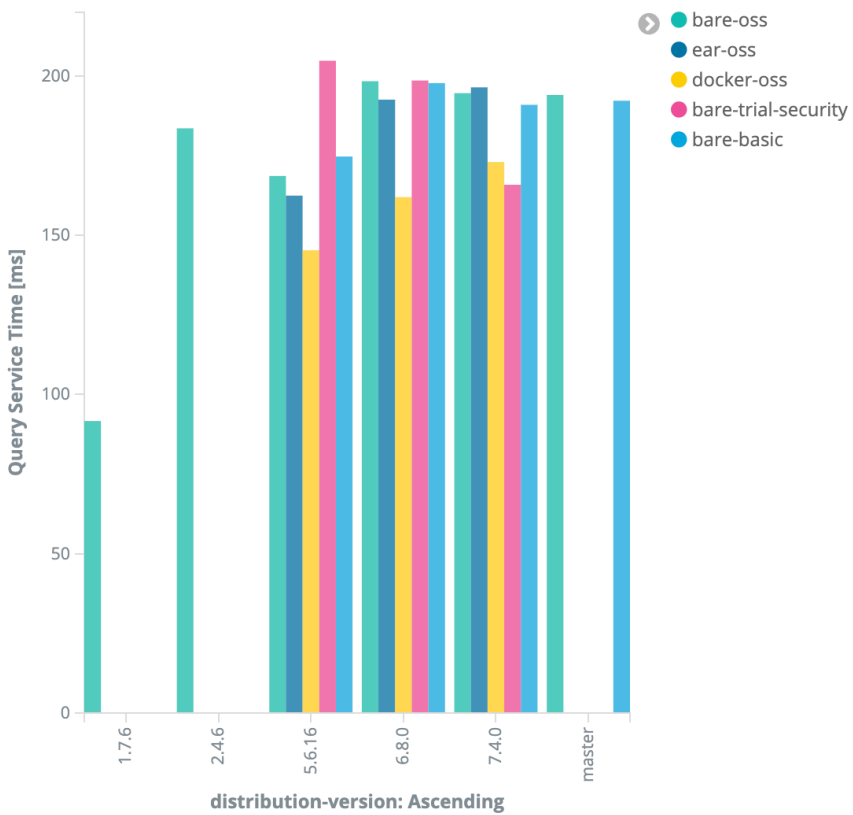


Nested Data Type

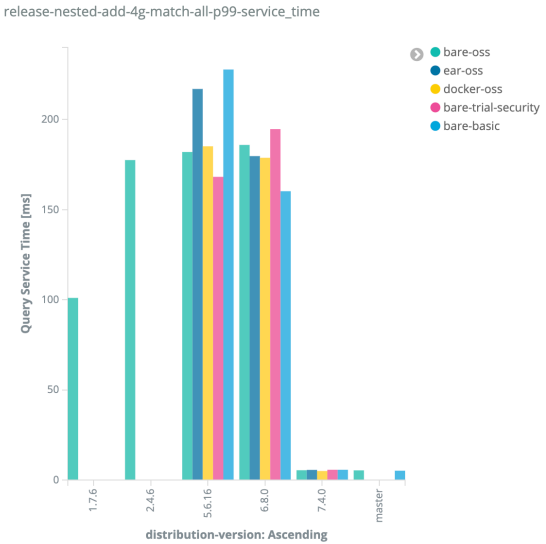
release-nested-add-4g-randomized-term-queries-p99-service_time

