**Vodpush内存置换策略模拟器设计说明书**

**文档历史**

|  |  |  |  |
| --- | --- | --- | --- |
| **日期** | **作者** | **描述** | **版本** |
| 2017-12-15 | 冒雨楠 | Draft for review | 0.1 |
| 2017-12-15 | 冒雨楠 | 修改了模拟器需要动态配置的参数 | 0.2 |
| 2017-12-15 | 冒雨楠 | 更新文档表述不清晰的地方 | 0.3 |
| 2017-12-18 | 冒雨楠 | 1. 增加cdn simulator的说明  2. 增加风险项：vodpush是否需要返回数据（需与开发澄清） | 0.4 |
| 2017-12-20 | 冒雨楠 | 增加详细设计 | 0.5 |
| 2017-12-25 | 冒雨楠 | 增加附录：vodpush输出log的内容 sdk simulator  输出log的内容 | 0.6 |
| 2017-12-27 | 冒雨楠 | 增加附录：如何搭建独立的测试环境(待补充) | 0.7 |
| 2018-1-3 | 冒雨楠 | 增加测试场景 | 0.8 |
| 2018-1-4 | 冒雨楠 | 改变模拟器实现 | 1.0 |

目录

[1. 需求分析 3](#_Toc501035670)

[1.1 背景 3](#_Toc501035671)

[1.2 需求 3](#_Toc501035672)

[1.3 需求分析 3](#_Toc501035673)

[2. 概要设计 4](#_Toc501035674)

[3. 详细设计 8](#_Toc501035675)

[4. 设计实现 8](#_Toc501035676)

6. 附录

6.1 log格式

6.2环境搭建

6.3测试场景设计

# 需求分析

## 背景

目前vod push内存置换策略未达到预期效果，开发需要对vodpush内存置换策略进行持续优化。由于线上环境复杂且搭建vodpush内存置换效率测试环境复杂等问题，导致测试复杂且不准确量化优化的效果，所以需要vod push内存置换策略模拟器。

模拟器主要的功能是：控制变量（例如：vod push中硬盘存储的文件数量，大小），量化vod push的各项指标（例如：vod push的带宽利用率）；简化环境搭建，快速测试开发感兴趣的各种场景（通过接口传入参数，快速搭建需要场景）；

## 需求

* 模拟器动态配置参数：



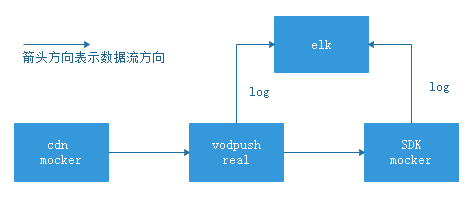
* 模拟器模拟sdk向vod push请求数据，统计vod push对sdk请求数据请求的响应状况（时间点和状态码）并且能在elk中直观展示；
* 模拟器根据vod push的log在elk中展示vod push的状况（带宽的利用率，内存的利用率）；

## 1.3需求分析

* 模拟器可以动态配置参数：
* 模拟器模拟SDK向vod push请求数据，当收到vod push的返回时，记录接受到返回的时间点和返回的状态码，将记录打印到log中用于在elk中展示，当收到足够数量（file size / ppc / piece\_per\_request\_get）的状态码为200的返回时表示缓存完成
* 模拟器根据vod push的log能在elk中展示vod push的内存利用率和带宽利用率
* cdn simulator由开发提供主要用于控制vod push中的文件（或者直接将文件注入vod push的工具）

# 概要设计

模拟器设计结构图如下：



需要环境如下：

1.cdn simulator（用于直接将文件写入vod push所在机器的磁盘上，用于控制vod push 上保存的文件）

2.真实的vod push

3.vod push运行log

4.可用的elk （复用策略模拟器环境的elk即可）

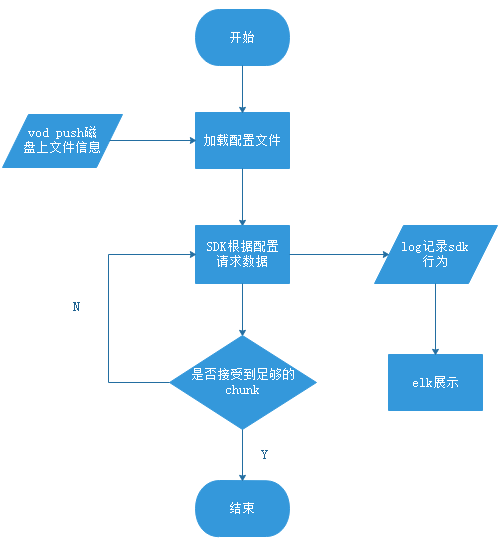
需要开发sdk simulator模块有以下功能：

1. sdk simulator 可以模拟sdk请求数据
2. sdk simulator可以收集每个sdk的log，提供给elk
3. sdk simulator可以动态配置参数（例如：sdk的总数，每个文件请求的sdk数量等）

