# 基于Flume+Kafka+ Elasticsearch+Storm的海量日志实时分析平台

袁晓亮

猎聘网

架构中间件负责人

互联网技术联盟

ITA1024讲师团成员

目录树

0. 背景介绍

1. EFK平台简介

2. 架构设计

3. 实时日志浏览

4. 历史日志检索

5. 历史统计、用户追踪、实时日志流式分析

6. 参数配置

7. 监控运维

8. 细节及技巧处理

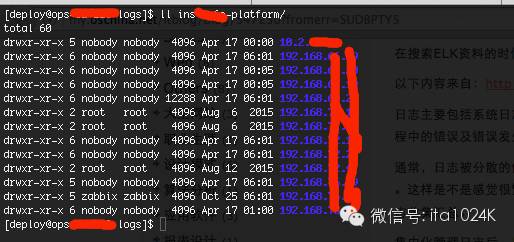
9. 结束语

## 0背景介绍

随着机器个数的增加、各种服务、各种组件的扩容、开发人员的递增，日志的运维问题是日渐尖锐。通常，日志都是存储在服务运行的本地机器上，使用脚本来管理，一般非压缩日志保留最近三天，压缩保留最近1个月，其它直接删除或迁移到日志服务器上。

运维会将这些日志mount到远程的日志服务器上，然后开发人员使用运维分配的账号登陆堡垒机器跳转到日志服务器上查看不同项目不同机器的日志。

下图是日志服务器某一个项目的所有ip日志目录截图，相信大家传统的查看日志类似这样。



如果你要查阅不同的项目，项目机器数十上百的话，将会是非常繁琐和低效的事，并且运维管理、账户安全都是需要解决的问题。另外如果你要对这些日志进行实时分析统计，你会发现无从下手，用shell等脚本语言只能处理非常简单需求，难于满足业务的发展，所以当务之急是要将所有日志集中化管理，将所有服务器上的日志收集汇总。

为此需要设计一个集中式海量日志实时处理系统，它需要满足产品需求（实时看日志、统计历史日志、实时行为分析、用户轨迹跟踪等）、满足运维需求（项目上下线日志管理、日志开关在线管理等）、性能需求（具有高吞吐能力、高扩展性、高容错性）。

## 1EFK平台简介

这样集中管理后，需要考虑解决实时查看日志类似tail -f命令，日志统计或检索，类似grep、awk、wc等Linux命令，并对更高要求的多机器日志全文检索、排序和统计等问题。

目前业内比较流行开源实时日志分析ELK三件套为我们提供了思路和帮助，结合猎聘自己的业务模式及现有服务架构，经过多种组件的优劣对比，我们设计了自己的EFK海量日志实时分析平台。

先大致介绍下EFK组件，EFK是ElasticSearch、Flume、Kafka的简称，这三者是核心部件，但并非全部组件，后文还会介绍架构设计中的其他组件。

ElasticSearch是个开源实时分布式搜索引擎，它的特点有：

分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。

Flume是Cloudera提供的一个高可用的，高可靠的，分布式的海量日志采集、聚合和传输的系统，它支持在日志系统中定制各类数据发送方，用于收集数据；同时提供对数据进行简单处理，并写到各种数据接受方（可定制）。

Kafka是一种高吞吐量的分布式发布订阅消息系统，它适合处理海量日志发布订阅，提供消息磁盘持久化、支持物理分片存储、多组消费等特性。

目前EFK平台接收日志的平均QPS为6w/s，日增20亿条日志，日志大小970G，磁盘存量10T日志，3台物理机器支撑。该平台通过Flume为线上集群的上千个实例实时收集日志，日志种类目前主要分为nginx和tomcat日志（后续还会迭代其他组件日志），并将日志实时传入Kafka集群。

日志消费方式分为三种：

第一会通过Web平台使用浏览器实时在线浏览日志，基于redis发布订阅方式实现，浏览延迟基本小于5s；

第二会通过Web平台全文检索、统计历史日志，基于将kafka日志灌入ES集群实现，日志搜索延迟平均控制1分钟左右；

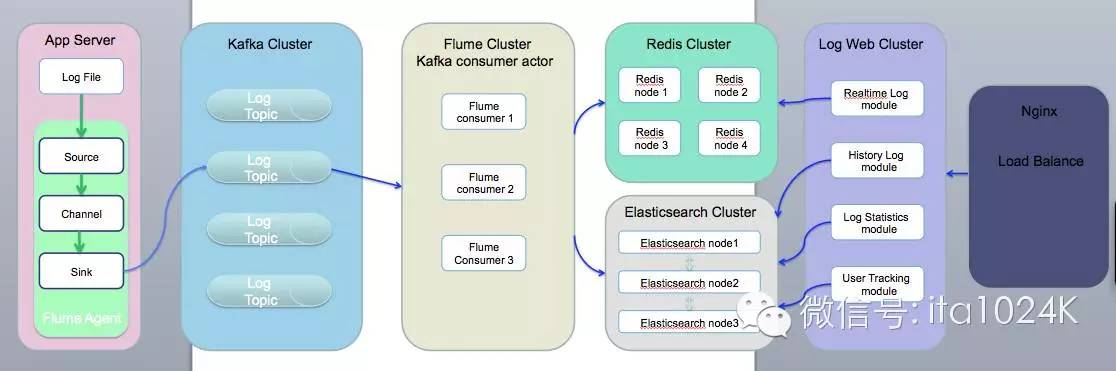
第三会在后台进行实时统计分析日志，做一些内部规则处理，基于Storm流式计算实现。

## 2架构设计

普通的ELK架构是将Logstash部署于各个AppServer节点上搜集相关日志，经过过滤处理后发送给远端服务器上的Elasticsearch集群进行存储，然后用户使用kibana从ES集群中进行日志查询生成图表。

