分类号	TP393	密级	公开
UDC	004.7	学位论文编号	D-10617-308-(2016)-01136

重庆邮电大学硕士学位论文

中文题目	基于众核平台的 Web 流量产生系统
英文题目	**A Web Traffic Generation System Based on Many-core Platform**
学　　号	**S130101161**
姓　　名	唐小军
学位类别	工学硕士
学科专业	信息与通信工程
指导教师	唐红 教授
完成日期	**2016** 年 **4** 月 **8** 日

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含他人已经发表或撰写过的研究成果，也不包含为获得 重庆邮电大学 或其他单位的学位或证书而使用过的材料。与我一同工作的人员对本文研究做出的贡献均已在论文中作了明确的说明并致以谢意。

作者签名：　　日期：　　年　月　日

学位论文版权使用授权书

本人完全了解 重庆邮电大学 有权保留、使用学位论文纸质版和电子版的规定，即学校有权向国家有关部门或机构送交论文，允许论文被查阅和借阅等。本人授权 重庆邮电大学 可以公布本学位论文的全部或部分内容，可编入有关数据库或信息系统进行检索、分析或评价，可以采用影印、缩印、扫描或拷贝等复制手段保存、汇编本学位论文。

（注：保密的学位论文在解密后适用本授权书。）

作者签名：　　导师签名：

日期：　　年　月　日　　日期：　　年　月　日

摘要

流量产生器是为实验网络性能、协议、安全等方面的测试及验证提供网络背景流量的重要工具。随着未来网络技术研究的不断深入，迫切需要满足易扩展、高性能、低成本的新型流量发生器。

目前，常用的流量产生器主要有两种类型，一种是基于硬件实现的专用设备，一种是运行于通用计算机上的软件工具。前者一般是为某种特定的测试场景而专门设计的，灵活性和可扩展性较差，且价格昂贵。后者有较好的灵活性，且成本较低，但产生的流量速率不高，难以满足未来网络研究中的测试需求。

针对上述问题，本文在分析现有流量产生器的基础上，重点对 Web 流量的产生进行了研究，设计并实现了一个基于众核平台的 Web 流量产生系统。本文采用分层设计方法，提出了 Web 流量产生系统的软件体系结构，包含控制层、虚拟用户层和流量产生层，提高了系统的可扩展性。在分析 Web 用户行为模拟方法的基础上，本文提出了一种基于三次样条插值算法的用户控制方法。该方法使系统能够根据不同场景下网民上网时段数据，生成大时间尺度下符合实际网络特征的背景流量。为了解决现有软件工具产生流量速率不高的问题，本文提出了一种从 Linux 内核参数优化、基于事件驱动的并发和基于众核平台的系统并行化三个方面提高系统并发性能的方法。该方法使系统能够同时模拟大量的虚拟用户，从而产生较大的 Web 流量。系统测试结果表明，该系统能够同时模拟 5 万个用户访问 Web 服务器，并且系统的并发性、稳定性以及产生流量的真实性等均达到设计目标。

关键词：流量产生器，众核平台，Web 流量，并发性能，并行化设计

Abstract

Traffic generator is the major tool which provides network background traffic to test and verify the performance, protocol, security of the experimental network. The new traffic generator which has features like high-performance, flexibility, low cost is urgently required with the development of the future network research.

At present, there are mainly two types of traffic generators. One is a special equipment based on hardware, the other is a software tool which is running on general computer. The former is generally designed for a particular test scenario which has lack of flexibility and extensibility and more expensive. The latter has better flexibility and lower cost, but the produced traffic rate is not high so that it is difficult to satisfy the requirement of the future network research.

For the above problems, this thesis focuses on the generation of Web traffic on the basis of analyzing the current traffic generators, designs and builds a web traffic generation system based on many-core platform. In this thesis, the software architecture of Web traffic generation system is proposed by using the hierarchical design methodology that includes the control layer, virtual user layer and traffic generation layer, which improves the scalability of the system. Based on the analysis of Web user behavior simulation method, this thesis presents a large scale user control method using the three spline interpolation algorithm. The method enables the system to generate the background traffic with the real network characteristics in large time scale according to Internet users access time data in different scenarios. In order to solve the problem that the traffic rate of existing software tools is not high, this thesis presents a method to enhance the concurrent performance of traffic generation system from the three aspects of Linux kernel parameter optimization, concurrency based on the event-driven and system parallelization based on the many-core platform. This method enables the system to simulate a large number of virtual users at the same time, and generates larger Web traffic. Experiment results show that the system can simulate 50 thousand users to access the Web server at the same time. And the concurrency, stability and traffic authenticity of the system have reached the design goal.

Keywords: Traffic Generator, Many-core Platform, Web Traffic, Concurrency Performance, Parallel Design

目录

图录

表录

第 1 章 绪论

1.1 研究背景及意义

流量产生器是一种产生网络流量的工具，主要用于为网络应用软件、网络设备及实验网络性能、安全、协议等方面的测试与验证提供背景流量[1]。随着未来网络技术研究的不断深入，网络体系架构、软件定义网络、信息中心网络、未来移动通信网络、新型光传输网络、路由器与交换机设计、网络协议设计、网络安全等领域的研究成果不断涌现。这些面向未来网络的新成果在实际部署应用之前，需要在实验网中对其进行测试和验证。因此，需要为实验网提供大规模且接近于实际网络的背景流量。

目前，常用的流量产生器主要有两种类型，一种是基于硬件实现的专用设备，一种是运行于通用计算机上的软件工具。前者一般是为某种特定的测试场景而专门设计的，灵活性和可扩展性较差，且价格昂贵，难以应对多变的实验网环境。后者有较好的灵活性，且成本较低，但产生的流量速率不高，难以满足未来网络研究中的测试需求。

然而，随着网络设备制造技术的不断更新，众核的出现无疑给流量产生器的实现带来了新的生机。目前的众核处理器可集成数十个至数百个核，每个核是一个执行单元，可以单独对一个子任务进行处理[2]。因此，众核平台的强大处理能力为产生大流量的流量产生器的实现提供了有力的硬件基础。在软件方面，众核平台提供了一个真正并行编程的环境。这为流量产生器的并行化设计提供了有利的软件条件。

表 1.1 常见的业务流量比例

流量类型	所在比例
Video Streaming	26%~36%
P2P	16%~20%
Web	15%~18%
Downloading	3%~26%
Other	6%~26%

随着互联网的快速发展，新的业务不断涌现，如 Video Streaming、P2P 等。虽然这些新业务对传统 Web 业务的增长有所抑制，但最近的研究表明 Web 业务仍占据 15%~18%的流量[3]，如表 1.1 所示。因此开发一个适合实验网的 Web 流量产生器具有非常重要的意义。

1.2 研究现状

1.2.1 Web 流量产生器研究现状

1. Web 流量产生方法研究现状

文献[4]指出，Web流量产生器产生的Web负载有两个要求：真实性和代表性。其中，真实性是指一系列能够被真实环境中Web服务器接收的请求；代表性是指产生的负载必须要有与实际Web负载相同的特点，如请求的数量、平均思考时间、会话的长度等。衡量这两个要求满足的程度主要取决于Web流量产生器所采用的流量产生方法。当前生成Web业务流量的方法主要有以下三种：

(1) 基于流量回放的方法[5]

该方法首先对某个实际网络中的数据包进行捕获，并将结果记录在日志文件中。当需要在一个新的网络环境中生成业务流量时，可以该日志文件为基础，通过流量回放的方式将捕获地数据包注入到新的网络中，实现业务流量的生成。该方法生成的网络流量是真实网络环境中的业务流量。但是需要原始流量数据，且这些业务流量只能反映某个时段（流量捕获时期），某个或者某些已有的特定网络的业务流量，因此通用较差。

(2) 基于流量模型的方法[6]

该方法通过对业务流量自身的特征进行数学建模，然后相应的流量产生器以该数学模型为基础，通过构造数据包生成总体的业务流量。这种方法的优点是可以针对不同的网络环境，通过调节模型中的参数，如数据包发送速率，以适应不同的网络环境和需求。但是流量模型通常比较复杂，只能反映网络流量的宏观特征。

(3) 基于用户行为的方法[7-9]

该方法从形成网络流量特征的源头——用户访问行为出发，对单个用户的访问行为进行建模，主要有基于 ON/OFF 的用户模型和基于马尔可夫链的用户模型。然后通过模拟用户行为产生流量。这种方法可避免直接对网络流量进行建模，具有较高的灵活性。但是与前两种方法相比，该方法需要额外模拟大量的虚拟用户，增加了流量产生器实现的复杂性，且流量的产生能力较差。

2. Web 流量产生器的实现现状

目前，流量产生器主要基于硬件平台和软件平台进行实现。其中，基于硬件平台实现的流量产生器一般具有强大的流量生成能力和较好的性能。这类流量产生器主要有Spirent Avalanche [10]和Ixia IxLoad[11]。其中，Spirent Avalanche是由思博伦（Spirent）公司推出的测试产品，能够实现第2到3层的流量产生和第4到7层的用户仿真，支持多种网络协议，可用于Web应用、网络性能、网络安全等方面的测试；Ixia IxLoad是由易测（Ixia）公司推出的测试产品，可对防火墙、服务器、负载均衡器和安全设备等进行性能测试。但是这类设备开发难度大，灵活性和可扩展性较差，且价格昂贵，难以应对多变的实验网环境。

与基于硬件平台实现的流量产生器相比，基于软件实现的流量产生器具有开发难度低、部署简单、易推广、跨平台等特点。常见的基于软件实现的Web流量产生器有：

(1) Monkey[5]是一个基于流量回放的流量产生器，包含 Monkey See 和 Monkey Do 两个部分，通过 Monkey See 从 TCP 层捕获流量轨迹，然后由 Monkey Do 根据流量轨迹生成流量。

(2) Webstone[12]、ProWGen[13]、GlobeTraff[14]都是基于流量模型的流量产生器。其中 Webstone 是 HTTP 服务器基准测试中较早的流量产生工具；ProWGen 用于为网络代理缓存仿真评估提供流量；GlobeTraff 是一款较新的流量产生器，可产生多种业务流量，如 Web、P2P 和 Video，用于为未来网络体系结构的评估提供流量。

(3) SURGE[15]、Geist[16]都是基于用户行为的流量产生器。其中，SURGE 由波士顿大学的 Barford 和 Crovella 开发，用于分析服务器和网络的延迟问题。Geist 是一款用于 Web 服务器和电子商务前端服务器测试的流量产生工具，支持 SSL 的握手以及加解密。但是这类流量产生器产生地流量速率不高，仅适合小范围的低速环境中使用，难以满足实验网测试的要求。

1.2.2 众核处理器的发展及应用现状

目前，众核技术已经成为最受关注地话题和研究方向。“众核”一词来源于英文“Many-Core”。一般认为，众核处理器比多核处理器具有更多的处理器核心。在业界，人们普遍将核心数大于等于 8 的处理器称为众核处理器[17]。对于众核技术的研究主要体现在众核处理器的体系结构研究和众核软件开发方法两个方面。

在众核处理器的体系结构研究方面，国外主要处理器厂商，包括 IBM、Intel、AMD、SUN、Tilera 都已纷纷转向众核 CPU 的研发，通过众核 CPU 进行并行计算来提高计算性能。Tilera 公司推出全球首款核心数量多达 100 个的微处理器 Tile-Gx100，同时问世的还有 64 核心(Tile-Gx64)、36 核心(Tile-Gx36)、16 核心(Tile-Gx16)等版本。

随着众核处理器的发展，软件开放也由原来的并发编程模式迈向了并行编程的新高度[17-18]。同时，基于众核处理器上的网络应用开发也进入了新的时代。如今在高校的科研中也掀起了一股热潮。例如西安电子科技大学研究地网络安全测试设备[19]、H.264 多线程并行编码[20]、浙江大学研究地 3G 服务器视频转码软件设计[21]、西安工程大学研究地 NetFlow 的 P2P 协议识别与检测系统[22]等，充分发挥了众核处理器在网络应用上的优势。

综上所述，目前 Web 流量产生器存在以下问题：

(1) 没有考虑大时间尺度下网络流量随时间变化的规律。现有 Web 流量产生器主要是为了满足单个网络设备如 Web 服务器的测试，生成的流量在大时间尺度下（如一个小时或一天）是相对平稳的。而实际的网络流量会随人们的作息规律而变化，呈现出白天流量大，晚间流量小的特征。

(2) 难以直接在实验网中部署。对于基于硬件实现的 Web 流量产生器虽然性能好且能够产生较大的流量，但其可扩展性差，难以与实验网融合。而基于软件实现的 Web 流量产生器由于产生流量的速率较低，如果直接应用到实验网，部署量大，维护成本较高。

(3) 基于众核平台的网络应用开发已成为一种新的潮流，而目前的 Web 流量产生器仍局限在传统平台。

1.3 论文主要研究工作及贡献

1.3.1 研究目标

依托国家重点基础研究发展计划（973 计划）项目课题：“未来互联网性能评估与实验验证（2012CB315806）”，研究、设计并实现基于众核平台的 Web 流量产生系统。该系统利用 Tilera 众核平台的高并行处理能力，采用基于事件驱动的并发策略，通过模拟 Web 用户浏览网页的行为，向网络中已有的 Web 服务器发送 HTTP 请求，从而“诱导”Web 服务器返回响应数据，为实验网提供所需的 Web 背景流量。

1.3.2 主要工作及贡献

论文的主要工作及贡献包括以下几个方面：

(1) 系统需求分析：通过对现有 Web 流量产生器研究现状进行分析，并结合项目的实际需求，明确 Web 流量产生系统的功能需求和性能需求。

(2) 系统总体设计：按照系统需求分析，将 Web 流量产生系统分为负责系统界面展示、逻辑业务处理的前端子系统和主要负责 Web 用户模拟、Web 流量产生的后端子系统。其中，后端子系统又可分为：控制层、虚拟用户层和流量产生层。

(3) 关键技术研究：首先对影响 Web 背景流量真实性的关键技术——Web 用户模拟方法进行研究，确定 Web 用户的行为模拟方法和控制方法；然后对提高系统并发性能的方法进行研究，通过优化内核参数、采用基于事件驱动的并发策略等方式来提高系统的并发能力，并对内核参数优化和基于事件驱动的并发进行了分析。

(4) Web 流量产生系统的设计与实现：根据系统的总体设计，首先对系统进行流程分析并对其进行基于 Tilera 众核平台的并行化设计，然后分别对系统的控制层、虚拟用户层和流量产生层进行详细设计和实现。

(5) 系统测试：通过搭建测试环境，对实现后的 Web 流量产生系统进行了多方面的测试，其中测试包括功能测试、性能测试、并发性测试、稳定性测试等，并对测试进行分析，充分验证了该 Web 流量产生系统的有效性。

1.3.3 论文组织结构

论文内容共分为 7 章，论文的整体结构和章节安排如下：

第 1 章，绪论。本章首先介绍了论文选题的研究背景及意义，然后分析了 Web 流量产生器以及众核处理的发展和研究现状，最后阐述了论文的研究目标、主要工作及贡献和组织结构。

第 2 章，系统需求分析与总体设计。本章对 Web 流量产生系统的功能和性能需求做出分析，并在此基础上提出了系统的总体设计，包括系统软件结构设计和功能模块设计。

第 3 章，关键技术研究。本章对实现 Web 流量产生系统所涉及到的关键技术进行了研究，包括 Web 用户模拟方法研究和提高系统并发性能的方法研究。

第 4 章，基于 Tilera 众核平台的系统并行化设计。本章首先对 Tilera 众核平台的并行化编程技术进行了介绍，然后从层次角度对系统的运行流程进行了分析，最后对系统进行并行化设计，包括任务分解、任务映射、任务间通信方案以及负载均衡方案的选择。

第 5 章，系统详细设计与实现。本章根据第 2 章中提出的系统总体设计分别对系统的控制层、虚拟用户层和流量产生层进行详细设计与实现。

第 6 章，系统测试与结果分析。本章通过搭建实验环境，对系统的功能和性能分别进行测试，并对测试结果进行分析。

第 7 章，总结与展望。本章首先对论文所做的研究工作进行总结，然后对未来可以开展的研究工作方向进行了分析与展望。

第 2 章 系统需求分析与总体设计

2.1 设计目标

利用 Tilera 众核平台强大的并行处理能力，结合 Web 用户行为模型，设计并实现一款能够为实验网提供 Web 业务流量的 Web 流量产生系统。通过该系统，实验者可以申请进行实验所需的虚拟 Web 用户资源，并对虚拟用户行为进行相关配置，然后由 Web 流量产生系统生成实验所需的 Web 背景流量。因此该系统应具备以下特点：

(1) 能够产生接近真实的 Web 流量

为实验网提供真实的网络流量是保障实验网络环境下相关研究成果可直接实际运用基础。如果实验网中的流量不够真实，将直接影响到研究成果测试与验证的准确性。所以系统应具备产生真实 Web 流量的能力。Web 流量的真实性主要体现在两个方面：1) 产生的流量应满足实际网络流量的自相似性特征；2) 产生的流量能够较准确的描述大时间尺度下实际网络流量的变化趋势。

(2) 支持大量 Web 用户的模拟

系统的 Web 用户模拟能力直接关系到系统产生流量的大小。而在实验网络的应用场景下，Web 背景流量的大小又直接制约着实验网络的规模。因此系统应当支持上万数量级的 Web 用户模拟，以满足实验网络环境下相关研究的流量需求。

(3) 支持多个实验者同时使用

在实验网络环境中，往往存在多个实验者同时需要进行实验，并且他们所需的背景流量大多不同。因此 Web 流量产生系统需支持多实验者的同时使用，以满足不同实验者的不同需求。

2.2 系统需求分析

根据系统的设计目标，下面将从系统的功能需求和性能需求两方面对整个系统进行需求分析。

2.1.1 功能需求分析

根据系统的目标及特点，Web 流量产生系统的功能可分为两大块：前端子系统功能和后端子系统功能，具体的功能需求如下图 2.1 所示。

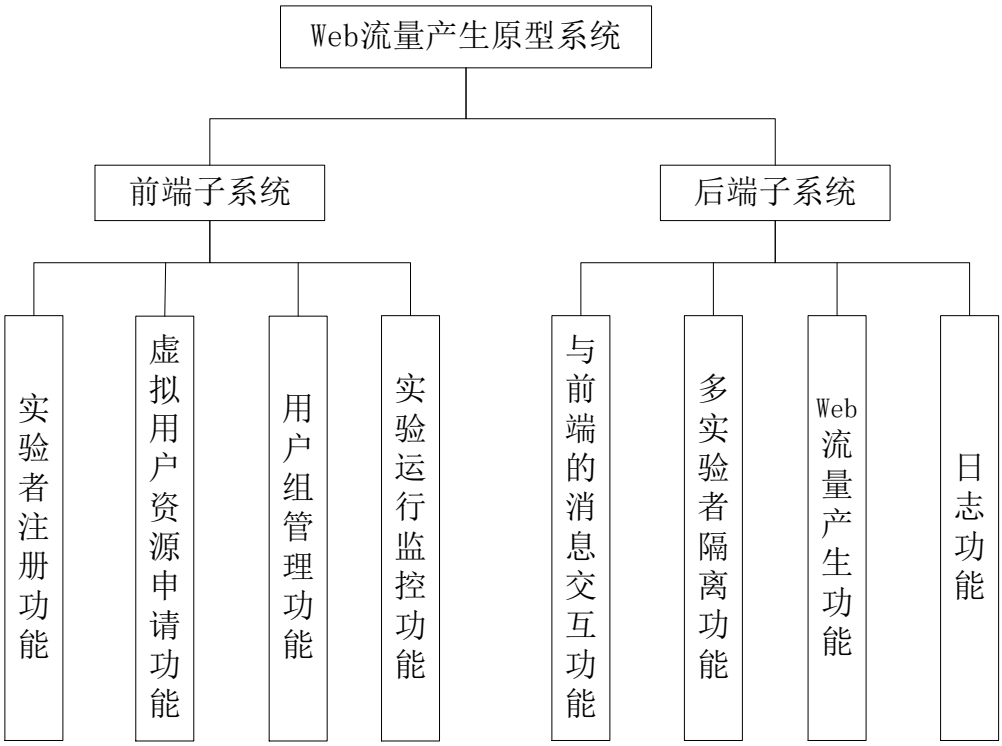


图 2.1 系统功能需求图

1. 前端子系统功能需求

(1) 实验者注册功能

新的实验者只有通过注册并在实验者数据集添加有关的描述信息后才能登录系统进行实验。这样能够有效地实现对不同的实验者的操作及数据进行隔离处理。

(2) 虚拟用户资源申请功能

对于不同的实验者而言，由于实验内容的差异，需要的虚拟用户资源是不同的。为了充分利用后端子系统能够支持的虚拟用户资源，实验者可以根据实际实验的需要以及系统当前虚拟用户资源的限制情况申请合适的用户资源。

(3) 用户组管理功能

对于单个实验者而言，在不同的实验场景通常需要不同的用户资源配置（如虚拟用户数、用户行为参数等）。因此，系统引入用户组管理功能。实验者可以将

申请的虚拟用户资源根据实验需要划分为不同用户组并进行配置。这样在再次实验时只需选择合适的用户组即可。

(4) 实验运行监控功能

该功能通过对从后端子系统推送过来的访问日志信息进行分析，提取出对应的模拟状态信息（如背景流量的变化等），并实时动态地展现给实验者。这样，实验者可以通过观察这些状态信息来调整模拟参数。

2. 后端子系统功能需求

(1) 与前端的消息交互功能

实验者的前台操作通过前端子系统处理后封装成预定的控制消息，并发送给后端子系统。该功能对接收到的控制消息进行分类处理后将相应的处理结果反馈给前端子系统。

(2) 多实验者隔离功能

为了支持多实验者的同时使用，不仅需要前端子系统进行业务的隔离处理，还需要后端子系统进行数据的隔离处理。后端系统以实验者作为后端子系统的基本管理单位，通过维护一张实验者对象的 hash 表为每个实验者提供独立的管理虚拟用户资源和用户组的功能，从而实现后端子系统中多实验者的隔离处理。

(3) Web 流量产生功能

产生 Web 业务流量是系统最基本也是最重要的功能。由于系统是通过控制虚拟用户来产生 Web 流量，因此系统 Web 流量的产生能力将直接影响整个系统能够支撑的虚拟用户数，并制约着系统的实际运用价值。

(4) 日志功能

日志功能是记录系统运行状态的重要手段，以便于系统的使用者随时查看、监视和分析系统的运行情况，方便系统的开放者跟踪系统运行过程调试程序。

2.1.2 性能需求分析

为了支持大量 Web 用户的模拟并产生高强度的 Web 背景流量，该系统在性能方面应具备以下特点：

(1) 高并发性

高并发性是系统能够产生大流量的基础。因为系统是通过模拟和控制大量的虚拟 Web 用户访问目标服务器来产生流量，所以将存在大量的用户并发处理和网络连接并发处理。因此，为了产生适合实验网络应用场景下的高强度 Web 背景流量，系统应具备高并发特性。

(2) 稳定性

稳定性是系统实际运用的基本要求，关系到系统能否长时间正常工作。由于 Web 流量产生系统中涉及到大量虚拟用户的实时调度和管理，如果系统在长时间运行过程中出现内存泄漏或溢出、CPU 使用率过高等情况，将会造成产生的流量不够稳定，甚至导致系统崩溃无法运行。因此，系统需要满足稳定性要求。

