58各BG技术沉淀分享交流平台

Jaguar
技术实践分享会

【出品人】技术委员会 联合 HR·神奇研修院

58集团技术委员会 人力资源部 神奇研修院

《如何统计响应时间以及告警实践》

讲师：亢伟楠 – 信息安全部

58课程重要性

为什么要学这门课程

1. 集团内采用微服务体系，服务之间**分布式调用多**，**链路复杂**，伴随网络波动、gc、服务上下线、宿主机故障等情况，常会出现耗时升高问题
2. 耗时统计方式多样、服务提供方和调用方对于服务质量**无统一标准**，出现冲突很常见
3. 告警阈值**难设置**，阈值低了频繁误告警，阈值高了会错过告警；管理多个服务时，阈值设定成本巨大
4. 目前互联网整体流量瓶颈，更需要维护好现有的系统，提供更好的服务和用户体验

学习这门课程的收获

1. 掌握如何正确统计服务响应耗时，如何制定耗时标准
2. 掌握如何评估告警方案的优良，掌握如何设计出一个表现优秀的告警方案

目录

耗时统计面临的问题

耗时统计-案例

你是刚入职的小a，有一天领导找到你，告诉你，小a啊，有其他人投诉咱们这个服务超时多，你看看怎么处理

哎，看完发现服务的平均耗时是100ms，确实有点高，而业务能接受的时间是60ms

然后把平均耗时优化到了50ms，告诉领导，性能提升了一倍了，也满足业务需求了

耗时统计-案例

第二天，业务还是来反馈，说有超时

你一看监控，平均还是50ms啊，这群业务是不是有毛病？于是你怒向胆边生，恶从心中来

然而别人还是发来了超时图

耗时统计-案例

```
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.176.75.33],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.177.117.224],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.144.82.204],Receive data timeout or error!timeout:50ms,queue length:2
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.176.74.175],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.145.57.232],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.177.107.248],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.176.75.20],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.176.75.4],Receive data timeout or error!timeout:50ms,queue length:2
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.145.87.173],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.176.74.220],Receive data timeout or error!timeout:50ms,queue length:1
com.bj58.spat.scf.protocol.exception.TimeoutException: ServiceName:[listcheck],ServiceIP:[10.176.98.105],Receive data timeout or error!timeout:50ms,queue length:2
```

耗时统计-平均值

为什么用平均值代表响应时间会出问题?

avg = (min + max) / count

avg = 50ms , 代表可能:

1.全部都是50ms(显然不可能)

2.有超过50ms，有小于50ms，但是偏离多大，量有多少，无法表达。

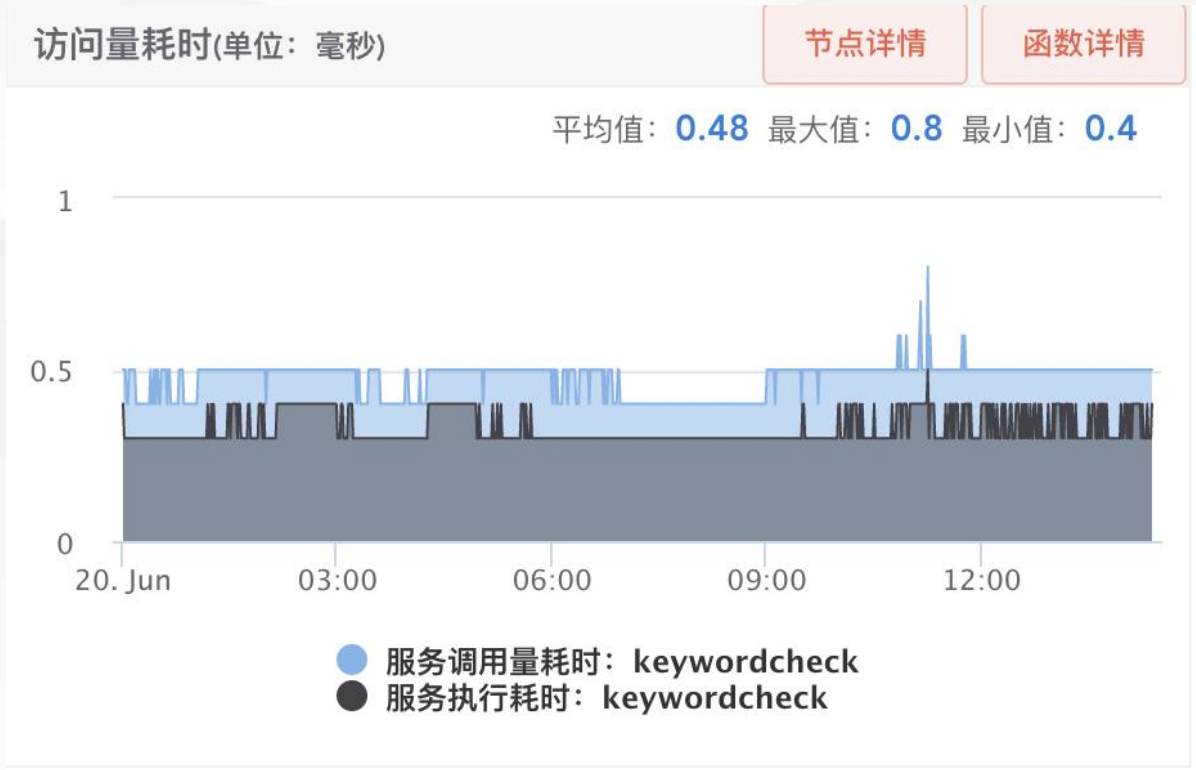
即使平均耗时优化到0.1ms , 能保证没有超时吗?