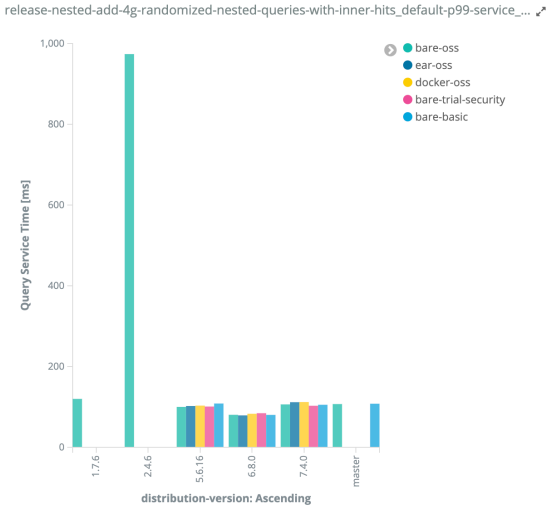


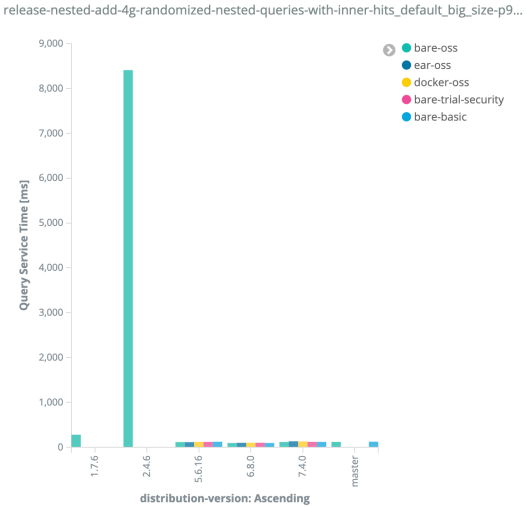
release-nested-add-4g-randomized-sorted-term-queries-p99-service_time



Nested Data Type







elastic

使用Rally为自己建立基线

- 使用和线上集群相同的硬件配置搭建单节点集群
- 使用和线上集群相同的mapping，创建0副本，1分片的索引
- 使用和线上集群相同的数据进行压测
- 观察写入的性能，并使用和业务类似的查询请求观察搜索和聚合的性能
- 压测数据持续导入ES集群，作为基线

benchmark

```
esrally --pipeline=from-distribution --distribution-version=1.7.6
esrally --pipeline=from-distribution --distribution-version=2.4.6
esrally --pipeline=from-distribution --distribution-version=5.6.1
esrally --pipeline=from-distribution --distribution-version=6.8.0
esrally --pipeline=from-distribution --distribution-version=7.4.0
```