而相对说，EFK架构会有所区别：我们使用Flume实时日志采集系统收集日志，并且引入了Kafka消息队列，将flume采集到的日志存入kafka，然后使用flume集群消费kafka中的日志，经过过滤处理分别将日志传递给redis集群和Elasticsearch集群，最后使用自研的Log Web程序将实时日志和历史日志呈现给用户。

下面是系统整体的架构设计图：



为什么没有使用Logstash而使用开源的日志收集系统Flume是出于几点因素考虑：

* Logstash是ruby语言开发的，跟团队使用语言java不符合，不容易排查问题及二次开发 ；
* Flume是java开发，高可用，高可靠的分布式日志采集系统，业内也大量使用，并且它系统框架好，模块分明，易于订制开发；
* 公司有统一的开发框架，所有日志打印都是标准的规范格式，不需要太多的订制filter处理日志，所以Logstash使用场景就不大了。

我们使用flume的 Exec source方式组织数据，也就是tail -F [file]，实时进行日志读取传输，达到准实时日志同步，后文会分享flume这块代码逻辑。

引入Kafka是因为几点考虑：

* 日志采集后的处理方式多样化 ；
* 线上业务集群规模较大，日志产生量巨大，如果直接同步日志对下游服务负荷较重，容易因为故障导致日志阻塞延迟和丢失，所以引入了kafka ；
* 消息可以持久化，并且可以进行日志回溯。有了消息队列，上游服务和下游服务完全解藕，网络传输会更稳定、更高效、更均衡，避免级联效应。

下面介绍整个系统的分工：

A. Agent层，在每台服务器上部署一个flume进程，负责对单机上所有项目日志进行收集，使用Command： tail  --pid{pid} -F {file} ，另外如果在文件名有变化时，会从文件的第一行开始读日志，命令会变为tail -n +1 --pid {pid}-F {file} 。Flume的Source目前做的工作比较简单，实时读取日志封装成Event批量放入Channel，目前提供tomcat错误日志、调试日志、事件日志、SQL日志、慢日志、nginx等类型日志接入。

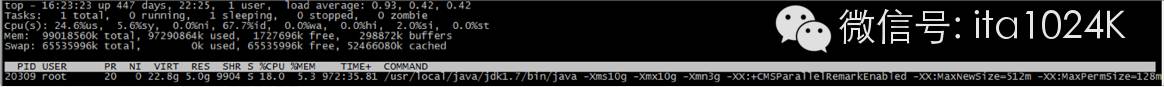
Flume的Sink主要工作是从Channel里拿到Event，然后在原始日志前面加上header，批量发往Kafka。header的信息包括：{numberId , ip ,appName, logType}，numberId：表示该IP这个日志类型的一行日志的唯一编号，id是递增的；ip：表示日志所在的机器ip；appName：表示日志归属的应用名称，如：ins-xxx-platform；logType：表示日志的类型，如：tomcat\_event。这个header在后续其他系统进行日志处理时，起着表明日志身份的作用。

目前Agent 部署规模：上千实例，单机接入10+的实例， 单实例6+的日志类型，业务日志量20亿+/天，每天高峰6w+/秒，日增20亿条日志，日增970G日志。

下面是单机flume运行截图：

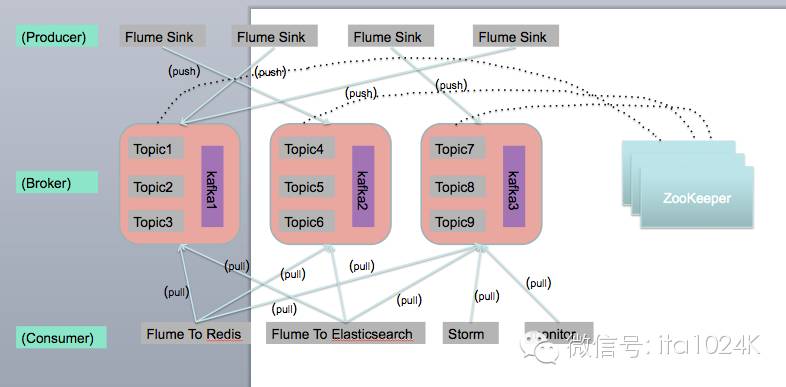
基于Flume+Kafka+ Elasticsearch+Storm的海量日志实时分析平台

flume运行高峰时负载情况截图：



B. Kafka层，这一层架构非常简单，由三个实例组成集群，日志备份保留3天。Topic定义规则是：$appName\_$logType，确保一个项目的一个日志类型都在一个topic里，目前Partition分区为1，是为了保证日志顺序，如果以后日志量变大了可以根据一个项目的ip数进行Partition设置，提高kafka处理能力。

Consumer目前配置了多个group组，分别消费日志到redis集群、Elasticsearch集群、Storm集群、监控项目。

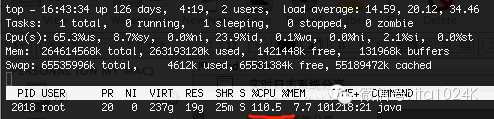


目前Kafka部署规模：3台物理机，每台机器一个实例，组成一个集群。

下面是单机Kafka运行截图：



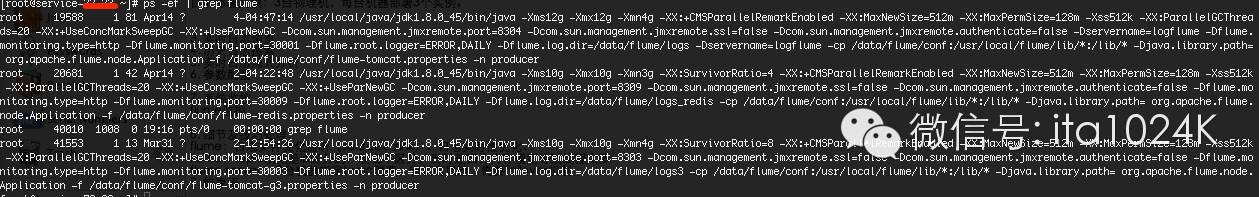
kafka运行平均cpu情况截图：



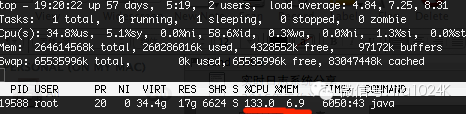
C.Flume Cluster层，主要是将kafka的日志（app和nginx）分别消费到redis集群、Elasticsearch集群。这一层大家可能有疑问，为什么用flume来开发？因为我们对这一层的定位是可以分布式部署，可以水平扩容，以java进程模式运行，并且可以实时消费kafka数据到不同目标源。

