第 39 卷 第 12 期
Vol.39 No.12
计 算 机 工 程
Computer Engineering
2013 年 12 月
December 2013

• 移动互联与通信技术 • 文章编号：1000—3428(2013)12—0070—05 文献标识码：A 中图分类号：TP393

高性能网络服务器框架的研究与实现

郭庆涛[1,2]，孙强强[1]，李永攀[1]，于晓军[2]，郑 滔[3]

(1. 深圳供电局有限公司，广东 深圳 518001；2. 腾讯研究院，广东 深圳 518001；3. 南京大学软件学院，南京 210093)

摘 要：为构建面向海量数据和连接的互联网应用服务器，提出一种轻量级的高性能服务器开发框架 LHP-Svrframe。除服务器开发框架的常用模块(如网络通信管理、自定义协议开发以及进程处理模型等)外，LHP-Svrframe 特别提供针对 TCP/IP 通信协议和进程模型额外的优化设计，如对 NIC 网卡中断的负载均衡、拥塞窗口的动态调整、ACK 延时机制的改进等。与 Apache、Lighttpd、ACE 等网络服务器或开发框架的对比结果表明，使用 LHP-Svrframe 开发的应用服务器在最大连接数和吞吐量上性能可提升 4~8 倍。

关键词：海量连接；海量数据；高性能服务器框架；TCP 协议栈；拥塞窗口；Nagle 算法

Research and Implementation of High Performance Network Server Framework

GUO Qing-tao[1,2], SUN Qiang-qiang[1], LI Yong-pan[1], YU Xiao-jun[2], ZHENG Tao[3]

(1. Shenzhen Power Supply Co., Ltd., Shenzhen 518001, China;

2. Tencent Research Insitute, Shenzhen 518001, China,

3. Software Institute, Nanjing University, Nanjing 210093, China)

【Abstract】This paper proposes a brand new light-weight high performance server framework(LHP-Svrframe) for new generation server development, whose intension is for building a high performance server of massive connection and massive data. The difference from current frameworks is LHP-Svrframe focuses on TCP stack and process model and does especial optimization design, such as load balancing in NIC interrupts, dynamic adjust size of the congestion window, and optimization of the delayed ACK mechanism, et al. Experimental results in the end prove that the LHP-Svrframe performs much better than its counterparts like Apache, Lighttpd and ACE, with four times even eight times better.

【Key words】massive connection; massive data; high performance server framework; TCP protocol stack; congestion window; Nagle algorithm

DOI: 10.3969/j.issn.1000-3428.2013.12.015

1 概述

目前互联网的用户数和流量正以几何级数增长，海量的用户连接和数据对网络后台的服务器提出了严峻的挑战。随着用户的急剧增长，互联网应用或者 Web 站点开始不能及时处理用户的请求，导致用户进行长时间的等待，大大降低了服务质量。随着互联网环境的变化和硬件体系的多核进化[1]，TCP 协议中的一些实现理念开始显得陈旧，甚至限制了现有的互联网应用性能的提升。目前国内外的服务器开发框架仍未能将这些关键因素考虑入内，而服务器的性能短板却往往集中在海量数据的 I/O 效率以及海量连接的维护上。不难发现在现有的服务器开发技术之下，许多服务器在面临海量连接和海量数据的时候显得捉襟见肘，其性能仍较差，主要还是因为现有服务器开发技术或框架只保证能够开发出一个服务器，但却不能对其性能做任何担保。所以在服务器开发完毕后，开发人员后期的服务器调优工作仍然繁重。

本文提出一个轻量级的高性能服务器开发框架 LHP-Svrframe，用以构建服务于海量连接和数据的互联网应用服务器。除了一个普通的服务器开发框架应提供的模块(如基本的网络通信管理、自定义应用层协议开发以及服务器进程处理模型等)之外，LHP-Svrframe 还提供了额外的优化设计，如多核计算处理优化、TCP/IP 通信拥塞窗口优化、半同步-半异步进程处理模型的改良设计以及后端数据缓存设计等，使开发人员能够更有效率地构建一个高性能的服务器，以应对海量用户的请求，提供更优秀的海量服务性能。

