

基于线程池技术 DHCP 服务器的设计与实现

刘海燕¹, 吕延岗², 张红瑞²

(1. 北华航天工业学院 计算机系, 河北 廊坊 065000;

2. 石家庄职业技术学院 计算机系, 河北 石家庄 050081)

摘 要: DHCP 服务器在 IP 地址的集中管理和计算机网络参数的配置方面发挥着重要作用。但是, DHCP 服务器效率问题影响其在大型网络管理中的应用。文中提出了基于线程池技术 DHCP 服务器的设计思想, 提出了动态调整线程池大小的可行算法。测试结果显示, 基于线程池技术 DHCP 服务器的服务效率有了明显改善。

关键词: DHCP 服务器; 多线程; 线程池

中图分类号: TP311

文献标识码: A

文章编号: 1673-629X(2006)07-0076-03

Design and Implementation of DHCP Server Based on Thread Pool Technique

LIU Hai-yan¹, LÜ Yan-gang², ZHANG Hong-rui²

(1. North China Institute of Astronautic Engineering, Langfang 065000, China;

2. Shijiazhuang Vocational and Technology Institute, Shijiazhuang 050081, China)

Abstract: DHCP server plays an important role in the IP addressing and host configuration management. But, the efficiency of DHCP server affects its applications in managing a large network. In this paper we present a DHCP server model that is based upon thread pool technique, and bring forth an available arithmetic to adjust the size of the thread pool. Results of the test show great improvement on DHCP efficiency.

Key words: DHCP server; multithread; thread pool

0 引言

DHCP(Dynamic Host Configuration Protocol)^[1]可以为网络主机动态分配 IP 地址, 提高 IP 地址的利用率, 为解决当前紧张的 IP 地址资源问题提供了良好的方法; 同时, 它可以自动地配置客户机的网络参数, 在减少企业网管工作量的同时, 又避免了手动配置出错的情况。在 IP 地址日渐紧缺的今天, 在计算机日趋“傻瓜”化的今天, DHCP 的出现可以说是众望所归, 但是人们对它的要求也越来越高了。传统的 DHCP 服务器采用单线程的工作方式, 这种模式势必会影响它的工作效率, 因此考虑将多线程和线程池技术应用到 DHCP 服务器上, 肯定会大大提高 DHCP 服务器的性能。

1 线程技术和线程池机制

1.1 多线程技术

多线程^[2]是这样一种机制, 它允许在程序中并发执行多条指令流, 每条指令流为一个线程, 彼此间互相独立。

多个线程的执行是并发的, 也就是在逻辑上“同时”, 而不管是否是物理上的“同时”。如果系统只有一个 CPU, 那么真正的“同时”是不可能的, 但是由于 CPU 的速度非常快, 用户感觉不到其中的区别, 因此也不用关心它, 只需设想各个线程是同时执行即可。

1.2 线程池机制

所谓线程池^[3], 就是应用进程在启动或运行过程中创建一定数量的线程并存储到一个‘池子’中, 当客户请求到达时, 不是为之创建一个新的服务子线程, 而是从已存在的线程池中选一个空闲的线程为新的客户请求服务, 服务完成后, 子线程再回到空闲池中。通常, 系统还可以对线程的总数加以限制, 当客户请求太多的时候, 或者丢弃请求或者对请求排队, 采取何种方法取决于系统策略。采用线程池技术以后, 可以省去客户请求到来后创建线程的开销, 加快了服务器的响应速度, 同时还可以防止客户请求过多而造成服务器的崩溃。

2 基于线程池技术 DHCP 服务器设计思想

2.1 总体架构设计

基于线程池技术 DHCP 服务器设计采用 POSIX C 标准, 便于和 DHCP 服务器的其它模块整合。系统主要包

收稿日期: 2005-11-10

作者简介: 刘海燕(1981-), 女, 河北廊坊人, 助教, 研究方向为网络和数据库; 导师: 杜承烈, 副教授, 研究方向为计算机应用。

括分发请求模块、线程池模块及请求处理三大部分。

2.1.1 分发请求模块

该模块功能是监听客户的请求,当有客户请求时,接收客户的服务请求,将请求数据包交给线程池模块,继续监听。系统启动运行后,分发请求模块作为后台守护进程运行。

2.1.2 线程池模块

线程池模块主要包括线程链表、线程调度模块、线程池大小调整模块、线程池性能监视和报告模块。各模块功能概述如下:

* 线程链表:线程链表是用来存储工作线程信息的数据结构。所有线程作为 pthread_t 类型数组被组织到线程链表中,系统运行期间自动调整线程链表的结构。

* 线程调度模块(thread scheduling):为接收到的新任务分配执行线程。

* 线程池大小调整功能模块(adjustment):实时调整线程池的大小。

* 线程池性能监视和报告模块(monitor & performance report):监视线程池的性能,向管理员报告线程池的性能。

2.1.3 请求处理模块

当有新的客户请求到来时,线程调度模块会调用一个空闲线程来为其进行服务,该线程根据接收到的任务类型调用不同的处理模块,来实现客户的请求,比如请求新的 IP 地址、延长租约、释放租约等。处理完客户请求后,给客户应答处理结果。各模块之间关系如图 1 所示。