2.1.3 技术指标

根据上述系统的功能及性能需求分析，本节提出如下技术指标，包含功能指标和性能指标。

1. 功能指标

(1) 前端系统能够为实验者提供虚拟用户申请、虚拟用户组配置和实验监测三个核心功能。

(2) 支持多个实验者同时使用，且能够完成实验者之间实验数据的相互隔离。

(3) 后端系统需支持系统配置、进程管理和系统日志记录功能。

2. 性能指标

(1) 能够产生比较真实的流量。流量的真实性体现在两方面：1) 具有较好的自相似性。根据文献[23]对实际网络流量的分析，其流量的 Hurst 指数位于 0.71~0.89 之间，因此系统产生的流量的 Hurst 指数也应该位于这个范围。2) 符合大时间尺度下网络流量的变化。因此系统应支持 24 小时大尺度的流量产生。

(2) 具有较高的并发性能。系统至少支持模拟上 1 万个 Web 用户。

(3) 具有良好的稳定性。系统至少在 24 小时内不会出现内存泄漏、系统崩溃等情况，且 CPU 使用率保持在 80%以下。

2.3 系统总体设计

2.3.1 系统软件结构设计

Web 流量产生系统的软件结构由前端子系统和后端子系统两部分组成。其中，前端子系统采用 B/S 架构，通过 PHP 语言结合 Ajax、HighCharts 图表库等技术进行开发，主要是为实验者提供一个友好的交互界面，可以在前端子系统实现虚拟用户资源申请、用户组创建与配置、实验运行监控等功能。后端子系统在 Tilera 提供的 MDE 开放环境下采用 C 语言以及 Tilera 平台 API 开发，主要用于实现大量虚拟用户的行为模拟、高强度的 Web 业务流量产生等功能。Web 流量产生系统的软件结构如图 2.2 所示。

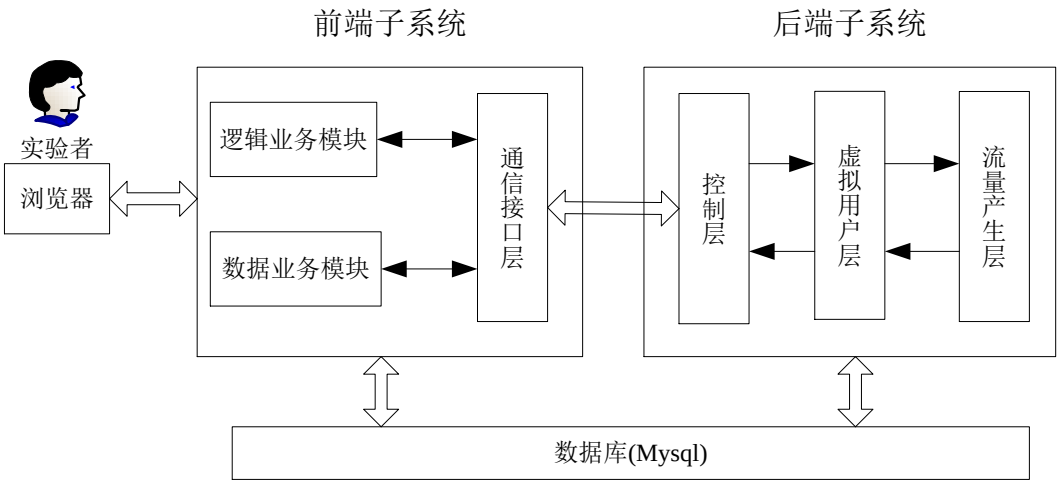


图 2.2 系统软件结构示意图

系统软件结构说明如下：

1. 前端子系统

(1) 逻辑业务模块

该模块主要负责子系统的逻辑业务处理，为实验者提供一个可视化的实验配置与管理接口。实验者可通过该接口向系统申请虚拟用户资源，并对虚拟用户资源进行分组配置和管理。

(2) 数据业务模块

该模块主要负责子系统的数据业务处理，便于实验者查看虚拟用户的访问过程以及直观地显示虚拟用户产生的 Web 背景流量。

(3) 通信接口模块

该模块将实验者的前端操作封装为预定格式的操作消息，通过 socekt 数据通道传给后端子系统，并处理后端子系统返回的实时消息。

2. 后端子系统

(1) 控制层

该层负责整个后端子系统的配置管理以及与前端子系统的消息交互。控制层通过读取后端子系统的配置文件初始化整个后端子系统，主要包括日志系统、配置对象、事件驱动对象、虚拟用户资源、请求对象资源初始化以及负责实现流量产生层的工作进程的创建和销毁管理、前端子系统消息交互、信号处理等。

(2) 虚拟用户层

该层负责虚拟用户池的管理、活跃用户的实时调度、Web 用户行为的模拟以及与流量产生层交互的消息处理。

(3) 流量产生层

该层主要负责请求对象池的管理以及与目标 Web 服务器进行数据交互的并发处理。流量产生层不断地接收来自虚拟用户层的请求消息，然后从请求对象池中获取可用的请求对象，并建立与目标 Web 服务器的连接，最后基于事件驱动进行 HTTP 数据异步处理，将访问结果记录到日志文件并定时推送到前端子系统。

3. 数据库

该模块负责存储系统初始化以及运行过程中产生的数据。其中，前端子系统中包括系统配置数据库、实验者数据库以及由后端子系统产生的日志数据库；后端子系统包括存储目标 URL 的 URL 数据库。

2.3.2 系统功能模块设计

根据系统的功能需求分析，论文将整个 Web 流量产生系统分为两个功能子系统：前端子系统和后端子系统，其主要功能模块如图 2.3 所示。其中，前端子系统主要是为实验者提供一个与后端子系统交互的可视化窗口，主要完成前台展示、业务处理；后端子系统主要根据前端子系统发送过来的业务请求，完成虚拟用户资源分配、用户行为动作的实时调度与管理，从而实现 Web 背景流量的产生。

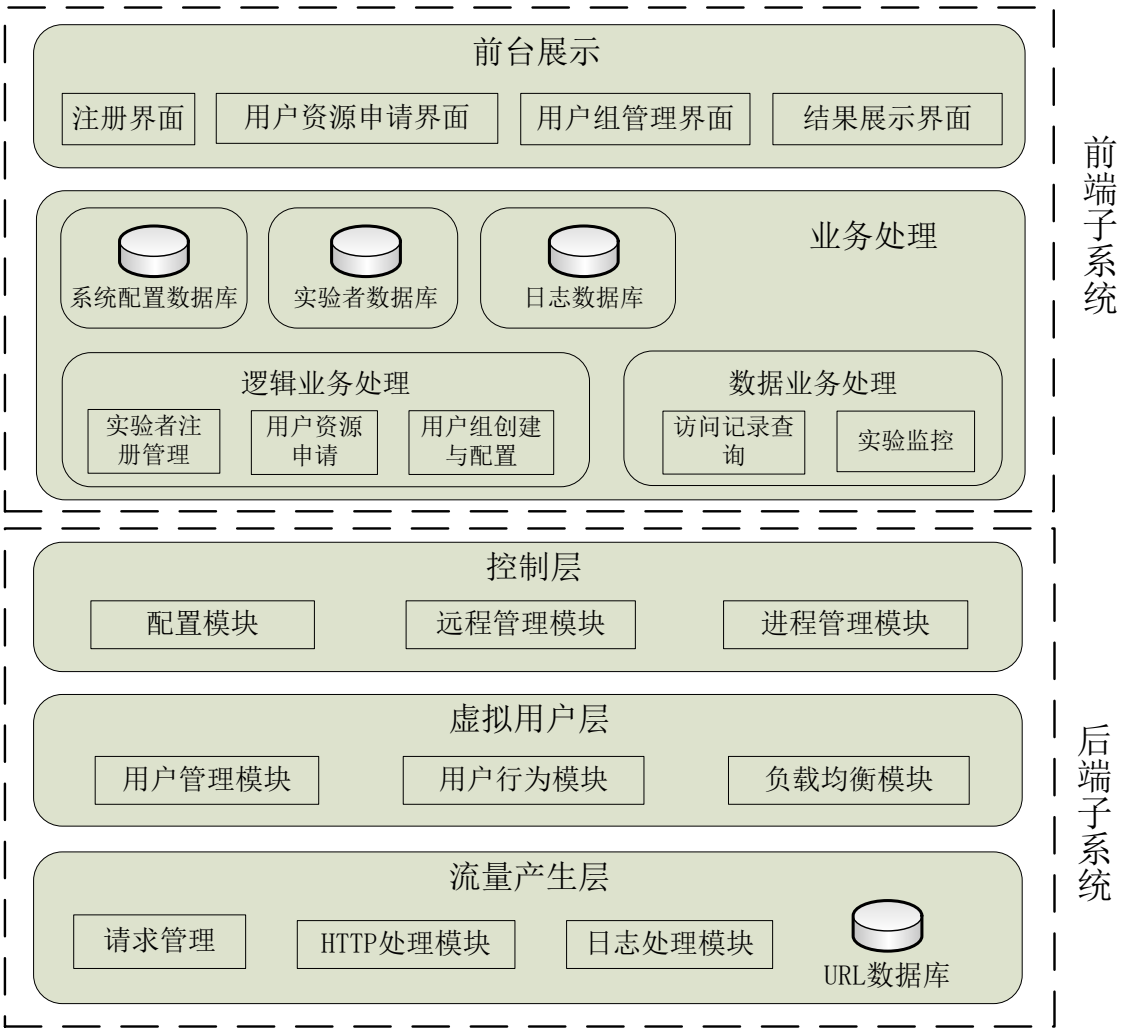


图 2.3 系统功能模块

1. 前端子系统

(1) 前台展示

该部分主要包括注册界面、用户资源申请界面、用户组管理界面和结果展示界面。

(2) 业务处理

系统的业务可分为两种：逻辑业务和数据业务。其中，逻辑业务模块主要实现实验者注册管理、用户资源申请和用户组创建与配置三个逻辑业务的处理；数据业务模块负责数据业务的处理，包括实验监控和访问记录查询。

2. 后端子系统

(1) 控制层

该层主要的模块有进程管理模块、配置模块和远程管理模块。其中，进程管理模块负责进程创建、多进程并行化设置、事件驱动初始化以及信号事件注册等；配置模块负责对后端子系统配置文件进行分模块的解析，为系统正常启动提供所需的参数；远程管理模块主要负责监听和处理来自前端子系统的业务消息。

(2) 虚拟用户层

该层包含的模块有用户管理模块、用户行为模块和负载均衡模块。其中，用户管理模块负责实现多实验者同时进行实验时虚拟用户的资源分配、隔离和调度；用户行为模块负责完成 Web 用户行为模型的实现，控制每个虚拟用户的浏览过程；负载均衡模块负责将请求消息均衡地发送给位于流量产生层的多个流量产生进程。

(3) 流量产生层

流量产生层可分为请求管理模块、HTTP 处理模块和日志处理模块。其中，请求管理模块负责请求对象资源池的创建和管理、目标 URL 获取与解析、HTTP 请求消息的构造、TCP 连接的建立以及网络 I/O 事件的注册和回调处理；HTTP 处理模块作为响应数据的处理模块，主要完成 HTTP 响应消息的异步解析以及 HTTP 响应实体的丢弃处理；日志处理模块完成虚拟用户访问过程的日志记录并将访问日志定时地推送到前端子系统的日志数据库中。

2.4 本章小结

本章首先对系统的设计目标及特点进行说明。然后分别从系统功能、性能以及技术指标三个方面对 Web 流量产生系统进行需求分析。最后，在明确 Web 流量产生系统的设计目标以及系统需求的基础上，完成了 Web 流量产生系统的总体设计，包括软件结构设计和系统功能模块设计。

第 3 章 关键技术研究

3.1 Web 用户模拟方法研究

根据系统需求，Web 流量产生系统需要产生接近于实际网络的 Web 业务流量，以保证实验网测试结果的可靠性。对于基于用户行为的流量产生器而言，虽然可以通过模拟大量的用户，实现多个 ON/OFF 源的叠加来产生具有自相似性的 Web 业务流量，但是在较大的时间尺度下（如一个小时或一天），其流量变化是相对平稳的。然而在实际网络中，网络流量将会随着网民作息规律的变化而变化，即白天网络中的流量较大而晚间较少。因此，为了更加真实地模拟网络流量，还需要进行大时间尺度下的流量模拟。由于本系统采用基于用户行为的方式产生流量，所以可以通过控制大尺度下用户数的变化实现大尺度的流量模拟。通过以上分析，本文的 Web 用户模拟方法包括两部分：(1) 针对单个 Web 用户的行为模拟方法；(2) 基于三次样条插值算法的用户控制方法。下面分别对其进行详细说明。

3.1.1 Web 用户行为模拟方法

实际生活中，单个 Web 用户的一次浏览过程大致可以分为以下三个阶段：(1) 用户通过浏览器等工具，选择一个自己感兴趣的链接进入某个主题，如新闻头条、娱乐消息、军事热点等，开始一次会话；(2) 用户对浏览器呈现的网页信息进行浏览。浏览完毕后，通常用户会在相对较短的时间里点击另一个与之关联或感兴趣的新链接，继续浏览；(3) 当用户获得所需的信息后会停留较长的时间，进入非活跃状态，结束会话。根据 ON/OFF 模型，Web 用户的浏览过程描述如图 2.4 所示。

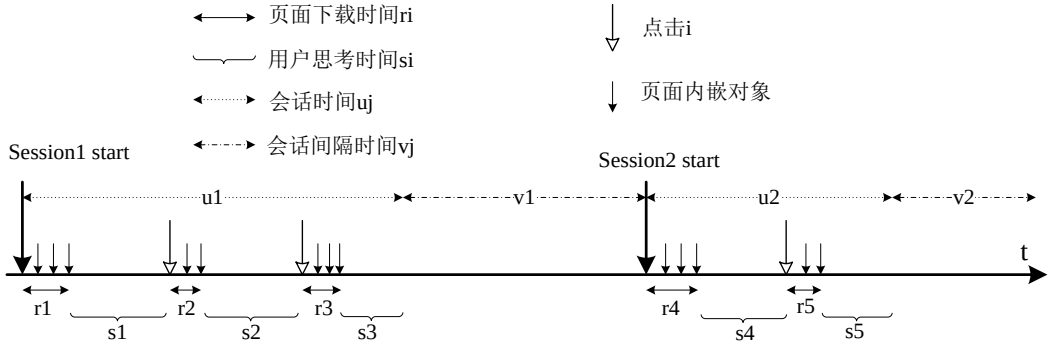


图 3. 1 用户浏览行为的 ON/OFF 描述

用户的浏览过程由两个层次的 ON/OFF 过程构成：一是页面间的 ON/OFF 过程，包含页面下载 ON 时间 ri 和用户思考 OFF 时间 si；二是会话间的 ON/OFF 过程，包含会话 ON 时间 uj 和会话间隔 OFF 时间 vj。其中，ri 的长短主要取决于网络环境的好坏，如网络带宽、服务器的处理性能等，一般不能直接控制；而 uj 的长短则主要由会话中的点击数决定，通常点击数越多会话时间越长。因此，为真实地模拟 Web 用户的浏览行为，需要确定下面几个随机变量的分布。

(1) 每次点击后用户的思考时间：文献[24-27]都提出了各自 Web 流量模型，其中用户思考时间的分布具有重尾特性，这类分布有韦伯分布（Weibull）、对数正态分布（Lognormal）等。系统默认值设为对数正态分布，最佳匹配参数为 $\mu = 0.495$,$\sigma = 2.773$ 。

(2) 每次会话中的点击数：文献[25,28]对每次会话的点击数进行了分析，匹配的分布有逆高斯（Inverse Gaussian）分布和韦伯分布。系统将默认值设为韦伯分布，最佳匹配参数为 $a = 0.31, b = 0.65$ 。

(3) 每次会话的间隔时间：文献[28-30]别对 Web 用户的会话特征进行了分析，其中会话的间隔时间服从的分布有指数分布（Exponential）和韦伯分布，系统默认值设为韦伯分布，最佳匹配参数为 $a = 0.93, b = 1.67$ 。

(4) 会话到达率：文献[25]对多个 Web 网站的流量模型进行研究，并指出会话到达是一个泊松（Poisson）过程，其强度参数 λ 可选值有 0.39、0.21、0.037。

这些随机变量的系统默认分布及参数如表 3.1 所示（这些默认值都是选自最新的参考文献）。

表 3.1 用户行为模型中的随机变量与最佳匹配分布对照表

随机变量	最佳匹配分布
思考时间	Lognormal ($\mu = 0.495, \sigma = 2.773$)
点击数	Weibull ($a = 0.31, b = 0.65$)
会话间隔	Weibull ($a = 0.93, b = 1.67$)
会话到达率	Poisson ($\lambda = 0.39$)

通过上面的分析，本文的 Web 用户行为模拟流程如图 3.1 所示。

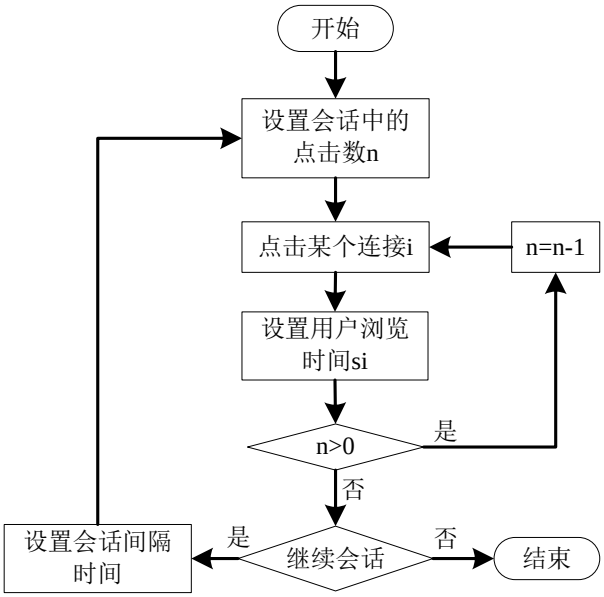


图 3.2 Web 用户行为模拟流程

3.1.2 基于三次样条插值算法的用户控制方法

在由“中国网站排名”网发布的《网站流量分析报告》中包含了对网站日访问量变化的分析[31]。这份报告对不同类型的网站，即场景（如新闻类、博客类、搜索引擎类等）的用户数时段变化情况及特征进行了详细的描述，其中图 3.2 和图 3.3 分别描述了新闻类网站和博客类网站的用户数时段变化情况。此外，由中国互联网络信息中心（CNNIC）在 2010 年 1 月发布的第 25 次《中国互联网络发展状况统计报告》中也包含了网民上网时段变化的描述[32]。同时，在百度统计的流量学院网站上可以查看到最近 1 年半的网民上网时间分布的数据[33]。这些数据的统计覆盖了超过 150 万的站点，因此能够较真实地反映出整个网络中用户的上网特征。图 3.4 是 2016 年 2 月份网民上网时间的变化情况。

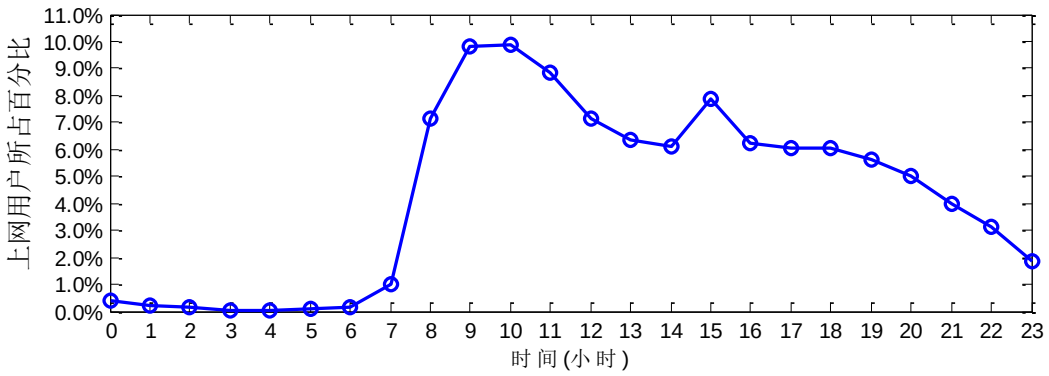


图 3.3 新闻类网站的网民时段变化曲线

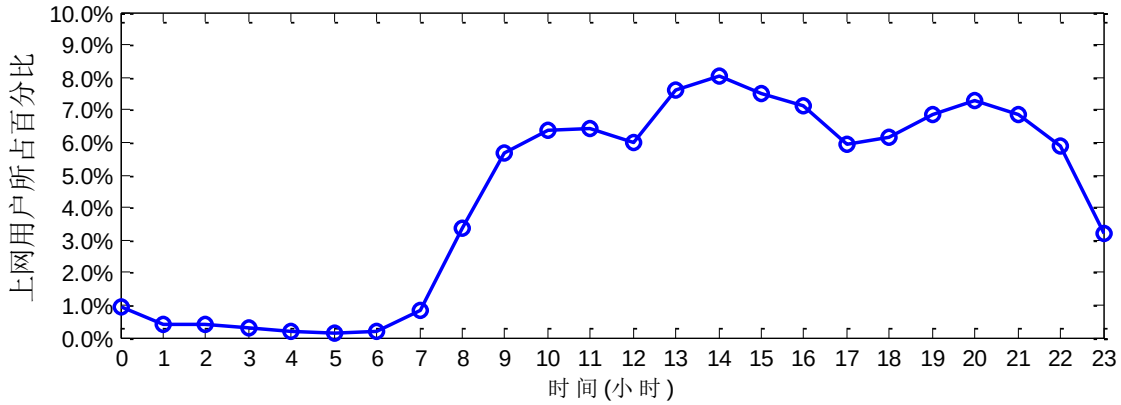


图 3.4 博客类网站的网民时段变化曲线

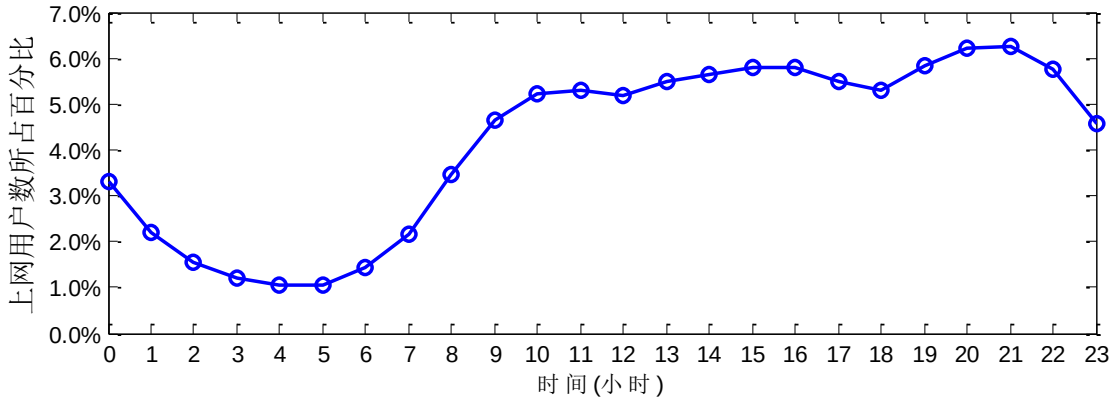


图 3.5 源自百度统计的 2016 年 2 月网民时段变化曲线

从上面的用户上网时段分布可以看出：1) 不同场景的用户时段变化曲线具有不同的特征。2) 实际网络中每个时段的活跃用户数是比较少的，这样可以通过较少的虚拟用户来模拟具有较大用户群的网络或网站。从图 3.4 可以看出，一个拥有 10 万用户的网络可通过约 7 千个虚拟用户来模拟。3) 进行大尺度的用户模拟还需要对用户时段变化曲线进行细化，这样才能更加细粒度的控制用户数，从而更加准确的反应网络流量的变化。

