

TCP 负载均衡分发器&&桥接器 (G6)

Calvin

版本修订

版本	日期	修订人	内容
1.0.0	2016-03-13	calvin	创建文档
1.0.1	2016-03-20	calvin	新增章节 性能测试

目录索引

1	背景	4
2	概述	4
2.1	使用场景	4
2.2	支持几乎所有主流负载均衡算法	4
2.3	功能优势	5
3	场景与架构.....	5
3.1	与无负载均衡功能的通讯软件配合实现负载均衡分发	5
3.2	网站反向代理.....	6
3.3	双网卡桥接不同网段	6
4	安装	7
4.1	编译安装	7
4.1.1	编译	7
4.1.2	试运行	8
5	使用参考.....	9
5.1	系统结构	9
5.2	启停和更新.....	10
5.2.1	启动	10
5.2.2	停止	10
5.2.3	更新	10
5.3	配置文件格式.....	11
5.4	管理命令	14
6	性能测试.....	15
7	设计与实现.....	19
7.1	内部结构	19
7.2	平滑重启	20

1 背景

2014 年我研发了负载均衡软件 G5，作为开源项目发布到网上，被很多公司采用，反馈了大量意见和建议，比如希望增加后端心跳功能等，在这里感谢广大朋友们的支持和帮助。近两年的开源发展，G5 已经趋近成熟，也暴露了不少设计不足，比如没有充分利用多核环境。两周年之际，我决定重新研发 G5 第二版，名字就叫做 G6，重点解决 G5 设计不足，也从代码架构上深度优化性能。

2 概述

G6 是一款高性能、易使用、支持远程管理的 TCP 桥接器&&负载均衡分发器，基于 Linux 的 epoll 事件驱动非堵塞全异步框架实现。

G6 工作在网络 4 层 TCP/IP，这意味着不仅可以用于网站 HTTP 协议，还能用在 SMTP、POP、TELNET、SSH 等协议上。

2.1 使用场景

- * 与无负载均衡功能的通讯软件配合实现负载均衡分发
- * 网站反向代理
- * 双网卡桥接不同网段

2.2 支持几乎所有主流负载均衡算法

- * 主备
- * 轮询
- * 最少连接数
- * 最小响应时间

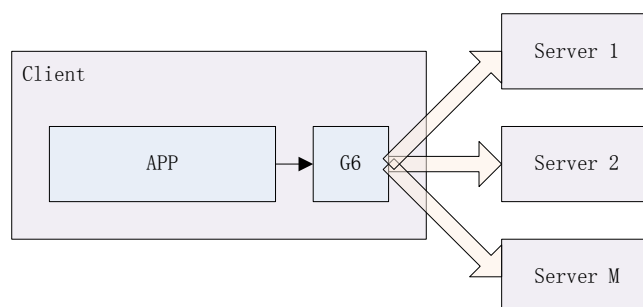
- * 随机
- * 哈希

2.3 功能优势

- * 充分利用多核环境，采用父子进程（监控进程+工作进程）+多线程（缺省数量为 CPU 核数）的软件架构。
- * 平滑重载配置和自我升级，真正的无缝更新，不会影响现有会话和侦听服务端口。
- * 基于红黑树的会话超时管理，这是与 `epoll` 配合最好的超时管理方式。
- * 性能是同类软件中最高的，比 `nginx` 还快，具体见性能测试章节。
- * 配置文件格式简洁、灵活，配置参数丰富，如总会话数控制、总 IP 数控制、每个 IP 会话数控制、超时控制、心跳周期、暂禁时间等，拥有全局继承机制。
- * 提供了出错暂禁、心跳报告、管理命令主动暂禁三种隔离后端服务器模式，最后一种可与运维脚本配合实现更复杂的定制化的健康检测。
- * 源码文件分布合理、代码结构清晰便于阅读，适合定制化改造。编译成可执行程序不到 200KB，无第三方软件依赖。内存占用恒定。
- * 可通过管理端口远程查询和管理。