基金项目：深圳供电局有限公司科技基金资助项目“电力信息系统性能评估模型的研究及应用”(090000KK51120041)

作者简介：郭庆涛(1985－)，男，硕士研究生，主研方向：高性能服务器设计，网络性能优化，分布式计算；孙强强，学士；李永攀，硕士研究生；于晓军，学士；郑 滔，教授

收稿日期：2013-06-28 **修回日期：**2013-09-18 **E-mail：**guo.qingtao@foxmail.com

2　LHP-Svrframe 整体架构设计

LHP-Svrframe 基本架构如图 1 所示。不同于 ACE 框架的是，LHP-Svrframe 没有大量的设计模式堆砌，不强调可移植性和通用性，而是专门针对互联网海量连接和数据的特点，在满足基本网络通信和业务逻辑处理的需求之上，对网卡中断信号进行负载均衡、TCP 协议栈的成块数据传输优化，以求尽可能地在高负载的海量请求下能够更高效地进行网络 I/O 传输。LHP-Svrframe 的专用性表现在：只支持 Linux 操作系统，针对特定版本的内核(Linux-2.6.35 及以上版本)和特定的网络传输协议(如 TCP/IP)，针对多核 CPU 架构等。

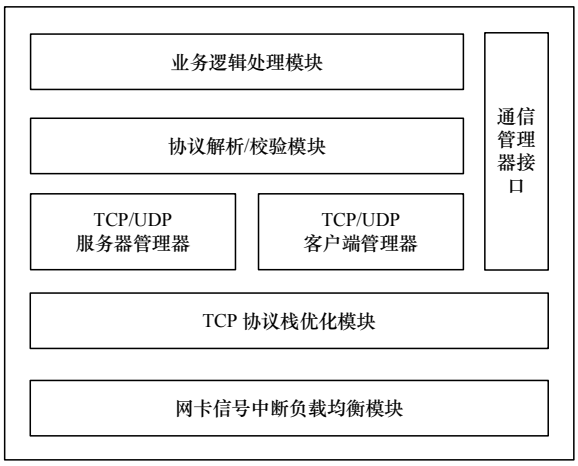

图 1　LHP-Svrframe 基本架构设计

针对报文全生命周期的各个状态阶段，LHP-Svrframe 分别具有对应的处理优化模块，从网卡收到数据向 CPU 发起中断处理信号开始，到数据传输的缓冲处理，到报文的解包或解密，到对报文信息的解析提取，再到具体的业务逻辑处理和最后对报文的响应回复，至此一个基本的报文生命周期完整地结束。据此，LHP-Svrframe 主要包含以下模块：

(1)NIC 中断负载均衡模块：该模块包含了一个针对网络报文头部的哈希算法，可以对网卡的数据接收在 CPU 多个核心上面做压力分配，从而趋避多核 CPU 上出现单核接收瓶颈的现象，利用多核架构优势来提高网卡的数据接收性能。

(2)TCP 协议栈传输优化模块：该模块是针对 TCP 由于自身的慢启动算法和拥塞控制机制[2-3]与现行网络连接场景的矛盾提供的一种解决方案，对 TCP 协议栈进行参数修改，改进其慢启动算法以更好地适配海量连接的网络环境和服务。

(3)TCP/UDP 通信管理模块：提供 TCP 和 UDP 的服务端和简单的客户端的数据读写接口，实现了以高效的 epoll 机制为核心的网络接入 I/O 模型，默认情况下将 epoll 的工作模式设置为对内核更少发出中断信号的 ET 模式[4]。

(4)自定义协议校验和解析模块：提供基本的 HTTP 协议校验和解析的接口，当然更重要的是，提供针对自定义协议进行报文头部校验和报文体解析的接口，与下层的 TCP/UDP 通信管理模块可以进行无缝的嵌入连接。目前提供基于 TVL、XML 格式的协议解析器。