鉴于flume天然的source、channel、sink架构设计，完美支持目前设计需要，只需要在它之上开发source和sink，节省开发成本。从kafka消费数据，我们开发了KafkaSource，将日志传输到channel，sink这块我们开发了RedisSink和ESSink，分别将日志发布到redis里提供实时订阅、发布到es里提供历史日志检索。

目前部署规模：3台物理机，每台机器部署3个实例。



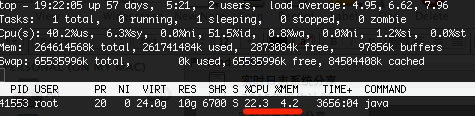
flume2es运行平均cpu情况截图：



flume2redis运行平均cpu情况截图：



flume2es（nginx）运行平均cpu情况截图：

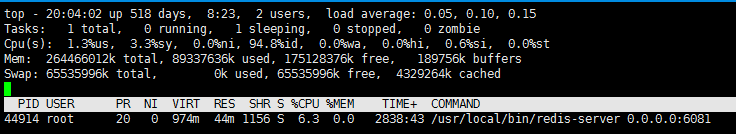


D. Redis层，主要用到了它的发布订阅功能和list数据结构。主要使用在实时日志浏览的场景，下文会介绍功能实现。

目前部署规模：6个实例，在6台机器上，一致性hash部署

基于Flume+Kafka+ Elasticsearch+Storm的海量日志实时分析平台

redis运行情况截图：

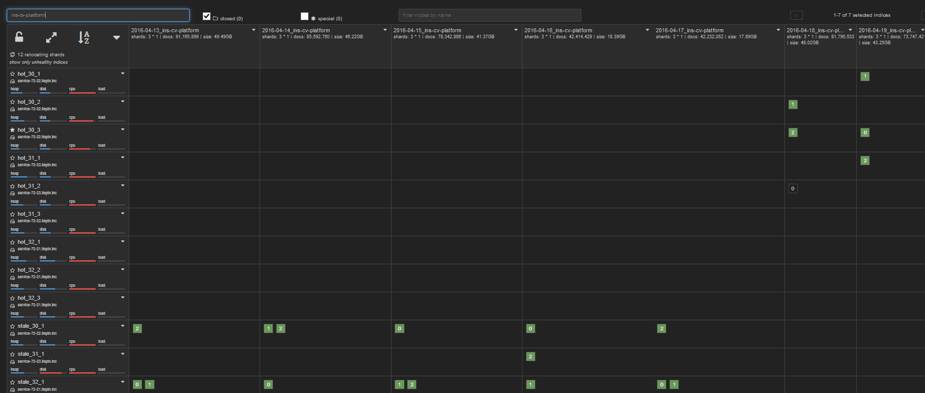


E. Elasticsearch层，主要存放历史日志，使用场景在对历史日志进行检索。目前日志是分片存储，分片规则是按机器路由，默认针对中等及偏小规模的项目日志分3片存储，大规模日志分6片存储，这取决于机器规模。

目前部署规模：3台物理机，每台机器4个实例，每台配置1.5TPCIE + 8T普通SAS硬盘，1天内的日志放在PCIE上，加快索引及检索速度，6天前的日志放在普通硬盘上，目前es集群只保留一周的日志数据。



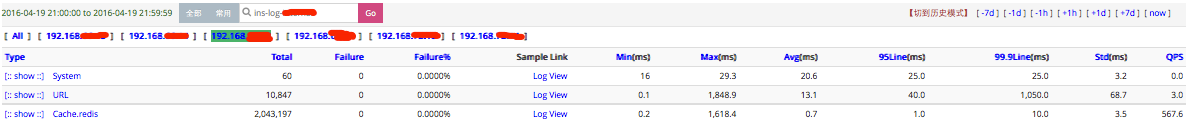
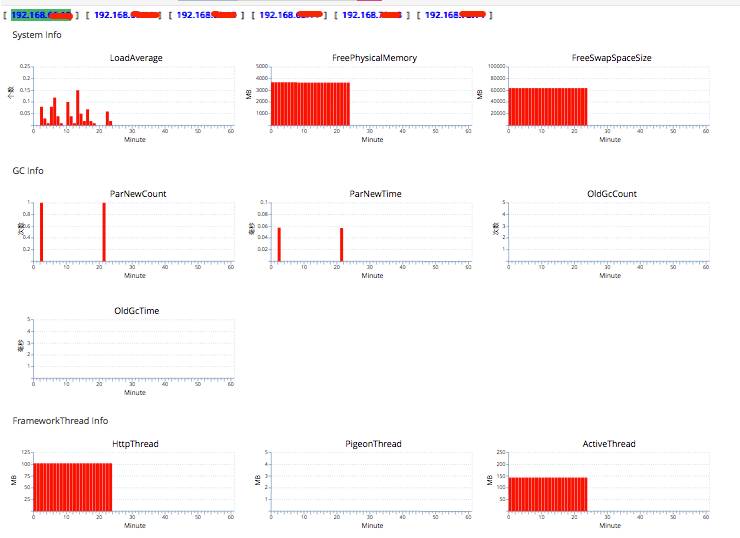
ES中某一个项目日志分布情况截图：



