

一个模拟的负载均衡系统的实现

设计文档



V2.00

pengzhengpu@foxmail.com

**目录**

[1. 引言 2](#_Toc363415829)

[1.1. 目的 2](#_Toc363415830)

[1.2. 背景 2](#_Toc363415831)

[2. 总体架构 2](#_Toc363415832)

[2.1. 需求概述 2](#_Toc363415833)

[2.2. 系统框架 3](#_Toc363415834)

[2.3. 界面设计 4](#_Toc363415835)

[3. 分系统描述 5](#_Toc363415836)

[3.1. 通用常量和数据结构定义 5](#_Toc363415837)

[3.2. 服务器 5](#_Toc363415838)

[3.2.1. 实现功能 5](#_Toc363415839)

[3.2.2. 程序逻辑 6](#_Toc363415840)

[3.2.3. 测试要点以及限制条件 7](#_Toc363415841)

[3.3. 负载均衡器 7](#_Toc363415842)

[3.3.1. 实现功能 7](#_Toc363415843)

[3.3.2. 输入、输出 8](#_Toc363415844)

[3.3.3. 核心数据结构和算法 8](#_Toc363415845)

[3.3.4. 程序逻辑 11](#_Toc363415846)

[3.3.5. 测试要点以及限制条件。 12](#_Toc363415847)

[3.4. 客户端 12](#_Toc363415848)

[3.4.1. 实现功能 12](#_Toc363415849)

[3.4.2. 程序逻辑 13](#_Toc363415850)

[3.4.3. 测试要点以及限制条件。 14](#_Toc363415851)

# 引言

## 目的

为了构建可伸缩的，高可用的网络服务，很多大型网站都采用了负载均衡技术。利用负载均衡技术，可以将多台廉价的、低性能的服务器，组合成一台性能强劲的，高可用的虚拟服务器。

负载均衡的常见实现方式大致如下：将网络服务的地址（如公网IP地址、tcp套接字等）部署在负载均衡器上，而不是真实的服务器上；将负载均衡器作为网络服务的总入口，接收用户的所有访问请求；负载均衡器接收到用户的访问请求后，将访问请求按照一定的策略分发给某一台真实的服务器进行处理；真实服务器，对访问请求进行处理后，将处理结果发送给负载均衡器；负载均衡器接收到真实服务器的处理结果，将他发送给用户。

## 背景

本系统设计工作来源于2013年“中兴捧月杯”5th校园赛事嘉年华-程序设计大赛的第五道题目“**一个模拟的负载均衡系统的实现**”。具体要求见官网：

初赛要求<http://company.dajie.com/zte/task?t=1>；

复赛要求<http://company.dajie.com/zte/task?t=16>；

开发语言C++，开发平台：VC6.0，运行环境Windows XP / Windows 7。若有疑问，请联系[pengzhengpu@foxmail.com](mailto:pengzhengpu@foxmail.com)。

# 总体架构

## 需求概述

在PC机上实现一个模拟的负载均衡系统。他包含如下3个可执行程序：

* 服务端(server.exe)――辅助程序，通过UDP端口，提供时间查询服务；
* 负载均衡器(LB.exe)――核心程序，用于实现负载均衡功能；
* 客户端(client.exe)――辅助程序，通过UDP端口，访问时间查询服务；

表 1 各分系统需求对比

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **对象** | **多进程** | **进程 id** | **usr id** | **udp端口** | **调试** | **统计** | **日志** |
| 服务器 | 支持 | 唯一 | 无 | 绑定唯一 | 开/关 | 一直 | 无 |
| 负载均衡器 | 不支持 | 唯一\* | 无 | 绑定唯一\* | 开/关 | 一直 | 有 |
| 客户端 | 支持 | 不唯一 | 唯一 | 系统默认 | 长开 | 结束 | 无 |

