集群管理

1. 集群配置：
   1. 节点：
      1. 相同角色的节点，避免使用差异较大的服务器配置，
      2. 避免使用“超大杯”服务器（SS：Super Server），比如128核CPU，1 T的内存，2T的固态硬盘。这样可能会产生较大的资源浪费。
      3. 等量的配置，使用较少的物理机好于使用较多的虚拟机。比如一个一个五台4核16G的物理机，好于10甚至11台2核8G的虚拟机，这里不仅仅是虚拟机本身可能也会消耗一部分性能的问题，也涉及数据安全的问题。
      4. 避免在同一台服务器上部署多个节点，会增加集群管理的难度。
   2. 内存：根据业务量不同，内存的需求也不同，一般生产建议不要少于16G。ES是比较依赖内存的，并且对内存的消耗也很大，内存对ES的重要性甚至是高于CPU的，所以即使是数据量不大的业务，为了保证服务的稳定性，在满足业务需求的前提下，我们仍需考虑留有不少于20%的冗余性能。一般来说，按照百万级、千万级、亿级数据的索引，我们为每个节点分配的内存为16G/32G/64G就足够了，太大的内存，性价比就不是那么高了。
   3. 磁盘：对于ES来说，磁盘可能是最重要的了，因为数据都是存储在磁盘上的，当然这里说的磁盘指的是磁盘的性能。磁盘性能往往是硬件性能的瓶颈，木桶效应中的最短板。ES应用可能要面临不间断的大量的数据读取和写入。生产环境可以考虑把节点冷热分离，“热节点”使用SSD做存储，可以大幅提高系统性能；冷数据存储在机械硬盘中，降低成本。另外，关于磁盘阵列，可以使用raid 0。
   4. CPU：CPU对计算机而言可谓是最重要的硬件，但对于ES来说，可能不是他最依赖的配置，因为提升CPU配置可能不会像提升磁盘或者内存配置带来的性能收益更直接、显著。当然也不是说CPU的性能就不重要，只不过是说，在硬件成本预算一定的前提下，应该把更多的预算花在磁盘以及内存上面。通常来说单节点cpu 4核起步，不同角色的节点对CPU的要求也不同。服务器的CPU不需要太高的单核性能，更多的核心数和线程数意味着更高的并发处理能力。现在PC的配置8核都已经普及了，更不用说服务器了。
   5. 网络： ES是天生自带分布式属性的，并且ES的分布式系统是基于对等网络的，节点与节点之间的通信十分的频繁，延迟对于ES的用户体验是致命的，所以对于ES来说，低延迟的网络是非常有必要的。因此，使用扩地域的多个数据中心的方案是非常不可取的，ES可以容忍集群夸多个机房，可以有多个内网环境，支持跨AZ部署，但是不能接受多个机房跨地域构建集群，一旦发生了网络故障，集群可能直接GG，即使能够保证服务正常运行，维护这样（跨地域单个集群）的集群带来的额外成本可能远小于它带来的额外收益。
   6. 集群规划：没有最好的配置，只有最合适的配置。
      1. 在集群搭建之前，首先你要搞清楚，你ES cluster的使用目的是什么？主要应用于哪些场景，比如是用来存储事务日志，或者是站内搜索，或者是用于数据的聚合分析。针对不同的应用场景，应该指定不同的优化方案。
      2. 集群需要多少种配置（内存型/IO型/运算型），每种配置需要多少数量，通常需要和产品运营和运维测试商定，是业务量和服务器的承载能力而定，并留有一定的余量。
      3. 一个合理的ES集群配置应不少于5台服务器，避免脑裂时无法选举出新的Master节点的情况，另外可能还需要一些其他的单独的节点，比如ELK系统中的Kibana、Logstash等。
2. 集群搭建：
   1. 过程参见0701：ES集群

curl -H "Content-Type: application/json" -XPOST http://172.16.10.184:9200/sys-tcp-2020.07.18/\_search?pretty -d ' {"query": {"match\_all": {}}}'