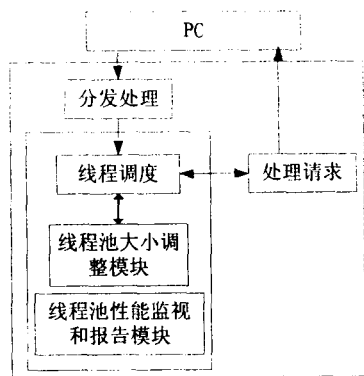


图 1 模块关系图

2.2 流程设计

根据 DHCP 工作原理和线程池技术,设计 DHCP 服务器工作流程和实现机制如下:

* 初始化线程池:创建线程池中基本线程、任务队列等数据结构。

* 创建 UDP 套接口:监听客户请求。

* 当请求数据包到来时,分发请求模块将任务给线程调度模块,继续监听。

* 线程调度模块从线程链表中找一个空闲线程来处理这一任务,空闲线程个数减一,判断空闲线程个数是否到了线程池的下限,若有,调用线程池调整模块,增加空闲

线程;若没有,等待处理新任务。

* 回收到空闲线程链表,此时空闲线程个数加一,判断空闲线程个数是否到了线程池的上限,若有,调用线程池调整模块,减少线程个数;若没有,等待处理新任务。

* 线程处理完客户请求后,给客户相应的应答,同时将应答信息进行封装,上传给 EM(M),存入数据库中租约信息的最新状态,以备重新启动 DHCP 服务或移交给其它 DHCP 服务器的时候,不至于丢失租约信息。

* 线程池调整模块是根据上下限来判断是该增加线程,还是减少线程个数。当空线的线程个数少于下限时,就要增加批量的线程,放入空闲线程链表最后面;当空线的线程个数大于上限时,就要从空闲线程链表最后面减少批量的线程。

* 分析线程池性能模块将会纪录系统吞吐量与线程个数的关系,找出最大系统吞吐量与线程个数的对应关系,并纪录此时的线程个数,称之为最大线程数。这样在线程池调整模块想要增加线程个数的时候,如果增加后的结果大于最大线程数则不增加任何线程。如图 2 所示。

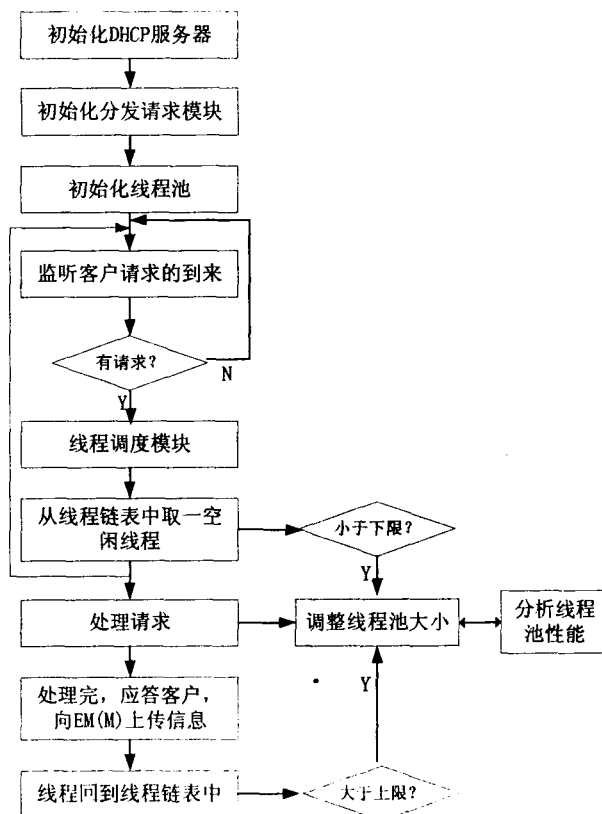


图 2 DHCP 服务器内部处理流程图

3 基于线程池技术 DHCP 服务器技术实现

3.1 线程链表设计

线程链表是用来存储空闲线程有关信息的数据结构,所有的线程在空闲线程链表里以 pthread_t 类型被组织和管理,线程的数量由系统自动调整。线程有两种状态:忙和空闲。系统初始化时,所有的线程处于空闲状态等待新任务到来的通知。当新任务到来时,线程调度模块发出信

号,所有等待此信号的线程将会接收到通知并竞争互斥量。取得互斥量的线程(称活动线程)执行该次任务。任务完成后,该线程又会成为空闲状态,再次将它放到线程链表中,以备处理新的任务。

