【第三期分享主题】

基于ELK的亿级实时日志分析平台实践

【分享嘉宾】

斗鱼数据平台部基础数据架构组负责人 黄歆

【分享内容】

大家好，我是黄歆，目前担任斗鱼数据平台部基础架构组Leader，主要负责斗鱼数据平台部基础环境建设（Hadoop、ELK、容器集群等）及基础服务开发（发布系统、监控告警、任务调度等）。首先自我介绍一下，我2010年毕业后进入传统行业从事JavaWeb方向近4年时间，随后转入互联网行业进入一号店。2015年3月份加入斗鱼，开始从事基础环境、基础服务相关工作。

今天跟大家分享的主题是《基于ELK的亿级实时日志分析平台实践》，主要跟大家分享一些斗鱼在ELK上的实践、架构演变以及一些采坑、调优的经历。此次分享我分成3个部分，分别是：

1. ELK是什么，为什么要使用ELK；

2. 斗鱼ELK日志分析平台实践；

3. 高并发环境下的ELK相关优化；

在讲解ELK之前我先来阐述一下日志在互联网应用中的重要性。在互联网行业里日志数据非常重要，形式也多种多样。通过日志我们可以计算请求量、流量来源分析、了解用户行为、鉴别作弊用户（如：是否是机器人）等等。

在计算PV、UV的场景下，根据业务需求我们通常以离线方式（MR / HIVE）隔天进行报表相关数据生产。但是对于故障排查肯定是希望能够快速的进行日志查询、定位、解决问题，对于实时性要求非常高。

举个例子，对于一个大流量的Web应用通常以Stateless方式设计，这样可以更方便的进行水平扩容。但是随着应用实例数量越来越多，我们查询日志就越来越困难。在没有日志系统的情况下，首先我们需要定位到请求的服务器地址，如果每台服务器都部署了多个应用实例，我们则需要去每个应用实例的日志目录下去找日志文件。每个服务可能还会设置日志滚动策略（如：每200M一个文件），还有日志压缩归档策略。

我们查询一条出错信息就要在茫茫多的日志文件里去找到它，于是使出我们的十八般武艺head less tail grep wc awk count cut，但是如果需要统计最近3天的某个接口的异常次数。。。。

除了上面出现的状况我们还需要考虑：日志量太大如何归档、文本搜索太慢怎么办、如何多维度查询，ELK就是帮我们来解决这些问题的。

1. ELK是什么，为什么要使用ELK

ELK 是elastic公司提供的一套完整的日志收集、展示解决方案，是三个产品的首字母缩写，分别是ElasticSearch、Logstash 和 Kibana。



ElasticSearch简称ES，它是一个实时的分布式搜索和分析引擎，它可以用于全文搜索，结构化搜索以及分析。它是一个建立在全文搜索引擎 Apache Lucene 基础上的搜索引擎，使用 Java 语言编写。

Logstash是一个具有实时传输能力的数据收集引擎，用来进行数据收集（如：读取文本文件）、解析，并将数据发送给ES。

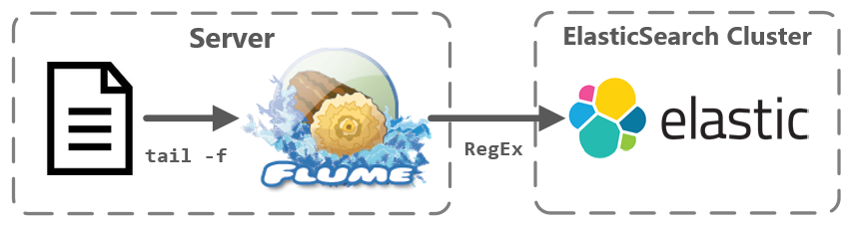
Kibana为 Elasticsearch 提供了分析和可视化的 Web 平台。它可以在 Elasticsearch 的索引中查找，交互数据，并生成各种维度表格、图形。

2. 斗鱼ELK日志分析平台实践；

最初我们引入ELK的原因是实现一个小需求，实时统计服务器500错误数量，在使用过程中发现ELK的功能确实非常强大，随后接入其它业务部门应用日志，解决日志查询慢的痛点，加速错误定位。于2016年底接入全站所有服务器日志，并完成了ES 5.0的升级。