F. Log Web及Nginx层，主要曾担着实时日志展示、历史日志检索、历史日志个性化统计、用户行为追踪等功能。这块完全自主开发，满足研发、测试、运维的需求，目前一直收集用户反馈，持续迭代中。

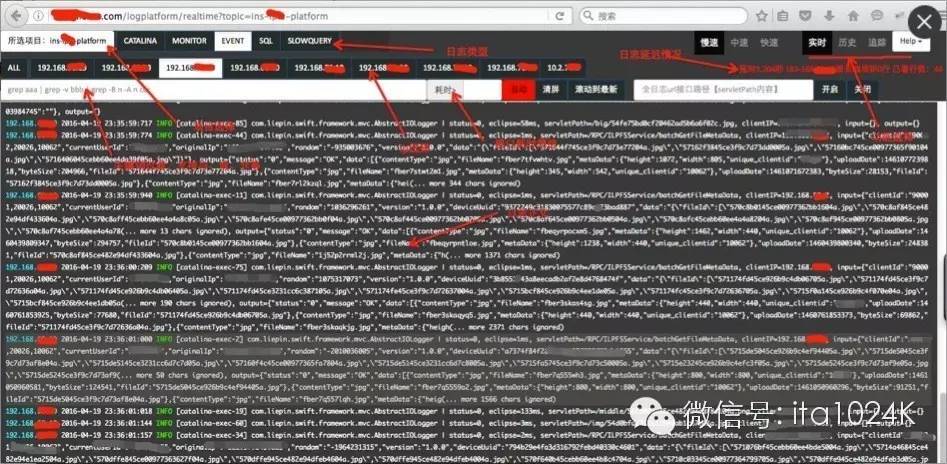
目前部署规模：分布式部署6个实例，nginx反向代理，iphash的路由规则。

下面是单个实例运行监控情况截图：

## 3实时日志浏览

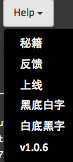
下图是Log Web系统里，实时日志页面截图：



实时日志页面功能：支持项目、日志类型、ip（单个或组合）选择，搜索支持grep语法、关键词高亮、接口耗时过滤、esc快捷方式暂停、清屏、日志滚动到最新（当浏览器显示慢时）、日志显示速度控制、日志延时显示（日志产生到页面显示耗时）等功能。



帮助页面还提供操作手册、在线反馈、日常上线管理、页面颜色控制等功能。



实现日志实现逻辑：

这块实现经历了三个阶段：单机内存kafka消费模式、分布式内存kafka消费模式、分布式redis发布订阅模式。页面都是使用Ajax定时轮询的方式获取日志，然后显示在页面中。

a. 单机内存kafka消费模式比较复杂，用户在log web页面查看一个实时日志，后台立即从kafka订阅topic消费日志，放入jvm里维护的一块日志消费队列，做一层缓冲。页面ajax每次请求时会带上上次已查看日志的偏移量，在内存中算出这次需要返回的日志列表，如果是多用户查看一个日志，会因为每个用户页面滚动的速度不同，算出不同的日志列表来。

这种方式会有些瓶颈：

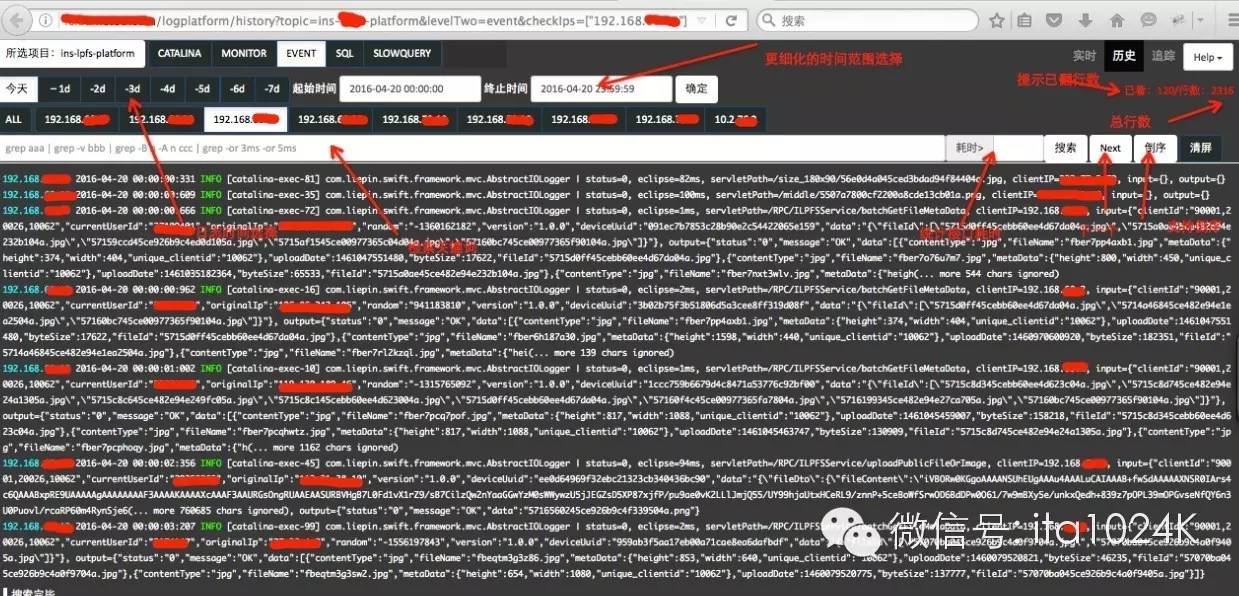
* 算法复杂，问题难定位
* 内存开销大，gc比较频繁
* 线程数过多，每个日志类型是一个线程占用
* kafka链接数较大，初始订阅延迟较长。

