申请上海交通大学硕士学位论文

基于 Clearwater 的虚拟网络功能资源映射模型研究

论文作者 _____

学　　号 _____

导　　师 _____

专　　业 软件工程

答辩日期 _____

Submitted in total fulfillment of the requirements for the degree of Master
in Software Engineering

Research on Resource Mapping Model of Network Function Virtualization on Clearwater Platform

Depart of Software Engineering, School of Electronics and Electric Engineering
Shanghai Jiao Tong University
Shanghai, P.R.China

Dec. 17th, 2018

基于 Clearwater 的虚拟网络功能资源映射模型研究

摘 要

网络功能虚拟化 (NFV) 利用虚拟化的基础设施展现了业界所期待的敏捷性和可扩展性，但是软件化网络服务的性能仍然是阻碍其大规模普及应用的瓶颈之一。与传统虚拟化应用不同，大多数网络功能虚拟服务是由多个虚拟机 (VMs) 以服务链 (SFC) 的形式运行在标准的大容量服务器中。每台虚拟机中运行着专用的虚拟网络功能 (VNF)，现有的虚拟机监视器 (Hypervisor) 仍然将这些虚拟机视作单独的虚拟机实例来为分配资源和维护。当一条服务链被请求服务时，现有的虚拟化资源管理缺少对每个所需要的 VNF (VM) 之间逻辑连接的识别，独立地为服务链中每个实例映射物理资源。这样的资源映射方式会由于多核物理机底层架构的原因 (如 NUMA) 导致组成的服务链的性能波动。在这样的背景下，如何提升大容量多核服务器的资源利用效率成为了一个亟待解决的挑战。

IP 多媒体子系统 (IMS) 作为一项成熟的基于互联网协议提供多媒体服务的电信架构，广泛的应用于通信运营商的解决方案中。IMS 服务也是由多个不同的功能组根据不同的业务需求组织成特定的服务链来提供的。同样的，IMS 系统作为一类典型的 NFV 业务部署在虚拟化的环境中时也面临着缺乏从服务链角度来进行资源映射的问题。

为了解决这个问题，本文提出了针对服务链的资源映射模型和优化映射算法，结合实际的 IP 多媒体子系统 Clearwater 服务，实现本文的优化框架原型。通过采样底层服务器的性能信息作为模型和算法的输入，生成优化后的映射策略。随后，根据生成的映射策略对实际参与的服务网络功能实例进行组链，提供所需的网络服务。

最后，在实验室的真实网络环境下对本文所提出的设计与默认的资源映射方法进行对比和性能评估。数据分析表明，本设计可以减少平均 40% 的服务延迟，并提升接近 3 倍的网络吞吐。通过这些实验数据可以看出，本文所提出的优化原型在 Clearwater 系统下可以有效地提升服务的性能。

关键词： 网络功能虚拟化 IP 多媒体子系统 网络功能服务链 资源映射

Research on Resource Mapping Model of Network Function Virtualization on Clearwater Platform

ABSTRACT

Network Function Virtualization (NFV) shows agility and scalability in industry when deployed in cloud computing infrastructure, but the performance of softwarized network service still impedes its popularization. Most of the network service in cloud run as service function chain (SFC) in Standard High Volume Server, consisting of virtual machines (VMs) in tandem. Each VM runs a dedicated virtual network function (VNF) in SFC. The hyper-visor treats these VM with running VNF as single instance to maintain. When a certain SFC is called for serving, the resource manager lacks the recognition of details on the connections between running VFs and separately allocate the resource which consequently lead to low efficient resource utilization for SHVS. This kind of resource allocation will result in performance fluctuation in SFC. How to optimize the SHVS resource utilization for chain-style NFV application poses a great challenge for us.

IMS (IP Multimedia Subsystem) is an architectural framework for delivering IP multimedia services which has been broadly applied in communication solutions. Inspired by the idea of NFV, the service providers and open source communities have released a bunch of projects combined with the support of virtualization techniques and cloud computing. The IMS system comprises several function groups which are linked by specific orders to fulfill the service requirements. This form of organization serves as the service function chain. To deploy in the virtualized environment, the IMS system also meets the challenges of performance fluctuation and resource utilization.

To resolve this problem, in this paper, we propose a new resource abstraction considering the chain-style NFV workload with underlying characteristics and design a greedy algorithm based on our model to map resource. We implement our framework in real IP Multimedia IMS platform called Clearwater.

At last, the framework is tested and compared to existing generic methods. By the examination statistics, our framework can decrease 40% average IMS service latency and improve the

throughput within the SFC up to 300%.

KEY WORDS: NFV, IMS, Clearwater,Service Function Chain, Multi-core Server

目 录

插图索引 **vi**

表格索引 **vii**

算法索引 **viii**

第一章 绪论 **1**
1.1 主要研究内容及背景 1
1.2 国内外研究现状 3
1.2.1 网络功能服务链 4
1.2.2 网络 I/O 优化 4
1.2.3 NFV 资源分配问题 5
1.2.4 其他相关研究 6
1.3 论文安排 7
1.4 本章小结 7

第二章 网络功能虚拟化相关技术 **8**
2.1 网络功能虚拟化发展概况 8
2.2 Clearwater 平台介绍 10
2.3 常见的网络 I/O 虚拟化技术 13
2.3.1 设备模拟模式 13
2.3.2 分离驱动模式 14
2.3.3 硬件辅助模式 15
2.4 非一致性存储访问架构 17
2.5 多核物理机虚拟化应用性能观测与分析 18
2.6 本章小结 21

第三章 IMS 服务链资源映射分析与建模 **22**
3.1 问题概述 23
3.2 关键概念及模型定义 24

3.2.1 相关建模定义 25
3.2.2 网络服务约束定义 26

第四章 方案设计与实现 **29**
4.1 总体设计概述 29
4.1.1 总体设计目标 29
4.1.2 总体设计思路 29
4.1.3 总体设计架构 30
4.2 子模块设计与实现 32
4.2.1 Clearwater 服务链分析 32
4.2.2 信息采样模块 33
4.2.3 动态映射模块 35
4.3 本章小结 41

第五章 实验分析 **42**
5.1 实验平台和测试环境 42
5.2 Clearwater 压力测试实验 45
5.3 基础网络性能对比实验 47
5.3.1 Ping 实验 47
5.3.2 iPerf 实验 47
5.3.3 ApacheBench 实验 52
5.4 本章小结 54

第六章 全文总结 **55**
6.1 主要结论 55
6.2 研究展望 56

参考文献 **57**

攻读学位期间发表的学术论文 **61**

攻读学位期间申请的专利 **62**

插图索引

1–1 云环境中的 IMS 服务 2

2–1 NFV 标准框架 9
2–2 IMS 系统架构图 10
2–3 Clearwater 系统架构图 11
2–4 设备模拟模式示意图 14
2–5 Split-driver 模式示意图 15
2–6 硬件辅助模式示意图 16
2–7 Intel Haswell 架构非一致性存储访问示意图 17
2–8 模拟应用性能测试示意图 18
2–9 模拟应用性能观测结果 19

3–1 Clearwater 用户注册服务流程图 22
3–2 Clearwater 用户注册服务链资源映射示意图 24

4–1 映射优化系统架构示意图 31
4–2 Clearwater 核心业务服务链 33
4–3 使用 PCM 工具获取运行时负载信息 35
4–4 动态映射模块工作流程 37
4–5 Enum 配置文件信息 40

5–1 实例分布示意图 44
5–2 ClearWater 压力测试结果图 46
5–3 Ping 测试结果图 48
5–3 Ping 测试结果图（续） 49
5–4 iPerf 测试结果图 50
5–4 iPerf 测试结果图（续） 51
5–5 ApacheBench 测试结果图 53
5–5 ApacheBench 测试结果图（续） 54

表格索引

2–1 IMS 服务核心网络功能组 12
2–2 验证性实验基准测试工具 20
2–3 模拟应用性能测试结果数据分析（一） 20
2–4 模拟应用性能测试结果数据分析（二） 20

3–1 符号表 28

4–1 服务器实时监测参数 34

5–1 实验平台配置参数表 43

算法索引

4–1 基于贪心的映射策略 38

第一章 绪论

1.1 主要研究内容及背景

网络功能虚拟化 (Network Function Virtualization，NFV) 是由欧洲电信标准组织 (ETSI) 从网络运营商的角度出发提出的一种软件和硬件分离的架构，是当前学术界和工业界十分热门的话题之一。其实质是通过标准化的 IT 虚拟化技术，采用业界标准的大容量服务器、存储和交换机承载各种各样的网络软件功能，实现软件的灵活加载，从而实现可以在数据中心、网络节点和用户端等不同位置的灵活部署。NFV 利用标准 IT 虚拟化技术来加速网络运营商和服务提供商的服务更新，随着近些年来学术界和工业界的携手推进，NFV 正在逐步地演进并替代传统的通信服务平台。如今的网络是仍然由各不相同的网络物理设备彼此相连从而实现特定的服务功能，但这样的架构恰恰成为了网络服务更新的掣肘。NFV 的目标是使用虚拟的网络功能来替代这些特定的网络设备，在 x86 等通用性硬件上利用虚拟机化技术来承载大量的网络功能软件，从而降低昂贵的网络设备成本。通过软硬件结构及功能抽象，NFV 技术使得网络设备功能不再依赖专用硬件，资源可以充分灵活共享，实现新业务的快速开发，并基于实际业务需求进行自动部署、弹性伸缩、故障隔离和自愈等具体目标 [1]。

在 NFV 的应用场景中，网络功能服务链 (Service Function Chain, SFC) 因为其接近实际应用场景的特点，受到众多研究的重点关注 [2–4]。在传统网络中，负载均衡器、网关、虚拟防火墙等网络功能共同被称为业务功能点 (Virtual Network Functions, VNFs)，而流量在经过了一系列处理后，形成了链式的链式转发拓扑结构即所谓的网络功能服务链。在 NFV 的应用背景下，各个业务功能点实际上被抽象为软件运行在虚拟化环境 (虚拟机或容器) 中。与软件定义网络中流量调度方式不同，网络功能服务链的方式更倾向于通过对虚拟网络中各网络功能 (虚拟机或容器) 的本身进行组合，即以更为灵活的方式实现流量到业务功能点的调配和处理。在网络功能服务链中，网络的转发效率决定了整体网络的性能，而虚拟化资源的映射会影响虚拟网络功能之间实际的通信效率，所以如何优化虚拟化资源映射对于提升 NFV 应用性能具有重要意义。

在 NFV 的具体实施中，多媒体子系统 (IP Multimedia Subsystem, IMS) 是较为成熟的一项具体业务，其中更是涌现了众多高质量的达到电信运营级别的开源项目，如 Clearwater [5]，Kamalio [6] 等。在 IMS 的应用场景下，以 Clearwater 为例，传统通信的业务与云计算虚拟化技术深度结合，如图 1–1 所示，各网络功能如代理呼叫会话控制功能 (Proxy-Call Session Control Funtion, P-CSCF)、询问和服务会话控制功能 (Interrogat-

ing&Serving Session Control Funtion, I/S-CSCF)、SIP 服务器 (Application Server, AS) 等等，以软件的形式运行在特定的虚拟机中，进而由特定功能的虚拟机组成虚拟网络功能的资源池，从而实现网络功能软件化和资源池化的目标。有别于传统的基于专有硬件节点的静态组链 (Static Chaining)，虚拟化环境中的 IMS 通过动态地组建网络功能实例形成服务链 (Dynamic Chaining) 来提供 IMS 服务，这样的做的好处是可以依托云计算技术和虚拟化的基础设施，实现服务的快速部署、弹性伸缩。但是，由于 IMS 业务有其链式服务的特性，各功能节点之间存在逻辑上的链接关系，这对虚拟化基础设施的资源映射提出了新的挑战。

首先，传统的虚拟化应用是以单个虚拟机为中心的，缺乏多个虚拟机协同合作的应用场景，从而缺乏相应的资源映射策略。在 NFV 实际应用场景下，尤其是 IMS 服务中，服务链的服务形式对虚拟化的资源映射和 NFV 业务的管理编排需要对协同服务的一组虚拟机进行操作，在考虑到物理资源数量的同时也要考虑资源本身的亲和度关系，这样的资源映射方式对现有的虚拟化资源管理平台提出了全新的挑战。其次，当前市场上广泛使用的大通量服务器 (Commodity Server) 大多为多核的非一致性存储访问架构架构 (Non-Uniform Memory Access, NUMA)。在这种架构下，不同的物理资源之间的数据传输性能具有较大的差异。而现有的虚拟化平台缺乏有效的手段将与物理资源性能相关的信息传递给已有的资源映射算法，从而导致其失去优化效果。

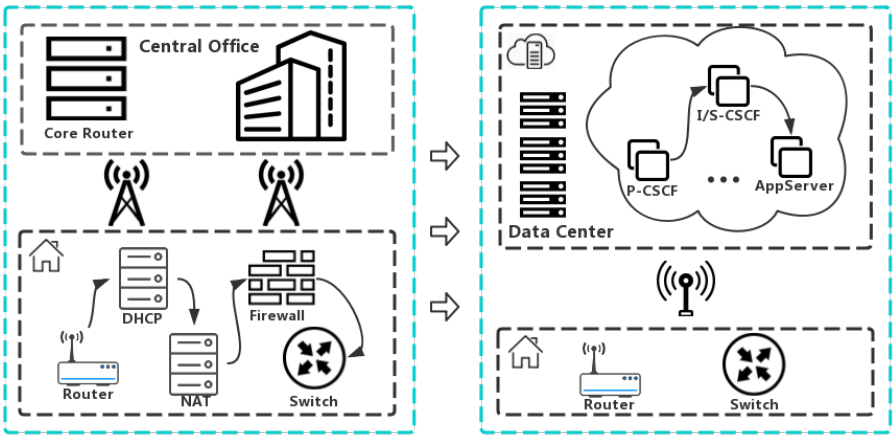


图 1–1: 云环境中的 IMS 服务

Fig 1–1: IMS in Cloud

学术界目前对于优化 NFV 性能的研究主要集中在两方面：(1) 优化通用操作系统和网络协议栈，自底向上地提升 NFV 应用的性能，打破网络服务中来自底层通用平台和

通用操作系统的性能瓶颈。(2) 提出针对 NFV 应用的资源映射算法，优化网络功能服务节点间网络流量的实际路径。

其中，在优化通用操作系统网络协议栈的方向，近些年学术界涌现了一系列的研究[7–12]，结合工业界提出的高速网络包转发框架 DPDK [13] ，从简化网络协议栈，重新分配虚拟化资源和减少无关开销等角度优化基于通用服务器的操作系统和虚拟化框架。这些研究都提高了标准通用服务器的在 NFV 场景下的网络性能，为 NFV 的大规模推广奠定了基础。然而，已有的成果中并未有对实际的 NFV 应用进行针对性的优化，忽略了 NFV 应用的特有的各网络功能节点间如服务链这样的组织形式。大部分的优化方法仍然从一般系统软件的角度去提升基础网络性能，缺乏对 NFV 应用组织结构的认识和建模。

文献 [14] 指出在 NFV 资源分配问题 (NFV resource allocation，NFV-RA) 中，如何解决基于 NFV 网络基础设施的资源分配是一项重要的挑战。进一步的，文献 [15] 中将该问题划分为更细的粒度进行深入研究，为研究此类的问题提供了具体的研究方向。NFV 网络基础设施的资源分配问题与文献 [16, 17] 所提出的传统网络中的网络功能映射问题 (Virtual Network Embedding，VNE) 相关，已有的研究参照 VNE 问题的研究成果和 NFV 背景下的具体问题，提出了一系列与实际应用相结合的算法来解决在 NFV 场景中资源映射问题。其中，有的从网络时延角度出发 [18]，对 NFV 的部署和映射问题进行建模，针对虚拟网络功能转发图映射和虚拟网络功能调度子问题提出了基于启发式算法的求解。文献 [19] 从网络开销角度出发，提出了基于混合整数线性规划的方法来解决最小化网络功能的部署和网络流量的路由问题。这些研究都从 NFV 物理资源角度结合 NFV 的应用特点给出了解决问题的思路，但是大部分研究使用仿真的平台进行算法有效性的验证，缺乏与实际 NFV 应用平台上的结合。

本文着眼于实际的 IMS 平台，利用验证性实验来说明在大容量通用服务器中 NFV 应用可能面对的性能波动，并针对网络功能服务链的资源映射问题进行了建模，提出了基于底层动态采样信息的优化映射算法。本文在 Clearwater 平台基础上实现了原型系统来验证本文所提出的资源映射模型和算法的有效性。实验证明，经过本系统的优化过后，Clearwater 服务的性能得到了有效的提升。

1.2 国内外研究现状

本节介绍并讨论与研究内容相关的国内外研究现状与发展趋势，重点包括国内外研究中对网络功能虚拟化中服务链问题的介绍，针对网络 I/O 优化，NFV 资源分配问题，以及国内外研究者在多核架构下针对该问题的一些研究成果。同时，本章中也将本文所设计的优化解决方案与已有的解决方案和处理思路进行对比，对相关研究的研究内容进

行比较和优缺点的分析。

1.2.1 网络功能服务链

根据文献 [20] 的定义，网络功能服务链 (SFC) 是抽象服务功能 (Service Function，SF) 的有序或部分有序集合，包括对数据包，数据帧和数据流的有序约束分组。在实际的 NFV 场景下，由于服务拓扑的复杂性，各个功能节点在服务链中的位置应具有相对的灵活性。当前，学术界中对于 NFV 中网络功能服务链的研究方兴未艾，文献 [21] 总结了现阶段 SFC 研究所面临的挑战，主要包括：(1) 复杂的网络拓扑依赖。原有的服务链是在硬件层面实现的，网络功能与底层硬件拓扑是绑定的，因此网络功能的增加与删减需要直接操作硬件，而 NFV 所带来的虚拟网络功能需要网络管理员最大可能的优化利用现有的网络资源，复杂的网络拓扑会给动态管理网络功能带来挑战。(2) 严格的配置管理。服务链的网络功能节点顺序有着严格的要求，错误的服务配置会导致服务失败，严重影响服务质量，因此如何保证服务链的正确配置也是业界重点关注的领域。(3) 受约束的服务弹性。服务链复杂的拓扑限制了服务链快速伸缩的弹性，服务链中任何一个网络节点的失效都可能导致服务链的伸缩失败，保证服务链所有功能节点的可用性对实现可扩展的服务链具有重要意义。文献 [22] 研究了在弹性网络功能下的服务链放置问题。该问题虽然与传统的网络放置问题相似，但为了满足弹性网络功能额外的灵活有序的要求，需要对该类问题进行重新建模，并基于新的模型进行求解和优化。本文所研究的网络功能服务链是基于传统的串行服务链，针对服务链所运行的通用服务器平台资源进行建模并优化求解。相比于已有的研究，本文使用更接近实际应用平台的具体 NFV 业务，并借助虚拟化平台和具体业务平台来具体实现 NFV 业务，可以使用真实的应用场景和平台来验证本文所提出系统的有效性。

1.2.2 网络 I/O 优化

为了减少在虚拟化环境中运行虚拟机所带来的额外开销，文献 [23] 对虚拟机操作系统进行了裁剪，使用微内核的方式构建了基于 MiniOS [24] 的最小化操作系统 ClickOS，并在 Xen [25] 的基础上将后端交换机 Open vSwith [26] 替换为了高速的 VALE 交换机 [7]，使用 Click 语言 [27] 进行 NFV 网络功能逻辑搭建，从而大大减少了虚拟化所带来的性能开销，提升了虚拟机的网络吞吐。但是在小包的测试环境中，该系统出现了剧烈的性能抖动，而且所能实现的虚拟网络功能仍处于单个功能节点阶段，尚未组成服务链，并不能满足 NFV 复杂的业务需求。相比文献 [23]，文献 [10] 基于 KVM 平台 [28] 推出了 NetVM，利用了高速的包转发框架 DPDK [13] 大大提升了网络协议栈的包转发速率，并通过共享大页内存、零拷贝数据传输以及优化 CPU 调度器的方式提升了数据在 VM

之间的传输效率，减少了由于虚拟化中虚拟 CPU 的调度所引起的上下文切换开销。本文中借鉴了文献 [10] 的平台，在 KVM 基础上使用 virtio [29] 方式来优化 VM 之间的网络传输性能，通过优化多核服务器中虚拟机间资源的亲和度关系，解决在 SFC 的背景下 NFV 应用的性能优化问题。