为了实现用户时段变化曲线的细化，而又保持用户数的变化趋势，本文将采用插值算法使其连续化。常用的插值算法有 Lagrange 插值、Newton 插值、Hermite 插值和三次样条插值[34]。其中三次样条插值算法精度最高且达到曲线的二阶光滑，在实际应用中较为广泛。因此，本文将采用三次样条插值算法对用户时段变化曲线进行处理。下面对三次样条插值算法进行简单介绍。

首先，假设有一组离散控制点 $(x:a=x_0<x_1<\ldots<x_n=b;\mathrm{y}:y_0,y_1,...,y_n)$，则其三次样条曲线 $S(x)$ 满足以下条件：

(1) 在每个分段区间$[x_i, x_{i+1}](i=0,1,...,n-1), S(x)=S_i(x)$都是一个三次多项式；

(2) 满足$S(x_i)=y_i(i=0,1,...,n)$；

(3) $S(x)$，导数$S'(x)$，二阶导数$S''(x)$，在[a,b]区间都是连续的，即$S(x)$曲线是光滑的。

因此，n 个三次多项式分段函数可记为：

$$S_i(x)=a_i+b_i(x-x_i)+c_i(x-x_i)^2+d_i(x-x_i)^3,(i=0,1,...,n-1) \tag{3.1}$$

其中：a_i, b_i, c_i, d_i表示三次多项式中的系数。因此共有 4n 个未知系数。

经过推导可得到：

$$h_i m_i+2(h_i+h_{i+1})m_{i+1}+h_{i+1}m_{i+2}=6\left[\frac{y_{i+2}-y_{i+1}}{h_{i+1}}-\frac{y_{i+1}-y_i}{h_i}\right],(i=0,1,...,n-2) \tag{3.2}$$

其中：$h_i=x_{i+1}-x_i, m_i=S_i''(x_i)=2c_i$。$h_i$表示离散控制点的距离，$m_i$表示$S_i(x)$的二次导数。

但是这只有 n-1 一个方程，却有 n+1 个未知数m_i。因此需要指定边界条件才能求解。通常有三种边界条件取值：

(1) 自由边界（Natural），即$S''(x)=0$，具体表示为$m_0=m_n=0$；

(2) 固定边界（Clamped），即指定首尾两端的微分值，记为

$$S_0'(x_0)=A, S_{n-1}'(x_n)=B \tag{3.3}$$

其中：A 和 B 为指定的常数；

(3) 非节点边界（Not-A-Knot），即

$$S_0'''(x_1)=S_1'''(x_1), S_{n-2}'''(x_{n-1})=S_{n-1}'''(x_{n-1}) \tag{3.4}$$

为了方便本文将选择自然边界，具体的求解推导过程可以参考文献[35]。

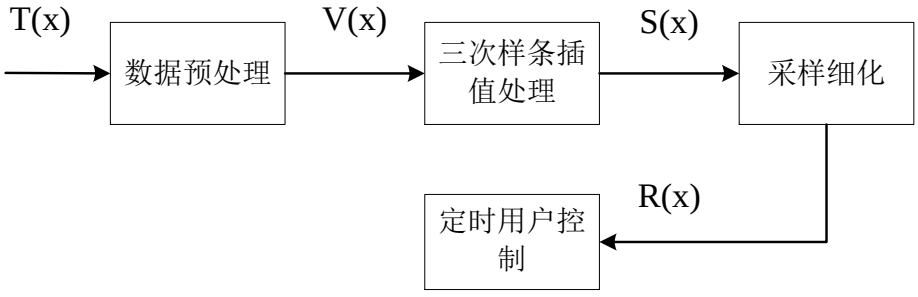


图 3.6 大尺度用户控制流程

根据三次样条插值得到的$S(x)$，可以得到$[a,b]$区间内任意位置的值。因此，本系统的用户控制方法包含数据预处理、三次样条插值处理、细化抽样处理和定时用户控制几个部分，其处理流程如图 3.5 所示。

假设实际用户总数为 *N*，用户时段百分比数据为 *T(x)*，对应的虚拟用户数为 N_v，且 $N_v \geq N * \max[T(x_i)], (i = 0,1,...,23)$，则用户控制过程如下：

(1) 对 *T(x)*进行预处理得到 *V(x)*，以实现实际用户变化百分比到虚拟用户变化百分比的映射。根据每个时段实际活跃用户与虚拟活跃用户相等，即

$$N * T(x_i) = N_v * V(x_i), (i = 0,1,...,23) \tag{3.5}$$

可得，每个时段的虚拟用户百分比 $V(x_i) = N * T(x_i) / N_v, (i = 0,1,...,23)$；

(2) 使用三次样条插值算法，按照时间顺序对 *V(x)*进行插值处理得到 $S(x)$；

(3) 以 30 秒为步长，获取 2880 个更小粒度的数据 *R(x)*并存储到数组 *B*；

(4) 以 30 秒为周期进行用户数的调整。假设现在处于第 *i* 个周期，活跃的虚拟用户数为 *A*。如果 $A > B[i] * N_v$ 则减少 $A - B[i] * N_v$ 个活跃用户；否则增加相应的用户，以满足用户变化曲线。

3.2 提高系统并发性能的方法研究

由前一章的系统需求分析可知，系统需要对大量的虚拟用户和网络连接进行并发处理。因此，并发性能是制约系统整体性能的关键。后端子系统运行在 Tile-Gx36 众核平台上，该平台以修改后的 Linux 作为操作系统，并提供了并行化编程的接口。对于应用层软件，一般可从两个层次提高并发性能：一是从操作系统层次，对操作系统进行优化，为上层应用的高并发提供基本保障。二是从应用软件层次，根据系统自身的特点，选择合适的并发或并行技术。其中，第二个层次通常又可细分为两类：一是单处理器上进行的多任务并发或伪并行处理，相关技术有多进程、多线程、事件驱动等；二是利用多处理器平台（如众核平台）进行系统的并行化处理，通过多任务的并行达到高并发的效果。

通过以上分析，提高 Web 流量产生系统并发性能的思路如下：

(1) 对 Tile-Gx36 平台的 Linux 操作系统进行内核参数优化；

(2) 通过对单核（处理器）上多进程、多线程和事件驱动的并发性优劣进行分析，选择合适的并发策略；

(3) 利用 Tile-Gx36 平台，对系统进行并行化处理。

其中，系统的并行化处理将在下一章进行详细的说明。接下来对提高 Web 流量产生系统并发性能的前两个方面进行分析。

3.2.1 Linux 内核参数优化

由于默认的 Linux 内核参数考虑的是最通用的场景，这明显不符合支持高并发处理的 Web 流量产生系统，所以需要修改 Linux 内核参数，使得系统可以拥有更高的性能。根据 Tilera Gx36 平台及 Web 流量产生系统的特点，下面主要对与系统并发性能密切相关的常用参数进行优化[36]。

1. 文件描述符数优化

在 Linux 系统中，单个进程能够同时打开地最大文件描述符（或句柄）数通常默认为 1024 或 4096。但 Web 流量产生系统后端需要与服务器建立大量的 socket 连接，所以需要对这一默认设置进行调优。一般将其设为 65536，以使其不能成为系统并发的瓶颈。通过 root 身份在/etc/security/limits.conf 文件中添加如下内容：

```
* soft nofile 65536
* hard nofile 65536
```

其中，星号表示针对所有用户，若仅针对某个用户登录 ID，需替换星号；soft 是一个警告值，表示当前系统生效的设置值；hard 是一个真正意义的阀值，超过则会报错，表示系统所能设定地最大值；nofile 表示对文件描述符进行限制。

此外，与文件描述符数有关的参数还有/proc/sys/fs/file-max 和/proc/sys/fs/nr_open，但其默认值通常都比较大，无需优化，在此不再赘述。

2. 端口数量优化

在 TCP 协议中，端口字段为 16bit，因此单机端口的上限为 65536 个。在 Linux 系统里面，1024 以下端口都是超级管理员用户（如 root）才可以使用，普通用户只能使用大于 1024 的端口值，且其提供地默认端口范围为：32768 到 61000，即最多只有 28232（61000-32768）个端口可以使用。因此，需要对端口数量进行优化，使其最大化，方法如下：通过 root 身份，在/etc/ sysctl.conf 文件中增加一行内

容“net.ipv4.ip_local_port_range= 1024 65535”，然后保存并使用“sysctl –p”命令使其生效。

假设 Linux 系统中没有其他程序占用大于 1024 的端口，上面的优化可以使 Web 流量产生系统的可用端口数达到理论的最大值 64510 个。如果 Linux 系统中其他程序占有端口太多造成端口不足或需要的端口数超过 64510 个，可通过两种方法来解决端口数量的限制问题：一是对单个本地网卡添加多个虚拟 ip 地址；二是提供多块物理网卡或采用虚拟化技术增加多个虚拟网卡。由于 Tilera Gx36 众核平台自带 8 个 1Gb/s 网卡，和两个 10Gb/s 的光口，所以这两种方法都适用。限于论文篇幅，具体的配置过程不再赘述。

3. TCP 网络参数优化

这些参数主要是对 TCP 网络的性能进行优化，以使 Web 流量产生系统支持更多的并发请求。首先，需要以 root 身份修改/etc/sysctl.conf 来更改内核参数。一个基本的配置如下：

```
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_tw_recycle = 1
net.core.netdev_max_backlog = 262144
net.ipv4.tcp_syn_retries = 1
net.ipv4.tcp_fin_timeout = 1
net.ipv4.tcp_keepalive_time = 600
net.ipv4.tcp_max_tw_buckets = 6000
net.ipv4.tcp_mem = 524288 699050 1048576
net.ipv4.tcp_rmem = 4096 32768 262144
net.ipv4.tcp_wmem = 4096 32768 262144
net.core.rmem_default = 262144
net.core.wmem_default = 262144
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152
```

然后执行/sbin/sysctl –p 命令，使上述修改生效。上面的参数意义解释如下：

tcp_tw_reuse：表示允许将 TIME_WAIT 状态的 sockets 重新用于新的 TCP 连接。默认为 0，表示关闭，这里设置为 1。

tcp_tw_recycle：表示开启 TCP 连接中 TIME_WAIT 状态的 sockets 的快速回收，默认为 0，表示关闭，这里设置为 1。

netdev_max_backlog：表示当网卡接收数据包的速率比内核处理这些包的速率快时，允许送到队列的数据包的最大数目。

tcp_syn_retries：表示在内核放弃建立连接之前发送 SYN 包的数量，默认为 6，这里设置为 1。

tcp_fin_timeout：表示主动关闭连接时，socket 保存在 FIN_TIMEOUT 状态的最大时间，默认值通常为 60 或 180 秒。

tcp_keepalive_time：表示当启用 keepalive 时，TCP 发送 keepalive 消息的频度。默认是 2 小时，若将其设置得小一些，可以更快速地清理无效连接。

tcp_max_tw_buckets：表示操作系统允许 TIME_WAIT 状态套接字的最大数目，如果超过这个值，TIME_WAIT 状态套接字将立刻被清除并打印警告信息。该参数默认值为 180000，过多的 TIME_WAIT 会使系统处理变慢。

tcp_mem[]：表示内核分配给 TCP 连接的内存范围，以页面（Page）为单位(通常 1 Page = 4096 Bytes)。其中 tcp_mem[0]，表示 TCP 并不以为自身存在着内存压力时的页数上限，即当 TCP 使用地页数少于 tcp_mem[0] 时，内核不对其进行任何的干预；tcp_mem[1]，表示 TCP 进入内存压力区域时的页数；tcp_mem[2]，表示 TCP 拒绝后续 socket 分配时的页数，超过它将会出现“Out of socket memory”错误。

tcp_rmem[]和 tcp_wmem[]：用于对 TCP 协议套接字的读写缓冲区进行管理，其中，tcp_rmem[]表示读缓冲区配置，tcp_wmem[]表示写缓冲区配置。它们都包含三个参数，分别表示最小值、默认值和最大值。

rmem_default 和 wmem_default：分别表示内核套接字接收和发送缓冲区的默认大小。

rmem_max 和 wmem_max：分别表示内核套接字接收和发送缓冲区的最大值。

3.2.2 基于事件驱动的并发

1. 常用并发技术分析

(1) 多进程并发

进程是操作系统资源分配的最小单位，相比其他并发技术，其创建、上下文切换和销毁等开销较大。因此，多进程并发通常只适合小并发量的应用场景。此外，每个进程都是一个的独立运行空间，进程间的数据共享和通信将变得困难。

(2) 多线程并发

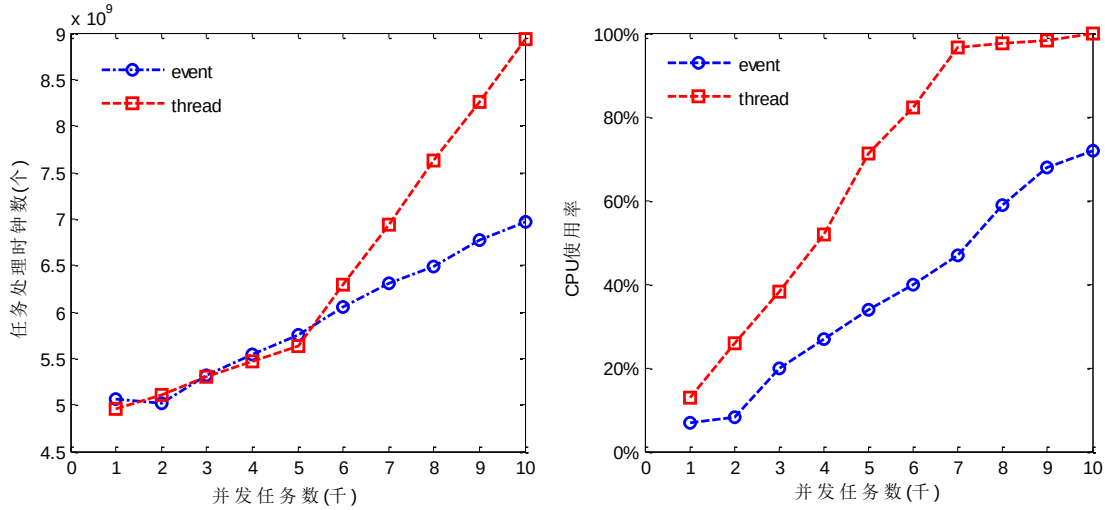
在 Linux 系统中，线程是系统的最小调度单位，也被称为轻量级进程。多个线程运行在一个进程空间中，所有线程共享进程资源。因此线程间的数据共享将变得简单。但为了实现临界资源的互斥访问，需要对其进行同步操作（如加锁），这不仅增加了同步开销，而且容易引起死锁。此外，与进程相比，线程的创建和撤销较小。因此，多线程技术比较适合并发要求不太高的应用场景。

(3) 基于事件驱动的并发

事件驱动通常是指通过操作系统提供的 I/O 复用接口（如，Linux 中的 epoll、poll、select）实现了对 I/O、信号、定时等异步事件统一管理的框架。这种框架能够处理大量的网络 I/O 且具有低开销的特点，非常适合高并发的网络应用。如主流的轻量级、高并发的 Web 服务器 Nginx[37]和高性能缓存系统 Memcached[38]都采用事件驱动提高其并发性能。常用的事件驱动框架有 ACE[39]、Libevent[40]等，其中 Libevent 是纯 C 语言实现、开源、轻量级、高性能的事件驱动库，现已被 Chromium、Memcached、Transmission 等广泛运用，这也是本系统所采用的。

2. Tile-Gx36 平台上多线程与事件驱动并发性能比较

通过上面分析可知，在较高并发场景下，一般采用多线程或事件驱动进行并发处理。为了对这两种并发技术在 Tile-Gx36 上的性能有个更清晰地认识，将一个进程绑定在 Tile-Gx36 的一个 tile 核上，然后分别采用多线程和事件驱动方式对不同并发任务下的任务处理耗时和 cpu 使用率进行测试。测试中每个线程和事件重复执行 10 次如下处理过程：1) 从事先生成且固定大小为 50，区间为[10，20]的随机数组中轮询选取一个随机数，然后乘以计算基数 cbase 得到累加上限 n；2) 实现从 1 到 n 的累加计算；3) 设置休眠或超时时间为 sleep，时间到达后进入步骤 1）。进行不同的累加计算，主要是为了模拟每次网络 I/O 读写不同的处理时间。本次测试的 cbase=1000，sleep=2s，任务数从一千到一万。测试结果如下图 3.6 所示。其中，任务处理耗时是指对每个任务创建一个线程或事件进行并发处理所消耗的总 CPU 时钟数。



(a) 任务处理耗时对比 (b) CPU 使用率对比

图 3.7 多线程与事件驱动并发性能比较

由图 3.6(a)可知，在并发任务数较小的情况下，基于多线程的处理耗时与基于事件驱动的耗时基本一致。当任务数超过 5 千时，基于事件驱动的耗时增速几乎没变，而基于多线程的耗时增速将会明显增大。从图 3.6(b)可以看出，多线程的 CPU 使用率增速较事件驱动快的多。出现这种并发性能差异的原因主要是多个线程在单个 tile 核上的频繁切换。因此系统将采用基于事件驱动的并发方式，以提高单进程的并发处理能力。

3.3 本章小结

本章首先对 Web 用户模拟方法进行了研究，包括 Web 用户行为的模拟方法和基于三次样条插值算法的大尺度用户控制方法。然后结合系统高并发的要求和 Tile-Gx36 众核平台的特点，提出了一种从 Linux 内核参数优化、基于事件驱动的并发和基于 Tile-Gx36 众核平台的系统并行化三个方面提高系统并发性能的方法，并对 Linux 内核参数优化和基于事件驱动的并发进行了详细介绍。

第 4 章 基于 Tilera 众核平台的系统并行化设计

4.1 Tilera 众核平台的并行编程技术

4.1.1 Tile-Gx36 平台简介

1. 硬件平台

Tile-Gx36 由 Tilera 公司于 2012 年推出地集成了 36 个 CUP 核的众核处理器，其硬件架构如图 4.1 所示。与传统的多核处理器不同，在 Tilera 处理器内部，每个 Tile 核都是功能完整的执行单元并通过专有的 iMesh 结构进行互联，能够完成核间高效的数据路由和通信。这种高效的 iMesh 网络架构，使得 Tilera 处理器有高性能、低功耗、易扩展等特点[41]。

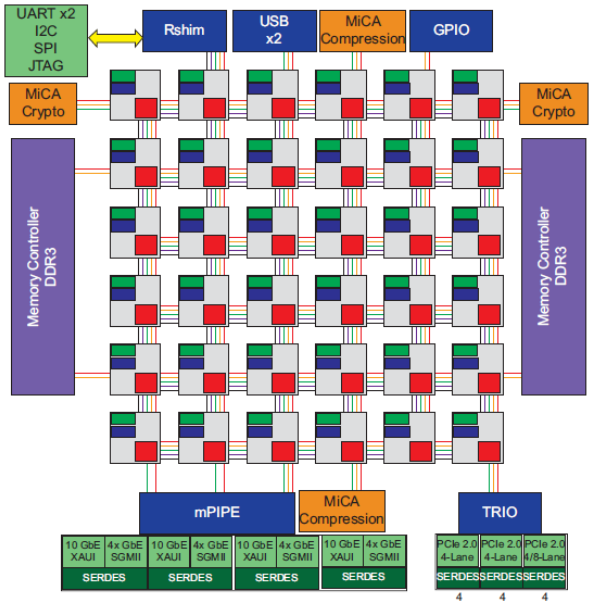


图 4.1 Tile-Gx36 处理器架构

2. 软件平台

为了便于在 Tilera 众核平台上进行开发，Tilera 为开发者提供了一个多核开发环境 MDE（Multicore Development Environment），其整体框架如图 4.2 所示。主要包括集成开放环境（Tilera IDE）、开发工具（tile-gcc、tile-g++、tile-gdb 、tile-monitor、tile-sim 等）以及仿真平台。通过 MDE 开发者可对程序进行编辑、编译、调试和优化等操作，并可在仿真平台和硬件平台上运行程序。

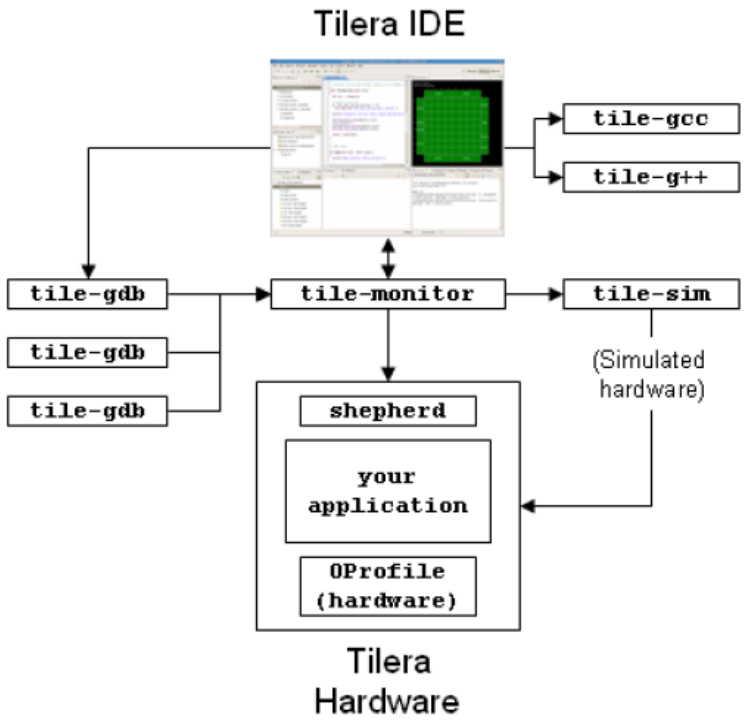


图 4.2 MDE 整体框架

此外，为了加快在 Tilera 处理器平台上进行高性能的应用开发，Tilera 提供了多种可由用户空间的应用程序使用的库，主要有：

(1) tmc 库（Tilera Multicore Components Library）：为构建并行程序提供了许多有用的原语，包含了任务的绑核管理、特殊功能的内存分配、任务间的通信等相关的 API。

(2) Tile 体系架构头文件（Tile Architecture Headers）：提供了 TILE 体系架构的多种低级别访问接口，这些头文件可被任何层次（包括用户空间、Linux、Hypervisor 和裸金属环境（Bare Metal Environment））的软件使用，主要包括原子指令宏（arch/atomic.h）、低级别 UDN 支持（arch/udn.h）、CPU 周期计数器（arch/cycle.h）和模拟器控制（arch/sim.h）。

(3) gxio 库（Low-Level IO Device Control API ）：提供了直接从用户空间分配和控制输入输出设备的接口。

(4) gxcr 库（User-Level Crypto Acceleration API ）：提供了硬件加速的加密和包处理操作的 API。

(5) gxpci 库（PCIe and StreamIO Data Transfer API）：提供了在 Tile-Gx PCIe 端口与 Tile-Gx PCIe 端口以及 Tile-Gx PCIe 端口与一个 x86 主机间直接从用户空

间进行数据传输操作的 API。

4.1.2 Tilera 众核的并行编程

众核并行编程是指利用众核处理器的众核资源对程序进行并行化（go parallel）的过程。在 Tilera 平台上可通过标准的 pthread_create()或 fork() API 对程序进行并行化。通常涉及任务分解、选择处理模型（Process Model）、任务映射、通信与同步四个过程。下面对这几个基本过程进行简单介绍。