b. 分布式内存kafka消费模式，这个方案也是临时过渡阶段，因为随着项目日志接入越多，日志topic是几何增长，hash分配到每个logweb节点上的topic还是会非常多，性能开销非常大。

c. 分布式redis发布订阅模式，是目前正在使用的方案。此方案比较简单，性能开销非常小，服务运行比较平稳。过程大致是这样：用户在页面查看一个类型的日志，log web会启动一个线程处理这个用户请求，它会通知后台flume集群中的flume2redis进程开始消费kafka数据，将日志发布到redis里，这样log web里处理用户请求的线程就能订阅到redis的日志，并且把日志缓冲到redis list队列里，页面ajax请求轮询从list队列里顺序消费日志，使用redis LTRIM命令，并且在用户暂停滚动日志5分钟或切换日志时，主动销毁线程、取消订阅、删除list队列。另外这块nginx配置成iphash的方式，一个用户始终落到一个节点。

## 4历史日志检索

下图是Log Web系统里，历史日志页面截图：



历史日志页面功能：支持项目、日志类型、ip（单个或组合）选择，搜索支持grep语法、关键词高亮、接口耗时过滤、日志时间范围选择、快捷方式支持、搜索正序倒序、页面滚动行数、检索总记录数等功能。实时日志页面和历史日志页面相互切换，所选择的参数项都会带着，避免重复选择。

历史日志实现逻辑：

将用户选择的参数及填写的检索内容翻译成ES的Query DSL查询表达式，放到es里检索，将数据再返回到页面上。

下图是检索“java”关键词截图：



翻译成DSL表达式截图：



## 5历史统计、用户追踪、实时日志流式分析

这块不详细讲了，因为涉及公司内部产品，只说下思路。

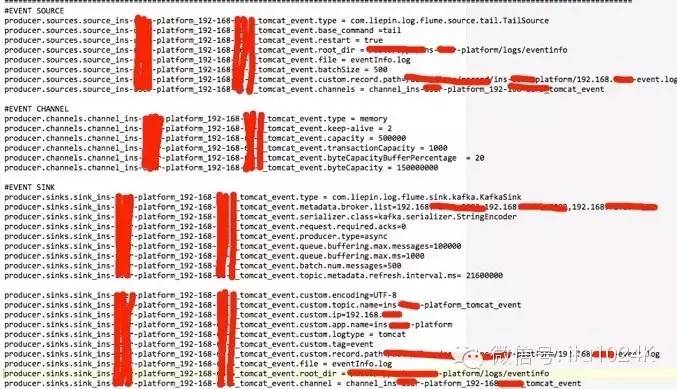
历史统计：将日志统计分析及报表功能，抽象成统计模板，用户只需要创建任务，录入统计参数，历史统计模块就会自动帮用户在规定的时间或周期性统计出报表或原始日志，提供展示、下载、邮件发送功能。

用户追踪：目前公司统一的开发框架会对每一次用户请求及内部RPC调用都会记录详细事件日志，收集这些日志，将这些数据抽到es里，然后在用追踪页面展现用户访问的行为轨迹。

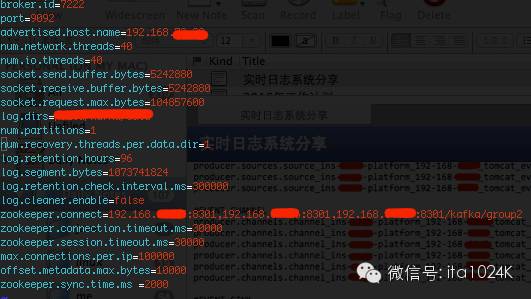
实时日志流式分析：收集nginx日志，实时分析日志中的业务接口访问量、频次情况、域名流量分布、用户行为分析及内部规则处理等。

## 6参数配置

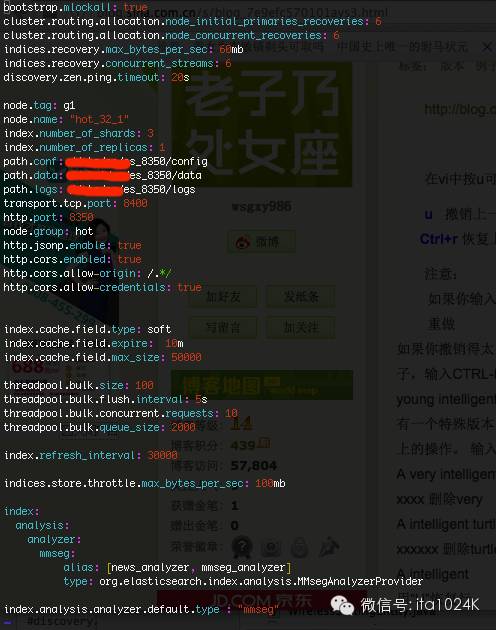
flume的source、channel、sink参数配置截图，这是一个实时日志量比较大的项目配置



kafka的server.properties配置



es的elasticsearch.yml配置



es的模板配置：


## 7监控运维

服务搭建起来了，后期的高效、稳定的运行也是至关重要的，特别是对这种相对复杂的大型系统。整个架构体系涉及到的中间件比较多，各个环节都需要监控起来，我们除了使用第三方监控软件，也自己开发了一些维度的监控手段，做到在用户之前发现问题。