文献 [30] 针对基于虚拟化技术实现的 NFV 提出不同的看法，认为基于现有的虚拟化技术 (VM 或容器) 所提供的隔离性会带来很高的性能开销，并且针对虚拟化来开发相应的网络功能是个十分繁琐的过程，为了减少这类与服务无关的开销，他们提出了一个新的 NFV 框架 NetBricks 来解决这两个问题。对于构建 NF，文献 [30] 使用 Rust 语言构建了一组可定制的网络处理元素，利用该语言的类型检查和安全运行机制来提供基于软件的隔离，将网络功能虚拟化与虚拟化框架解耦，使得软件化的网络功能不再依赖虚拟化平台，极大地减少了构建网络功能的开销，从而实现整体 I/O 性能的提升。文献 [30] 所提出的优化思路十分值得借鉴，但是现阶段完全抛开虚拟化框架实现 NFV 的成本太高，而且现有的虚拟化管理框架所带来的优势并不能体现，这样的做法会使得 NFV 失去与当前云计算平台相结合的优势。本文依然采用的是基于 KVM 的虚拟机实现模式，尽管虚拟化会引起较高的开销，但是同时所带来的更加简单的实现方式，以及更好的隔离性保障可以使得本设计专注于解决服务链的资源映射问题。

1.2.3 NFV 资源分配问题

在 NFV 资源分配问题方面，文献 [15] 详细介绍了 NFV 资源分配问题的研究现状，总结了资源分配问题的具体子问题，即虚拟网络功能组链 (VNFs Chain Composition，VNFs-CC), 网络转发图映射 (VNF-Forwarding Graph Embedding，VNF-FGE) 和虚拟网络功能调度 (VNFs Scheduling，VNFs-SCH)。文献 [15] 将 NFV 下的资源分配问题归结为一类 NP-Hard 问题，并指出了解决此类问题的三种解法思路：确定解法、启发式解法和元启发式解法。具体来讲，针对 VNF-FGE 问题，确定解法中的整数线性规划方法可以在可接受的时间内给出最优解。而在时间敏感的 NFV 资源分配问题下，确定解法的时间开销超出了预期限制，这时就需要借助基于启发式和元启发式的高级算法来减少算法的运行时间和额外开销。本文所研究的网络功能服务链资源映射问题属于一类简单的网络转发图映射问题。为了解决这个问题，本文参考了文献 [15] 解决该类问题的思路，使用了基于贪心思想的确定解法来解决较小规模下的网络功能服务链资源映射问题。

文献 [31] 认为在数据中心中，网路功能服务链使得网络流以特定的顺序遍历不同的网络功能，为客户提供不同级别的服务。由于服务链中任何相邻网络功能之间的距离将决定该链路的总带宽消耗，那么虚拟化网络功能在数据中心中的放置便成为了影响整体带宽的重要问题。在文献 [31] 中，这种放置问题被归结为一个多重背包问题 (Multiple

Knapsack Problem)。文献 [31] 针对树状网络拓扑，文章提出了两种基于贪心的算法：多层最差拟合和多层最佳拟合。在仿真平台的实验结果表明，与传统的最优算法相比，优化算法可以将带宽消耗降低 15%，而使用服务器数量仅增加 1%。文献 [31] 使用组合优化的思想来归结此类问题可以使得问题变得更加清晰明了，但是算法仅在仿真平台上得到了验证，本文则利用真实的 NFV 应用平台对所提出的算法进行了验证。

1.2.4 其他相关研究

除了对于网络 I/O 的优化和 NFV 资源分配问题的研究，很多文献和论文也从其他角度对网络功能虚拟化进行了研究。

文献 [32] 总结了在网络功能虚拟化管理和编排时可能会遇到的挑战。文章结合虚拟化所带来的资源体量的增加，分析了如何在现有的庞大的基础设施中获取有效的产出所需要面对三个主要问题：(1) NFV 服务点 (Points of Presence, PoP) 的地理分布，为了更好地服务用户，如何确定虚拟网络功能集群的服务点地里位置对于电信运营商来说是重要挑战之一。严格的服务质量要求按照一定的最优化问题的方法来合理部署 NFV 服务的存在点，从而使得 NFV 服务可以保证覆盖所有服务范围。(2) 虚拟网络功能放置，网络功能虚拟化是以一定逻辑链接的虚拟网络功能所组成的，服务链就是一种典型的组成形式，在 NFV 服务点中如何去合理部署具体的虚拟网络功能对于 MANO 框架来说具有重要意义。(3) 动态资源管理，NFV 服务的弹性需求要求 MANO 系统同样能够支持动态地分配和调度物理资源来实现服务链的伸缩目标，能够满足不同需求规模下的服务实现。

文献 [33] 从 NFV 具体部署的大通量标准服务器出发，认为在实际运行时会面临许多挑战。因为 NFV 数据平面及其服务功能链的复杂性特征，现代 NFV 应用以异构软件流水线的方式部署在大容量通用服务器上。文献 [33] 从操作系统运行线程的角度出发，考虑到 NFV 应用流量必须由异构处理软件从网卡到终端接收器接收数据。由于流程的端到端性能是由每个处理阶段的性能共同决定的，基于底层物理架构的的大容量通用服务器中的资源分配映射方案必须考虑线程间的依赖调度来减少资源竞争所带来的影响。因此，在文献 [33] 中，他们提出了一个可以协同放置最小化端到端访问开销的线程调度机制来最小化 NFV 线程应用线程端到端的数据传递开销。为了更加方便地评估各线程的数据传递开销，文献 [33] 还提出了一个基于线程资源的 NFV 应用性能下降模型，通过这个模型来衡量数据异构的软件流水线中传递时所造成的开销。本文相比于文献 [33]，更加侧重于考虑虚拟化的资源分配所造成的影响，虽然从主机的角度来看，虚拟机也是以线程方式运行在主机上，但是虚拟机软件栈会屏蔽虚拟机内部应用的具体信息，而且现有虚拟机的 I/O 虚拟化技术也会对虚拟机线程间的数据传递有重要影响，本

文采用 virtio 的 I/O 虚拟化方式，从虚拟机间实际的数据传输带宽和延迟角度出发来解决服务链的资源映射问题，从而提高 NFV 应用的整体性能。

1.3 论文安排

本文对网络功能虚拟化中，虚拟机资源调度进行了研究，论文的具体结构和内容安排如下：

第一章作为绪论，主要介绍了本课题的主要研究内容和研究背景。进一步的对于本研究相关的工作进行了介绍和梳理，主要从网络 I/O 优化研究和 NFV 资源分配研究两个角度以及其他相关研究介绍了国内外已有的研究成果。最后本章对全文的结构和安排做了介绍。

第二章重点介绍了与本文的研究紧密相关的研究背景和相关技术，包括 NFV 的发展历程，IMS 服务以及本文所使用的应用平台 Clearwater，目前主流的网络 I/O 虚拟化技术分类，多核服务器非一致性存储访问架构，最后还介绍了在多核服务器下的虚拟机性能观测实验，为读者了解与研究相关的知识和下文的展开做了准备。

第三章着重介绍了在 IMS 服务的背景下服务链资源的分析和建模，以具体的 Clearwater 注册服务为例，分析了具体服务链的数据流向和建模的关键定义，在该模型的基础上提出了基于贪心的优化方案。

第四章将介绍本研究中的研究方案的设计与实现，以总体算法框架为切入点，分模块介绍了本文系统各子模块的实现方法。

第五章从实验的角度对本文的设计进行了性能上的测试。通过利用 Clearwater 自带的压力测试工具和基础性能测试工具 iPerf、Ping、Apache 等，在真实的测试环境下对本文所提出的系统进行了严格的测试。

第六章归纳全文，总结和回顾了全文的研究内容和主要研究结论，并提出了一些研究展望。

1.4 本章小结

本章主要整理和归纳与本研究密切相关的国内外研究现状和发展趋势。本章中，重点介绍了国内外的相关研究中，网络功能虚拟化带来的一系列挑战和存在的问题。同时，本章中也对现有解决方案进行了深入的探讨，分析这些方案的共同点和优缺点，与本研究中的设计相对比。本章也对网络功能虚拟化的研究热点进行了介绍，特别是对当前研究较为火热的资源调度问题及相关研究的研究成果进行了简单的归纳和总结。

第二章 网络功能虚拟化相关技术

2.1 网络功能虚拟化发展概况

欧洲电信标准化协会 (ETSI) 作为 NFV 的发起标准组织，自 2013 年开始陆续发布了 NFV 参考架构等系列白皮书。虽然 ETSI NFV 工作小组发布的各项白皮书不是强制执行的标准，但是得到了业界的普遍认可，已经成为了业界的事实标准。目前 NFV 的标准框架已得到各方的基本认可，各厂商也基本按照现有的标准架构开发和推进各自的服务框架。如图 2–1 所示，NFV 标准框架主要由 NFV 基础设施 (NFV Infrastructure，NFVI)、虚拟网络功能 (Virtual Network Functions, VNFs) 和 NFV 管理与编排系统 (NFV Management and Orchestration, NFV MANO) 三个主要部分组成。其中，NFV 基础设施指的是承载具体业务的通过虚拟化技术所管理的数据中心资源。与传统电信机房不同，数据中心具有统一的管理标准和更加普通的 x86 通用服务器，其维护管理成本和业务升级的成本相较与传统电信业务的中心机房 (Central Office) 要更加低廉。虚拟网络功能是具体业务的运行实体，结合传统电信的业务支撑系统和管理支撑系统 (OSS/BSS) 作为上层应用系统，以虚拟化软件的形式，运行在 NFV 基础设施中，并接受管理编排系统的功能管理。而 NFV 管理和编排子系统作为衔接 IT 虚拟化基础设施和传统电信业务的关键系统，受到了众多厂家的重点关注，也是目前研究的重点之一。MANO 系统负责解析上层业务、映射并管理具体的物理资源、编排具体业务系统等多项重要业务，其重要性不言而喻。需要特别指出的是，底层资源即通用的大容量服务器资源的管理对于传统电信服务来说是全新的问题和挑战。

自白皮书发表以来，ETSI 的 NFV 规范小组制定了每两年为一个发布周期的工作计划，持续推出更加具体的协议规范和标准术语定义以及 NFV 的使用案例。截止本文，NFV 小组已经进入了第三个发布周期。

在第一个周期 2013-2014 年内，小组的主要工作目标是：*a)* 推动网络运营商融合 NFV *b)* 将已有的适用标准纳入网络服务和产品中 *c)* 以促进创新和培育供应商开放的生态系统为目标，同步地开发新的技术规范

截止 2014 年，工作组除了前期发布的 6 篇定义性文档，又陆续发布了 11 篇详细的技术规范。第一套文档标志着 NFV 的标准化初期工作已经完成，需要对划分的各个工作领域进行进一步的详细定义和约束。第二个周期 2015-2016 年，小组的工作重心已经转移到了 VIM，VNFM，NFVO 以及各功能接口的具体定义中。进一步的，在该阶段的工作中，小组还重点推出了适用于当前 NFV 服务需求的信息模型参考，在模型中

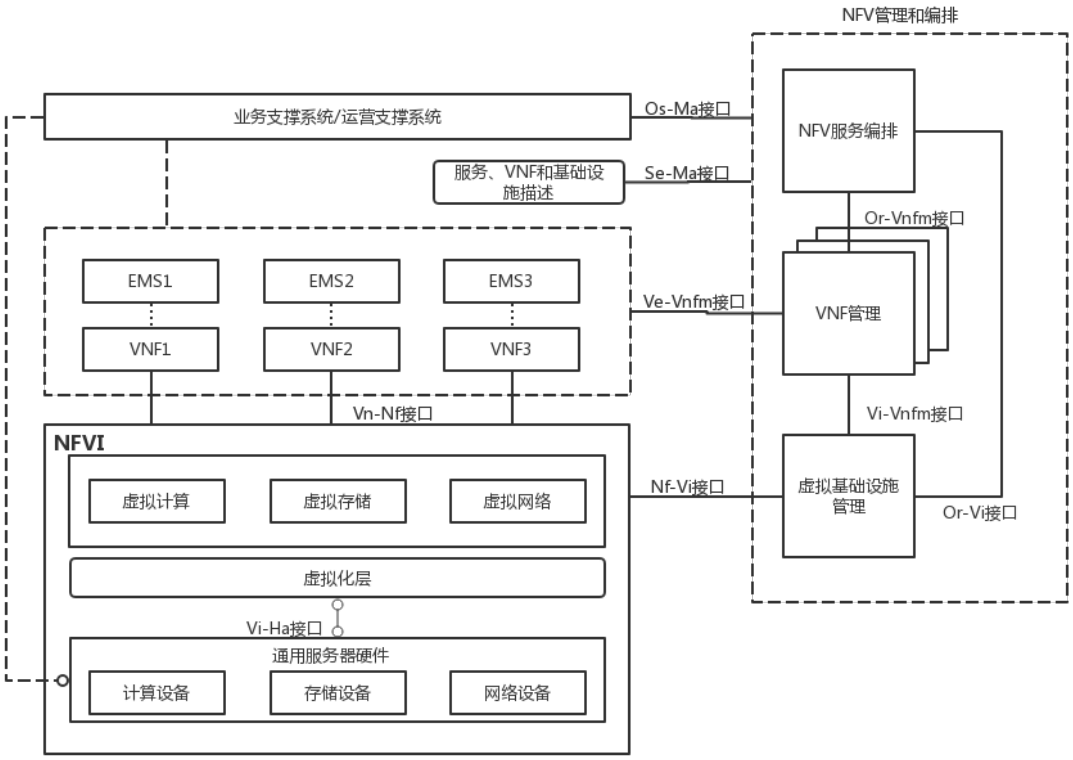


图 2–1: NFV 标准框架 [1]

Fig 2–1: NFV Standard Architecture[1]

预留了解决各项实际问题的参数接口，为各厂商开发自己的 NFV 服务框架提供指导。各项文档均已在 ETSI 官方网站发布。目前，NFV 规范小组正处在 2017-2018 发布周期内。根据已发布的计划，在本周期内他们有 23 个工作目标，包括 20 个新功能点和 3 个对上一周期的已发布文档的补充。在该周期内，小组的工作重点是深入定义核心的 NFV 信息模型，完善支持 NFV 非业务子系统接口如计费，账单，用户管理，监管，安全等，以及对 MANO 服务的多方面支持。同时在近些年来，NFV 工作组联合了如 Linux Foundation，OpenDayLight 等开源社区以及工业界各服务提供厂商，旨在携手推进 NFV 落地实现，与现有业务平台平滑过渡衔接。他们推出了各种开源的 NFV 平台如 OPNFV，OpenSource MANO，Open-O, Cloudify 等等。学术界近年来对 NFV 的研究也进行得如火如荼，相关介绍在第 一 章节中已有详细的介绍。

2.2 Clearwater 平台介绍

目前全球的运营开发商都在加速商用的 VoLTE (Voice over Long-Term Evolution) 开发，IP 多媒体子系统 (IP Multimedia Subsystem, IMS) 是 4G VoLTE 商用组件之一，因此也引起了工业界格外的重视。NFV 兴起以来，工业界纷纷在尝试将 IMS 系统部署在 NFV 平台环境中，探索 IMS 应用场景在 NFV 生产环境中的实现，希望能够利用虚拟化技术加速电信业务的研发和部署，降低业务升级的成本和周期。根据本文的调研，在开源的 IMS 系统领域，有两个十分知名的项目：Clearwater [5] 与 Kamalio [6]。而在这两者之中，Clearwater 系统更是因为其为云计算平台量身打造的多媒体子系统定位而广受众多厂商和研发人员的欢迎，开发社区十分火热，产品也在迅速的迭代中。它采用大型互联网软件架构的设计思路，以微服务的方式设计各个组件，使得系统本身具有很好的伸缩能力，成为了一个商用级别的开源 IMS 系统。

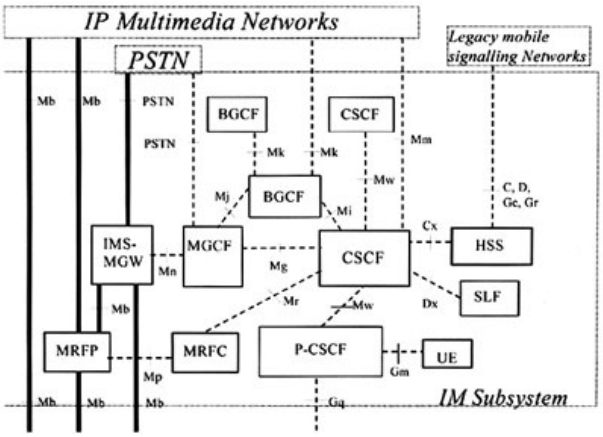


图 2–2: IMS 系统架构图

Fig 2–2: IMS Architecture

IMS 服务是由一系统功能组组成的，各个功能组之间由一组标准接口联链起来，组成一个 IMS 管理网络，其架构如图 2–2所示。一个功能组并非是一个节点，它的实现方式是开放的，允许将多个功能组布署在一个节点，同时也允许一个功能组由多个节点实现。考虑到容量、负载均衡和管理等方面，IMS 服务还允许在一个网络存在有多个相同的功能组。在 IMS 标准中，其核心网络功能组包括：归属用户服务器 (Home Subscriber Server, HSS)，呼叫会话控制功能 (Call Session Control Function, CSCF)，应用服务器 (Application Server, AS)，出口网关控制功能 (Breakout Gateway Control Function, BGCF)，媒体网关控制功能 (Media Gateway Control Function, MGCF) 等等 [1]。这些核心网络功能组的功能介

[1] http://www.3gpp.org/technologies/keywords-acronyms/109-ims

绍如下。

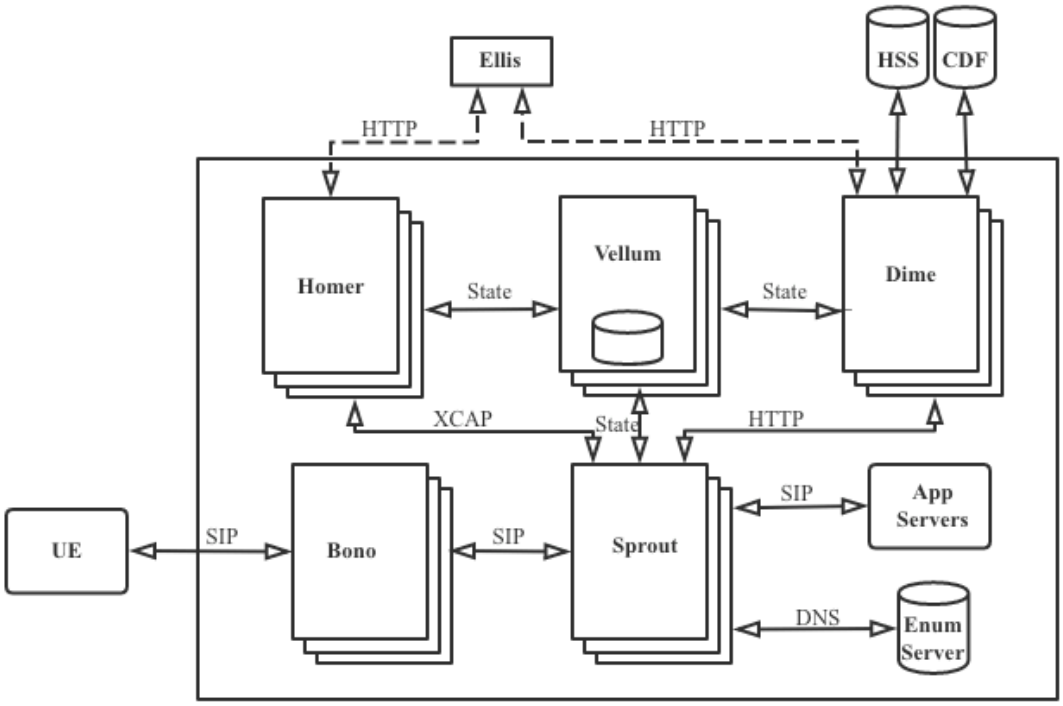


图 2–3: Clearwater 系统架构图 [5]

Fig 2–3: Clearwater Architecture[5]

本文所参考的 Clearwater 分布式版本的系统架构图如图 2–3 所示，可以看出该系统主要有五个核心的组件：Bono，Sprout，Dime，Vellum 和 Homer。分别对应实现了 IMS 标准服务中的各项功能。其中，Bono 节点作为 SIP 的边界代理，负责承担 IMS 服务的接入业务，通过 WebRTC 接口为客户端提供服务接入点。Sprout 节点作为注册和权限管理的路由代理，主要负责处理客户端的认证和应用服务器 ISC 接口的通信。Sprout 节点还包含内置的 MMTEL(multimedia telephony) 应用服务器，在运行过程中 Sprout 不负责存储任何持久化数据，而是通过 HTTP 协议从 Dime 和 Homer 服务中读取 HSS 的配置信息，如用户信息和 MMTEL 服务配置等。Sprout 还通过远程调用接口从 Vellum 中获取用户的注册信息和当前状态。Dime 节点主要负责运行 Homestead 和 Ralf 服务，包括存储 HSS 信息的缓存等具体功能。Homer 是一个标准的用户设置存储服务，通过标准的接口来增删改查用户持久化数据。而 Vellum 则负责存储各种持久化信息，包括用户的注册信息，当前服务的始终状态以及各种权限和密钥等。Ellis 作为一个前端管理工具，并不参与 IMS 实际业务，只是官方提供的一个用于管理整个平台的可视化工具原

