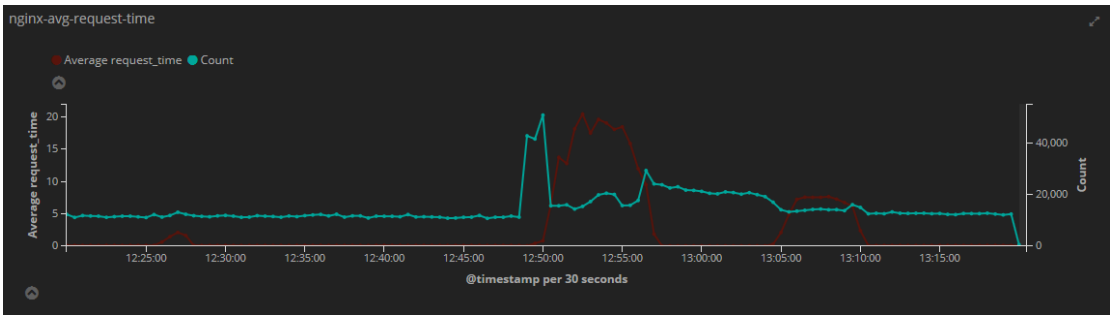
Spring Cloud 网关访问下游服务超时故障分析

Email: geekidea@gmail.com

2019/01

0. 问题描述

01.19 中午接到运维及业务开发同学反馈 App 刷 feed 白页，同时部分节点的某具体业务健康检查失败报警，并伴随有 spring cloud 网关组件健康检查失败报警。nginx 监控大盘故障时段如下：



