看大众点评如何通过实时监控系统CAT打造7*24服务

2015-06-08 尤勇 高可用架构

本文根据尤勇在【QCon高可用架构群】中的分享内容整理而成。

> 尤勇是大众点评网资深工程师，开源监控系统CAT(Central Application Tracking)的开发者，目前主要负责C和大众点评私有云平台的开发。

CAT是一个实时监控系统，它侧重于Java应用的监控，基本接入了点评所有核心应用。CAT已经在中间件框架（MVC框架、RPC框架、数据库框架、缓存框架等）中得到广泛应用，为点评各业务线提供系统的性能指标、健康状况、基础告警等。

CAT很大的优势是它是一个实时系统，从数据生成到服务端处理结束是毫秒级别；第二个优势，数据是接近全量统计。

CAT背景介绍

大众点评监控系统CAT是由@吴其敏@携程（前大众点评首席架构师，现携程架构负责人）主导设计。我们平常都称吴其敏为老吴，老吴之前在eBay工作超过10年，对于eBay的CAL系统有深入了解，CAL是CAT系统当初的原型。

为什么要做监控

1. 线上发布了服务，怎么知道它一切正常，比如发布5台服务器，如何直观了解是否有请求进来，访问一切正常。
2. 当年有一次将线上的库配置到了Beta，这么低级的错误，排错花了一个通宵，十几个人。
3. 某个核心服务挂了，导致大量报错，如何确定到底是哪里出了问题。
4. SOA带来的问题，调用XX服务出问题，很慢，是否可以衡量?
5. 应用程序有性能瓶颈，如何提供一些有效工具发现?

   ……

监控应该是一个很宽泛的问题，任何可能出问题地方都需要加入监控。

服务端监控

我们把服务端监控的报表分为两类：

1. 故障快速发现类，这类主要是面向运维，让运维直观看到生产环境出现的问题。
2. 系统问题分析类，这类主要是面向开发，让开发能了解自己系统实时运行状态，发现问题，分析瓶颈等。

故障发现类的报表有如下几个：

1. 实时业务指标监控： 核心业务都会定义自己的业务指标，这不需要太多，主要用于24小时值班监控，实时发现业务指标问题，图中一个是当前的实际值，一个是基准值，基准值是根据历史趋势计算的预测值。

2. 实时报错大盘： 所有应用的topN的报错大盘，下图是一个出现故障的图。

3. 实时数据库大盘

   A. 实时数据库大盘，实时知道数据库访问情况的大盘。如何确定存在问题，是根据实时的数据在加一些配置的访问规则。
   B. 这里不要用DB服务端性能采集的数据（比如io，load，qps等），要用应用程序访问这个database得出的响应时间、错误、访问量的数据，这里称之为端到端的数据。
   C. 应用程序采集的数据和服务端的数据得出的结果会有很大很大的差异。
   D. 后面的闪电符号是一个url link，这边可以直接跳转到运维的自动化平台上，做database故障降级处理。
4. 实时核心网络拓扑大盘
   这里采集了核心接入层网络交换机的一些信息，将一些状态定制到监控大盘上，主要的采集指标包括：进入口流量、丢包、错包等。

以上讲了我们做的给运维，更准确的是24小时值班监控团队的监控大屏，主要目的是快速发现问题，这里不需要很华丽的数字，主要用精确的红色来代表发现故障，需要立刻通知解决。

什么东西才可以作为一个大盘：这里需要看公司整体运维的故障情况，TopN以及业务指标应该属于通用，数据库和网络大盘是点评在实际经验中经常容易出故障的地方，所以我们做成大盘这样比较直观的形式，用于发现问题。比如平时因为发布引起故障比较多，他们也做了一个发布大盘，实时监控线上的发布情况。

下面我会讲到服务端分析问题的报表，在这之前我介绍几个CAT监控的几个基础概念。

首先第一点就是监控的模型，监控最需要解决的两个问题，响应时间和访问量。

1. 响应时间，一段代码的响应时间，一段代码可大可小，比如一段sql，一个服务访问，一个url整个请求。
2. 访问量，走到一行code的次数，比如抛出的异常，走到代码某个if的路径等。