表 2–1: IMS 服务核心网络功能组

Table 2–1: IMS Core components

名称	功能介绍
归属用户服务器 (HSS)	是一个核心的用户数据库，它为 IMS 网络中实际管理通话的实体提供支持。如存取用户相关的信息即用户配置，对用户认证和授权以及提供用户位置 IP 地址等相关信息。
呼叫会话控制功能 (CSCF)	由会话发起协议 (Session Initiation Protocol, SIP）服务器和代理共同实现通话控制功能，在 IMS 系统中负责处理 SIP 信号数据包。
SIP 应用服务器 (AS)	负责使用 SIP 协议与 CSCF 之间通信并提供 HSS 服务查询接口。
媒体网关控制功能 (MGCF)	负责完成 IMS 网络与 PSTN 网络之间的呼叫控制协议转换。其主要功能是将 SIP 消息进行转换，并管理 PSTN (public switched telephone network) 网络的负载以及与 IMS 网络 IP 流间的连接。
出口网关控制功能 (BGCF)	是一个 SIP 代理，它处理来自 CSCF 的路由请求。BGCF 有基于电话号码的路由功能，用来选择与 PSTN 网络的接口点。当 BGCF 发现被叫网络位于一个 PSTN 网络时，BGCF 就会选择一个媒体网关控制功能 (MGCF)，将会话路由到 MGCF 服务，由 MGCF 服务负责与 PSTN 网络交互。

型。

Clearwater 是针对云计算数据中心开发的 IMS 系统。众多大型电信运营商纷纷采用基于 IMS 的标准架构作为其 IP 语音通话、视频和消息服务的基础构建，用以取代基于传统电路交换系统的上一代 VoIP 业务系统。Clearwater 遵循 IMS 架构原则，实现了 IMS 核心网络所需的所有关键标准化接口。有别于别的 IMS 实现，Clearwater 是以虚拟化云环境作为基础设施的角度设计的。Clearwater 通过结合已在许多 Web 应用程序中验证过的设计模式和开源软件组件，实现了前所未有的大规模可扩展性和卓越的成本效益。正因为有这样的特点，本文选取了 Clearwater 作为 NFV 应用平台，在其基础上测试本文系统原型。

2.3 常见的网络 I/O 虚拟化技术

虚拟化技术作为 NFV 基础，是实现 NFV 的关键技术。I/O 虚拟化，特别是网络 I/O 虚拟化，更是直接影响 NFV 性能的关键因素。在本节中，将介绍和分析三种主流的网络虚拟化模型：设备仿真模型 (Device Emulation)，分离驱动程序模型 (Split-driver) 和硬件辅助模型 (Hardware-assisted)。本文实现的系统所采用的 I/O 虚拟化方式为 Virtio[28, 29]，接下来将介绍这三种 I/O 虚拟化方式的原理，并从每种模型的原理阐述选择 Virtio 作为 I/O 虚拟化方式的优势。

2.3.1 设备模拟模式

设备模拟模式是一种通过在虚拟机监视器中模拟敏感指令和特权指令的方式来模拟物理设备 I/O 操作的全虚拟化实现方式。如图 2–4 所示。这种模式充分利用二进制翻译技术和直接执行技术的组合来从客户操作系统中抽象和分离底层硬件，实现纯软件方式的硬件仿真。虚拟机操作系统实例和用户级别指令可以在没有感知底层虚拟化环境的情况下不加修改地运行，所以设备模拟模式为 I/O 操作提供了最佳性能隔离和安全性，简化了客户端虚拟机的迁移操作和提升了其可移植性。工业界有很多成熟的产品如 VMware Workstation [34] 和 VMware ESX Server [35] 都支持这种 I/O 虚拟化的方式。此外，XEN 和 KVM 的 QEMU 也可以配置支持设备模拟模式的 I/O 虚拟化。但是，这种方法也存在很大的缺陷，由于虚拟机监视器需要动态翻译所有操作系统指令并缓存所有结果，因此虚拟机监视器设备驱动程序一旦发生故障将导致所有客户虚拟机的 I/O 中断。并且，由于在运行时对这些敏感和特权的指令请求处理需要进行代价高昂的陷入和仿真操作，频繁的此类操作将引起巨大的性能开销，成为影响 I/O 性能的瓶颈。根据已有的研究，每个 I/O 操作的陷入和仿真都将导致客户操作系统和虚拟机监视器之间的上下文切换，其成本约为 3000 到 5000 个 CPU 周期 [36]。根据已有的研究实验显示，这样

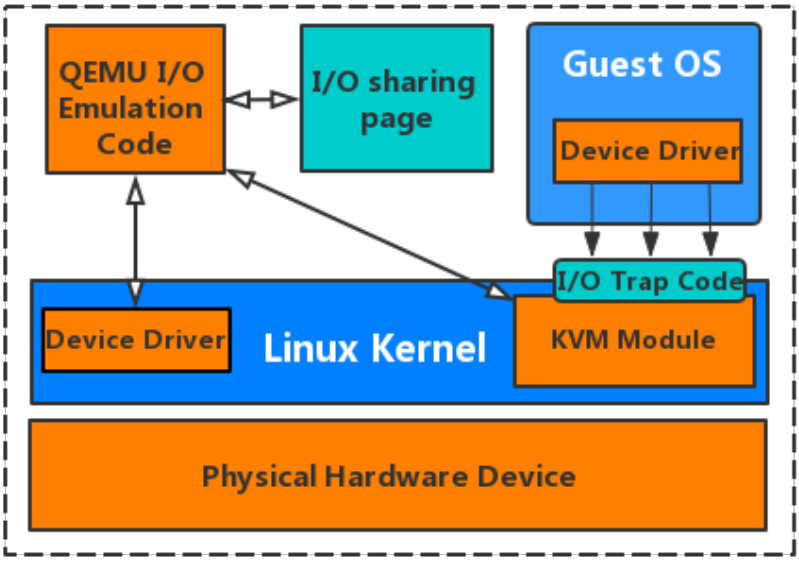


图 2–4: 设备模拟模式示意图

Fig 2–4: Device Emulation

的开销显着地降低了近 10 倍的 I/O 吞吐量。这一性能瓶颈挑战严重阻碍了将设备仿真方法部署到高性能网络和配备现代高性能网卡 (例如 40/100 Gbps 网卡) 的数据中心中。NFV 的应用场景下需要数量众多的虚拟机频繁的使用各自的网络 I/O 设备，设备模拟的方式并不能胜任该场景下对 I/O 设备高性能的需求。

2.3.2 分离驱动模式

为了缓解虚拟机监视器参与模拟的高昂开销，Xen 团队首先在半虚拟化场景下开发了分离驱动模式 (Split-Driver) [25] 。紧随其后，KVM 的 virtio [28, 29] ，VMware tool 和 VM interface [37] 也支持了在半虚拟化的分离驱动模式。本文以 virtio 为例，介绍了在 KVM 平台下奋力驱动模式的实现原理。KVM 的分离驱动模型如图 2–5 所示，它由主机中的前端 virtio 驱动和后端驱动组成。当虚拟机向服务器内的另一台虚拟机发送数据时，虚拟机首先将数据加载到其 vring 缓冲区中的队列，并修改其描述符表。随后通知 KVM 产生 vm-exit 信号，并使用 ioeventfd 方法允许 vHost 驱动程序接收来自客户端虚拟机的信号。当 vHost 驱动获得信号后，前端程序将获取有效队列的物理地址，并将需要传输的数据复制到绑定 tap 设备地址空间。数据通过网络协议栈被传送到目的地址上。在传

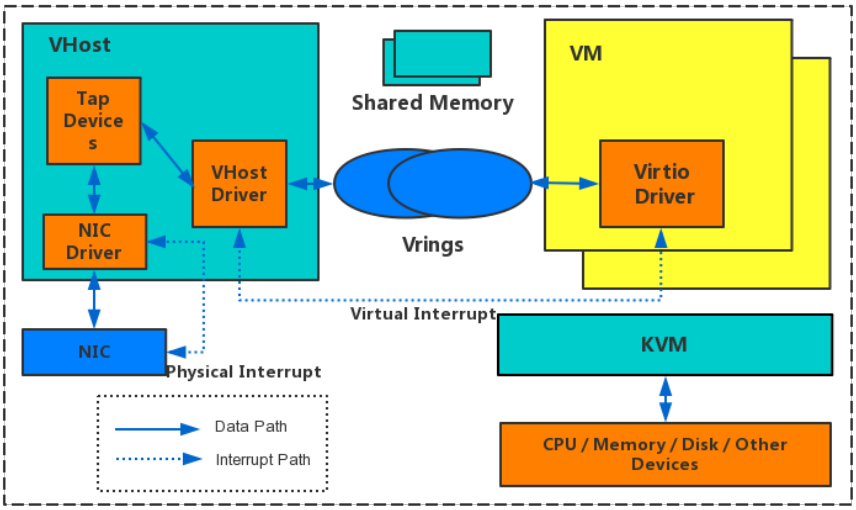


图 2–5: Split-driver 模式示意图

Fig 2–5: Split-driver Model

送过程中，前端驱动发现如果目标地址是同一台机器上的虚拟机，vHost 将利用零拷贝机制，通过共享内存直接将数据的地址指针发送到到目标虚拟机的对应驱动中。当目标虚拟机收到数据后会触发一个回调信号函数，并让虚拟机将传输的数据拷贝放入自己的 vring 读取队列。总而言之，在分离驱动模式下的虚拟机之间的数据传输可以被看作是将数据从一个内存区域复制到另一块存储区域。分离驱动模式采用高效的 I/O 通信通道 (I/O Channel) 来避免虚拟机监视器对模拟端口 I/O 和内存映射功能的干预。因此，分离驱动模式可以利用同样的物理资源达到比设备仿真模式下更高的吞吐量，同时降低 CPU 的利用率。即使半虚拟化方式需要对操作系统内核进行深度修改，无法做到像全虚拟化方式那样对不同客户机操作系统广泛的支持，分离驱动模式还是显示出了其在虚拟机管理和虚拟机迁移方面的巨大灵活性，在当下的虚拟机监视器和虚拟机操作系统中得到了广泛的支持。

2.3.3　硬件辅助模式

设备仿真和分离驱动模式都被分类为基于软件的 I/O 虚拟化，这种基于软件的模式实现了丰富的 I/O 设备功能同时简化了对虚拟机 I/O 设备的管理。然而，这种方式也并不是完美无缺的，纯软件实现方式在面对大量的陷入-仿真操作或者大块数据拷贝的时候会面临高昂的系统开销，这个时候可以借助硬件辅助的模式来解决这种情况下

的瓶颈。硬件辅助 I/O 虚拟化模式是为共享本地设备而开发的，它通过继承 I/O 内存管理单元 (IOMMU) 的用户权限来使用直接 I/O 技术从而实现绕开虚拟化的内存保护和地址转换，为虚拟机直接分配真实的 I/O 设备。PCI-SIG(Peripheral Component Interconnect Special Interest Group, 周边元件互连特别兴趣小组) 工作组针对这种 I/O 虚拟化的方式制定了规范，通过为每个虚拟机提供独立的内存空间，中断和 DMA 流来绕过虚拟机监视器直接参与数据处理。特别的，PCI-SIG 推出了单根 I/O 虚拟化 (SR-IOV) 技术，这项技术为针对 PCIe 设备设计了一组硬件增强技术，通过去除虚拟机监视器在数据包分类和内存地址翻译的干预来实现基于专门硬件的性能提高。SR-IOV 虚拟化模式结构如

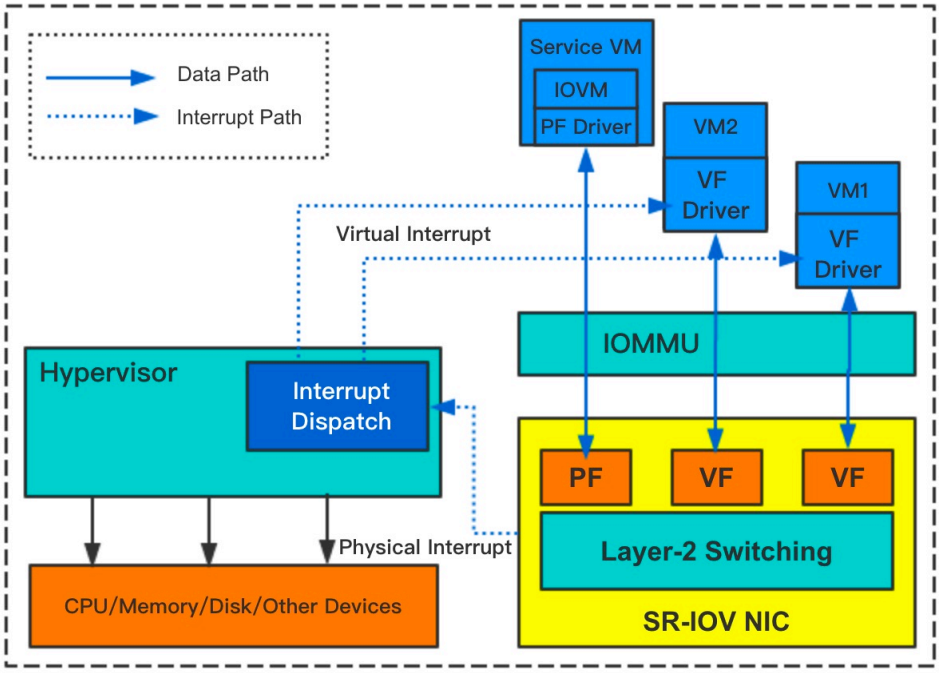


图 2–6: 硬件辅助模式示意图

Fig 2–6: Hardware-assisted Model

图 2–6 所示，一个支持 SR-IOV 功能的设备可以由虚拟机监视器分配多个虚拟设备 (VF, Virtual Function), 以普通 PCI 设备的身份被分配 PCI 地址空间。宿主机中的物理设备 (PF, Physical Function) 的驱动程序负责管理和配置 VF，而在被分配 VF 的虚拟机操作系统中，驱动程序将其当做普通的直接分配的物理设备来使用，通过 IOMMU 直接访问自己专有的 VF。通过这样的方式，SR-IOV 可以绕过虚拟机监视器实现数据的传递，从而减少移动数据的开销，在降低 CPU 利用率的同时，减少系统的延迟并提高网络吞吐量。但是需要指出的是，这种硬件辅助的方式需要手动进行虚拟网络功能的配置，无法实现像软件实现方式那样灵活的迁移和扩展。并且一块 PF 所能分配的 VF 是有上限的，这

也从硬件角度限制了硬件辅助方式的在多虚拟机应用场景下的应用。

2.4　非一致性存储访问架构

众所周知，NFV 的目标是在大容量的标准通用服务器上运行传统通信业务，而现在通用的标准服务器基本都是遵循非一致性存储访问架构 (NUMA, Non-Uniform Memory Access) 的服务器阵列，为了更好地完成 NFV 到通用服务器上的迁移，了解通用服务器的基础架构十分必要。图 2–7 所示的是 Intel Haswell 架构下多核处理器示意图，在 NUMA 服务器中每个处理节点 (Socket/Die) 通过内存访问控制器 (MC，Memory Controller) 与相邻的内存直接相连。同一个处理节点中多个物理核拥有自己独占的 L1/L2 级缓存并且通过节点内部链路通道共享 L3 缓存，内存访问控制器以及 PCI-e 接口。在一个处理节点内部数据传输是通过点对点的高速数据通路 (例如 Intel 的 QPI) 实现的。除了同一个节点内通信，不同的节点间也由相同的高速数据通路相连。不同的处理器节点要访问别的节点内存时，需要通过内存控制器进行远程访问。在 NUMA 架构下，本地和远程内存

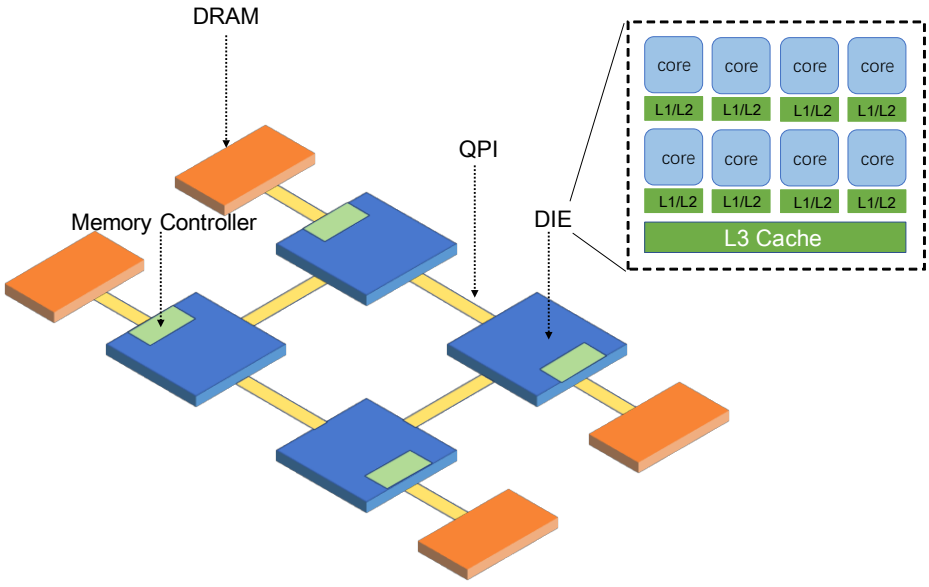


图 2–7: Intel Haswell 架构非一致性存储访问示意图

Fig 2–7: NUMA Architecture

访问由于机制的差异存在性能上的差异，而节点内共享的内存控制器在竞争过高的情况下也容易成为性能瓶颈导致内存访问性能的剧烈下降，所以如何针对多核通用服务器下非一致性访问的特性来优化应用一直是学术界所面对的经典挑战之一。NFV 应用迁

移到多核服务器中，不可避免地会遇到 NUMA 架构所带来的问题，而 SFC 的服务组织形式下同一条服务链上的虚拟机彼此之间存在大量的数据拷贝传输，如何处理好基于 NUMA 物理架构的物理资源对实现高性能的 NFV 具有重要意义。

2.5 多核物理机虚拟化应用性能观测与分析

在多核物理机中，由于存在非一致性存储访问架构的影响，分配了不同物理资源的两台虚拟机之间的数据传输性能会有较大的差异。在 NFV 应用的场景下，网络功能单元是以虚拟机的形式存在，节点间的数据传输实质上是虚拟机利用网络虚拟化的技术进行通信。从服务器硬件的角度看，可以当做一个虚拟机线程向另一个虚拟机线程拷贝或者接收数据。那么在这样的前提下，虚拟机线程所分配的物理资源之间的数据传输性能对虚拟机线程间通信的性能会造成影响。为了验证这一想法，本文在一台通用服务器上进行了模拟 NFV 应用的的测试。测试平台具体参数见实验部分的表 5–1，使用的虚拟化平台为 KVM，网络虚拟化方式为 Virtio。实验的设置如图 2–8 所示。实验使用两台虚拟机，其中一台固定在一个处理器节点上并绑定本地内存作为虚拟机的分配内存，而另一台虚拟机则不断更改所绑定的物理核及内存，以遍历所有绑定情况作为测试样本，测试的结果如图 2–9 所示。这里我们选用的基准测试工具 Netperf 和 Httperf 如表 2–2 所示，分别模拟微观的数据包负载和宏观的应用负载来测试不同资源组合间虚拟机通信的性能表现。