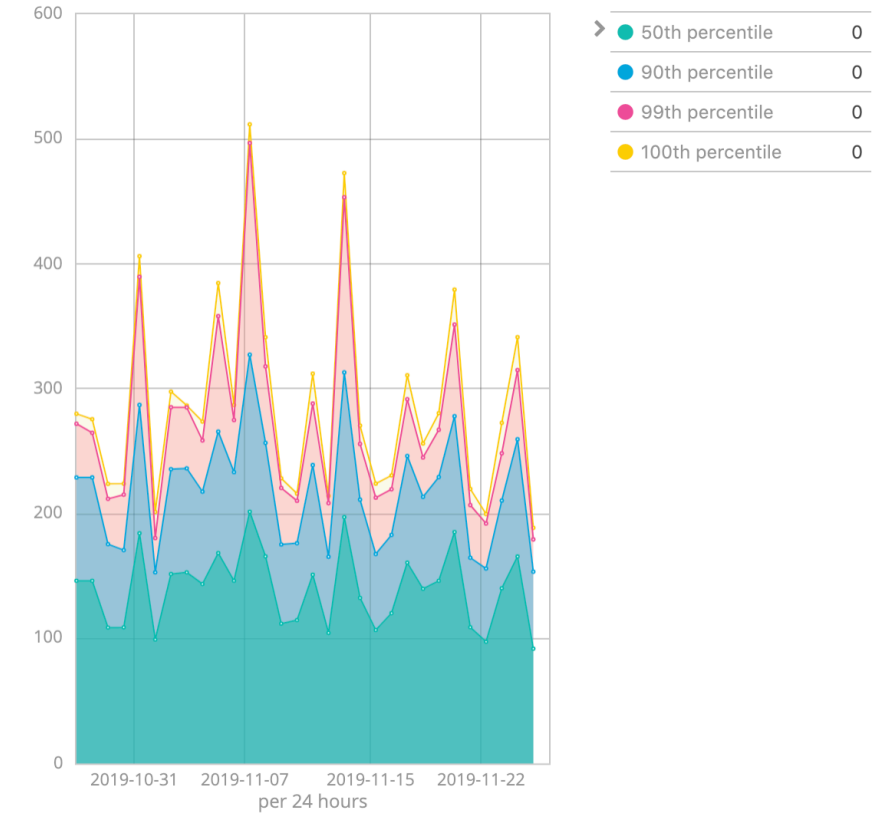
Release compare

https://esrally.readthedocs.io/

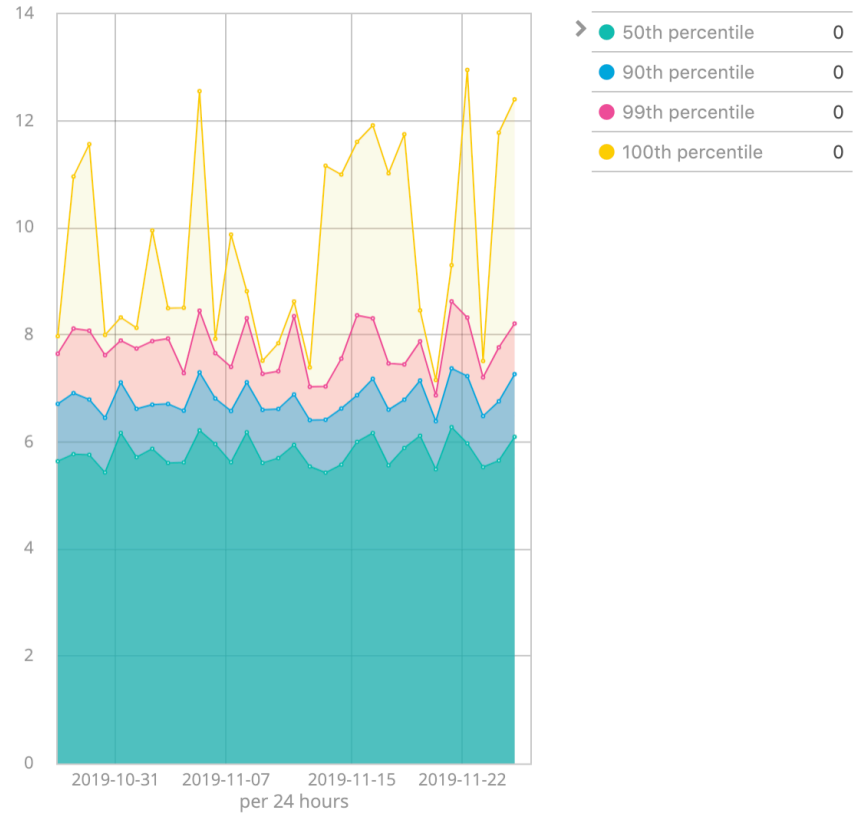
elastic

保证变动的影响可控

nightly-basic-nested-add-4g-randomized-sorted-term-queries-latency



nightly-basic-nested-add-4g-randomized-term-queries-latency



设计进阶

7.x为什么不再支持mapping type

- 错误的暗示（“索引”类似于SQL数据库中的“数据库”，而“类型”等同于“表”）
- 同一个index下，不同的mapping type下的同一个字段名不是独立的，必须有同样的字段类型
- 同一个index下，不同的mapping type，如果具有很少或没有相同字段的不同实体会导致数据稀疏并干扰Lucene有效压缩文档的能力。

