## ElasticSearch调研总结

**Elasticsearch简介**

Elasticsearch 是分布式，REST风格，搜索和分析系统。具有实时数据，实时分析，分布式，高可用性，多租户，全文搜索，面向文档，冲突管理，自由模式，rest风格API，每个操作的持久性，，Apache 2的开源许可证，基于Apache Lucene之上的特点。

Es 官网地址： http://www.elasticsearch.org/

### Elasticsearch 分布式环境搭建

从<http://www.elasticsearch.org/download/> 下载解压到某个目录。

**1.单机环境：**

单机版的elasticsearch运行很简单，linux下直接 bin/elasticsearch就运行了，windows运行bin/elasticsearch.bat。如果是在局域网中运行elasticsearch集群也是很简单的，只要cluster.name设置一致，并且机器在同一网段下，启动的es会自动发现对方，组成集群。

**2.服务器环境：**

如果是在服务器上就可以使用elasticsearch-servicewrapper这个es插件，它支持通过参数，指定是在后台或前台运行es，并且支持启动，停止，重启es服务（默认es脚本只能通过ctrl+c关闭es）。使用方法是到<https://github.com/elasticsearch/elasticsearch-servicewrapper> 下载service文件夹，放到es的bin目录下。以下是elasticsearch-servicewrapper 操作命令

|  |
| --- |
| bin/service/elasticsearch +  console 在前台运行es  start 在后台运行es  stop 停止es  install 使es作为服务在服务器启动时自动启动  remove 取消启动时自动启动 |

在service目录下有个elasticsearch.conf配置文件，主要是设置一些java运行环境参数，其中列出比较重要参数

|  |
| --- |
| #es的home路径，不用用默认值就可以  set.**default**.ES\_HOME=<Path to ElasticSearch Home>  #分配给es的最小内存  set.**default**.ES\_MIN\_MEM=    #分配给es的最大内存  set.**default**.ES\_MAX\_MEM=    # 启动等待超时时间（以秒为单位）  wrapper.startup.timeout=    # 关闭等待超时时间（以秒为单位）  wrapper.shutdown.timeout=    # ping超时时间(以秒为单位)  wrapper.ping.timeout= |

**Elasticsearch 目录结构：**

bin 运行ElasticSearch实例和管理插件的一些脚本

config 放置两个配置文件：elasticsearch.yml和logging.yml

lib ElasticSearch使用的依赖包

当启动elasticsearch，elasticsearch还会创建以下文件夹：

data ElasticSearch存放数据的地方

logs 日志的文件夹

plugins 已经安装的插件的存放位置

work 一些临时文件

**Elasticsearch 插件安装**

**elasticsearch-head安装**

作用： 查看索引情况，搜索索引，查看集群状态和分片分布等。

命令行安装：

|  |
| --- |
| D:\elasticsearch\elasticsearch-1.0.1\bin>plugin –mobz/elasticsearch-head |

下载安装： <https://github.com/mobz/elasticsearch-head>

安装完毕后：重新启动elasticsearch。

在浏览器中输入: <http://localhost:9200/_plugin/head/> 查看结果

**bigdesk****安装**

作用： 查看集群的JVM信息，磁盘IO，索引创建删除信息等，适合查找系统瓶颈，监控集群状态等。

命令行安装：

|  |
| --- |
| D:\elasticsearch\elasticsearch-1.0.1\bin> plugin -install lukas-vlcek/bigdesk |

下载安装包：<https://github.com/lukas-vlcek/bigdesk>

安装完毕后：重新启动elasticsearch。在浏览器中输入:

<http://localhost:9200/_plugin/bigdesk/>

访问其它插件列表：

<http://www.elasticsearch.org/guide/reference/modules/plugins/>

思考：ES 不同网段分布式配置？

|  |
| --- |
| # discovery.zen.ping.unicast.hosts: ["host1", "host2:port"] |

预知更多详情：参见4.elasticsearch配置文件

### Elasticsearch是如何实现分布式实时搜索的？

思考：Es是基于lucene的，那么怎样在lucene之上构建一个分布式，高伸缩，接近实时的索引呢？

以下为了更好的理解，贴上一些lucene的关键概念。以便得知要构建一个分布式接近实时的搜索引擎，有哪些局限性。

**Directory**

Lucene Directory 是一个抽象的文件系统的接口，用来允许你读写文件，不管lucene的索引是存放在内存中还是在物理磁盘上，它都是通过lucene的Directory抽象层来访问和维护的。

**IndexWriter**

IndexWriter 用来添加、删除和更新lucene里面的索引文档。这些操作是在内存中完成以保证更好的性能，但是如果要保证这些操作的持久化，这些操作是需要flush到磁盘的。并且，flush操作或者是显式的commit提交开销都是比较大的，因为这些操作通常需要处理很多的文件，而处理这些文件又涉及到大量的磁盘io。

此外, 每次只能有一个IndexWriter对象来对一个索引目录进行索引操作，并且创建这个对象的开销很大，所以必须尽可能去重用这个对象.