随着对ELK技术栈的不断深入，我们将监控报警系统的数据存储层由HBase替换成ES。监控报警系统的数据存储层主要数据类型是时间序列数据，如某一时刻的应用CPU使用率。通常在做数据展示时我们会将数据先以某一时间粒度（如：5分钟）进行后端聚合汇总再交给前端进行渲染，并且这个时间粒度会随着用户选择的时间区间而变化，针对此类需求我们依靠ES提供的强大数据聚合特性极大减少了开发成本。

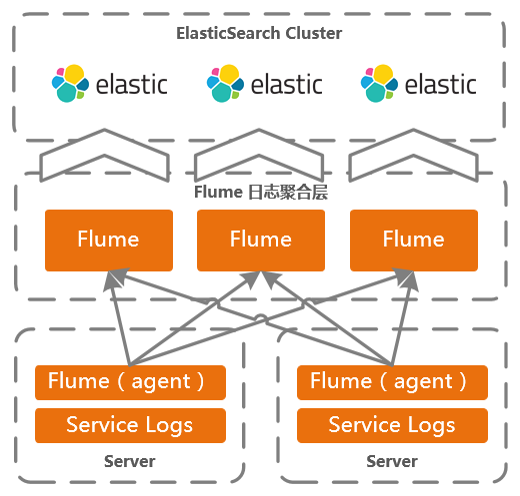
最初我们并没有使用ELK全套技术栈，我们使用了Flume（一个Java编写的日志采集、聚合和传输系统类似上述的Logstash）作为日志收集Agent。Flume使用 tail -f 的方式实时收集日志文件中追加的文本内容。通过Flume配置文件中定义的正则表达式对日志文本进行字段分割，随后写入ElasticSearch，整体架构如下图：



在使用过程中发现Flume偶尔会OOM，原因是使用了Flume的Memory Channel。使用此方式时，Flume会将 tail -f 读取到的数据写入内存，随后输出到ES。但是一旦数据输出速度小于数据读取速度时内存中积压的数据就会越来越多，从而产生OOM。

于是我们将Flume中的Memory Channel替换成File Channel，该方式在数据读取之后，将数据写入Flume中的本地文件存储再读取（个人觉得不是很优雅）。

使用过程中还发现在日志收集过程中Flume抢占了大量CPU资源用于日志解析（通过正则），导致服务器中其他应用资源不足。于是我们搭建了一个Flume日志聚合层，将日志解析工作放在聚合层中完成，Agent只做数据收集、推送。随后架构变成下面这个样子。



聚合层中的Flume实例为无状态节点，如果性能不足可以增加聚合层中的Flume实例数量，但是要在Agent端配置文件中添加新Flume聚合层实例的IP端口信息，并重新启动。

稳定一段时间后，问题又来了。tail -f 方式收集日志在Flume Agent重启时会丢失重启时间段的数据。尝试使用Flume提供的 Spooling Directory Source 也并不能完美的解决问题（需要修改所有日志文件存储方式，每分钟去切割文件，如果跟其他业务团队这样说，估计会被打死）。并且我们当时正在准备进行ES的1.x to 2.x版本的升级，但是Flume并不支持数据写入ES 2.x版本，于是我们果断放弃了Flume投入Logstash的怀抱。

Logstash相对于Flume提供了更加丰富的日志处理器，并且支持最新ES版本（毕竟是亲儿子）。Logstash在进行文本数据收集时并没有使用 tail -f 这种简单粗暴的方式，而是在本地文件中记录了日志文件被读取到的位置，完美解决了上面说的升级重启时丢失数据的问题。

将Flume聚合层替成了Kafka消息队列，我们的监控系统通过Kafka提供的API实时获取Kafka中的队列Metrics，并设置阈值进行报警，这种集中式的队列监控也是Flume聚合层无法做到的。