* 1. 重要的节点配置（一定要修改的配置）：
     1. cluster.name: 配置集群的名称，集群发现是通过集群名称来识别不同集群的。很重要。
     2. node.name: 节点名称
     3. path.data：数据的存储路径，必须修改为和es安装目录不同的位置，如果同一台服务器部署了多个节点，那么多个节点的路径必须不同。
     4. path.logs：日志的存储路径，规则同上。
     5. bootstrap.memory\_lock：禁用swap，只要记住，想要es快，这个一定要打开，想弄明白看0701，或者查阅文档。
     6. network.host：给当前节点绑定的ip地址
     7. http.port：服务端口号，默认9200，取值范围[9200-9300)
     8. transport.port：通信端口号，节点之间联络传输数据的端口，取值范围[9300,9400)。
     9. discovery.seed\_hosts：master候选节点的列表，注意不是及群众所有的节点，通常为数组，比如（IPV4）：["172.16.10.184:9300", "172.16.10.185:9300"]，数组中可以使用节点名称代替IP地址，不过建议使用IP地址。
     10. cluster.initial\_master\_nodes：初始化集群的master节点，就是集群首次启动的时候指定谁为默认的master。
     11. http.cors.enabled：开启跨域支持
     12. http.cors.allow-origin：设置哪些地址可以跨域，\*代表允许任何地址跨域。
  2. 其他配置：es的默认配置是一个非常合理的默认配置，绝大多数情况下是不需要修改的，如果不理解某项配置的含义，没有经过验证就贸然修改默认配置，可能造成严重的后果。比如max\_result\_window这个设置，默认值是1W，这个设置是分页数据每页最大返回的数据量，冒然修改为较大值会导致OOM。ES没有银弹，不可能通过修改某个配置从而大幅提升ES的性能，通常出厂配置里大部分设置已经是最优配置，只有少数和具体的业务相关的设置，事先无法给出最好的默认配置，这些可能是需要我们手动去设置的。关于配置文件，如果你做不到彻底明白配置的含义，不要随意修改。
  3. JVM：
     1. jvm heap分配：7.6版本默认1GB，这个值太小，很容易导致OOM。Jvm heap大小不要超过物理内存的50%，最大也不要超过32GB（compressed oop），它可用于其内部缓存的内存就越多，但可供操作系统用于文件系统缓存的内存就越少，heap过大会导致GC时间过长。修改jvm heap有两种方式：
        1. 在ES进程启动的时候加上参数：ES\_JAVA\_OPTS="-Xms512m -Xmx512m",或在jvm.options中修改-Xms和-Xmx，注意最大值和最小值要相同，避免jvm heap在运行中resize，这是一个非常耗性能的过程。
        2. 修改环境变量：ES\_HEAP\_SIZE，环境变量的优先级高于jvm.options中的设置的数值。
     2. 通常来说JVM其他配置是不需要修改的，比如说GC。
  4. 特殊配置：
     1. 避免脑裂：discovery.zen.minimum\_master\_nodes=N/2+1
  5. Threadpool：
     1. 查看：curl -XGET 'http://172.16.10.184:9200/\_nodes/stats?pretty'
        1. GET /\_cat/thread\_pool
        2. GET /\_nodes/stats?pretty
     2. **ES中不同线程池解释：**
        1. **Generic：**用于常规操作（例如，后台节点发现）。线程池类型为scaling。
        2. **Search：**用于计数/搜索/搜索推荐（建议词）操作。线程池类型 fixed\_auto\_queue\_size的大小为：可获得的线程数(默认情况下是CPU的线程数，比如4核8线程的CPU默认就是8) x 3 / 2 + 1，初始queue\_size为 1000。当超过queue最大值，ES将拒绝请求。