\*表示参数只能从配置文件中读取，其他均可以通过配置文件、命令行、运行时来指定参数

此外，系统须提供服务器心跳健康检测功能，负载均衡器提供轮转、比例权重、最快响应速度等负载均衡算法，负载均衡器还需维护会话的功能。

## 系统框架



图 1 系统核心架构

负载均衡器创建三个子线程，其作用分别为：

子线程A：完成心跳健康检测功能和定时清理超时的会话；

子线程Ｂ：通过客户套接字接收客户端请求，并通过服务套接字将其转发给服务器；

子线程Ｃ：通过服务套接字接收服务结果，并通过客户套接字转发给客户端；

负载均衡器共创建有３个套接字，在套接字的收发不需要进行两线程之间的同步处理。此外，为了避免在大量数据交互时，服务器和客户端的界面主线程出现长期阻塞，也都采用了子线程完成数据发送和接收功能。

整个服务流程描述：客户端通过子线程发送时间请求消息，负载均衡器通过Ｂ子线程接收该消息，并建立会话（可通过配置文件可改），通过特定算法（配置文件可改）选取服务器，对消息进行适当修改，将其发送给选取的服务器，服务器完成运算将结果发送给负载均衡器，负载均衡器由Ｃ子线程接收到该结果并做适当修改将其转发给客户端，客户端子线程接收结果，时间请求服务到此完成。在此过程中，负载均衡器的Ａ子线程负责心跳健康检测和会话超时检测。

## 界面设计



图 2　服务器界面设计



图 3　负载均衡器界面设计



图 4客户端界面设计

# 分系统描述

## 通用常量和数据结构定义

1. **常量定义**

*#define CMDTYPE\_TIMEREQ 0x00* //时间请求

*#define CMDTYPE\_TIMEACK 0x01* //时间应答

*#define CMDTYPE\_HEARTREQ 0x02* //心跳请求

*#define CMDTYPE\_HEARTACK 0x03* //心跳应答

1. **消息结构体定义：通过msg\_type字段的值，来区分不同类型的消息。**

*typedef struct*

*{*

*unsigned src\_id;* //消息的发送进程是谁，就填谁的id

*unsigned dst\_id;* //消息的接收进程是谁，就填谁的id

*unsigned usr\_id;* //发送"时间请求"消息时填写， 回复"时间应答"消息时，其值要与请求消息保持一致

*unsigned msg\_type;* //消息类型：0, 时间请求；1, 时间答应；2, 心跳请求；3, 心跳应答

*char data[32];* //服务端回复"时间应答"消息时，在data中填入当前时间的字符串，形式如"2013-06-20 13:56:28"即可

*} t\_msg;*

1. **消息统计结构体定义：**

*typedef struct*

*{*

*unsigned int recv\_num;*  //接收消息总数

*unsigned int send\_num;*  //发送消息总数

*unsigned int correct\_inrecv;* //接收中正确数

*unsigned int wrong\_inrecv;* //接收中错误数

*} t\_msg\_statistics;*

## 服务器

### 实现功能

**设置服务器参数：**运行时设置进程ID、UDP端口号；

**服务开关：** 提供手动切换“开启服务”、“停止服务”的功能；

**调试开关：**打开调试开关后，会以报表的形式显示每条信息的详情（消息类型，src\_id, dst\_id, usr\_id, 若有时间信息则显示否则为空串），关闭调试开关则不显示

**心跳报表显示开关：**在调试开关打开的基础上，可以控制心跳信息是否需要显示在报表中；

**统计功能：**在运行过程中，随时可以看程序接收消息条数（正确条数和错误条数），应答消息条数；

**清除记录功能：**该功能可以清除之前运行的统计数据和消息报表显示的信息；

### 程序逻辑



图 5 服务器核心逻辑框图

### 测试要点以及限制条件

服务器发起多个进程时，编辑框的UDP端口绝不能重复，否则会遇到端口绑定失败的情况，进程ID需要保证唯一性。进程ID、端口号以及获取的主机IP地址均要手动转存至负载均衡器的配置文件中，如果没有这些信息，负载均衡器无法向其分发客户端请求。