**Index Segments**

Lucene 索引被分解为很多段（segments）。每个索引段实际上是一个功能完整的lucene索引，一旦一个索引段创建完成，它将是不可变的，并且不能删除段里面的索引文档。commit提交操作用来往索引里面添加一个新段。lucene内部会来对这些段进行合并，所以我们必须要有策略来控制这些合并(MergePolisy, MergeScheuler, … etc)。Because segments need to be kept at bay they are being merged continuously by internal Lucene processes (MergePolisy, MergeScheuler, … etc).

因为段是不可变的，所以用来做缓存（caching）是一个很好的选择，你可以加载所有的term词条并且创建一个跳跃列表（ [skip lists](http://en.wikipedia.org/wiki/Skip_list) ) ，或者用来构造FieldCache，如果段没有变化，你就不需要重新加载。

**IndexReader**

IndexReader 用来执行搜索索引。这个对象通过IndexWriter来提供，并且创建代价也是比较高。一旦IndexReader打开之后，它就不能够发现打开之后的索引变化，如果要知道这些由IndexWriter产生的索引变化，除非刷新IndexReader对象（当然前提需要flush操作）。搜索操作在内部其实是按段来进行的（每次一个段）.

**Near Real-Time Search**

获取一个新的IndexReader开销很大，所以也是我们不能每次一有索引操作就真的去获取一个新的IndexReader，你可以隔一段时间去刷新一下，比如每隔一秒钟等等，这也是我们在这里称之为接近实时( **near** real-time )的原因.

#### 实时数据：

……

#### 分布式：

在分布式系统中有一个很重要的方面就是——复制（replication）。

**复制的好处：**

**High Availability** (HA高可用性)： 一个节点挂了，另一个节点能够顶上，如果一个节点上有索引分片，而另一个节点持有该分片的副本，那么表示数据有了一个备份。

（思考：ES 如何实现备份？）

**scalability** (可伸缩性)： 通过增加副本或从节点（slave nodes），可以实现提升搜索吞吐量，提高搜索能力。

两种实现复制方式： **Push Replication（推模式）** 和 **Pull Replication（拉模式）**

**Push Replication （ElasticSearch使用）**：

当你往[master]主分片上索引一个文档，该分片会复制该文档（document）到剩下的所有[replica]副本中。从而这些分片也会索引这个文档。

优缺点:

一旦文档索引完毕，那么文档所在所有分片及副本都是立即可用的，索引操作会等待直到确认所有的副本也执行了同样的索引操作(注意: 如果需要，你也可以使用异步复制)，这意味着索引的实时性. 然后你只需要 refresh 一下 IndexReader 就能搜索到新的数据了。这种架构能够非常方便的在节点之间进行切换，如果包含主分片（primary shard）的节点挂了，能够很快的进行切换。

缺点：

同一个文档重复索引多次。对于同一个文档的频繁索引，在主分片和副本分片执行的时候，必须保证每次都是按照正确的顺序或通过版本来拒绝旧版本操作。

**Pull Replication（Solr使用）：**

拉模式是一种主从方式，当一个文档在master上面进行索引时，并且数据通过commit操作产生了新的段文件（segment），这时从节点（slave）会把这些segment拉到自己的机器然后执行相应的刷新操作，并保证lucene能够使用这些新数据。

缺点：

需要在master上面执行commit操作来产生这些段文件（segment），这样slave才能够执行pull操作。注意lucene的commit开销是非常大的..

数据的传输会有不必要的冗余. 在分布式系统里面，网络通常来说是非常宝贵的资源(如果你跑在EC2上面，那将更加宝贵,$$$) 并且最终要移动的数据会越来越多,举例来说，如果你有2个段文件，里面包含了文档，文档里面的字段都是存储的（stored fields）,并且Lucene决定要合并这2个段文件，那么你也必须要传输这部分数据（合并之后的段文件），因为这是一个新的段文件，但是实际上你传输的是一份相同的数据.

这将造成一个这样的局面，所有的slaves确实是在master后面. 也可能是确实没有理由每次都进行commit或者花大量时间来传输一个大的段文件。但是至少意味着你的slave会丢失 high availability，并且不可能当成是一个实时的slave（a real time high available slave）. 实时搜索不可能存在，并且（使用拉模式）也不可能有这种1秒的刷新率，然后lucene就能实时搜索.