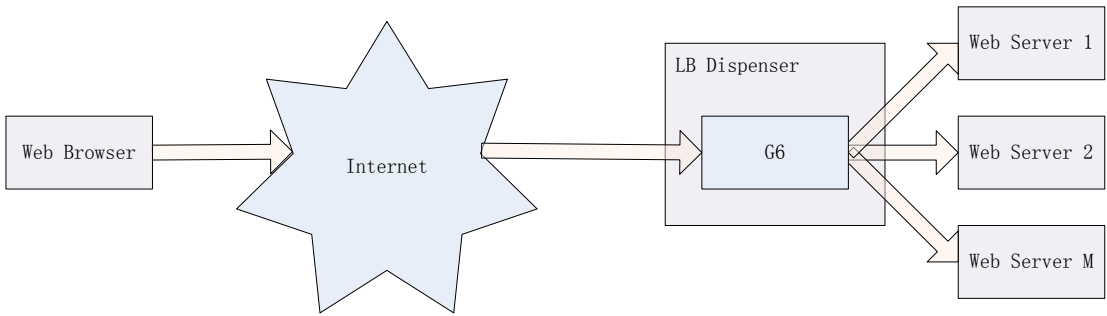
3 场景与架构

3.1 与无负载均衡功能的通讯软件配合实现负载均衡分发



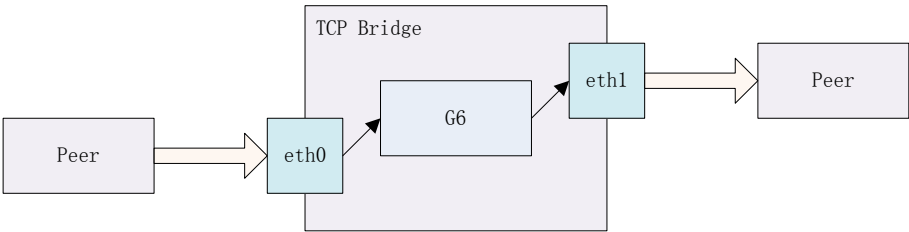
无须改造老的通讯软件，中间接入 G6 即可实现负载均衡分发，避免改造通讯软件带来的工作量和风险。

3.2 网站反向代理



网站服务器集群前面的反向代理，高并发（理论上可支持上万条连接）、高性能（比 `nginx` 快一倍）、高伸缩性（根据负载均衡算法和地址配置，在线动态伸缩服务器集群）。

3.3 双网卡桥接不同网段



（这种偷鸡摸狗的场景自己领会吧）

4 安装

4.1 编译安装

4.1.1 编译

（以 Linux 环境为例）

从源码托管网站下载解开安装包或 git 克隆源码库

<http://git.oschina.net/calvinwilliams/G6>

<https://github.com/calvinwilliams/G6>

进入源码目录

```
$ cd src
```

修改安装配置文件

```
$ vi makeinstall
```

如有必要，修改目标可执行程序 G6 安装目录变量"_BINBASE"，这里以安装到本用户中为例

```
_BINBASE      =      $(HOME)/bin
```

编译源码

```
$ make -f makefile.Linux clean
rm -f LOGC.o
rm -f rbtree.o
rm -f Util.o
rm -f main.o
rm -f Envirment.o
rm -f Config.o
rm -f MonitorProcess.o
rm -f WorkerProcess.o
rm -f AcceptThread.o
rm -f ForwardThread.o
rm -f TimeThread.o
rm -f G6
$ make -f makefile.Linux
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c LOGC.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c rbtree.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c Util.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c main.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c Envirment.c
```

```
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c Config.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c MonitorProcess.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c WorkerProcess.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c AcceptThread.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c ForwardThread.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -I. -c TimeThread.c
gcc -g -fPIC -O2 -Wall -Werror -fno-strict-aliasing -o G6 LOGC.o rbtree.o Util.o
main.o Envirment.o Config.o MonitorProcess.o WorkerProcess.o AcceptThread.o
ForwardThread.o TimeThread.o -L. -lpthread
$ make -f makefile.Linux install
cp -f G6 /home/calvin/bin/
```