        3. search\_throttled**：**用于对进行计数/搜索/建议/获取操作search\_throttled indices。线程池类型fixed\_auto\_queue\_size的大小为1，初始queue\_size为100。
        4. get：线程池类型fixed 的大小为可获得的线程数，默认情况下是cpu的线程数，queue\_size为1000。
        5. **analyze：**用于分析请求。线程池类型fixed的大小为1，队列大小为16。
        6. **write：**对于单文档索引/删除/更新和批量请求。线程池类型fixed的大小为#可用线程数，queue\_size为1W。该池的最大大小为1 + 可用线程数。
        7. **snapshot：**用于快照/还原操作。线程池类型scaling的keepalive为5m，最大值为min(5, (CPU线程数)/2)。
        8. **warmer：**用于段预热操作。线程池类型scaling的keepalive为5m，最大值为min(5, (CPU线程数)/2)。
        9. **refresh：**用于刷新操作。线程池类型scaling的keepalive为5m，最大值为min(10, (CPU线程数)/2)。
        10. **listener：**主要用于Java客户端，当侦听器线程设置为时执行动作 true。线程池类型scaling的默认最大值为 min(10, (CPU线程数)/2)。
        11. **fetch\_shard\_started：**用于列出分片状态。线程池类型为scalingkeep-alive 5m，默认最大大小为2 \* CPU线程数。
        12. **fetch\_shard\_store：**用于列出碎片存储。线程池类型为scalingkeep-alive 5m，默认最大大小为2 \* # of available。
        13. **flush：**对于[flush](https://www.elastic.co/guide/en/elasticsearch/reference/7.6/indices-flush.html)，[sync flush](https://www.elastic.co/guide/en/elasticsearch/reference/7.6/indices-synced-flush-api.html)和[translog](https://www.elastic.co/guide/en/elasticsearch/reference/7.6/index-modules-translog.html) fsync操作。线程池类型scaling的keepalive为5m，默认最大大小为min(5, (CPU线程数)/2)。
        14. force\_merge：用于[强制合并](https://www.elastic.co/guide/en/elasticsearch/reference/7.6/indices-forcemerge.html)操作。线程池类型fixed的大小为1，队列大小为无限制。
        15. management：用于集群管理。默认最大大小为5。
     3. **线程池类型：**在Java标准的线程池里面,主要有3个重要的参数会影响线程数的变化：corePoolSize、maximumPoolSize和keepAliveTime。
        1. fixed：**线程数固定，任务队列大小可调（海底捞的水杯）**
           1. **解释：**开始提交任务时，会不断创建新的线程，直至线程数达到配置的线程数，在这之后，再创建任务，如果任务队列没满，任务会进入任务队列；如果队列满了，再提交任务会被阻塞。线程数一旦达到阈值就不会减少，即使任务都执行完毕，线程是空闲的，也不会销毁
           2. **配置：**thread\_pool:

write:

size: 30

queue\_size: 1000

* + - 1. fixed\_auto\_queue\_size：线程数固定，但是**不允许用无界任务队列**
         1. **解释：**
         2. **配置：**thread\_pool:

search:

size: 30

queue\_size: 500

min\_queue\_size: 10

max\_queue\_size: 1000

scaling：（发动机机油）

* + - * 1. 当线程池个数低于corePoolSize时，每次提交新任务都会触发新的线程创建。当线程数量高于corePoolSize时，ES会首先检查创建的线程数是否达到了maximumPoolSize，如果没有，继续创建，否则才会进入任务队列。如果线程空闲时间超过keepAliveTime且线程数大于corePoolSize，线程会被销毁。
        2. 设置：thread\_pool:

warmer:

core: 1

max: 8

keep\_alive: 2m

* + 1. 线程数设置：
       1. 一般来说，ES默认会自动检测处理器的核心线程数量，并且基于处理器的线程数自动设置ES的线程池，但是在以下几种情况需要我们手动设置ES的线程池：
          1. 如果需要在同一主机上运行多个ES实例，则应修改为node.processors做对应的修改，比如，一台 8核16线程的服务器应该把ES的两个实例分别设置 node.processors= 8。不过，这是骨灰级玩家操作，因为你修改了线程数，影响范围不仅仅是线程池这么简单，还涉及很多其他因素，比如：Jvm processors。所以之前也曾经提到，尽量不要在同一台服务器上部署多个节点，避免给自己挖坑。
          2. 不排除ES会获取到错误的处理器的核心线程数量，所以我们可以手动的设置node.processors来避免这些问题。
       2. 设置方法：node.processors: 8
  1. Bootstrap Checks（引导检查）：
     1. 开发模式和生产模式：如果Elasticsearch节点无法通过非环回地址与另一台机器形成集群，则认为该节点处于开发模式；如果它可以通过非环回地址加入集群，则该节点处于生产模式。
     2. 单节点发现：设置discovery.type = single-node，当前节点会选举自己为master节点并且不参与任何集群，用于开发模式，此时ES启动会绕过引导检查。
     3. 引导检查：在ES启动时，自身有非常严格的检查机制，起检查的项包含非常多的内容，其目的是为了确保ES所有设置的正确性和合理性，从而保证ES服务运行时的稳定和最佳性能。如果我们做出了不合理的ES配置，ES宁愿拒绝服务，也不会为我们启动一个有问题的服务节点，这也是为了我们好，避免了服务启动后给我们带来不可预料的重大问题，从而导致严重后果。在开发模式下，一些不重要的问题可能只是以警告信息输出，但是在生产模式下，引导检查时会强制把这些警告升级为错误，除非我们解决了这些错误，不然ES无法启动。
     4. 引导包含的项：
        1. 堆大小检查：最大值最小值要保证相等，并且最好是在4G-32G之间。
        2. 文件描述符检查：
        3. **内存锁检查**：bootstrap.memory\_lock: true
        4. **最大线程数检查**：Elasticsearch需要具有创建大量线程的能力。检查线程的最大数量可确保Elasticsearch进程有权在正常使用下创建足够的线程。仅在Linux上强制执行此检查。如果您使用的是Linux，要通过最大线程数检查，必须将系统配置为允许Elasticsearch进程创建至少4096个线程。这可以通过/etc/security/limits.conf 使用nproc设置来完成（请注意，您可能还必须增加root用户的限制）。
        5. **最大文件大小检查**：作为单个分片的组成部分的段文件以及作为跨记录的组成部分的跨记录世代​​可能会变得很大（超过数GB）。在Elasticsearch流程可以创建的文件的最大大小受到限制的系统上，这可能导致写入失败。因此，这里最安全的选择是最大文件大小不受限制，这就是最大文件大小引导检查所强制执行的内容。要通过最大文件检查，您必须配置系统以使Elasticsearch进程能够写入无限大小的文件。这可以通过 /etc/security/limits.conf使用fsize设置来完成unlimited（请注意，您可能还必须增加root用户的限制）。
        6. **虚拟内存最大大小检查**：Elasticsearch和Lucene mmap很有用，可以将索引的某些部分映射到Elasticsearch地址空间中。这将某些索引数据保留在JVM堆之外，但保留在内存中，以实现快速访问。为了使此方法有效，Elasticsearch应该具有无限的地址空间。虚拟内存的最大大小检查要求Elasticsearch进程具有无限的地址空间，并且仅在Linux上强制执行。要通过最大大小的虚拟内存检查，您必须将系统配置为允许Elasticsearch进程具有无限的地址空间。可以通过添加<user> - as unlimited来完成/etc/security/limits.conf。这可能还需要您增加root用户的限制。
        7. **最大映射技术检查**：最大映射计数检查可检查内核是否允许进程至少具有262,144个内存映射区域，并且仅在Linux上强制执行。要通过最大地图计数检查，您必须将vm.max\_map\_countvia 配置sysctl为至少为262144。