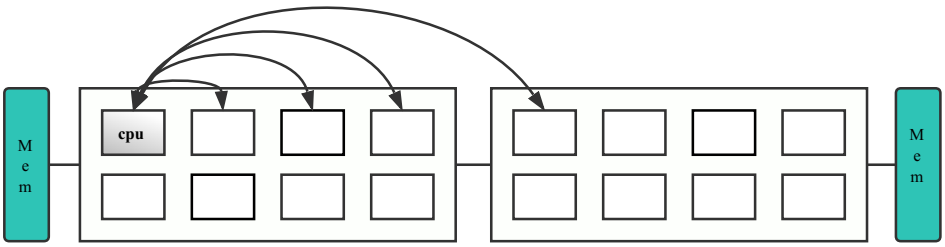
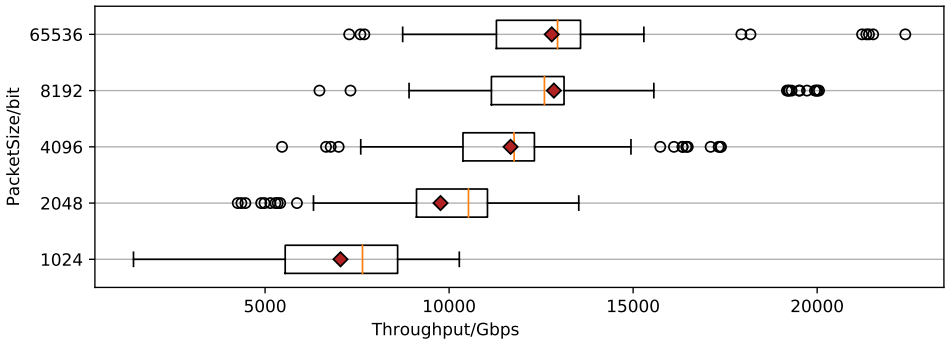


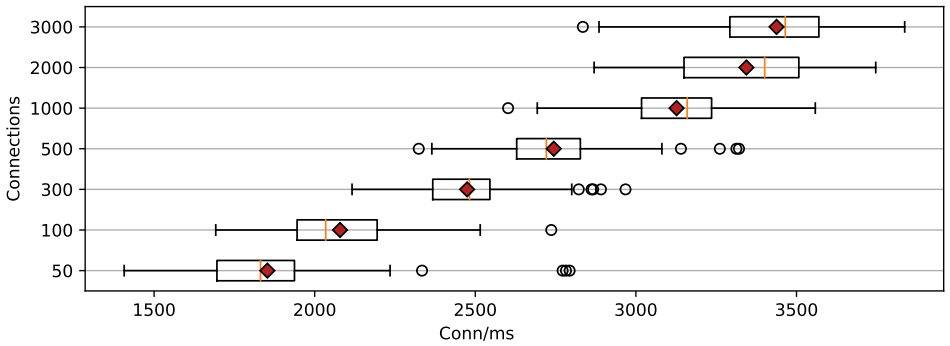
图 2–8: 模拟应用性能测试示意图

Fig 2–8: Illustrative Experiment Setup

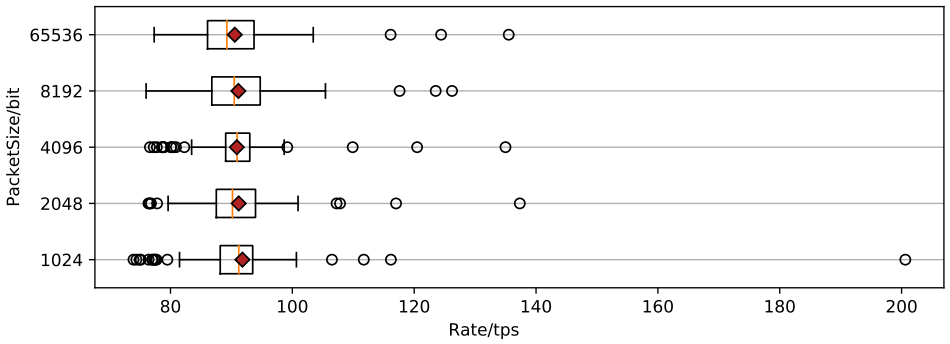
本文使用盒图的方式来展现模拟应用性能测试观测结果，将所获得的样本数据直观地体现在盒图中。实验分别观测了微观下数据包带宽和延迟以及宏观下网络应用访问连接数。据根据测试结果不难发现，在分配了相同数量的物理资源前提下，盒图中数据盒的宽度较大并且有大量的离群点出现，离群点之间的观测值差异甚至达到了接近三倍的差距如图 2–9(a) 所示，这表明在不同的资源组合模式下测试性能结果分布的离散程度



(a) Netperf 带宽测试



(b) Httperf 带宽测试



(c) Netperf 延迟测试

图 2–9: 模拟应用性能观测结果

Fig 2–9: Application Performance Observation

表 2–2: 验证性实验基准测试工具

Table 2–2: Benchmarks

基准测试工具集	**选用的实际负载**
Netperf 2.6.0	TCP_STREAM, TCP_RR
Httperf 0.9.0	Apache 2.4.7

表 2–3: 模拟应用性能测试结果数据分析一

Table 2–3: Data Analysis I

tcp_stream_1024	分组/Gbps	频率	累计百分比
	[1000-3000)	2	2.17%
	[3000-5000)	15	18.47%
	[5000-7000)	20	40.21%
	[7000-9000)	42	85.86%
	[9000-12000)	13	100%

较大，即虚拟机之间的数据传输性能在不同的虚拟机资源亲和度下存在较大的波动。具体来看，选取模拟应用测试中 Netperf 工具中 TCP_Stream 负载在数据包大小为 1024 bit 和 Httperf 工具在并发数为 2000 时的测试结果进行数据频率分布统计，统计结果如表 2–3 和表 2–4 所示。根据统计结果可以看出，在相同的测试负载下，测试结果在各个分组区间中所占百分比较为接近，数据较为均匀的分布在各个区间中，对于实际应用来说存在提升性能的空间。根据这样的实验观测和分析结果，可以认为针对虚拟机物理资源间的数据传输性能来优化服务链中的资源映射从而提升服务链数据传输性能是可行的。

表 2–4: 模拟应用性能测试结果数据分析二

Table 2–4: Data Analysis II

http_2000	分组/Connections	频率	累计百分比
	[2800-3000)	7	7.29%
	[3000-3200)	21	29.17%
	[3200-3400)	19	48.96%
	[3400-3600)	36	86.46%
	[3600-3800)	13	100%

2.6 本章小结

本章主要介绍了与网络功能虚拟化技术已经本课题相关的一些技术背景。首先，本章简单总结了下网络功能虚拟化的发展历史和当前的发展现状，介绍了网络功能虚拟化出现的背景，包括与网络功能虚拟户实现相关的虚拟化技术以及多核服务器的架构背景。本章着重介绍了本研究所基于 NFV 实际应用平台 Clearwater 和 IMS 服务，包括其业务功能简介和基本架构介绍。最后本章通过多核服务器的性能观测实验说明了在多核服务器中存在的实际性能问题，为接下来的模型和设计提供必要支撑。

第三章 IMS 服务链资源映射分析与建模

为了更加直观地处理 IMS 系统中服务链的资源映射问题，我们需要对服务链和底层所运行的平台资源进行抽象。本章主要针对 IMS 服务链背景下的网络功能映射问题，从所使用的虚拟资源角度出发，对其进行分析和建模。该模型重点关注了服务链实例之间的通信带宽和延迟，从这两个角度来量化不同的虚拟资源组合所带来的服务收益。

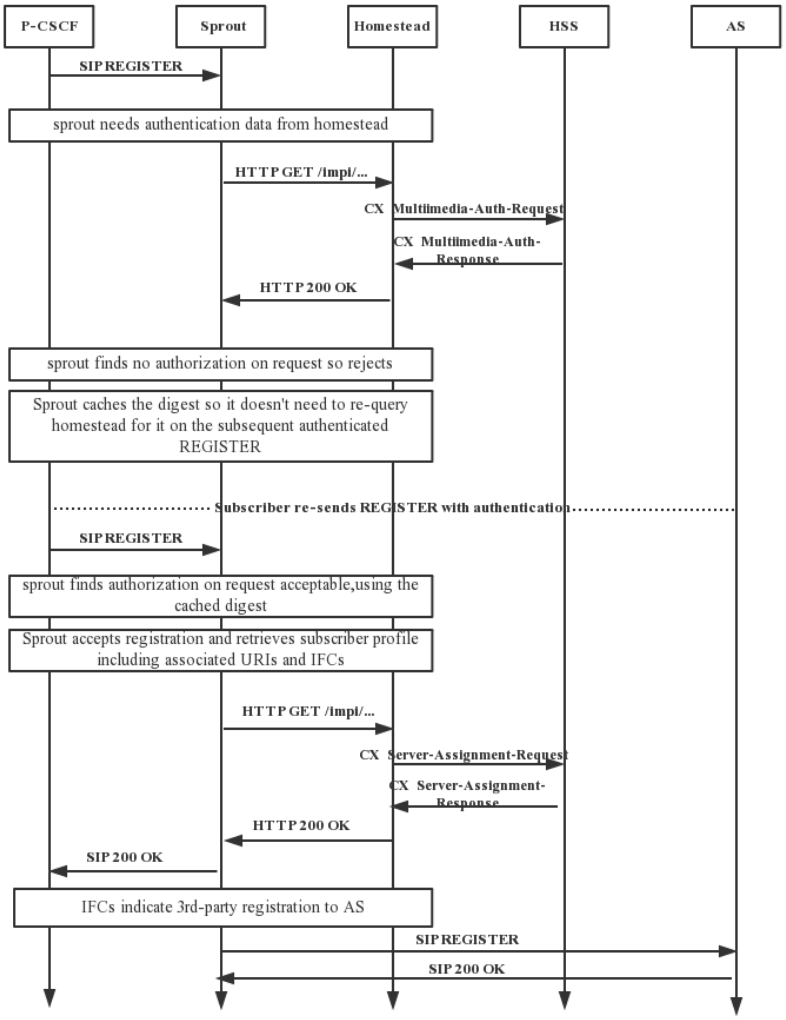


图 3–1: Clearwater 用户注册服务流程图

Fig 3–1: Flow of User Registration in Clearwater

3.1 问题概述

网络功能服务链的**资源映射问题**实质上可以理解为将抽象的逻辑服务链转化为实际提供服务的虚拟机群组。其中，服务链中的每个功能节点被映射为一个个实际运行的虚拟机实例。在由虚拟机实例所组成的服务链中，网络流在端到端的路径上，按照一定顺序经过一组特定的网络功能，实现客户所需的网络服务。在 Clearwater 系统中，每种服务功能会预先初始化各自的虚拟机实例，服务链的资源映射即选择服务功能组中的实例进行组链来实现特定的网络服务。为了详细说明 Clearwater 中的服务链资源映射，这里以 Clearwater 系统中的用户注册服务为例，其具体流程如图 3–1 所示。该过程中所涉及到的功能节点的信息见第 2.2 章，不再详细介绍。

当 CSCF 功能代理 (位于 Bono 节点中) 向 Sprout 节点发起注册请求，Sprout 节点需要通过 HTTP 请求向 Homestead 服务 (位于 Dime 节点中) 获取用户的认证信息，并根据查询结果来反馈注册结果。此时的服务链的数据流向为 $Bono \longleftrightarrow Sprout \longleftrightarrow Dime$。整个用户注册请求过程中涉及到 Bono，Sprout 和 Dime 这三个功能节点，整个过程可以抽象地看做数据在由这三个节点所构成的服务链中传递。在默认情况下，当 Clearwater 系统接到注册请求时，系统会从当前运行的这三个功能组的实例中随机地选取来组成服务链来满足服务需求。此时，根据所选取的实例的资源映射情况，我们可以得到不同的实际数据传输路径，整个映射的过程如图3–2 所示。图中展示了对于同一条服务链选取了两种不同资源映射组合的情况。这两种映射情况都可以满足注册服务的功能需求，但是由于所选取的虚拟资源不同所以性能上存在差别。由第2.5章中的验证性实验可知，由于所选取的实例间彼此资源的数据传输路径不同，这两种资源映射组合下的服务链的实际性能是不同的。在默认的随机选取的情况下，服务链资源映射的方式是未知的，这给实际服务带来了潜在的性能波动。

从物理机器中实际的数据流角度来看，由于不同的虚拟机实例分配了不同的物理资源，所以数据流的路径会由于所选取实例的不同有所差异。为了更加准确地刻画不同资源组合，我们将基于虚拟化的物理资源对 IMS 服务链进行建模。

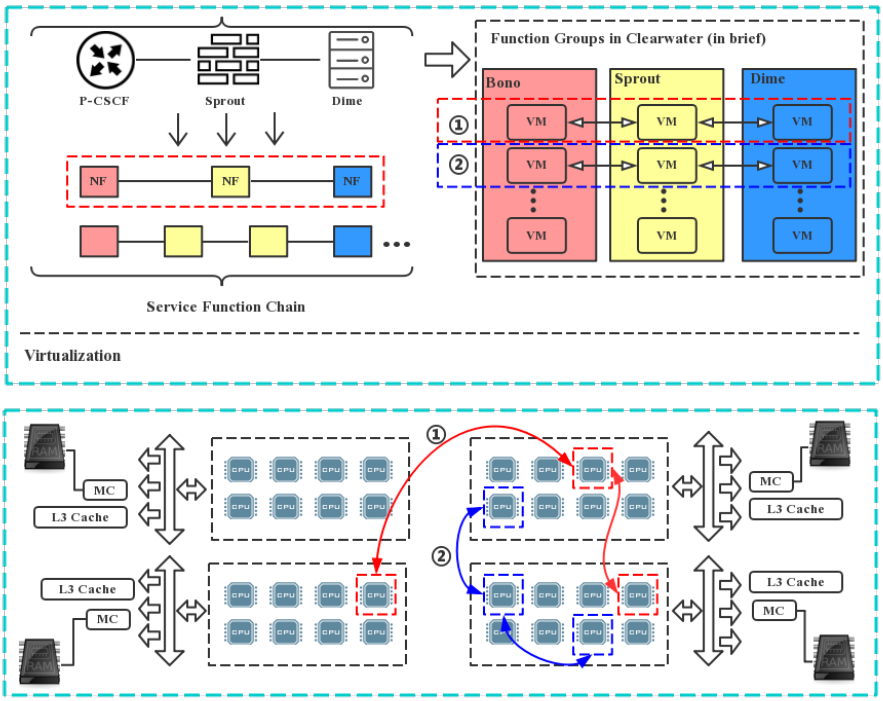


图 3–2: Clearwater 用户注册服务链资源映射示意图

Fig 3–2: Resource Mapping Diversity of Clearwater register service

3.2 关键概念及模型定义

为了对虚拟化环境下 IMS 服务链资源映射问题进行建模，我们需要对一些关键概念进行定义。需要指出的是，在进行服务链资源映射之前，还存在着虚拟机到物理机的映射问题，这一问题已经有大量的研究来解决 [17, 38, 39]，本文主要针对在运行实例已被初始化完毕前提下的网络功能选取和组链的问题，在后续的实验中本文也在不同的虚拟机到物理机的映射组合下进行了实验，验证本文所提出的优化方法的有效性。为了在实际场景中解决此问题，我们需要对该问题的前提做一些约束。本文所解决的问题基于以下的假设：

1. 每个虚拟机实例仅运行一种特定的网络功能应用。网络功能与虚拟机存在多种多样的映射关系，其中，一对一的这种映射关系目前成为了主流。随着轻量级虚拟机的出现 [23, 40], 一台物理机上所能承载的虚拟机数量得到了极大地提升，因此这种单一虚拟机单一网络功能的模式可以实现复杂的 NFV 应用。
2. 每个虚拟机应当专注于处理本身运行的 NFV 负载，并且尽量避免由于其他无关业务所带来的性能开销。因为我们选择了单核虚拟机并绑定了虚拟 CPU 到特定

的物理 CPU 从而减少多核调度所带来的额外开销以及上下文切换。

3. 针对每种特定的网络功能，都有多个运行实例处于待机状态，随时可以根据需求被映射提供服务。

3.2.1 相关建模定义

网络功能组 考虑到云计算场景下，各种资源都聚合以资源池的方式对外提供，在 IMS 场景下尤其是 Clearwater 平台下也不例外。在这里，我们以集合 D 来表示某种特定网络功能的资源池，也称为网络功能组。

服务链 我们用集合 S 代表一条被映射到具体网络上的网络功能服务链。S 由一群有序的虚拟网络功能实例组成 $S = \{f_a \rightarrow f_b \rightarrow f_c\}$，其中，$f$ 是从特定的网络功能组 D 中 (例如 Sprout 功能组) 选取的某一个运行实例。网络流量根据服务链的连接顺序，依次通过每一个功能节点。

多层级映射 如图 3–2 所示，除了从服务链描述的抽象逻辑服务链到实际运行的网络功能域中运行实例的映射关系之外，还存在着潜在的从服务链到物理资源的映射。本文用集合 R^{cpu} 表示一台物理机上所有的物理 CPU，为了简化问题在上文的叙述中已经假设所有虚拟机仅配置单个虚拟 CPU 并绑定物理 CPU，故在此假设下 R^{cpu} 也可表示所有虚拟 CPU 的集合。当服务链与运行实例绑定后，服务链与底层物力资源的映射也随之建立。另外，本文使用 R^{mem} 表示一台物理机的所有物理内存。由于虚拟化软件栈的存在，上层的服务链对于所绑定的底层硬件信息并不知晓。但是实际情况是，如同第 2.5 章中验证性实验所示，底层资源之间的选择组合对于整条服务链的性能有着重要影响，所以优化服务链的底层资源组合对提升服务链的整体性能具有重要意义。

因为 IMS 中主要的负载是数据流量，所以本文使用衡量网络流量的延迟和带宽参数来衡量 NFV 的网络性能。我们令 $B_{i,j}(i,j \in D, i \neq j)$ 来表示任意两个运行实例 (虚拟机) 的的网络通信带宽，令 $L_{i,j}(i,j \in D, i \neq j)$ 来表示两者之间通过真实或者虚拟网络通信的网络通信延迟。按照本文的假设，所有的虚拟机仅配置单个物理核，那么显然也存在一个从 $f \longleftrightarrow m(f \in D, m \in R^{cpu})$ 的映射。对于服务链上每个虚拟机来说，数据流量将按照串行的顺序沿着数据路径依次通过每个虚拟网络功能节点。

为了衡量整个服务链的带宽和延迟，需要分析任意两个相连的两个实例之间的带宽和延迟。对于带宽而言，根据串行系统的特性，我们选取整个数据链路上最小带宽作为整条链的带宽值如公式 3–1 所示。

$$Bandwidth(S) = \min\left(B_{i,i+1}\right), i \in S \tag{3–1}$$

对于任意两个网络功能 (虚拟机) 之间的通信延迟来说，本文进一步定义了其主要

的组成如下:

$$L(i,j) = L_{i,j} + L_i + L_j, i,j \in D \tag{3–2}$$

其中，$L_{i,j}$ 定义如上文所示表示两个实例的网络传输延迟，L_i,L_j 则分别表示虚拟网络功能处理相关流量而产生的延迟。对于任意一个特定的虚拟网络功能，如果配置了相同的物理资源，则其处理相同网络流量所花费的时间是相同的，也就是说所产生的这部分的延迟是一个与具体服务相关的固定值。

在此基础上我们进一步定义数据流量通过某条服务链 S 在数据路径上的传输延迟为 $\Delta Latency(S)$，其定义如下:

$$\Delta Latency(S) = \sum_{i=0,j=i+1}^{|S|-1} L_{i,j}, i \in S \tag{3–3}$$

同理，服务链上累计的数据流量处理延迟为与服务链业务相关的常数。但是 $L_{i,j}$ 这部分的延迟则由于不同物理资源的组合会产生不同的数据传输路径，存在着很大的变化空间。总的来说可以认为，服务链 S 上的传输延迟 $Latency(S)$ 最终定义如公式 3–4 所示。由公式不难看出，服务链上 S 的传输延迟主要取决于 $L_{i,j}$。

$$\begin{aligned} Latency(S) &= \sum_{i=0,j=i+1}^{|S|-1} L(i,j) \\ &= \sum_{i=0,j=i+1}^{|S|-1} L_{i,j} + C \\ &= \Delta Latency(S) + C, i \in S \end{aligned} \tag{3–4}$$

3.2.2 网络服务约束定义

基于上文中已有的模型，这里我们进一步定义模型中的约束，包括服务收益和服务开销。

服务收益 一个服务链的服务收益 P 可以被定义为通过一定的映射和调度策略下从所使用的物理资源中收获的定量的服务。参照已经由公式 3–4 和 3–1 定义的带宽和延迟，总体的服务收益可以被视为这两个指标的线性和，其中 α 和 β 为线性表达式的系数，由服务的具体需求来调整。例如，对带宽要求要求的服务则 α 的值较大，而对延迟敏感的应用则有较大的 β 值。

$$P = \alpha * Bandwidth(S) + \beta * (Latency(S) - \Delta Latency(S)) \tag{3–5}$$