7.X中如何使用mapping type

- 可以自己定义type，然后使用bool – filter，来实现类似的功能。
- 这个bool-fileter，可定义为alias
- 这个功能从设计上，可以通过gateway/proxy层做到对用户端无感

```
PUT twitter
{
  "mappings": {
    "_doc": {
      "properties": {
        "type": { "type": "keyword" }, 1
        "name": { "type": "text" },
        "user_name": { "type": "keyword" },
        "email": { "type": "keyword" },
        "content": { "type": "text" },
        "tweeted_at": { "type": "date" }
      }
    }
  }
}
```

```
GET twitter/_search
{
  "query": {
    "bool": {
      "must": {
        "match": {
          "user_name": "kimchy"
        }
      },
      "filter": {
        "match": {
          "type": "tweet" 4
        }
      }
    }
  }
}
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html

elastic

为字段选择最合适的类型

- 尽量避免动态映射
- 了解各种datatype的含义，为字段选择原则上正确的datatype
  - IP地址映射为IP类型；经纬度映射为Geo-point
  - 数据类型，应根据数值的大小选择合适的类型(byte, integer, short, long, double ...)
  - 了解text和keyword的区别，term query应该使用keyword
  - 知道何时使用array，nested，parent-child类型
- 能够根据业务的需求，自主的选择看似”不正确”，但最合适的的类型和参数
  - 鲜少被用range query的数值、日期选择使用keyword更有效 (比如轮次，顺序等数值，应该是keyword)
  - 不会被用来聚合和排序的字段，关闭doc_value
  - 像部门ID这种常被聚合的Keyword，设置eager_global_ordinals

合理使用Multi-fields

It is often useful to index the same field in different ways for different purposes.

- 比如，一个字符串字段可以被映射为 text和keywrod，前者用于全文检索，后者用于排序、聚合

```
PUT my_index
{
  "mappings": {
    "properties": {
      "city": {
        "type": "text",
        "fields": {
          "raw": { 1
            "type":  "keyword"
          }
        }
      }
    }
  }
}
```

```
GET my_index/_search
{
  "query": {
    "match": {
      "city": "york" 2
    }
  },
  "sort": {
    "city.raw": "asc" 3
  },
  "aggs": {
    "Cities": {
      "terms": {
        "field": "city.raw" 4
      }
    }
  }
}
```


elastic

合理使用Multi-fields

analyze the same field in different ways for better relevance.

- 我们可以将一个字段同时映射多个分析器，尝试命中更多的内容

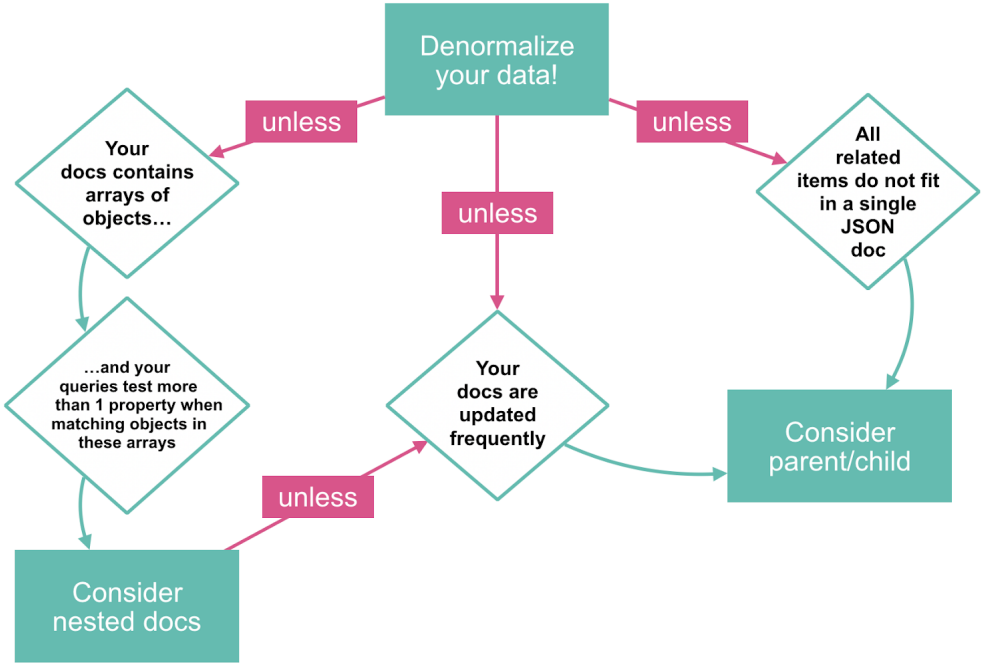
```
PUT my_index
{
  "mappings": {
    "properties": {
      "text": { 1
        "type": "text",
        "fields": {
          "english": { 2
            "type":     "text",
            "analyzer": "english"
          }
        }
      }
    }
  }
}
```

