并行與分布式計算通訊

地　址：武汉市华中科技大学东五楼二楼
邮　编：430074
电　话：（027）87541924 或 87543529
传　真：（027）87557354
E-mail：wwu.hust@hust.edu.cn
网　址：http://grid.hust.edu.cn

服务计算技术与系统教育部重点实验室
集群与网格计算湖北省重点实验室

并行與分布式計算通訊

BING XING YU FEN BU SHI JI SUAN TONG XUN

2017年第3期　总第30期　2017年09月

封面人物：牛超——厚积薄发，循序渐进

2016-2017年度实验室各研究方向成果展示

浅谈中断虚拟化

感恩

http://grid.hust.edu.cn

教育部科技委信息学部 2017年度工作会议

实验室暑期年会

风采

2016-2017年度实验室各研究方向成果展示

华中科技大学“服务计算技术与系统教育部重点实验室”暨“集群与网格计算湖北省重点实验室”在过去的一年里风雨兼程，锐意进取，在老师和同学们的共同努力下，取得了一个个令人欣喜的成果！在这个秋高气爽，丹桂飘香的收获季节，本期季刊将对实验室过去一年获得的成果进行回顾与总结。

在云计算与移动计算方面，针对目前网络功能虚拟化环境下由于缺乏合理的网络流调度进而导致整个系统的性能降低以及现有的静态调度方式无法适应动态变化的网络环境等问题，建立了基于动态队列的网络模型并使用李雅普诺夫分析框架提出了一种动态的网络流调度算法。相比静态的网络流调度，它能实时地保证网络处于稳定状态，并且使网络平均性能达到最优。此外，现有的NFV系统中的网络功能大都运行在CPU上，并不一定能支持高吞吐率，从而导致CPU性能成为影响网络流吞吐率的瓶颈。针对此问题设计了一种基于GPU的网络流处理加速系统，该网络流处理系统的调度模块基于上述提出的调度算法，能够根据当前队列负载情况，选择最优队列进行服务，在兼顾网络服务公平性的基础上达到提高网络吞吐率的目的。

在系统软件与体系结构方面，围绕利用多核/众核处理器（GPU、FPGA）以及存储器件（MRAM、PCRAM）等新型硬件为上层新型应用提供有效的支撑平台这一主旨取得以下成果：1）研发了基于NUMA架构的轻量级异构内存模拟器HME，解决了现有NVM模拟器模拟速度较慢、不支持网络通信模拟以及写延迟模拟不精确的问题；2）研发了基于细粒度页面迁移以及动态重映射的异构内存大页管理系统Rainbow，解决了现有大页管理机制难以兼顾异构内存介质特性的问题；3）针对图计算性能受限于底层硬件架构的问题，提出了一种基于数据流模型的方法以优化CPU核内性能，基于上述方法分析了设计数据流图计算加速器的挑战并提出了初步设计方案。4）研发了基于视频流的、支持资源细粒度调度的云游戏系统ShareRender，解决了虚拟环境下云游戏系统GPU资源利用率低下的问题。

在网络空间安全方面，围绕当前热门的IT技术引入的安全挑战，在以下五个方面取得以下成果：1）在云网络安全方面，针对云主机既可能受到攻击又可能被攻击者利用发起攻击的问题，开发了结合软件定义网络技术和分布式处理技术的云主机全面保护系统；2）在大数据安全方面，设计了带条件的基于身份的广播代理重加密技术解决了传统的邮件加密技术无法兼顾安全性和用户使用时的友好性的问题；3）在数据存储安全方面，实现了高密度闪存系统的可靠性优化技术，解决由于性能优化带来的可靠性问题，同时保证了闪存系统的优化机制的高效性；4）在区块链安全方面，提出了POV机制激励真正为社会贡献价值的群体，以激发更多的价值创造，解决了目前区块链主流共识机制POW或POS没有激励价值创造者的问题；5）在Web安全方面，提出了可应用于浏览器的基于标签的细粒度信息流控制模型，高效灵活地保障用户数据的隐私性和应用程序的完整性，解决了当前的浏览器无法保障用户数据的隐私性和应用程序的完整性的问题。

在大数据方面，通过对不同领域大数据问题进行深入研究，从算法、系统等多个维度开展研究，研发了一系列的典型系统、探讨了多个典型大数据领域数据处理问题解决思路：1）针对当前大规模图随机游走中心度顺序计算复杂度高问题，设计了首个近似最优的线性时间分布式算法；2）针对当前动态图中k-core维护问题，提出了一种可以在多核系统中并行处理的高效算法；3）研发了基于负载感知的混合写缓冲系统SSDUP，解决了当前突发性IO写缓冲方案不能合理利用SSD资源的问题；4）设计出MURS调度器，解决了分布式处理系统中大量的GC操作严重影响系统执行效率的问题；5）研发了分布式数据区分处理系统DStream和动态流数据集合管理工具—动态Cuckoo滤波器，解决了数据所具有的实时性强、倾斜性高、动态变化大等特征，给分布式处理系统带来了处理时空开销高、吞吐率受限、难以扩展等严重问题。

上一年实验室硕果累累，在品尝收获的喜悦时，我们也要恪守本心，既要仰望星空也要脚踏实地。在新学年，实验室注入了新鲜的血液，实验室在注重学术创新的同时也兼顾学生培养，对学生严格要求，制定合理的培养计划，提供优良的科研条件，为同学们的科研之路保驾护航。实验室不仅仅是同学们知识的升华地，同时也是一个由老师和同学组成的大家庭，实验室暑期年会、春节联欢晚会等活动让老师与同学们在学校也能够体会到家的感觉。最后，在新的学年里，衷心祝愿实验室的所有老师和同学都能够大步向前，在科研之路上收获更多的成果！

代炜琦

二〇一七年九月

主　编：金　海
本期执行主编：代炜琦
编　委：（按姓氏拼音排序）陈汉华、代炜琦、丁晓锋、耿　聪、顾　琳、胡　侃、华强胜、蒋文斌、廖小飞、刘方明、刘海坤、刘英书、陆　枫、黄　宏、吕新桥、马晓静、羌卫中、邵志远、石宣化、王多强、吴　松、肖　江、谢　夏、徐　鹏、余　辰、于东晓、袁平鹏、章　勤、张　宇、赵　峰、郑　龙、郑　然、邹德清
责任编辑：吴　未

地　址：武汉市华中科技大学东五楼二楼
邮　编：430074
电　话：（027）87541924 或 87543529
传　真：（027）87557354
E-mail：wwuhust@hust.edu.cn
Homepage：http://grid.hust.edu.cn
（此刊仅供内部交流学习）

卷首语
…… 1

热点
…… 3

封面人物
厚积薄发，循序渐进 …… 牛　超　10

专栏
系统软件与体系结构组典型成果介绍
…… 王孝远，姚鹏程，张　伟，刘海坤，廖小飞　12
云计算与移动计算组典型成果介绍
…… 陶　晟，顾　琳　23
网络空间安全组典型成果介绍
…… 袁　斌，徐　鹏，杜亚娟，代炜琦，羌卫中　31
大数据组典型成果介绍
…… 华强胜，于东晓，张　凡，廖良翌，刘　伟，石宣化　40

声音
Js 渲染引擎比较 HtmlUnit/Selenium/PhantomJs …… 王　摧　50
浅谈中断虚拟化 …… 刘本熙　52

动态
教育部科技委信息学部 2017 年度工作会议在武汉召开 …… 吴　未　55
实验室 2017 年暑期年会召开 …… 吴　未　56

推荐
What Makes a Link Successful on Wikipedia? …… 吴　尧　推荐　57
Version Traveler: Fast and Memory-Efficient Version Switching in Graph Processing Systems …… 武斯杰　推荐　59

学术报告
Developing the Graph-based Methods for Optimizing Job Scheduling on Multicore Computers …… 何黎刚　61
Memory Centric Optimization: Keep the memory hierarchy but nothing else …… 孙贤和　61
Optimizing MapReduce Framework through Joint Scheduling of Overlapping Phases …… 吴　杰　62

交流
感　恩 …… 杜亚娟　63

首个以太坊虚拟机（EVM）反编译软件Porosity正式发布

（代春凯 整理）

以太坊虚拟机（EVM）如今似乎迎来了第一个反编译软件，设计用来将智能合约恢复为源代码。

2017年7月27日，网络安全创业公司Comae Technologies在美国拉斯维加斯举办的DefCon黑客大会上正式公布了这个反编译软件。这个开源EVM反编译软件设计用来更加容易地识别以太坊智能合约中的bug。

目前一连串的以太坊黑客攻击事件暴露出编写安全智能合约代码的困难。这个叫做Porosity的反编译软件承诺能够让开发者恢复难以理解的EVM字节码到初始状态。

Porosity开发者和Comae创始人Matt Suiche告诉CoinDesk说：

“我编写反编译软件最初想要解决的问题就是希望能够获得真实的源代码，并且无需通过逆向工程就能获得真正的源代码。”

同时，Porosity现在还整合了摩根大通的开源企业级Quorum区块链，目前已经可以从该银行的GitHub中获取。

经过在摩根大通自己工程师的帮助下进行测试，Porosity和Quorum预计将共同帮助运行实时智能合约安全检查。这次结合直接整合了GO语言以太坊Geth，结合了私有网络安全性和修补流程与正式的治理模式。

摩根大通区块链主管Amber Baldet向Coindesk描述了她认为的这种技术的重要性，她说：

“Porosity是第一种从EVM字节码中生成可读的Solidity语法的智能合约反编译软件。”

Porosity的问世使得区块链上智能合约的漏洞能够通过反汇编发现，但是这也使得区块链上的智能合约都能够通过反汇编来得到智能合约源代码，这样就可能侵犯了智能合约编写者的权益，进而导致智能合约的版权问题。

全球顶级安全会议首次迎来中国公司

（贺双洪 整理）

在8月16日——18日召开的USENIX Security会议上，360冰刃实验室将会带来《Digtool：A Virtualization-Based Framework for Detecting Kernel Vulnerabilities》的分享，介绍四种类型的Windows漏洞自动挖掘技术。

USENIX Security是信息安全领域“四大”顶级学术会议（此外还包括S&P，CCS，NDSS）之一，始于上世纪90年代初。从2014年复旦大学接收第一篇开始，截至2016年，大陆地区共计5篇被录用，均为高校的联合研究成果，但尚未有中国公司的独立研究成果被发表。今年，冰刃实验室在USENIX Security的投稿被成功收录，意味着360成为了独立研究并以第一作者身份发表研究成果的首家中国公司。

Digtool是360冰刃实验室自主开发的一款Windows漏洞自动化挖掘系统，主要利用基于硬件的路径探测方法和基于硬件虚拟化的错误检测机制进行检测，捕获程序执行过程中触发的漏洞。“Digtool自动化挖掘系统成果显著，具有学术和工业双重价值。” 360冰刃实验室掌门人潘剑锋表示。

Digtool是第一款利用硬件虚拟化技术的实用化自动漏洞挖掘系统，也是一款“黑盒”漏洞挖掘系统，不需要基于源码即可完成漏洞挖掘。更惊艳的一点是，Digtool可以实现自动化、批量化工作，“可能只需要跑一局游戏，十几个漏洞就挖到了。”

Digtool的工作流程就像挖沙淘金一样：首

先，Digtool可以记录内存访问日志，这就实现了第一步挖沙的过程；进而，Digtool的分析模块会进行分析，一旦符合主要的六种漏洞行为特征规则，便实现了一次“淘金”，也就意味着找到一个漏洞。利用Digtool，冰刃实验室在短期的初步功能验证实验中就已经挖掘到20个微软内核漏洞，以及41个来自杀毒厂商的驱动漏洞。

可以说，Digtool的出现让Windows内核漏洞挖掘有了质的飞跃。以往的人工挖掘耗时耗力，安全研究员需要一行一行分析代码，Digtool系统大大提高了漏洞挖掘的自动化程度，改变了漏洞挖掘的运作模式，同时也提高了速度和精准度，能够在第一现场发现漏洞。

阿里云城市大脑

（杨婉璐 整理）

城市大脑于去年杭州云栖大会公开亮相，采用阿里云的 ET 人工智能技术，对整个城市进行全局实时分析，自动调配公共资源，修正城市运行中的 Bug 。目的既是在杭州起到“眼、脑、心、手”的作用，而让这些器官良好运作的“血液”，则是数据。以交通为例，数以百亿计的城市交通管理数据、公共服务数据、运营商数据、互联网数据被集中输入杭州城市大脑。这些数据首先成为这个大脑智慧的起源。拥有数据资源后，城市大脑还需要五大系统才能高效运转——超大规模计算平台、数据采集系统、数据交换中心、开放算法平台、数据应用平台。

7 月 20 日，据阿里云方面消息，城市大脑三项技术论文同时入选国际顶级学术会议——第 25 届国际多媒体会议 ACM Multimedia（简称 ACM Mm ），论文研究成果将用于智能判断交通事故、人流轨迹，同时将解决研究过程中交通数据样本不足等问题。三篇论文的同时入选，证明了将城市作创新平台的城市大脑，已经成为智能研究的第一平台，从学术角度佐证了城市大脑的智能研究价值已获国际顶级会议的肯定。

三篇论文通俗来说，城市大脑能看懂交通事故；实现不看脸认人，看体态就能实现跨摄像头的搜索；它可以自己生成数据，自己训练自己。

1 、城市大脑能看懂交通事故

该团队设计了一种时空自编码器来进行视频异常检测。其核心模型是 3D 自编码器，通过 3D 卷积神经网络对正常视频片段进行特征提取和数据建模。同时，针对交通监控视频的特点，在自编码器的解码部分设计了一个新的预测分支，通过对下一段视频进行预测，来增强网络对视频中物体运动趋势的建模能力。论文结果表明，通过在真实的交通场景的视频片段上对算法进行了评测，在 AUC 和 EER 两个专业指标上都超过了目前最好方法。

2 、不用看脸就能识别出你

由于不同摄像头下的行人图片在光照、姿态、遮挡、视角等情况下均存在十分大的差异，传统的一些基于 hand-carfted 特征的方法都不够鲁棒。近几年，深度学习技术被广泛的应用于计算机视觉领域，并在 person re-id 问题上也有重要突破。深度神经网络学习到的特征相对于传统特征更加抽象也更加鲁棒，在分类、检测、检索等应用场景下有极大的优势。该文的出发点是融合了已有的两类框架的优势，同时弥补了其劣势。在此基础上，将相似度限制拓展到了不同的层次上，并根据不同层次 feature 的特点自适应的设计不同的相似度度量