在这个基础上我们定义监控最基础的模型，Transaction和Event，这个也是后续问题最多的地方。

后面还扩展定期执行某些代码以及一个指标变化值，第三个可以理解为心跳（主要是框架使用）。最后一个一个指标变化值，主要用于业务指标的监控。 在这个基本模型的基础上，我们定义了我们监控的Java的API，Sample大致如下：

这个API里面Transaction以及Event都有两层分类，为什么是两层分类，从实际的经验中两层分类可以解决大部分的问题，所以定义了两层。 当业务程序完成了API的情况，后面会产生CAT的原始监控请求，这里我们称之为Logview。

这个Logview是一个树的结构，根节点一般都是Transaction，Transaction可以嵌套子的Transaction，Event就是一个单个节点，不能嵌套。这个longview将应用程序的执行路径按照时间序列组成一个Logview。
它还有另外一种展现形式，它很直观看到整个调用路径中最慢的地方。

整个点评迈向SOA化，Web调用服务，服务再调用其他的服务，调用关系的跟踪很有必要。

在点评Call表示一个远程访问，当遇到一个远程访问时候，会有一个show节点，点击show就能知道服务端对应这次Call的它的访问序列，如果这个服务还调用其他的服务，一样可以跟踪下去。比如如下请求是URL，调用了groupService的服务，groupService的服务还调用了userBaseService的服务

以上部分讲诉了整个CAT监控最核心的模型以及在此基础上产生的Logview，后续监控产生所有的原始数据都来自于Logview。在点评监控，目前是做的全量监控，也意味着每次URL请求，SQL请求、缓存请求、远程请求都有对应的实时Logview发送CAT以供实时处理。

监控里面，尽量做全量监控，这样看到的问题才足够接近真实。监控尽可能做到实时，延迟一

分钟或者几分钟都意味着故障处理总时间的延长。

CAT每天大约有300亿Logview，3500GB消息，Logview太多了人已经没办法处理，所以基于Logview肯定有一些实时分析的report，下面我讲诉监控系统基于这些logview产生几个核心的report。

1、TransactionReport

这个是TransationReport的第一个视图，左上角可以选择项目进行切换，一个项目定义为一个独立的war包上线单元。报表时间是一段时间，CAT统计报表的时间跨度是一个小时。根据一个小时的report，后续可以合并为天、周、以及月。

Machines: 有一个All以及具体的机器明显，可以看所有机器，也可以看单个某台机器，如果某台机器出现的问题，则可以很容易看到。

这里展示了一个Transaction的第一层分类的视图，可以知道这段时间里面一个分类运行的次数，平均响应时间，延迟，95线以及99.9线。95line表示95%的请求的响应时间比这个值要小。

上图这里面是点击URL进去的第二层，可以看到第二层的具体明细，统计值也是一样。

看到这个数据能帮助我们什么?

比如看到URL下面dependency的访问量为566，响应时间为778ms，就需要多想，这样的耗时有么与办法优化，这样的访问量是否合理。主要响应时间和访问量，任何优化就可以开始以及有了根据。

TransactionReport是我们平时做优化最最常用的一张report。 点击最左边Show可以按照时间展示，可以看到按照分钟粒度的精确的展示，1分钟是CAT监控最小的粒度单元。

这里的Transaction的埋点的Type和Name是业务自己定义。

2、EventReport

EventReport整体是和TransactionReport结构几乎一样，可以根据项目，ip进行导航，但是少

了一个响应时间的概念，这里只有访问量。可以按照type展示，查询二级目录的情况，可以看到show，看到分钟级别的趋势图。具体参考transaction，这边我不多讲了。

Event有个比较作用能方便帮助业务做一些业务上统计，举个例子比如看每天用户登录次数或者代码里面有三个分支，可以很简单使用Event API帮你做统计。

3、ProblemReport

之前讲述过每个应用每次的请求都会有CAT的监控的Logview，这样庞大的数量下，ProblemReport等于定义了很多特征，特征包括如下，ProblemReport就是把一堆logview中存在这些特征的给聚合为一个报表，让用户发现问题。