1. 任务分解

任务分解是指按照一定的方法将应用程序分解为若干个子任务（sub-task），以便进行程序的并行化设计。常用的分解技术有：

(1) 功能分解（Functional Decomposition）：是指将应用程序按照功能的差异划分成不同的模块。常用方法是根据程序流程对应用程序进行流水线式的功能划分。

(2) 数据分解（Data Decomposition）：是指将应用程序中处理的数据划分成基本相等的数据片，然后以相同的算法对每个数据片进行处理。

2. 处理模型

任务分解完成后，需要决定选择何种处理模型使程序并行执行。Tile-Gx36 支持多程序多数据（Multiple Program Multiple Data，MPMD）和单程序多数据（Single Program Multiple Data，SPMD）两种模型，其结构如图 4.3 所示。在实际开发中，通常会同时使用这两种模型。

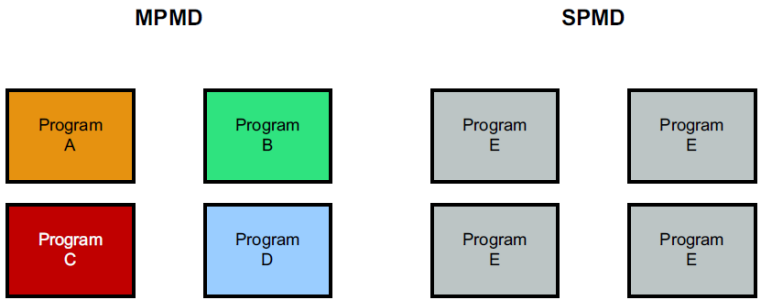


图 4.3 多处理编程模型

(1) MPMD：与功能分解相对应，通过多个执行不同功能的程序来共同完成任务。

(2) SPMD：与数据分解相对应，将数据分解后交由多个执行大致相同任务的程序进行处理。

3. 任务映射

任务映射是指通过 Tilera 提供地 API，将应用程序中包含的多个进程或线程绑定到不同 tile 核的过程。这样可以避免进程或线程在多个核间切换的开销，从而提升程序的性能。

4. 通信与同步

对于并行化应用程序通常包含多个进程或线程，所以通信和同步是不可缺少的。常用的同步原语有互斥锁、条件变量、信号量等，除了标准的 Posix 同步 API，Tilera 还提供了多种基于其平台的高性能同步 API，如 <tmc/sync.h> 和 <tmc/spin.h> 头文件。同样，对于通信方式，除了支持 Posix 的通信接口，Tilera 还提供了两种核间通信机制：共享内存（Shared Memory，SM）和用户动态网络（User Dynamic Network，UDN）。

4.2 系统流程分析

根据系统的总体设计，从层次角度，系统产生流量的过程大致可以分为四个阶段，如图 4.4 所示。

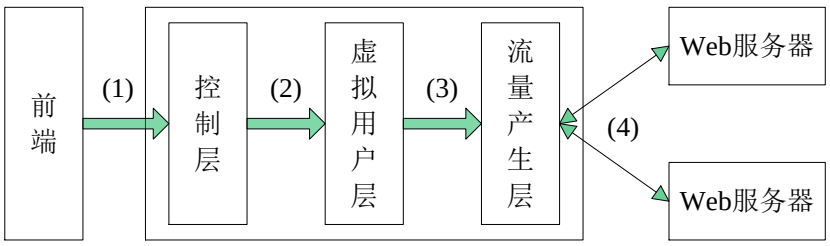


图 4.4 系统流量产生过程

第一阶段：实验者通过前端向控制层发送模拟业务消息，主要包括实验者信息、模拟用户的数量、用户行为模型参数等。

第二阶段：控制层对接收到的业务消息进行预处理，将其封装为特定的消息格式，然后发送给虚拟用户层并等待处理结果。当处理结果到达时，将其发送给前端。

第三阶段：虚拟用户层通过对控制层发送消息进行解析，分配相应的资源（如，虚拟用户资源），然后将消息的处理结果反馈给控制层。之后将激活每个虚拟用户并注册对应的定时事件。当有虚拟用户的定时触发时，调用其回调函数向流量产

生层发送请求消息，然后通过用户行为模型确定用户下一个动作的触发时间并重置定时时间。

第四阶段：流量产生层根据请求消息，获取指定的 URL，然后对其进行解析并建立与目标 Web 服务器的连接，同时将注册连接的可读或可写的 I/O 事件。当有事件触发时，调用相应的回调函数。通过多次事件触发完成与目标服务器的 HTTP 数据交互过程。最后将请求过程以一定的格式记录到日志数据库中。

4.3 系统并行化设计

4.3.1 任务分解

通过 4.1 节的系统流程分析可知，流程的一、二阶段相对简单，系统的主要处理任务集中在流程的三、四阶段，分别对应于系统的虚拟用户层和流量产生层。为了充分利用 Tile-Gx36 的众核资源并有效地实现系统的并行化，需要对这两个层次进行进一步的任务分解。

1. 流量产生层

流量产生层主要负责流量的产生以及日志记录。因此，按照功能不同将其分解为流量产生引擎和日志处理引擎两个部分。此外，为了实现系统的高并发，单个流量产生引擎是难以胜任的。因此，将采用数据分解的方法，将流量产生任务分解到多个流量产生引擎中去，实现多流量产生引擎的并行化处理。

2. 虚拟用户层

虚拟用户层主要负责虚拟用户的管理。作为中间层次，它既需要处理控制层发送过来的任务消息，也需要将自身产生的大量请求消息发送到流量产生层中的多个进程中。为此，按照功能不同将其为三个部分：

(1) 任务处理：负责处理控制层的任务消息以及根据任务消息进行资源的分配和回收处理并将处理结果反馈给控制层。

(2) 用户调度：负责实现大量用户的定时调度，并将产生地请求消息加入到请求队列。

(3) 负载均衡：负责将请求队列中的消息，均衡地分配到每个流量产生引擎。

通过以上任务分解，系统的并行化处理流程设计如图 4.5 所示。其中绿色背景表示控制层，蓝色背景表示虚拟用户层，紫色背景表示流量产生层。此外，由于

虚拟用户层中三个子任务的数据共享程度较高，将采用线程方式进行处理，其余子任务都将采用进程方式实现。

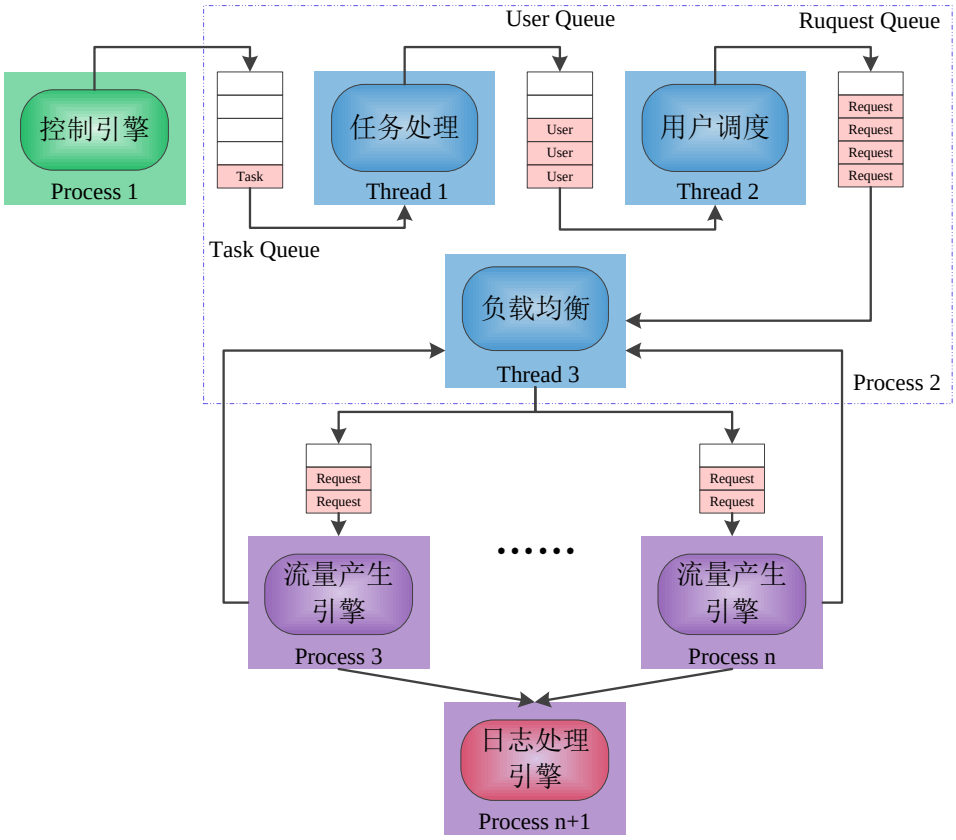


图 4.5 系统并行化处理流程

4.3.2 任务映射

在 Tile-Gx36 平台上，任务的映射是通过设置任务进程或线程的 CPU 亲和性（CPU Affinity）来实现的，即将进程或线程绑定到一个或多个 CPU 上。这样将有助于提高程序 CPU cache 的命中率，从而减少内存访问次数，提高程序的执行速度。此外，当进程或线程绑定了 CPU，Linux 将不会对其进行 CPU 的调度，可以减小程序的调度开销。在 Tile-Gx36 平台上， CPU 亲和力的设置过程一般有两步：1) 获取程序的亲和集；2) 根据任务事先在 CPU 亲和集中映射的序号，绑定到指定的 CPU 上，其关键代码如下：

```
// 获取程序的 cpu 亲和集
cpu_set_t cpus;
if (tmc_cpus_get_my_affinity(&cpus) != 0)
```

```
        tmc_task_die("tmc_cpus_get_my_affinity() failed.");
    // 根据任务序号，绑定到指定的 cpu 上
    if (tmc_cpus_set_my_cpu(tmc_cpus_find_nth_cpu(&cpus, rank)) < 0)
        tmc_task_die("tmc_cpus_set_my_cpu() failed.");
```

通过上面的方法，可将系统按照 4.2.1 节划分地任务进行绑核处理。其中，控制引擎分配一个核，虚拟用户引擎分配一个核，日志处理引擎分配一个核；每个流量产生引擎独立分配一个核，可按照实际需求调整流量产生引擎数目。图 4.6 给出了一个流量产生引擎为 6 的系统映射示意图。

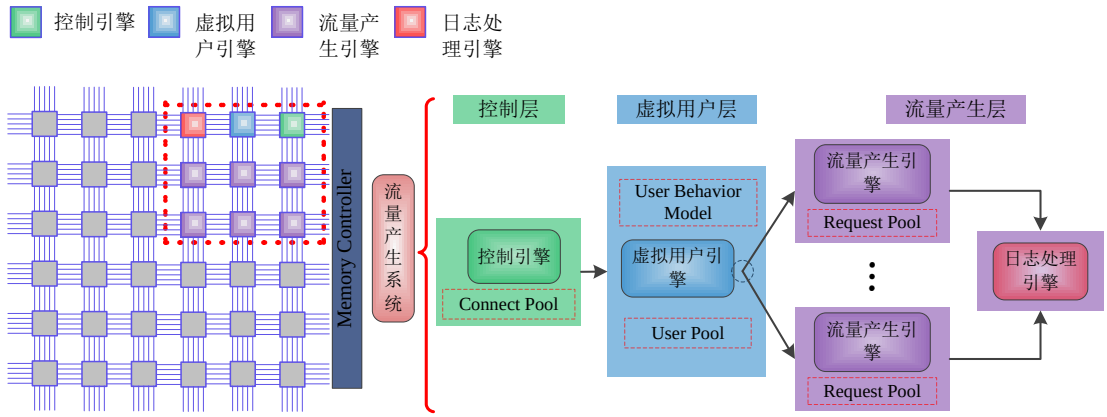


图 4.6 基于 Tile-Gx36 平台的系统映射框架

4.3.3 任务间的通信方案

众核编程不仅需要对系统进行任务分解及映射处理，还需要选择合适的任务间通信方式，对于多进程的应用而言更是如此。常用的进程间通信方式有管道（Pipe）和有名管道（FIFO）、消息队列、Unix 套接字、信号、共享内存等[42]。此外，Tile-Gx36 还提供了平台专有的共享内存和用户动态网络（UDN）两种通信方式[43]。下面对它们各自的特点进行简单说明（以 Posix 为规范描述）。

(1) Pipe/FIFO：以字节流的方式，进行单向数据传输，支持亲缘（FIFO）或非亲缘（Pipe）关系进程间的通信，具有随进程特性。基于描述符标识，能直接与事件驱动框架结合。

(2) 消息队列：单向数据流，以消息作为基本的传输单位，支持（非）亲缘进程间的通信，具有随内核特性。此外，提供了基于信号的异步通知接口，可直接与事件驱动框架结合。

(3) Unix 套接字：采用全双工的通信模式，支持字节流和数据报两种数据传输方式。因为数据传输无需经过协议栈处理，所以速度较 TCP 和 UDP 套接字快。此外，支持进程间文件描述符的传递，可实现（非）亲缘关系进程间通信，能与事件驱动框架结合。

(4) 信号：“天生”的异步特性，事件驱动框架内在支持。可向任意指定地进程 ID 发送信号，但不能携带数据进行传输，常用于通知进程某个事件已发生。

(5) 共享内存：采用内存映射方式，实现亲缘或非亲缘关系进程间数据的共享。此外，由于存在多个进程同时对共享数据进行操作的情况，所以通常需要与同步技术（互斥锁、条件变量、信号量等）结合使用。这种方式常用于进程间需要进行大块数据的交互或处理的情况。

(6) UDN：这是 Tilera 平台特有的通信方式，用于实现两个 tile 核间直接传输小的数据包，数据的大小不能超过 128 字节。如果数据包过大或接收缓存溢出，将会导致死锁。此外，Tilera 提供的接收接口是采用轮询机制实现的，如果没有数据到达将会导致额外的 CPU 消耗。因此，Tilera 开发文档也建议除非必要，否则应尽量少用 UDN。

通过上面的分析，能直接与事件驱动框架结合的有 Pipe/FIFO、消息队列、Unix 套接字和信号。与 Pipe/FIFO 和消息队列相比，Unix 套接字直接支持全双工模式，能够较好的满足任务间的双向数据通信。此外，Linux 系统中提供了多种不同的信号能实现对应用程序不同事件（如退出、终止、超时等）的监听，且大部分可以自定义信号处理函数。因此，系统将采用 Unix 套接字加信号的方式实现任务间的通信。

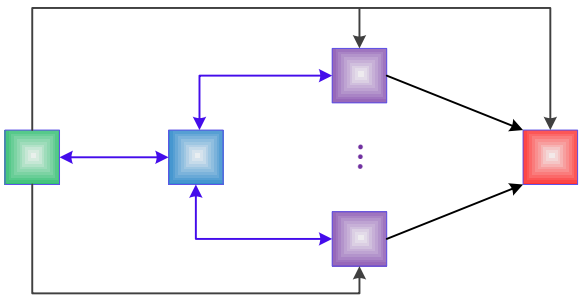


图 4.7 基于 Unix 套接字的任务间通信关系

根据任务的不同特点，将系统的 Unix 通信划分成单向数据通道和双向数据通道两种形式，其通信关系如图 4.7 所示。其中，黑线表示单向数据通道，存在与

控制引擎与流量产生引擎和流量产生引擎与日志处理引擎之间；蓝线表示双向数据通道，存在与控制引擎与虚拟用户引擎和虚拟用户引擎与流量产生引擎之间。

此外，系统还实现了 SIGTERM、SIGKILL、SIGQUIT、SIGUSR1 和 SIG CHLD 五种主要信号的处理，其对应的功能描述如表 4.1 所示。

表 4.1 信号与功能对照

信号名称	功能说明
SIGTERM	用于系统的退出。如果退出延时小于 2s，将向子进程发送 SIGTERM 信号，等待子进程进行资源释放，否则将直接向所有子进程发送 SIGKILL 信号终止进程。
SIGKILL	这是 Linux 系统中两个不能被捕获或忽略的信号之一。用于直接可靠地杀死进程。
SIGQUIT	这一个信号也用于系统的退出，与 SIGTERM 不同，它提供了从终端按“Ctrl+\”终止程序的方式。此外，还会产生一个 core 文件。它主要是为了方便程序的开发与调试。
SIGUSR1	用于添加一个流量产生进程。如果流量产生进程数已到达系统启动时配置的最大值，这操作将被忽略。
SIGCHLD	子进程退出时发送给控制进程的信号。当这个信号到达，控制进程将检测子进程的退出状态码。如果为异常退出，将重启这个进程，以保证系统的稳定运行。

4.3.4 基于最小请求数的动态负载均衡

根据系统的并行化设计，在存在多个流量产生引擎的条件下，需要考虑如何将虚拟用户引擎产生地请求任务均匀地分配到每个流量产生引擎。因此，选择一个好的负载均衡算法是解决这一问题的关键。常用的负载均衡算法按照其特点可分为三类：轮询算法、基于权重的算法和基于哈希的算法[44]，下面它们对进行简单介绍。

(1) 轮询算法：采用公平策略，依次选择节点分配等量的任务。它是所有调度算法中最简单也最容易实现的，常用于所有节点处理能力和性能均匀的情况。

(2) 基于权重的算法：这类算法根据不同节点的性能或实际负载等因素的差异，为每个节点设置一定的权重，通常权重越大的节点被选中的概率越高。不同应用

场景节点权重的计算方法一般不同。一般的考虑因素有节点 的 CPU、内存、IO、响应时间、连接数等。此外，权重因素可以是单个的也可以是多个的，权重设置可以是静态的也可以是动态的。一般采用动态方式，即动态的从节点获取权重因素值，然后对该节点的权重值进行实时调整。

(3) 基于哈希的算法：这类算法通过哈希的方式将具有相同特征的任务映射到同一节点，从而减少节点资源分配、迁移等方面的开销。常用的任务特征有 IP 地址、URI、会话 ID 等。这类算法常用于具有某种内在关联性的任务。

对于本系统虽然每个流量产生引擎节点的性能是基本相同的，可以采用简单的轮询算法进行负载均衡。但是流量产生引擎节点是以请求对象作为基本的管理单位，且每个请求对象的生命周期并不相同，所以每个节点的真实负载主要取决于该节点当前的活跃请求数。此外，当系统运行过程中突然增加流量产生进程或出现进程异常恢复，简单的轮询算法难以做到快速的恢复负载均衡。因此，系统将采用一种基于最小请求数的动态负载均衡算法（简称为 DMR）。每个流量产生进程实时的将其自身的活跃请求数发送给虚拟用户进程，然后负载均衡模块进行数据更新，选择请求数最小的流量产生进程发送请求消息。它可以归属于权重类算法，权重因素为进程的活跃请求数且权重越小被选择的概率越高。

4.4 本章小结

本章首先对 Tilera 众核平台及并行编程技术进行了介绍。然后从层次角度对系统的流程进行了分析。最后结合 Tilera 众核的并行编程技术和系统的流程，对系统进行详细的并行化设计，主要包括任务分解、任务映射、任务间通信方案以及负载均衡策略的选择。

第 5 章 系统详细设计与实现

5.1 控制层设计与实现

5.1.1 进程管理模块

根据 4.2 节系统并行化的描述，系统采用多进程的方式实现系统的并行化。按照进程完成的功能不同，可分为控制进程、虚拟用户进程、流量产生进程和日志处理进程四类。此外，为了方便控制进程对其他进程的控制和管理，系统采用了父子进程的工作模式，即一个父进程（控制进程）加多个子进程（虚拟用户进程、流量产生进程和日志处理进程）。图 5.1 给出了系统启动及子进程创建的流程，其中系统初始化包含错误日志初始化、配置文件解析、守护进程设置、事件驱动初始化、信号事件注册、系统 CPU 亲和集设置等过程。

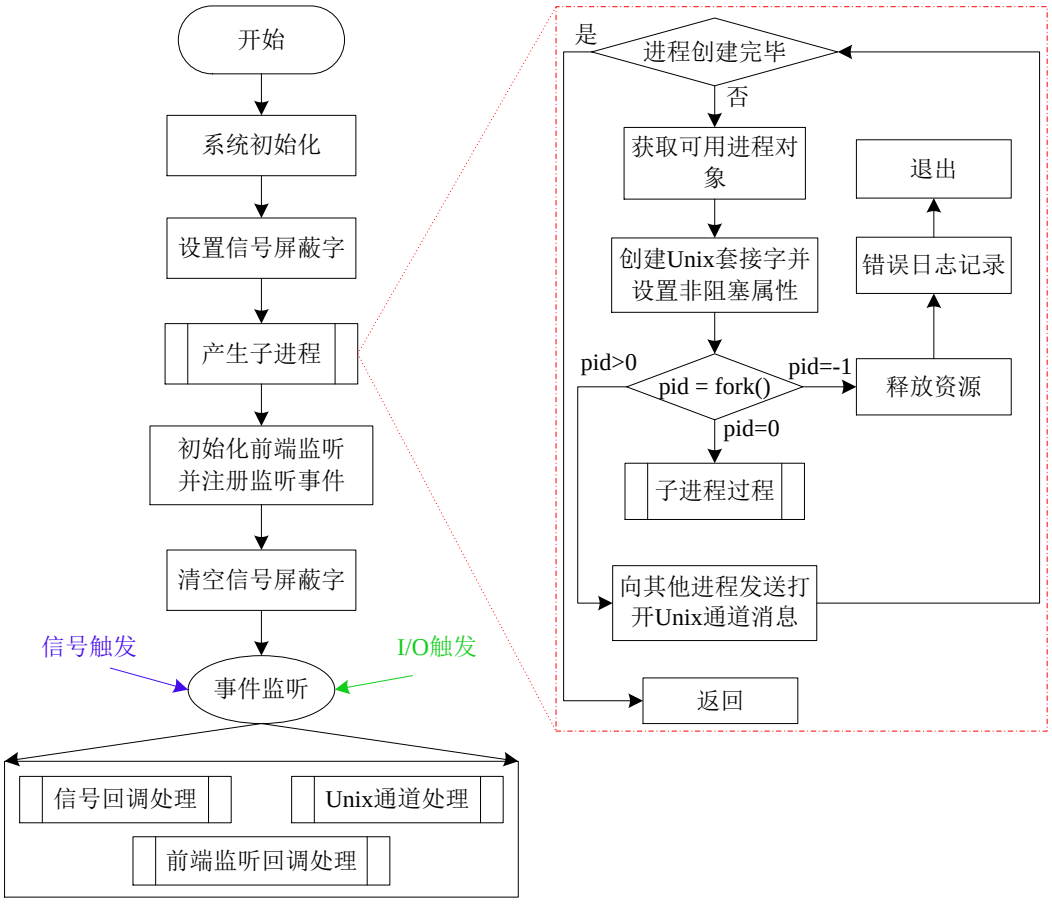


图 5.1 系统启动及子进程创建过程

控制进程在产生子进程的过程中，首先从进程对象槽中获取一个可用的进程对象，然后通过 socketpair 函数创建一对 Unix 套接字存放于进程对象中 fd[2]字段，并将这对套接字设置为非阻塞模式，接着调用 fork 函数创建子进程。子进程会关闭从控制进程继承过来的其他所有子进程的 fd[1]以及自身的 fd[0]，并对自身的 fd[1]注册读事件，然后根据自身的类型进入不同的循环过程。控制进程则通过 Unix 套接字向其他所有子进程发送该进程的 fd[0]。这样所有进程将会形成一个基于 Unix 套接字的互联的通信网络。