和损失函数，使得不同层次的特征都可以进行有效的学习。

3 、自己训练自己

车牌识别是智能交通监控系统中的一个很基础的任务，不同于一般的停车场、门禁等场所，监控视频中的车牌图片往往分辨率很低，图片噪声较高，需要人工标注大量的样本进行训练。而 iDST 团队通过基于风格化对抗自编码器的图像生成算法可以将车牌识别准确率提高至 91% 。

苹果推出全球最大AR平台ARKit

（张晓冬 整理）

在6月5日的苹果WWDC 2017全球开发者大会上，苹果发布了AR开发平台ARKit。凭借苹果庞大的iOS用户群，ARKit一举成为最大的AR开发平台。

从功能上来看，苹果ARKit所展示的功能与谷歌早前推出的Tango很相似，他们都可以通过摄像头对环境进行扫描识别，结合SLAM等计算机视觉技术，将虚拟的物体融合到真实的世界里。但这不是仅仅的图像叠加，除了当你移动手机时，虚拟物体在真实世界里的位置保持不动外，苹果在物体的一些细节上也做得很棒。比如现场演示的桌面上放置了一盏台灯和一杯咖啡，当演示者将咖啡靠近台灯时，其反射的灯光明亮也会随之改变，可谓栩栩如生。

发布会上苹果使用的是一台单目摄像头的iPad，这表明不依赖于硬件的升级，ARKit可以支持大部分的苹果设备。那我们再来看看，ARKit 给我们带来了什么样的新功能。

第一，快速稳定的运动定位。这是最基本的AR功能，从演示中可以发现整个定位非常稳定准确，这说明精度很不错，而桌面的特征并不算丰富，表明鲁棒性很好。最后其渲染的Demo模型很复杂，但是感觉流畅，说明实时性和算法能耗都进行过深度优化。从发布会的整个演示来看，ARkit的水准应该是业内顶级的。

第二，平面和边界的估计。平面估计在单目SLAM上并不稀奇，初始化的方式看起来更像是基于IMU的。边界估计在之前应用中不常见，但从演示中虚拟小人掉到桌子外，我们可以看出ARKit可能不是像snapchat那样简单的VIO类算法，点云构建的部分是有一定输出的。

第三，尺度估计。做单目SLAM研究的都有了解，单目是无法解决尺度问题的，虽然视频里没有展示，但是如果真的解决了尺度问题，说明苹果在IMU和视觉融合方面做了非常先进的工作，而且工程化得非常好。

第四，对各个开发平台或引擎的支持。这点说明苹果做AR绝对是“蓄谋已久”的，并且野心很大，不给其他对手留余地，一上来就要建立一个完整且广泛的AR内容开发生态。

总体来说，ARKit基本实现了单目+IMU的SLAM算法可以提供的大部分功能，并且质量很高。毫无疑问，凭借苹果广泛的硬件覆盖率，iOS会成为最有活力的AR内容发布平台，《Pokémon Go》的体验肯定有质的飞跃，一大波真AR游戏应用将会到来。

阿里的分布式机器学习平台——鲲鹏

（叶阁焰 整理）

近年来，随着“大”数据及“大”模型的出现，学术界和工业界对分布式机器学习算法的研究引起了广泛关注。针对这一刚需，阿里集团和蚂蚁金服设计了自己的分布式平台——鲲鹏。鲲鹏取名自《庄子·逍遥游》，文中记载“北冥有鱼，其名曰鲲。鲲之大，不知其几

千里也；化而为鸟，其名为鹏。鹏之背，不知其几千里也。怒而飞，其翼若垂天之云。” 研究人员表示，在他们的鲲鹏系统中，“鲲”即是超大规模分布式计算系统，拥有超强的计算能力；而“鹏”即是超大规模分布式优化算法，建立在“鲲”之上。“鲲鹏”即同时拥有超大规模分布式计算系统及超大规模分布式优化算法，合二为一使得它有“一飞冲天”的能力。

鲲鹏结合了分布式系统及并行优化算法，解决了大规模机器学习算法带来的一系列问题，不仅囊括了数据/模型并行、负载平衡、模型同步、稀疏表示、工业容错等特性，而且还提供了封闭好的、宜于调用的 API供普通的机器学习者开发分布式算法，降低使用成本并提升效率。

超大规模分布式学习系统能力对大数据的公司而言尤为重要。鲲鹏在阿里和蚂蚁众多实际场景中发挥出了巨大的优势。例如，在 2015 年“双11”中，鲲鹏系统上实现的“楼层”排序（LR算法）使得 UV CTR提升了21%，GMV 提升了10%。再如，基于鲲鹏实现的 GBDT+DNN 算法，应用在支付宝交易风险评估业务中，上线以来在相同覆盖度的情况下，案件召回率从91%增加到98%，每天减少了几千万次用户的打扰。此外，在鲲鹏上实现的 Deep Structured Semantic Model（DSSM），已经广泛被应用于搜索、广告和智能客服等业务中。总体来说，鲲鹏系统上的 10多个成熟算法已经被广泛应用于 120多个产品中，这些无一不是阿里生态体系内最大规模的算法。

让好奇心驱动人工智能：UC Berkeley提出自监督预测算法

（张杨松 整理）

无监督学习一直被认为是让人工智能在真实世界中有效工作的研究方向，此前大多数研究都会在训练时为人工智能加入奖励机制以明确目标。UC Berkeley 最近发表的论文提出了一种更为先进的方式，研究人员称这种「好奇心驱动」的人工智能算法不需要奖励机制就能学会如何进行《超级马里奥兄弟》和《Doom》两种游戏，并能达到超越以前方法的表现。该论文已被将于8月6日召开的 ICML 2017 大会接收。

在很多现实世界场景中，对于外部人工智能代理（agent）完成任务的奖励是稀疏的，有时甚至完全不存在。在这种情况下，好奇心可以成为人工智能算法内在的回报信号，让代理可以探索环境并学习可能「终身受用」的技能。在该项研究中，作者们将好奇心构型，作为人工智能代理在动态环境中自我监督学习时预测自身行动结果造成的错误。他们的方法适用于图像这样的高维连续状态空间，跳过直接预测图像困难的问题，也大量忽略了不能影响代理的环境情况。他们提出的方法在两种环境中进行了评估：VizDoom（一种基于视频游戏《Doom》的人工智能研究平台）和《超级马里奥兄弟》。评估主要研究三个方面：1. 稀疏的外部奖励，这样好奇心将在达到目标的过程中占据重要位置；2. 没有外部奖励的探索，在这种情况下好奇心会推动代理进行更有效的探索；3. 全新的环境（如一个游戏的新关卡），在这种情况下此前获得的经验可以帮助人工智能快速上手。

谷歌DeepMind发布最新研究：人类认知心理学开启人工智能的“黑盒子”

（李 羿 整理）

深度神经网络已经掌握了解决各种问题的方法——从识别、推理图像中的物体，到成为

“围棋上帝”。随着这些任务变得越来越复杂，神经网络摸索出来的解决方法也变得越来越繁琐。

因为这个系统太复杂了，即使是设计该系统的工程师可能也无法分析出它发出某一指令的原因。当然，你也不能强求这个神经网络能够给出它下达每一个指令的原因：目前为止还没有一套能够让AI自己解释自己行为的系统。

实际上，这就是人工智能领域著名的“黑盒子”问题，随着神经网络在现实世界中的应用越来越广泛，对这一问题的研究也开始变得无比重要。对此，《麻省理工科技评论》曾以“人工智能核心地带的黑暗秘密”为题刊发表专题文章，来深入探讨神经网络的不可解释性问题。

作为人工智能研究领域的先锋，DeepMind对“黑盒子”问题也在持续关注——目前，该团队正着手开发更多的工具，用于解释人工智能系统。6月26日，在最新发布的一篇论文中，DeepMind提出了一种基于认知心理学来研究深度神经网络的新方法。

那么，什么是认知心理学?一般而言，该学科通过评估行为来推测认知机制，并涵盖了大量与认知机制相关的细节，同时还设计了很多实验来证明这些机制。随着神经网络在解决某些具体问题上的能力已经达到或超越人类水平，认知心理学的研究方法将与人工智能的黑盒子问题愈发相关。

为了证明这一点，DeepMind设计了一个实验来解释人类认知，从而帮助人类进一步了解深度神经网络是如何解决图像分类问题。

实验结果显示，认知心理学家观察到的人类行为，在深度网络中也有类似的体现。总体上说，实验的成功也证明，认知心理学完全可以用来帮助人们更好地理解深度学习系统。

在DeepMind的案例研究中，甚至考虑到了孩子们是如何辨识物体的，这是认知心理学的一大研究领域。

通常而言，孩子们从单个例子中猜测词语意思的能力，被称为“单次语义学习”(one-shot word learning)——这种认知能力看起来非常自然，会让人们觉得，这一过程其实没什么太复杂的机制。

然而，美国著名哲学家威拉德·奥曼·奎因多年前设计的一个经典思想实验，却向人们展示了这一过程到底有多复杂：一个语言学家要去一个地方，但那里的语言和这个语言学家所使用的完全不一样。于是，这位语言学家想找一位当地人来学习一些本地语言中的词汇。这时，正巧有一只兔子从他们身边跑过，当地人脱口而出“gavagai”，于是语言学家就开始推测这个词的意思。

当然，这个词可能表示很多意思，可以是“兔子”、“白色的东西”，甚至指兔子身上的某一部位。

那么，面对这么多的可能性，人类如何去选择哪个正确的意思?

时至今日，我们在使用深度神经网络进行单次语义学习时也会遇到了同样的问题，而DeepMind团队所开发的名为“匹配网络”(Matching Network)的新型神经网络模型或许可以解决这些问题。

实际上，匹配网络凭借着在关注度和记忆方面的进步，完全可以做到仅凭一个案例就对ImageNet图像识别数据库中的海量数据进行分类筛选，而且表现绝对是无可争辩的。

为了阐明这一点，研究团队参照了认知心理学家的研究结果——他们发现，孩子们往往是通过归纳出偏好来消除许多错误推论，从而

得出正确推论。这些典型的偏好包括：整体偏好(whole object bias)：孩子们会更倾向于认为一个词指代的是整个物体，而不是它的组成部分(兔子未被关注的部分)；类别偏好(taxonomic bias)：将一类物体归于类别中的某一种物体(所有动物都可能被当做“兔子”)；形状偏好(shape bias)：孩子们往往会根据事物的形状，而不是颜色或条纹来描述一个物体(所有白色的东西都可能被归类为“兔子”)。

在上面三种偏好中，DeepMind的研究团队选择了“形状偏好”作为检测神经网络的切入点，这是因为“形状偏好”在人类偏好研究中占据了很大的比重。

更为关键的是，在接下来的试验中，DeepMind研究团队向深度神经网络展示三个物体的图像——原物体、形状匹配物体(形状相似但颜色不同的参照物体)以及颜色匹配物体(颜色相似但形状不同的参照物体)。同时，研究团队记录了原物体和形状匹配物体被归为一类的次数，以及原物体和颜色匹配物体被归到一类的次数。

在对上述归类次数进行比对后，研究人员发现，正如人类一样，该网络对形状的感知有着超过对颜色和材质的偏好——也就是说，神经网络也有着“形状偏好”。

DeepMind在匹配网络中进行的认知心理学实验。A为形状匹配物体，B为颜色匹配物体。由于匹配网络对形状的偏好，会倾向于将原物体于A相匹配。

然而，除了观察到深度网络的形状偏好外，还有一些值得注意的结果：首先，形状偏好在对网络训练的早期就开始出现。这很容易令人联想到了人类思维方式对于形状的偏好。心理学家表示，儿童所展示的形状偏好比青少年要少，而成年人的形状偏好最为明显；其次，取决于不同的组合和训练模式，深度网络对形状偏好的程度也不一样。所以当研究团队对深度学习系统进行测试时，必须使用大量的训练模型来得到可靠的结果，就像心理学家对不同对象的测试一样；最后，即使形状偏好程度不同，每个网络在同种单次语义学习测试上的表现是一样的。这说明不同的网络可以针对同一个复杂问题找到同样有效的解决办法。

总之，DeepMind团队认为，此次实验的重大成功就在于证明了认知心理学在解释和推测神经网络行为方面的潜力。由于人类在认知心理学研究领域具备丰富的知识储备，这将为解决人工智能“黑盒子”问题提供全新的思路，让人类更深刻的理解深度神经网络的行为。

Android Oreo 问世：谷歌正式发布Android 8.0

（吕现伟 整理）

历经一年多的开发和数月以来开发者与早期用户的反复测试，Android 8.0 Oreo于2017年8月22日正式面向全球发布。Android 8.0为用户带来了诸如画中画 (Picture in picture)、自动填充 (Autofill)、免安装应用 (Instant Apps)、Google Play保护机制 (Google Play Protect)以及更快的启动时间等诸多新功能。

Android 8.0 Oreo的宗旨是提供流畅的体验，让Android更加强大且易用。**画中画(Picture-in-picture)** 功能让用户能够以任意窗口大小同时进行两项任务的操作，该功能的App适配也很简单。**通知标志(Notification dots)** 使得通知的概念得以延伸，为应用的互动提供更多可能。对大多数应用来说，这一功能会自动实现，Android甚至会根据应用图标的颜色自动选择通知标识的颜色。**自动填充框架(Autofill framework)** 简化了用

户设置一台新设备以及同步密码的过程。需要用到表格数据的应用可通过自动填充框架进行优化，密码管理应用通过新的API接口，能够让用户在自己最喜欢的应用中使用密码自动填充服务。自动填充功能将会在接下来的几周中，随同Google Play 的服务进行升级。

Android 8.0 Oreo通过给予开发者更具可视化的应用数据，来帮助优化电池寿命，缩短启动时间，提高图形渲染效率和稳定性。**系统优化：**通过系统底层优化使得应用运行更迅速且流畅。试举一例：应用运行时，通过新的并发压缩垃圾回收机制 (concurrent compacting garbage collection)，代码局域化 (code locality)以及其他一些方式优化其体验。**后台限制：**对在后台拿取位置和Wi-Fi扫描增加了限制，并改变了应用在后台的运行模式。这些限制能够避免被动的电量和内存流失，而且对所有应用都适用。**备有多种功能的Android Vitals信息中心以及集成开发环境分析器（IDE profilers）：**在Play 控制台中，现在看到应用的数据汇总，这些数据能够帮助开发者发现一些常见问题。数据包括超常的应用崩溃率，应用程序无响应率，框架卡死时间，慢速渲染耗时，过度唤醒等等。开发者也能够在Android Studio 3.0中找到新的性能分析器，以及新的仪表布局。