**Elasticsearch ：Translog**

解决问题： 保证elasticsearch中数据一致性。即**实现数据可靠性（Data Persistency）**

ElasticSearch通过使用 **transaction log**记录发生在索引上的各种操作，使算没有调用commit操作也能保证数据的持久化。并且能够很自然的支持推送复制（push replication），因为我们能够让每个不同的shard都拥有 **transaction log** ，就算某些节点崩溃了，如果有必要，可以很轻松对日志操作进行重放（replay）。

Transaction log 周期性的将数据刷新(flushed)到磁盘，通过参数 来进行控制. 简单来说就是保存两次提交之间的连续数据操作的记录。

注： es 的Transaction log即使被强制结束进程（”kill -9”）也不会丢失任何数据。

每个碎片都有一个事务日志文件，事务日志文件主要是为了保证索引写和删除过程的可靠性，你不需要显式的去提交每个请求（对应于lucene的commit），这一切都是自动的，你也可以显式的执行flush操作来进行请求的提交，还可以使用下面这些参数来进行控制:

|  |  |
| --- | --- |
| 配置 | 说明 |
| index.translog.flush\_threshold\_ops | 设置当累计操作达到多少时就执行flush操作，默认值 5000. |
| index.translog.flush\_threshold\_size | 一旦你的事务日志文件的大小（translog）达到设置的这个值，则开始执行flush操作，默认值500mb. |
| index.translog.flush\_threshold\_period | 每隔多长时间执行一次flush，默认 60m. |

### Elasticsearch Java客户端编程实践

1. **获取java client**

常用的几种方式：

1. 使用[jest](https://github.com/searchbox-io/Jest) JAVA API ：

可以很好处理java对象序列，无须处理JSON基础文档结构，即可方便的进行POJO序列化。

1. 使用反射方式

网上反映效率明显高于new客户端，并可避免线上环境内存溢出和超时等问题。

Eg:

|  |
| --- |
| **static** Map<String, String> m = **new** HashMap<String, String>();  *// 设置client.transport.sniff为true来使客户端去嗅探整个集群的状态，把集群中其它机器的ip地址加到客户端中，*  **static** Settings settings = ImmutableSettings.settingsBuilder().put(m).put("cluster.name",clusterName).put("client.transport.sniff", **true**).build();  *// 创建私有对象*  **private** **static** TransportClient client;    **static** {  **try** {  Class<?> clazz = **Class**.forName(TransportClient.**class**.getName());  Constructor<?> constructor = clazz.getDeclaredConstructor(Settings.**class**);  constructor.setAccessible(**true**);  client = (TransportClient) constructor.newInstance(settings);  client.addTransportAddress(**new** InetSocketTransportAddress(host, port));  } **catch** (Exception e) {  e.printStackTrace();  }  }    *// 取得实例*  **public** **static** **synchronized** TransportClient getTransportClient() {  **return** client;  } |

1. **建立索引查询实例**

创建Mapping，

创建Index

Search Index

……

### Elasticsearch 配置文件

Elastisearch.yml 和logging.yml 第一个是ES的基本配置文件，第二个是日志配置文件。

**Cluster**

cluster.name :elasticsearch ： 配置集群名称，如果你在同一个网段下运行多个集群，可以用这个配置区分不同集群。

**Node**

node.name: "Franz Kafka" ：节点名称，节点名称是自动生成的（名称列表在es的jar包中config文件夹里name.txt文件中）， 可以通过配置给节点取一个特殊的名称。

node.master: true：指定该节点是否有资格被选举成为master node，默认true，ES默认第一台机器为master，如果挂了，则重新选举。

node.data: true ： 指定这个节点存储索引数据，默认true

|  |
| --- |
| 通过配置完成先进的拓扑集群：  1. 将节点设置为主节点 并保存索引数据：  node.master: false  node.data: true  2. 将节点配置为主节点，不保存任何数据。（用作集群的“协调者”）。  node.master: true  node.data: false  3. 将节点永不配置成主节点，并不保存索引数据。（用做“搜索平衡器”）  node.master: false  node.data: false |