1. 异常（Java里面抛出的异常）
2. URL访问出错以及慢
3. 缓存访问出错以及慢
4. 数据库访出错以及曼
5. 服务访问出错以及慢

上诉报表就是tuangou-web项目一段时间内的产生的各种问题，右边Looooooooog之类，就可以看到原始出错的logview。这个loview可以看到很多明细，比如输入的参数等，还有前后一些逻辑。这样的logview信息是很容易帮助用户发现问题。

上图我点击一个异常的后面的O，就看到这次异常出现的完整的上下文信息。

点击show，可以看到此类错误对应的机器分布情况以及分钟粒度的趋势图，通过这个很容发现是否某台机器故障还是全局问题等。

业务开发他们需要fix里面一些比较严重的问题。比如SQL慢响应很多（比如通过索引或者缓存），服务慢响应（如何优化），调用下游错误（如果推动下游服务端解决），异常（NPE）等等。

4、Heartbeat Report

这里主要CAT客户端定期一分钟向服务端汇报当前运行时候的一些状态，几个比较常用的指标有：

1. 系统内存、swap、load
2. GC信息，GC时间以及数量，Java内存状态

3. 核心的一些线程数，比如http，一些RPC框架线程等

展示方式按照某台机器的维度查询

心跳报表还需要有一定扩展性，这里CAT提供了一个SPI让业务实现并注册上来，这样可以每分钟上报心跳并回调，让后把数据一起传回服务端，并作监控展示。常见的使用场景有：当前机器每个数据库的连接数、当前jvm使用队列的当前队列数量等。

最后一个Report Cross Report

这个报表主要作用了解决在RPC框架里面的一些上下游调用链路的问题，统计粒度支持项目、具体某一IP、具体的服务方法。

比如上图是当前项目xx-web调用了4个服务，调用每个服务的响应时间，访问量以及错误量。

点击某个项目进去可以知道具体调用此服务的具体每台机器的情况如下，记得前端时间微博的同事分享过线上一台服务挂了，导致了1/8的请求有问题，如果CAT能接入进去，应该是很快就能立刻发现是其中一台服务出现故障。

点击ALL或者单台机器可以知道调用具体此服务某个方法耗时情况。

这个报表还有很多其他作用了，比如作为一个服务端，他可以知道当前有多少客户端访问我，可以查询服务端一个方法被多少客户端调用，每个客户端调用的响应以及耗时等等。（这里往往会发现个别比较糟糕应用用一些不太合理的方式调用，比如夜里的Job，大量的batch调用等，这样就会考虑独立部署几台给个别应用调用）。

CAT还有其他的一些报表，这里我就不一一介绍了，有兴趣同学可以参考下CAT的文档。

1. Matrix 调用链路分析，统计调用链路上缓存，数据库，远程服务情况。
2. Storage 存储节点分析，调用某个数据库节点或者缓存节点情况。
3. Cache 缓存命中率分析，针对category的缓存命中率。
4. Dependency 依赖节点分析，分钟级别依赖数据库、缓存等情况。

CAT一些离线分析报表，比如jar版本分析，服务可用性排行等等。

系统设计

如上图CAT在当初设计的基本几个点，在再次基础上我们后面很多方案都是基于此来实现。对应用无影响是必须的，实时性我们要求做到毫秒级，由于是全量监控，所以吞吐量也是需要保证，最后是开销，开销不能因为应用接入监控提高很多的延迟。

最后两个可靠性，应用端是否确定把所有消息传输到服务端，以及服务端是否必须处理所有消息，我们认为是没有必要的，但从实际的结果看来，有4个9的可用性。比如昨天数据，一共处理240亿，丢了50w消息。

下面讲到客户端设计一些要点

设计思路

1. 基于Java TheadLocal实现，跨线程的调用目前不能自动实现（需要显示调用API）。
2. 主线程同步构建消息，构建消息结束之后，放入内存队列。
3. 消息传输是基于Netty实现。

内存开销

1. 由于埋点问题，消息足够大，比如一个message有几万个节点。做消息切割。（不做消息会导致oom）
   比如查询一个商户月交易额，for循环商户的门店，然后查询数据库每天计算累加，结果遇到了肯德基，几万家店铺吧。监控需要做到在用户任何极端不正确使用的场景下，也要保证业务正常。
2. 内部使用队列，限定长度，full即丢弃。（messageQueue也限定长度，不然也会oom）