```
GET my_index/_search
{
  "query": {
    "multi_match": {
      "query": "quick brown foxes",
      "fields": [ 5
        "text",
        "text.english"
      ],
      "type": "most_fields" 6
    }
  }
}
```


elastic

选择合适的一对多的关系

- 嵌套类型允许对象数组彼此独立地建立索引和查询
- 更新嵌套对象需要对根对象及其所有其他嵌套对象进行完全的重新索引
- join数据类型的主要优点是能够独立于父对象修改子对象
- 避免join数据类型



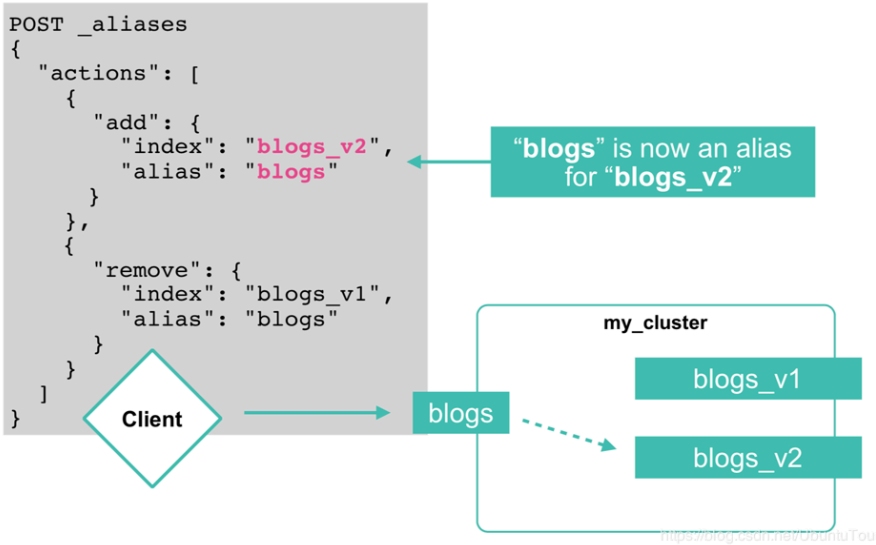
elastic

使用auto generated ID

在索引文档的时候，尽量避免显式地指定的ID (比如在数据同步场景，使用其他数据库的主键作为id)，因为Elasticsearch需要检查具有相同ID的文档是否已存在于同一分片中，这是一项昂贵的操作，并且随着索引的增长而变得更加昂贵。通过使用自动生成的ID，Elasticsearch可以跳过此检查，这使索引速度更快