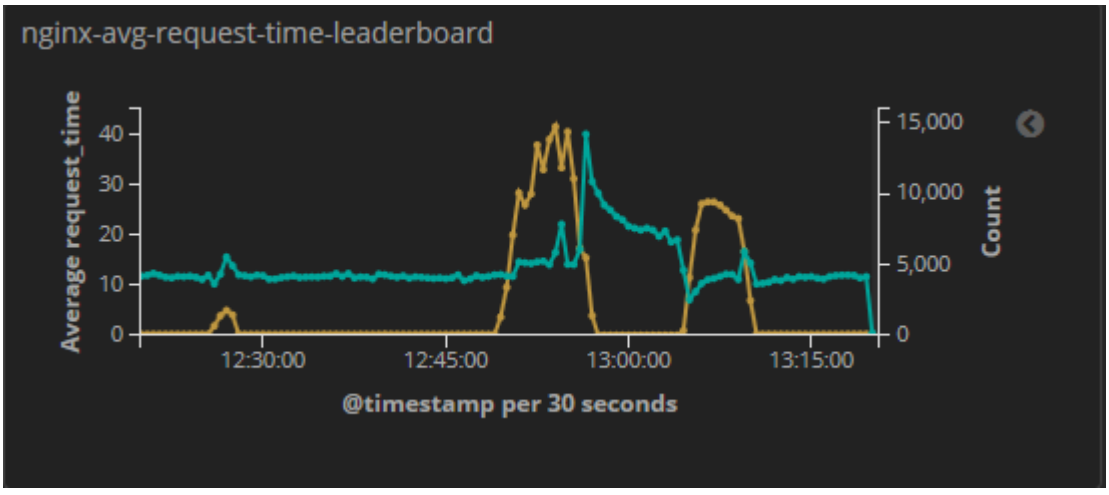
上图可见，nginx 响应时段陡增有三个时段：

故障时段 A　12:25-12:27

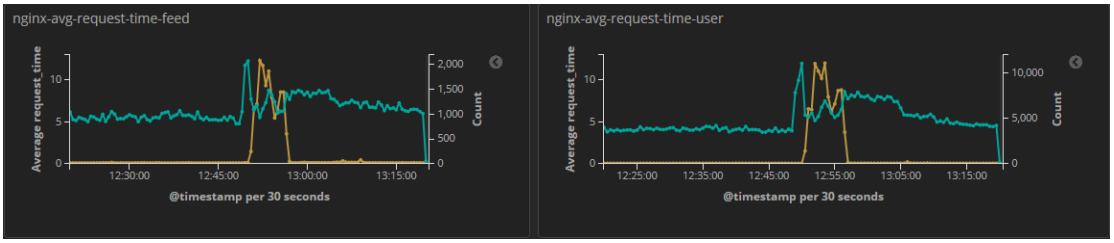
故障时段 B　12:50-12:57

故障时段 C　13:05-13:10

故障报警业务在同时段 RT 图如下：



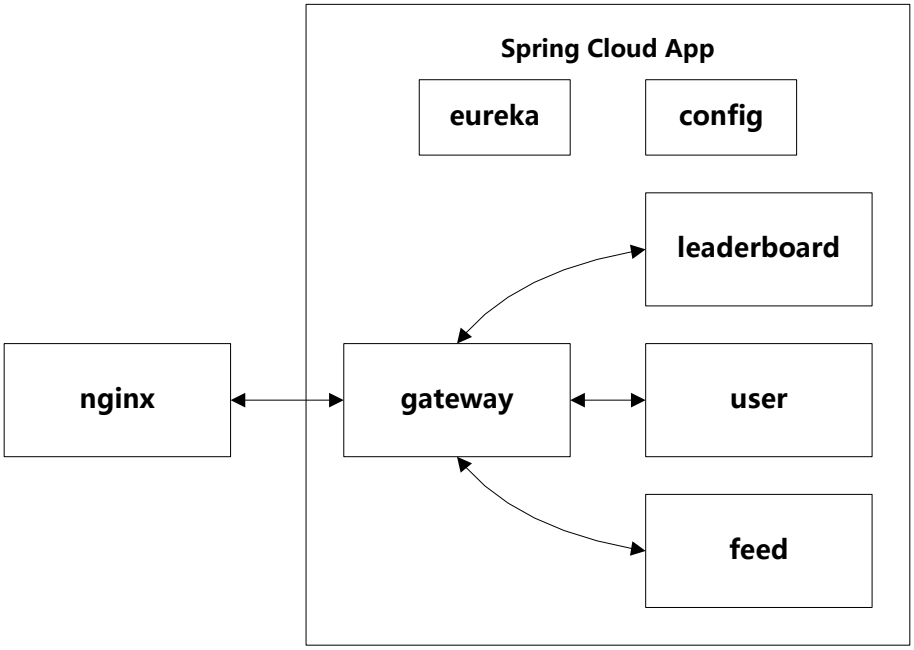
该报警业务（leaderboard）的 RT 变化曲线与 nginx 入口 RT 变化曲线基本吻合；其他相关核心业务在同时段 RT 图如下：



上图可见，feed 和 user 业务 RT 陡增区间基本出现在故障时段 B （12:50-12:57），该区间的业务响应时间在 10 秒左右，另外其他业务在该时段也出现了 RT 陡增的情况。

1. 问题分析

故障所涉及主要的服务组件架构拓扑图如下：



其中每个服务组件实际采用多 ECS 节点（或多 docker 混部）部署模式，核心业务采用 Spring Cloud 架构：来自 App 的入口流量由 nginx 集群统一承载，然后转发给 gateway 组件，该组件主要基于 zuul 二次开发进行用户鉴权，对指定 URI 进行路由转发；转发至后端的具体业务服务，如：leaderboard，user，feed 服务，其中后端的具体业务之间会有互相调用的情况。

开发同学反馈 leaderboard 业务于前一天晚上在部分机器灰度上线新版后，第二天（01.19）中午其中某台 ECS 上该业务 CPU 接近 200%，于是对该业务做配置在线回退，但 CPU 仍旧无法下降；然后对该进程手动重启，CPU 依旧无法下降，最终将业务回滚至之前版本，线上服务逐渐随之恢复。

由**故障时段 B** 的现象，可以初步判断为，当某个后端业务 hang 时，大量的客户端请求（连接）堆积至应用网关（gateway）组件，触发了 gateway 中某个瓶颈，导致其他业务也随之阻塞，因此出现了故障时段 B 的所有请求的慢响应。