A. 系统负载、cpu、磁盘、内存等使用zabbix监控报警

B. Kafka使用的监控工具：KafkaOffsetMonitor

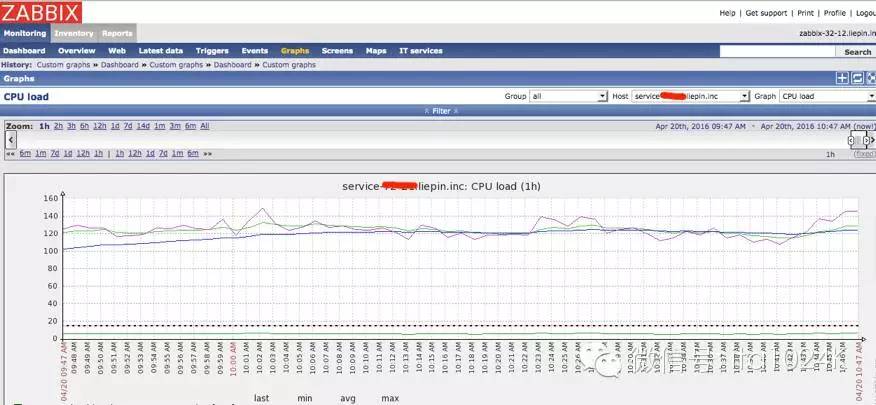
C. Elasticsearch使用的监控工具：kopf、bigdesk

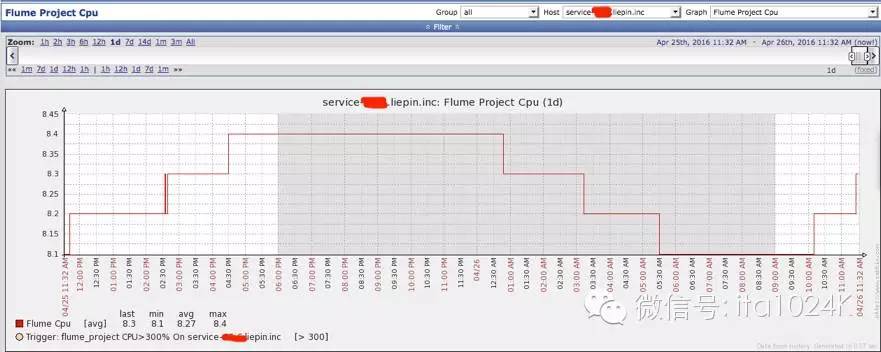
D. redis使用zabbix监控cpu及内存使用情况

E. log web使用zabbix、cat监控服务各维度性能指标。

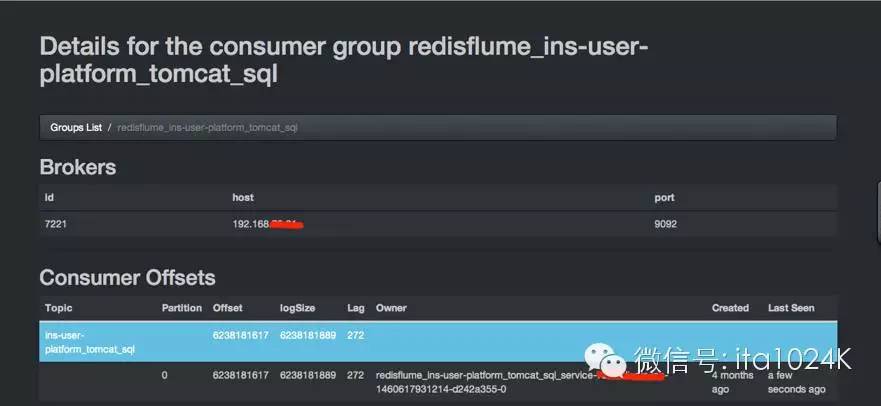
另外我们也开发了针对整个日志系统的监控大盘及日志流量实时统计图

这个是zabbix监控机器cpu load情况截图：

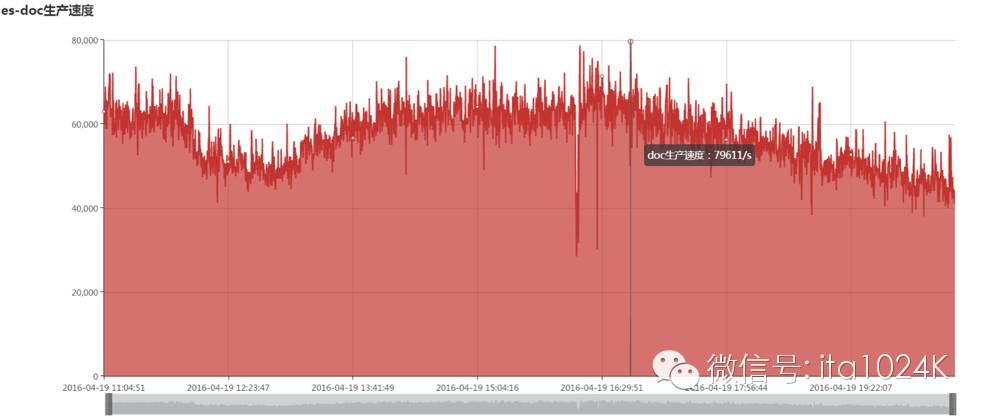