掌握alias的技巧

- 尽量使用alias访问索引
- 使用alias来保存过滤条件，达到和type类似的效果
- 可以使用alias来交换索引，达到索引重命名的效果
- 使用字段的alias，可以帮助我们重新命名一个字段，并让这个字段的名称符合我们的命名规则，比如，对齐ECS。



高品质的Elasticsearch集群需要专业人士

ES作为一个大数据平台，需要一个类似DBA的角色

- 业务层只专注业务逻辑
- 传统运维只关心集群机器稳定，可用
- 谁负责：
  - 索引mapping设计
  - 索引生命周期管理、镜像备份策略
  - 开发中间层，屏蔽ES管理细节，解耦
  - 容量规划
  - 性能调优

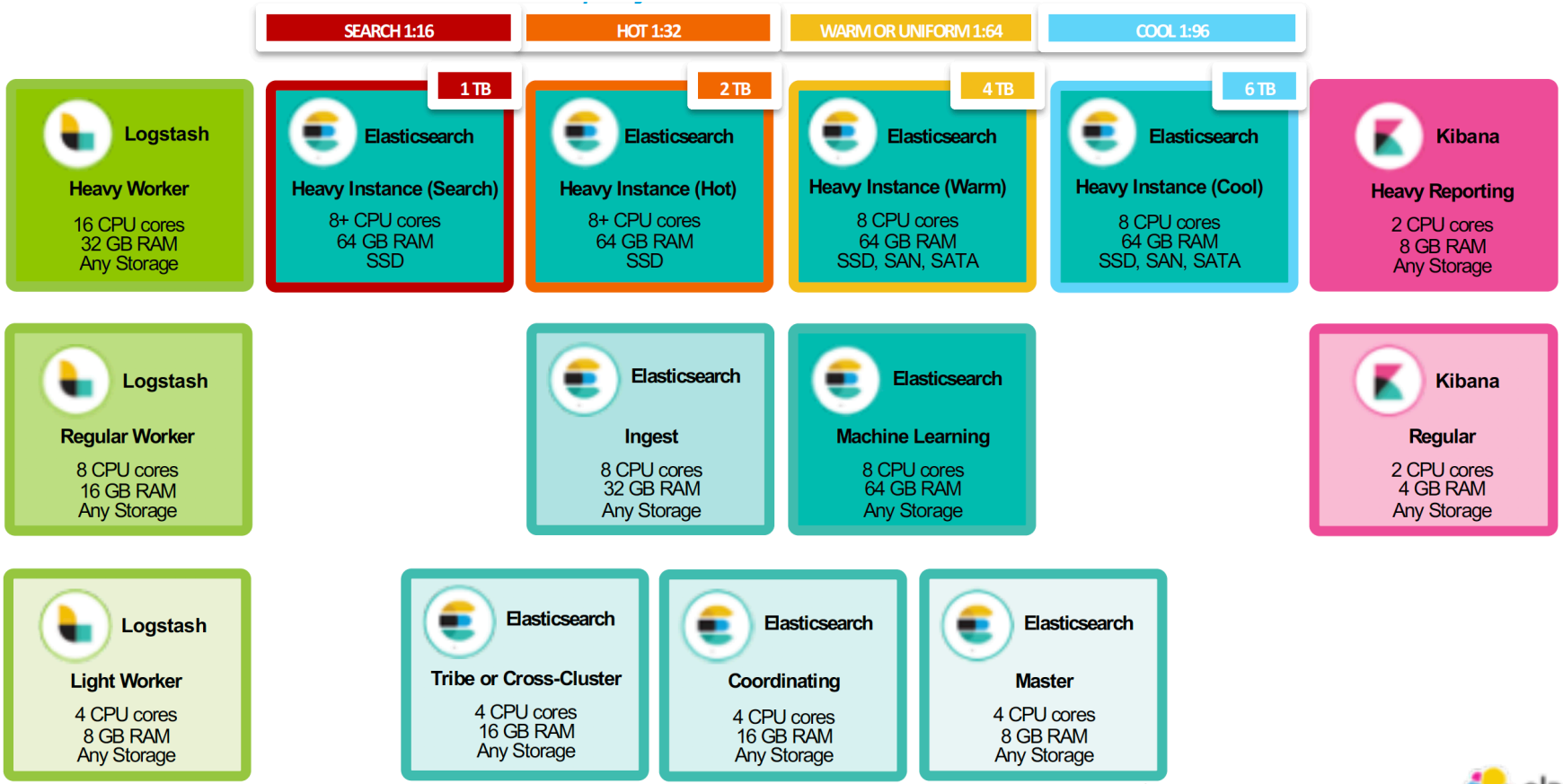
elastic

性能优化进阶

内存>ssd>hdd
算法择优
数据预处理
空间换时间

elastic

硬件建议配置



** Data volumes are guidelines*

elastic

内存>ssd>hdd

- Hot-Warm架构
- Global ordinals
- Warm up filesystem cache