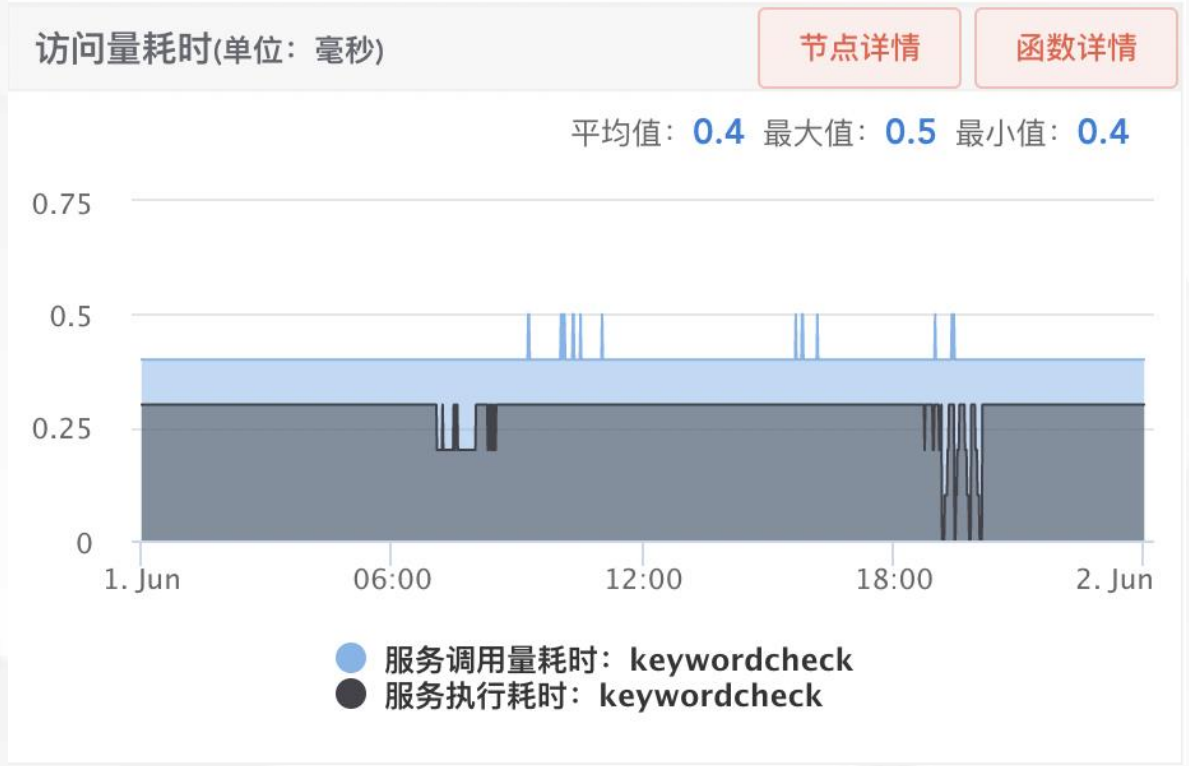
搭配 方差/标准差，能表达波动/偏离度，但是无法表达偏离量.
例如:avg 50ms , std 20ms，依然无法表达出超时量。

耗时统计-平均值

已知调用量和平均耗时，求超时量???