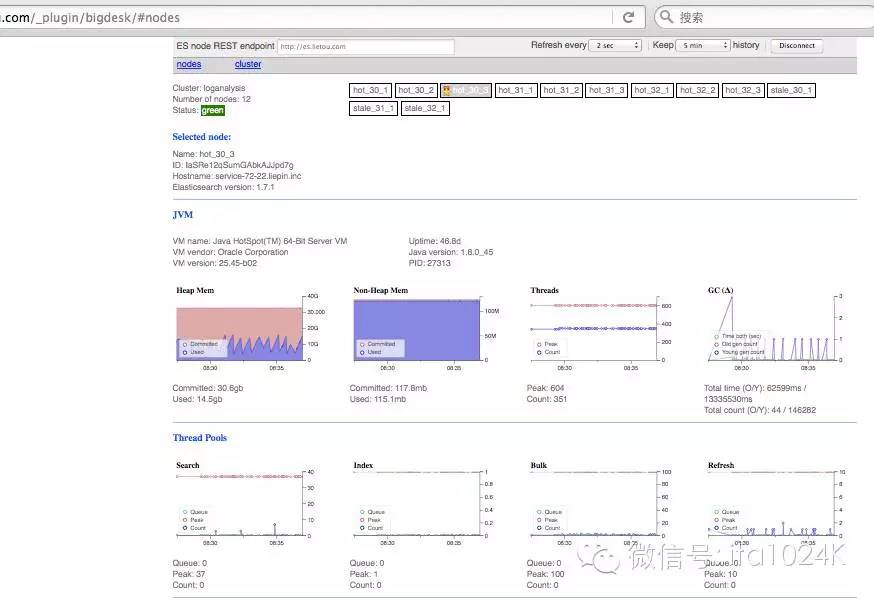
使用KafkaOffsetMonitor查看kafka的topic情况截图：

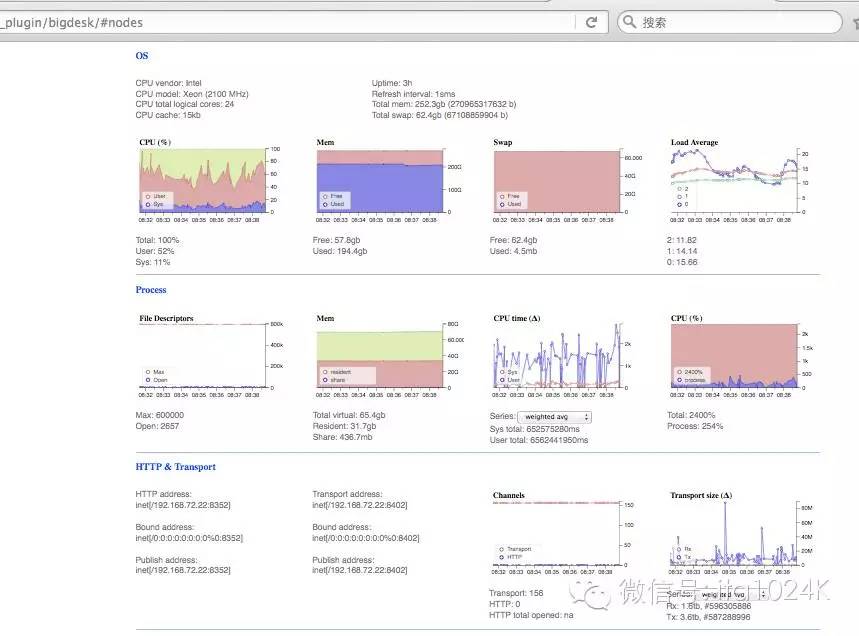


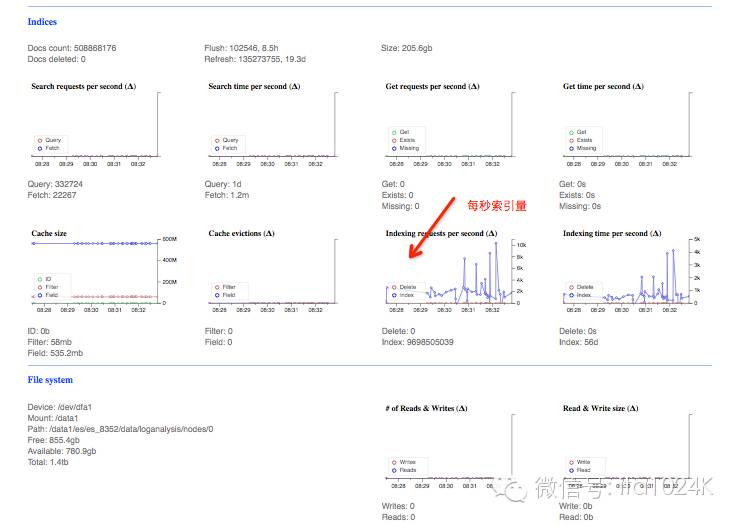
es实时写入document速率情况截图：



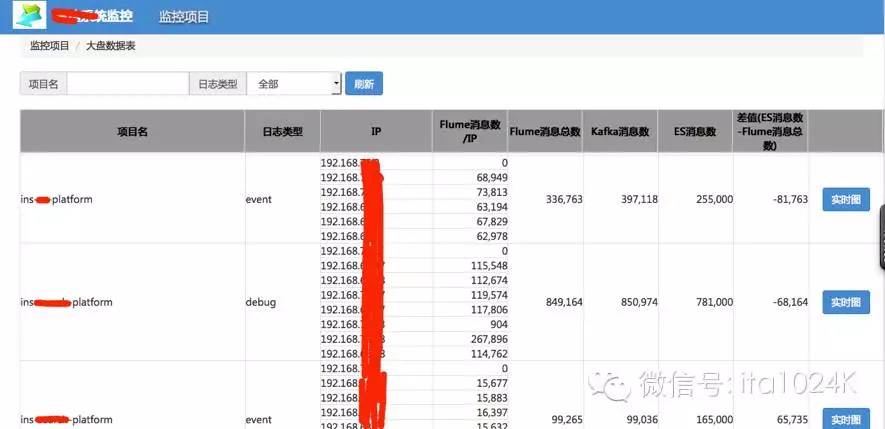
es集群及节点运行情况截图：







下图是项目日志监控大盘，实时列出日志经过Flume、kafka、es流水量及堵塞情况，可以一目了然知道那个项目那个ip消息在那个环节有延迟。



下图是flume、kafka、es三个环节收到日志实时数量统计图，可以对几天的数据进行对比。正常情况下三条线是拟合状态，如果某一个线出现波动，证明它消费数据慢了，需要check。