(5)服务器业务逻辑处理模块：提供基于报文体的服务器业务逻辑处理接口，实现了领导者-随从者的多进程报文分发处理机制，全局无锁，并且将报文的处理和请求回复统一由单个进程来完成，减少了因要求主进程进行请求回复而导致的多一次进程切换，从而节省内核上下文切换的开销。另外，还提供了基于管道的请求分发的多进程处理模型，在该模型中，各个进程直接相对独立、分离，单个进程的崩溃不会影响到服务器主进程和其他业务处理进程的正常运行，表现出优秀的容错性和可扩展性，对于服务器扩容来说，是十分高效的解决方案。

(6)服务器通信管理模块：这个是服务器主进程的统一管理入口，该模块管理了服务器所使用的协议通信管理、报文解析、业务逻辑处理等模块的依赖注入，是整个服务器程序的启动入口点。初次之外，这个模块还需要管理整个服务器所能够维持的同时在线的最大客户数、用户态的数据缓冲区大小分配、每个客户连接拥有的数据缓冲区大小等基本信息，这些参数的大小将决定服务器系统的整体内存使用情况和服务器的稳定运行[5]，通常来说，都必须事先进行客户连接规模的估计，然后使用一些经验数据进行压测验证，最后正式上线，才能保证服务器的在线性能以及负载能力。

(7)其他模块：LHP-Svrframe 同时提供了后端数据库如 MySQL、Oracle 的读写支持 CDBUtility 等，文件系统的统一读写模块 CFileUtility 等，这些后端 I/O 模块专门针对具体的场景进行一些优化设计。

2.1　NIC 中断信号负载均衡

在单核 CPU 的时代，没有对 NIC 中断信号进行压力平摊的需求，因为所有的 NIC 中断都由唯一的 CPU 核在处理，不存在可以分摊的概念。但单核的时代已经成为过去了，多核已经成为 CPU 处理器架构的主流，这样就引发一个新的问题，通过#cat /proc/interrupts 命令可以查看到，几乎所有的 NIC 中断都压在了 CPU0 上面，与此同时 top 命令也可以显示出 CPU0 和其他几个核明显存在繁忙和闲置状态的强烈对比，CPU0 的占用率几乎达到 100%，而其他的核大多处于 20%~30%的使用状态，即是说 CPU0 的中断处理能力成为了性能瓶颈，而这是可以通过平摊压力去得到性能改进的。

如图 2 所示，LHP-Svrframe 采用 Linux-2.6.35 内核版本所发布的新特性 RPS(Receive Pakcet Steering)和 RFS (Receive Flow Steering)对网卡中断压力在多个 CPU 核之间分摊。这是网络性能优化技术中 Generic Receive Offload 的一种，通过对网卡收到的数据报文进行有效的 CPU 中断信号分流，达到对 CPU 进行网络负载分摊的目的。其工作原理如下：通过使用一定的哈希算法，对 TCP/UDP 协议数据

报文的包头四元组信息(比如 IP 和端口等基本信息)建立与具体某个 CPU 核的映射关系，从而使数据报文不仅能够在多个 CPU 核之间进行分流，并且保证属于同一数据流的数据报文每次都会被分配到同一个 CPU 核处理，这样即可充分发挥 CPU Cache 的作用，减少 Cacheline-pingpong 现象，提高 Cache 命中率，从而从 NIC、CPU 层面上降低网络负载，提高网络数据接收效率和服务器的吞吐量。

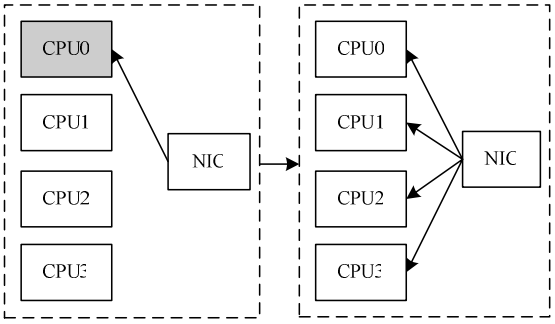

图 2 网卡中断信号负载均衡示意图

根据实验数据统计，使用 RPS 对服务器的网络数据接收性能可以提升约 3 倍。tg3 驱动的 NIC 性能由 90 000 连接/秒提升到 285 000 连接/秒，而 e1000 驱动的 NIC 性能由 90 000 连接/秒提升到 292 000 连接/秒，其他驱动 NIC 也得到类似的测试结果。