进程管理模块除了负责进程的创建管理外，还需要完成子进程异常退出的重启、流量产生进程的添加等功能。需要注意的是子进程的异常退出一般会发生在流量产生进程，因为它的处理过程相对复杂且资源消耗多，但这种情况极少发生的。这些功能是通过信号方式完成地，过程相对简单在此不再详细描述。表 4.1 给出了系统使用的信号及对应功能的描述。

5.1.2 配置模块

为了实现灵活的配置管理，增强系统的可扩展性，引入了配置上下文的概念。首先将配置信息按模块划分成多个主配置，然后对每个主配置依据功能不同进一步划分成多个子配置。这样所有的主配置处于系统配置上下文中，而子配置处于对应的主配置上下文中。当系统需要添加新的模块或功能时，只需在相应的上下文中进行处理，无需对其他模块进行修改，从而有效地增强了系统的可扩展性。图 5.2 给出了系统配置在内存中的映射关系。

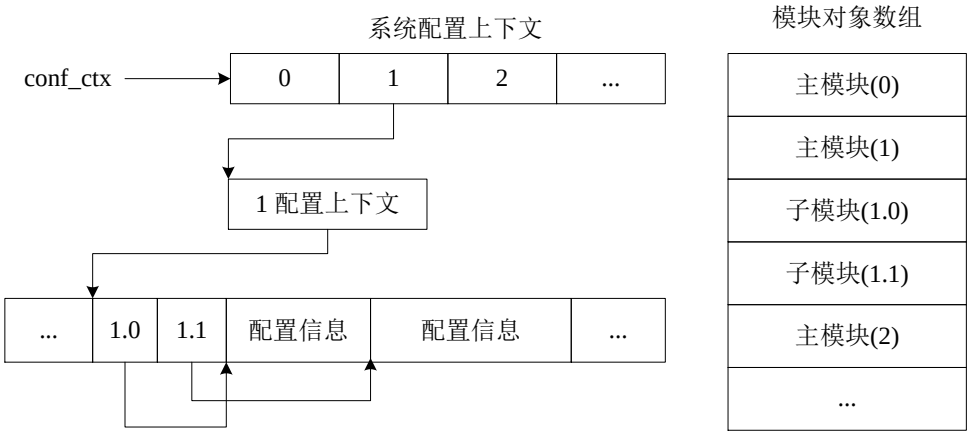


图 5.2 配置的内存映射

模块对象数组包含系统所有模块的引用，且按照主模块加子模块的方式依次排列。每个模块对象都包含一个命令集合和一个配置上下文，其中命令集合表示该模块在配置文件中支持的命令以及相应的命令配置函数；配置上下文提供了配置信息对象的创建与配置后的处理函数。其中模块对象的实现代码如下：

```
///模块对象
struct ntg_module_s {
    ntg_uint_t    ctx_index;///系统上下文索引
    ntg_uint_t    index;///内部索引
    void    *ctx; ///模块上下文
    ntg_command_t    *commands; ///命令集
    ntg_uint_t    type; ///模块类型
    /*  不同情景下的模块处理函数  */
    ntg_int_t     (*module_init)(ntg_cycle_t *cycle); ///模块初始化
    ntg_int_t     (*master_init)(ntg_cycle_t *cycle); ///主进程初始化
    ntg_int_t     (*process_init)(ntg_cycle_t *cycle); ///子进程初始化
    void          (*process_exit) (ntg_cycle_t *cycle); ///子进程退出
    void          (*master_exit)(ntg_cycle_t *cycle); ///主进程退出
};
```

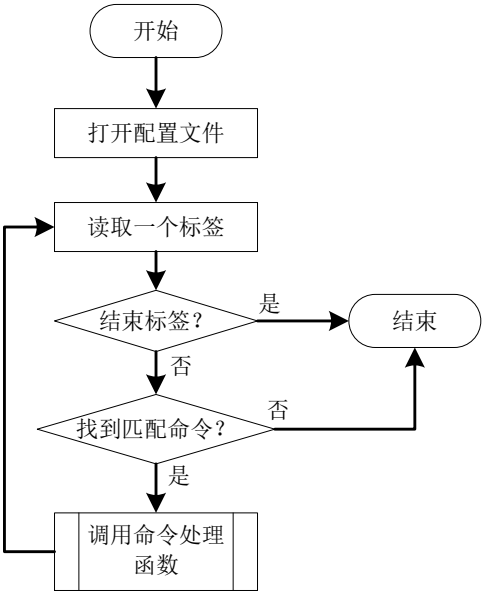


图 5.3 配置模块处理流量

配置模块对配置文件进行逐行扫描。当读取到一个完整的标签时，就遍历模块数组查找匹配的模块及对应的命令。若匹配成功则调用相应的处理函数。其过程如图 5.3 所示。

5.1.3 远程管理模块

系统启动后，远程管理模块将根据系统配置在指定的 IP 和端口上进行监听，并将监听事件注册到事件驱动中，以等待前端业务消息的到达。实验者通过浏览器将模拟配置信息或管理信息发送给前端服务器。前端服务器对其进行校验和过滤处理后，与远程管理模块建立连接并发送业务消息。然后，远程管理模块对业务消息进行处理并向前端服务器返回处理结果。其处理流程如图 5.4 所示，包含接收和响应两个子处理过程。

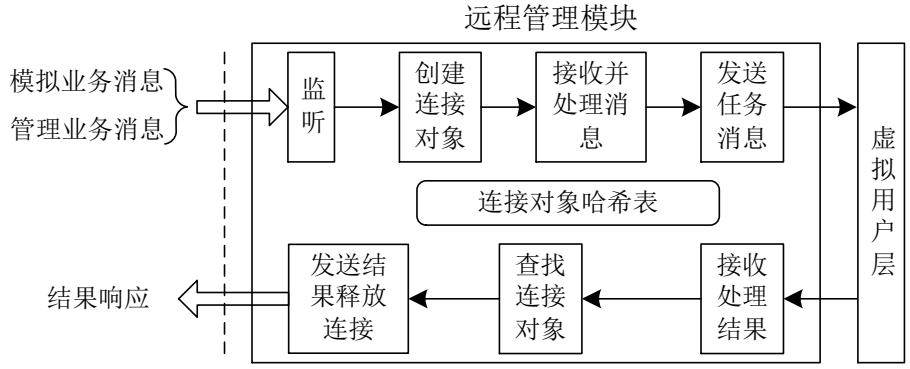


图 5.4 远程管理模块处理流程

1. 接收处理过程

前端服务器主要向远程管理模块发送两种业务消息：模拟业务消息和管理业务消息。其中模拟消息格式为：<模拟业务类型><实验者 ID><用户组 ID><用户数目><行为模型类型><模型参数><场景类型><场景参数>；管理业务消息格式为：<管理业务类型><实验者 ID><操作类型>。当远程管理模块监听到前端的连接请求后，会创建一个连接对象用于记录本次连接的套接字 fd、业务消息以及处理结果等信息。然后接收和解析 json 格式的业务消息，并以实验者 ID 作为 key 值将其加入连接对象哈希表中。然后向虚拟用户层发送任务消息。

2. 响应处理过程

结果消息的格式为：<实验者 ID><状态码>，其状态码含义如表 5.1 所示。当收到一个结果消息时，将以消息中的实验者 ID 为 key 值，在连接哈希表中查找对

应的连接对象。然后将结果消息以 json 格式发送到前端服务器，并关闭连接、释放资源。

表 5.1 状态码含义

状态码	状态含义
0	处理成功
1	创建连接失败
2	系统正在退出
3	资源不足

由于前端服务器会对实验者发送地信息进行过滤，所以大部分的业务逻辑错误将由前端服务器直接处理，所以需要返回的状态码较少。下面对部分状态码进行解释如下：状态码 2 表示系统正在退出的过程中，收到了前端服务器的业务消息；状态码 3 表示的资源不足主要是指内存的不足，虚拟用户、用户组等资源校验由前端服务器处理。

5.2 虚拟用户层设计与实现

5.2.1 用户管理模块

为了实现系统灵活的用户管理，以支持多个实验者同时使用，本模块将虚拟用户按照实验者的需求划分成不同的用户组。每个实验者可拥有多个用户组，并对其进行虚拟用户数、用户行为参数、场景参数等配置和管理。通常实验者在每次实验时，需要创建大量的虚拟用户以及对应的用户组，在实验结束时又需要对这些资源进行释放。这种频繁的内存操作会导致大量的系统开销，且容易引起内存泄漏等问题。因此，在用户管理模块中引入了“池技术”，即在系统启动时创建足够的资源形成资源池，当需要这种资源时直接从池中获取，而在使用完后又归还给池。这样可以有效地避免资源重复创建和释放地开销，从而提高系统性能。该模块中主要包含用户对象池（user_pool）、组对象池（group_pool）、实验者对象池（epm_pool）、实验者哈希表（epm_table）和一个对这些资源进行统一管理的管理对象。

用户管理模块主要包含两个过程：任务处理过程和用户调度过程。任务处理过程对控制层发送过来的管理任务消息和模拟任务消息进行处理，其流程如图 5.5

所示。在模拟任务消息处理过程中，会根据消息内容对组对象进行初始化，主要包括：空闲队列（free_q）和活跃队列（live_q）的初始化；从 user_pool 中获取指定数目的虚拟用户，并将其插入到 free_q；如果消息中包含场景参数，将使用 3.1.3 节描述的三次样条插值算法对场景参数进行处理，并生成以 30s 为周期包含 2880（24*60*60/30）个点的用户数变化曲线。

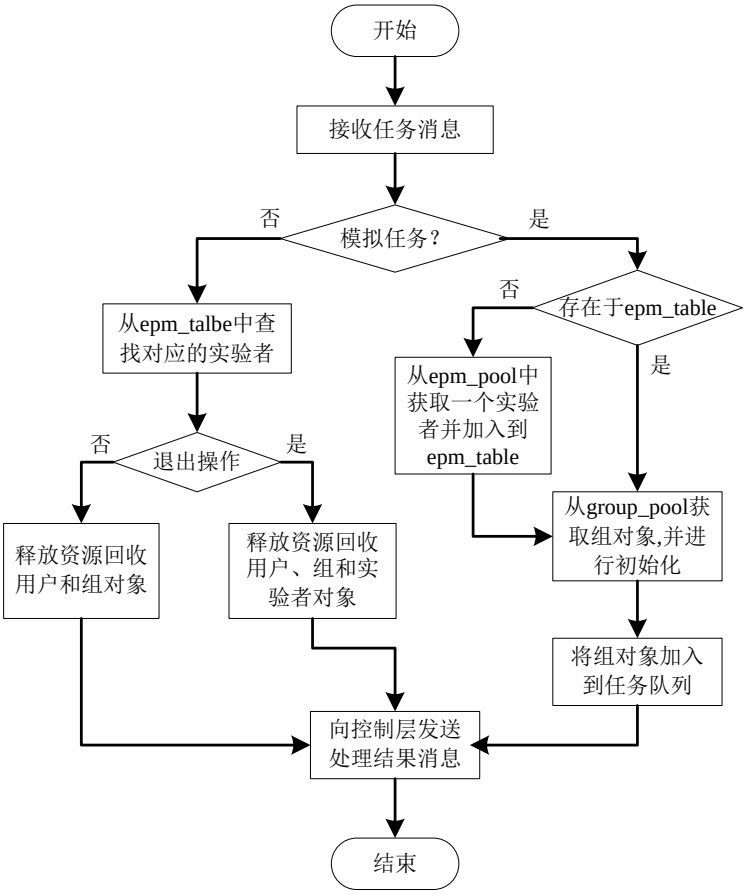


图 5.5 任务处理过程

用户调度过程可分为两个子过程：用户激活和用户数控制。如果任务队列不为空，调度程序会以一定的周期对任务队列进行扫描，并以一定的频率激活每个组内的用户，将其加入到 live_q 中。如果用户组包含场景参数，当激活用户数达到场景开始点的数量时，则交由用户控制过程进行处理。它会根据用户变化曲线对活跃的用户数目进行控制。否则，激活组内所有用户，由用户行为模块控制每个用户的行为过程。

5.2.2 用户行为模块

根据 3.1.2 节描述的 Web 用户行为模拟方法，该模块主要对用户模拟过程中的相关分布函数进行封装实现，然后对外提供能够产生符合相关分布的随机数的函数接口。生成符合指定分布的随机数的算法有很多，常用的有两种：逆变换法（Inverse Transform Method，ITM）和舍取法（Acceptance-Rejection Method，ARM）[45][46]。下面对这两种算法进行简单描述。

1. ITM 算法

该算法有两步：1) 生成一个服从均匀分布的随机数 *U~Uni(0,1)*；2) 假设 *F(X)* 为指定分布的累积分布函数（Cumulative Distribution Function ，CDF），$F^{-1}(Y)$是其逆函数，则 $X = F^{-1}(U)$为符合该分布的随机数。这是一种简单且高效的算法，如果可以这是第一选择。但是 ITM 有其自身的局限性，就是要求必须能给出 CDF 的逆函数表达式。有些时候要做到这点比较困难，这就限制了 ITM 的适用范围。

2. ARM 算法

ARM 是一种比 ITM 适用范围更广的算法，只要给出概率密度函数（Probability Density Function，PDF）表达式即可，而大多数常用分布的 PDF 是可以查到的。因此，在无法给出 CDF 的逆函数表达式时，通常选择 ARM。该算法分为 3 步：1) 设 PDF 为 $f(x)$。首先生成一个均匀分布随机数 $X \sim Uni(x_{\min}, x_{\max})$；2) 独立的生成另一个均匀分布随机数 $Y \sim Uni(y_{\min}, y_{\max})$；3) 如果 $Y \le f(x)$，则返回 X，否则回到第 1 步。ARM 本质上是一种模拟方法，而非直接数学方法。虽然 ARM 从效率上不如 ITM，但在无法得到 CDF 的逆函数时，ARM 是不错的选择。

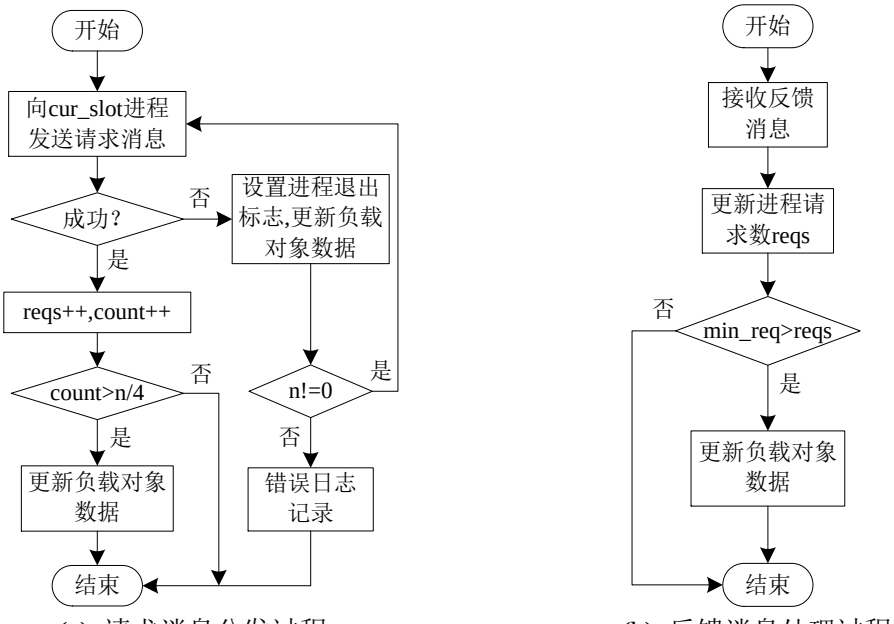
通过两种方法则可以生成系统所需的所有分布的随机数。下面仅给出服从正态分布和韦伯分布的随机数实现，它们分别对应 ITM 和 ARM 算法。

```
//获取服从韦伯分布的随机数
float ntg_math_weibull_random(float a, float scale, float shape){
    return a + scale*pow(-log(ntg_math_average_random(0,1,4)),1/shape);
}
//获取服从正态分布的随机数
float ntg_math_normal_random(float miu,float sigma,float min,float max){
    float x, y, scope;
    do{
```

```
            x = ntg_math_average_random(min,max,4);
            y=ntg_math_average_random(0, ntg_math_normal(miu,miu,sigma),4);
            scope = ntg_math_normal(x, miu, sigma);
        }while( y > scope);
        return x;
    }
```

5.2.3 负载均衡模块

为实现 4.2.4 节中所描述的负载均衡策略，负载均衡模块主要包含一个负载均衡对象和一个进程槽。其中，负载均衡对象中主要包含最小请求数（min_reqs）、具有最小请求数的进程槽位（cur_slot）和进程槽指针（pro_ptr）三个字段；进程槽中的每个槽位对应一个进程对象，包含了该进程的一些状态信息，如请求数（reqs）。该模块包含请求消息分发和反馈消息处理两个过程，其中反馈消息格式为：<进程槽位><请求数目>。图 5.6 对这两个过程进行了简单描述。



(a) 请求消息分发过程 (b) 反馈消息处理过程

图 5.6 负载均衡处理过程

由于请求消息分发是一个比较频繁的过程，因此在负载均衡对象中另外引入了分发计数字段（count）和存活的进程数字段（n）。count 表示连续向 cur_slot 进程发送请求消息的次数，只有当 count>n/4 或收到反馈消息时才会对其清零，这样

能够有效地减少查找最小请求数进程的次数。其中，更新负载对象数据主要是对负载对象中的 cur_slot、min_reqs 和 n 字段进行更新并将 count 设置为 0。

5.3 流量产生层设计与实现

5.3.1 请求管理模块

请求管理模块是流量产生层的核心模块，也是系统并发性能的关键制约要素。为此，将基于事件驱动对其进行全异步的编程实现。与通用的函数式的过程编程相比，异步编程通常将会更加复杂，往往会因事件到来的不可预测性，带来编程逻辑的复杂化，因此在该模块引入数学领域中的有限状态机思想。同时，还对本模块的所有状态进行层次化处理，将其划分为两个层次：全局状态和内部状态，从而更加清晰地描述编程逻辑。

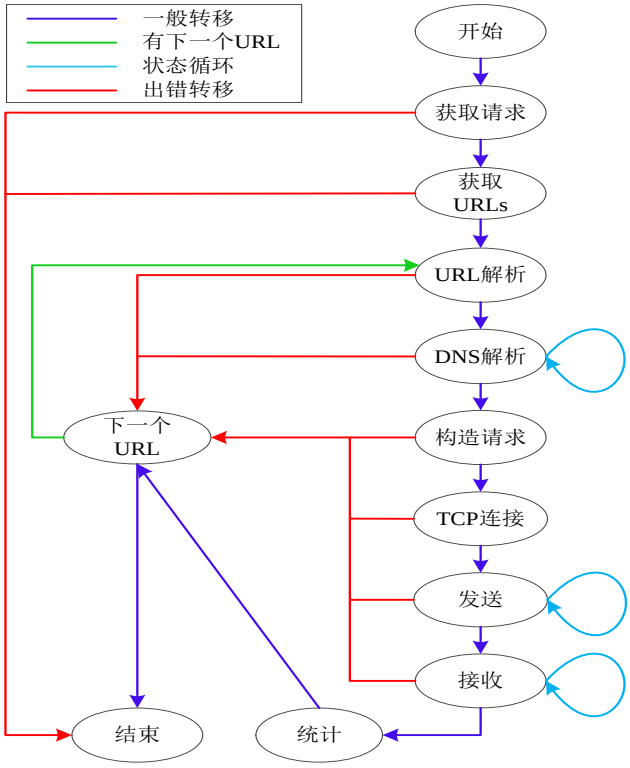


图 5.7 全局状态转移关系

根据系统需求，本层次的全局状态机描述如图 5.7 所示。其中蓝色线表示一般状态转移过程，绿色线表示存在待处理的 URL 转移过程，浅蓝色线表示这个状态可能需要经过多次异步处理才能完成，红色线表示状态处理过程中出错情况的转移。当接收到一个请求消息表示进入开始状态，然后从请求池中获取一个请求

对象并获取一个本次请求访问的页面 URLs（包含多个内嵌资源的 URL），之后对所有的 URL 进行访问并统计相关的数据，最后进入结束状态将本次请求的结果发送给日志处理进程，并回收请求对象。

请求管理模块中的核心结构体为请求对象。无论是 HTTP 协议的解析还是数据的发送和接收处理都依赖于它实现，下面给出了请求对象实现的一个简单描述。

```
//http 请求对象
struct ntg_http_request_s {
    void *data;///关联指针，可做 next 用
    ntg_cycle_t *cycle;///循环对象
    struct event_base *base;///事件驱动
    struct evdns_base *dns;///DNS 对象
    struct evhttp_uri *uri;///请求的 URL 对象
    evutil_socket_t sock;///关联的 sock
    char hostname[NTG_HTTP_MAX_HOSTNAME];///主机名
    struct sockaddr_in addr;///关联的地址
    struct event * ev_write;///读事件
    struct event * ev_read;///写事件
    struct timeval timeout;///超时时间
    ntg_int_t exp_id;///实验者 id
    ntg_int_t gp_id;///用户组 id
    ntg_int_t user_id;///用户 id
    enum ntg_req_state_e req_state;///全局状态机变量
    ntg_buf_t out_buffer; ///  输出缓存区
    off_t out_size;///输出缓存大小
    ntg_buf_t n_buffer;///  输入缓冲区
    off_t in_size;///接收缓存大小
    /*  请求统计相关字段  */
    ……
```

```
    /* 相关的处理函数指针 */
    ……
    /* 用于 HTTP 解析的相关字段*/
    ……
    ntg_array_t *urls;///url 集
    ntg_pool_t   *pool;///内存池
    ntg_log_t    *log;///错误日志
};
```

5.3.2 HTTP 处理模块

根据系统需求，在“诱导”目标 Web 服务器产生流量的过程中对 HTTP 响应的具体内容是不关心的，即不必对 Web 文件的内容进行解析。此外也无需要保存接收的 Web 文件，可以对 HTTP 响应实体直接进行丢弃。因此，HTTP 处理模块只需对 HTTP 的响应头部进行解析，确定响应实体的长度、传编码以及服务器是否支持长连接等信息并根据不同的情况对响应数据进行丢弃处理。此外，由于系统采用事件驱动的并发方式处理网络 I/O，所以对于 HTTP 数据的发送和接收必须考虑异步带来地各种边界确定问题。为此也将采用有限状态机方式实现异步条件下 HTTP 协议的解析。该模块的主要处理流程实现如图 5.8 所示。其中，绿色虚框部分表示可写事件回调函数过程，蓝色虚框表示可读事件回调处理过程。

HTTP 模块主要负责对 HTTP 响应数据的处理，下面给出了该模块实现的主要数函数接口。

```
//解析 http 的响应状态行
ntg_int_t ntg_http_parse_status_line(ntg_http_request_t *r, ntg_buf_t *b,
ntg_http_status_t *status)
//解析 http 头域
ntg_int_t ntg_http_parse_header_line(ntg_http_request_t *r, ntg_buf_t *b,
ntg_uint_t allow_underscores)
//解析 chunked 编码的数据
ntg_int_t ntg_http_parse_chunked(ntg_http_request_t *r, ntg_buf_t *b,
```

ntg_http_chunked_t *ctx)

//丢弃响应实体

ntg_int_t ntg_http_discard_response_body(ntg_http_request_t *r)