## 负载均衡器

### 实现功能

**LB配置信息显示**：显示进程id、与服务端通信的UDP端口号，与客户端通信的UDP端口号；

**服务器接口统计**：接收信息条数（正确条数，错误条数），发送信息条数，

**客户端接口统计**：接收信息条数（正确条数，错误条数），发送信息条数；

**心跳检测功能：**负载均衡器每隔0.5秒,向每个服务端发送一个心跳报文。如果连续4次收不到某个服务端的心跳响应，就认为服务端出现了故障。后续对时间请求消息做负载均衡时，就不再分发给此服务端处理；

**多种负载均衡算法：**支持轮转方式分发、支持按比例分发、支持基于最快响应的负载均衡算法。在程序启动时，可通过配置文件选择相应的算法；

**会话保持功能：**支持三种方式：无会话保持功能、基于src\_id 会话保持功能、基于usr\_id会话保持功能；同样，在程序启动时可通过配置文件选择相应的会话保持功能；

**调试开关**：打开后会显示每条信息详情，关闭后不显示信息详情；

**心跳报表显示开关：**在调试开关打开的基础上，可以控制心跳信息是否需要显示在报表中；

**清除记录**：清除当前的统计信息、消息详情报表显示以及缓存的客户端地址信息；

**后台日志功能**：程序运行时，若出现异常后台会写日志，供后续分析；

**后台核心功能**：接收客服端请求，将请求转发给通过算法选择的特定在线服务器，服务器将运算结果发给负载均衡器，后者将其转发给客户端，即对外完成一次服务；

### 输入、输出

**输入**：进程id，一个UDP端口（收发客户端的消息），一个UDP端口（收发服务端的消息），支持服务端个数，采取负载均衡算法，采用会话保持方式，以及每个服务端的id、ip、udp端口、权重值。配置文件中的命名方式必须统一。负载均衡器在进程启动时通过配置文件读入这些信息，且运行过程中不会改变。配置文件名：”lbconfig.ini”。

**输出**：程序运行过程出现异常，将相关信息（发生时间、事件描述、可能原因以及错误码）写到日志文件中。日志文件名：”LoadBalancer.log”。

### 核心数据结构和算法

**核心数据结构**：

1. 服务器信息管理：

*struct t\_server\_info*

*{*

*UINT proc\_id;* //服务器的进程ID

*SOCKADDR\_IN server\_addr;* //地址信息（AF\_INET IP，UDP端口）

*unsigned respond\_speed;* //服务器心跳包的响应速度,相对单位： 最快响应分发策略

*char ratio;*//服务器权重值，取值范围1-10：权重分发策略用

*char unrecv\_heart\_count;*  //心跳计数，连续4次收不到，则判定服务器故障

*};*

*vector<t\_server\_info> m\_vecServerPoint;*

在程序启动时通过配置文件读入服务器信息，以server\_info封装，保存在vector容器中。服务器心跳健康检测功能每隔0.5秒,向每个服务端发送一个心跳报文。如果连续4次收不到某个服务端的心跳响应，就认为服务端出现了故障，以此来更新 m\_vecServerPoint中保存的服务器信息，供负载均衡器调配使用。

1. 客服端地址管理

*multimap<unsigned, SOCKADDR\_IN> m\_mapClientAddr;*

通过multimap容器来管理客服端地址，键为客服端的usr\_id，值为在接受客户端数据时保留的地址。负载均衡器在接收客户端请求时，将usr\_id和套接字地址信息缓存到multimap容器中，服务功能完成后，负载均衡器通过usr\_id查找相应地址，将服务结果转发给客户端之后，删掉该键和地址对。

1. 会话管理

*struct t\_session\_hold*

*{*

*unsigned client\_flag;*  //客户端标志,usr\_id | src\_id

*unsigned server\_id;*  //服务端id

*long last\_time;*  //最后一次时间请求的时间戳

*};*

*map<unsigned, t\_session\_hold> m\_mapSessionHold;*

因为来自同一src\_id（usr\_id）的请求只能算作一个会话，所以采用map容器来存放会话信息。键为客服端src\_id或usr\_id，根据配置文件设定，值为t\_session\_hold结构体元素。