在这种标准的ELK架构下，日志收集已经非常稳定，但是还有2点不足：

1. Logstash体积太大，依赖Java环境，其他部门的运维同学每次都要先部署一套JDK，不开心；

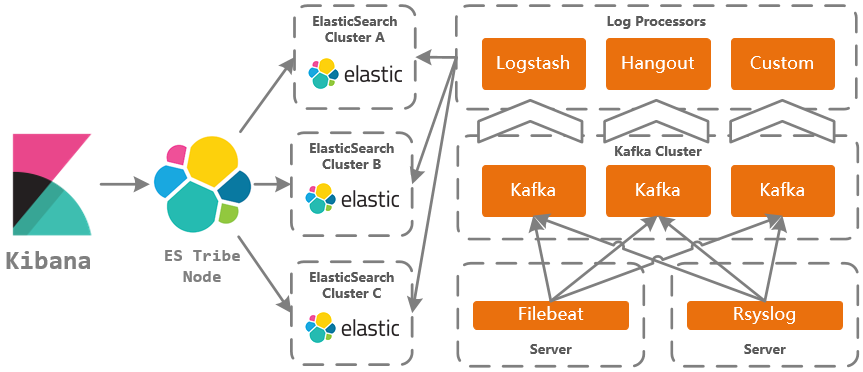
2. Logstash日志解析资源占用偏高；

为了减少Agent体积，方便运维同学部署，我们将Logstash替换成了FileBeat（也是elastic公司的产品）、Rsyslog（用来收集系统日志），而且这2个组件无需依赖Java环境就能运行，安装包10M不到。并且Agent内存占用也得到了极大改善。

针对第二个日志解析资源占用偏高的问题。Logstash的核心代码是用ruby语言开发，虽然是运行在jruby上，但是由于中间涉及到数据结构的转化，性能是跟用原生的java语言运行在jvm上肯定是有所差距的，于是我们在部分场景使用Hangout替换了Logstash。

最后针对日志存不规范的情况，一个日志文件中日志有多种结构，可能正则表达式（或者GROK）不是很好写，即便是写出来之后性能也达不到要求。针对这种日志不规范的这种情况我们自己写了一些基于规则判断的解析器代替正则表达式去完成解析工作，性能获得极大提升，日志解析时的CPU占用率降低了一个数量级。

说到这里，一个能应对高并发的ELK架构终于成型了，上图。



从上图中可以看到，这里部署了多套ES集群（A、B、C）。这样做是为了避免所有业务放在一个大集群中相互影响。如此一来每个业务（或者某些业务）都有独立的集群，更加方便运维。

Kibana与多个ES集群之间通过Tribe Node进行交互，Tribe Node作为集群联邦节点负责将请求路由到正确的后端ES集群上，做到后端ES集群拓扑变化时对使用方透明。

目前整个斗鱼负责日志这一块的ELK集群总共包含50+物理服务器、700T数据、日增数据量为15T。

3. 高并发环境下的ELK相关优化；

首先来介绍索引的。在最初我们以天为单位，使用ES中的Index templates方式自动创建索引。但是一旦ES节点出现故障，由于分片中的数据量非常大，分片恢复速度十分缓慢，于是我们将索引改为按小时划分。按小时划分后每个索引中的数据自然少了许多，当ES故障时可以更快的恢复当前索引，不阻塞后续的数据写入。

针对索引的优化我们关闭了\_all及其他不必要的field以降低索引大小切关闭字段分词。这样设置以后查询只能通过某个具体的Key去查询（类似level:ERROR），在日志场景下完全够用。可以将日志打印成JSON格式，减少日志解析工作量。通过索引模板设置按照某一规则解析Field的类型，如：将xxxCount、xxxSize的Field自动解析成数字类型，其它字段解析成text等等。

关于索引分片，在服务器压力不大的前提下，索引的分片数量尽量少，只要能满足业务正常执行，越多的分片在查询时需要合并的次数也就越多。通常每个分片只设置1个副本，副本分片在ES中默认配置下是以同步方式写入的，每次写入数据时当所有副本写入完成以后才返回结果，也就是说副本越多写入压力越大且耗时越长，所以设置1个副本保证最基本的容灾即可。

如果使用副本异步同步，ES主分片写入完成后不会等待副本分片写入完毕，直接返回结果给客户端。在高并发环境下日志生产的速度很快，在日志解析器速度够快的情况下会直接对ES发起第二批写入请求，这样循环下来会导致ES副本在高峰期永远同步不完，失去容灾的意义，所以不推荐使用。

当索引稳定无数据写入时，对索引进行ForceMerge。ForceMerge操作会合并分片中的segment，简单点解释就是可以把多个小文件中的数据合并到一个大文件中再进行索引排序，可明显提升查询性能。ForceMerge对服务器资源消耗比较高，并且执行时间很长（基于索引的大小），建议在集群压力最小的时候（比如凌晨，通过定时任务）执行。