CPU开销

1. 构建消息足够轻量，纯内存计算。
2. 异步编码以及异步传输，目前消息没有做gzip压缩。

服务端的设计

1. 目前CAT在高峰时候吞吐量单台机器有16w/s，千万网卡高峰时候已经打满（近期会换万兆网卡）。
2. 服务端实时处理主要按照时间的分桶（一个小时一个桶）以及基于内存队列全异步处理。
3. Analyzer是每个小时都是独立线程，线程之间没有任何锁，在高并发高吞吐场景下，尽可能避免锁的竞争。
4. 原始消息的存储，我们这里使用了最简单的文件存储，文件索引以及内存都是自己定义以

及实现。

存储上的一些考虑

1. 文件足够简单，简单的东西就是好东西（老吴说过）。
2. hdfs的作用就是一个集中式的存储，将写好的文件异步传输存到hdfs上，hdfs作为一个集中式的存储。异步的话当hdfs挂了可以做到容错，hdfs维护升级通常是几个小时。 1 原始消息大小大约3500GB，我们希望文件需要是压缩来节省空间。
3. 消息内容查询需要支持到Key-Value，Key就是消息ID，Value就是消息内容。
4. 所以存储需要做到顺序写，压缩后随机读。

下图是整个文件存储的设计图

在消息ID设计上，是${domain}+${ip}+${timestamp}+${index}，这作为唯一的KEY，所以CAT不支持同一个IP部署两个相同的应用。同一个IP部署多个不同的应用是没有问题的。所有的文件都加上cat的consumer的ip作为后缀，保证文件上传到hdfs上不会有文件的冲突。

压缩采用的是分块压缩，数据块的大小在64K以内，所以记录每个消息对应的所在块以及消息在所在块的偏移量。当需要找到某个消息，先随机访问到某一个块，将这个块全部读取出来，然后根据块内偏移地址找到对应的Logview，因为消息可能是在多个文件中，最后检查下读取到的logview的id等于查询时候的id，即找到消息。

总结

这里说下点评实际的一些经验。
很快到了今天总结的时候，这部分我会讲讲监控在点评一路的情况，有些问题的解决，如果还有其他的疑问，后续大家可以提问

第一个重点是埋点工作

1. Transaction埋点，主要记录一些跨项目，跨模块一些调用。最主要有三个，远程服务，数据库以及缓存。点评是在中间件上做的埋点，比如PRC框架，数据访问层框架，缓存框架，然后在业务线全部升级框架。
2. Event埋点，主要记录事件以及异常。最常见的场景就是当埋Transaction时候，需要event作为补充，比如记录当时访问参数等。代码一些特殊诡异路径的分析，还有异常信息记录。
3. Metric埋点，主要用于记录了一些实际的业务指标，用于运维监控。Heartbeat不需要业务埋点。
4. 当业务需要监控自己一段代码访问行为，他可以自己使用API，常见有访问第三方服务等等，一些比较复杂的逻辑算法等。当业务用好之后发现能解决问题之后，基本用了就停不

下来，有的team第三方购买的系统，都会加入cat埋点，因为遇到过第三方系统性能问题没办法分析。

5. 监控系统需要做成开放式系统，每个核心的report最好能提供api，让其他程序能够访问，每个业务的想看的一些指标展示各不相同，监控是无法完成业务定制化report需求，所以需要每个报表都能api，把自己做成一个数据平台。目前有不少项目是自己写程序爬CAT数据，作为自己整个项目的周报，技术指标。还有一些用来推动业务进行技术优化的业务线横向报表原始数据都来自CAT。
6. CAT不仅仅在生产环境做了部署，在线下以及性能环境也做了部署，qa做压力测试、功能测试都会去看cat上面的数据，相比于以前测试，可以大量节省他们的时间，跑一次自动化case回归，看看CAT上面的报错，响应时间，访问量，大概就知道程序是否存在问题。
7. 在实际推广过程中，培训很重要，并不是所有人都能很清楚知道和接受CAT怎么用，相对资深同事更加容易接受。很多其他的公司部署了CAT，就是因为有好几个同事都是从点评离职过去，部署起来的。