2.2 TCP 传输优化模块

TCP 目前实现的是一种闭环的流量控制方案，即发送方会根据一些预定的隐含信号去动态地调整拥塞窗口的大小，以适应不同负载的网络环境。其所采用的慢启动算法、Delayed ACK 算法[6]会极大地降低 TCP 链路上在 buld data 场景下的传输效率。

2.2.1 拥塞窗口的优化与服务器吞吐量

首先，尝试探索 TCP 拥塞窗口大小对吞吐量的影响，定义如下变量：W 为 TCP 窗口大小(Byte)；R 为一个给定的 TCP 连接在 TCP 发送方的数据速率(bit/s)；D 为在一个给定的 TCP 连接上 TCP 发送方与接收方之间传播延时(s)。则归一化吞吐率 S 可表示为：

$$S=\begin{cases}1 & W>RD/4\\ \dfrac{4W}{RD} & W<RD/4\end{cases}$$

首先，简单起见，TCP 报文端中的开销比特将会被忽略。设想一个 TCP 发送端在某个链路上向接收端传输一系列的字节。第一个字节到达接收端要花 D s 而 ack 确认返回还要花费 D s。在这段时间里，发送端如果不受到限制就可以传输 $2RD$ bit，或者 $RD/4$ Byte。然而实际上，发送端是要受限的，在收到 ack 确认之前它不能传输超过窗口大小及 W Byte 的数据。因此，如果 $W>RD/4$，在这条链路上就可以取得最大可能的吞吐率。如果 $W<RD/4$，可取得的最大归一吞吐率就仅仅是 W 与 $RD/4$ 的比率。

示例：在一条 1 Mb/s 的链路上，通常来说传播延时大约为 60 ms，那么根据上文公式，当归一吞吐率 S 达到最优时，拥塞窗口 W 的大小应该达到约 2 013 265 Byte。而在常用的 Linux-2.6 系统中，默认的拥塞窗口大小往往只有 87 380 Byte。来自 Google 的一项 initcwnd 实验证明了本文的观点，通过增大初始的拥塞窗口大小，TCP 通信报文的延时可以被显著地缩短。

LHP-Svrframe 对系统默认的 TCP 拥塞窗口大小进行优化，这是内嵌在框架中对 TCP 网络传输优化最原生的支持，而不需要开发人员等到服务器性能出现了问题再进行 TCP 传输的优化工作，并且这降低了对开发人员拥有较为深刻的 TCP 协议背景知识的要求，使更多的开发人员能够轻易、快速地进行高性能、高吞吐量的服务器开发。然而，对 TCP 拥塞窗口的调整必然涉及到协议在网络资源的分配上会产生公平性问题，从传统的 TCP 和 UDP 协议实现上来说，采取了拥塞控制的 TCP 在网络资源的争抢上显然会比无拥塞控制的 UDP 弱得多，因此，从整体网络环境上来说，LHP-Svrframe 对资源使用的公平性的影响不大。而在同样采用 TCP 的不同服务源端的对比来说，LHP-Svrframe 相比 TCP 默认实现的性能优势其实也就来源与对资源的抢占策略，而与目前业界实现的其他拥塞算法如 Vegas、Reno 等相比，LHP-Svrframe 在公平性上显得可控得多，毕竟在 LHP-Svrframe 的策略中，拥塞窗口并不是毫无限制地增长。另外，上世纪 80 年代所设计的 TCP 协议是适应当时较为落后的网络环境，在网络带宽日益增长的当下，传统的 TCP 协议实现已经凸显出它的性能瓶颈，LHP-Svrframe 对拥塞窗口的优化是适合现在的网络环境特点的。

2.2.2 Nagle 算法与响应延时的优化

Delayed ACK 是另一种降低网络通信报文数量而提高网络传输效率的机制。

在特定的情景下，Nagle 算法与 Delayed ACK 机制会导致类似死锁一样的等待，详细的过程描述如下：(1)由于 Nagle 算法的启用，链路上只允许一个未经 ack 回复确认的报文存在，因此当前的停留在缓冲区的数据报文不会被立即发送；(2)然而 Delayed ACK 机制在服务端未发送响应数据的情况下，也不会立即放松 ack 确认报文；(3)但此时的服务器进程无法为客户端产生响应数据，因为当前请求的报文仍未接收完整；(4)而 Nagle 算法不允许报文的最后一份数据立即向服务器端发送，因为前一份数据报文的 ack 还没达到[7]……。