耗时统计-平均值





已知平均值、最大值，求哪一个性能更好？？？

耗时统计-tp

考虑另外一种流行的方式：tp（百分位数）

该方式统计所有的请求，并计算出某个百分位的值。

例如：1～100，step ＝ 1 的数据中，排序后，tp90 代表处在90%位置的数据，此时为90，tp99就是99,tp 50 就是我们熟知的中位数

看起来没问题了，tp采用排序+百分比的方式，既能表达偏离率，又能表达偏离量

我们已经找到完美的解决方案了。

耗时统计-tp

统计学上通常使用四分位数方式来表达统计结果：
tp 50 ~ 代表50%的数据都在该指标范围内
tp 75 ~ 代表75%的数据都在该指标范围内
tp 90 ~ 代表90%的数据都在该指标范围内
tp 99 ~ 代表99%的数据都在该指标范围内

举一个栗子：某服务 tp 50 1ms , tp75 3ms ,tp90 5ms, tp99 15ms

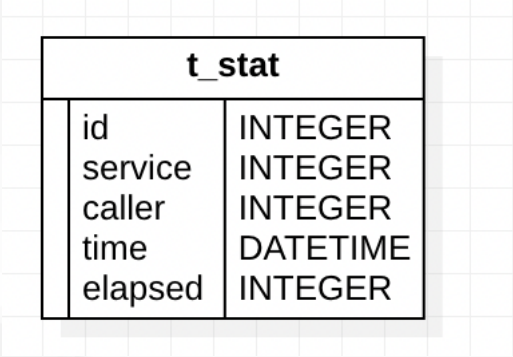
代表:
50%的数据都在1ms内(50%的数据超过了1ms)
75%的数据都在3ms内(25%的数据超过了3ms)
90%的数据都在5ms内(10%的数据超过了5ms)
99%的数据都在15ms内(1%的数据超过了15ms)

耗时统计-tp

设计 基于 tp的耗时统计服务

该服务作为基础服务，收集其他所有服务调用数据，因计算tp时，需要对所有数据排序，需全量保存数据。

数据库表结构



耗时统计-tp

统计今天的tp数据，伪代码：

```
list sort_data = select caller,time,elapsed from t_stat where service = a and  time
between today_start_time and todat_end_time group by caller order by elapsed desc

tp99 = pick_p99(data);
tp90 = pick_p90(data);
tp75 = pick_p75(data);
tp50 = pick_p50(data);
```

t_stat	
id	INTEGER
service	INTEGER
caller	INTEGER
time	DATETIME
elapsed	INTEGER

问题：

1. 存储压力：需要存储全量请求，o(n).无压缩空间，因为需要保留时间
2. 计算压力:

    每次都需要实时计算，且计算量级随调用量增长,这是一个o(n)的算法.

    跨长时间范围的查询，压力巨大。 想象一下，imc服务,每天110亿调用量，统计一个季度的耗时情况（假设我们有这样的okr：imc服务在q1 tp99 达到5ms）,量级为110亿*90 = 99000亿,当前的数据库基本歇菜

    结果难复用，需要计算的次数多;耗时长，基本无法满足在线使用;

解决方案

耗时统计-解决方案

基于okr 来思考:xxx服务在q1 tp99 达到5ms，两个核心数值：tp99,5ms

前一种方式，重点在计算出tp99上；换一个思路，计算出有多少百分比的数据在5ms以内.

此时，我们只需要统计 小于5ms的请求个数，总个数 即可。

表达式为 result = count_5ms / count_total 。

考虑更复杂更精细的okr: 在q1 tp90 达到1ms， tp99 达到5ms，tp99.9 达到10ms

依然很简单， result_90 = count_1ms / count_total
result_99 = count_5ms / count_total
result_99.9 = count_10ms / count_total

耗时统计-新方案

数据库设计

t_stat_new	
id	INTEGER
service	INTEGER
caller	INTEGER
count_1ms	INTEGER
count_5ms	INTEGER
count_10ms	INTEGER
count_total	INTEGER
time	DATETIME

伪代码：

```
select
sum(count_1ms),sum(count_5ms),sum(co
unt_10ms),sum(count_total) from
t_stat_new where time between start to
end and service = xxx

result_90 = sum(count_1ms) /
sum(count_total)
...
```

优点：

1. 数据量级从o(n)变成o(1)常量级。在一个时间单位内，无论多大的流量，都可以用一个int/long列来存储，存储压力和计算压力得到有效解放。
2. 基于计算压力的指数级下降，即席查询和长时间范围查询得以实现,真正可用了