Android 8.0 Oreo同时也为开发者准备了许多新功能，助其提升效率，开发出更好的App。**自动调整 TextView 的文字大小：**Android Oreo 8.0允许开发者自定义TextView实现文本内容自动调整字体大小以适应TextView大小，各种文本长度均适用。开发者须预设一系列文本大小，或者设定出最大最小值（两值之间须设固定数字间隔），这样文本就能根据TextView 的大小自动展开或收缩。**XML中的字体资源：**Android 8.0中已充分支持Fonts字体资源类型，允许开发者直接在XML布局中访问字体并定义字体系列。**可下载字体和表情：**Android 8.0允许开发者从共有供应商获取可下载字体资源，而无需将字体绑定到 APK中。供应商和Android支持库负责下载字体，并将这些字体分享到各个App中。同样的操作也可用于获取表情资源，让开发不再止步于设备内置表情包。**自适应图标：**这项功能帮助开发者更好地使用设备用户界面，创建自适应图标，根据设备厂商选定的不同模板，系统显示图标形状会有所改变。系统还添加了与图标的交互动画，置于启动器、快捷方式、设置、共享对话框和概览屏幕当中。**快捷方式锁定：**App快捷方式和桌面小部件是吸引用户的利器。从Android 8.0 Oreo开始，用户能够在应用内添加和锁定快捷方式至启动器。与此同时，新系统还支持添加特定活动，帮助用户创建快捷方式，该活动需要通过自定义选项并经过用户同意才可完成。**支持更广域应用的颜色：**Android图像应用可以充分利用新设备带来的广域色彩支持，显示宽色域图像。应用程序将需要在其清单文件（每个活动）启用一个标志，加载启用宽位图嵌入颜色配置文件（如AdobeRGB，Pro Photo RGB，DCI-P3等）。**WebView网页视图：**Android 8.0 Oreo默认开启WebView多进程模式，新增API控制应用处理错误和崩溃，增强安全性和提升应用稳定性。为了进一步提高安全性，开发者可以选择通过Google Safe Browsing来对自己应用内WebView的URL来进行安全检查。**Java 8语言APIs和runtime的优化：**Android现在支持几种新的Java语言 API，包括新的java.time API。另外Android Runtime相比以前更快，在某些基准测试程序中可以提升多达2倍。

厚积薄发，循序渐进

前言

首先，很荣幸成为实验室期刊的封面人物，在研究生期间这就一直是我的心愿。转眼之间，距离我离开实验室并踏上社会工作岗位已经一年了。在这一年的工作中，我越发体会到研究生期间实验室对我培养的重要性。借此机会，和实验室同学们分享交流一下我在科研过程中的一些心得，也希望这些经验能对实验室在读的同学们有些许帮助。

厚积薄发

在研一开始的时候，相信大部分同学和当初的我一样对将要到来的2-3年科研时光感到迷茫。由于研一开始阶段的课程压力较大，我并未感受到和本科学习期间的差异。直到正式进入实验室打卡阶段后，才慢慢了解接触到研究生阶段的科研任务和本科生阶段的学习任务的差距。对这种转变的适应是非常关键的，越早进入状态对自己工作的开展越有帮助。

在研一期间，相信大部分同学都在尝试探索自己未来的论文命题，同时通过阅读大量的参考文献为相关领域的知识打下基础。个人认为，这是在科研阶段中极其重要的一个环节。尤其是任务课题的挑选，将直接影响到今后研究过程中的工作。只有在准备阶段有一定量的积累，才有可能在论文的研究过程中做到得心应手。

在阅读论文的开始阶段，我觉得应该做到有目标的“涉猎”。在明确自己的研究课题之前，其实并没有必要太过深入的研究专一方向。在阅读过吴松老师推荐的“移动云”领域的几篇优秀论文后，我也研读了论文中提到的一些高质量的参考文献。正是对大领域中研究现状的充分了解，才帮助我确定了该领域现有的研究还未覆盖的部分，同时确立了自己的研究课题——“基于容器的移动云平台”。同时，文献阅读阶段中更为关键的一点是做好笔记，尤其是对概读的论文，如果当时没有记下研究的重点，将来回过头看这篇论文的时候又要花时间找到关键部分，这样就大大降低了科研效率。这里也给大家推荐我用来记录论文笔记的工具Mendeley，希望可以帮助各位在读的同学。

相信导师

可以说，我的论文投稿能被接收，最需要感谢的就是我的指导老师——吴松老师。在论文研究的整个过程中，吴松老师对我的帮助无时不在。在论文选题中，我一直都没能明确自己的选题目标，一直到开题时都对研究方向只是一个非常笼统的总结。在此期间，吴松老师多次在和我的交流中提出了他的研究想法，这些想法都很有意思，有一些天马行空，正是这些idea的碰撞帮助我完成了论文的选题。

我研究方向中的容器正是和吴老师早期交流过程中提到的概念，当时我是第一次听到这个概念，在大概了解了这个概念后又将其搁置了很长一段时间。直到后来随着Docker等容器解决方案的大火，一次偶然的机会我和吴老师谈到容器，一个将容器概念和移动云概念结合在一起的想法出现在我的脑海中。在对这个想法进行了充分的背景研究后，果断地将其选为自己研究生阶段的研究方向。

在论文撰写阶段，由于我的英文写作能力不强，表达过程中很难抓到重点，尤其是在论文的组织结构上，总是头重脚轻。再加上作为一篇系统方向的论文，工程部分占比很大，这让我在论文中很难抓住工作的学术价值。在此

过程中，吴老师先后多次亲力亲为，修改论文架构以及语言表达。同时，在系统设计章节，吴老师帮我挖掘出了工作的难点和学术意义，并指导我完成了论文的撰写。

有很多同学或出于对自己能力的自信，或出于对老师的疏远，不愿意去和导师进行沟通交流。我认为这在研究生阶段是非常致命的错误，可以说，实验室每一篇高质量学术论文的产出背后，都必然会有指导老师的关键性努力。所以，希望实验室在读的同学们一定要和老师多沟通交流，至少要做到每周一次的工作总结以及征求指导老师的想法建议。实验室老师都是经验丰富的研究学者，他们对研究生的帮助指导要比学生自己盲目的试错有价值得多。

兴趣使然，循序渐进

我的课题研究能有成果的另一个重要原因，是我个人对自己论文中系统的开发探索十分有兴趣。在当初明确研究课题后，我就对自己论文中的系统有了一个大概的了解，这将是一个涉及到操作系统内核、Server端调度和Client端请求的复杂系统。尽管系统开发在开始阶段摸不到边，但是正是这样一个充满挑战的系统激发了我的斗志。作为一个系统开发的狂热爱好者，论文研究中的这一系统正合我的胃口，这也是我能够花研二将近一年的时间将这个包含三大部分的系统开发完成的前提。设想，如果当时我的选题是我个人不太感兴趣的更偏向算法类型的课题，那我很有可能缺乏完成实验设计的动力，最后产出的学术成果质量也必然不尽如人意。

所以，我希望实验室在读的研究生同学能尽可能找到自己感兴趣的研究课题。众所周知，如果做一件事是你的兴趣，那你将不再是工作而是享受。实验室同学们可以让指导老师指导自己感兴趣的研究点，这样可以让导师帮助你一起寻找你希望在研究生阶段几年内每天面对的工作。

在真正开展研究工作时，不要急于求成，切忌心浮气躁。在面对大量工作要做时，首先要做到不慌不忙，制定好自己的科研规划。如果在研一期间就明确了自己的研究方向，研二一整年的时间都是用来完成自己的研究，时间还是相对比较充裕的。以我个人的经历为例，论文系统包含三大部分，在实现过程中我就为自己设定了几个里程碑，在某一时间完成第一部分，然后花多久完成第二第三部分，然后按照自己的计划循序渐进，花了一年的时间去实现并完善自己的研究系统，为研三期间开始的论文书写打下了坚实的基础。尽管我的论文一直到研三下学期才第一次投出，但也总算是取得了不错的成果。

尾语

以上就是我对研究生阶段科研过程的心得体会，总而言之，我觉得学术科研之路就像登山，从远处看，高山耸入云端难以企及。但当你真正开始登山时，一步一个脚印，不骄不躁，总有一天你能登到山顶体会一览众山小的美景。我也感谢研究生阶段能够完整的攻克一个科研课题，这段经历对我一年来的工作有莫大的帮助，我能够从零开始地完成工作中一个个的难题，正是因为我把他们当成一次次小小的科研探索，依靠研究生阶段的方法论逐个将其攻克。

值得一提的是，论文第一次投稿SoCC会议被拒，经过戴小海师弟对论文的修订后才得以让研究成果在IPDPS会议上得以展示，在这里感谢小海对我科研的帮助。也希望实验室能够越办越好，各位实验室老师科研成果丰收不断，各位实验室的同学们能够早日论文高中，在今后的学习生活中一帆风顺。

牛 超
2013级硕士研究生
研究方向：移动云，容器技术
Email：chaoniu@amazon.com

系统软件与体系结构组典型成果介绍

王孝远，姚鹏程，张 伟，刘海坤，廖小飞

关键词：内存计算，异构图计算，运行时优化

1. 介绍

随着各种多核/众核处理器（GPU、FPGA）以及存储器件（MRAM、PCRAM）的不断涌现，人们将可为大数据处理等耗费巨大计算、存储开销的应用提供新的处理架构。本组旨在利用这些新型硬件为上层新型应用提供有效的支撑平台。基于所承担的863项目“内存计算系统软件研究与开发”和其他项目的支持下，在上一年度，本组主要在如下三个方面进行了研究：（1）异构内存相关系统，包括轻量级内存计算模拟器、异构内存大页管理系统等；（2）异构图计算系统，包括基于FPGA的图计算加速器等；（3）运行时相关系统，包括数据竞争检测机制、面向云游戏的细粒度资源调度系统等。在这三个方面，本组研发了一系列的典型技术和系统：

1）在异构内存方面，针对现有NVM模拟器模拟速度较慢、不支持网络通信模拟以及写延迟模拟不精确的问题，研发了基于NUMA架构的轻量级异构内存模拟器HME，利用远端节点的DRAM模拟NVM，满足了异构内存环境的需求。实验显示HME模拟错误率低、对运行应用的影响较低、开销小，对于异构内存相关的研究有一定的实际应用价值；针对现有大页管理机制难以兼顾异构内存介质特性的问题，研发了基于细粒度页面迁移以及动态重映射的异构内存大页管理系统Rainbow。该系统提高了系统性能，降低了系统能耗，同时增加了NVM的写寿命，具有一定的实际应用价值。

2）在异构图计算方面，针对图计算性能受限于底层硬件架构的问题，研究了影响图计算性能的核内因素，结合图计算的特性提出了一种基于数据流模型的方法以优化CPU核内性能，基于上述方法分析了设计数据流图计算加速器的挑战并提出了初步设计方案。在主流的数据流系统（Naiad和Tensorflow）上的实验结果表明数据流模型可以有效提升指令级并行度并降低分支预测和取指令开销。

3）在运行时优化方面，针对于虚拟环境下云游戏系统GPU资源利用率低下的问题，开发了基于视频流的、支持资源细粒度调度的云游戏系统ShareRender。通过系统支持绕过GPU虚拟化，利用细粒度的资源调度策略，极大地提升了云游戏系统的资源利用率，并且能减小过载带来的影响，在保证较高并发度的情况下保证了服务质量。该系统在云游戏领域具有一定的实际应用价值。

2. 异构内存计算模拟器

当今以数据为中心的应用程序需要大容量的内存才能有效执行。但是，DRAM 技术的扩展无法满足内存容量增长的要求。新兴的可字节寻址的非易失性存储器 (NVM) 技术是高内存密度、接近零待机功耗和较低的每比特成本的有前途的替代方案，但却有着更高的写入延迟和较低的带宽。大量的研究工作已提出使用NVM结合DRAM的混合内存体系结构。因为商用NVM 硬件尚不可用，这些研究主要依赖于周期精确的体系结构模拟器。但是，当前的模拟方法要么太慢，要么无法模拟复杂和大规模的工作负载，并且并不兼容网络通信、虚拟化和分布式应用。我们提出了一种基于DRAM性能仿真器—HME，

以模拟即将到来的NVM 技术的性能和能耗特性。HME利用商业级CPU中的NUMA架构中可用的功能来模拟两种内存：快速、本地DRAM和其他NUMA节点上的较慢的远程NVM。通过在远程NUMA节点上注入不同的内存访问延迟，HME可以模拟范围广泛的NVM 延迟和带宽。

为了帮助程序员和研究人员评估NVM对应用程序性能的影响，我们还提供了一个高级编程接口，用于从NVM或DRAM中分配内存。由于在商品级硬件中没有编程接口直接控制内存访问延迟，因此我们采用了一种软件方法来模拟NVM读延迟。其基本思想是在合适的时间间隔内为应用程序注入软件生成的额外延迟。我们通过对应用程序的内存访问进行监控，并通过内存访问行为对滞后时间进行建模，以逼近一段时期内的NVM读延迟。对于每个内存写入操作，因为它不在应用程序执行的关键路径上。一般情况下，数据写入可以归类为写入和回写。在前一种方法中，数据立即通过缓存写入主存中。在后一种方法中，数据首先写入缓存，并在以后将相应的缓存行逐出时写入主存。但是，内存写入操作仍占用内存总线带宽，可以推迟内存读取请求。不应忽视这些拖延。我们通过特殊的性能监控硬件来监控写入和写回，并通过写延时建模来构建内存读写次数与写延迟之间的关系来模拟NVM写延迟。

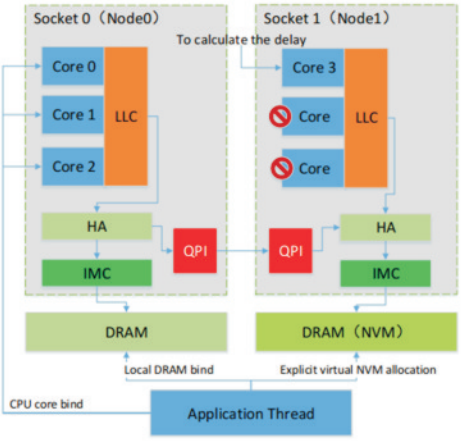


图1 HME在不同的socket上利用NUMA架构来模拟NVM 和 DRAM 混合内存环境

在带宽模拟方面，我们通过限制最大可用DRAM带宽来模拟NVM带宽。带宽节流是通过利用在商品英特尔至强处理器中提供性能计数器来控制DRAM能通过的最大访存指令实现的。我们还建立了能源消耗模型来计算NVM的能耗。由于缺少用于计算主存储器所消耗能量的硬件级特征，我们选择了一种统计方法来对NVM能耗进行建模。不像DRAM，NVM 不产生静态功耗，所以我们只需要计算NVM 读写能耗。我们通过记录一段时间内NVM的读写访存次数，通过建立简单模型来模拟NVM单位时间内的能耗。我们使用NUMA体系结构在双插槽服务器上模拟混合内存系统。每个插槽都包含三级 CPU。NUMA体系结构支持通常使用本地DRAM，同时在远端socket上提供更高的DRAM访问延迟。我们在远程NUMA节点上使用DRAM来模拟更高延迟和低带宽的NVM。如图1所示，我们在远端socket中选择一个core，用来计算应向本地socket上的应用程序注入的附加延迟。我们将关闭远端socket上的其他core以避免对计算核的性能干扰。HME使用英特尔至强处理器Uncore PMU(性能监视单元)中Home Agent(HA)来记录正在本地节点0上运行的应用程序对远程节点1上的DRAM内存读取访问的次数(LLC 未命中)和写访问次数。HME通过核间中断(IPIs)向节点0的核心注入额外的延迟。这样，HME增加了远程内存访问延迟，以模拟NVM延迟。