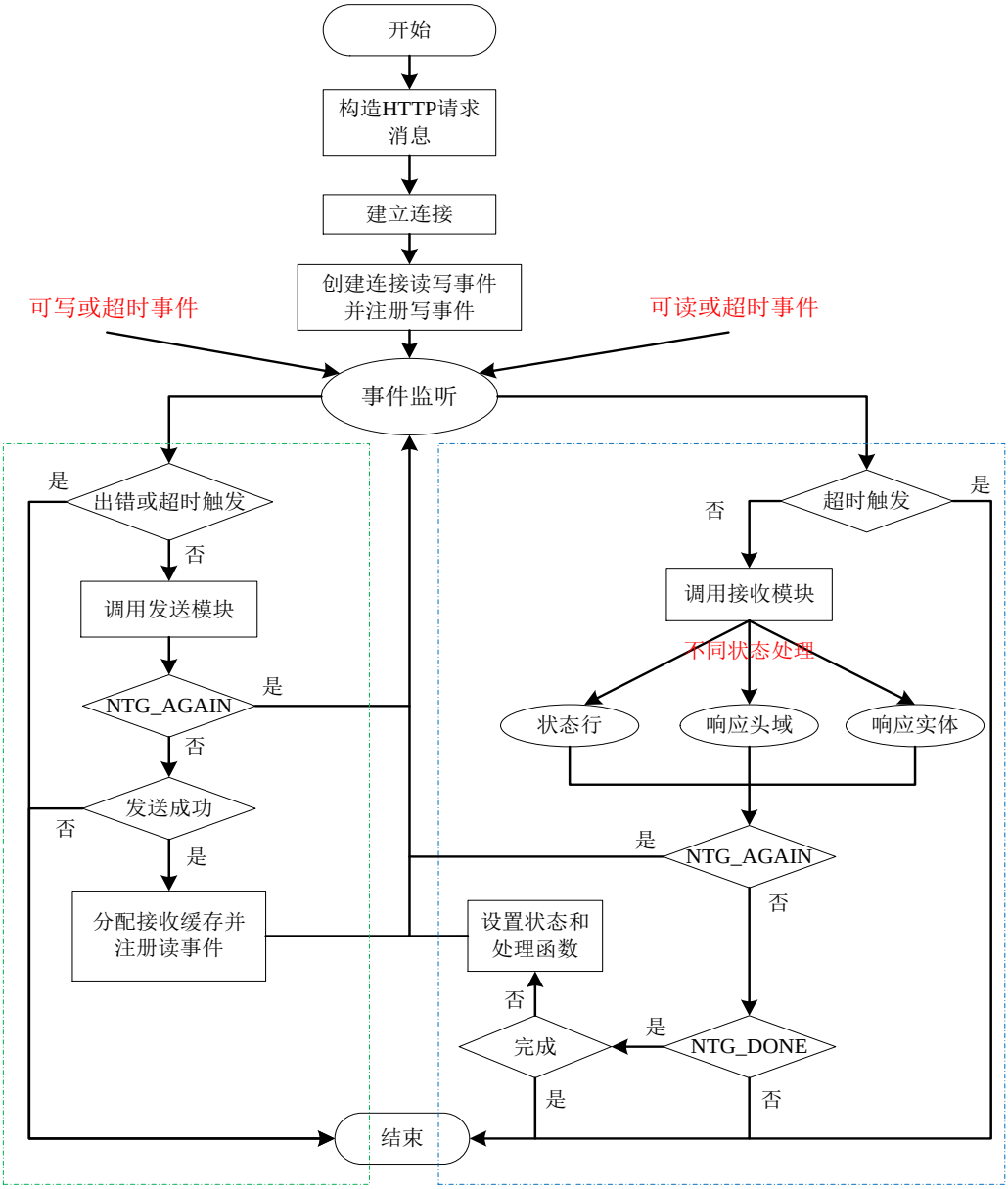


图 5.8 HTTP 模块处理过程

5.3.3 流量日志处理模块

流量日志处理模块需要将大量的流量信息插入到流量日志数据库中，以供前端使用。为了避免该模块频繁的数据库操作对数据库造成压力，将采用批量插入的方式来提高插入效率。首先该初始化一个日志缓存，然后将接收到的日志消息

（其格式为:<实验者 ID><用户组 ID><用户 ID><URL_ID><状态码><本次请求所花时间><发送数据大小><接收数据大小>）添加时间戳后记录到日志缓存中。此外，还初始化一个固定周期（通常为 1s-5s）的定时事件。当缓冲区已满或定时触发将调用相应的处理函数将日志数据批量地插入到数据库中。

5.4 本章小结

本章按照系统的层次结构，分别对控制层、虚拟用户层和流量产生层中的关键模块进行详细的设计与实现。其中，控制层包括进程管理模块、配置管理模块和远程管理模块；虚拟用户层包括用户管理模块、用户行为模块和负载均衡模块；流量产生层包括请求管理模块、HTTP 处理模块和流量日志处理模块。

第 6 章 系统测试与结果分析

6.1 测试环境

为了验证基于 Tilera 众核平台的 Web 业务流量产生系统是否达到设计需求，建立了实际的网络环境对系统进行功能及性能方面的测试。测试环境网络拓扑如图 6.1 所示。

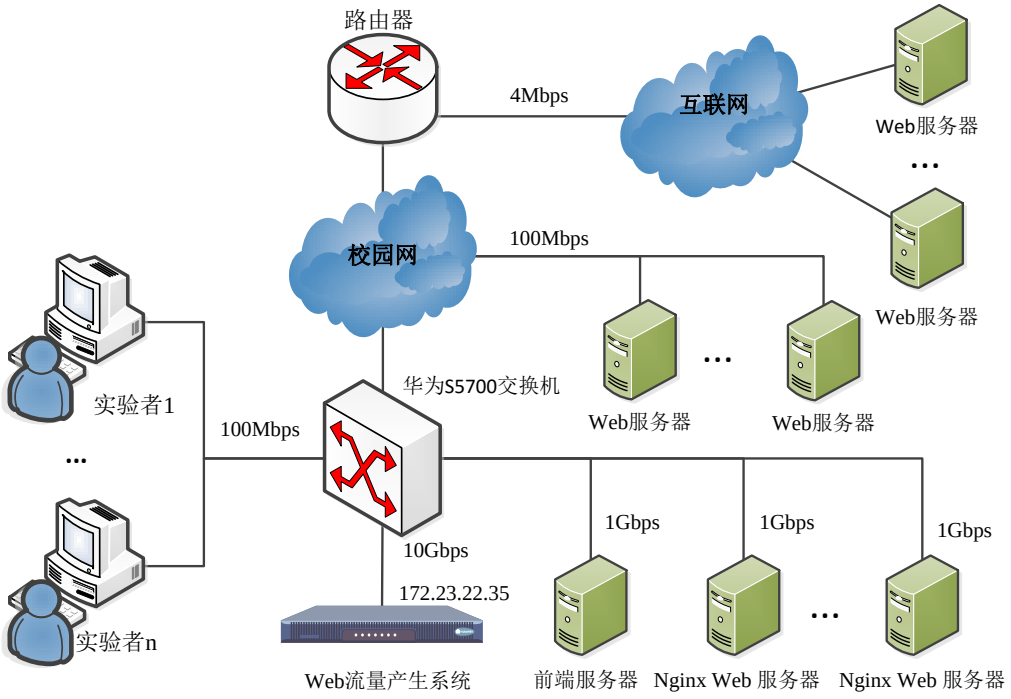


图 6.1 测试环境网络拓扑

表 6.1 测试环境配置信息

网络设备	操作系统	CPU	内存	网卡速率
Web 流量产生器	Tilera Enterprise Linux release 6.3	Tile-Gx36 1.2GHz ×36	8GB	10Gbps
Nginx Web 服务器	CentOS release 6.5	Inter(R) Xeon(R) 2.10GHz ×4	4GB	10Gbps
前端服务器	CentOS release 6.5	Inter(R) Xeon(R) 2.60GHz ×2	2GB	100Mbps

测试环境中的主要实体有：Nginx Web 服务、基于 LNMP 的前端服务器、基于 Tile-Gx36 众核平台的 Web 流量产生器以及交换机等。目前主流的三大 Web 服务器有 Apache、IIS 和 Nginx。其中，Nginx 是一个开源、轻量级、高性能的 Web 服务器且具有较高的并发性能。因此在局域网中搭建了多台基于 Nginx 的 Web 流量服务器以对 Web 流量产生系统进行全面的测试。测试网络中的主要配置信息如表 6.1 所示。此外，由于校园网和外网的带宽限制，系统的性能测试都在局域网中进行。

6.2 系统功能测试

6.2.1 前端功能测试

本节将从实验者的角度，对使用流程中涉及到的主要功能：虚拟用户申请功能、虚拟用户组创建功能和实验监测功能进行测试。通过测试，观察相关功能是否运行正常。

1. 虚拟用户申请功能

在使用系统产生流量前，需要向系统申请所需的虚拟用户资源。实验者可以通过申请虚拟用户菜单进入虚拟用户申请界面。目前实验者可申请的最大用户数为 1 万。图 6.2 展示了实验者申请 6 千个虚拟用户的结果。

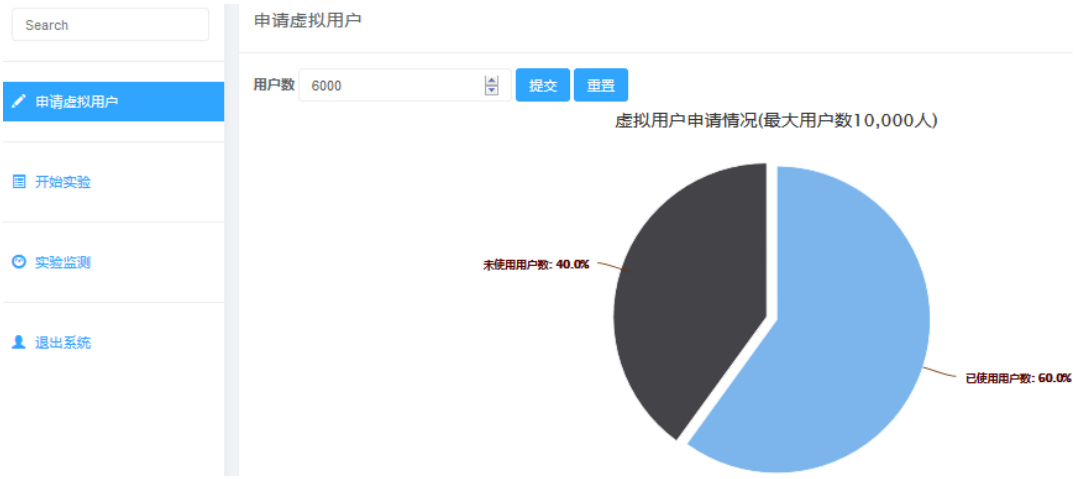
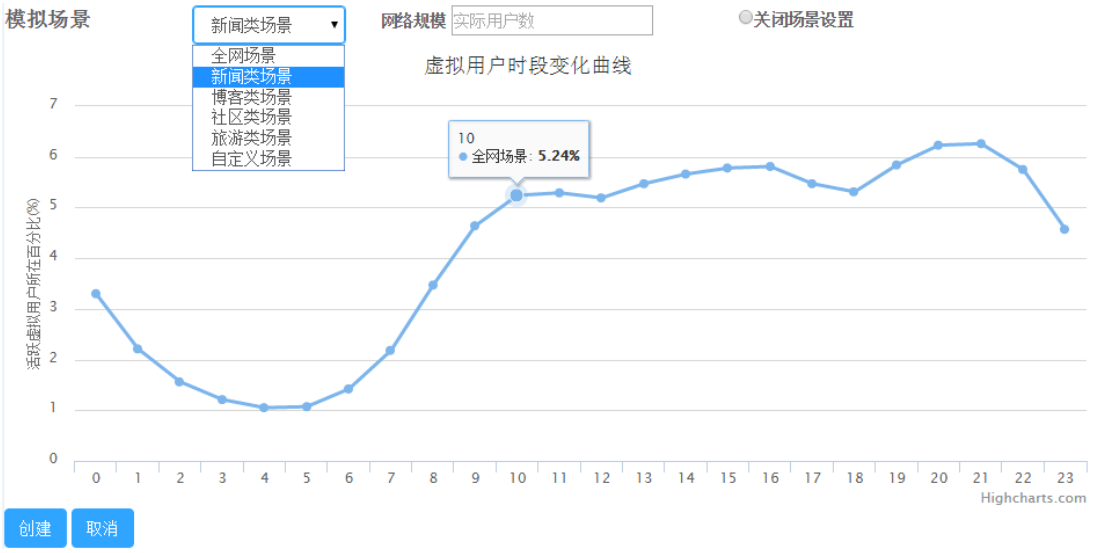


图 6.2 虚拟用户申请界面

2. 虚拟用户组创建功能

实验者可以将申请的虚拟用户资源，按照不同的需求创建虚拟用户组。可通过开始实验菜单进入虚拟用户组创建页面如图 6.3 所示。通过该功能可以指定用户组的虚拟用户数，并对整个组内的用户行为模型参数进行配置，主要配置项有会话数、会话间隔、点击数和点击间隔的分布函数及其参数，如图 6.3(a)所示。同时，实验者还可以选择开启或关闭场景配置，如图 6.3(b)所示。

(a) 虚拟用户行为配置



(b) 模拟场景配置

图 6.3 虚拟用户组创建页面

3. 实验监测功能

实验检测功能用于检测虚拟用户组产生的流量变化情况，以便于实验者对比流量变化进行相关研究的分析，主要包括实时流量、流量分析和流量日志查询等功能，其中实时流量检测如图 6.4 所示。

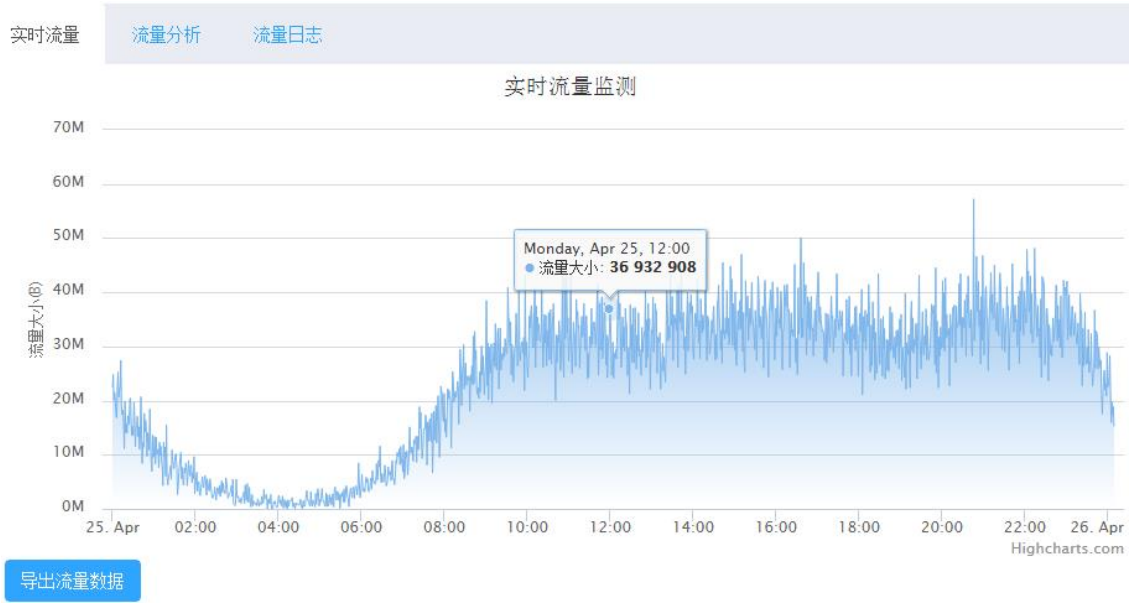


图 6.4 实时流量监控页面

6.2.2 后端功能测试

本节主要对后端系统的一些基本功能进行测试，主要包括系统配置功能、进程管理功能、负载均衡功能和系统日志功能。

1. 系统配置功能测试

```
-sh-4.1# cat conf/ntrig.conf
user  root;
worker_processes  1;
worker_cpu_affinity 3 4 5 9 10 11 15 16 17;
working_directory ./;

daemon  off;
error_log  logs/error.log  notice;
pid        logs/nginx.pid;

listen_address 172.23.22.35 3456;
database mysql 172.23.22.25 3306;

events {
    method  epoll;
}

users {
    worker_users 50000;
    req_num 2;
    balance dynamic;
}
```

图 6.5 一个基本的系统配置示例

系统配置功能让使用者能够通过系统配置文件为系统提供启动时所需的一些参数。图 6.5 给出了一个系统的基本配置。此外，系统还为使用者提供一个测试配置文件设置是否有效的命令“./ntrig_tile –t”。上述基本配置的测试过程及结果如图 6.6 所示。从图 6.5 和图 6.6 中可以看出系统提供了比较方便且全面的配置功能，包括进程 cpu 亲和性配置、日志配置、事件驱动方式配置和虚拟用户层配置等。

```
-sh-4.1# ./Debug/ntrig_tile -h
ntrig version: ntrig/1.0
Usage: ntrig [-?hvt] [-s signal]

Options:
  -?,-h         : this help
  -v            : show version and exit
  -t            : test configuration and exit
  -s signal     : send signal to a master process: stop, quit, add, reload
-sh-4.1# ./Debug/ntrig_tile -t
ntrig: the configuration file /root/.tilera/workspace/ntrig_tile/conf/ntrig.conf syntax is
ok
ntrig: configuration file /root/.tilera/workspace/ntrig_tile/conf/ntrig.conf test is succes
sful
```

图 6. 6 系统配置测试过程及结果

2. 进程管理功能测试

为了测试进程管理模块的有效性，对其进行两方面的测试：1) 任务映射功能测试；2) 添加流量产生进程测试。为了测试这两个功能，将使用 MDE 提供的网格视图（Grid View）以便于直观的观察进程与核的对应关系。

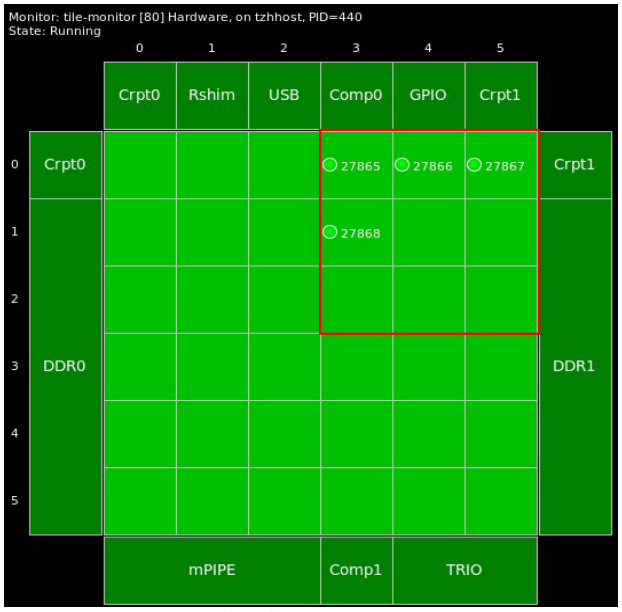


图 6.7 系统启动时的 Grid View

1) 任务映射功能测试

首先对系统配置文件进行如下设置：

traffic_processes 1;

cpu_affinity 3 4 5 9 10 11 15 16 17;

其中，traffic_processes 表示系统启动时流量产生进程数目，cpu_affinity 表示系统的 cpu 亲和集。然后，通过 MDE 启动系统程序，系统启动后的 Grid View 如图 6.7 所示，其中红线区域表示 cpu 亲和集。图 6.7 中有四个进程 id 为 2785、2786、2787、2788 的进程，分别对应控制进程、虚拟用户进程、日志处理进程和流量产生进程。它们根据配置的 cpu 亲和集依次绑定在 tile3、tile4、tile5 和 tile9。

2) 添加流量产生进程测试

通过图 6.8 所示方式添加两个流量产生进程，通过 Grid View 观察到的结果如图 6.9 所示。从图 6.9 中可以看出，添加的进程 id 分别为 31587 和 31937，且它们是不连续的。这是因为在创建它们时，有些进程 id 以被使用。此外，新添加的进程按照系统启动时配置的 cpu 亲和集顺序依次绑定在第一个可用的核上。其中 31587 进程绑定在 tile10，31937 进程绑定在 tile11。

```
-sh-4.1# pwd
/root/.tilera/workspace/ntrig_tile/Debug
-sh-4.1# ls
ntrig_tile
-sh-4.1# ./ntrig_tile -s add
-sh-4.1# ./ntrig_tile -s add
```

图 6.8 添加流量产生进程的过程

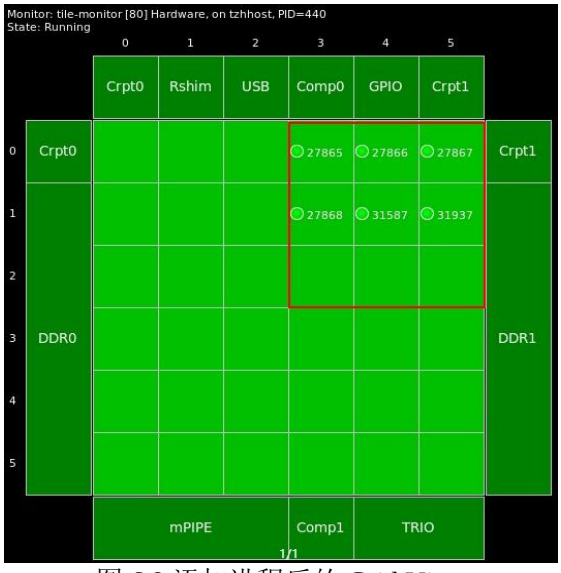
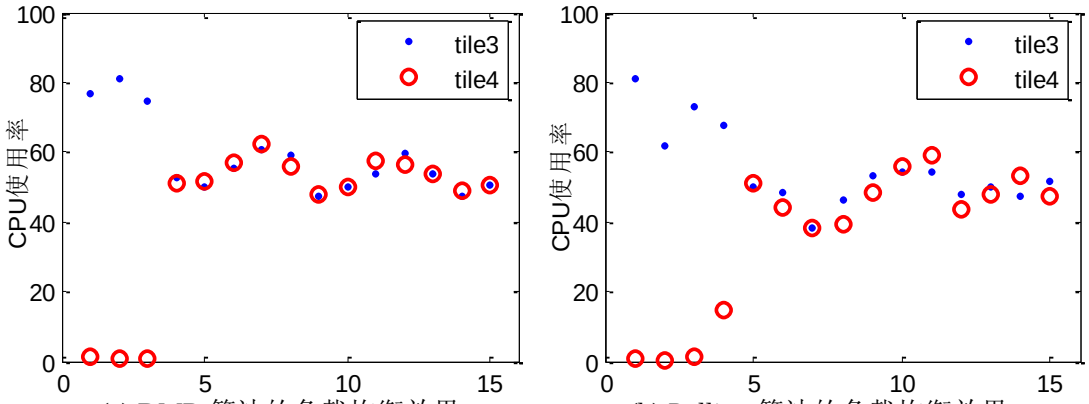


图 6.9 添加进程后的 Grid View

3. 负载均衡功能测试

为了验证 4.2.4 节提出的负载均衡算法的性能，将从两个方面进行测试：1) 正常情况下的测试；2) 节点变化情况下的测试。测试方法如下：首先开启一个流量产生进程并绑定到 tile3 核，同时让其 CPU 使用率保持在 80%左右。等到系统运行稳定后，添加另一个流量产生进程并将其绑定在 tile4 核。通过编写的 Shell 脚本以 0.001 秒为周期，同时捕获上述过程中两个核的各自 CPU 使用率。由于采集周期比较短，CPU 使用率波动性较大。为了更加清晰的描述 CPU 使用率变化情况，对每个核采集的数据进行如下处理：选取包含添加进程过程的 1500 个数据，然后每隔 100 个数据点计算一次该段的平均 CPU 使用率，得到 15 个分段描述的 CPU 使用率。