缺点：

1. 只能计算特定区间的耗时数据，需要根据业务特性设定

耗时统计-告警

回顾一下我们现在的情况，我们得到了正确设定耗时的方法以及正确的目标。

xxx 服务
服务质量:一个月的90%的请求在1ms以内；99%的请求在5ms以内；99.9的请求在10ms以内；

基于上述情况，我们能够实现对服务耗时的监控，但是我们不可能全天候7*24看着监控图表来发现有没有问题，所以我们需要一个告警方案来解决人力，该系统会在耗时不达标时，会主动通知我们，这样我们就能及时介入排查问题。

耗时统计-告警

回顾一下我们现在的情况，我们得到了正确设定耗时的方法以及正确的目标。

xxx 服务
服务质量:一个月的90%的请求在1ms以内；99%的请求在5ms以内；99.9的请求在10ms以内；

基于上述情况，我们能够实现对服务耗时的监控，但是我们不可能全天候7*24看着监控图表来发现有没有问题，所以我们需要一个告警方案来解决盯监控的人力问题，该系统会在耗时不达标时，会主动通知我们，这样我们就能及时介入排查问题。

对于告警方案，我们基于以下四个指标来比较：
1. 准确率：告警中是否存在故障的比例，如果每个告警都是事故，则准确率为100%
2. 召回率：告警是否漏过故障的比例，如果每个事故都导致警报，则召回率为100%。
3. 触发时间：从发生故障到发出告警的时间
4. 重置时间：从故障恢复到停止告警的时间，较长的重置时间可能导致混乱或问题被忽略。

耗时统计-告警

方案1:
过去1分钟的指标不达标时，告警

优点：最简单的方案，也是被很多系统采取的方案；实现成本低，符合直觉;
召回率高，触发时间短，重置时间短

缺点:
准确率低;流量波动、网络波动，gc导致的stw等都会频繁导致告警,让人陷入告警风暴并对告警感到疲倦

耗时统计-告警

方案2:
基于方案1延长时间
过去60分钟的指标不达标时，告警

优点:
准确率高

缺点:
召回率低；会丢失持续了9分钟的异常情况，假设出现 9m(bad)-1m(good)-9m(bad)-1m(good),尽管线上90%的情况都不达标，该方式依然不会告警

重置时间长；即使线上情况恢复正常，也会统计到过去n分钟的情况，导致持续告警。

耗时统计-告警

方案3:
基于方案1添加持续时间,大多数告警系统支持这种方式
过去1分钟的指标不达标并持续10分钟时，告警

优点:
准确率相比方案2更高

缺点:
召回率低；每10分钟的前5分钟故障，永远会告警

耗时统计-燃烧率

引入一个新概念：错误预算

我们的目标为 90%小于1ms，此时错误预算等于 1-90%=10%；

这意味着，当线上一直保持10%不达标率时，到月底，我们的刚好达标；

再引入一个新概念，燃烧率。将刚好达标的情况定义为燃烧率为1，即错误预算刚好消耗完的速率。

此时很容易计算出不同燃烧率下，错误预算消耗完的时间。

燃烧率 --	对应时间
1	30 d
2	15 d
10	3 d

耗时统计-燃烧率

方案4
此时针对错误预算的消耗情况做告警，即得得知在一个完整时间周期内的质量情况。

一个月 = 30 * 24 = 720 ;推测出 燃烧率为1时，一个小时对于错误预算的消耗为 1/720.

我们设定一小时消耗错误预算的5%应该告警,此时的燃烧率为 (5/100) / (1/720) = 36 ,即燃烧率为36时告警。

优点
准确率高,按照错误预算的5%告警，能过滤掉很多抖动和微小事件
时间窗口短，便于计算
触发时间短

缺点
低于36的燃烧率都无论持续多久都不会告警,但实际上20.5天就消耗完所有错误预算了
重置时间：58分钟，太久了

耗时统计-燃烧率

方案5
基于方案4，如果我们在三天内消耗了10%的错误预算，方案4将无法检测，即使这种会导致我们的错误预算一直在边缘被消耗；同时一小时消耗5%的错误预算还是比较高。