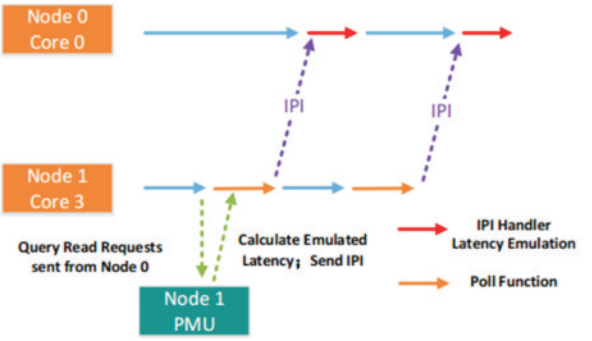


图2 HME延迟模拟过程

HME的延迟注入过程如图2所示。该程序运行在节点0的核心0，而节点1的核心3用来模拟NVM。当在核心0上运行的程序访问远程DRAM (即模拟NVM) 时，核心3调用一个轮询函数。在轮询功能中，核心3周期性地读取节点1上的PMU，并在这个轮询期间获取核心0对远程内存访问的数量。之后，核心3计算模拟滞后时间，并通过IPIs向核心0注入延迟。核心0处理IPIs请求，并导致CPU空转和生成延迟，从而模拟NVM的滞后时间。

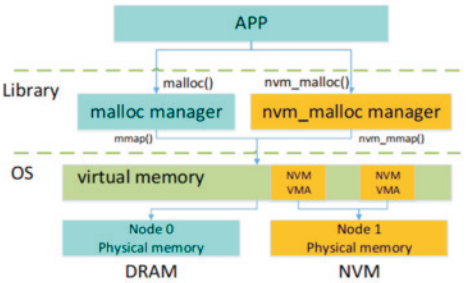


图3 使用AHME实现细粒度内存分配

如图3所示，我们扩展了Glibc库以提供nvm_malloc函数，以便应用程序可以通过malloc和nvm_malloc分配混合内存。为了实现nvm_malloc函数，我们修改Linux内核以提供nvm_mmap分支来实现内存分配。这个分支为nvm_malloc函数分配页面映射。同时,也会将NVM页面和DRAM页面在VMA中的区分开来。AHME在VMA中为nvm_malloc分配的NVM页打上NVM标志，并在page fault时调用指定的NVM page fault函数，以分配NUMA远程节点中的物理地址空间。整个分配流程如图4所示。

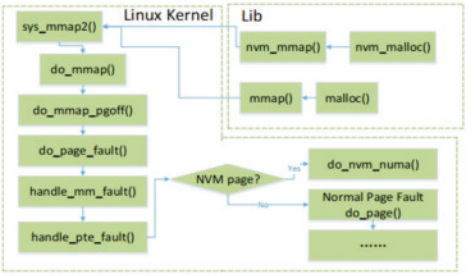


图4 AHME内核内存分配路径

3. 异构内存大页管理系统

计随着大数据应用的内存计算为中心的趋势愈发明显，硬件地址转换开销逐渐成为大内存系统的新的系统瓶颈。在大页系统中，由于一个TLB表项可以映射较大的地址空间，所以TLB命中率较高，从而降低了地址转换的开销。然而大页机制通常不利于轻量级的内存管理，比如页面迁移。所以对于需要利用页面迁移提升系统性能的异构内存系统来说，传统的大页管理机制将不再适用。为此，我们提出了同时支持大页机制和细粒度页面迁移机制的新型大页管理系统Rainbow。Rainbow对两种内存介质采用不同页面粒度进行管理。我们使用大页机制管理内存容量相对较大的NVM，我们使用细粒度、多粒度的管理策略以管理内存容量相对较小的DRAM。相应的，通过采用分离式TLB（split TLB）的架构和聚合机制，Rainbow支持多种粒度的页面。我们设计了一个轻量级的访存监测模块，使用此模块对NVM页面进行多粒度访存监控，以及细粒度的热页筛选，然后在迁移时，把这些连续的热页面聚合成一个相对较小的大页。与此同时，Rainbow使用位图标识大页中小页的迁移情况，通过地址重映射机制在保证迁移前后地址转换的正确性的同时保证了大页的完整性。实验结果显示，Rainbow可以显著的降低地址转换开销，从而提升系统整体性能。

3.1 轻量级热页监测机制

随着容量的增加，以4KB页粒度监测内存访问情况开销越来越大。比如说，如果仅仅是用两个字节记录任一4KB页的访存次数，1TB的内存需要512MB存储空间。因此，使用片上SRAM来存储这些访存信息是不切实际的。而且我们发现，监测所有大页内部小页的访存信息是不必要的。我们的实验表明，一个访存频次较高的小页（例如4KB热页）在一个访存频次较低的冷大页（例如2MB大页）的概率约为1%。基

于此，为了降低存储开销，我们提出了一种二维访存频次监测机制。如图5所示，Rainbow分两个阶段进行访存信息监测，每个阶段分两步。首先，Rainbow以大页的粒度监测NVM的访存信息（①）。我们使用2字节去存储每个大页在一个时间周期（10^8个时钟周期）内的访存信息。对于每一个内存请求，内存控制器负责大页的访存信息的监测与更新。在时间段的后半部分，根据监测结果筛选出最热的N个页面用于细粒度的页面监测（②）。接下来，Rainbow拆分这N个大页，并且记录这些小页的访存信息（③）。最后，Rainbow将每个小页访存信息与阈值相比较，筛选出热的小页（④）。由于Rainbow不需要以小页粒度监测冷页面的访存信息，因此显著地降低了热页监测的开销。

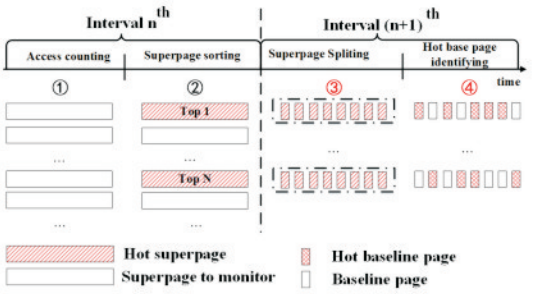


图5 Rainbow二维访存监测机制

3.2 基于效用的迁移策略

尽管迁移NVM页面到DRAM中可以显著提升内存访问性能和能效，但是迁移操作本身会增加访存时延。并且如果DRAM内存压力较高时，不恰当的页面迁移可能会导致DRAM与NVM之间的内存颠簸。所以，我们必须权衡迁移操作的利弊。假设C_r和C_w分别表示一个时间段内（10^8个时钟周期）的读写次数，t_{nr}和t_{nw}分别代表NVM的读写时延，t_{dr}和t_{dw}分别表示DRAM的读写时延，T_{mig}代表页面从NVM迁移到DRAM的时延，$T_{writeback}$代表脏页面从DRAM逐出并写回到NVM的时延。当DRAM使用率较低时，迁移的收益如公式1所示。

$$Benefit_{ming}=(t_{nr}-t_{dr})C_r+(t_{nw}-t_{dw})C_w-T_{mig} \quad (1)$$

随着NVM页面迁移到DRAM中，DRAM的利用率逐渐增加，当DRAM没有空闲空间用来存储新的NVM迁移页面时，Rainbow会从DRAM中筛选合适的页面进行写回NVM，所以，迁移的收益需要再减去写回的开销。假设p1表示DRAM中的需要被写回的页，p2表示新的需要迁移到DRAM中的NVM的热页面，所以收益如公式2所示。

$$\Delta Benefit_{mig}=(t_{nr}-t_{dr})(C_r^{p2}-C_r^{p1})+(t_{nw}-t_{dw})(C_w^{p2}-C_w^{p1})-T_{mig}-T_{writeback} \quad (2)$$

3.3 分离式TLB与地址重映射机制

Rainbow利用分离式的TLB进行NVM和DRAM的虚实地址转换，每次需要访存的时候，各种TLB并行查找，如图6所示。对于地址转换，有四种场景可能会出现：1）多粒度TLB命中且大页TLB命中；2）多粒度TLB命中但大页TLB不命中；3）多粒度TLB不命中但大页TLB命中；4）多粒度TLB和大页TLB均未命中。在第一种和第二种情况下，Rainbow返回多粒度TLB查询结果（如图6中①所示）。第三种情况，Rainbow通过查询迁移位图确定相应页面是否发生了迁移，如果已经发生了迁移，Rainbow读取相应地址页面的前8字节，从而返回迁移的目的地址（如图6中②所示）。相对的，如果页面未发生迁移，则返回大页TLB的查询结果（如图6中③所示）。在最后一种情况下，Rainbow查询页表得到相应的地址（如图6中④所示），当页表查询返回物理地址时，Rainbow仍需查询迁移位图，其余操作和第三种情况相同。正如图6所示，尽管NVM中的热页面被迁移到DRAM中，Rainbow并不需要拆分NVM大页和修改相应的大页TLB表项。这种地址转换机制保证了应用透明的页面迁移机制。此外，Rainbow并不需要为DRAM页面设计额外的页表结构。所有的对迁移页面的访问都可以

通过读取NVM大页获取迁移后的DRAM地址。由于大页的TLB命中率较高，所以Rainbow可以避免绝大多数页表操作。即使在最差情况下（两种TLB都未命中），Rainbow的地址转换的开销也只是和使用页表管理的DRAM开销相当（传统的小页页表有4级，页表操作需要4次访存；大页页表只有3级，最差情况需要3次页表查询加上一次NVM读取迁移地址共4次访存）。

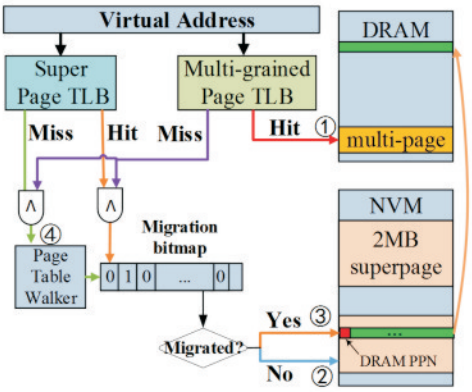


图6 Rainbow 地址映射机制

3.4 多粒度TLB

Rainbow修改了TLB表项，使得多粒度TLB中的每一个表项都支持任意粒度（2MB以内）的页面映射。通过使用TLB表项中的9位预留位，实现了页面大小的标识（Npages），如图7所示。假设一个多粒度TLB表项起始虚拟页号用VSN表示，那么这个表项可以表达（VSN+Npages）个连续的4KB页面的地址映射关系。例如在图7中，其中一个表项VSN是“0x1c9a”、Npages是“0x101”，这表明这个多粒度表项可以映射217个连续的4KB页。接下来讲多粒度TLB的四个关键问题：TLB聚合机制、查询机制、作废机制和替换机制。由于Rainbow把迁移操作推迟到时间段末尾进行，实验表明会有很大概率从产生一组连续的热页面被同时迁移的情况，Rainbow尽可能的把一组连续的页面迁移到一段连续的DRAM地址空间中，从而提高多粒度TLB的使用效率。此外，Rainbow设计了一套迁移判定策略，在每个时间段内的迁移之前，会监测两组热页中间的“距离”（地址间隔），当此间隔小于预设阈值时，Rainbow会迁移整段页面（包括中间的间隔的冷页面），从而进一步提升多粒度TLB的效率。

如图7所示，由于每一个表项都可以表示任意粒度页面，所以我们设计了一个全相联的TLB。每一个TLB表项有两个比较器进行查询操作，如果一个4KB虚拟页号（VPN）需要进行地址转换的时候，硬件逻辑会把VPN与任一一个表项进行比较，假设用VSN_i表示第i个可用表项的起始虚拟页号，用$Npage_i$表示这个表项所能表达的页面个数。如果$VSN_i \leq VPN < (VSN_i + Npages)$，那么多粒度TLB命中，此时返回相应的物理地址。当写回DRAM页面到NVM中的时候，Rainbow 需要作废相应的地址映射的TLB表项。因为多粒度TLB表项表示了一组页面的映射关系，为了降低作废TLB表项对性能的影响，Rainbow在替换页面时总是从一个大页的高端地址处进行替换，从而只需更改TLB表项中的Npages的数值，而不是作废整个TLB表项。当多粒度TLB 满的时候，若此时需要插入新的表项，Rainbow 使用伪LRU算法选出相应的表项进行替换。通常情况下，长时间未被使用的表项会优先替换，但是由于有些表项映射多个页面，所以我们给与多页面的表项较高的权重，尽可能的增加其在TLB中的驻留时间。

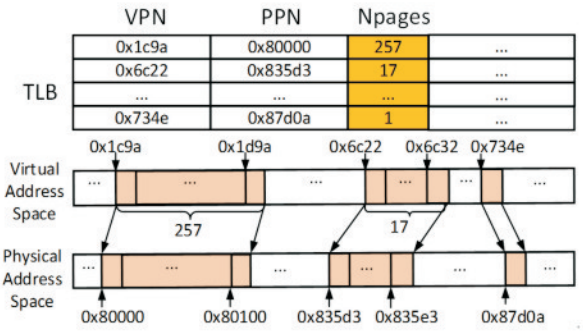


图7 Rainbow多粒度TLB

4. 基于FPGA的图计算加速器

当前的图计算系统通过设计高效的划分和访存策略，极大地提升了内存子系统的性能。然而如表1所示，这些系统往往忽视了底层硬件的限制，导致了图算法在执行时面临着严重的核内性能低效的问题（主要体现在指令级并行低和分支预测开销高两方面）。研究表明，该问题主要来源于图计算对大规模并行的需求和当前针对顺序程序设计的控制流模型之间的不匹配。一方面，控制流模型中遵循的顺序语义极大地限制了图计算的并行度；另一方面，图计算中复杂的控制指令极大地增加了预测的开销。