下面再来谈一下ES节点的部署方式。最重要的一点就是节点角色独立，在一个小型ES集群里每个节点通常既作为Master又负责数据处理，但是在一个大型集群中节点角色混合会发生不稳定的情况。比如当数据压力过大时导致频繁GC、甚至节点掉线。无论GC导致的停顿还是节点掉线均会影响该节点上Master角色所提供的服务，如果该Master是当前活动的主节点的话，掉线就又会产生重新选主的行为，代价还是蛮大的。

我们服务器的配置为2CPU32核 / 128G RAM / 16 \* 6T SATA。每个集群上Master部署奇数个（3~7个），同时设置discovery.zen.minimum\_master\_nodes=master总数/2+1 避免集群脑裂，内存不宜过多，控制GC导致的停顿时间。

每个Data节点分配30G内存，最多不超过31.xxG内存，不同JDK版本的这个边界值不一样，少于该边界值时JVM会采用内存对象指针压缩技术提高内存利用率（至于如何确定这个边界值，我会在最后给出链接）。那么对于128GB内存的服务器我们该如何分配内存呢？我们将每台服务器部署了2个Data节点实例，加起来占用大约一半内存，剩余的内存留给操作系统缓存，在搜索时Lucence会查找Segment文件，这时如果命中操作系统缓存会大幅度提升搜索速度。

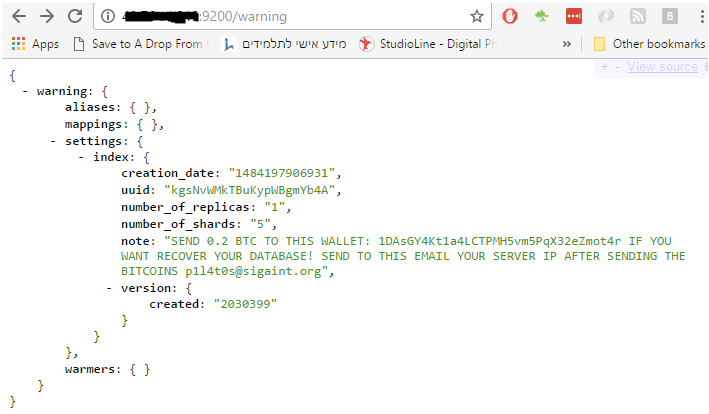
在服务器启动多个Data节点实例的情况下要注意一个问题，一旦服务器宕机有可能导致主从分片均不可用（主从分片被分配在同一台服务器的不同实例上了）。针对这种情况需要开启 cluster.routing.allocation.same\_shard.host 选项，禁止主从分片被分到同一台服务器上，保证服务器宕机时有副本分片可用。

还有，注意给每个ES节点分配不同的磁盘，避免节点之间的IO竞争。

关于ES相关的优化其实还有很多，网上一搜一大把这里就不一一列举。调优的过程中最基础的还是对物理服务器以及集群做持续监控，通过ES提供的CAT API我们可以很方便的获取集群中的各项指标，如查询负载、查询延迟、索引队列长度、请求拒绝次数等等，我们可以将这些指标数据通过脚本定时读取然后回写到ES中，在KIBANA上建立这些数据的可视化图形，这样一个简单的监控系统就出来了。

服务器级别的监控能最快落地的方式就是搭一个Zabbix，主要了解CPU、内存、IO等硬件资源使用情况，方便定位问题，找出性能瓶颈。

在来说一下ES的安全问题。在2017年年初的时候暴露在互联网上的ElasticSearch集群在全球范围遭到大量劫持，黑客在ES中留下了一条与赎金要求相关的索引定义，具体如下所示：



文本内容写道，如果希望恢复数据需要向黑客的账户中支付比特币，如何避免这种悲剧发生呢？

1、最关键的一点就是**不要将ES暴露在互联网中**，这一点应该是针对所有服务器应用。除非直接与用户交互，所有应用都不应该暴露在互联网中，Web应用应该通过Nginx反向代理并且仅暴露需要交互接口，防止黑客调用那些私有接口以及使用Web容器漏洞。