如果没有报错则成功编译链接出可执行程序 **G6** 并安装到目标目录（如果安装到\$HOME 以外，注意权限）

编译安装完成！

4.1.2 试运行

复制配置示例文件到配置文件目录

```
$ cp ../etc/G6.conf $HOME/etc/
```

测试可执行文件

```
$ G6
G6 v1.0.0 build Mar 13 2016 01:23:00
TCP Bridge && Load-Balance Dispenser
Copyright by calvin 2016
USAGE : G6 -f (config_pathfilename) [ -t (forward_thread_size) ] [ -s
(forward_session_size) ] [ --log-level (DEBUG|INFO|WARN|ERROR|FATAL) ]
[ --log-filename (logfilename) ] [ --no-daemon ]
```

以最小缺省配置运行（只有管理端口）

```
$ G6 -f $HOME/G6.conf
```

如果没有报错，发送管理命令“显示装载的所有配置”到管理端口

```
$ echo "show rules" | nc 127.0.0.1 8600
admin_rule_id | G | 127.0.0.1:* - 127.0.0.1:8600
```

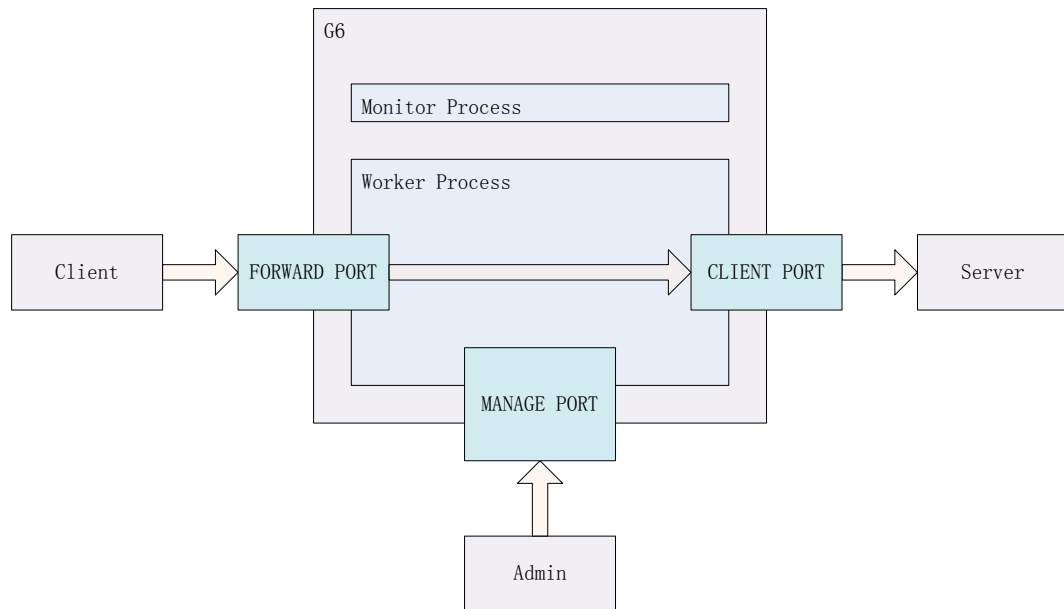
启动成功！

停止

```
ps -ef | grep G6 | grep -v grep | awk '{if($3==1)print $2}' | xargs kill
```


5 使用参考

5.1 系统结构



G6 由管理进程（父进程）和工作进程（子进程）组成，其中工作进程由接受连接线程、转发线程组和时间线程组成。

G6 对外侦听转发端口（FORWARD PORT），当有客户端连接上转发端口时，创建一条服务端新连接，并产生数据传输方向反向的两个会话用于后续数据交换。

当两个方向有一端有数据到达时，转发到另一端。

当两个方向有一端连接断开时或定时器达到设定超时，拆毁另一端，同时销毁两个会话。

G6 还提供可配置的管理端口（MANAGE PORT），通过管理端口可以查询内部状态、发送命令等。

5.2 启停和更新

5.2.1 启动

不带命令行参数执行 G6 将得到完整的命令行参数列表

```
USAGE : G6 -f (config_pathfilename) [ -t (forward_thread_size) ] [ -s  
(forward_session_size) ] [ --log-level (DEBUG|INFO|WARN|ERROR|FATAL) ]  
[ --log-filename (logfilename) ] [ --no-daemon ]
```