**核心算法：**

//LB负载均衡算法--分发策略：{轮转方式，权重方式，最快响应速度方式}

*enum em\_DistributeStrategy {Poll\_Dist, Ratio\_Dist, Fast\_Dist};*对应在配置文件中取值为{0,1,2}

//会话保持功能：{无会话保持，基于src\_id方式，基于usr\_id方式}

*enum em\_SessionHold {No\_Session, Based\_Src\_id, Based\_Usr\_id};*对应在配置文件中取值为{0,1,2}



图 6 基本的负载均衡算法

在程序启动时，通过配置文件读取算法的代码{0,1,2}，在运行时通过对应的算法选取服务器编号。如果需要添加其他的负载均衡算法，只需在enum结构中添加一个字符串常量，在分发函数中修改switch-case语句即可完成添加新算法。保证了良好的可扩展性。

设置了三种会话保持功能：无会话，基于src\_id，基于usr\_id。在配置文件中的代码分别为{0,1,2}；src\_id和usr\_id统称客户端id，根据配置文件中的设置，t\_session\_hold数据结构的client\_flag保存不同的值，另外该client\_flag也会作为map容器的键。

注意：在会话功能基础上，负载均衡算法中基于权重的分发，本例程采用的一个会话占用一个权重而非一次请求占用一个权重。



图 7在会话保持功能下的选取服务器的流程图

### 程序逻辑



图 8 负载均衡器实现主要逻辑框图

### 测试要点以及限制条件。

**测试要点：**

配置文件中process\_id指定进程id；client\_udp\_port指定与客户端通信的端口；server\_udp\_port指定与服务端通信的端口号；server\_point\_number指定服务器的数量（服务器详情的键名需为ServerPoint\*格式递增）；distribute\_policy指定负载均衡算法，只在集合{0,1,2}取一个值，分别代表{轮转，按比例，最快响应速度}；session\_hold\_policy指定会话功能，只在集合{0,1,2}取一个值，分别代表{无会话，基于src\_id，基于usr\_id}。对于各个服务器详细信息，需保证process\_id唯一性，权重ratio取值范围为[1,10]。

如果要测试负载均衡算法和会话保持功能，请在打开测试按键的基础上，关闭心跳报表显示功能。这样好对比各个服务接收分发消息数。

**限制条件：**

在程序运行过程中，如果向离线服务器发送数据（心跳检测尚未达到4次），该请求就会丢失，导致multimap缓存该请求的地址冗余，可能会引起系统功能异常。

服务器判定为异常的过程不可逆，即服务器判定为不可用后，即时后面恢复正常也不可能再向其分发时间请求消息，除非重启LB.exe。

## 客户端

### 实现功能

**设置服务器端：**运行时设置包括IP地址，端口号，进程ID，在负载均衡系统中，服务器端设置实际就是负载均衡器配置的参数；

**设置客户端：**运行时设置包括进程ID和用户ID(usr\_id)，用户须保证各个客户端程序的usr\_id不同，否则负载均衡器数据转发出现错乱；

**请求消息条数：**设置在程序选择发送后，一次性发送消息的条数；

**显示消息详情：**在发送请求后，会实时显示每天消息的详细信息（消息类型，src\_id, dst\_id, usr\_id, 若有时间信息则显示否则为空串）；

**显示统计信息：**在与服务器端交互完成后，显示相关的统计信息（发送消息条数，接收消息条数，其中正确条数和错误条数），之后可以选择退出程序，也可以选择继续运行；

### 程序逻辑



图 9 客户端核心逻辑结构

### 测试要点以及限制条件。

客户端程序开启后，点击“发送”即可向负载均衡器发送时间请求信息，如果有请求不能反馈回来，消息详情报表刷新会出现停顿现象；

客户端在发起多进程，须保证各个进程的usr\_id不同，否则会导致负载均衡器转发结果异常；

此外，客户端可以直接设置服务器的IP、端口和进程ID，不通过负载均衡器而直接向服务器请求时间信息。