		Prior Work	Our Work
Algorithm	*Not Stalled*	*Memory Sys.*	*Inside Core*
Breadth-First Search	25.486	38.022	36.492
PageRank	23.704	46.683	29.613
Connected Components	20.691	43.751	35.558
Triangle Counting	23.674	37.750	38.576

表1 CPU在运行图算法时的CPU时间分布情况

为了解决上述问题，我们提出了一种基于数据流模型的方法来降低顺序语义架构的限制并减少分支预测开销。一方面，由于数据流模型中指令的执行和提交仅依赖于数据的存在性，顺序语义的限制可以被消除，从而提供较高的并行度；另一方面，数据流模型可以通过同时执行同一控制指令的所有分支的方式降低图计算中的分支预测开销。在主流数据流系统（Naiad，Tensorflow）上的实验结果表明，数据流模型可以使低指令级并行的CPU时间从62%降低至42%，将取值开销从4.25%降低至1.49%，并将分支预测开销从10.49%降低至2.57%。基于上述分析，我们进而提出了一个初步的数据流图计算加速器的设计方案。

4.1 图计算指令级并行度分析

为了充分理解核内性能低效的原因，我们设计了详细的实验以观察图计算在微体系下的行为特征。由于在传统的顺序结构中，流水线在执行复杂指令时会长时间的停滞。因此，现代处理器通常采用乱序执行的策略来同时执行多条独立指令以避免流水线停滞。在我们实验使用的测试平台中，每个CPU核在每个时钟可以调度4条微指令并分配给相应的流水线槽。在理想状态下，所有被分配的流水线槽最终都会提交。然而在运行图算法时，仅仅只有20 %~25% 的流水线槽最终提交。我们可以发现，图算法的平均IPC仅为0.867。这意味着图计算仅利用了流水线1/5的计算能力。IPC低的原因可以分为两类：一是指令的平均延迟较大；二是同时执行的指令数较少。当前研究通常认为前者是图计算的主要瓶颈，因而致力于提升图计算的内存子系统性能。然而从表1中可以看出，内存访问仅导致了54%的流水线停滞，意味着后者也可能潜在地限制图计算的性能。

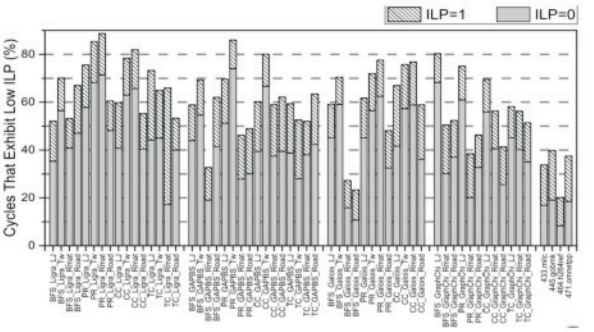


图8 图计算的指令集并行度

为了评估后者的影响，我们进一步研究了图计算的指令级并行度。从图8中我们可以看到图计算中大约62%的CPU时间只执行较低的ILP（0或者1）。而传统的SPEC应用仅在41%的时间内执行较低的ILP。更加严重的是，图计算中41%的时间只执行0条微指令。这意味着在40%的时间内，流水线完全停滞并等待之前的指令执行完毕。相比之下，SPEC应用仅有13%的时间执行0条指令。如此低的ILP一方面来源于图计算中较重的依赖性，即每条指令的执行通常都依赖于之前指令执行的结果。通过分析

图计算的依赖关系图，我们发现在大多数情况下，图计算中可执行的指令数不大于2。此外向活跃点集中写入新数据时，写入操作会由于数据竞争而被顺序化，从而引入了额外的依赖关系。低ILP的另一个原因是控制模型中的顺序语义。例如，当一个读操作导致了一个L3 不命中，乱序缓冲区（如重排缓冲区）可能很快被后续已经执行完毕的指令占满。对于一般的应用，由于cache命中率较高，顺序语义的影响可以忽略。然而在图计算较低的cache命中率使得顺序语义成为了一个较为严重的问题。通过实验我们发现，在图计算中乱序缓冲区满所造成的流水线停滞占所有资源相关类停滞的50%。

4.2 图计算分支预测性能分析

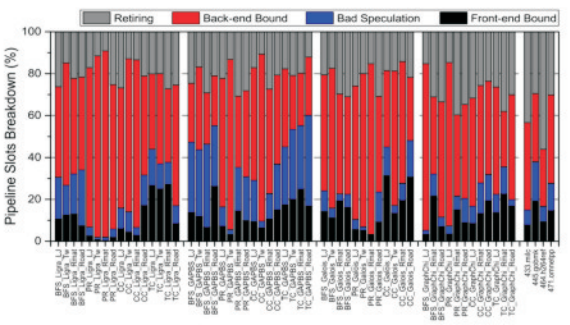


图9 执行图算法时CPU的流水线槽分布情况

尽管图计算中同时执行的指令数很低，并不是所有被执行的指令最终都能提交。处理器在处理控制分支时，通常会根据分支结果的历史对分支进行预测。当预测失败时，被去除和执行的指令将会被抛弃，从而导致相应的流水线槽被重置和浪费。从图9中我们可以看到分支预测失败导致了大约15%的流水线槽浪费，该值和SPEC中最难预测的应用之一gobmk相近（18.83%）。该实验结果表明尽管预测器可以很好的预测传统的SPEC应用，但它并不能为图计算提供较为满意的性能。相比于传统应用，图应用的分支会更加频繁和复杂，因而对预测器的要求更加苛刻。以BFS为例，一共有两种分支：一种是控制循环的while和for，另一种是计算语句中的if。前者虽然较容易预测，但是其较高的出现频率会增加取指令的开销。而后者由于控制指令的结果严重依赖于图结构，其结果难以通过历史分支结果预测，从而导致了较高的分支预测失败率。

4.3 在图计算中引入数据流

可以发现上述问题主要来源于图计算对大规模并行的需求和当前针对顺序程序设计的控制流模型之间的不匹配。为了解决上述问题，我们提出了采用数据流执行模型以对图计算的核内性能进行优化。不同于强制要求顺序提交的控制流模型，数据流模型提供了一种简单而有效的方式以描述并行计算。在数据流模型中，每个程序以一个有向图进行表述，其中节点表示操作而边表示不同节点间的数据依赖。当一个节点的所有输入数据存在时，该节点会被激活并向出边推送计算结果。

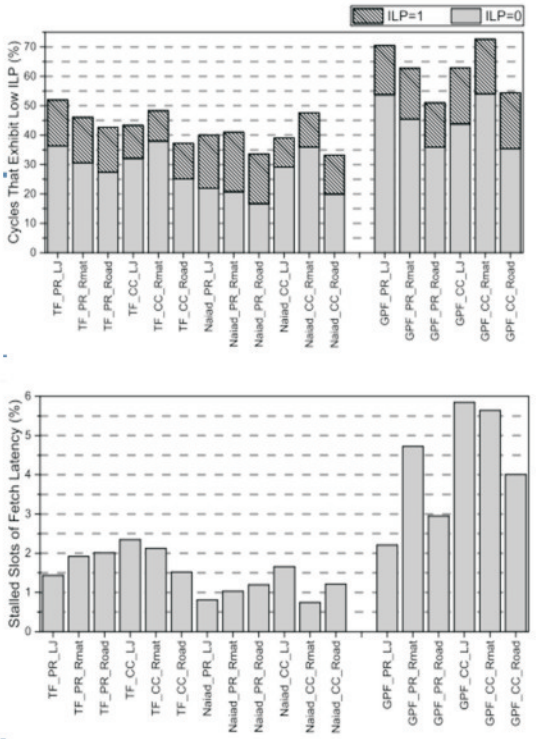


图10 数据流模型下图计算的性能

数据流模型可以有效的解决上述问题。对于ILP，一方面由于数据流模型仅仅依赖于数据依赖并将其作为指令调度顺序的唯一标准，因此图计算中的一些不必要的依赖可以被移除；另一方面，无数据依赖的指令的提交顺序并没有任何的限制，从而减少了控制流模型中顺序语义所带来的限制。对于分支预测，一方面由于循环控制逻辑隐含在数据流的开始和结束中，因此避免了频繁的判断，从而降低取指令的开销；另一方面，对于计算分支，数据流模型可以通过同时执行所有分支的方式降低分支预测失败带来的开销。从图10中可以发现，在主流的数据流系统（Naiad和Tensorflow）上，数据流执行低ILP的时间从63%降低到了42%，取指令的开销从4.25%降低至1.49%，并将分支预测开销从10.49%降低至2.57%。尽管数据流模型对于提升图计算性能有着很大的潜力，利用其解决图计算中的问题依旧是一个难题。一方面，数据流模型通常会遇到指令效率的问题。在查找可执行指令和构建输出数据时会造成较大的额外开销。与上述结论相应，我们在数据流系统上的实验也表明，当把图算法以数据流形式实现时，其总的指令数通常几倍于专用系统。另一方面，图计算的数据流图较大且与图算法紧密相连，难以设计一个通用的图计算加速器架构。

针对上述问题，我们提出了一个基于PFGA的初步方案以提升指令效率和灵活地适应不同的图算法。我们将数据流图中所有的操作映射为硬件模块，并通过硬连接的方式连接存在数据依赖的模块。由于所有指令直接通过硬件相连且可以通过数据的传输触发，因此取指令和调度的开销可以得到有效地降低。为了进一步解决数据流图过大的问题，我们对原始的图计算重新了进行了抽象和简化。

5. 面向云游戏的细粒度资源调度系统

云游戏中，游戏程序运行在云端，用户可以在移动设备上通过视频流或者图形流的方式获取游戏服务。对于云游戏提供商而言，有良好的扩展性并同时能保证一定的服务质量是至关重要的。传统的云游戏系统将游戏服务部署在虚拟机中。游戏负载通常包含三类计算密集型任务，分别为逻辑计算（通常是CPU密集型负载），高分辨率的3D渲染（GPU密集负载）和视频生成负载（CPU或者GPU负载）。这些负载将极大地影响服务质量和系统扩展性。所以发掘GPU并行处理成为了云游戏中的一个关键挑战。目前GPU虚拟化仍旧在尽力提升扩展性并减小开销。一方面，硬件虚拟化方面，诸如NVIDIA GRID，能为GPU密集型负载提供高性能和扩展性，但是在大规模部署的时候会产生巨额的花销。另一方面，以软件实现的GPU 虚拟化能通过一些诸如内存映射技术来提升扩展性和提高虚拟机性能，但缺乏对一些至关重要的硬件特性的抽象，比如硬件编码。这会导致过多的负载被加载到CPU上而导致系统扩展性较低。这些究其根本都是粗粒的资源调度引起的。

ShareRender能够充分发掘GPU的性能，扩展性好，并且开销较低。其核心理念是将GPU渲染从虚拟机中解耦出来。GPU渲染部分从虚拟机中提取出来，并独立运行在物理机上。这样CPU资源和GPU资源就能被单独进行调度，从而提升资源利用率。ShareRender进行了轻量级设计来实现这种解耦合。具体来说，ShareRender为每一个运行在虚拟机中的游戏实例启动一个守护进程Graphic wrapper，它会拦截所有游戏中产生的渲染请求。同时，多个渲染代理会在不同的物理机上启动。这些渲染代理会接收与之关联的Graphic wrapper产生的渲染请求，并进行图像渲染。这些渲染代理直接运行在物理机上，从而消除了GPU虚拟化带来的影响。为了保证正确性和高效率，ShareRender 设计了上下文同步机制来保证Graphic wrapper和渲染代理的

状态同步，并设计了影子对象来加速对GPU数据的访问。

5.1 ShareRender系统框架

ShareRender是一个基于视频流的云游戏系统。主要由一个轻量级客户端和三个主要部件构成：Graphic wrapper，渲染代理和调度器。如图12所示，ShareRender中，用户通过轻量级客户端拦截用户输入并发送游戏请求到云端。同时，客户端还负责视频接收和播放。云端会为每一个用户的游戏请求启动一个虚拟机。在每个虚拟机中，Graphic wrapper工作在游戏实例和操作系统之间，拦截3D图形库的API，并将其和相应的图形数据一起封装成图形任务。

在每一个物理机上运行着负责处理图形任务的渲染代理，并生成视频流。ShareRender为每个游戏实例创建多个渲染代理。这样，Graphic wrapper可以将渲染任务分发给任意一个与自己相关联的渲染代理上。并且，这些渲染代理能够在不同的物理机之间进行迁移。与虚拟机迁移相比，渲染代理迁移能更快完成并且开销极小。

调度器用于优化负载分发。调度器周期性地收集系统信息，例如图形任务负载和GPU利用率，但不干扰游戏服务。并决定渲染代理应该迁移到哪里，图形任务应该如何分配到渲染代理上。ShareRender的优化目标是将图形任务装箱到最少的服务器中以减少能耗。同时，需要避免频繁的渲染代理迁移以减小系统开销。

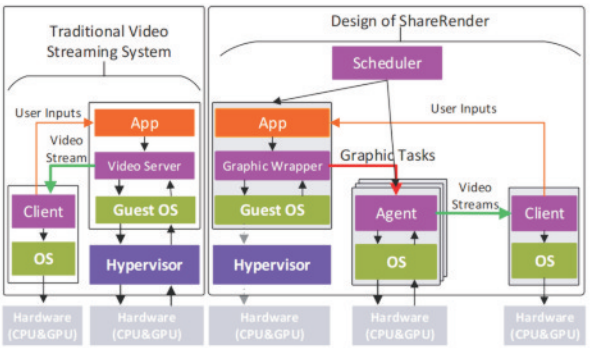


图11 云游戏架构对比

5.2 ShareRender系统设计

ShareRender由一个轻量级客户端和三个主要部件组成：Graphic wrapper，渲染代理和调度器。

Graphic wrapper的一个重要的功能是将GPU渲染负载拆分成以帧为单位以便进行细粒度调度。这和GPU的计算模型息息相关。游戏初始化GPU 并进入一个无限循环，每一个循环负责一帧画面的渲染。每一帧中，游戏首先调用FrameSetup() 设置渲染管线，包括照明设置为场景中的物体照明，几何数据的转换等。接着调用DrawPrimitives()，绘制几何对象。最后调用Present()提交这些渲染命令到GPU并产生渲染结果。当图形任务分发到渲染代理上，渲染代理需要做如下几步：1）初始化GPU，如Initialization()；2）为每一帧进行特殊的设置，如FrameSetup()；3）定位并传输DrawPrimitives()所需的几何对象数据。在系统中，设计了一个游戏上下文来表示当前的GPU状态，这个上下文包含初始化所需的数据和操作，渲染管线的设置，几何对象等等。游戏上下文对于保证基本图形任务能够在不同的渲染代理上得到正确的渲染结果至关重要。