```
Ordinal | Term
--------------------
0       | status_deleted
1       | status_pending
2       | status_published

Doc     | Ordinal
------------------------
0       | 1  # pending
1       | 1  # pending
2       | 2  # published
3       | 0  # deleted
```

Fielddata的使用

- Fielddata是将该字段的倒排索引进行翻转，生成一个词（term，而非字段的value）↔文档的正排索引。然后保存在JVM的堆内存当中。
- 对text字段进行aggs, sort, script的时候，是对Fielddata进行对应的aggs, sort, script。其效果和keyword完全不一样

```
POST _bulk
{ "index" : { "_index" : "test", "_id" : "1" } }
{ "fav" : "Messi,cr7,james,kobe" }
{ "index" : { "_index" : "test", "_id" : "2" } }
{ "fav" : "kobe,harden,Messi" }
{ "index" : { "_index" : "test", "_id" : "3" } }
{ "fav" : "cr7,james" }
{ "index" : { "_index" : "test", "_id" : "4" } }
{ "fav" : "cr7,harden" }
{ "index" : { "_index" : "test", "_id" : "5" } }
{ "fav" : "Messi,james" }
```

```
GET test/_search
{
  "size": 0,
  "aggs": {
    "fav_term": {
      "terms": {
        "field": "fav",
        "size": 10
      }
    }
  }
}
```

elastic

```
"aggregations" : {
  "fav_term" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : "cr7",
        "doc_count" : 3
      },
      {
        "key" : "james",
        "doc_count" : 3
      },
      {
        "key" : "messi",
        "doc_count" : 3
      },
      {
        "key" : "harden",
        "doc_count" : 2
      },
      {
        "key" : "kobe",
        "doc_count" : 2
      }
    ]
  }
```

Fielddata

```
"aggregations" : {
  "fav_term" : {
    "doc_count_error_upper_bound" : 0,
    "sum_other_doc_count" : 0,
    "buckets" : [
      {
        "key" : "Messi,cr7,james,kobe",
        "doc_count" : 1
      },
      {
        "key" : "Messi,james",
        "doc_count" : 1
      },
      {
        "key" : "cr7,harden",
        "doc_count" : 1
      },
      {
        "key" : "cr7,james",
        "doc_count" : 1
      },
      {
        "key" : "kobe,harden,Messi",
        "doc_count" : 1
      }
    ]
  }
}
```

keyword

elastic

数据预处理 - Script

尽量避免使用script对大量的查询结果直接进行检索

- Script query 通常是非常消耗内存的。 通常用于探索和分析数据，不建议用于生产
- 使用预计算代替script query，以空间换时间
- 如果必须要用script，则应首选painless和expressions引擎。

```
{
  "query": {
    "script": {
      "script": "doc['job_name.keyword'].value.startWith('Elastic')"
    }
  }
}
```

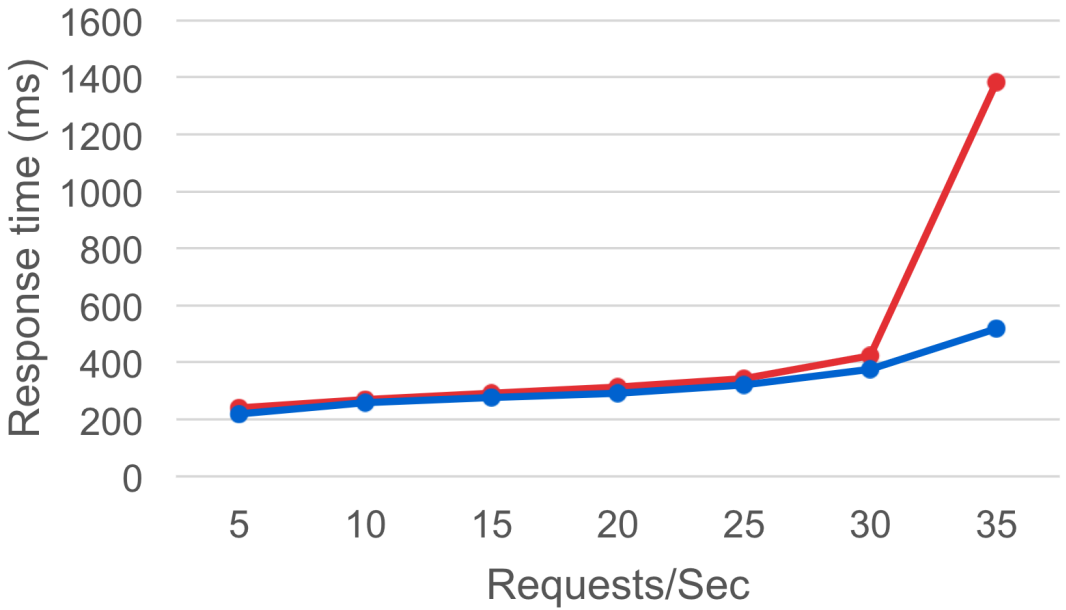
```
{
  "query": {
    "match": {
      "job_name_prefix.keyword": "Elastic"
    }
  }
}
```

elastic

算法 - Query vs Filter

如果可以，尽量使用Filter来替代query

- 在使用filter context时，ElasticSearch 不需要做相关性的计算（打分）
- Filter的搜索结果可以被缓存



elastic

算法 – prefix、RegExp、wildcard

- 如果满足你的需求，前缀匹配是优于wildcard和regexp
- 根据提供的正则表达式，regexp查询的性能可能有所不同。要提高性能，请避免使用一个以通配符开头的模式(比如，*foo或者正则表达式: .*foo)，wildcard同样需要避免
- 尽量避免让用户直接使用正则表达式
- 特定工具去检查正则表达式的步长：https://regex101.com/

搜索尽可能少的字段

尝试使用Copy-to

- query_string或 multi_match查询目标的字段越多，它的速度就越慢。提高多个字段搜索速度的常用技术是在索引阶段将其值复制到单个字段中，然后在搜索时使用此字段。这可以通过copy-to映射指令自动执行，而不必更改文档的来源。