服务开销 我们从所使用的物理资源的角度来定义一条服务链的服务开销。服务链中的运行实体主要是虚拟机，而对于虚拟机而言主要的资源开销为物理 CPU，内存，虚拟硬盘和所分配的其他虚拟 I/O 设备。在本文中，虚拟硬盘和其他虚拟 I/O 设备与所研究的内容无关故不做考虑，同时服务链所分配的物理资源在理论上不应超过物理机所能提供的物理资源上限。所以，服务开销 E 定义如公式 3–6 所示。

$$\begin{aligned} E =& \delta * \sum_{i \in S} r_i^{cpu} + \theta * \sum_{i \in S} r_i^{mem}, \\ & \sum_{i \in S} r_i^{cpu} < R^{cpu}, \\ & \sum_{i \in S} r_i^{mem} < R^{mem} \end{aligned} \tag{3–6}$$

公式中的 δ 和 θ 分别是根据不同应用需求来调节 CPU 和内存权重的系数。在本文中，结合所使用的实际应用 Clearwater 服务，我们令 $\delta = 10^6$，$\theta = 0.001$ 。同时为了保证服务链物理资源的可用性，本文约束所使用的 cpu 和内存资源均不超过物理机的资源上限。

从以上定义不难看出，实际的服务开销和服务收益的主要区别在于实际收益中只计算了数据流量在网络功能节点中的处理时间，而实际服务开销则包涵了服务链在传输线路上的传输延迟。

表 3–1: 符号表

Table 3–1: Symbols

符号	意义
D	网络功能组
S	某一网络功能服务链
R^{cpu}	一台物理机上所有的物理 CPU 资源
R^{mem}	一台物理机上所有的内存资源
$B_{i,j}$	任意两个运行实例 (虚拟机) 之间的通信带宽
$L(i,j)$	任意两个运行实例的通信延迟
L_i	实例处理数据所产生的延迟
$L_{i,j}$	任意两个实例在数据传输路径上所产生的延迟
P	一条服务链的服务收益
E	一条服务链的服务开销
α	带宽调节参数
β	延迟调节参数
δ	CPU 资源调节参数
θ	内存资源调节参数

第四章 方案设计与实现

本章重点介绍基于 Clearwater 系统的网络功能服务链资源映射优化的设计和实现。在本设计中，需要结合底层运行平台运行时信息和具体服务请求来作为系统输入，生成之后的优化决策并映射到实际的物理系统中。当系统初始化时，本设计将物理平台的动态性能测试的结果作为运行时参数输入系统，后续算法将以该参数作为算法执行时的参照标准。当收到一个服务链的请求后，将上层抽象的业务需求转化为具体的服务链业务链表。系统的动态映射模块根据服务链的实际组成来映射相应的具体实例。在映射实例的过程中，本设计的原型平台 Clearwater 默认使用的是随机映射的策略，而这种策略在实际的运行环境中缺乏对服务链中所涉及的虚拟机和所分配的底层资源的感知。本设计在第三章所提出的模型基础上，使用基于底层性能信息的优化算法生成新的映射策略替代默认的随机映射策略，从而提升资源的利用效率和网络服务的性能。

4.1 总体设计概述

4.1.1 总体设计目标

一个有效的资源映射策略应当在服务功能完整的前提下进行资源的有效配置。在服务器中，数据从一个网络功能节点传输到另一个节点的过程中不可避免的会产生传输的延迟，数据传输的带宽也会因为底层不同数据路径的原因有所差别。服务器平台的处理器数量和内存空间也是有限的，不平衡的负载分布会导致剧烈的资源竞争甚至导致意想不到的性能下降。因此，本文所设计的系统需要考虑到以上各个因素，生成相对优化的资源映射策略。在本文中，设计主要关注以下的三个目标：

1. 有效的资源映射策略应当首先保证网络功能服务链的完整性。在对物理资源进行资源分配时应保证服务链中功能节点的完整性，能够提供有效的网络服务。
2. 有效的资源映射策略应该能够提升在使用相同数量物理资源下的服务性能。
3. 有效的资源映射策略应当考虑实际的负载分布，避免因为局部负载过高而产生的性能下降。

4.1.2 总体设计思路

鉴于已有验证性实验的观测结论和以上对此类问题的目标分析，本文所设计的系统需要获取实际服务链的具体组成结构，并考虑具体实例之间物理资源的数据传输性能来

满足实际应用的最小带宽和最大延迟要求，同时在保证服务完整性的前提下提升物理资源的利用效率。

为了实现以上的设计目的，本设计提供了一种基于底层多核运行平台的虚拟化网络功能的映射方法。首先，当平台接受到一个服务请求时，应当根据服务请求的解析来确认所需要的网络功能组和具体提供服务的网络功能服务链。本设计会评估当前服务资源是否满足提供服务的最小需求，即组成服务链的不同功能组中是否有足够的运行实例。当资源充足时，系统开始根据参与服务链具体功能实例来组织服务。在挑选具体服务实例时，系统根据预先处理生成的物理资源数据传输性能矩阵作为调度算法的输入，根据矩阵中资源间互相访问的性能大小关系和当前机器的负载信息，依照网络功能服务链的连接顺序，使用局部最优的方法搜索构造出相对较优的服务链与实例的映射关系，根据所挑选出的实例来组织网络功能服务链。

4.1.3 总体设计架构

基于上文所提的设计思路，本文提出了一套底层运行平台感知的优化系统。该系统架构如图4–1所示，本系统设计了两个模块来实现前文中的设计目标：信息采样模块 (Hardware Prober) 和动态映射模块 (Dynamic Coordinator)。

信息采样模块使用标准的采样工具动态获取服务器物理资源间的数据传输性能信息。这些不同物理核之间的数据传输带宽和延迟信息可以反映在 NUMA 架构平台下物理资源的实时传输性能，为动态协调模块利用优化算法生成映射结果提供具体的数据参数，并且该模块还实时监控当前系统的负载情况为生成映射策略提供负载信息。

动态协调模块收到组链请求后，根据服务请求的类别来确定所参与服务的功能实例组，并通过本文所提出的基于贪心的优化算法，以信息采样模块的数据作为输入参数，挑选服务收益较高的功能组实例来组成服务链响应服务。下文中将具体介绍信息采样模块和动态协调模块的模块设计与实现。

通过这个两个和模块的数据传递和协同合作，并实现优化资源映射策略的生成和注入，从而达到设计目标。

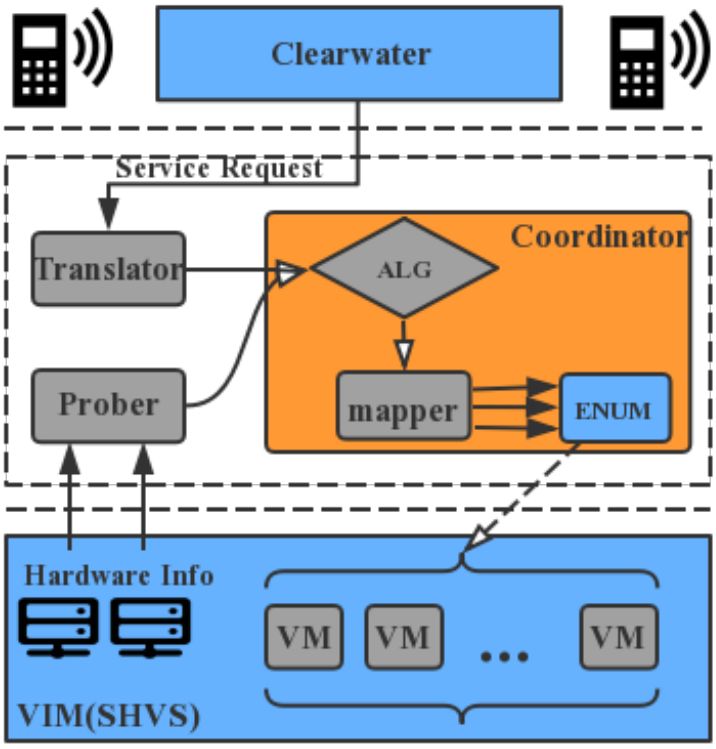


图 4–1: 映射优化系统架构示意图

Fig 4–1: Overview of the Optimization System

4.2 子模块设计与实现

4.2.1 Clearwater 服务链分析

为了保证设计系统的完整性目标，本节首先对 Clearwater 的服务链进行分析并确定各种服务所需的服务功能组，在保证服务完整性的前提下优化 Clearwater 服务的资源映射。Clearwater 平台的核心功能服务包括：**用户注册流程**，**初始化拨号流程**，**通话流程**和**通话终止流程**。根据总结的服务流程和对应的服务链，将每个服务链以链表的形式记录下来存储到预定义的配置文件中。当收到某一类服务请求时，根据服务请求类型在预定义的文件中选择对应的服务链功能实例组，完成服务链服务分析。完整的服务链分析如图4–2所示，各流程的完整描述如下：

用户注册流程 当客户端发起注册请求时，Sprout 节点将从 Dime 节点中获取认证信息，如果没有相关认证记录则请求拒绝，Sprout 节点同时缓存该请求记录，一旦后续出现相同请求则直接处理不再查询。如果查询到相关认证记录认证通过，则向客户端返回认证成功。如果有第三方认证，则跳转到第三方登记流程。该流程主要涉及到的网络功能节点为 Bono，Sprout 和 Dime。

初始化拨号流程 该流程主要参与的网络功能有 Sprout、Homestead 当客户端向 Sprout 节点发起通话请求，Sprout 节点通过 HTTP 请求从 Homestead 节点缓存中获取拨号者和被拨号信息，同时生成本次服务的 iFC[1]。随后 Sprout 向 ENUM 组件中查询所拨打用户是否在线，如果在线则 Sprout 完成查询终止 iFC，并根据查询结果向被拨打者发起 SIP INVITE 请求，同时向 AS(Vellum) 中记录 SIP Ringing 状态。当被拨叫客户端接听后，返回 SIP 200 OK，则通过 Sprout 节点更新 AS (Vellum) 记录。整个过程中所涉及到的功能节点为：Bono，Sprout 和 Dime。

通话流程 客户端通过 Sprout 不断在通话过程中不断更新状态信息到 AS(Vellum) 中，更新数据同样通过 Sprout 反馈到被叫客户端中。整个过程中所涉及到的功能节点为：Bono，Sprout，Vellum。

通话终止流程 当通话结束后，客户端发起挂断请求，Sprout 将状态存储到 AS(Vellum) 中，并向被拨号者发起 SIP BYE 请求，请求响应后同样更新 AS(Vellum) 记录，返回 SIP 200 OK，通话终止。整个过程中所涉及到的功能节点为：Bono，Sprout，和 Vellum。

根据本小节中对平台核心服务的分析，可以获得核心服务请求所对应的具体功能组以及服务链结构，动态映射模块将根据具体服务链结构进行映射策略的生成，从而保证所生成的服务的功能完整性。

[1] http://www.3gpp.org/ftp/Specs/archive/29_series/29.228/29228-b70.zip appendices B and F

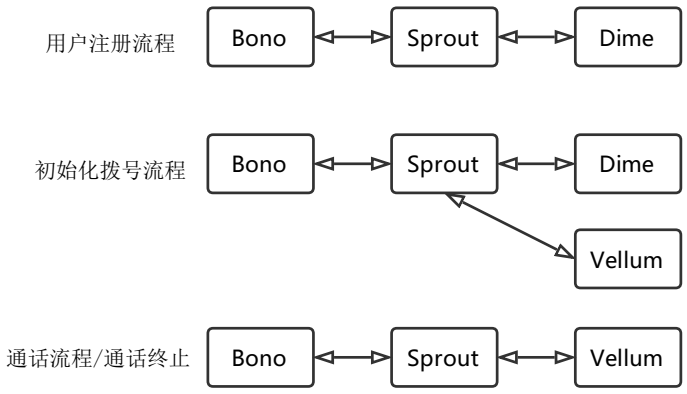


图 4–2: Clearwater 核心业务服务链
Fig 4–2: Core SFCs of Clearwater System

4.2.2 信息采样模块

信息采样模块负责从底层硬件获取运行平台架构信息和运行时各处理器节点的数据传输性能信息。在高性能的 NUMA 服务器上，先前的研究 [41, 42] 主要使用跳距离 (Hop-Distance) 来量化不同物理核间的数据传输性能。跳距离是高级配置与电源接口 (Advanced Configuration and Power Interface, ACPI) 定义的代表 NUMA 节点间的距离信息。我们可以使用 numactl[1]工具获取当前的机器的 NUMA 节点之间的距离信息。但是仅仅获取这样的信息不足以满足第三章中提出的模型所需的带宽和延迟信息。因此在信息采样模块中，本文提出通过预处理采样的方式将获取的带宽和延迟信息存储在矩阵 D 中，元素 $D_{i,j}$ 表示当使用第 i 号 CPU 访问在位于 j 号 CPU 时，所需要的带宽和延迟信息。该矩阵的大小和具体物理平台的处理器数量相关，即服务器有 N 个 CPU 时，该矩阵大小为 $N \times N$。

$$D = \begin{pmatrix} 10 & 8 & \cdots & 4 \\ 9 & 10 & \cdots & 5 \\ \vdots & \vdots & \ddots & \vdots \\ 4 & 3 & \cdots & 10 \end{pmatrix}$$

同时，该模块还负责对运行平台进行实时的监控和信息反馈，获取服务器当前的负载状

[1]https://linux.die.net/man/8/numactl

态来作为动态映射模块的实时输入参数。

为了满足如上设计中对底层平台信息的实时采样，本文使用经典的内存访问测试工具 Stream[1] 和 Intel Memory Latency Checker[2] 作为采样工具，预先运行并获取所有运行时参数，作归一化处理分别后存入如上的信息矩阵中。本文所使用的平台有 32 个物理核，即可以得到两个 32×32 的矩阵。所生成的信息参考矩阵被以文件的形式存储在系统硬盘上，供动态映射模块读取。

为了获取当前平台的实时负载信息，信息采样模块使用了一个动态的节点负载的监测工具。通过加载系统 msr 模块[3]来调用 PCM [4] 工具动态地监测每个物理核当前的负载信息。监测模块每秒更新一次并将结果输出至文件中。本文中重点关注如表格 4–1 中所列出的信息，其中主要选取 L3 cache 的命中率将作为当前处理器节点的负载参考指标，将这个信息输入作为算法4–1中参数 ϕ 的实际值。

表 4–1: 服务器实时监测参数

Table 4–1: Table of State Parameters of Running Server

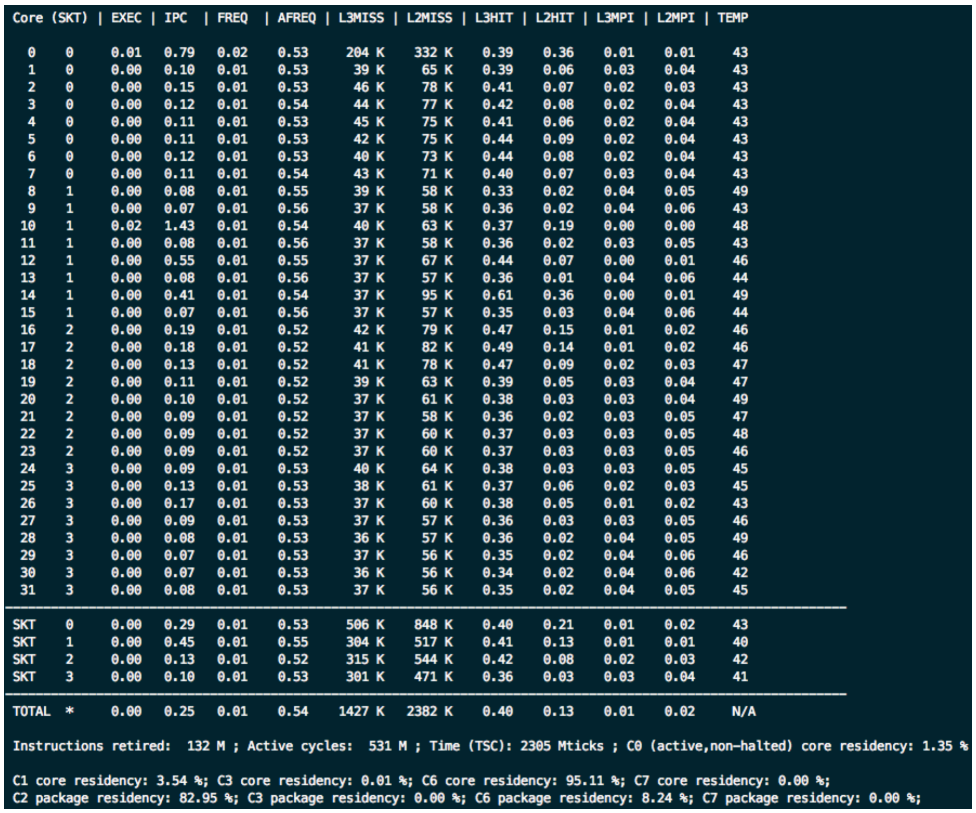
参数	参数涵义
IPC	当前 CPU 周期指令数
L3MISS	L3 cache 未命中次数
L2MISS	L2 cache 未命中次数
L3HIT	L3 cache 命中率 (0.00-1.00)
L2HIT	L2 cache 命中率 (0.00-1.00)
L3MPI	每个指令周期内的 L3cache miss 数
L2MPI	每个指令周期内的 L2cache miss 数
READ	内存读取数 (GBytes)
WRITE	内存写入数 (GBytes)

[1] https://www.cs.virginia.edu/stream/

[2] https://software.intel.com/en-us/articles/intelr-memory-latency-checker

[3] http://man7.org/linux/man-pages/man4/msr.4.html

[4] https://software.intel.com/en-us/articles/intel-performance-counter-monitor



Core	(SKT)	EXEC	IPC	FREQ	AFREQ	L3MISS	L2MISS	L3HIT	L2HIT	L3MPI	L2MPI	TEMP
0	0	0.01	0.79	0.02	0.53	204 K	332 K	0.39	0.36	0.01	0.01	43
1	0	0.00	0.10	0.01	0.53	39 K	65 K	0.39	0.06	0.03	0.04	43
2	0	0.00	0.15	0.01	0.53	46 K	78 K	0.41	0.07	0.02	0.03	43
3	0	0.00	0.12	0.01	0.54	44 K	77 K	0.42	0.08	0.02	0.04	43
4	0	0.00	0.11	0.01	0.53	45 K	75 K	0.41	0.06	0.02	0.04	43
5	0	0.00	0.11	0.01	0.53	42 K	75 K	0.44	0.09	0.02	0.04	43
6	0	0.00	0.12	0.01	0.53	40 K	73 K	0.44	0.08	0.02	0.04	43
7	0	0.00	0.11	0.01	0.54	43 K	71 K	0.40	0.07	0.03	0.04	43
8	1	0.00	0.08	0.01	0.55	39 K	58 K	0.33	0.02	0.04	0.05	49
9	1	0.00	0.07	0.01	0.56	37 K	58 K	0.36	0.02	0.04	0.06	43
10	1	0.02	1.43	0.01	0.54	40 K	63 K	0.37	0.19	0.00	0.00	48
11	1	0.00	0.08	0.01	0.56	37 K	58 K	0.36	0.02	0.03	0.05	43
12	1	0.00	0.55	0.01	0.55	37 K	67 K	0.44	0.07	0.00	0.01	46
13	1	0.00	0.08	0.01	0.56	37 K	57 K	0.36	0.01	0.04	0.06	44
14	1	0.00	0.41	0.01	0.54	37 K	95 K	0.61	0.36	0.00	0.01	49
15	1	0.00	0.07	0.01	0.56	37 K	57 K	0.35	0.03	0.04	0.06	44
16	2	0.00	0.19	0.01	0.52	42 K	79 K	0.47	0.15	0.01	0.02	46
17	2	0.00	0.18	0.01	0.52	41 K	82 K	0.49	0.14	0.01	0.02	46
18	2	0.00	0.13	0.01	0.52	41 K	78 K	0.47	0.09	0.02	0.03	47
19	2	0.00	0.11	0.01	0.52	39 K	63 K	0.39	0.05	0.03	0.04	47
20	2	0.00	0.10	0.01	0.52	37 K	61 K	0.38	0.03	0.03	0.04	49
21	2	0.00	0.09	0.01	0.52	37 K	58 K	0.36	0.02	0.03	0.05	47
22	2	0.00	0.09	0.01	0.52	37 K	60 K	0.37	0.03	0.03	0.05	48
23	2	0.00	0.09	0.01	0.52	37 K	60 K	0.37	0.03	0.03	0.05	46
24	3	0.00	0.09	0.01	0.53	40 K	64 K	0.38	0.03	0.03	0.05	45
25	3	0.00	0.13	0.01	0.53	38 K	61 K	0.37	0.06	0.02	0.03	45
26	3	0.00	0.17	0.01	0.53	37 K	60 K	0.38	0.05	0.01	0.02	43
27	3	0.00	0.09	0.01	0.53	37 K	57 K	0.36	0.03	0.03	0.05	46
28	3	0.00	0.08	0.01	0.53	36 K	57 K	0.36	0.02	0.04	0.05	49
29	3	0.00	0.07	0.01	0.53	37 K	56 K	0.35	0.02	0.04	0.06	46
30	3	0.00	0.07	0.01	0.53	36 K	56 K	0.34	0.02	0.04	0.06	42
31	3	0.00	0.08	0.01	0.53	37 K	56 K	0.35	0.02	0.04	0.05	45
SKT	0	0.00	0.29	0.01	0.53	506 K	848 K	0.40	0.21	0.01	0.02	43
SKT	1	0.00	0.45	0.01	0.55	304 K	517 K	0.41	0.13	0.01	0.01	40
SKT	2	0.00	0.13	0.01	0.52	315 K	544 K	0.42	0.08	0.02	0.03	42
SKT	3	0.00	0.10	0.01	0.53	301 K	471 K	0.36	0.03	0.03	0.04	41
TOTAL	*	0.00	0.25	0.01	0.54	1427 K	2382 K	0.40	0.13	0.01	0.02	N/A