各项参数说明如下：

- f (config_pathfilename) 指定规则配置文件名
- t (forward_thread_size) 指定转发线程数量，缺省为 CPU 核数
- s (forward_session_size) 最大会话数量。连接数*2=会话数
- log-level (DEBUG|INFO|WARN|ERROR|FATAL) 指定日志等级，缺省为 INFO
- log-filename (logfilename) 指定日志文件名，缺省为\$HOME/log
- no-daemon 非守护进程模式，缺省为守护进程模式

5.2.2 停止

向管理进程（ppid 为 1）发送 TERM 信号即可优雅停止 G6，也可以向所有 G6 进程发送 KILL 信号以强制立即停止。

5.2.3 更新

向管理进程发送 USR1 信号可重新生成日志文件。

向管理进程发送 USR2 信号可优雅重启，用于重载配置、可执行程序平滑升级。

5.3 配置文件格式

试运行时使用的是最小配置，只配置了管理规则，以下为分发规则配置示例：

接受来自互联网 HTTP 请求，连接反向代理服务端口 80，然后轮询分发给内网 WEB 服务器集群 192.168.0.101~192.168.0.105:80，这样配置（可能需要 root 权限）：

```
my_rule1_id RR *:~ 0.0.0.0:80 > 192.168.0.101:80 192.168.0.102:80
192.168.0.103:80 192.168.0.104:80 192.168.0.105:80 ;
```

在双网卡连接两个网段的机器 192.168.0.51 里，接受来自 192.168.0.* 的连接，转发端口为 8080，转发给 192.168.1.101:8080，这样配置：

```
my_rule2_id RR 192.168.0.*:* - 192.168.0.51:8080 > 192.168.1.101:8080 ;
```

本地软件连接本地 G6 转发端口 8601，然后主备机制分发给 192.168.0.101~192.168.0.102 的 10601 端口，这样配置：

```
my_rule3_id MS 127.0.0.1:* - 127.0.0.1:8601 > 192.168.0.101:10601
192.168.0.102:10601 ;
```

转发时限制来源地址最多接受一个连接，除了 192.168.0.1

```
my_rule4_id RR ( max_connection = 1 ) 192.168.0.100:* ( max_connections = 0 )
*: 192.168.0.2:8080 > 192.168.0.3:808 ;
```

转发时设置会话超时时间为 120 秒

```
my_rule5_id RR 192.168.0.1:* - ( timeout = 120 ) 192.168.0.2:8080 >
192.168.0.3:8080 ;
```

转发时限制转发端与目标服务器最多 10 个连接

```
my_rule6_id RR *:~ 192.168.0.2:8080 > ( max_connections = 60 ) 192.168.0.3:8080
192.168.0.4:8080 192.168.0.5:8080 ;
```

设置所有目标服务器为报告模式，必须每隔 60 秒内报告才能有效，否则失效隔离

```
my_rule7_id RR *:~ 192.168.0.2:8080 > ( heartbeat = 60 ) 192.168.0.3:8080
192.168.0.4:8080 192.168.0.5:8080 ;
```