整个CAT项目从2012初到现在，到现在大约3年多时间，整个过程持续2-3个人，几点感触和大家分享

1. 先做小做精，再做大做全
2. 持续集成，持续发布，不断监控
3. 单机开发和调试
4. Everything Fails
5. 关注客户，快速响应
6. 站在巨人的肩膀上

在今天演讲的最后，请允许我打一次广告，CAT的地址 https://github.com/dianping/cat 有兴趣可以按照步骤搭建下，搭建过程我们写了一键工具。 CAT很多其他的文档都写搭建的系统首页上，只要本机有个mysql即可，CAT就能单机部署并运行起来。

Q1：监控系统数据采集的频率如何把控?

CAT是全量数据采集，心跳数据是1分钟一次。如果服务器或者硬件监控，一般几秒一次。如果是特别重要的，可以一秒一次。

Q2：监控项一般都有哪些?

比较通用的监控项，也就是核心框架里面监控项目，比如远程访问，数据库访问，缓存访问的响应时间，访问量等。

心跳里面的监控项有

1. 系统内存、swap、load
2. GC信息，GC时间以及数量，Java内存状态
3. 核心的一些线程数，比如http，一些RPC框架线程等

Q3：报表系统现在有很多开源的东西可以做，尤老师有没推荐的?

建议报表系统还是自己做比较合适，报表这类需求定制太多了，比如按照部门、产品线等统计。

Q4: 跟其他开源监控方案对比如何?

开源的监控系统，比如zabbix，nagios，cacit算是很成熟的一套监控系统，他们能通过脚本或者snmp协议等收集很多服务端的性能数据，并配置很多监控规则来发现服务器等一些问题。点评这些也在用，主要是zabbix，他和CAT互相补充。

之前小米开源的系统应该也是基于指标的画图以及告警，和CAT应该是两类不同的系统。

Q5: 能不能举例说明一下服务监控和App监控的具体做法,有没有最佳实践?

服务端监控，就应该类似于我上面讲的CAT在服务端监控一些做法，不仅仅包括问题发现，还包括了性能调优，路径分析等等，这样才能把帮助业务分析并找到问题。

App监控，由于时间问题，今天没有讲到，这里说几个点吧。App监控点评做了三个部分：

1. 返回码系统（多维度下，API、城市、运营商、网络、APP版本等）
2. 实时Crash日志（版本、平台、模块等维度）
3. 测速系统（打开一个APP某个页面的分段速度测试，一个页面可能包括广告，导航、图片等，每个阶段的速度测试）

Q6: 有没有CAT跟Dubbo集成的案例?

Dubbo是阿里的一套PRC开源框架，目前我们没有集成的case，CAT和点评PRC组件集成得很好，坚信应该不是问题。

Q7: Cat是否适合创业公司用？希望不要太重。

1. 创业公司可以根据自己实际需要，以及对于CAT的理解做出的一个评估。我理解的监控，是承接了公司整个开发流程上闭环的一个重要角色。监控能帮助应用发现问题，分析问题，找到问题，然后在快速进行迭代。如果要我说，我觉得应该是适合的。
2. 重和不重，这点我不太好说，埋点的深度决定了监控的高度，可以轻量级使用，也可以重度使用。服务端对于创业公司来说应该很轻量，只需要MySQL即可，只要本地磁盘足够OK，HDFS都可以不需要。

Q8：实施&运维的成本多大?

个人觉得运维成本不大，主要是使用以及依赖组件很少，MySQL，极端情况HDFS都可以不需要。

Q9：数据传输用Netty而不是基于现有消息队列，是为什么?

基于性能考虑。Netty本身是非常成熟的开源网络组件，event机制、全异步、高性能，还有简单易用，非常适合于框架和基础设施开发。CAT用Netty做长连接通讯，吞吐量非常高，健康检测、故障转移和故障恢复能力都非常强。消息队列根本不适合这么底层的数据传输，它的overhead太大，很多feature CAT并不需要。

Q10：有没有实现内部服务调用链分析，如果有，能不能讲一下大概思路?