Instructions retired: 132 M ; Active cycles: 531 M ; Time (TSC): 2305 Mticks ; C0 (active,non-halted) core residency: 1.35 %

C1 core residency: 3.54 %; C3 core residency: 0.01 %; C6 core residency: 95.11 %; C7 core residency: 0.00 %;
C2 package residency: 82.95 %; C3 package residency: 0.00 %; C6 package residency: 8.24 %; C7 package residency: 0.00 %;

图 4–3: 使用 PCM 工具获取运行时负载信息

Fig 4–3: Using PCM Tool Monitoring System Real-time Workload

4.2.3 动态映射模块

动态映射模块主要负责服务链资源映射策略生成和策略的部署。在该模块中，本设计按照服务链分析中所总结的各种服务类型，通过算法4–1映射成为具体的实例链表，并根据实例链表完成服务链的组建，生成对应的映射策略。根据生成的映射策略，覆盖平台原有的实例选择方法，完成映射过程。整个过程如图4–4所示。从整体架构上分析，实现本文中所设计动态映射模块包括以下关键步骤:

步骤 1、对底层平台进行预处理，获得与平台底层运行时相关的信息矩阵。与普通从硬件中读取参数不同，通过动态采样的方法获取平台运行时实时性能参数能更加准确地反映底层平台的架构差异以及所引起的性能差异。首先，动态采样可以实现与操作系统无关性能信息获取，不管多核平台运行的操作系统版本如何，都可以通过采样程序获取该操作系统下的实时运行结果。其次，相比于直接从主板芯片中读取多核架构中不同处理器节点的预定义信息，动态采样的方法通过动态地运行测试程序，可以获得更加接近实际情况的量化数据。基于这样的数据信息，可以更加准确地刻画底层物理资源之间

的数据传输性能。最终信息被存进格式化的信息矩阵中，作为步骤 2 的输入。

步骤 2、当一条服务请求到来时，分析该请求所对应的服务链。

步骤 3、根据请求所对应服务链的数据流向确定所需要的虚拟网络功能服务点，

步骤 4、使用基于贪心的方法并依照信息矩阵中的参数生成映射决策序列来确定待映射的具体运行实例，进入步骤 5。

步骤 5、根据步骤 4 生成映射的实例序列，依次检查实例在对应的物理资源节点当前的负载情况，如果某一实例所在节点的负载过高，增加负载会加剧资源竞争导致当前运行实例的性能下降，则返回步骤 4 将本次的映射序列标记为失效，重新生成新的实例映射序列。检查通过后进入步骤 6。

步骤 6、根据生成的映射序列，替代默认的随机选取策略，实现优化后的服务链到实际物理资源的映射。

4.2.3.1 基于贪心的映射算法

在上文建立的基于 IMS 服务链资源映射模型基础上，本小节主要介绍了在这样的应用场景下的一种基于贪心的资源映射策略算法。本设计的映射策略如算法4–1所示。本文提出两种基于贪心思想的映射策略。第一种是基于服务链上任意两个相邻节点具有最大通信带宽的最大带宽策略 (GMB, Greedy Maximum Bandwidth)，第二种是任意两个相邻节点具有最小通信延迟的最小延迟策略 (GML, Greedy Minimum Latency)。但带宽和延迟这两个影响因素同时可以提高整体映射策略的收益时，这两种策略可以同时被用于同一个服务链的映射生成。如果被映射的服务需要较高的通信带宽并根据其带宽来衡量其服务质量，例如数据传输服务，那么这种服务就适合使用 GMB 算法来生成映射结果；如果服务链的服务质量是由整体的通信延迟所决定的，那么 GML 便更适合这类的应用场景。这两种算法通过收益函数中的系数来具体调节。

算法 4–1 的具体步骤如下：服务链请求被解析成对应服务链链表输出后，算法 4–1 将该链表 L 作为算法输入，顺序遍历该链表，针对链表中每个元素 i, 利用公式 3–5 计算其收益函数，从待选实例中筛选出收益最大的实例并判断该实例所在计算节点的 L3cache 命中率，若未命中的比例超过 90% 则放弃该实例选择次优收益的实例并继续比较其 cache 的命中率，直到有合适的实例被选中为止，将被选中实例加入结果链表 L' 中。当筛选流程完后，检查 L'，若链表为空，则映射失败，否则根据链表中所选取的实例来组织相应的服务链。通过以上的算法 4–1，理论上我们可以选择出相对收益最高的服务链与实例的映射链表。算法的输出结果将作为输入传入动态映射模块，该模块将根据该结果进行服务链组链操作。

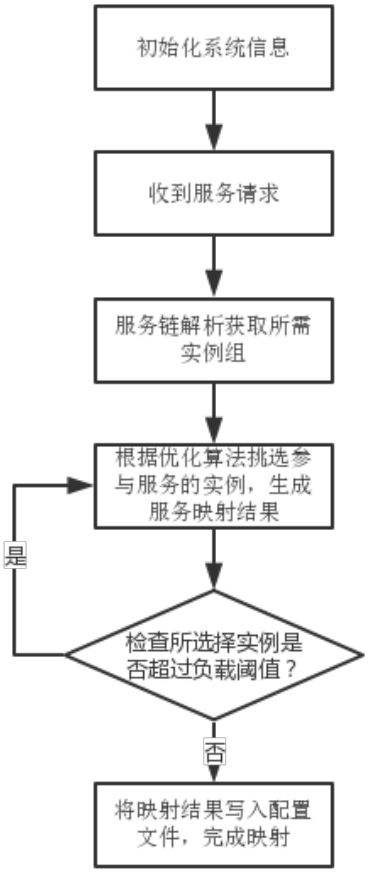


图 4–4: 动态映射模块工作流程

Fig 4–4: Flow of Coordinator Module Working Procedure

算法 4–1 基于贪心的映射策略

Start
Input List L of SFC S
Backup Substrate Network State
for Function $i \in L$ **do**
    Initialize: Capable Instances Set $L' = \emptyset$
    Got Max P from All Instances in the Function Domain D
    Got L3 Cache Miss Rate ϕ of Selected Node
    if $\phi < 90\%$ **then**
        $L' = L' \uplus j$
    else Delete Selected Instance from D and Pick New One with Max P
    end if
    $L' = L' \uplus j$
    if $L' \equiv \emptyset$ **then**
        Mapping Failed
        Reset Substrate Network Status **return**
    end if
    Sort L' According to L.
end for
Map the Function and Update D.
End

4.2.3.2 映射策略生成

在初始化阶段将所有的实例具体信息录入系统，包括其具体的网络功能、所分配的物理资源信息以及详细的配置参数。当有服务请求到来时，根据具体的业务请求来解析并翻译成该服务的响应服务链。此时动态映射模块根据算法4–1生成具体的映射策略，核心实现方法见下的 Coordinator.java 文件所示。

Coordinator.java

```
```
final public Map<Integer,VNF> getOptimumMapping(SFC sfc){
    ArrayList<VNF_TYPE> descriptor = sfc.getSFCDescriptor();//获取所需要的映射的
        服务链
    Map<Integer, VNF> result = new HashMap<>();

    Integer nextCpuId = -1;//初始化数据
    Integer preCpuId = -1;
    Integer curCpuId = -1;
    Double curLoad = 0.0

    for (VNF_TYPE type:descriptor) {
        int optimalChose = Integer.MAX_VALUE;
        VNF nextVNF = null;
        for (VNF vnf : runningVNFs) {
            if (vnf.getType() == type && vnf.isIdle) {
                //随机选取服务链中第一个网络功能
                if(curCpuId < 0) {
                    curCpuId = vnf.getVcpuNumber();
                    preCpuId = curCpuId;
                    vnf.setIdle(false);
                    result.put(curCpuId,vnf);
                    break;
                }
                curCpuId = vnf.getVcpuNumber();
                if (latancyMatrix[preCpuId/8][curCpuId/8] < optimalChose){
                    optimalChose = latancyMatrix[preCpuId/8][curCpuId/8];
                    nextCpuId = curCpuId;
                    nextVNF = vnf;
                }
            }
        }
        curLoad = getL3CacheMiss(nextCpuId); //获取所选取节点的L3CacheMiss率作为
        负载参考。
    if(nextCpuId != -1 && curLoad < 0.9) result.put(nextCpuId, nextVNF);
```
```

```
    }
    return result;
```
}
```

#### 4.2.3.3 映射策略注入

具体的映射策略生成后，接下来需要实现具体的映射策略注入操作。我们从实例列表中将所选出的实例的 ip 提出，通过修改 Clearwater 的 Sprout 节点中的 */etc/clearwater/enum.json* 文件和绑定规则，可以实现服务链的动态修改。平台提供了两种修改绑定的方式，分别为利用 BIND 服务或者 dnsmasq 服务。本设计使用 BIND 服务来修改 DNS 绑定，具体的配置文件如图4–5所示，并按照 *<enum domain name> <order> <preference> <flags> <service> <reg-exp>* 的语法规则来添加相应服务。当服务映射规则更新到配置文件中之后，Clearwater 系统会按照更新后的解析规则来选取实例完成组链，提供相应的服务功能。通过动态映射模块的从策略生成到配置文件修改这一些列操作，完成 IMS 服务链的动态组建。

```
$TTL 1h
@ IN SOA e164.arpa ngv-admin@example.com (
    2009010910 ;serial
    3600 ;refresh
    3600 ;retry
    3600 ;expire
    3600 ;minimum TTL
)
@ IN NS e164.arpa.
@ IN A <this server's IP address>

; Demo system number 2125550270
0.7.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
; Demo system number +12125550270
0.7.2.0.5.5.5.2.1.2.1 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:\\1@10.147.226.2!" .
; Clearwater external numbers 2125550271-9
*.7.2.0.5.5.5.2.1.2 IN NAPTR 1 1 "u" "E2U+sip" "!(^.*$)!sip:+1\\1@ngv.example.com!"
~
~
~
```

图 4–5: Enum 配置文件信息

Fig 4–5: Enum configuration overview

经过动态映射模块重新组建的服务链在虚拟资源的数据传输性能上充分考虑了底层运行平台的 NUMA 特性，通过结合信息采样模块中所获取的运行时带宽和延迟信息，生成优化后的虚拟资源映射组合来提供相应的服务。
```

4.3 本章小结

本章主要介绍了基于 Clearwater 平台的优化资源分配实现方法的设计与实现，主要包括本设计的设计目标、设计思路以及详细的模块设计分析与实现。在所建立模型的基础上，本文提出了设计的所关注的三个目标，并以这些目标作为约束详细阐述了本文的总体设计思路和模块的设计细节。最后，详细介绍了平台的总体算法以及底层信息采样模块和动态映射模块的一些实现细节信息。

第五章 实验分析

本章将对前文中设计并实现的 IMS 资源映射优化系统进行实验分析和性能测试。我们首先对使用的实验平台和实验环境进行详细的介绍，然后针对每一种具体的实验方案进行展开说明，包括进行实验数据归纳和分析。最后，对实验的总体效果和系统的实际收益进行总结。

5.1 实验平台和测试环境

实验中主要使用了一台浪潮英信大通量标准通用服务器，将其作为虚拟化平台，支持虚拟机的创建、运行和销毁。该测试服务器配置有 4 颗 8 核 2.13GHz 的 Intel Xeon 至强 E7-4830 CPU, 以及 128G 大小的 DDR3 内存，并且配有 1T 容量的标准容量 SATA 磁盘作为外部储存设备。为了真实考察所设计的系统在实际生产环境中所能达到的实际优化效果，实验测试的进行均在真实的实验室网络环境中，而运行的 NFV 应用为真实的 IP 多媒体子系统 Clearwater，可以提供语音视频通话业务。在软件配置方面，宿主机上安装的是 7.3 版本的 CentOS Linux 操作系统，使用系统自带的 KVM 作为虚拟化监视器，所有的虚拟机均安装的 14.04 版本的 Ubuntu 操作系统。除了 NFV 服务自身的软件栈，系统中没有安装运行其他多余的软件，以排除不必要的软件因素的干扰并且将系统运行时的额外性能损失降到最低。服务器的具体配置参数如表 5–1 所示。

表 5–1: 实验平台配置参数表

Table 5–1: Table of Testing Environment Configuration

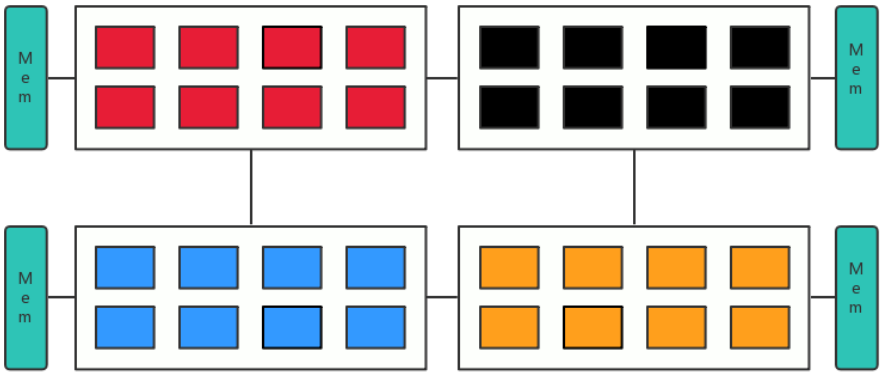
项目	配置
服务器	浪潮 NF8560M2, 4 节点 NUMA 架构
处理器	Intel Xeon CPU E7-4830, 2.13GHz x 4
缓存	24M L3, 256K L2, 64K L1
内存	每个节点 32GB DDR3, 共 128G
硬盘	1 TB SATA disk
操作系统	Hypervisor: CentOS Linux 7.3, Virtual Machine Ubuntu 14.04

从测试工具及实验手段的角度来说，本章中主要使用了 Clearwater 平台自带的压力测试工具和主流的性能测试和评估工具：Ping、iPerf[1]和 ApacheBench[2]。从具体的业务性能到通用的性能测试两个层次进行了深入、翔实的实验。从实验对象上来说，主要的实验集中在对系统网络性能的测试和评估上。从实验目的上来说，本章的实验设计主要从服务链的具体业务和通用网络性能两方面出发，分别对本文所提出的系统进行了测试。实验内容包括：1.Clearwater 压力测试工具的性能测试对比。2.Ping、iPerf、ApacheBench 工具的基础性能测试对比。

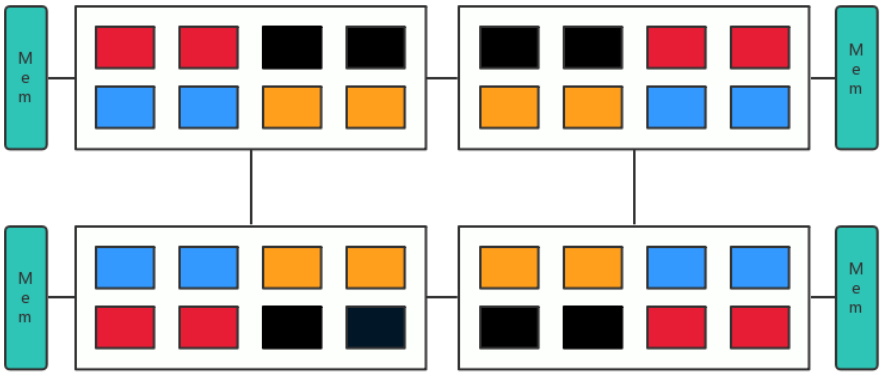
除了使用以上所述的测试工具，本文充分考虑了在实际环境下各种网络功能实例在多核物理机上的分布情况，故本文列举了三种可能的实例分布方式如图 5–1 所示，分别为离散分布、随机分布和聚合分布。图中以不同的色块来代表不同的网络功能，以实例所绑定的物理 CPU 来代表某个实例的物理资源分配情况。可以看出，在离散分布的情形下，相同的虚拟网络功能实例聚集在同一个处理器节点中，在这样的资源分布下，所有虚拟机实例间具有相近的物理资源亲和度。在聚合分布下，每个处理器节点上都被均匀地绑定了实现网络服务链的所有功能实例。而在随机分布的情形下，所有的网络功能实例以随机的方式绑定在所有的物理 CPU 上。本文后续的所有本文后续的所有测试将覆盖这三种资源分布的前提，以检验本设计系统在实际应用场景下的有效性。

[1]https://iperf.fr/

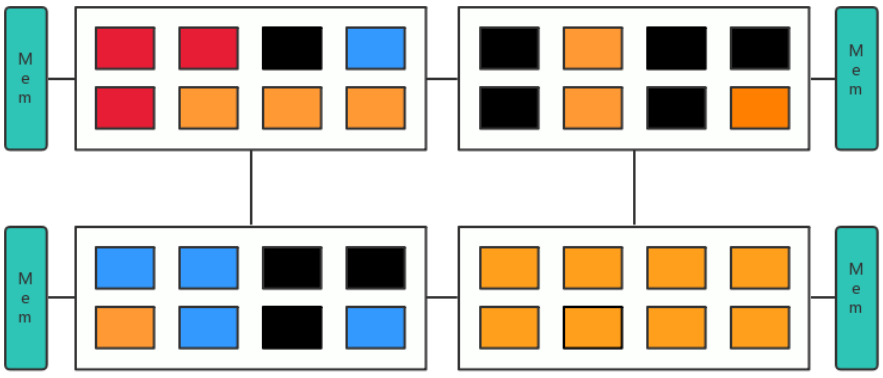
[2]http://httpd.apache.org/



(a) 离散分布



(b) 聚合分布



(c) 随机分布

图 5–1: 实例分布示意图

Fig 5–1: Resource Distribution

5.2 Clearwater 压力测试实验

本文以按照分布式的方式在配置如5–1所示的大通量标准服务器中部署了 Clearwater 所需的所有功能节点，以 KVM 作为 Hypervisor，所有虚拟机均配置了单个虚拟核、1G 的运行内存和 10G 的磁盘容量，并利用 Virtio 的网络虚拟化方式提供虚拟网卡。测试负载方面，本文使用了 Clearwater 平台所提供的测试工具集，选择了其中的标准语音通信业务中用户注册-解注册业务 *reg-dereg* 作为本文的测试负载。一次 *reg-dereg* 服务由三个请求组成，分别为：注册请求、认证请求和注销请求。其中任意一个请求一旦超过了最大返回等待时间 (10s)，则该次请求失败。为了测试系统默认随机策略的性能，我们设定每次实验指定运行 300s，运行 10 次后去平均值作为最终结果。测试工具可以指定不同的初始请求速率来测试不同负载强度下的平均请求响应延迟。使用从发起注册请求到最后一个请求的 200 回执为止的平均时间作为衡量延迟的具体参数，单位为毫秒。最后以平均的延迟时间和请求的成功率来作为性能比较的依据。

压力测试实验结果如图 5–2 所示，本文测试了在 light (50)，light-medium(100)，medium(500)，heavy(1000) 四种请求速率下默认系统与本设计所实现的系统在性能上的差异，在这四种负载下分别测试了三种不同分布下的性能结果作为对照组。深色的柱状条代表默认的随机映射系统的测试结果，浅色的柱状条代表在本设计优化过后的结果。为了方便比较，本文对数据进行了归一化处理，这样能够比较直观的体现出优化前后的性能对比。根据图 5–2 ，可以看出在四种不同大小的负载下，本设计的系统在离散分布和随机分布中都取得了较好的优化效果，服务的延迟平均有接近 40% 的下降，而在聚合分布的情况下，服务在优化前后并没有表现出较大的性能差别。进一步的，对比三种分布下的基准和优化的平均性能，不难发现聚合分布的平均性能最高随机分布次之而离散分布最差，而经过本设计的优化之后，离散和随机分布下的平均延迟均低于聚合分布。这样的测试结果基本上是符合测试结果预期的。首先，针对离散和随机分布，在默认条件下 Clearwater 系统会随机的选取实例进行组链，在完全没有考虑资源间的数据传输差异的前提下，所产生服务链就可能会有较大的平均服务延迟，而且由于资源分布的离散程度，不同资源之间的亲和度具有不同级别的差异；而在聚合分布下，所有的功能实例之间具有相近的数据传输性能，平均性能比较稳定，可以优化的空间较小，故默认系统和优化后的系统并没有显示出明显的性能差异。综合来看，本设计的优化系统在大部分情况下能够有效地提升 Clearwater 服务的性能。