以上的过程就是类似死锁的无限循环等待，当然 TCP 协议栈现有的设计并不会让这个死锁真的存在，因为在步骤(2)，Delayed ACK 并不是一定得等到服务器进程产生对客户端的响应数据，而是采用了定时器的设计，让这个死锁在一定时间内失效，由 TCP 协议的具体实现可以知道，这个定时器的时间最多可达到 500 ms，而一般是 200 ms[8]。如图 3 所示，这种死锁会经常能够引起响应延时。

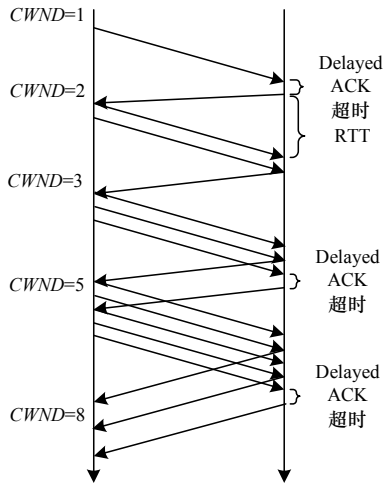

图 3　Delayed ACK 响应延时案例

LHP-Svrframe 的优化方案之一是，对内核 TCP 协议栈的实现做了一个小小的调整，将定时器的时间缩短，以内核补丁的形式在框架内提供。根据用户实际的通信场景、报文大小和带宽的具体情况，开发人员可以对框架安装前进行配置，合理地调整定时器的时间至 10 ms 或 20 ms，那么由于 Nagle 算法和 Delayed ACK 导致的性能损耗将被降至原来的 1/10~1/20。

如果开发人员不想对内核进行改动，则 LHP-Svrframe 提供另外一种优化方案：对每个接收到的满负荷大小(达到 MTU)的报文，都发送一个无意义的比特进行响应。这样的话，同样可以解除前面所描述的死锁，做到毫无延时的 ack 回复，但缺点是链路上的无意义的 ack 回复报文会增多，在一定程度上造成了一些带宽浪费，不过就目前的实验数据来看(详见本文第 3 节中的实验结果分析)，在 1 Mb/s 带宽的链路下，不会对性能造成显著的影响。

经过 LHP-Svframe 对 Nagle 算法和 Delayed ACK 机制的优化，所构建的高性能服务器在客户端响应延时上将会有显著的降低，这对客户端的交互性能有着很重要的意义。

总的来说，LHP-Svrframe 的 TCP 传输优化模块的优化工作主要集中在接收端，即服务器端，优化策略类型包括了 Large Receive Offload[9](如降低 ack 延时的 overhead)和吞吐量提升等。当然，同时也对客户端的 TCP 传输优化提出了建议，如在特定的场景下可以选择禁用 Nagle 算法以提高应用的交互性、实时性等。具体的实现形式，LHP-Svrframe 是 Linux 内核补丁和框架 API 提供给服务器开发人员使用，内核补丁需要预先编译后，以内核模块的形式加载入现行 Linux 内核当中去。

3　实验与结果分析

本文将基于 LHP-Svrframe 框架的服务器与现有的 Web 服务器进行整体的性能对比，性能指标采用常用的最大并发客户端连接数、整体吞吐量和平均客户端响应延时为主要对比的数值。对比的 Web 服务器有在 Linux 平台上非常流行的静态前端网络服务器 Apache Httpd、Lighttpd 和 Litespeed，实验方式为通过在客户端和服务器端传输不同大小的文件，并观察不同条件之下的服务器能够达到的最好性能，测试工具为 LoadRunner-9.5。被测 3 个 Web 服务器的后端都被放置了一个相同的 PHP 脚本，其逻辑很简单，即单纯的数据接收，没有任何其他的逻辑处理，同时也不需要为这些接收到的数据做存储的准备。所有的性能实验都尝试了不同的并发级别，从 10、100、1 000、2 000、5 000 甚至到 10 000，实验停止的基准线是服务器的丢包率开始达到 1%。