在相同的条件下，采用上述测试方法分别对本文提出的 DMR 算法和轮询（Polling）算法进行了测试，其测试结果如图 6.10 所示。



(a) DMR 算法的负载均衡效果　　(b) Polling 算法的负载均衡效果

图 6.10 DMR 与 Polling 负载效果对比

从图 6.10(a)和图 6.11(b)可以看出，在添加一个流量产生进程后，DRM 算法经过 1 个周期达到两个进程的负载同步，而 Polling 算法则需要 2 个周期。这主要是因为 DRM 算法具有动态反馈功能，能够实时的获得每个进程的真实负载情况，而 Polling 算法只能随着请求的不断完成达到负载同步。此外，从图 6.10(a)和图 6.11(b)还可以看出，在正常情况下，与 DMR 算法相比，Polling 算法在时段上两个进程的 CPU 使用率差值波动性更大。这主要是由每个请求对象的生命周期不等导致的。虽然 Polling 算法能将请求消息“公平”地分配到每个流量产生进程，但是每个请求对象的生命周期并不同。当某个进程分配到较大生命周期的请求消息时，将占用更多的 CPU 资源，反之则较小，从而造成较大的波动。而 DMR 算法

可以通过动态反馈功能来减小这种波动性。通过上述两方面的对比分析可以看出，本文提出的 DMR 算法具有较好的性能。

4. 系统日志功能测试

系统日志记录了系统自启动以来的运行情况。在配置文件中可以设置错误日志文件路径以及日志输出的等级，主要包括严重错误（alert）、一般错误（error）、警告错误（warn）、提醒信息（notice）和调试信息（debug）。当系统生成一条日志时，如果其级别高于或等于系统启动时设置的级别则会输出，否则不输出。系统默认的错误日志文件为 logs/error.log，级别为 notice。

测试方法：采用默认的系统日志设置，首先切换到系统可执行文件所在路径，通过“./ntrig_tile”命令启动系统，然后通过“./ntrig_tile –s stop”命令关闭系统，最后查看错误日志文件的信息。其中部分日志片段如图 6.11 和 6.12 所示

```
2016/03/26 13:06:05 [notice] 21656: level "notice" in /root/.tilera/workspace/ntrig_tile
/conf/ntrig.conf:10
2016/03/26 13:06:05 [notice] 21656: ntrig/1.0
2016/03/26 13:06:05 [notice] 21656: OS: Linux 2.6.40.38-MDE-4.1.2.149467
2016/03/26 13:06:05 [notice] 21656: start virtual processes 21657
2016/03/26 13:06:05 [notice] 21657: process(21657) set the affinity in no.(1) position
2016/03/26 13:06:05 [notice] 21656: start record process 21658
2016/03/26 13:06:05 [notice] 21658: process(21658) set the affinity in no.(2) position
2016/03/26 13:06:05 [notice] 21656: start traffic process 21659
2016/03/26 13:06:05 [notice] 21659: process(21659) set the affinity in no.(3) position
2016/03/26 13:06:05 [notice] 21656: process(21656) set the affinity in no.(0) position
```

图 6.11 系统启动日志片段

```
2016/03/26 13:08:31 [notice] 31053: level "notice" in /root/.tilera/workspace/ntrig_tile/conf/ntrig.conf:10
2016/03/26 13:08:31 [notice] 31053: signal process started
2016/03/26 13:08:31 [notice] 21656: signal 15 (SIGTERM) received, exiting
2016/03/26 13:08:31 [notice] 21658: exiting
2016/03/26 13:08:31 [notice] 21657: exiting
2016/03/26 13:08:31 [notice] 21658: exit
2016/03/26 13:08:31 [notice] 21657: exit
2016/03/26 13:08:31 [notice] 21659: exiting
2016/03/26 13:08:31 [notice] 21656: signal 17 (SIGCHLD) received
2016/03/26 13:08:31 [notice] 21656: record process 21658 exited with code 0
2016/03/26 13:08:31 [notice] 21656: signal 17 (SIGCHLD) received
2016/03/26 13:08:31 [notice] 21656: virtual processes 21657 exited with code 0
2016/03/26 13:08:31 [notice] 21659: exit
2016/03/26 13:08:31 [notice] 21656: signal 17 (SIGCHLD) received
2016/03/26 13:08:31 [notice] 21656: traffic process 21659 exited with code 0
2016/03/26 13:08:31 [notice] 21656: exit
```

图 6.12 系统退出日志片段

从图 6.11 可以看出，由控制进程（21656）依次启动了虚拟用户进程（21657）、日志记录进程（21658）和流量产生进程（21659），并分别绑定到 0~3 号核上。从图 6.12 可以看出，控制进程接收到退出信号 SIGTERM 后向其他进程发送退出消息，待所有子进程都退出后自身才退出。可见，系统日志功能能够按照时间顺序将所有进程的运行过程记录下来。

6.3 系统性能测试

6.3.1 流量的自相似性测试

文献[47]中指出，网络流量具有自相似特征。此外，Web 业务流量作为网络流量中的重要组成部分也同样具有自相似性[48]。因此，Web 流量产生系统能否产生自相似的流量是衡量其是否有效的重要标准之一。

Hurst 指数（H）是度量网络流量自相似性的重要指标，取值范围为$0 \le H \le 1$ 。其中$0.5 \le H \le 1$，表示网络流量具有自相似性，且值越大自相似性越强[49]。计算 Husrt 指数的方法主要有：方差时间法、R/S 分析法、绝对值法、周期图法等[50]，本文采用方差时间法和 R/S 分析法，下面对这两种方法进行简单介绍。

1.方差时间法[51]

假定有 N 个时间序列的观察样本$\left(X_t : t = 1,2,\ldots,N\right)$，将其依次划分成 k 个等长的子序列，并对每个子序列$\left(X_t : t = 1,2,\ldots,N/k\right)$进行如下运算：

a. 样本均值 $\overline{X} = \frac{k}{N}\sum_{i=1}^{N/k} X_i$；

b. 样本方差 $S^2 = \frac{k}{N}\sum_{i=1}^{N/k}(X_i - \bar{X})^2$

c. 取对数 $\log s^2 / \log m$；

用最小二乘法拟合 $\log s^2 / \log m$ 曲线，可得到斜率为$-\beta$的曲线，则 Hurst 指数：$H = 1 - \beta/2$。

2. R/S 分析法[52]

假定有 N 个时间序列的观察样本$\left(X_t : t = 1,2,\ldots,N\right)$，将其依次划分成 k 个等长的子序列，并对每个子序列$\left(X_t : t = 1,2,\ldots,N/k\right)$进行如下运算：

a. 样本均值 $\overline{X} = \frac{k}{N}\sum_{i=1}^{N/k} X_i$；

b. 累积误差 $Y_i = \sum_{m=1}^{i}\left(X_i - \bar{X}\right), \left(i = 1,2,...,N/k\right)$；

c. 样本标准差 $S = \sqrt{\frac{k}{N}\sum_{i=1}^{N/k}(X_i - \bar{X})^2}$；

d. 样本极差 $R=\max_{1\le i\le N/k}(Y_i)-\min_{1\le i\le N/k}(Y_i)$;

计算子序列 R/S，记为 R_t/S_t，称为重整化范围，则 k 个子序列的重整化范围均值：$(R/S)_k=\frac{1}{k}\sum_{i=1}^{k}R_i/S_i$。可知 $(R/S)_k$ 的值随 k 的大小而变化。统计表明，当 $k\to\infty$ 时有：$(R/S)_k\sim cn^H$，对其两边取对数，可得 $\log(R/S)_k=\log c+H\log n$，所以 Hurst 指数为：$H=\frac{\log(R/S)_k-\log c}{\log n}$。

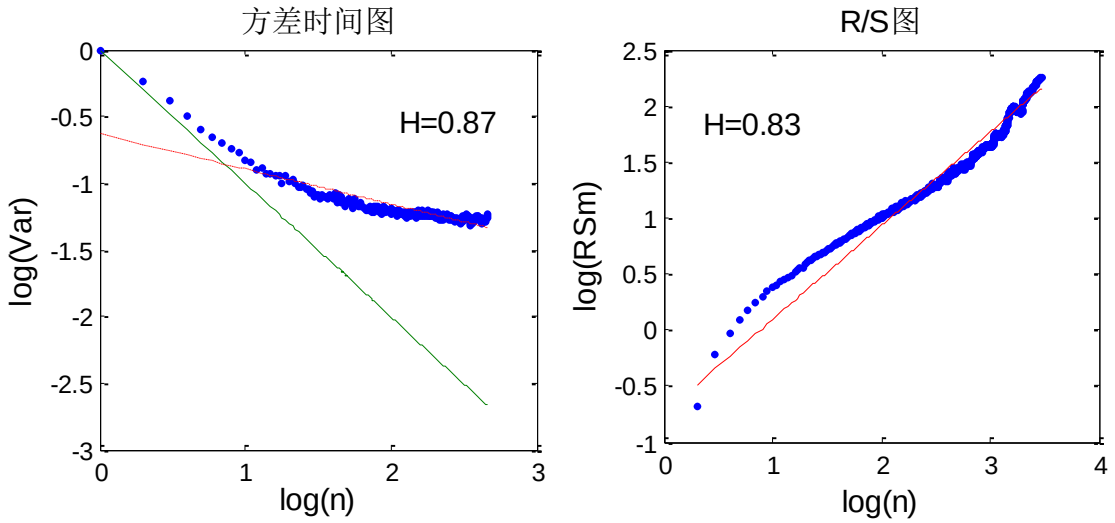


图 6.13 流量产生系统的 Hurst 指数

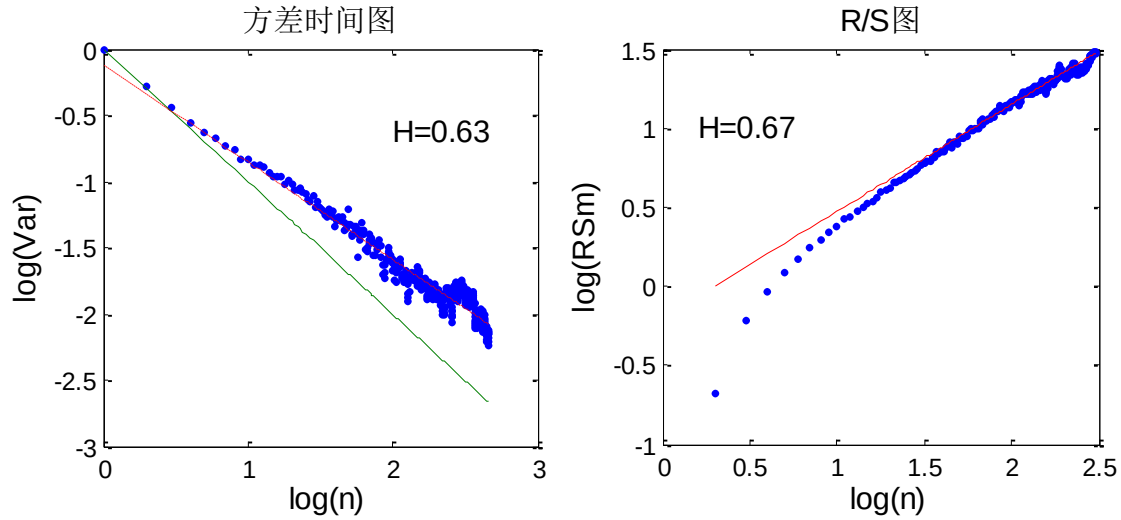


图 6.14 http_load 流量的 Hurst 指数

为了验证本文基于用户行为方式产生的网络流量与压力测试软件产生的网络流量的区别，将选取压力测试软件 http_load 与之对比。具体的测试方法为：首先

对同一个 URL 集合，通过调整两者的负载参数使它们产生的流量保持在 65MB/s 左右（平均流量差别小于 5%）；然后使用 Shell 脚本以 1s 为周期分别采集 2 万个流量数据；最后分别使用方差时间法与 R/S 分析法对这两组数据计算 Hurst 指数，如图 6.13 和图 6.14 所示。

从图 6.13 和图 6.14 中可以看出，本文基于 Web 用户行为的流量产生方法所产生的网络流量较 http_load 产生的流量具有更大的自相似性。文献[23]对广域网的流量自相似性进行了研究，其 Hurst 指数范围通常在 0.71~0.89 之间。可知，本文的流量产生系统的 Hurst 指数是符合这一范围的，http_load 则相差较远。这是因为本文系统采用了基于 ON/OFF 的用户行为模型，多个 ON/OFF 源叠加能够产生比较符合实际网络的流量[53]。而 http_load 流量的弱自相似性则主要是由请求资源的大小和请求频率的较小差异引起。通过上面分析可知，本文的流量产生系统能够产生比较符合实际网络的流量。

6.3.2 大尺度的流量产生测试

为了更好的测试系统产生大尺度流量的性能，将对一个拥有 10 万用户的网络或网站进行大尺度的流量模拟，并根据百度统计的 2016 年 2 月网民时段变化数据进行用户控制。由 3.1.2 节可知，实际网络中不同时段的活跃用户百分比是比较低的。根据用户控制方法，系统可通过 7 千个虚拟用户模拟该网络或网站的流量。图 6.15 和图 6.16 分别给出了预处理后的虚拟用户时段变化曲线及其 240 点的插值结果。本次测试的模拟开始点时刻为 0 点，持续运行 24 小时。通过 shell 脚本以 60 秒为周期进行流量采集，得到流量变化曲线如图 6.17 所示。此外，图 6.18 给出了没有用户控制的流量变化情况。

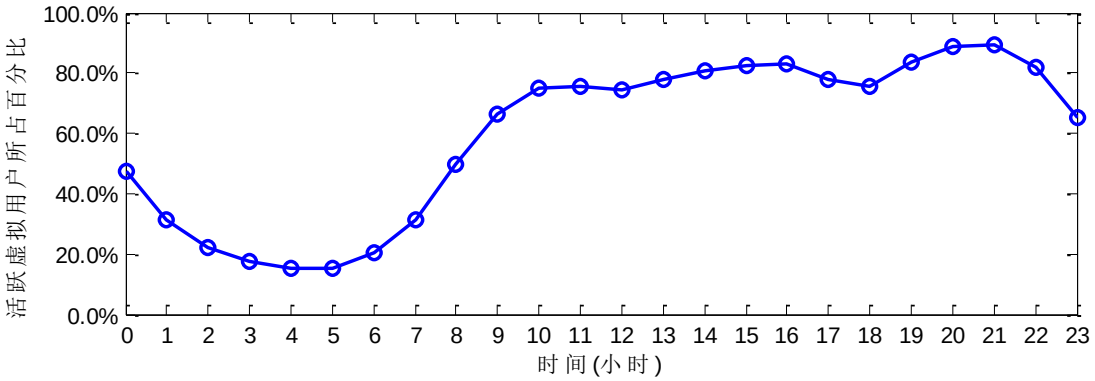


图 6.15 虚拟用户时段变化曲线

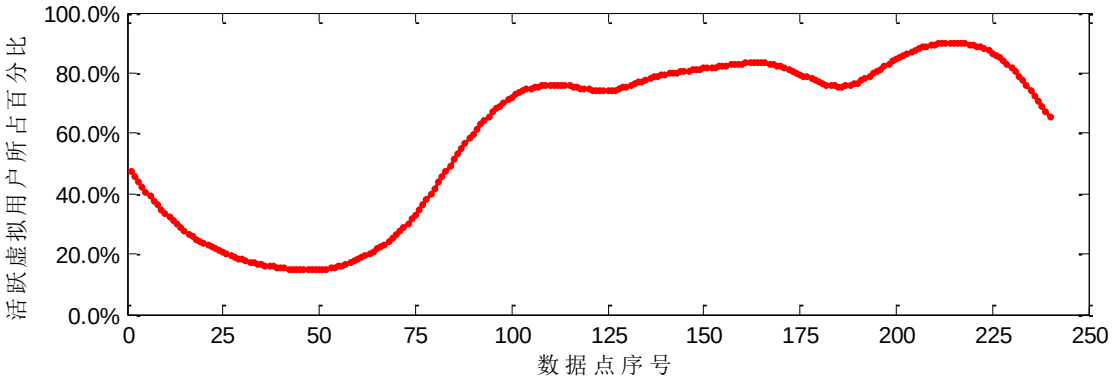


图 6.16 虚拟用户时段变化曲线插值结果（240 个点）

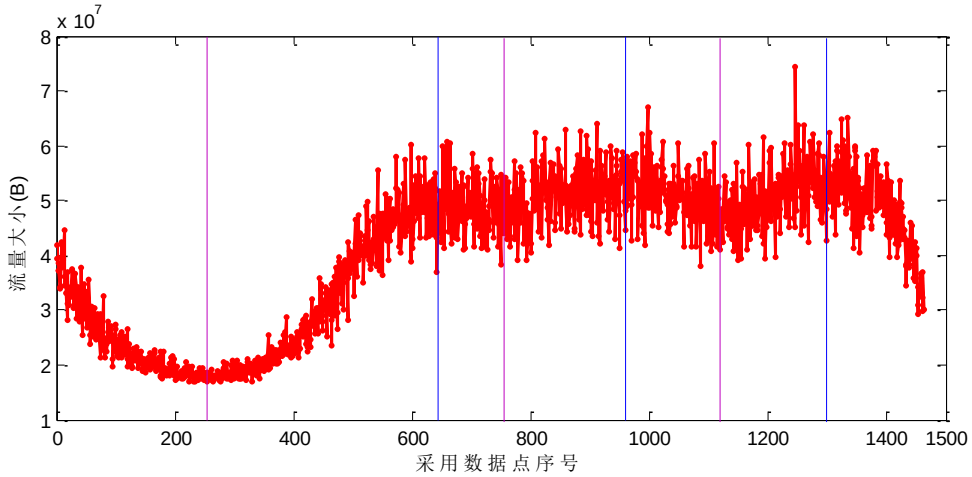


图 6.17 有用户控制的流量变化情况

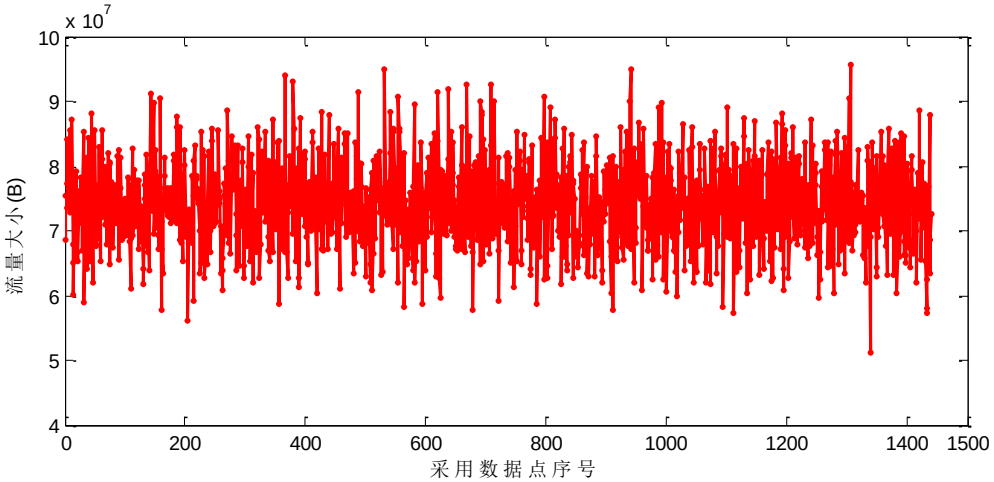


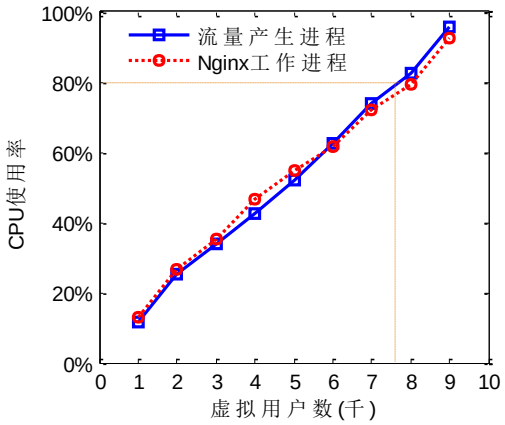
图 6.18 没有用户控制的流量变化情况

从图 6.15 和图 6.16 可以看出，通过三次样条插值算法插值后得到的虚拟用户变化曲线能够很好的保持原曲线的变化趋势。通过图 6.16 和图 6.17 可以看出，产生的流量在保持局部具有较大波动性的情况下，整体流量变化趋势与指定的虚拟

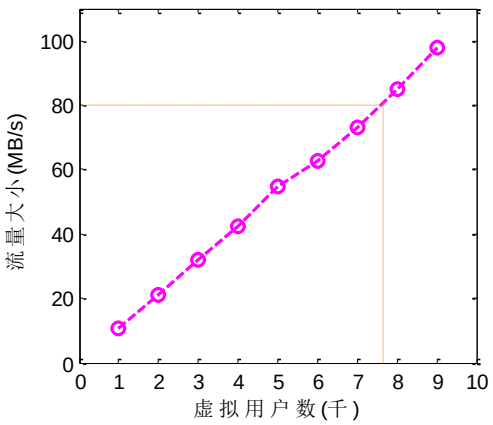
用户变化趋势相吻合。此外，从图 6.17 还可以看出，当活跃用户较少时，流量的波动性较小，反之，流量的波动性增大。由图 6.17 和图 6.18 可知，与有用户控制的流量变化相比，没有进行用户控制的流量虽然在局部也有比较好的波动性，但是整体上呈现直线趋势。通过上面的分析可知，本系统能够通过控制虚拟用户数有效地实现大时间尺度下流量的产生，使系统能够产生在大尺度下更加符合实际网络的流量。

6.3.3 并发性能测试

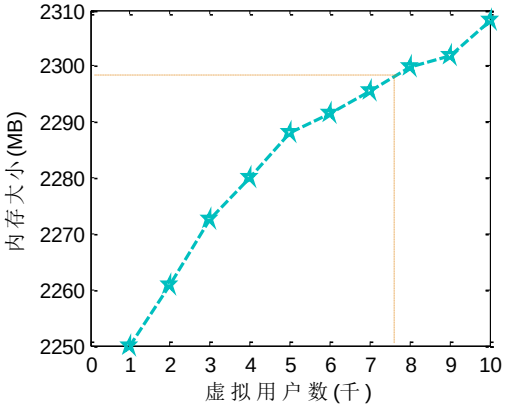
系统的并发性能是衡量系统设计是否有效地重要指标之一。为了有效地测试系统的并发性能，将关闭系统的场景控制，让所有虚拟用户处于活跃状态。测试主要包括两个方面：单个流量产生进程的并发性能测试和不同流量产生进程数目下支持的虚拟用户数测试。