G6.conf 缺省配置文件自带完整使用说明注释

```
////////////////////////////////////
// G6 配置文件格式
// 1.各字段（包括括号、分号等符号）之间必须用白字符隔开，白字符包含空格、TAB 和换行
// 2.每条配置用;结尾，中间可以任意折行
// 3.可以用 //注释到到行尾 或 /*...注释...*/
```

```

////////////////////////////////////
// 全局属性 :
(
    // moratorium = 60 , // 服务端不可用时暂禁时间, 默认不暂禁
    // timeout = 60 , // 客户端连接超时, 默认不超时
    // max_ip = 1 , // 最大 IP 数量, 默认无限制
    // max_connections = 2 , // 最大客户端连接数量, 默认无限制
    // max_connections_per_ip = 2 , // 每个 IP 最多客户端数量, 默认无限制
);

////////////////////////////////////
// 管理规则 :
// 规则名 (白字符) G (白字符) 客户端地址列表 (白字符) - (白字符) 侦听地址列表 (白字符);
admin_rule_id G 127.0.0.1:* - 127.0.0.1:8600 ;

/*
规则名为唯一的 64 个字符以内的字符串

客户端地址列表 :
    客户端 IP:客户端 PORT
或
    客户端 1IP:客户端 1PORT 客户端 2IP:客户端 2PORT?...
IP 和 PORT 可以使用 '*' 和 '?' 通配, 如 "127.0.0.1:*"、"192.168.0.*:*"、"*:*"
*/

////////////////////////////////////
// 转发、分发规则 :
// 规则名 (白字符) 负载均衡算法 (白字符) 客户端地址列表 (白字符) - (白字符) 侦听地址列表 (白字符) > (白字符) 服务端地址列表 (白字符);

// 接受来自互联网 HTTP 请求, 连接反向代理服务端口 80, 然后轮询分发给内网 WEB 服务器集群
192.168.0.101~192.168.0.105:80, 这样配置 (可能需要 root 权限):
// my_rule1_id RR *:80 - 0.0.0.0:80 > 192.168.0.101:80 192.168.0.102:80
192.168.0.103:80 192.168.0.104:80 192.168.0.105:80 ;

// 在双网卡连接两个网段的机器 192.168.0.51 里, 接受来自 192.168.0.* 的连接, 转发端口为 8080,
转发给 192.168.1.101:8080, 这样配置:
// my_rule2_id RR 192.168.0.*:* - 192.168.0.51:8080 > 192.168.1.101:8080 ;

// 本地软件连接本地 G6 转发端口 8601, 然后主备机制分发给 192.168.0.101~192.168.0.102 的
10601 端口, 这样配置:
// my_rule3_id MS 127.0.0.1:* - 127.0.0.1:8601 > 192.168.0.101:10601
192.168.0.102:10601 ;

```

```
// 转发时限制来源地址最多接受一个连接，除了 192.168.0.100
// my_rule4_id RR ( max_connection = 1 ) 192.168.0.100:* ( max_connections = 0 )
*: 192.168.0.2:8080 > 192.168.0.3:808 ;

// 转发时设置会话超时时间为 120 秒
// my_rule5_id RR 192.168.0.1:* - ( timeout = 120 ) 192.168.0.2:8080 >
192.168.0.3:8080 ;

// 转发时限制转发端与目标服务器最多 10 个连接
// my_rule6_id RR *: * - 192.168.0.2:8080 > ( max_connections = 60 )
192.168.0.3:8080 192.168.0.4:8080 192.168.0.5:8080 ;

// 设置所有目标服务器为报告模式，必须每隔 60 秒内报告才能有效，否则失效隔离
// my_rule7_id RR *: * - 192.168.0.2:8080 > ( heartbeat = 60 ) 192.168.0.3:8080
192.168.0.4:8080 192.168.0.5:8080 ;

/*
my_rule9_id RR
    ( max_connections = 100 )
    192.168.0.1:* ( max_connections = 10 )
    192.168.0.2:* ( max_connections = 5 )
    192.168.0.*:* ( max_connections = 1 )
    -
    ( timeout = 60 )
    192.168.0.50:8080 ( timeout = 10 )
    192.168.0.50:8081
    >
    ( max_connections = 100 , heartbeat = 60 )
    192.168.0.101:80 ( max_connections = 20 )
    192.168.0.102:80 ( max_connections = 10 )
    192.168.0.103:80
    192.168.0.104:80
    192.168.0.105:80 ( heartbeat = 30 )
    ;
*/
*/

/*
```

侦听地址列表 和 服务端地址列表 同 客户端地址列表 但不允许出现通配符

地址属性格式：

((白字符) 属性名 (白字符) = (白字符) 属性值 (白字符))

或

((白字符) 属性 1 名 (白字符) = (白字符) 属性 1 值 (白字符) , (白字符) 属性 2 名 (白字符) = (白字符) 属性 2 值 (白字符) , ...)

全局属性会自动继承到地址列表前属性，地址列表前配置的属性会自动继承到地址列表中，每个地址后也可以配置属性，越后者优先

客户端地址列表中可以配置的属性有

`max_connections` : 最大连接数

转发端地址列表中可以配置的属性有

`timeout` : 双向会话超时时间（秒）；可以加后缀单位's'或'm'或'h'

服务端地址列表中可以配置的属性有

`max_connections` : 最大连接数

`heartbeat` : 心跳周期，当配置该参数时，必须在周期内通过管理端口向该IP:PORT报告，否则该服务端被失效隔离；可以加后缀单位's'或'm'或'h'