图 4 为实验报文大小为 1 KB 传输时，各个服务器与 LHP-Svrframe 的性能对比。其中，服务器吞吐量指并发请求个数。

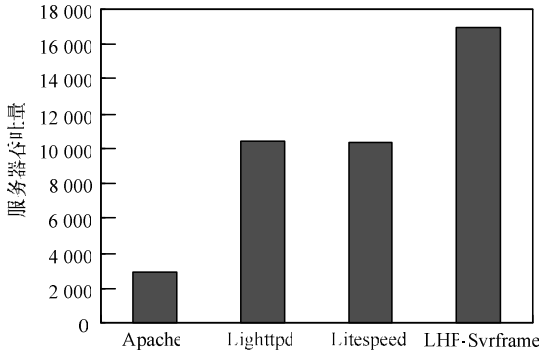

图 4　1 KB 报文的服务器吞吐量对比

其中，Apache、Lighttpd 和 Litespeed 的最佳吞吐量都是在并发级别为 1 000 时取得的，可以看到，Apache 的吞吐量表现最差，约为 2 909；而 Lighttpd 和 Litespeed 的吞吐量基本持平，都差不多在 10 500 的水平左右；LHP-Svrframe 的最佳吞吐量是在并发级别为 5 000 的时候取得，具体值为 17 000，是 Apache 的 5 倍、Lighttpd 和 Litspeed 的约 1.6 倍。

接着又继续增大传输报文大小至 100 KB，然后尝试了不同的并发级别，从 10、100、1000、2 000 到 5 000，实验停止的基准线是服务器的丢包率开始达到 1%。具体实验结果数值如图 5 所示。

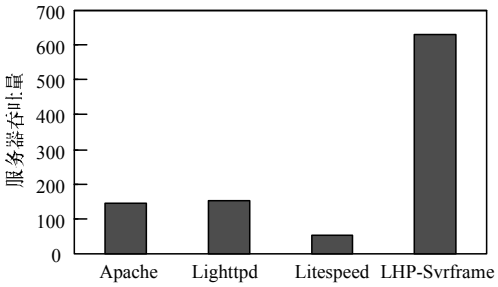

图 5　100 KB 报文的服务器吞吐量对比

在将传输报文的大小增大为原来的 100 倍之后，服务器的吞吐量都出现了显著的变化，前 3 个对比 Web 服务器的最佳吞吐量分别为 146、151、53，而 LHP-Svrframe 的最佳吞吐量为 629，是前三者中最好者的 4.2 倍，最差者的

11.9 倍，这说明在大块数据的传输时，LHP-Svrframe 的网络 I/O 优化机制出现了更加明显的优势，因为随着传输的数据越大，每次报文被 TCP 协议栈进行分组传输的次数也就越多，从而由于拥塞窗口大小、Delayed ACK 而导致的延时的差距也就愈发明显，所以经过了 TCP 协议栈、NIC 中断压力平摊等策略优化的 LHP-Svrframe 的性能优势就更容易体现出来。

不可避免的，LHP-Svrframe 框架将会面临与 ACE 框架的性能对比，因为如果 LHP-Svrframe 框架并没有比 ACE 有更加好的性能[10]，那么开发人员就没有理由放弃业界成熟的服务器开发框架而转向一个全新的、并且并不知名的 LHP-Svrframe 框架。在本次实验中，笔者编写了一个基于 ACE 的简单的服务器，不使用磁盘 I/O，简化了逻辑处理部分，服务器端只对客户端回复“OK”，也没有任何的自定义协议的解析和字段提取。

同样地，对之前的 BRSvr 做了一些业务逻辑的简化，省略了磁盘 I/O 的部分逻辑，然后进行大规模客户的并发压力试验，整个实验的逻辑具体如下：

(1)在同等配置下的 4 台测试客户端部署 LoadRunner 测试环境。

(2)在服务器上部署好性能指标的监测脚本，并首先通过 tcpdump 抓包试验确保每台测试机与服务器之间发包的正确性和完整性。

(3)使用 LoadRunner 的 Generator 定时从 4 台测试机发起对服务器的并发请求，发送的请求总数为 6 000 000 个。

(4)重复试验，针对不同并发级别[11](500、1 000、2 000、4 000、6 000、8 000、10 000、12 000、15 000)，每次试验停止的基准线是丢包率开始达到 1%。