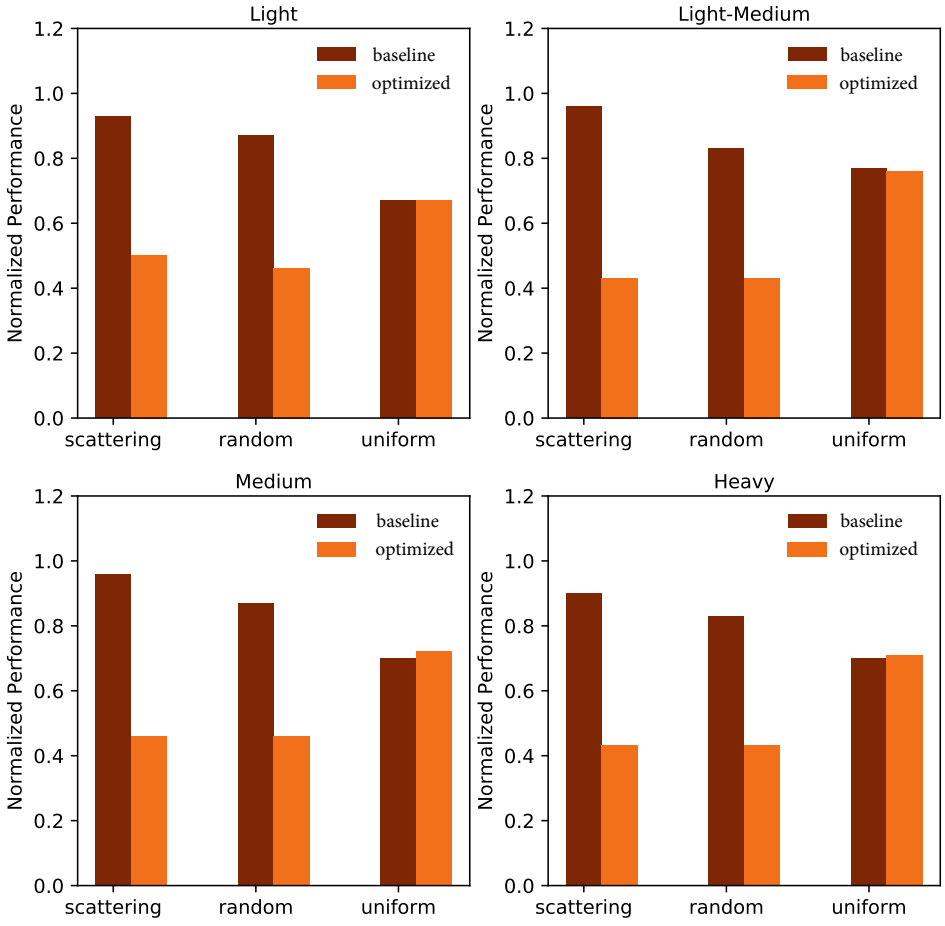


图 5–2: Clearwater 压力测试结果图

Fig 5–2: Clearwater Performance Test

5.3 基础网络性能对比实验

本文除了使用 Clearwater 自带的压力测试工具测量 IMS 业务性能之外，也使用了一些基础的测试工具对 IMS 业务进行模拟测试，用来验证本设计系统的通用型和优化的有效性。为了模拟实际 NFV 服务的网络功能服务链，本文将优化系统所选择的服务链实例挑选出来与系统随机选择的服务链做对比，分别用 Ping，iPerf 和 ApacheBench 来进行端对端的性能测试，并将其结果累计算作服务链的性能结果进行比较和验证优化系统的有效性。同样的，本文测试了在三种不同实例分布的下的网络功能服务链映射结果。在基础测试中，本文以默认的随机生成策略的测试结果作为基准数据 (Baseline)，使用优化后的测试数据进行对比。

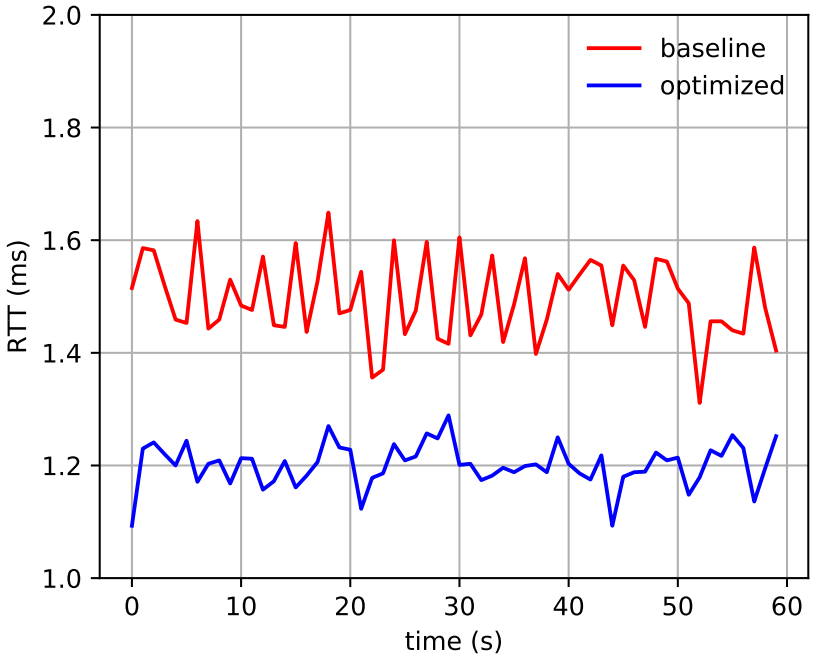
5.3.1 Ping 实验

在本小节中，本文使用了轻量级的网络命令 Ping 来测试系统的累计网络延迟性能。Ping 是一个非常轻量级的网络命令，其运行过程中的所引起的 CPU 和内存额外开销可以忽略不计，因而在网络测试中 Ping 命令经常被用来测试网络的延迟大小，测试的结果基本与真实网络链路上产生的延迟相等。在小节的实验中，所有的虚拟机依然采用单个虚拟核、1G 运行内存和 10G 磁盘的配置，测试的虚拟机使用 Ubuntu 14.04 Server 版本的系统。为了获取服务链累计的延迟信息，本文设置了链路上每一个上游节点都同时向自己相邻的下游节点进行 Ping 请求，并将每台客户端的测试结果输出并统计。实验所使用的发包时间间隔为 0.5 秒，测试时间长度为 1 分钟。

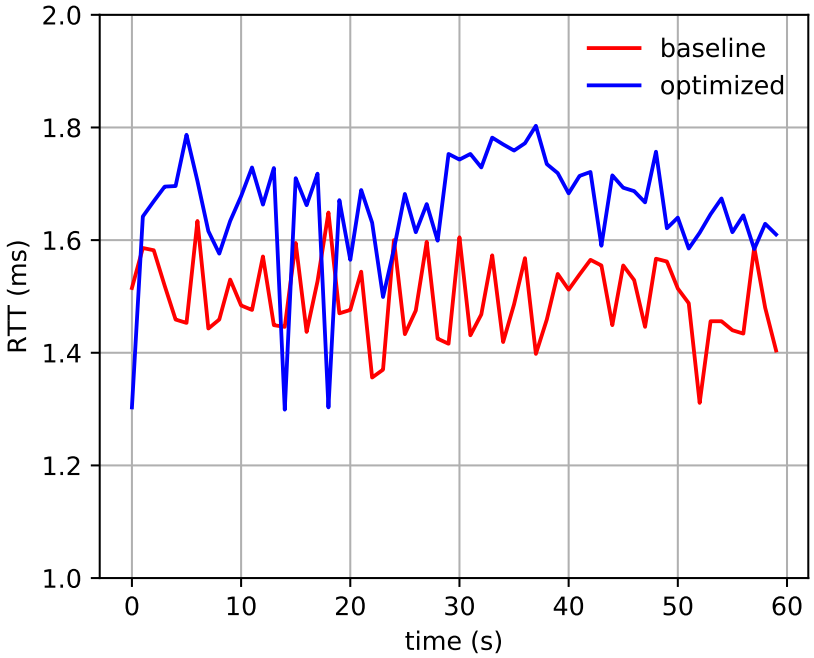
如图 5–3 所示, 红色的折线代表默认的服务链映射策略所选的实例组合所产生的累计延迟，蓝色的折线代表优化系统所选择实例所产生的累计延迟。在统计过程中发现，虽然端到端的 Ping 测试结果差异比较小，但是当服务链的所有节点的延迟累计起来后，延迟的下降就相对明显了。在离散和随机分布中，如图 5–3(a) 和图 5–4(c) 所示，本文观测到了对于每条服务链，都有接近 0.2ms 的延迟下降，下降的比例接近 15%。而在聚合分布下，优化前后的性能差异也并不明显。对比三种分布下的延迟数据，可以看出，在离散分布的设定下，服务链可以达到最低的累计延迟，相比于最高的随机分布，有了接近 30% 的延迟下降。

5.3.2 iPerf 实验

本节中，本文使用 iPerf 工具来测试系统的网络带宽性能。iPerf 是一套可以动态测量基于 IP 网络的最大执行带宽的测试工具集。它支持调整执行时间、协议和缓冲区大小来进行性能调优。本小节的实验中使用的是更加轻量级的 iPerf3 来进行测试，测试虚



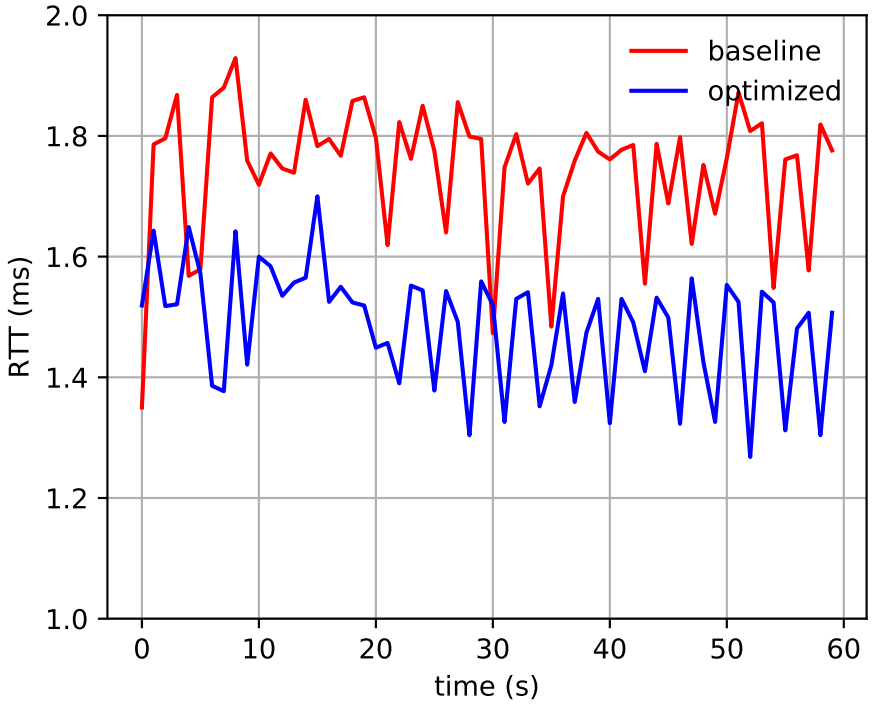
(a) 离散分布下 Ping 测试结果



(b) 聚合分布下 Ping 测试结果

图 5–3: Ping 测试结果图

Fig 5–3: Ping Test Result



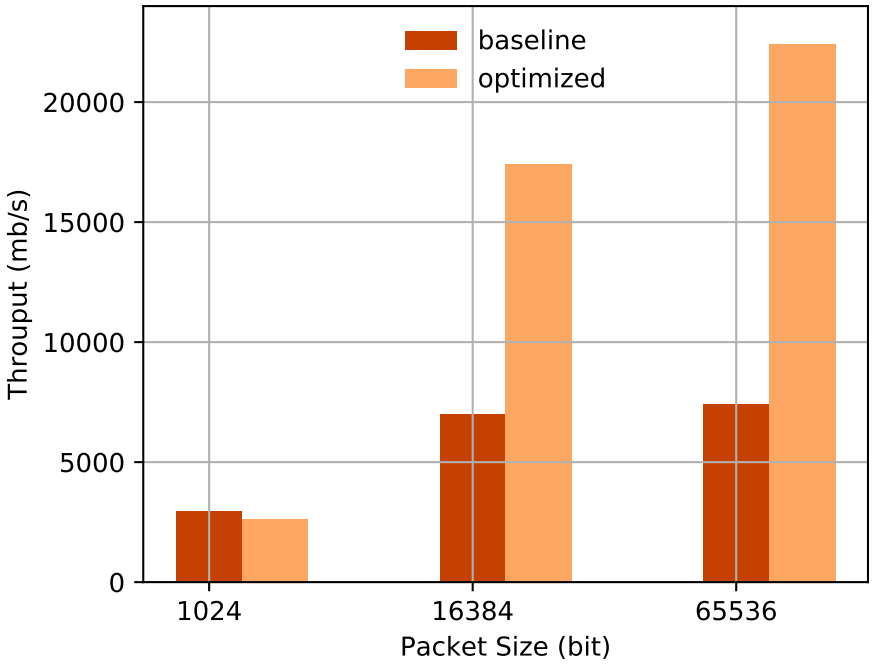
(c) 随机分布下 Ping 测试结果

图 5–3: Ping 测试结果图（续）

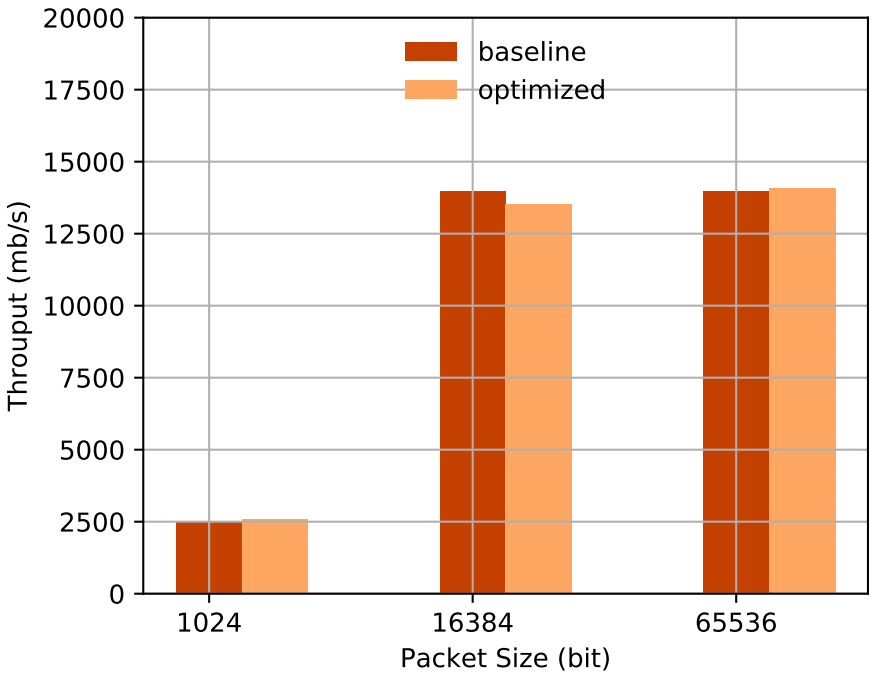
Fig 5–3: Ping Test Result (Con't)

拟机的而配置与 Ping 实验中的相同。iPerf 测试需要手动配置客户端和服务端，因此在所有运行的虚拟机中都配置 iPerf 服务监听端口，当获取实际链路组合后，以链路上每个前置节点作为客户端向相邻的后置节点发送数据，统计链路上的累计带宽。单次实验的执行时间为 30 秒，分别使用了数据包大小为 1K、4K、16K 的测试负载来针对三种不同的实例分布进行了测试，每组实现均执行 10 次取平均值作为最后的输出结果。

如图 5–4 所示，深色柱状条代表默认随机的选取策略所选取的服务链实例组合在 iPerf 测试场景下的性能测试结果，浅色斜杠条纹柱状条则代表本文优化策略所选取的服务链组合的测试性能测试结果。通过统计可以明显发现，在离散分布和随机分布的情况下，服务链的累计带宽有了明显的提升。特别是如图 5–4(a) 所示，在离散分布的 16K 数据包的测试对照组中，优化后的服务链相比于随机选择的组合，带宽有了接近 3 倍的提升。但是在 1K 数据包的测试对照组下，可以发现优化前后的性能变化并不大。通过监测实验时的实时 CPU 利用率，可以发现当发送 1K 数据包时，所有的运行实例的 CPU 接近满负载。结合已知的网络和操作系统知识来分析，在小包的情况下，限制网络



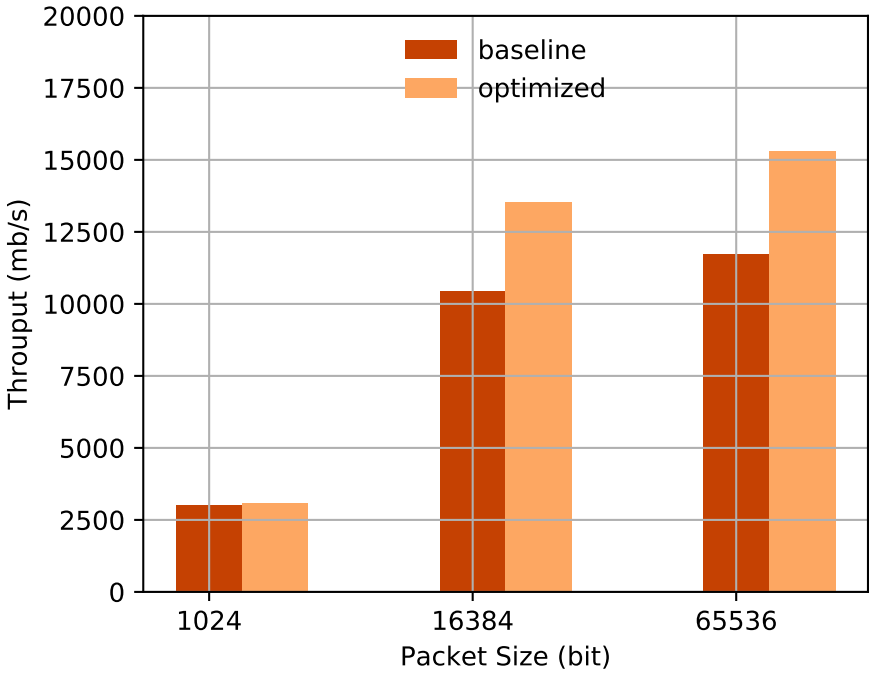
(a) 离散分布下的 iPerf 测试结果



(b) 聚合分布下的 iPerf 测试结果

图 5–4: iPerf 测试结果图

Fig 5–4: iPerf Test Result



(c) 随机分布下的 iPerf 测试结果

图 5–4: iPerf 测试结果图（续）

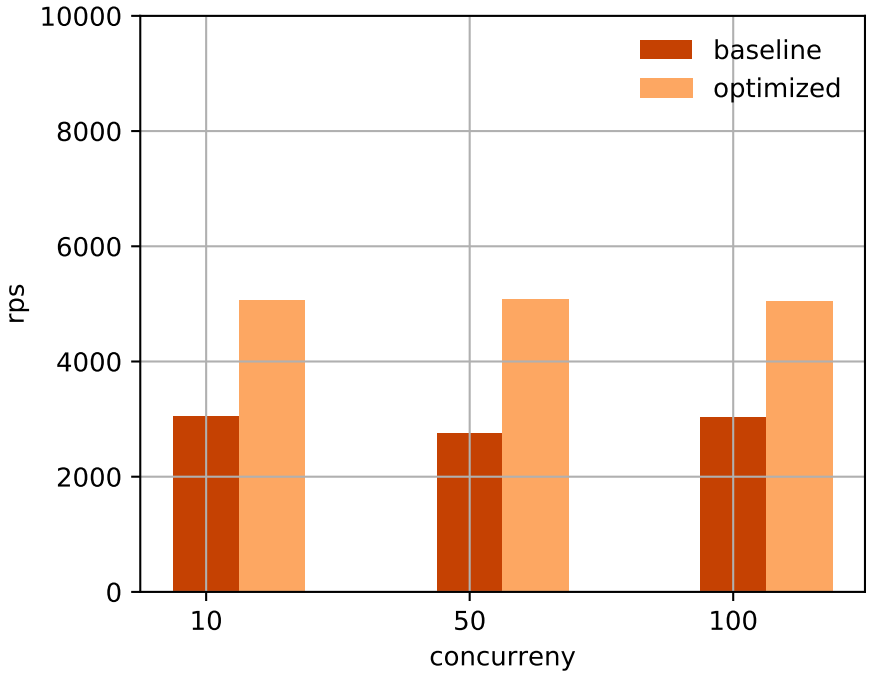
Fig 5–4: iPerf Test Result (Con't)