报告简易命令：

```
echo "heartbeat 192.168.0.101 80" | nc 127.0.0.1 8600
```

也可与复杂的检测脚本配合

负载均衡算法：

MS : 主备

RR : 轮询

LC : 最少连接数

RT : 最小响应时间

RD : 随机

HS : 哈希（根据来源IP）

*/

5.4 管理命令

直接telnet连接管理端口，即可进入管理界面，然后发送管理命令即可，如

```
telnet 127.0.0.1 8600
```

也可以通过其它工具如nc一次性发送执行完后立即返回

```
echo "show sessions" | nc 127.0.0.1 8600
```

管理命令如下：

查询以装载的所有规则

```
show rules
```

查询当前所有会话

```
show sessions
```

临时暂禁某个后端服务端口

```
disable (ip) (port)
```

恢复某个后端服务端口

```
enable (ip) (port)
```

向 G6 报告，以免被失效隔离（需要周期性发送）

```
heartbeat (ip) (port)
```

退出管理命令界面

```
quit
```

6 性能测试

在网站反向代理场景，我们拿使用最广泛的 Nginx 来做比对。

因为家里机器有限，性能测试用两台 Linux 来做，一台是实体机 A，一台是虚拟机 B（外面实体机是 Windows），A 机里用 ab 发起并发 300 共 50000 个 HTTP 请求到 B，B 上部署 G6 和 Nginx，配置轮询算法分发回 A 机上的 Apache 的 5 个虚拟站点。

A 机：

CPU AMD E-350 Processor 1.6GHz*2

内存 4GB

B 机：

CPU Intel(R) Core(TM) i3-3240 CPU @ 3.40GHz*2

内存 500Mbo

G6 配置如下：

```
admin_rule_id G
    127.0.0.1:*
    -
    127.0.0.1:8600
    ;
press_rule_id RR
    192.168.6.111:*
    -
    192.168.6.54:8601
    >
    192.168.6.111:8801
    192.168.6.111:8802
    192.168.6.111:8803
    192.168.6.111:8804
```

```
192.168.6.111:8805
```

```
;
```

Nginx 配置如下（只输出错误日志）：

```
events {
    worker_connections 1024;
}
http {
    include mime.types;
    default_type application/octet-stream;
    sendfile on;
    upstream site {
        server 192.168.6.111:8801 ;
        server 192.168.6.111:8802 ;
        server 192.168.6.111:8803 ;
        server 192.168.6.111:8804 ;
        server 192.168.6.111:8805 ;
    }
    server {
        listen 8701 ;
        server_name 192.168.6.54 ;
        location / {
            root html ;
            index index.html index.htm ;
            proxy_pass http://site ;
        }
    }
    access_log off ;
}
```

启动 G6（只输出错误日志）

```
G6 -f press.conf --log-level ERROR
```

启动 Nginx

```
nginx
```

压测脚本

```
if [ $# -eq 0 ] ; then
    MAX_ROUND=999
else
    MAX_ROUND=$1
fi

ROUND=0
while [ $ROUND -le $MAX_ROUND ] ; do
    TIME1=`ab -c 300 -n 50000 http://192.168.6.54:8601/index2.html | grep
```



```

"Requests per second" | awk '{print $4}'`
    sleep 2
    TIME2=`ab -c 300 -n 50000 http://192.168.6.54:8701/index2.html | grep
"Requests per second" | awk '{print $4}'`
    sleep 1
    echo "ROUND: $ROUND      G6: $TIME1      Nginx: $TIME2"
    TIMES1[$ROUND]=$TIME1
    TIMES2[$ROUND]=$TIME2
    sleep 1
    ROUND=`expr $ROUND + 1`
done

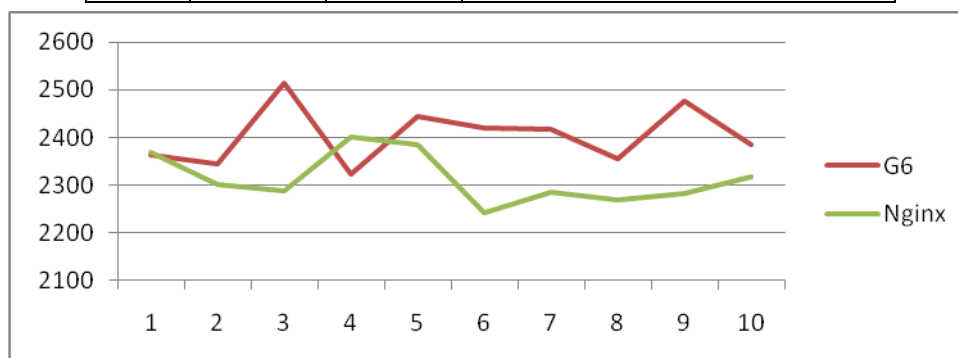
echo "--- result -----"

ROUND=0
while [ $ROUND -le $MAX_ROUND ] ; do
    echo "ROUND: $ROUND      G6: ${TIMES1[$ROUND]}      Nginx:
${TIMES2[$ROUND]}"
    ROUND=`expr $ROUND + 1`
done