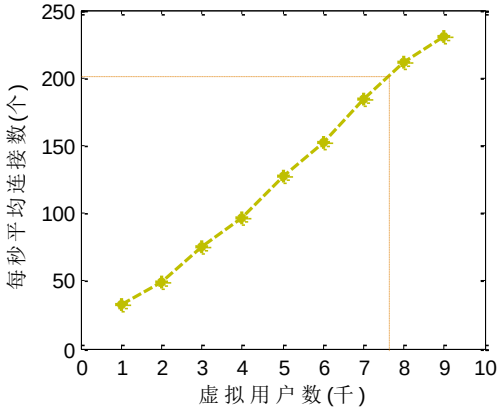
(a) 进程 CPU 使用率对比



(b) 流量变化情况



(c) 内存变化情况

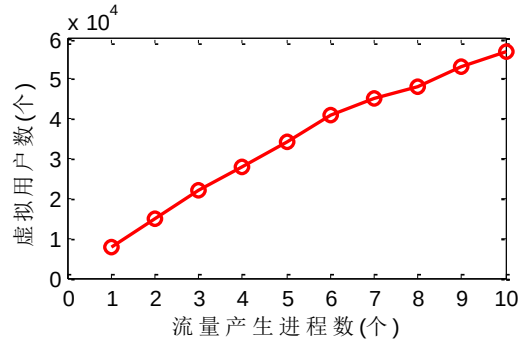


(d) 每秒连接数变化情况

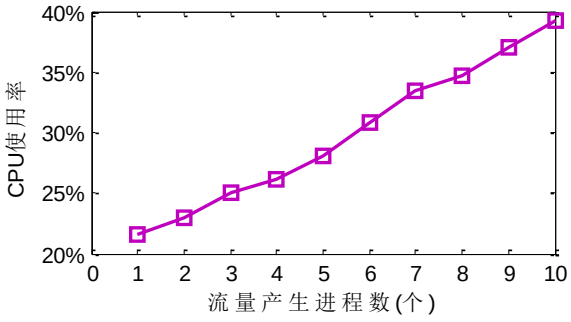
图 6.19 流量产生进程与 Nginx 工作进程并发性能对比

本文选取同样采用事件驱动实现高并发的 Nginx 作为测试用的 Web 服务器，以充分测试单个流量产生进程的并发性能。为了保证平台的一致性，将 Nginx 安装在另一个 Tile-Gx36 平台，并开启单工作进程模式。其测试结果如图 6.19 所示。其中，图 6.19(a)给出了随着用户数增加，流量产生系统的流量产生进程与 Nginx 的工作进程的 CPU 使用率对比情况；图 6.19(b)给出了随着用户数增加，系统产生的流量变化情况；图 6.19(c)给出了随着用户数增加，系统消耗的内存情况；图 6.19(d)给出了随着用户数增加，每秒平均的连接数变化。

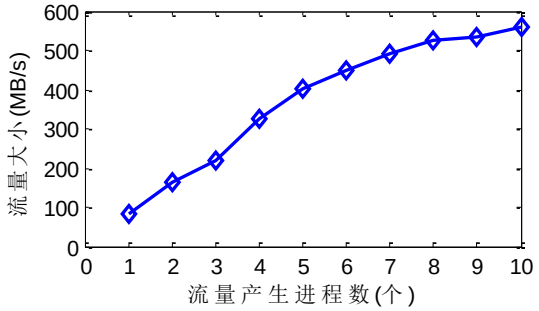
从图 6.19(a)可以看出，在不同的虚拟用户数目情况下，流量产生进程都能与 Nginx 工作进程在 CPU 使用率上保持基本相同，即两者的并发性能基本一致。从图 6.19(b)和图 6.19(d)可以看出，产生流量的大小和每秒平均的连接数基本上随虚拟用户数目成线性增长，且系统每秒的并发连接数是比较低的，这主要是因为系统采用了 HTTP 的长连接通信方式。从图 6.19(c)可以看出，系统消耗的内存是比较小的，且随用户数的增加而增加。此外，当 CPU 使用率在 80%左右时，单个流量产生进程能够支持超过 7 千个活跃虚拟用户的并发访问，产生的流量大小约为 80MB/s（640Mb/s），总共消耗的内存约为 2.3GB。



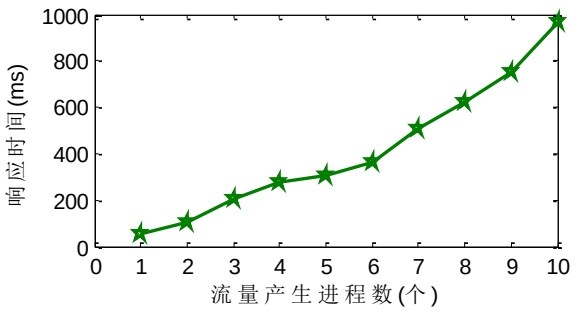
(a) 虚拟用户数变化情况



(b) 虚拟用户进程的 CPU 使用率



(c) 流量变化情况



(d) 服务器响应时间

图 6.20 虚拟用户数测试结果

通过上面的分析可知，单个流量产生进程具有较高的并发性能。下面在不同流量产生进程数目情况下，对系统所能支持的虚拟用户数进行测试。测试思路为：通过改变流量产生进程的数目，测试所有流量产生进程的平均 CPU 使用率为 80%时，虚拟用户数目、虚拟用户进程的 CPU 使用率、产生的流量以及服务器响应时间的变化情况。图 6.20 给出了在服务器响应时间小于 1s 情况下的测试结果。

从图 6.20 可以看出，虚拟用户数目、虚拟用户进程的 CPU 使用率、产生的流量以及服务器响应时间都随流量产生进程数的增加而增加。但从图 6.20 (a)可以看出，系统模拟的虚拟用户数并不没有随流量产生进程数的增加呈线性增长，可能的原因是随着模拟用户数的增加，产生的流量不断增大，引起服务器的响应时间变长，如图 6.20(c)所示。

综上所述，流量产生系统具有较好的整体并发性能，能同时模拟超过 5 万个用户，产生的流量高达 4Gbps。

6.3.4 稳定性测试

系统的稳定性测试主要为了测试系统在较大负载情况下能否稳定的运行。具体的测试思路如下：系统启动时开启 20 个流量产生进程，模拟用户数设置为 5 万。同时关闭系统场景控制，让所有虚拟用户一直处于活跃状态。系统持续运行 24 小时。通过 shell 脚本对系统产生的流量、系统流量产生进程的平均 CPU 利用率、内存使用情况以及整个 Tile-Gx36 平台的系统负载进行统计。统计方法为：每秒采集一个数据，然后每 150 个数据点求平均值得到测试结果，如图 6.21 所示。由图 6.21 可以得出以下几点结论：

(1) 流量产生系统能够比较稳定的产生流量。从连续运行的 24 小时的流量情况可以看出，该系统可用于需要长时间产生背景流量的应用场景。

(2) 系统的 CPU 使用率并没有随着时间的推移而增加，虽然不断上下波动，但始终保持在一个比较平稳的状态。

(3) 系统的内存随着时间的推移没有出现内存泄漏的情况，且波动性也比较小，幅度不超过 50MB。这主要是因为系统对经常使用的一些对象资源（如虚拟用户对象、请求对象等）都采用了“池技术”，在系统启动时就进行了内存预分配。

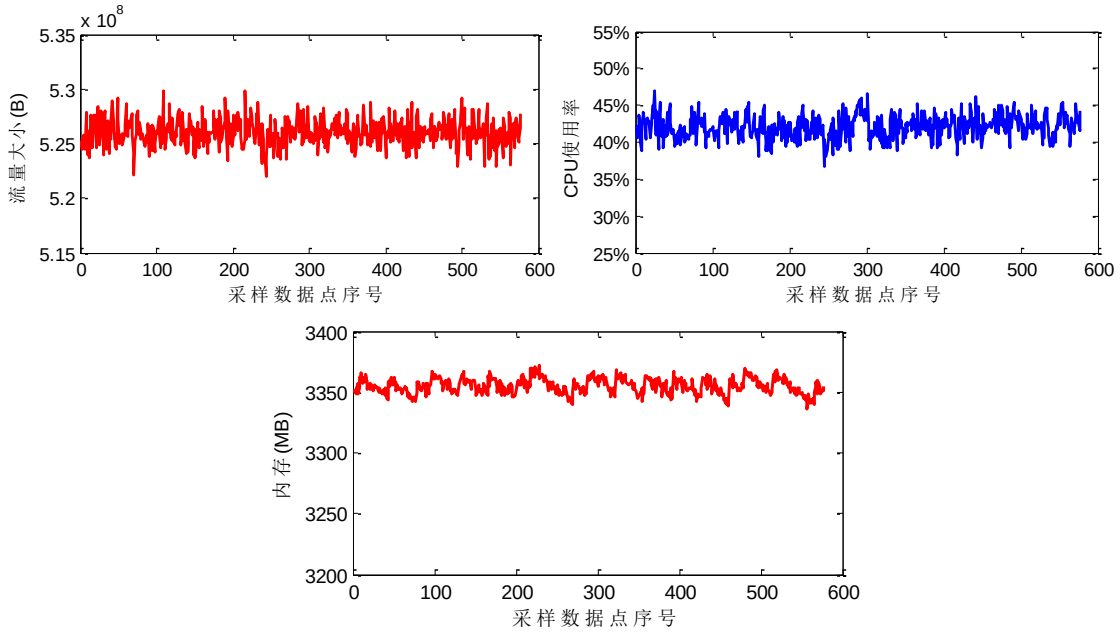


图 6.21 系统稳定性测试结果

综上所述，该系统具有较强的稳定性，可以长期且稳定的产生较大的流量，且随着运行时间的增加，系统的资源消耗相对稳定并不会出现内存泄露、系统崩溃等情况。

6.4 本章小结

本章从系统功能和系统性能两个方面对系统进行了较全面的测试。其中，系统功能测试包括两个方面：前端功能测试和后端功能测试；系统性能测试包括：流量的自相似性测试、大尺度的流量产生测试、并发性能测试以及系统稳定性测试。通过上述系统测试表明本文的 Web 流量产生系统在功能和性能上均达到设计目标，可满足实验网络对 Web 背景流量的需求。

第 7 章 总结与展望

7.1 总结

流量产生器是实验网建设中不可缺少的工具之一。而现有的流量产生器难以直接应用到实验网。针对上述情况，本文在国家 973 项目“未来互联网性能评估与实验验证”的资助下，设计并实现了基于 Tilera 众核平台的 Web 流量产生系统。该系统借助 Tilera 众核平台，采用基于事件驱动的并发技术，解决了基于软件实现的 Web 流量产生器产生流量速率过低的问题，同时又保留了其灵活性和可扩展性，能够满足实验网中的测试要求。现将本文的研究工作及取得成果总结如下：

(1) 采用分层设计方法，提出了一种包含控制层、虚拟用户层和流量产生分层的流量产生系统软件架构。该软件架构将用户模拟功能与流量产生功能进行分离设计，提高了系统的可扩展性。

(2) 在结合基于 ON/OFF 模型的 Web 用户行为模拟方法的基础上，本文提出了一种基于三次样条插值算法的用户控制方法。该方法能够通过控制虚拟用户数有效地实现不同场景下大时间尺度的流量产生。

(3) 对提高系统并发性能的方法进行了研究，从 Linux 内核参数优化、基于事件驱动的并发和基于众核平台的系统并行化三个方面提升了流量产生系统的并发性能。

(4) 基于 Tilera 众核平台完成了系统的并行化设计并实现了基于众核平台的 Web 流量产生系统。

(5) 完成了 Web 流量产生系统的功能和性能测试。测试结果表明，该系统能够同时模拟 5 万个用户访问 Web 服务器，并且系统的并发性、稳定性以及产生流量的真实性等均达到设计目标。

7.2 展望

由于时间与能力有限，本文仍存在一些不足之处，下一步将从以下几个方面进行研究与改进。

(1) 完善前端系统。目前前端系统只提供了系统必须的一些核心功能，且页面布局和设计都比较简单。今后将对前端系统进行优化升级，提升实验者的用户体验。

(2) 目前已完成了系统的本地开发与测试，下一步将与北京邮电大学的CENI(China Environment for Network Innovations)实验平台进行融合，对系统进行进一步的开发与测试。

(3) 目前本系统只实现了 Web 流量的产生，今后将对系统进行扩展，使其能够产生多种业务流量，比如 P2P 流量、视频流量等。

在论文的撰写工作中，由于本人学识和水平有限，论文难免有一些错误和不足之处，恳请各位老师批评指正。

参考文献

[1] Botta A, Dainotti A, Pescapé A. A tool for the generation of realistic network workload for emerging networking scenarios[J]. Computer Networks, 2012, 56(15) : 3531-3547.

[2] 陈远知,杨帆. Tilera多核处理器网络应用研究[A]. 中国高科技产业化研究会信号处理专家委员会.全国第五届信号和智能信息处理与应用学术会议专刊(第一册)[C].中国高科技产业化研究会信号处理专家委员会,2011:4.

[3] Feknous M, Houdoin T, Le Guyader B, et al. Internet traffic analysis: A case study from two major European operators[C]//Computers and Communication (ISCC), 2014 IEEE Symposium on. IEEE, 2014: 1-7.

[4] Zinke J, Habenschuss J, Schnor B. servload: Generating representative workloads for web server benchmarking[C]//Performance Evaluation of Computer and Telecommunication Systems (SPECTS), 2012 International Symposium on. 2012:1-8.

[5] Cheng Y C, Hölzle U, Cardwell N, et al. Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying.[C]//General Track: 2004 Usenix Technical Conference, June 27 - July 2, 2004, Boston Marriott Copley Place, Boston, Ma, Usa. 2004:87-98.

[6] 王会霞,石磊,卫琳,石云. Web缓存流量特征模型研究和应用[J]. 计算机应用,2007,04:776-779.

[7] Lee J J, Gupta M. A new traffic model for current user web browsing behavior[J]. Intel corporation, 2007.

[8] Xu L, Zhang W, Chen L. Modeling Users' Visiting Behaviors for Web Load Testing by Continuous Time Markov Chain[C]//Web Information Systems and Applications Conference. IEEE, 2010:59-64.

[9] Pries R, Magyari Z, Tran-Gia P. An HTTP web traffic model based on the top one million visited web pages[C]//Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on. 2012:133-139.

[10] Spirent Avalanche[EB/OL]. URL:[2016-03-20]. http://www.spirent.cn/Products/Avalanche.

[11] IxLoad[EB/OL]. URL:[2016-03-20]. http://www.ixiacom.cn/products/applications/Ixload.

[12] Thorndale C W. Webstone: The first generation in http server benchmarking[J]. Geophysical Prospecting, 1995, 29(4):541–549.

[13] Busari M, Williamson C. ProWGen: a synthetic workload generation tool for simulation evaluation of web proxy caches[J]. Computer Networks, 2002, 38(6):779-794.

[14] Katsaros K V, Xylomenos G, Polyzos G C. GlobeTraff: a traffic workload generator for the performance evaluation of future Internet architectures[C]//New Technologies, Mobility and Security (NTMS), 2012 5th International Conference on. IEEE, 2012: 1-5.

[15] Barford P, Crovella M. Generating representative Web workloads for network and server performance evaluation[J]. Acm Sigmetrics Performance Evaluation Review, 1998, 26(1):151-160.

[16] Kant K, Tewari V, Iyer R K. Geist: a generator for e-commerce & internet server traffic[C]//ISPASS. 2001: 49-56.

[17] 洪春涛. 众核处理器编程模式关键技术研究[D]. 清华大学,2011.

[18] 甘新标. 面向众核GPU的编程模型及编译优化关键技术研究[D]. 国防科学技术大学,2012.

[19] 李晨. 基于多核的网络安全测试设备的通用框架设计[D]. 西安电子科技大学, 2011.

[20] 张超. 基于TILE64的H.264多线程并行编码[D]. 西安电子科技大学, 2011.

[21] 韩笑. 基于TILE Pro64多核处理器的3G服务器视频转码软件设计[D]. 浙江大学, 2012.

[22] 杨三胜. Tilera多核环境下基于NetFlow的P2P协议识别与检测[D]. 西安工程大学, 2011.

[23] 朱海婷, 丁伟. 面向传输层协议的网络流量Hurst指数分析[C]//中国教育和科研计算机网cernet学术年会. 2010.

[24] Choi H K, Limb J O. A Behavioral Model of Web Traffic[C]//International Conference on Network Protocols. IEEE Computer Society, 1999:327-334.

[25] Liu Z, Niclausse N, Jalpa-Villanueva C. Traffic model and performance evaluation of Web servers[J]. Performance Evaluation, 2001, 46(2-3):77-100.

[26] Lee J J, Gupta M. A New Traffic Model for Current User Web Browsing Behavior[J]. Intel Corporation, 2007.

[27] Pries R, Magyari Z, Tran-Gia P. An HTTP web traffic model based on the top one million visited web pages[C]//Next Generation Internet (NGI), 2012 8th EURO-NGI Conference on. 2012:133-139.

[28] Zhu C, Wang Y, Zhang Y, et al. Different Behavioral Characteristics of Web Traffic between Wireless and Wire IP Network[C]//International Conference on Communication Technology Volume 1 of. 2003:267-271 vol.1.

[29] Bianco A, Mardente G, Mellia M, et al. Web user session characterization via clustering techniques[C]//Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE. IEEE, 2005.

[30] Bianco A, Mardente G, Mellia M, et al. Web user-session inference by means of clustering techniques.[J].Networking IEEE/ACM Transactions on, 2009, 17(2):405-416.

[31] 中国网站排名网. 网站流量分析报告[EB/OL]. URL:[2016-03-21]. http://www.Chinarank.org.cn/.

[32] 中国互联网络信息中心. 第25次《中国互联网络发展状况统计报告》[EB/OL]. URL:[2016-03-21]. http://www.isc.org.cn/.

[33] 百度统计[EB/OL]. URL:[2016-03-21]. http://tongji.baidu.com/data/hour.

[34] 刘永春, 王强. 三次样条插值函数的新解法[J]. Pure Mathematics, 2013, 03(06):362-367.

[35] 李庆扬 王能超. 数值分析(第4版)[M]. 华中科技大学出版社(原华中理工出版社)出版社, 2006:13-43.

[36] 赵永刚. linux系统优化大师[M]. 电子工业出版社, 2015:1-296.

[37] Nginx[EB/OL]. URL:[2016-03-21]. http://www.nginx.org/.

[38] Memcached[EB/OL]. URL:[2016-03-21]. http://memcached.org/.

[39] ACE[EB/OL]. URL:[2016-03-21]. http://www.cs.wustl.edu/~schmidt/ACE.html.

[40] libevent-an event notification library[EB/OL]. URL:[2016-03-21]. http://libevent.org/.

[41] Wentzlaff D, Griffin P, Hoffmann H, et al. On-Chip Interconnection Architecture of the Tile Processor[J]. IEEE Micro, 2007, 27(5):15-31.

[42] W.Richard Stevens, 杨继张. UNIX网络编程第2卷:进程间通信[M]. 清华大学出版社, 2000:22-275.

[43] Tilera Corporation. Programming the TILE-Gx Processor,UG505[M]. 2012:7-16.

[44] 罗拥军, 李晓乐, 孙如祥. 负载均衡算法综述[J]. 科技情报开发与经济, 2008, 18(23):134-136.

[45] 杨振海, 张国志. 随机数生成[J]. 数理统计与管理, 2006, 25(2):244-252.

[46] Professor Karl Sigman's Lecture Notes on Simulation[EB/OL]. URL:[2016-01-27]. http://www.columbia.edu/~ks20/4404-Sigman/Simulation-Sigman.html.

[47] Thompson K, Miller G J, Wilder R. Wide-area Internet traffic patterns and characteristics[J]. IEEE Network, 1997, 11(6):10-23.

[48] Crovella M E, Bestavros A, Crovella M E, et al. Self-similarity in World Wide Web traffic[C]//ACM SIGMETRICS International Conference. 1996:160-169.

[49] 朱海婷, 丁伟. 面向传输层协议的网络流量Hurst指数分析[C]//中国教育和科研计算机网cernet学术年会. 2010.

[50] 陈建, 谭献海, 贾真. 7种Hurst系数估计算法的性能分析[J]. 计算机应用, 2006, 26(4):945-947.

[51] Zhang H F, Shu Y T, Yang O. Estimation of Hurst parameter by variance-time plots[C]//Communications, Computers and Signal Processing, 1997. 10 Years PACRIM 1987-1997-Networking the Pacific Rim. 1997 IEEE Pacific Rim Conference on. 1997:883-886 vol.2.

[52] 傅雷扬, 王汝传, 王海艳,等. R/S方法求解网络流量自相似参数的实现与应用[J]. 南京航空航天大学学报, 2007, 39(3):358-362.

[53] Pruthi P, Erramilli A. Heavy-Tailed ON/OFF Source Behavior and Self-Similar Traffic[C]//Communications, 1995. ICC '95 Seattle, 'Gateway to Globalization', 1995 IEEE International Conference on. IEEE, 1995:445--450.

致谢

时光如流水，三年的硕士研究生生涯眨眼间即将过去。三年的时间感觉自己又成熟了许多，研究生阶段收获了太多。在即将完成学业之际，我想对在硕士研究生期间给予我帮助的各位老师说声感谢，对给予我关心的各位同学们说声谢谢！

首先，感谢唐红老师，感谢您三年来对我的教导，无论是在学术研究上，还是在做人做事方面，您都给予了很大帮助。您渊博的学识及对学生的负责任给我留下了深刻的印象，在今后的工作中我也会向您学习。正是您的帮助，才让我顺利完成毕业论文的撰写工作。

感谢赵国锋老师一直以来对我的严格要求和悉心指导，从赵老师身上我学到了学术研究要一丝不苟，您认真而严谨的科研态度使我终身难忘。感谢曾老师一年多来的耐心指导和关心，让我攻克一个又一个项目难题。感谢张毅老师，张老师教会我如何克服学术和生活中的困难，如何紧跟科技的前沿，努力向前。感谢徐川老师，徐老师每天都会和我们一起奋斗，您勤奋好学的态度是我们学习的榜样。

另外，感谢我实验室的同窗们，在困难的时候给我帮助，每次有解决不了的任务，一起思考，一起解决。大家共同学习，共同进步，在这里的每一天，都是充实的，都是快乐的。感谢我的父母，感谢你们对我的支持，感谢有你们的陪伴。

最后，谨向参加我论文答辩的各位老师表示衷心的感谢和祝福！

攻读硕士学位期间从事的科研工作及取得的研究成果

参与的科研项目：

[1] 未来互联网性能评估与实验验证(2012CB315806), 国家重点基础研究发展计划(973 计划), 2014.3~2016.8.

[2] 基于用户行为模型的高强度业务流量生成系统及试验验证(BY2013095-5-07), 2014.1~2015.12.

发表的论文、著作和专利：

[1] 曾帅, **唐小军**, 殷志坚等. 一种基于事件驱动的高并发WEB流量产生器. 中国发明专利, 201510849671.X[P]. 2015.11.30.

[2] 曾帅, **唐小军**, 殷志坚等. Web 流量产生系统 V1.0. 国家计算机软件著作权, 2015SR159932[Z]. 2015.08.18.

[3] 曾帅, 殷志坚, **唐小军**等. 一种基于多种网络数据业务的并发式流量发生系统. 中国发明专利, 201510196858.4[P]. 2015.04.24.

[4] 曾帅, 高宗彬, 殷志坚, **唐小军**等. 一种基于众核平台的流媒体流量发生系统中国发明专利,. 201510569417.4[P]. 2015.09.10.

[5] 曾帅, 高宗彬, 殷志坚, **唐小军**等. 网络流量发生器管控系统 V1.0. 国家计算机软件著作权, 2015SR159717[Z]. 2015.03.15.

[6] 曾帅, 高宗彬, 殷志坚, **唐小军**等. 流媒体流量发生系统 V1.0. 国家计算机软件著作权, 2016SR005957[Z].2015.10.15.