所以这里改进成多个时间窗口来检测

错误预算消耗值	时间窗口	燃烧率	告警
2%	1h	14.4	高优先级
5%	5h	6	高优先级
10%	3d	1	低优先级

优点
多种阈值+时间窗口，适应能力强：错误率高，很快告警，错误率低，最终会告警
召回率高，因为有3d的时间窗口，能避免长时间范围内的低错误率事故漏过

缺点
更多需要设定的阈值和窗口
3d的窗口，重置时间非常长

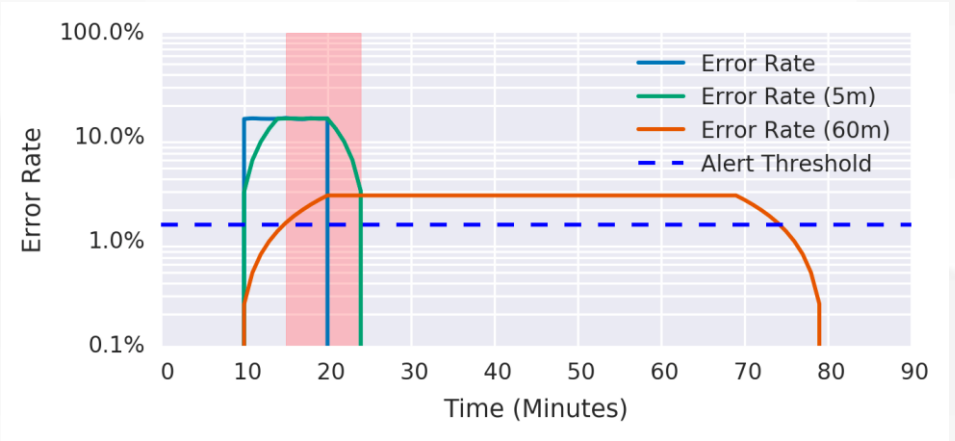
耗时统计-燃烧率

方案6
基于方案5，明显的缺点是重置时间较长，所以我们添加一个短窗口，目的是检测当前的燃烧率是否已恢复，当前的消耗是否是因为长窗口所造成的，如果已经恢复，那么就可以停止告警了。

长窗口	短窗口	消耗错误预算	燃烧率	告警
1h	5m	2%	14.4	高优先级
6h	30m	5%	6	高优先级
24h	6h	10%	1	低优先级

优点
高精度，灵活
召回率高

缺点
更多需要设定的阈值和窗口



实践成果

实践效果

验证码项目
质量要求为：99.99%的请求在50ms以内;实际质量为 99.99%

达标请求 2,719,014,336 / 总请求 2,719,197,918 = 99.9932%

方案1 告警数 1137 @ 20d
方案6 告警数 0 @ 20d

总结

总结

课程总结

1. 耗时统计是一个严肃的场景，如果统计方式不对，那么得出的结论也会失真。

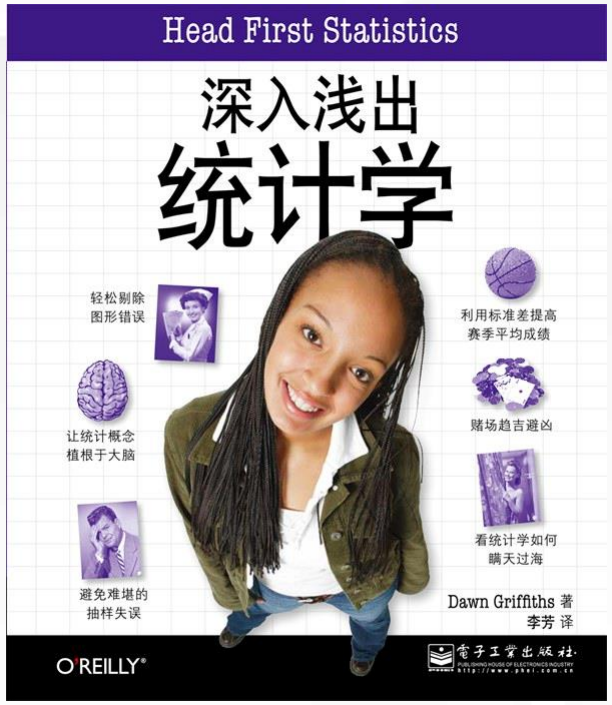
平均数：无法表达数据分布情况，误导对线上耗时的认知，tp实现中有难度；

百分比统计：数据降维，将流量级降低到常量级

燃烧率告警：量化用户体验，使用燃烧率概念来衡量消耗速度，使用错误预算来衡量整体用户体验损害。多角度衡量告警方式，不局限于单一角度（准确率）

2. 服务相互依赖时，需要服务提供方和调用方确认好 *服务性能标准* ,否则就成了公说公有理，婆说婆有理。

58拓展阅读



提供最符合直觉的理解方式，有趣又自然地理解统计理论.

Jaguar

感谢观看

请扫码对本次课程进行评估

58集团技术委员会 人力资源部 神奇研修院