```

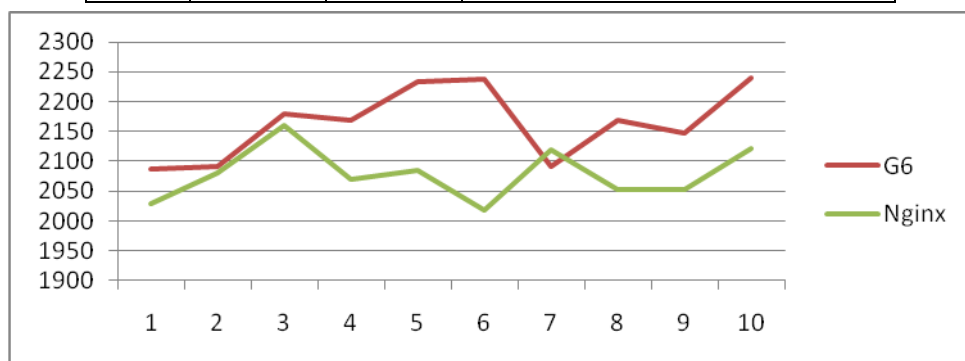
反复跑了 10*3 轮，结果如下：

ROUND	G6	Nginx	(G6-Nginx)/Nginx*100%
1	2362.43	2370.41	-0.34%
2	2342.65	2302.41	1.75%
3	2514.02	2287.89	9.88%
4	2322.4	2402.21	-3.32%
5	2444.29	2387.26	2.39%
6	2420.46	2243.33	7.90%
7	2418.13	2287.04	5.73%
8	2355.61	2270.59	3.74%
9	2475.75	2284.26	8.38%
10	2385.67	2318.5	2.90%

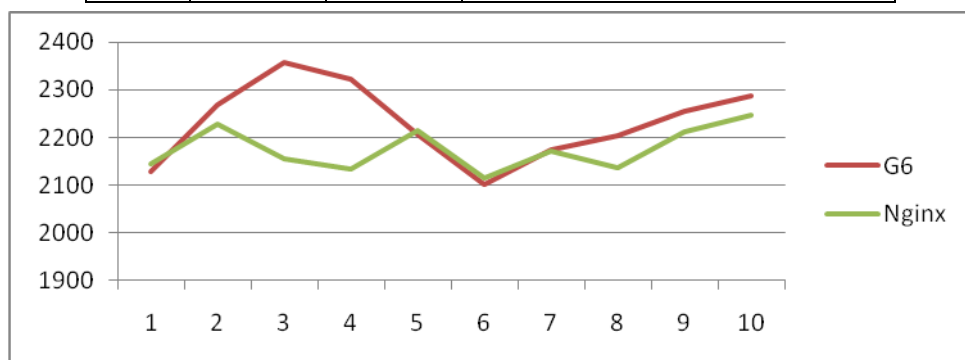


ROUND	G6	Nginx	(G6-Nginx)/Nginx*100%
1	2087.81	2029.56	2.87%

2	2091.75	2080.61	0.54%
3	2179.92	2159.78	0.93%
4	2169.49	2069.9	4.81%
5	2233.48	2085.19	7.11%
6	2237.65	2018.47	10.86%
7	2092.45	2119.8	-1.29%
8	2170.11	2052.72	5.72%
9	2147.43	2053.06	4.60%
10	2240.2	2122.01	5.57%