ShareRender设计上下文同步机制以实保证渲染代理上的渲染正确性，并设计了影子对象，影子对象保存在主存中以加速GPU数据的访问。

每一个渲染代理由图形渲染器和视频模块组成。图形渲染器提取渲染命令和参数，并根据本地上下文渲染出游戏场景。当Graphic wrapper 将一帧图像所需的图形命令提交时，通常是通过调用Present()函数完成，此时视频模块就会将渲染出的游戏场景编码压缩成视频流。视频模块可以使用CPU编码器，也能使用GPU编码器。

ShareRender中，渲染代理的迁移是通过关

闭当前的渲染代理，并在另一台被调度器指定的物理机上创建新的渲染代理完成的。在新的渲染代理创建后，它会同步Graphic wrapper上的游戏上下文。ShareRender实现了一种与预拷贝相似的在线迁移机制，ShareRender不会停机进行迁移，而且迁移时间远小于虚拟机的迁移时间。

调度器是为了优化图形任务的分配而设计的。ShareRender提出一种不需要利用未来负载信息进行调度的在线启发式算法。具体来说，调度器周期性地运行，每个游戏实例最多拥有K个渲染代理。在每一个调度周期开始，ShareRender预测出当前调度周期中图形任务的负载。为了减少所需物理机的数目，我们采用和装箱算法相同的准则，即，总是选择最合适合于当前图形任务的服务器。然而，选择的服务器上可能并没有该游戏实例的渲染代理存在，这就会导致一次渲染代理的迁移。为了控制迁移开销，选择服务器时更倾向于那些有渲染代理存在的服务器。如果候选服务器没有足够的资源，那么渲染代理就会迁移到拥有足够资源的服务器上。

ShareRender的客户端是为那些诸如智能手机和笔记本电脑等物理资源受限的计算机和设备设计的。我们采用可移植的库实现以增强其适应性。客户端有两个主要功能：拦截用户输入和播放游戏视频。用户输入通过SDL库进行拦截，并传送给Graphic wrapper。视频流和音频流是通过基于live555库实现的RTSP客户端进行处理。RTSP客户端负责处理多路视频流并切换帧数据源。

附成果列表：

专利：

- DRAM/NVM HIERARCHICAL HETEROGENEOUS MEMORY ACCESS METHOD AND SYSTEM WITH SOFTWARE-HARDWARE COOPERATIVE MANAGEMENT，专利申请号15287022，专利申请日期2016年7月1日，金海，廖小飞，刘海坤，陈宇杰，郭人通
- 软硬件协同管理的DRAM-NVM层次化异构内存访问方法及系统，专利申请号CN201610166238.0，专利申请日期2016年3月22日，廖小飞，刘海坤，金海，陈宇杰，郭人通
- 内存访问方法及装置，专利申请号CN201610822569.5，专利申请日期2016年9月13日，廖小飞，刘海坤，董诚，金海
- 一种内存访问方法与计算机系统，专利申请号CN201710289650.6，专利申请日期2017年4月27日，刘海坤，廖小飞，陈吉
- 分布式深度神经网络集群分组同步优化方法及系统，专利申请号201710191685.6，专利申请日期2017年3月28日，蒋文斌，金海，叶阁焰，张杨松，马阳，祝简，彭晶
- 深度神经网络模型并行的全连接层数据交换方法及系统，专利申请号201710191684.1，专利申请日期2017年3月28日，蒋文斌，金海，张杨松，叶阁焰，马阳，祝简，刘湃

论文：

- Wei Zhang, Xiaofei Liao, Peng Li, Hai Jin, Li Lin, ShareRender: Bypassing GPU Virtualization to Enable Fine-grained Resource Sharing for Cloud Gaming. ACM MM 2017.
- Haikun Liu, Yujie Chen, Xiaofei Liao, Hai Jin, Bingsheng He, Long Zheng, Rentong Guo: Hardware/software cooperative caching for hybrid DRAM/NVM memory architectures. ICS 2017: 26:1-26:10.
- Hai Jin, Pengcheng Yao, Xiaofei Liao, Long Zheng, Xianliang Li: Towards Dataflow-Based Graph Accelerator. ICDCS 2017: 1981-1992.
- Zaiyang Tang, Peng Li, Song Guo, Xiaofei Liao, Hai Jin, Daqing Zhang: Selective Traffic Offloading on the Fly: A Machine Learning Approach. ICDCS 2017: 2386-2392.
- Di Chen, Hai Jin, Xiaofei Liao, Haikun Liu, Rentong Guo, Dong Liu: MALRU: Miss-penalty aware LRU-based cache replacement for hybrid memory systems. DATE 2017: 1086-1091.
- Yu Zhang, Lin Gu, Xiaofei Liao, Hai Jin, Deze Zeng, Bing Bing Zhou: FRANK: A Fast Node Ranking Approach in Large-Scale Networks. IEEE Network 31(1): 36-43 (2017).

- Yu Zhang, Xiaofei Liao, Hai Jin, Lin Gu, Guang Tan, Bing Bing Zhou: HotGraph: Efficient Asynchronous Processing for Real-World Graphs. IEEE TC 66(5): 799-809 (2017).
- Xiaofei Liao, Rentong Guo, Hai Jin, Jianhui Yue, Guang Tan: Enhancing the Malloc System with Pollution Awareness for Better Cache Performance. IEEE TPDS 28(3): 731-745 (2017).
- Haikun Liu, Bingsheng He, F2C: Enabling Fair and Fine-grained Resource Sharing in Multi-tenant IaaS Clouds, IEEE TPDS.
- Long Zheng, Xiaofei Liao, Hai Jin and Haikun Liu, "Exploiting the Parallelism Between Conflicting Critical Sections with Partial Reversion", IEEE TPDS, acceptted.
- Wei Zhang, Xiaofei Liao, Peng Li, Hai Jin, Bingbing Zhou, Li Lin, Fine-grained Scheduling in Cloud Gaming on Heterogeneous CPU-GPU Clusters. IEEE Network, accepted.
- Haikun Liu, Bingsheng He, Xiaofei Liao, Hai Jin, "Towards Declarative and Data-centric Virtual Machine Image Management in IaaS Clouds," in IEEE TCC, accepted.
- Xiaofei Liao, Rentong Guo, Danping Yu, Hai Jin, Li Lin: A Phase Behavior Aware Dynamic Cache Partitioning Scheme for CMPs. IJPP 44(1): 68-86 (2016).
- Siyuan Wang, Xiaofei Liao, Xuepeng Fan, Hai Jin, Qiongjie Yao, Yu Zhang: Automatically Setting Parameter-Exchanging Interval for Deep Learning. MONET 22(2): 186-194 (2017).
- Huiming Lv, Zhiyuan Shao, Lang Li, Xuanhua Shi, Hai Jin. A Task-based Approach for Finding SCCs in Real-world Graphs on External Memory. CCPE, accepted.
- Jinbao Zhang, Xiaofei Liao, Hai Jin, Dong Liu, Li Lin, Kao Zhao: An Optimal Page-Level Power Management Strategy in PCM-DRAM Hybrid Memory. IJPP 45(1): 4-16 (2017).
- Zhiyuan Shao, Zhenjie Mei, Xiaofeng Ding, and Hai Jin, BlockGraphChi: Enabling Block Update In Out-of-core Graph Processing, IJPP, accepted.
- Xiaofei Liao, Long Zheng, Binsheng Zhang, Yu Zhang, Hai Jin, Xuanhua Shi, Yi Lin, "Dynamic Cluster Strategy for Hierarchical Rollback-recovery Protocols in MPI HPC Applications", CCPE, 2017, accepted.
- Chenxi Li, Xiaofei Liao, Hai Jin, "Enhancing Application Performance via DAG-Driven Scheduling in Task Parallelism for Cloud Center", PPNA, 2017, accepted.
- Wenbin Jiang, Yiming Chen, Hai Jin, Ran Zheng, Ye Chi, A Novel GPU-based Efficient Approach for Convolutional Neural Networks with Small Filters, JSPS, 86(2):313–325, (2017).
- Wenbin Jiang, Bin Luo, Hai Jin, Alan L. Yuille, and Jinsheng Xiao, A Novel Parallelized Feature Extraction in Grouped Scale Space Based on Graphic Processing Units, JIT, 17(5): 1061-1069, (2016).

王孝远

博士生

研究方向：内存计算

Email：xiaoyuanw@hust.edu.cn

姚鹏程

博士生

研究方向：图计算

Email：hustyaopc@gmail.com

张　伟

博士生

研究方向：异构计算

Email：alanzw@163.com

刘海坤

副教授

研究方向：虚拟化，内存计算

Email：hkliu@hust.edu.cn

廖小飞

教授

研究方向：系统软件

Email：xfliao@hust.edu.cn .cn

云计算与移动计算组典型成果介绍

陶 晟，顾 琳

关键词：网络功能虚拟化 GPU加速 网络流调度

引 言

云计算与移动计算小组长期致力于数据中心领域的研究，该领域也一直是云计算的热点。近年来，网络功能虚拟化（NFV）[1]的快速发展以及GPU等计算加速器的广泛使用，给数据中心领域带来了新的研究机遇和挑战。本次展示的成果是我们在该领域的一项重要工作，反映了我们组的最新进展。

随着通信网络的快速发展，传统的使用专用硬件的网络架构已经无法满足运营商的需求。为了提高网络业务部署的灵活性和可管理性，加快网络服务的更新，减少网络设备的投资维护成本，欧洲电信标准组织(ETSI)提出了一种软件与硬件分离的新型网络架构，即网络功能虚拟化。通过使用虚拟化技术，将网络功能部署到通用的大容量服务器、存储和交换机上。尽管网络功能虚拟化可以带来一系列的好处，然而在性能方面，通用服务器却远远比不上专用硬件[2]。为了解决NFV的性能问题，近年来一些研究工作通过将计算卸载到GPU上来提高网络功能的处理能力，在单个节点上满足了高速I/O的需求。然而，在一个NFV网络系统中，网络流会在不同节点上处理，缺乏合理的网络流调度依然会降低整个系统的性能，因此需要设计一种NFV下的网络流调度算法。然而，现有的调度方式为静态调度，无法适应动态变化的网络环境。针对目前研究的一些缺点，我们首先进行网络建模与数学分析，提出了一种动态网络流调度算法，通过模拟实验验证了其有效性，最后根据调度算法的思想，设计了一种基于GPU的网络流处理加速系统。

背景和相关工作

GPU加速

现有的NFV系统中的网络功能大都运行在CPU上，然而随着万兆网卡的普及，通用服务器上有限的核数在面对计算密集型网络功能时，并不一定能支持高吞吐率(10Gbps以上)，从而导致CPU性能成为影响网络流吞吐率的瓶颈。为了解决这个问题，人们开始探索硬件加速的方案，其中一种方案就是将网络功能放到GPU上运行。由于GPU与CPU相比具有更多的核数（上千个），因此被广泛应用于大规模并行计算。在处理网络流的时候，通常每一个数据包的处理都是独立的，相互之间没有依赖性，这种独立性使得我们可以将上千个数据包组合起来，同时放到GPU上进行并行处理，每一个GPU线程运行一段相同的代码来处理数据包。

根据这种思想，目前已经有多个研究工作提出了自己的GPU数据包处理框架和系统[6,7,8]，使用这些框架可以达到40Gbps的吞吐率，从而证实了GPU加速的可行性和有效性。然而这些框架和系统仅能够有效地提升了单个GPU节点上理数据包的处理性能，但它们并没有考虑网络流处理的问题。相比传统网络，NFV中网络流调度的一个最大特点是，网络流

从源点出发后，必须依次经过一个给定的服务链上的网络功能（如图1所示），最后才能交付到目的点。这种有序的限制加大了网络流调度的复杂程度，再加上各个节点计算能力的不平衡，不合理的调度将会影响单条网络流，甚至是整个网络服务的性能。如图2所示的一个例子，数字代表节点的计算能力，不同颜色节点代表不同网络功能，一条网络流需要经过图1所示的服务链，在两种不同的网络流调度方案下分别可以获得10和20的网络流吞吐率。因此，我们需要设计一种适用于NFV网络环境的网络流调度算法来提高网络服务的性能。

图1 一条典型的服务链

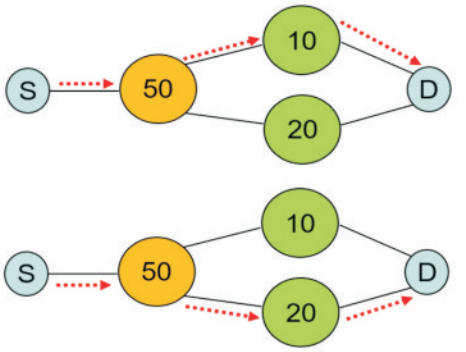


图2 不同的网络流调度方法对网络流吞吐率的影响

网络流调度

现有的研究工作[3,4,5]主要基于静态的网络流调度，它们假设网络流的大小是已知并且在一段时间内是稳定不变的。然而在实际网络环境中，网络流的大小可能随时发生变化。此时，静态的网络流调度将不再是一种最优调度，可能会导致网络资源分配不均，甚至导致整个网络性能下降或网络拥塞的情况。在这种情况下，就需要根据当前网络环境进行动态的网络流调度，使得系统资源得到充分的利用，保证网络性能最优。此外，由于不同网络功能需求的资源不同，会带来网络吞吐率和公平性的权衡问题，不合理的调度可能会出现资源需求高的网络流被饿死的情况。

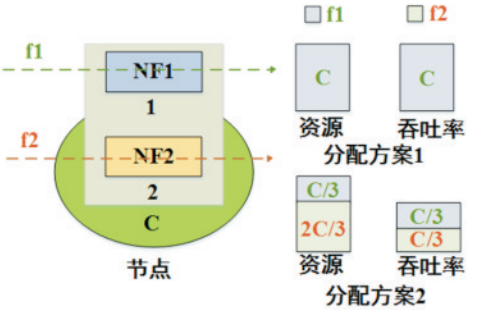


图3 不同分配方案对吞吐率和公平性的影响

如图3所示，在一个可用资源为C的节点上有两个网络功能NF1和NF2，它们的单位吞吐量的资源消耗比是1：2。现在有两条网络流f1和f2分别经过NF1和NF2，那么我们可以看到分配方案1为了最大化总吞吐率为C，分配C的资源给f1，此时f2的吞吐率为0，导致严重的不公平。而分配方案2解决了这种不公平问题，使得f1和f2分别达到了C/3的吞吐量，但同时也牺牲了总的吞吐率。