        8. 客户端JVM检查：OpenJDK派生的JVM提供了两种不同的JVM：客户端JVM和服务器JVM。这些JVM使用不同的编译器从Java字节码生成可执行的机器代码。调整客户端JVM的启动时间和内存占用量，同时调整服务器JVM的性能以最大化性能。两个VM之间的性能差异可能很大。客户端JVM检查可确保Elasticsearch不在客户端JVM内运行。要通过客户端JVM检查，您必须使用服务器VM启动Elasticsearch。在现代系统和操作系统上，服务器VM是默认设置。
        9. 使用串行收集器检查：针对不同工作负载的OpenJDK衍生的JVM有各种垃圾收集器。特别是，串行收集器最适合于单逻辑CPU机器或非常小的堆，这两种都不适合运行Elasticsearch。将串行收集器与Elasticsearch一起使用可能会破坏性能。串行收集器检查可确保未将Elasticsearch配置为与串行收集器一起运行。要通过串行收集器检查，您不得使用串行收集器启动Elasticsearch（无论是从您使用的JVM的默认值开始，还是用明确指定了它-XX:+UseSerialGC）。请注意，Elasticsearch随附的默认JVM配置将Elasticsearch配置为使用CMS收集器。
        10. 系统调用过滤器检查：Elasticsearch根据操作系统（例如Linux上的seccomp）安装各种样式的系统调用过滤器。安装这些系统调用过滤器是为了防止执行与分叉相关的系统调用的能力，以作为对Elasticsearch上任意代码执行攻击的防御机制。系统调用筛选器检查可确保如果启用了系统调用筛选器，则说明它们已成功安装。要通过系统调用过滤器检查，您必须修复系统上阻止安装系统调用过滤器的任何配置错误（检查日志），或者**将**设置bootstrap.system\_call\_filter为false来禁用系统调用过滤器。
        11. OnError和OnOutOfMemoryError检查：如果JVM遇到致命错误(OnError)或OutOfMemoryError (OnOutOfMemoryError)， JVM选项OnError和OnOutOfMemoryError允许执行任意命令。但是，在默认情况下，Elasticsearch系统调用过滤器(seccomp)是启用的，这些过滤器防止分叉。因此，使用OnError或OnOutOfMemoryError和系统调用过滤器是不兼容的。如果使用了这两个JVM选项，并且启用了系统调用过滤器，OnError和OnOutOfMemoryError检查将阻止Elasticsearch启动。这张支票总是强制执行的。要通过此检查，不要启用OnError或OnOutOfMemoryError;相反，升级到Java 8u92并使用JVM标志ExitOnOutOfMemoryError。虽然它没有OnError和OnOutOfMemoryError的全部功能，但是在启用seccomp的情况下，将不支持任意分支。
        12. Early-access check：
        13. **G1GC检查**：ES默认的GC收集器是CMS，若使用G1GC，可能会导致索引损坏，原因是JDK 8附带的HotSpot早期版本存在一些问题，受影响的版本为JDK 8u40之前的版本。
        14. 所有权限检查：所有权限检查可确保引导过程中使用的安全策略不会将权限授予java.security.AllPermissionElasticsearch。以授予的所有权限运行等同于禁用安全管理器。
        15. **集群发现检查**：默认情况下，Elasticsearch首次启动时，它将尝试发现在同一主机上运行的其他节点。如果在几秒钟内找不到任何当选的主节点，则Elasticsearch将形成一个包含所有其他已发现节点的集群。发现配置：
            1. discovery.seed\_hosts
            2. discovery.seed\_providers
            3. cluster.initial\_master\_nodes

ES灾备：数据备份恢复

1. hdfs分布式文件存储系统。
   1. 下载
   2. 解压：tar -xvf hadoop-2.10.0.tar.gz
      1. 配置文件：core-site.xml

<property>

<name>fs.defaultFS</name>

<value>hdfs://172.16.10.184:9000</value>

</property>

* + 1. 配置文件：hdfs-site.xml

<property>

<name>dfs.namenode.name.dir</name>

<value>/usr/local/data/namenode</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/usr/local/data/datanode</value>

</property>

配置文件：yarn-site.xml

<property>

<name>yarn.resourcemanager.hostname</name>

<value>172.16.10.184</value>

</property>

配置文件：mapred-site.xml

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value>

</property>

环境变量：export HADOOP\_HOME=/usr/local/hadoop

使环境变量生效：source /etc/profile

启动命令：$HADOOP\_HOME/sbin/start-dfs.sh

启动命令：$HADOOP\_HOME/sbin/stop-dfs.sh

访问验证：[http://172.16.10.184:50070](http://172.16.10.184:50070/)