ROUND	G6	Nginx	(G6-Nginx)/Nginx*100%
1	2127.61	2145.95	-0.85%
2	2268.37	2227.7	1.83%
3	2356.53	2155.04	9.35%
4	2322.03	2134.54	8.78%
5	2205.4	2214.02	-0.39%
6	2101.11	2115.53	-0.68%
7	2174.56	2171.36	0.15%
8	2203.29	2136.8	3.11%
9	2253.41	2211.55	1.89%
10	2286.82	2246.1	1.81%

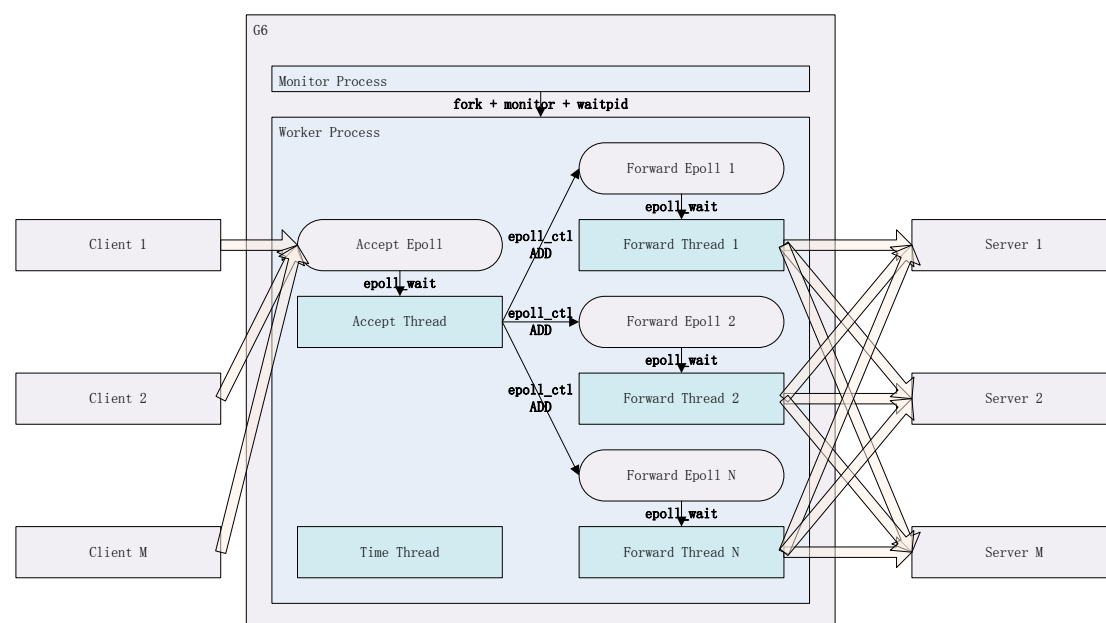


G6 在负载均衡性能要高于 Nginx！

功能上 G6 比 Nginx 多了隔离后端服务器机制，这是负载均衡器所必须具有的特性。

7 设计与实现

7.1 内部结构



G6 采用父子进程+多线程结构设计。

管理进程（src/MonitorProcess.c）派生工作进程，然后监控工作进程存活，一旦发现工作进程结束就重启之。

工作进程（src/WorkerProcess.c）创建侦听端口放入侦听 `epoll` 池，创建若干（缺省为 CPU 核数）转发线程、时间线程，本身转化为连接接受线程。

连接接受线程（src/AcceptThread.c）从侦听 `epoll` 池（ET 模式）等待接受新连接，均衡放入转发 `epoll` 池。

转发线程（src/ForwardThread.c）从自己的转发 `epoll` 池（LT 模式）中等待数据交换事件，并处理之。

时间线程（src/TimeThread.c）定时刷新时间，提供给其它线程使用，避免写日志等操作频繁取当前时间，提高性能。

7.2 平滑重启

平滑重启用于重载配置和可执行程序无缝升级。

管理进程收到平滑重启信号（USR2）后，保存所有侦听端口描述符到环境变量，通知工作进程关闭所有侦听端口，然后创建下一代新管理进程，最后优雅等待结束。新管理进程在创建侦听端口时，如果环境变量中存在老一代的侦听端口，则复用，否则新创建。