需要配置的模块：

1. 读取vod push 的log在elk中展示（filebeat 🡪 logstash🡪elasticserach）
2. 读取sdk的log在elk中展示（filebeat 🡪 logstash🡪elasticserach）

3.详细设计 SDK simulator模块详细设计如下图所示：



1.使用者根据测试场景修改配置文件内容（使用/tools/detect\_files.py 探测vod\_push磁盘上的文件，sdk只能请求vod push磁盘上存在的文件）

2.启动main.py时会主动加载配置文件

3.sdk\_faker 根据已加载的参数，启动一定数量的线程）每个线程管理多个SDK，向vod push请求不同的文件，当接受到足够的chunk数后，线程会被关闭（表示sdk下载完成）

sdk发送请求的频度：当接受返回的状态码为200时，间隔100ms或者120ms继续发送请求（间隔由配置文件决定）；当接受返回的状态码为503时，固定间隔或者随机间隔重新请求（间隔由配置文件决定）

4.当sdk\_faker 请求数据时，会记录每个sdk发出请求的信息（线程号，发出请求的时间，chunk数范围等）和收到请求返回的信息（接受到请求的时间，状态码等）

展示模块：

Simulator 模块会将sdk的行为写入sdk\_fake.log文件，filebeat读取该文件后交由logstash 和 elasticstah 处理最终展示在kibana上

# 4.设计实现

待补充

# 5.问题与风险

cdn simulator需要开发提供（或者能直接将文件写入vod push的工具）

目前计划一个线程表示一个sdk，如果需要模拟上万级别的sdk，可能需要采用多进程加多线程的方式来模拟sdk；vod push是否需要将数据返回给模拟的sdk（需与开发确认）

**6.附录**

## 6.1 vodpush 内存置换策略模拟器log格式

1.sdk simulator log格式

发送请求时打印log：

{topic: send\_request, sdk:1, timestamp: 1513936510379, start\_chunk\_id: 2520,end\_chunk\_id: 2532}

接受请求时，打印log：

{topic: receive\_response, sdk:0, timestamp: 1513936510294, start\_chunk\_id: 2508,end\_chunk\_id: 2508, status\_code: 200}  
 2.vod push信息使用monitor log

**6.2 搭建vodpush 内存置换策略模拟器环境**

1.nginx：用于mock vod push注册的返回和 vod push获取任务的返回

配置如下：

server {

listen 80;

server\_name push-hub.crazycdn.com;

client\_max\_body\_size 1m;

# vod push注册的返回

location /register {

return 200 '{"ip":"192.168.2.28"}';}

# vod push获取任务的返回

location / {

return 200 '[{"file\_id": "561B90A24D754AC6FAFF7D3A54E9DA2D","file\_

url": "http://127.0.0.1:8088/4k/test01.mp4","file\_size": 1073741824,"ppc": 32,"cpp

c": 1,"piece\_size": 1392,"operation": "download","priority": 0,"file\_type":"bigfil

e"}]';}

}

6.3 测试场景设计

环境：192.168.4.181 2核4G disk：240G 部署vod push

192.168.4.180 2核4G disk：40G 部署 内存置换策略模拟器

6.3.1 场景1

vod push：1G文件 -disk 20 –mem 4G

内存置换策略模拟器： 10个SDK

启动间隔[0,1]

vod push返回间隔120ms 50% 100ms 50%

下发任务完成用时：3min20s

6.3.1 场景2

vod push：1G文件

内存置换策略模拟器： 1000个SDK

启动间隔[0,1]

vod push返回间隔120ms 50% 100ms 50%

下发任务完成用时：15min 发现问题（单纯多线程时，某些线程发送的时间间隔5秒左右）（解决方案 多进程+多线程）

6.3.1 场景3

vod push：1G文件

内存置换策略模拟器： 1000个SDK

启动间隔[0,1]

vod push返回间隔120ms 50% 100ms 50%

下发任务完成用时：

6.3.1 场景3

vod push：1G文件

内存置换策略模拟器： 1000个SDK

启动间隔[0,1]

vod push返回间隔120ms 50% 100ms 50%

下发任务完成用时：