用CAT可以将在多台机器或多个进程的消息串起来，组成调用链。基本的思路是，在常用的场景中，客户端产生一个新的消息ID，连同自己的消息ID在调用时作为参数传递到服务端，然后在服务端的消息使用传过来的消息ID，来存储，并用客户端的消息ID作为向上的链接。这样客户端和服务器端就可以有双向的链接了。但是这需要在相关的框架中埋点。

Q11：如果网络断了，数据在本地能备份吗?

CAT客户端有一个queue，queue是固定大小了，CAT一台客户端会连向几个服务端，当一个服务端出问题，数据不会丢失。当所有服务端都挂掉，消息会存入queue，当queue满了，就丢弃了，没有做数据存储本地等工作。

Q12：传输数据是什么格式？有压缩吗?

1. 数据传输格式为普通文本，自己实现的序列化和反序列化，目前没有做压缩，压缩需要考虑到客户端和服务端的cpu压力。
2. 服务端存储数据是有压缩的，压缩比例大概为8:1

Q13：部署CAT，对监控的应用和服务器大概有多少影响？ CAT服务挂掉，如果避免不影响业务?

1. 对于应用的服务基本影响很小，在做极端的压力测试下，加上CAT以及不加入CAT，峰值qps影响在10%以内，均值基本不变。
2. CAT客户端和服务端的通信都是异步的，当服务全部宕机，客户端丢失消息即可。

Q14: 服务的调用链，怎么做到在中间件层面判断依赖关系？如果不是在中间件，分布式里面，怎么做设计的上下游依赖?

上下游调用，主要用到全局统一项目名，这是点评一套架构上的规范，可以理解为环境信息。RPC能在当前的运行环境下知道自己当前的项目名以及调用者的项目名。

Q15：是否有多语言client支持?

支持JAVA和 .Net。

Q16：CAT自身的健康状态监测?

当一个客户端不 work，client 会自动连接上 next server，所以有一个管理连接的线程，保持一个活动有效的连接，如果 server 全部挂了，消息就丢弃了。

Q17：如何检测到丢失的消息数?

所有的消息都是基于消息队列处理，如果队列满了，会有记录。

Q18：CAT有做字节码插桩么?

没有。

Q19: 对JVM的监控，是通过jstat脚本来收集数据的吗？数据是客户端上报，还是服务端拉取，数据格式也是logview?

不是通过脚本，具体的收集可以参考源码 StatusInfoCollector，主要用到 MemoryMXBean，MemoryPoolMXBean等。

数据上报的格式也是logview，是客户端主动一分钟上报一次。

Q20: 客户端上报数据的话，服务端可以不用高可用吧?

服务端尽量做到高可用吧。

Q21：手机客户端的监控是怎么做的，直接发到CAT还是业务系统转发呢?

手机监控其实是另外一套系统，最后展示层在CAT。

Q20：我们在用Nagios做系统和服务的监控，但对服务调用链的监控，基本上是空白。不知能不能有机会听一下中间件层面的服务调用是怎么监控的。

服务端会分析上报logview，埋点信息会有客户端和服务端的项目名信息，后端可以用此数据进行实时分析。

Q21: 这种数据用TCP长连接汇报的么？为什么没考虑使用UDP?是为了可靠性么？还是其他考量?

UDP在线下测试很不稳定，经常丢弃什么，TCP的比UDP多的开销其实对业务影响很小。

CAT系统最初的设计者吴其敏补充说： 如果系统的消息丢失率达到一定的比例，那么它的的可信度就会降低，所以保证足够多的消息能够被好好处理是很重要的。CAT设计成不可靠系统，

但实际上我们希望系统足够可靠，只是不愿为承诺可靠而付出高额代价。 以前研究过Dapper，觉得它的机制与CAT相差很远，监控的全面性不够。当然如果到了Google的量级，情况就很不一样了。

TCP是有连接的，有拥塞控制；UDP无连接，一般情况下内网成功率还是挺高的，极端情况下将非常糟糕。

感谢吕涛的记录与整理，臧秀涛的审核，以及Carson的公众号文章编辑，其他多位编辑组志愿者对本文亦有贡献。更多关于架构方面的内容，请长按下面图片，关注“高可用架构”公众号，获取通往架构师的宝贵经验。