下面是实时日志浏览和历史日志检索旁路监控报表，会模拟用户发起请求，监控页面功能是否可用，定时监控预警。





## 8细节及技巧处理

a. flume在处理日志时，需要区分日志名字，如事件日志以小时切割，当前文件名始终是 eventInfo.log，这样可以使用tail -F 命令。但错误日志以天切割，当前文件都是带时间的，如catalina.04-20.out ，这样当切换日志时，需要额外处理，变化tail-F 的文件名。为此我们采用文件夹监听（WatchService）的方式，动态切换文件，并且动态Source释放创建。

b. flume日志采集传输实时性优化：source在将Event批量传输到Channel时，需要加超时控制，避免批量大小一直不满足，导致日志一直不发送。sink里将日志传输到kafka一样，需要注意。伪代码如下：

if (eventList.size() >=bufferCount || timeout()) {

   flushEventBatch(eventList);

}

private boolean timeout() {

   return (当前时间 - 上次写channel时间) >= timeout(3秒);

}

c.flume目前配置内存模式，为了提高采集速度，channel是内存方式，但当flume重启或宕机时会丢失日志。目前考虑使用断点续传的方式，在磁盘上存储已读取偏移行数，此偏移是在sink往kafka成功后更新。

d. 针对日志量大的项目需要调大MemoryChannel的capacity

e. kafka的Topic定义以前是ip区分的，当时是因为内存模型导致线程太多，现在不区分了，这块需要考虑维护成本，topic的数量太多，因为后期需要监控运维。

f. 如果要保证日志顺序，假如topic不区分ip，这时topic必须设置一个分区；假如topic是区分ip的，那么可以根据ip来设置分区数，这样可以提高kafka吞吐能力。

g. kafka消费的flume集群，目前是人工指定topic消费的，如果flume宕机会有风险，需要采用failover机制。

h. elasticsearch磁盘需要分快盘和慢盘，也就是固态硬盘和机械硬盘。当天的日志放到固态硬盘里es写入快，查询也快，页面查询体验好，6天前的日志迁移到慢盘存储，因为相对来说查询量少，并且无更新删除操作。这块需要做一个执行计划每天晚上定时执行。

i.  elasticsearch索引生效也就是commit操作不需要特别频繁，如果太频繁会对es压力较大，目前我们控制在30秒1次commit。因为日志产生到能在es里检索出来，不需要太实时，开发人员使用的场景都是滞后的，不像前端业务搜索需要达到准实时。

j. 页面查询过滤语句一定要贴近linux的grep语法，这样开发人员在使用日志系统的学习成本会非常低。

k. 页面日志展现优化

问题：日志产生速度最快会达到5k条/s，实时页面如果展现，用户cpu高，浏览器占用高

优化过程：

ajax：数据异步批量获取，前端数据容器（数据缓存），dom模板化异步创建、dom释放、滚动、事件代理。

忽略策略：当数据延迟超过阀值说明数据生产太快，消费慢，则只看最近的数据，历史数据可以到历史日志中查看。

请求数据量伸缩：依据客户端消费情况，如果消费延迟到一定阀值，则每次向后端多请求一些数据。

压缩：在一定量时自动开启 gzip，需要权衡对客户端cpu的损耗。

l. 需要监控项目日志日生产量大小，同比监控预警，避免突然某个项目日志产生过大，超过预期配置和资源情况，也可以及时通知业务端查找问题及处理。

 上面提到的kafka集群、zookeeper集群、elasticsearch集群、flume集群，目前我们只用到了三台物理机，机器配置：24核cpu、260G内存、9T硬盘（此处应该有掌声），机器load average低峰10+，高峰时150+。

## 9结束语

一个好的系统诞生离不开三个阶段：工具化、产品化、运营化。我认为技术人员不光光只关注技术本身，还应该多关注下产品体验、用户反馈、系统运营，前期找一批天使用户，让他们多试用及反馈，这样你开发的系统才是一个真正易用、好用、想用的产品，系统的NPS才会高。 整个系统设计开发中遇到了很多问题及难题，我们都是通过架构调整和代码逻辑梳理解决的。

另外在这么一个庞大的系统运行中，一定会遇到资源紧张（cpu load高）的情况，这时建议大家不要随意扩容机器解决，这样会很容易漏掉一个坑或隐患，其实这些问题可以从架构设计、参数配置、代码优化着手考虑，相信性能的提升空间会非常多。

今天的分享就到这里，希望对大家有所借鉴，也希望大家多提点意见，欢迎大家拍砖。

## Q&A

问：就是刚才说的集群用了三台，zookeeper、flume，他们分别是三台物理机吗？

袁晓亮：是的。

问：header的信息包括：{numberId , ip , appName, logType}，numberId：表示该IP这个日志类型的一行日志的唯一编号，id是递增的；请问id递增这里有什么特殊设计吗，怎样的id递增？

袁晓亮：这个放入es的自增唯一id，排序用。id递增的方法，单机内存自增、分布式id自增（Snowflake算法）。

问：hive不是负责计算 计算最终都要落到Hadoop map reduce 或spark计算？

袁晓亮：后面会考虑使用es和hadoop的集成，做分析。

问：flume直接用tail，重启的时候怎么保证日志读完整呢？是准备结合后面要加发送到kafka行号一起解决吗？

袁晓亮：增加断点续传功能，在agent本地记录文件，消费日志的行号，这个在sink发给kafka后更新。

问：机器配置：24核cpu、260G内存、9T硬盘（此处应该有掌声）这里的260G不是指的内存，是指的SSD硬盘吧？

袁晓亮：内存是260G,ssd是1.5T。

问：kafka集群、zookeeper集群、elasticsearch集群、flume集群都放到三台物理机上，各个集群之间有隔离吗？

袁晓亮：目前是因为机器资源问题，都是分布式放在3台机器上，没隔离。

问：既然是分布式部署，对日志的顺序要求还非常严格吗，如果partition都是为1，怎么样多进程处理呢？

袁晓亮：用户需求非常严格，根据ip来分多channel处理，每个ip是顺序的