如果服务器配置了外网IP，可以在外网交换机上封锁不需要的端口。如果不方便这么做，则开启服务器上的iptables（记得高并发情况下增大iptables跟踪表大小，这里就不展开了），只允许部分端口的入站请求。还可以将所有服务设置成只监听本地局域网请求（ES中可以设置network.bind\_host=192.168.x.x）。

2. 关闭不必要的ES HTTP端口。用户能连接到ES的HTTP端口，就可以通过REST API去对集群进行操作。仅开启Client节点上的HTTP端口，并通过反向代理方式暴露给用户，配置反向代理屏蔽那些有风险的API接口并开启访问日志，做到操作能够被审计。

3. 及时更新ElasticSearch版本，ElasticSearch完全遵循语义化版本号（x.y.z），小版本的升级基本不会出现兼容性问题。

主版本号（x）：当你做了不兼容的 API 修改；

次版本号（y）：当你做了向下兼容的功能性新增；

修正版本号（z）：当你做了向下兼容的问题修正；

4. 修改默认的ElasticSearch集群名称，避免出现集群中出现陌生节点；

5. 禁用通配符删除索引（小手一抖，数据带走）；

6. 使用非Root用户启动ES；

**问答：**

1、斗鱼有用到RabbitMQ或者Kafka之类的消息队列吗？消息队列适合哪种使用场景，有哪些注意点？Redis如何和消息队列结合使用，Redis使用有哪些注意事项？

答：消息队列的场景还是在于数据驱动的场景，如果上游服务不关注下游的执行结果，很适合使用消息队列。Redis也有消息队列的功能，但是好像并不支持多消费者。只是一个FIFO的队列

2、目前斗鱼这边接入elk平台的业务有哪些共同特性？碰到es 慢写入和慢查询的时候，排查的步骤有哪些？关于es使用object与嵌套结构的分别是什么场景？

答：碰到ES写入慢和查询慢的时候，先是查看监控、硬件资源以及ES中各种线程池的状况。

3、kafka消息队列支持更新队列里面的消息吗？支持优先级吗？

答：不支持

4、在Flume无法支撑当前业务系统时如何进行技术替换以及可行性分析，

答：如果只是Flume写Kafka这一个流程，换Logstash就行了。关于技术选型，可以看一下谷歌指数、GITHUB星星数量对比一下同类框架，社区活跃才是真的好

5、kafka消息持久化支持吗？就是kafka服务挂了之后，消息是否会丢失

答：消息不会丢失的，但是Kafka删除数据是基于时间，比如一周，这个不会一直存储。Kafka还是针对于大数据场景，会发生数据丢失的情况，测试过。如果是敏感数据还是使用RabbitMQ

6、请问有没有好的社区推荐？

答：推荐ES的官方文档还是很全的，最好的社区就是GITHUB以及ES官方论坛然后国内也有个问答网站的 <https://elasticsearch.cn/>，还有关注ES公众号

7、kafka的消息也是存在内存中的吗？

答：是直接落盘， Kafka有个ACK机制，如果是不需要ACK是会丢的，看自己的设置

8、一般程序都是用log4j和log back的，为啥不自定义appender 通过zookeeper +kafka 消费kafka 数据直接建立索引到es 这样不是快？

答：1. 做基础组件一定要使用最小依赖，给应用依赖留有空间，让应用方不要有顾虑。

2. Agent收集方便停机维护，即使Kafka挂了也不会丢消息。

3. 不能保证Kafka的日志Appender日志不会产生其他阻塞情况

9、收集到的数据进入了HSDF了吗？

答：目前没有持久化需求，如果想永久保留日志，可以新建个Kafka消费者写HDFS

10、斗鱼有用日志做调用链吗？

答：有日志调用链，正在做可视化的东西，调用链可以参考PinPoint，用JavaAgent做logger的字节码织入，只不过PinPoint运行时依赖有点重。调用链主要还是记录调用的setp以及他的深度，可以想象成一个树形结构。可以定义一个ThreadLocal变量存放这些信息

11、日志告警的定制化需求，比如多少分钟内出现多少次错误就报警

答：我们开发了自己的告警平台，可以做到“包括日志告警的定制化需求，比如多少分钟内出现多少次错误就报警”这种定定制话的需求，如果想在公司里开发一套，可以先玩玩Zabbix，学习一些它的理念，很有帮助

12、用了jstom这种流式处理吧？

答：用了，还有spark streaming