线程链表的数据结构如下所示:

```
struct free_threads {
    pthread_mutex_t mutex; /* 互斥量 */
    pthread_cond_t t_cond; /* 条件变量 */
    pthread_t * next; /* 指向下一个线程 */
};
```

3.2 线程池大小动态调整算法思想

(1)初始化^[4]:系统初始化时,创建一定数目的线程放到线程池中。当有任务到来时,从池中取出一个线程处理这一任务,处理完后还放回线程池中,线程池中的线程只是空闲线程。

(2)调整大小^[5]:线程池有一个空闲线程个数的上限与下限,当池中的空闲线程个数低于下限时,需要创建一定数目的线程放到池中,当池中的线程个数高于上限时,需要销毁一定数目的线程。

(3)寻找最大值:线程池不能无限制增大,线程个数超过一定的数目,将会降低系统性能。这里用一个结构体变量存储系统吞吐量及所对应的线程的个数,在增加线程池中线程个数后,比较此时的线程个数和结构体变量中线程的个数的大小,如果比纪录的值大,则继续比较此时的吞吐量同结构体变量中存储的吞吐量的大小,若比存储的值

大,则更新结构体变量,若比存储的值小很多,则固定结构体变量的值,这个值就是所要找的最大线程数。以后系统中的线程个数不能超过此值。

4 总结和展望

测试结果显示,基于线程池机制 DHCP 服务器的响应速度明显加快,效率有了显著的改进。

文中给出了基于线程池机制 DHCP 的设计思想,提高了系统的效率。下一步工作要解决的问题主要是:如何实现对客户分类,并优先处理特殊客户的服务请求。

参考文献:

- [1] Droms R. Dynamic Host Configuration Protocol[S]. RFC 2131, 1997.
- [2] 李冠一,郑辉,韩维桓. 基于 WinSock 实现 Internet 协议多线程连接的关键技术[J]. 计算机应用, 2001, 21(12): 50-52.
- [3] 翟征得,李大兴. 基于线程池技术 WWW 代理服务器的设计与实现[J]. 计算机应用研究, 2004(5): 87-88.
- [4] 水超,李慧. 对象池模式的扩展与设计[J]. 计算机工程, 2004(9): 26-27.
- [5] Xu Dongping. Performance Study and Dynamic Optimization Design for Thread Pool Systems[DB/OL]. <http://www.scl.ameslab.gov/Publications/Brett/CCCTFinal-color.pdf>, 2004-12-01.

(上接第 75 页)

接口来使球员与 Soccer Server 进行通信。

2) Basic Player. Basic Player 类继承了 Player 类,拥有基本的行为代码。该类实现的技术包括:观察物体、跑向球的位置、预测球的位置、控球和截球等。

3) Advanced Player. 该类在基本动作代码的基础上,包括了一些特殊技术、动作的代码。

4) Abstract Defense Player, Abstract Midfield Player, Abstract Attack Player. 这些类分别是后卫、中场、前锋队员所对应的类的父类,要实现具体角色对应的类只需要扩充相应的代码即可。

5) Goalkeeper, Left Winger, Striker, Right Winger. 这些类分别是守门员、左边锋、中场,右边锋队员所对应的类。这是具体的角色,是具体队员的实现。

5 结束语

从目前的研究成果来看,多 Agent 系统的构造方法,主要借用了两方面的研究成果:一个是面向对象的方法;另一个是知识工程的方法。其中,面向对象的方法得到了更广泛的重视,这是因为:一方面,面向对象技术发展已比较成熟,有很多经过实践检验的方法和工具;另一方面,对象和 Agent 是两种比较一致的观察客观世界的工具,因此

从对象过渡到 Agent 是直观而且自然的。

文中从面向角色的多 Agent 结构设计的视点出发,介绍了角色的概念、角色之间的转换和合作关系,并且揭示了角色与 Agent 的结构设计相结合的优势所在,最后用一种球员类的设计方法作为理论在实践中的运用。基于角色的流程设计,从理论上讲还没有趋于完善,在实际工作中还有不少未知的方法,对于从事智能方面研究的软件工作者,尤其对从事多 Agent 研究的行业人士来说,还有巨大的发展空间。

参考文献:

- [1] 程显毅. Agent 计算[M]. 哈尔滨:黑龙江科学技术出版社, 2003.
- [2] 赵卫东,黄丽华. 面向角色的多 agent 工作流模型研究[J]. 管理科学学报, 2004, 7(2): 55-60.
- [3] 马军,闫琪,毛新军,等. 基于角色的多 Agent 系统软件设计方法[J]. 计算机工程与应用, 2004, 40(6): 118-120.
- [4] Reis L P, Lau N, Oliveira E C. Situation Based Strategic Positioning for Coordinating Team of Homogeneous Agents[EB/OL]. <http://www.ieeta.pt/robocup/archive.htm>, 2002-05.
- [5] 潘凌寒,楚威,程显毅. 基于角色的 RoboCup 足球策略[J]. 计算机工程与应用, 2004, 40(26): 66-67.