针对上述这些问题，我们通过建立模型和数学分析提出了一种综合考虑网络吞吐率和公平性的网络流动态调度算法，该算法能够在保证一定公平性的情况下最大化整个网络的吞吐率。

解决方案

网络流动态调度算法

1. 建模与分析

我们考虑数据包从源点出发需要依次被一条服务链上的网络功能处理，最后到达目的点。我们假设所有网络功能已经放置在不同的网络节点上，并且已经处于运行状态。不同类型的网络功能的实例可以放置在同一个网络节

点上并且可以共享其计算资源。我们要将数据包进行分类，将它们放入不同的队列中，并在每一个时刻决定如何处理和传输队列中的数据包以及接纳新到来的数据包，来最优化网络性能。

动态队列

首先，为了能够动态调度网络流，我们对网络系统进行抽象，建立了一个基于动态队列的离散时间系统网络模型。我们在每一个网络节点上，为每一条网络流所需求的每一个网络功能都维护一个队列，如一条网络流需要经过三个网络功能，那么我们就维护三个队列。我们假设队列没有容量限制，队列初始时刻的大小为0。t时刻下，在节点n上，网络流f需要被第k 个网络功能处理的数据包队列的大小为 $Q_n^{f,k}(t)$ 。队列的出队部分包括被网络功能处理的数据包 $\mu_n^{f,k}(t)$ ，以及传输到其他节点上的数据包 $\lambda_{nj}^{f,k}(t)$，队列的入队部分包括前一个网络功能处理完毕的数据包 $\mu_n^{f,k-1}(t)$，从其他节点传输到本节点上的数据包 $\lambda_{in}^{f,k}(t)$，以及从外部接纳的数据包 $R_n^f(t)$ 。可以看到，每一个时刻处理，传输和接纳的数据包数量将会影响队列大小。我们需要根据当前网络负载来确定这三种控制决策，从而确保网络的性能。

目标函数

我们关注的网络性能指标主要为网络流的平均吞吐率r_f，为了同时实现网络流之间的公平性，我们使用效用函数[9]作为最终的性能指标来代替网络吞吐率。效用函数的公式如下所示：

$$\mathrm{U}_f(r_f,\ \alpha_f)=\begin{cases}(1-\alpha_f)^{-1}r_f^{1-\alpha_f}, & \alpha_f\neq 1,\\ \log(r_f), & \alpha_f=1.\end{cases}$$

其中α_f是一个控制吞吐率和公平性权衡关系的参数，α_f 取值越大，那么我们可以取得更好的公平性，但同时也会牺牲一定的吞吐率，当α_f=0时，该效用函数就退化为平均吞吐率。我们的目标函数就是最大化所有网络流的效用函数之和，如下所示：

$$\max:\sum_{f\in F}U_f(r_f,\alpha_f),$$

约束条件

1. 资源约束：由于每个节点上的系统资源（CPU，带宽等）是有限的，数据包处理和传输的控制决策结果将受限于资源约束，如cpu资源，节点之间的带宽资源。

2. 吞吐率约束：由于我们不考虑缓存数据包，所以每一个时刻能够接纳的数据包不能超过该时刻到达的数据包，未被接纳的数据包将会被丢弃。此外，我们对每一条网络流的吞吐率都设置了上限和下限。

3. 网络稳定性约束：我们需要确保网络处于稳定状态，不会发生网络拥塞，即网络中的数据包个数不会无限增长。

李雅普诺夫分析

网络流动态调度面临的一个大问题就是网络流大小的波动性和不可预测性。为了解决该问题，我们使用李雅普诺夫分析方法[10]来求解我们建立的网络模型。将目标函数转化为与控制决策相关的多个优化函数，从而设计出相应的网络流动态调度算法，这样能够有效地处理网络中的随机性，同时保证网络的平均性能达到最优。我们首先通过引入辅助变量，建立虚拟队列，构造李雅普诺夫函数，以及进行一系列的公式转换，将原先的最大化问题转化为最小化问题。其次，由于最小化问题的多个表达式相互之间具有独立性(每一个表达式只含有一种决策控制变量)，该最小化问题又可以转化为四个独立的最小化子问题：辅助变量决策、接纳控制决策、传输控制决策和计算控制决策。

2. 算法描述

我们的动态调度算法的总体思想就是在每一个时刻根据当前队列负载情况和到达的数据包来决定每一个控制决策变量的取值，然后更新队列，完成一次迭代。

1. 辅助变量决策：辅助决策变量取值是通过对辅助变量子问题进行求导后取零点求得。

2. 接纳控制决策：接纳控制决策实际上表达了一个根据阈值来接纳新到来的数据包的思想，如果当前网络流的第一个队列小于阈值，那么我们就接纳所有新到来的数据包，否则表明当前队列负载过大，拒绝接纳所有新到来的数据包。

3. 传输控制决策：传输控制决策实际上是一个经典的背压算法[11]。其核心思想就是计算相邻两个节点上的所有同一种类型的队列长度差值。我们将所有带宽资源分配给具有最大队列长度差值的队列，只传输该队列上面的数据包，其他队列上的数据包不进行传输。

4. 计算控制决策：计算控制决策的核心思想是计算同一个节点上相邻两个队列的队列长度差值与网络功能资源消耗的比值，我们将所有计算资源分配给具有最大比值的队列，优先处理该队列上面的数据包。

5. 更新队列：最后，我们得到了各种控制决策后，就根据动态队列公式和虚拟队列公式更新相应队列。

3. 实验验证

为了验证算法的有效性，我们设置了模拟实验，我们使用一个具有11个节点以及14条链路的网络拓扑Abilene Network（图4），同时配置了四条不同的网络流，在四个节点上放置了不同的网络功能（图5）。我们使用Jain's Index作为评价公平性的指标。

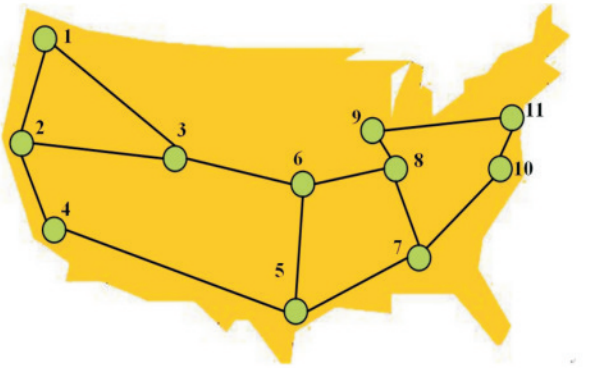


图4 网络拓扑图

TABLE II
FLOW SETTING

index	source node	destination node	service chain
1	1	11	$nf1 \rightarrow nf2$
2	2	10	$nf2 \rightarrow nf3$
3	2	11	$nf1 \rightarrow nf3$
4	1	10	$nf1 \rightarrow nf2 \rightarrow nf3$

TABLE III
COMPUTATION NODE SETTING

node index	capacity	vnf set
5	200	$nf1, nf2$
6	200	$nf1, nf3$
7	200	$nf2, nf3$
8	200	$nf1, nf2, nf3$

图5 实验设置

首先，我们通过调节参数V观察在不同α下效用大小和队列长度的关系，实验结果如图6(a)和图6(b)所示，我们可以发现随着V的增大，效用会快速增长直到收敛到一个最优值，而队列长度和V几乎成为一个线性增长的关系。该实验结果证明了我们算法的最优性。

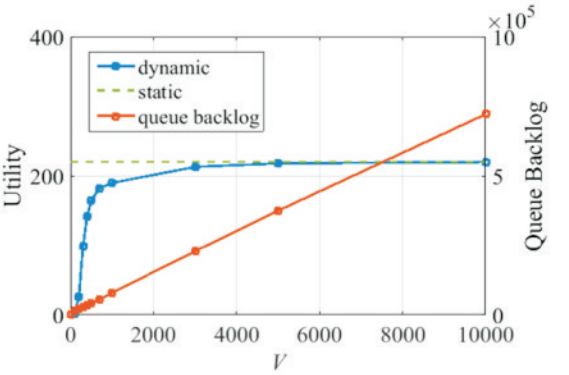


图6(a) α = 0

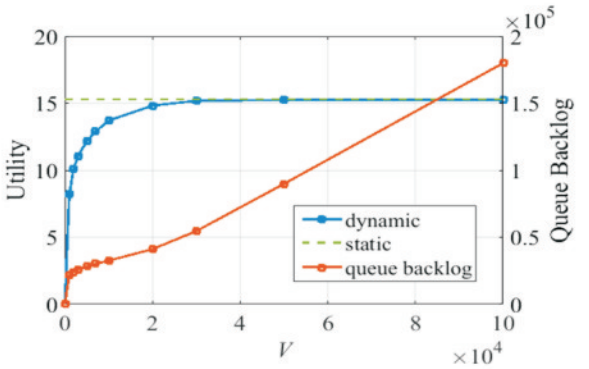


图6(b) α = 1

接着，我们通过调节参数α来观察它对吞吐率和公平性的影响，实验结果如图7所示，可以发现当α增大时，吞吐率将会减少，公平性将会提高，这与理论上的结果一致。

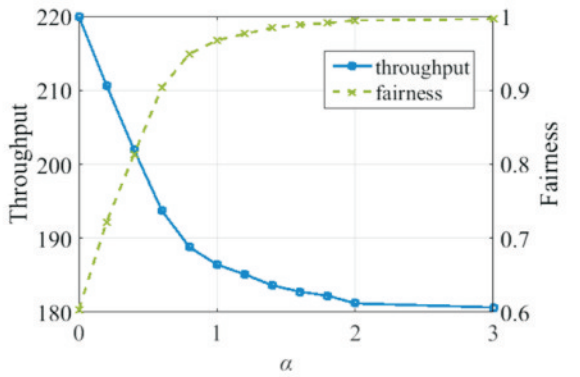


图7 吞吐率和公平性关系

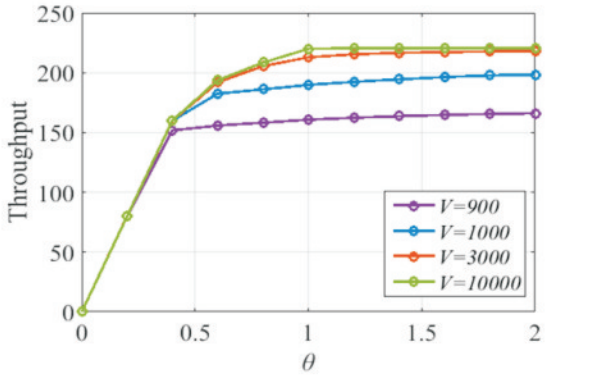


图8 接纳控制决策影响

然后，我们分析了接纳控制决策的作用，即网络流平均到达率θ和吞吐量的关系，实验结果如图8所示。我们可以发现，刚开始吞吐率和θ成为线性增长的关系，这是因为，网络中的资源足够能够接纳所有到达的网络流。一旦θ增长到一定程度后，吞吐率将会趋于不变，这是因为网络流大小超过了网络所能承载的范围，采用接入控制决策丢弃一定数据包使得网络系统处于稳定状态。

之后，我们分析了吞吐率约束对网络流吞吐率的影响，实验结果如图9(a)和图9(b)所示。我们发现在无吞吐率约束的情况下，不同网络流的吞吐率差异很大，导致网络流之间的公平性很差，且随着V的增长，这种情况越来越严重。当我们为网络流设置吞吐率约束以后，我们发现吞吐率约束起到了效果，网络流的吞吐率被限制在一定范围内，网络流之间的公平性得到了提升。

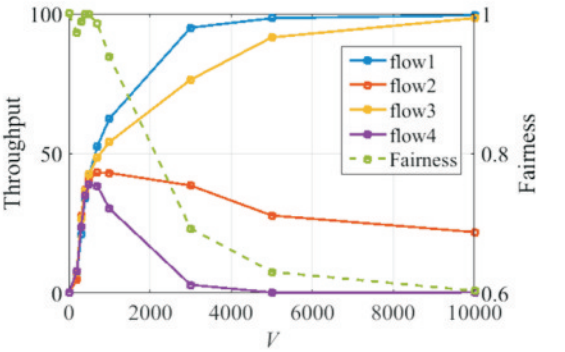


图9(a) 无吞吐率约束

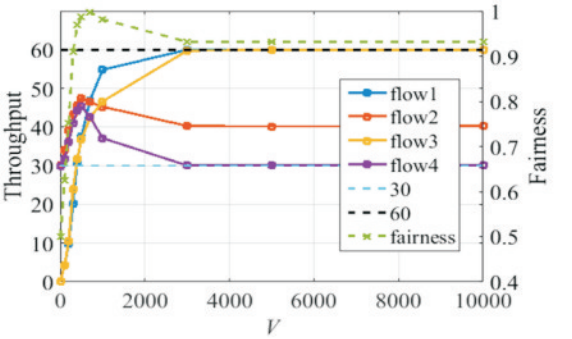


图9(b) 有吞吐率约束

最后，我们设置了不同的网络流数目来观察它对吞吐率和公平性的影响，实验结果如图10所示。我们可以发现，当α=0时，随着网络流数目增长，总吞吐率几乎保持不变，然而网络流之间的公平性却在减少。当α=1时，随着网络流数目增长，公平性几乎保持不变，然而总吞吐率在减少。可见，随着网络流数目的增加，公平性问题越严重，但是通过设置更大的α，我们可以改善公平性问题，但是会牺牲一定的总吞吐率。