```
PUT movies
{
  "mappings": {
    "properties": {
      "name_and_plot": {
        "type": "text"
      },
      "name": {
        "type": "text",
        "copy_to": "name_and_plot"
      },
      "plot": {
        "type": "text",
        "copy_to": "name_and_plot"
      }
    }
  }
}
```

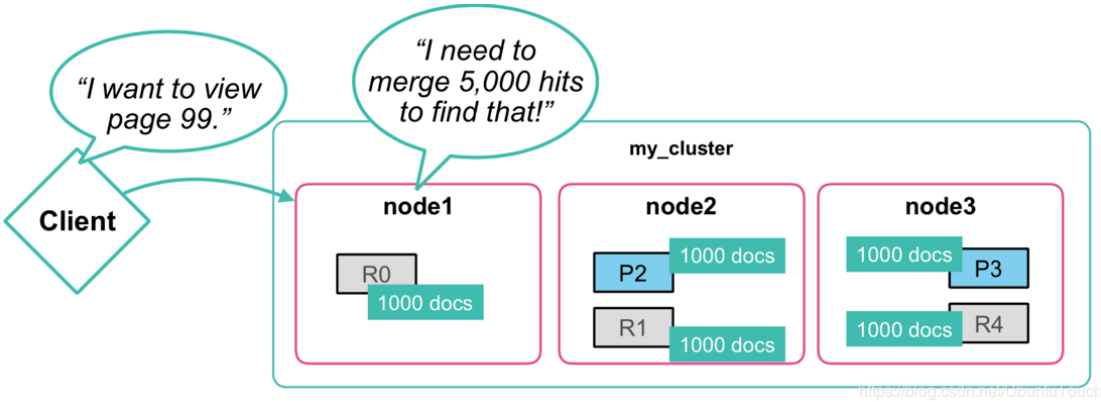
避免返回大的数据集

- Elasticsearch被设计为搜索引擎，使其非常适合检索与查询匹配的TOP N文档。但是，对于属于数据库领域的工作负载（如检索与特定查询匹配的所有文档），效果不佳。如果需要执行此操作，请确保使用Scroll API
  - 通过size 控制要返回的文档数（聚合的时候，size设为0）
  - 在使用scroll API的时候可以考虑用_doc进行排序
  - 在计数时，使用_count接口
  - 使用"_source"或"stored_field"，让ES只返回必须的字段
- 同时也要尽量避免索引并存储大文件。大文件会有很多的问题：
  - 索引时更多的内存开销
  - 搜索时更多IO开销
  - 更多的缓存开销
  - Highlight...
- 建议是ES索引大文件的关键信息，不存储，和hbase等数据库联合使用

深度翻页问题

了解from-size, scroll, search_after的区别和应用场景

- 单次请求的上限是10K个文档
- 数据量很大的时候，from-size会有严重的效率问题
- 深度翻页推荐使用scroll接口
- 对于实时翻页，推荐使用search_after



elastic

谢谢！

elastic