**Index**

index.number\_of\_shards: 5 ：设置索引的分片个数，默认设置5个

index.number\_of\_replicas: 1 ：设置索引副本数量，默认设置1个

**Paths**

path.conf: /path/to/conf ： 设置配置文件的存储路径，默认是ES根目录下的config文件夹。

path.data: /path/to/data ： 设置索引数据存储路径，默认是ES根目录下的data文件夹，设置多个用,分隔。 例：path.data: /path/to/data1,/path/to/data2

path.logs: /path/to/logs ： 设置日志文件存储路径，默认ES根目录下logs文件夹

path.plugins: /path/to/plugins ：设置插件存放路径，默认ES根目录下plugins文件夹

**Plugin**

plugin.mandatory: mapper-attachments,lang-groovy ： 设置当前节点强制依赖的插件，没有则不启动。

**Memory：**

# bootstrap.mlockall: true ： 设置true表示锁定内存，因为jvm开始swapping时ES的效率会降低，所以要保证不swap，可以把ES\_MIN\_MEM和ES\_MAX\_MEM两个环境变量设置成相同值，并且保证机器有足够内存分配给ES，和保持足够的内存给操作系统。

同时也要允许elasticsearch的进程可以锁住内存，linux下可以通过`ulimit -l unlimited`命令。

**Network And HTTP**

network.bind\_host: 192.168.0.1 ： 设置绑定ip地址

network.publish\_host: 192.168.0.1 ：设置其他节点和该节点交互的ip地址，如果不设置它会自动判断，值必须是真实IP地址

network.host: 192.168.0.1 ： 这个参数是用来同时设置bind\_host和publish\_host上面两个参数。

transport.tcp.port: 9300 ：设置节点间交互的tcp端口，默认是9300。

transport.tcp.compress: true ：设置是否压缩tcp传输时的数据，默认为false，不压缩。

http.port: 9200 ： 设置对外服务的http端口，默认为9200。

http.max\_content\_length: 100mb ： 设置内容的最大容量，默认100mb

http.enabled: false ： 是否使用http协议对外提供服务，默认为true，开启。

**Gateway**

gateway.type: local ： gateway的类型，默认为local即为本地文件系统（推荐）；

gateway.recover\_after\_nodes: 1 ：设置集群中N个节点启动时进行数据恢复，默认为1。

gateway.recover\_after\_time: 5m ： 设置初始化数据恢复进程的超时时间，默认是5分钟。

gateway.expected\_nodes: 2 ： 设置这个集群中节点的数量，默认为2，一旦这N个节点启动，就会立即进行数据恢复。

**Recovery Throttling**

cluster.routing.allocation.node\_initial\_primaries\_recoveries: 4 ： 初始化数据恢复时，并发恢复线程的个数，默认为4。

cluster.routing.allocation.node\_concurrent\_recoveries: 2 ： 添加/删除节点或负载均衡时并发恢复线程的个数，默认为4。

indices.recovery.max\_bytes\_per\_sec: 20mb ： 设置数据恢复时限制的带宽，默认20mb

indices.recovery.concurrent\_streams: 5 ：设置这个参数来限制从其它分片恢复数据时最大同时打开并发流的个数，默认为5。

**Discovery**

discovery.zen.minimum\_master\_nodes: 1 ： 设置这个参数来保证集群中的节点可以知道其它N个master资格的节点，默认为1，对于大的集群来说，可以设置大一点的值（2-4）

discovery.zen.ping.timeout: 3s ：设置集群中自动发现其它节点时ping连接超时时间，默认为3秒，对于比较差的网络环境可以高点的值来防止自动发现时出错。

discovery.zen.ping.multicast.enabled: false ：设置是否打开多播发现节点，默认是true。

discovery.zen.ping.unicast.hosts: ["host1", "host2:port"] ：设置集群中master节点的初始列表，可以通过这些节点来自动发现新加入集群的节点。

**Slow Log**

….

\* 补充索引设置

|  |  |
| --- | --- |
| **Setting** | **Description** |
| index.compound\_format | 索引的文件格式是否使用混合模式，如果没有设置，由实际使用的存储来控制，因为当使用文件系统来存储的时候,使用混合模式的索引格式，可以用来减少打开文件的句柄数，默认设置为@false@ 保证更好的性能（当使用文件系统的索引存储确实非常合适，不过需要调整最大打开文件数。 |
| index.term\_index\_interval | 用更多内存，访问词的速度就会提高，默认值为 128 |
| index.term\_index\_divisor | 那些词会被二次抽样加载到内存，使用效果和 index.term\_index\_interval 一样，只不过后者的设置必须在索引之前进行设置，而这个可以是按读或搜索来分别进行设置，当设置为N，那么索引里面的 N\*termIndexInterval个词则会加载到内存，当设置的值>1,可以减少内存的使用,默认是1，设置为-1将直接跳过索引中词的加载。 |
| index.refresh\_interval | 用来控制多长时间进行一次刷新操作，默认为1s，可以设置为 -1 来禁用刷新 |