实验结束后获得的关于 ACE 和 LHP-Svrframe 所构建的服务器的最佳性能参数比较如表 1 所示。

表 1 ACE 与 LHP-Svrframe 的性能对比

服务器	并发连接数	吞吐量/(req·s^{-1})	响应延时/ms	丢包率/(%)	CPU 占用率/(%)	MEM
ACE	3 000	4 500	200	1.25	28	879 MB
LHP-Svrframe	12 000	35 000	120	1.00	80	2 GB

需要说明的是，ACE 在并发级别 2 000 时发包成功率为 100%，而一旦到了 4 000 的并发，开始迅速地丢包，成功率只有约 90%，因此，特别地修改测试的并发级别，重新寻找最佳性能的临界点，最终锁定在 3 000。

可以看出，LHP-Svrframe 的优势明显，不管在客户端连接数的维持上还是最终的吞吐量，都分别是 ACE 的 4 倍和 7.8 倍，这说明 LHP-Svrframe 所采用的网络 I/O 优化机制以及 epoll 的高效的 ET 工作模式和对多核计算优势的充分利用是对服务器性能有着决定性的作用的。从 CPU 占用率一列来看，ACE 默认单进程单线程的工作模式在多核的处理器上表现仍有较大提升空间，因为单线程毕竟只能消耗一个处理器核心的计算资源[12]。

4 结束语

针对目前互联网环境、硬件体系架构的变迁、日益增长的带宽和 TCP 协议栈陈旧设计理念的矛盾，本文提出了一个轻量级的高性能服务器计算开发框架，从专用性、特殊性的角度，对特有的网络环境和硬件进行了不同程度的优化机制的探索，尝试将网络 I/O 效率的提升、多核计算优势的利用纳入一个服务器开发框架当中去，并且进行了大量的设计思考和实验，最终为网络服务器提供一个高性能的解决方案，服务于海量的用户连接和数据。关于 TCP 协议栈仍有那些原有的设计以及由此导致的性能损耗(如糊涂窗口综合症)，这是下一步优化和改进的方向。此外，如何设计更优的任务调度算法，更有效地降低进程除正常业务处理逻辑计算的开销，充分利用缓存等，都是 LHP-Svrframe 需要深入研究的工作。

参考文献

[1] Akhter S, Roberts J. Multi-core Programming: Increasing Performance Through Software Multi-threading[M]. Hillsboro, USA: Intel Press, 2006.

[2] Afifi H, Elloumi O, Rubino G. A Dynamic Delayed Acknowledgment Mechanism to Improve TCP Performance for Asymmetric Links[C]//Proceedings of IEEE Symposium on Computers and Communications. [S. l.]: IEEE Press, 1998: 34-39.

[3] Dukkipati N, Refice T, Herbert T, et al. An Argument for Increasing TCP's Initial Congestion Window[J]. ACM SIGCOMM Computer Communication Review, 2010, 40(3): 10-17.

[4] Bhattacharya S, Tran J, Sullivan M, et al. Linux AIO Performance and Robustness for Enterprise Workloads[C]// Proceedings of the 6th Annual Ottawa Linux Symposium. [S. l.]: IEEE Press, 2004: 65-79.

[5] Gammo L, Brecht T, Shukla A. Comparing and Evaluating Epoll, Select, and Poll Event Mechanisms[C]//Proceedings of the 6th Annual Ottawa Linux Symposium. [S. l.]: IEEE Press, 2004: 217-227.

[6] Wright G R, Stevens W R. TCP/IP Illustrated Volume1: The Protocols[M]. [S. l.]: Addison-Wesley, 1993.

[7] Wright G R, Stevens W R. TCP/IP Illustrated Volume2: The Implementation[M]. [S. l.]: Addison-Wesley, 1995.

(下转第 78 页)