带宽的瓶颈在于频繁的数据包处理，数据包数量越多，则 CPU 处理所需要的开销越高，故在小包情况下，服务链上的资源亲和度并没有对性能有主导影响。而随着数据包大小的提升，可以发现此时各实例之间的资源亲和度的影响逐渐增大，当数据包大小为 16K 时，更是出现了接近 3 倍的性能差距。这样的实验结果更是可以证明本设计系统相比于随机策略的优越性。在分配相同的物理资源的条件下，本设计可以获得最大接近 3 倍的网络带宽提升。

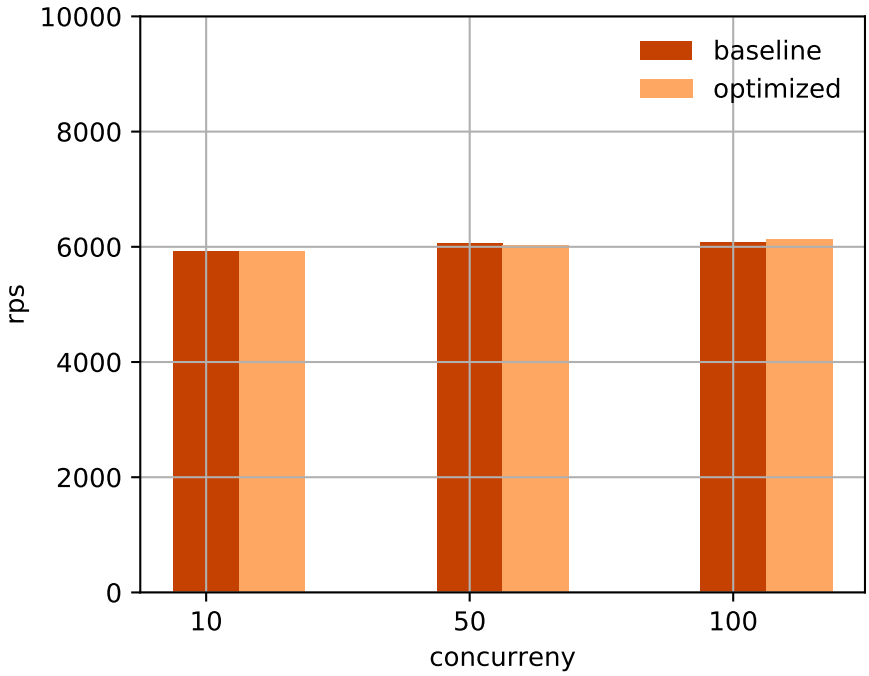
5.3.3 ApacheBench 实验

本节中，本文使用 ApacheBench 来测试在模拟真实业务下 NFV 服务链的标准性能。ApacheBench 是一款用于测量 HTTP Web 服务器性能的测试工具，最初作为 Apache HTTP Server 的测试工具，后来被扩展为可以用于测试任何 Web 服务器。IMS 业务纷繁复杂，仅仅使用不同包大小的数据从微观的角度去衡量服务链的性能在实际应用场景下缺乏说服力，所以本节中使用真实的 Web 服务来测试在真实网络应用场景下的实际性能。在实际应用场景下，业务请求可能包括不同大小不同长度的数据包，在这样的测试场景下进行测试更具有实际意义。ApacheBench 的测试原理是由客户单向目标服务器发起资源请求，可以通过命令来创建不同数量的并发访问线程，模拟多个访问者对同一个 URL 地址进行访问。测试工具会持续访问资源直到完成预先设置的访问总数，并输出每秒所完成的请求次数。本文仿照之前的测试设置，使用了相同的虚拟机配置，在不同映射策略下的虚拟机之间根据服务链数据流的先后关系，进行了累计请求统计。除了不同的实例分布场景这个变量之外，本实验还选择不同请求并发数来考察在不同数据压力负载下的的性能表现，预先设置的访问总数为 10 万次，每次测试重复 10 次取平均结果来作为衡量依据。

ApacheBench 的测试结果如图 5–5 所示，深色的柱状条代表默认的随机选取策略的服务链在 Apache Bench 测试场景下的每秒的所能达到的累计请求完成数，浅色斜杠条纹柱代表在优化策略下测试组的测试结果。通过测试数据的整理可以发现，在线程数为 10、50、100 的并发请求下，优化后的服务链累计吞吐在离散分布下由 3000 rps 左右提升到了 5000 rps 左右，在随机分布下由 6000 rps 提升到了 7500 rps 左右。根据实验所得数据可以看出在模拟实际应用负载下，本设计的优化系统的性能数据也是高于随机选择的。



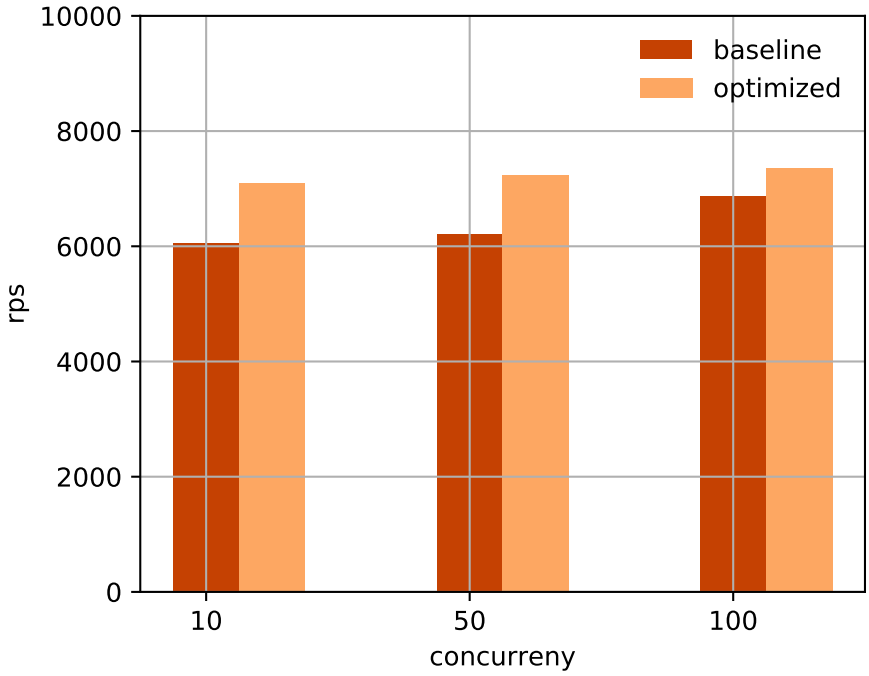
(a) 离散分布下的 Apache Bench 测试结果



(b) 聚合分布下的 Apache Bench 测试结果

图 5–5: ApacheBench 测试结果图（续）

Fig 5–5: ApacheBench Test Result



(c) 随机分布下的 ApacheBench 测试结果

图 5–5: ApacheBench 测试结果图（续）

Fig 5–5: ApacheBench Test Result (Con't)

5.4 本章小结

本章主要介绍了基于底层感知的高性能 NFV 实现系统实验相关的部分内容。首先，本章介绍了实验所采用的软硬件平台，包括物理机的配置、虚拟化环境的和 Clearwater 平台的配置，引入了不同实例分布情况的分类说明。其次，本章对优化前后的系统进行了 Clearwater 应用的压力测试和基础网络性能测试的对比实验。压力测试用的是 Clearwater 提供的压力测试脚本工具，基础网络性能测试使用的工具包括 Ping、iPerf、ApacheBench 三种。压力测试所涵盖的性能测试主要包括请求的响应时间和成功率，基础网络测试所涵盖的性能点主要包括网络 I/O 的吞吐率、网络响应速率和网络的带宽。从实验数据的角度上证明了本文所提出的系统对 Clearwater 应用所带来的可观的性能提升，并且在不同的虚拟机分布情况下展现出了有效的优化效果。

第六章 全文总结

6.1 主要结论

网络功能虚拟化为传统电信业务描绘了十分美好的前景，是对已有电信架构的一次革新，大大提升了电信业务的开发效率，缩短了开发周期，降低了部署的成本。通过与现有的云计算基础设施相结合，网络功能虚拟化将实现服务的快速部署、伸缩和升级。然而，在从专有硬件向通用服务器迁移，软件化网络功能的过程中，不可避免地会遇到性能和服务质量下降的问题。特别是考虑到现有的大容量通用服务器中，多核的非一致性存储架构是被采用的主流架构，虚拟化的物理资源之间存在亲和度差异，不同亲和度的物理资源（CPU，内存等）之间的数据访问也会有较大的性能差异。这样的底层物理特性结合网络功能虚拟化中特殊的服务链结构，会引起很高的性能波动，导致服务质量下降。

本设计的研究内容主要着眼于通用的多核服务器中网络功能服务链的资源映射问题。在由单核虚拟机所组成的服务节点集群中，上层应用对特定网络服务的需求仅仅以服务链的方式下发，对于具体组成服务链的物理实例的资源分布并不知晓，而在多核的非一致性存储访问平台下，被分配不同物理资源的虚拟机彼此之间具有不同的资源亲和度，即实例虚拟机间访问性能与其物理资源的数据传输性能。这种底层资源的亲和度关系对于上层应用是透明的，也就是说上层服务在组织时忽视了到这层底层信息。因此在组织服务链提供服务时，可能会出现较大的性能波动。然而，正是多核架构存在的这种亲和度关系，给组织网络功能服务链时提供了新的解决思路。

基于以上的考虑，本设计中利用多核物理核的非一致性存储访问和虚拟化相关技术，在 IMS 服务平台下对服务链的资源映射问题进行了研究。当服务请求被发出时，通过解析请求，获取实际服务链的组成情况，结合底层资源亲和度的采样信息，采用基于贪心的搜索方法，并同时考虑当前的运行负载，避免出现局部过热或资源竞争严重的不利情况，构造出理论上亲和度较好的服务链映射策略。之后将该策略应用到实际服务链的组链中，达到优化服务链性能的目的。

最后，本文在实际的大容量通用服务器上，利用真实的网络服务 Clearwater 来验证了本设计的实现原型，并在真实的网络环境中对本设计进行了性能的评估。数据分析的结果表明，本设计在实验环境中可以平均可以减少 40% 具体 IMS 业务服务延迟，并且在基础网络性能测试中，分别有最高 30% 的延迟下降，300% 的带宽提升和 20% 到 60% 的吞吐提升。在使用相同数量的物理资源前提下，本文的设计可以提高网络服务的

性能。

6.2 研究展望

本设计虽然已经在通用的服务器上实现了初级原型并获得了一定了性能提升，但仍然存在一些可以提升和改进的空间。首先，本设计从单台服务器的物理资源亲和度角度出发，而在实际生产环境中，尤其是云计算数据中心中，机架上的服务器都是以高性能网卡相连所组成的服务器集群，本文的资源映射模型可以扩展到多台物理机的范围。另外本文所提出的映射算法在问题规模较小的前提下具有很好的效果，但是当问题规模扩展到一定级别，算法的执行效率会大大下降，这时候需要结合一些已有的高级算法来提升算法的时间消耗以满足实际生产中的响应需求。

希望本文所完成的工作能够具有一些抛砖引玉的积极作用，为其他研究者在网络功能虚拟化及虚拟机调度方向的研究提供有益的帮助和参考。

参考文献

[1] ETSI G N. 001:" Network Functions Virtualisation (NFV)[J]. Architectural Framework, 2013.

[2] ZAVE P, FERREIRA R A, ZOU X K, et al. Dynamic service chaining with dysco[C]// Proceedings of the Conference of the ACM Special Interest Group on Data Communication. ACM. [S.l.]: [s.n.], 2017: 57–70.

[3] KULKARNI S G, ZHANG W, HWANG J, et al. NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains[C]// Proceedings of the Conference of the ACM Special Interest Group on Data Communication. ACM. [S.l.]: [s.n.], 2017: 71–84.

[4] MIJUMBI R, SERRAT J, GORRICHO J.-L, et al. Network function virtualization: State-of-the-art and research challenges[J]. IEEE Communications Surveys & Tutorials, 2016, 18(1): 236–262.

[5] Project Clearwater Release 1.0. `https://media.readthedocs.org/pdf/clearwater/latest/clearwater.pdf`.

[6] Kamailio. `https://www.kamailio.org/w/`.

[7] RIZZO L. Netmap: a novel framework for fast packet I/O[C]// 21st USENIX Security Symposium (USENIX Security 12). [S.l.]: [s.n.], 2012: 101–112.

[8] RAM K K, COX A L, CHADHA M, et al. Hyper-Switch: A Scalable Software Virtual Switching Architecture.[C]// USENIX Annual Technical Conference. [S.l.]: [s.n.], 2013: 13–24.

[9] BELAY A, PREKAS G, KLIMOVIC A, et al. IX: A protected dataplane operating system for high throughput and low latency[C]// Proceedings of the 11th USENIX Symposium on Operating System Design and Implementation (OSDI). EPFL-CONF-201671. USENIX. [S.l.]: [s.n.], 2014.

[10] HWANG J, RAMAKRISHNAN K K, WOOD T. NetVM: high performance and flexible networking using virtualization on commodity platforms[J]. IEEE Transactions on Network and Service Management, 2015, 12(1): 34–47.

[11] YASUKATA K, HONDA M, SANTRY D, et al. StackMap: Low-Latency Networking with the OS Stack and Dedicated NICs.[C]// USENIX Annual Technical Conference. [S.l.]: [s.n.], 2016: 43–56.

[12] PREKAS G, KOGIAS M, BUGNION E. ZygOS: Achieving Low Tail Latency for Microsecond-scale Networked Tasks[C]// Proceedings of the 26th ACM Symposium on Operating Systems Principles. EPFL-CONF-231395. [S.l.]: [s.n.], 2017.

[13] INTEL D. Data plane development kit. 2015.

[14] XIE Y, LIU Z, WANG S, et al. Service Function Chaining Resource Allocation: A Survey[J]. ArXiv preprint arXiv:1608.00095, 2016.

[15] HERRERA J G, BOTERO J F. Resource allocation in NFV: A comprehensive survey[J]. IEEE Transactions on Network and Service Management, 2016, 13(3): 518–532.

[16] BELBEKKOUCHE A, HASAN M M, KARMOUCH A. Resource discovery and allocation in network virtualization[J]. IEEE Communications Surveys & Tutorials, 2012, 14(4): 1114–1128.

[17] FISCHER A, BOTERO J F, BECK M T, et al. Virtual network embedding: A survey[J]. IEEE Communications Surveys & Tutorials, 2013, 15(4): 1888–1906.

[18] MIJUMBI R, SERRAT J, GORRICHO J.-L, et al. Design and evaluation of algorithms for mapping and scheduling of virtual network functions[C]// Network Softwarization (NetSoft), 2015 1st IEEE Conference on. IEEE. [S.l.]: [s.n.], 2015: 1–9.

[19] LIN T, ZHOU Z, TORNATORE M, et al. Demand-aware network function placement[J]. Journal of Lightwave Technology, 2016, 34(11): 2590–2600.

[20] BHAMARE D, JAIN R, SAMAKA M, et al. A survey on service function chaining[J]. Journal of Network and Computer Applications, 2016, 75: 138–155.

[21] QUINN P, NADEAU T. Problem statement for service function chaining[J]. 2015.

[22] GHAZNAVI M, KHAN A, SHAHRIAR N, et al. Elastic virtual network function placement[C]// Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on. IEEE. [S.l.]: [s.n.], 2015: 255–260.

[23] MARTINS J, AHMED M, RAICIU C, et al. ClickOS and the art of network function virtualization[C]// Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation. USENIX Association. [S.l.]: [s.n.], 2014: 459–473.

[24] POPURI S. A tour of the Mini-OS kernel. 2014.

[25] BARHAM P, DRAGOVIC B, FRASER K, et al. Xen and the art of virtualization[C]// ACM SIGOPS operating systems review. Vol. 37. 5. ACM. [S.l.]: [s.n.], 2003: 164–177.

[26] PFAFF B, PETTIT J, KOPONEN T, et al. The Design and Implementation of Open vSwitch.[C]// NSDI. [S.l.]: [s.n.], 2015: 117–130.

[27] KOHLER E, MORRIS R, CHEN B, et al. The Click modular router[J]. ACM Transactions on Computer Systems (TOCS), 2000, 18(3): 263–297.

[28] KIVITY A, KAMAY Y, LAOR D, et al. Kvm: the Linux virtual machine monitor[C]// Proceedings of the Linux symposium. Vol. 1. [S.l.]: [s.n.], 2007: 225–230.

[29] RUSSELL R. Virtio: towards a de-facto standard for virtual I/O devices[J]. ACM SIGOPS Operating Systems Review, 2008, 42(5): 95–103.

[30] PANDA A, HAN S, JANG K, et al. NetBricks: Taking the V out of NFV.[C]// OSDI. [S.l.]: [s.n.], 2016: 203–216.

[31] HSIEH C.-H, CHANG J.-W, CHEN C, et al. Network-aware service function chaining placement in a data center[C]// Network Operations and Management Symposium (APNOMS), 2016 18th Asia-Pacific. IEEE. [S.l.]: [s.n.], 2016: 1–6.

[32] MIJUMBI R, SERRAT J, GORRICHO J.-L, et al. Management and orchestration challenges in network functions virtualization[J]. IEEE Communications Magazine, 2016, 54(1): 98–105.

[33] HU Y, LI T. Towards efficient server architecture for virtualized network function deployment: Implications and implementations[C]// Microarchitecture (MICRO), 2016 49th Annual IEEE/ACM International Symposium on. IEEE. [S.l.]: [s.n.], 2016: 1–12.

[34] SUGERMAN J, VENKITACHALAM G, LIM B.-H. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor.[C]// USENIX Annual Technical Conference, General Track. [S.l.]: [s.n.], 2001: 1–14.

[35] AHMAD I, ANDERSON J M, HOLLER A M, et al. An analysis of disk performance in VMware ESX server virtual machines[C]// Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on. IEEE. [S.l.]: [s.n.], 2003: 65–76.

[36] DONG Y, XU D, ZHANG Y, et al. Optimizing network I/O virtualization with efficient interrupt coalescing and virtual receive side scaling[C]// Cluster Computing (CLUSTER), 2011 IEEE International Conference on. IEEE. [S.l.]: [s.n.], 2011: 26–34.

[37] AMSDEN Z, ARAI D, HECHT D, et al. VMI: An interface for paravirtualization[C]// Proc. of the Linux Symposium. [S.l.]: [s.n.], 2006: 363–378.

[38] CHOWDHURY N M K, RAHMAN M R, BOUTABA R. Virtual network embedding with coordinated node and link mapping[C]// INFOCOM 2009, IEEE. IEEE. [S.l.]: [s.n.], 2009: 783–791.

[39] FAJJARI I, SAADI N A, PUJOLLE G, et al. VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic[C]// Communications (ICC), 2011 IEEE International Conference on. IEEE. [S.l.]: [s.n.], 2011: 1–6.

[40] MANCO F, LUPU C, SCHMIDT F, et al. My VM is Lighter (and Safer) than your Container[C]// Proceedings of the 26th Symposium on Operating Systems Principles. ACM. [S.l.]: [s.n.], 2017: 218–233.

[41] TUDOR B M, TEO Y M, SEE S. Understanding off-chip memory contention of parallel programs in multicore systems[C]// Parallel Processing (ICPP), 2011 International Conference on. IEEE. [S.l.]: [s.n.], 2011: 602–611.

[42] PILLA L L, RIBEIRO C P, CORDEIRO D, et al. A hierarchical approach for load balancing on parallel multi-core systems[C]// Parallel Processing (ICPP), 2012 41st International Conference on. IEEE. [S.l.]: [s.n.], 2012: 118–127.

攻读学位期间发表的学术论文

[1] 第一作者. EI 国际会议论文, 2017.

攻读学位期间申请的专利

[1] 第二发明人，“虚拟化多核环境下非一致性 I/O 访问虚拟机资源迁移方法”，专利申请号 201610415935.5

[2] 第二发明人，“一种基于底层 NUMA 感知的 NFV 实现方法”，专利申请号 291710209194.X