由于 HTTP 请求一般基于 TCP，因此进一步推断该时段的 TCP 连接数到达了 gateway 组件设置的阈值，通过系统监控（Falcon）在 gateway 虚机上看到故障时段 B 时 TCP 建联（ESTABLISHED）的数量始终维持在 **2510** 左右，并且该 gateway 监听端口有大量处于的 CLOSE-WAIT 状态的 TCP 连接，TCP 建联的连接数监控截图如下：

图中红色箭头标注的是故障时段 B 的 TCP 连接状态；在故障时段 A 和 C 内，TCP 连接数并未达到该值，因此其他业务（feed，user 等）在该故障时段（A 和 C）并未受影响。

CLOSE-WAIT 分析：由 TCP/IP 基本原理可知 CLOSE-WAIT 是对端主动发送 FIN 报文后，gateway 端发送回应（ACK）报文后并未发送 FIN 报文，因此 gateway 端进入 **CLOSE-WAIT** 状态，此种情况一般出现在，当服务端出现了慢响应，客户端超时已到主动关闭了当前 HTTP（TCP）连接，而服务端一直处于阻塞状态未及时发送本端的 FIN 报文（通过 close 或 shutdown 等系统调用发送 FIN 报文）。对于 gateway 来说，该故障时段内 gateway 一直处于等待下游业务（leaderboard）的回应，造成了大量来自客户端连接处于 CLOSE-WAIT 状态。

2510 ESTABLISHED 连接数分析：由于 gateway 组件的 TCP listen backlog 是 **512**（`ss -an | grep tcp | grep LISTEN`，第四列），当 gateway 出现阻塞时，TCP 的连接也会占满该 backlog，所以通过监控看到的 ESTABLISHED 的连接数也包含了一个 backlog 的数量，减去该 backlog 后得到正常程序保持的连接数，约为 **2000**。对于 spring cloud 应用，TCP listen backlog 一般通过 tomcat 的 **acceptCount**（默认 100）配置值指定，我们的 spring cloud 应用统一设置该值为 512，对于 gateway 应用，与客户端正常保持的全局连接数依赖 **server.tomcat.max-connections** 配置，该值为 **2000**。由此可知该故障是由于

gateway 的下游服务（某个节点）出现了长时间的慢响应，导致 gateway 保持的客户端连接数触发了 **server.tomcat.max-connections** 瓶颈。

编写 tcp 客户端测试程序（test_tcpsock）在测试环境下可以模拟复现该场景：

```
test_tcpsock 3000 [gw_ip] [gw_port]
```

上述命令尝试与 gateway 建立 3000 个 TCP 连接，当 TCP 连接建立至 2510 左右时候，test_tcpsock 端出现连接超时，同时 gateway 的正常的业务请求（feed，user 等）出现超时。

2. 解决方案

针对以上原因分析，可以采用如下方案：

1. gateway 组件的全局连接总数(server.tomcat.max-connections)适当调大，增加 gateway 吞吐及容错，降低因个别下游节点故障阻塞导致的整体业务慢响应。

2. gateway 向后端业务请求超时调小。

3.去掉 gateway swaggerui 包，减少频发加载解析 jar 包中的 class 导致的慢响应（关于该问题的分析可参考文档"**基于 Linux 系统工具对 JavaSpring 架构业务的性能优化**"）。

4.业务容器化：在容器内配置健康检查会更快检测到失效并自动重启，减少故障时间。

最终采用的处理方案为：将 gateway 组件的全局连接总数调整为 10000，gateway jar 包去掉 swaggerui 类，将 gateway 及下游服务（feed/user/leaderboard 等）容器化，并配置详细的容器健康检查策略。

3. 后续思考

对于我们的 SpringCloud 应用，偶尔会出现在重载业务配置时出现无法正常加载的情况，通过 strace 跟踪初步诊断 java 主线程在循环等待获取某个锁（阻塞至 futex 系统调用），同时伴随出现没有向 eureka 反注册再重新注册的情况，需要进一步完善 SpringCloud 基础框架。

4. 参考资料

tomcat 官方配置参数说明

《TCP/IP 详解》卷一