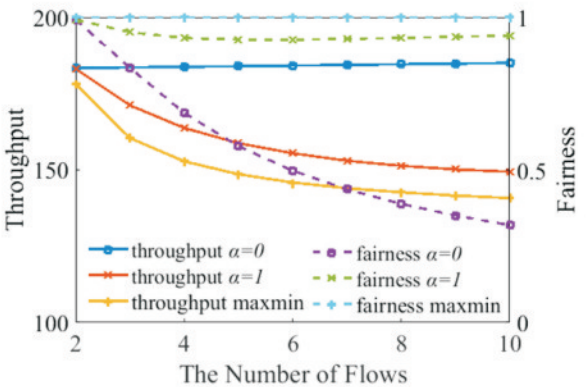


图10 网络流数目的影响

4. 基于GPU加速的网络流处理系统

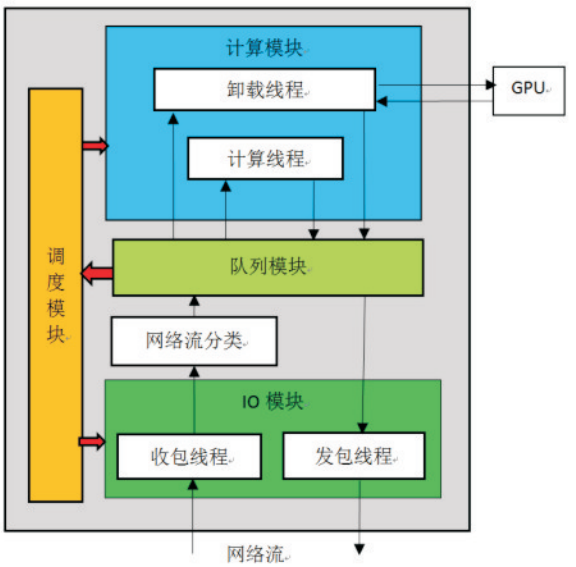


图11 网络流处理系统框架图

上述动态调度算法是从一种比较理想的抽象网络模型中运用数学方法推导出来的，实验证明其具有一定的理论价值。但是如果要应用到实际网络中，有些地方设计的时候必须要有近似处理。为了验证算法的可行性，我们设计了一种基于上述动态调度算法的网络流处理系统。

系统架构

目前我们设计的系统架构如图11所示。该系统主要分为四个模块，IO模块，队列模块，计算模块以及调度模块。

I/O模块：我们的系统目前使用PF_RING作为I/O框架用于接收和发送数据包，PR_RING可以绕过传统的Linux协议栈直接将数据包发送到用户空间的应用程序，从而达到提高IO性能的目的，并且不需要修改内核的网卡驱动。收包线程接收到一个数据包后，需要将数据包的五元组当作键值去查询流表。如果属于流表中的某一条流，我们就将它放到相应的缓存队列中去，否则，就丢弃该数据包。此外，我们为网络流的第一个队列的长度设置一个阈值，一旦队列长度达到阈值，我们就丢弃到来的数据包。发包线程根据调度模块的传输决策结果，从相应队列中取出数据包，发送到网卡中。

队列模块：由于系统内存的限制，实际队列的容量不可能是无限的，因此，在实际过程中，如果下一个队列已经满了，可能会在中间丢弃一些数据包，当然，由于我们使用了接纳控制决策，通常只会在进入第一个队列之前就丢弃数据包。

计算模块：计算线程根据调度模块的处理决策结果，从队列中取出数据包，并执行相应网络功能的代码，我们并没有为一个计算线程绑定一个网络功能，这是区别于传统框架的地

方。将一个网络功能与一个计算线程绑定，实际上是固定了这个网络功能需要的计算资源，而这个网络功能有时候未必需要这么多资源处理当前的负载。为了充分利用当前计算资源，在我们设计中，一个计算线程可以随时切换执行的网络功能，大大提高了数据包处理的灵活性。一旦我们处理完数据包，就立即将它放入下一个队列。

卸载线程目前不进行计算的工作，而是从队列中取出一组数据包组成一个批。组批的原因是为了提高内存带宽的吞吐率，然后将这个批形成一段连续内存传输到GPU内存中。之后发送执行核函数的命令，每一个核函数是一段在GPU上执行的网络功能代码。GPU上的代码执行完毕后，我们就从GPU内存取回数据包，并将这一组数据包放到下一个队列。我们为每一个批分配一个不同的流，这样的话如果GPU上当前空闲资源足够多，可以使得多个流上的批同时在GPU上运行。此外，为了提高性能，我们使用异步的方式不断地将组成的批放入队列中，而不是等待批的完成才去形成下一个批。卸载线程在每一次循环中负责查询队列头部的批的完成情况，一旦批完成，我们就进行内存传输工作，然后查询下一个批，否则就进行组批的工作。

调度模块：调度模块在每一个时间间隔开始的时候执行我们的动态调度算法，理论上我们要根据队列长度信息和当前可用的资源来计算这个时间间隔需要处理的数据包，在实际环境中，当前可用的资源可能会不停变化，因此我们在系统实现中作了如下的近似处理，由于我们的调度算法基于贪心策略将所有的可用资源都用来服务一个队列，因此我们将计算线程中的网络功能设置为该队列所需的网络功能，并且不断的从队列中取出数据包进行处理。对于发包线程，我们同样选一个最长的队列，然后在该时间间隔开始到结束阶段，只发送该队列的数据包。

存在的问题和将来的工作

由于GPU与CPU的硬件架构和特性有很大不同，当我们把CPU上的代码迁移到GPU上后，会存在如下的一些问题。

1. 如何决定批的大小?

在CPU上我们每次处理一个数据包，而在GPU上，我们每次处理一批数据包。根据GPU的特性，批越大，越能发挥GPU并行处理的优势，减少数据传输的开销，从而提高吞吐率。然而批越大也会导致组批时间的增加，尤其是在负载较低的时候，这样就间接增加了数据包的处理延时，对于一些延时要求较高的网络流，可能会有影响。一种可能的解决方案是针对网络流的特性，设置一个合理的批大小，并且根据当前负载情况，动态调整批大小。

2. CPU和GPU如何进行共同处理?

使用GPU的好处是可以提高吞吐率，然而我们在实验中发现GPU的使用会相应的增加额外的延时，尤其是在负载较低的时候。此时，可以考虑利用CPU的可用资源来进行数据包的处理。因此在共同处理模式下，何时使用CPU进行处理，何时加入GPU进行处理来提高吞吐率，是一个有待解决的问题。合理地使用CPU和GPU的资源才能达到最大化资源利用率同时提高吞吐量的目的。

3. 如何处理GPU网络功能中的分支?

CPU具有复杂的分支处理逻辑单元，能够指令并行，可以进行分支预测。而GPU指令按顺序执行，没有分支预测，遇到分支问题时，性能会发生较大变化。因此在编写GPU 网络

功能时，需要考虑在保证逻辑正确的前提下尽可能减少分支，来减少对GPU性能的影响。

总 结

针对目前网络功能虚拟化环境下的网络流加速机制研究的不足，我们首先建立基于动态队列的网络模型并使用李雅普诺夫分析框架提出了一种动态的网络流调度算法。相比静态的网络流调度，它能实时地保证网络处于稳定状态，并且使网络平均性能达到最优。随后我们设计了一种基于GPU的网络流处理加速系统，该网络流处理系统的调度模块基于我们提出的调度算法，能够根据当前队列负载情况，选择最优队列进行服务，在兼顾网络服务公平性的基础上达到提高网络吞吐率的目的。

参考文献

[1] NFV, http://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_white_Paper3.pdf

[2] Bronstein Z, Roch E, Xia J, et al. “Uniform Handling and Abstraction of NFV Hardware Accelerators,” IEEE Network, Vol. 29 No. 3, 2015, pp. 22-29.

[3] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba, “Elastic Virtual Network Function Placement,” in Proc. of IEEE CloudNet, Oct 2015, pp. 255-260.

[4] P. W. Chi, Y. C. Huang, and C. L. Lei, “Efficient NFV Deployment in Data Center Networks,” in Proc of IEEE ICC, 2015, pp. 5290-5295.

[5] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, “Deploying Chains of Virtual Network Functions: On the Relation Between Link and Server Usage,” In Proc. of IEEE INFOCOM, 2016, pp. 1-9.

[6] Sangjin Han, Keon Jang, KyoungSoo Park, Sue B. Moon, “PacketShader: a GPU-accelerated software router,” In Proc. of ACM SIGCOMM, 2010, pp. 195-206.

[7] Anuj Kalia, Dong Zhou, Michael Kaminsky, David G. Andersen, “Raising the Bar for Using GPUs in Software Packet Processing,” In Proc. of USENIX NSDI, 2015, pp. 409-423.

[8] Weibin Sun, Robert Ricci, “Fast and flexible: Parallel packet processing with GPUs and click,” In proc. of IEEE ANCS, 2013, pp. 25-35.

[9] J. Mo and J. Walrand, “Fair End-to-End Window-Based Congestion Control,” IEEE/ACM Transactions on Networking, vol. 8, no. 5, Oct 2000, pp. 556-567.

[10] M. J. Neely, “Stochastic Network Optimization with Application to Communication and Queueing Systems,” Synthesis Lectures on Communication Networks, vol. 3, no. 1, 2010, pp. 1-211.

[11] L. Tassiulas and A. Ephremides, “Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks,” IEEE Transactions on Automatic Control, vol. 37, no. 12, Dec 1992, pp. 1936-1948.

陶 晟

2015级硕士研究生

研究方向：网络功能虚拟化

Email：523717980@qq.com

顾 琳

博 士，讲 师

主要的研究兴趣包括网络功能虚拟化、软件定义网络、云计算等

Email：anheeno@gmail.com

网络空间安全组典型成果介绍

袁 斌，徐 鹏，杜亚娟，代炜琦，羌卫中

关键词： 云网络安全　大数据安全　数据存储安全　区块链安全　Web安全

1 介绍

网络与信息安全作为国家高度重视的核心技术，已经成为国内外的热点领域。随着被正式获批为国家一级学科，网络空间安全在国内的工业界和学术界掀起又一波研究热潮。围绕当前热门的IT技术引入的安全挑战，在上一年度，本组主要做了如下五个方面的研究：1）云网络安全；2）大数据安全；3）数据存储安全；4）区块链安全；5）Web安全。主要代表性成果体现在：在云网络安全方面，针对云主机既可能受到攻击又可能被攻击者利用发起攻击的问题，开发了结合软件定义网络技术和分布式处理技术的云主机全面保护系统。在大数据安全方面，设计了带条件的基于身份的广播代理重加密技术解决了传统的邮件加密技术无法兼顾安全性和用户使用时的友好性的问题。在数据存储安全方面，实现了高密度闪存系统的可靠性优化技术，解决由于性能优化带来的可靠性问题，同时保证了闪存系统的优化机制的高效性。在区块链安全方面，提出了POV机制激励真正为社会贡献价值的群体，以激发更多的价值创造，解决了目前区块链主流共识机制POW或POS没有激励价值创造者的问题。在Web安全方面，提出了可应用于浏览器的基于标签的细粒度信息流控制模型，高效灵活地保障用户数据的隐私性和应用程序的完整性，解决了当前的浏览器无法保障用户数据的隐私性和应用程序的完整性的问题。

2 成果介绍

2.1 HostWatcher：基于软件定义网络技术的云主机保护系统

在云主机保护方面，针对云主机既可能受到攻击又可能被攻击者利用发起攻击的问题，开发了结合软件定义网络技术和分布式处理技术的云主机全面保护系统HostWatcher。该系统既能有效缓解针对云主机的DDoS攻击，又能极大的限制被攻击者控制的主机的攻击速率。此外，通过流量缓存与重放、基于网络协议的多队列轮询以及多级流表的应用，能在缓解攻击的同时最大程度的保证服务质量。

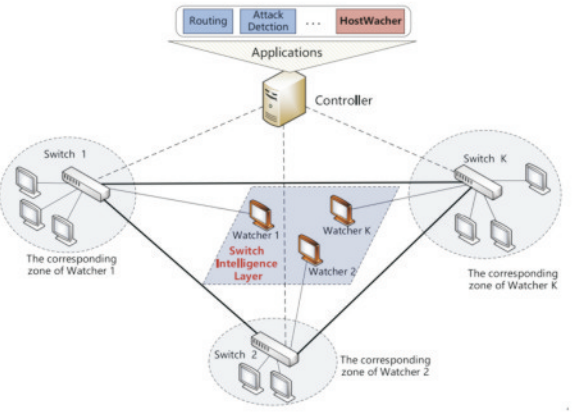


图1-1 HostWatcher系统部署图

一方面，云计算技术在全球范围内得到广泛应用，越来越多的服务都迁移到云计算环境中，因此，云主机成为了网络攻击的新目标；另一方面，云计算技术的高速发展，云资源的使用变得越发方便和便宜，因此，攻击者开始租用云主机来发起网络攻击。然而，现有的云

主机保护方案往往不能兼顾这两种情况，无法为云主机提供全面的保护。为解决该问题，设计并研发HostWatcher系统，该系统既能有效缓解针对云主机的DDoS攻击，又能高效限制恶意云主机的攻击速率。同时，流量缓存与重放、多队列轮询等机制能在保证缓解攻击的同时最大程度保证服务质量，进一步增强系统的实用性。

为增加云主机保护系统的灵活性与鲁棒性，HostWatcher系统同时利用了软件定义网络技术和分布式处理技术：基于软件定义网络技术提供的集中控制模式，HostWatcher能对云网络进行全面的分析，进而快速制定全局网络策略；基于软件定义网络技术的可编程性，可以实现转发规则的自适应产生、安装与删除，从而实现网络策略的高效执行与更新；通过子网划分，利用分布式处理技术的优势来实现负载均衡，避免单点故障，增强保护系统自身的鲁棒性。

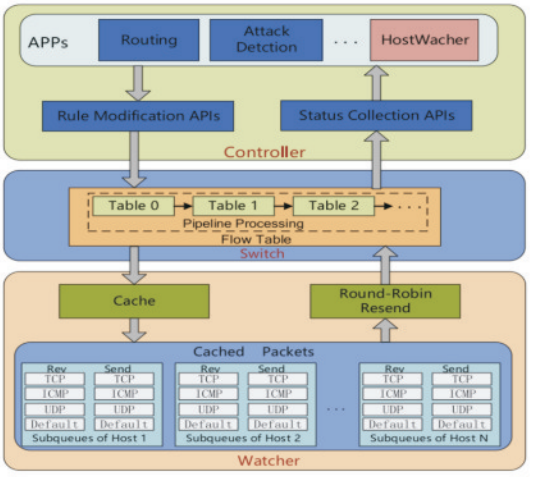


图1-2 HostWatcher的系统结构

为了保证攻击缓解过程中的服务质量，对可疑主机的流量进行先缓存再低速重放的操作，从而降低网络的丢包率；设计多协议队列，把网络数据包按其不同的网络协议缓存到不同的队列中，然后在多队列中进行轮询重放，从而降低对正常流量的延迟影响；采用多级流表，对处于不同主机的不同类型的流量进行不同的转发处理，实现网络策略的快速实施与转换，保证网络流量的正确转发。

结合软件定义网络与分布式处理两种技术的优势，HostWatcher能显著增加攻击流量的延迟，而保证正常流量的快速正确转发，实现了在为云主机提供全面保护的同时，最大程度的保证服务质量。

2.2 基于CIBPRE的云邮件系统

云邮件系统是部署在云服务器上，以软件即服务(Software-as-a-Service, SaaS)模式向用户提供邮件服务的云计算系统，其具体架构如图2-1所示。

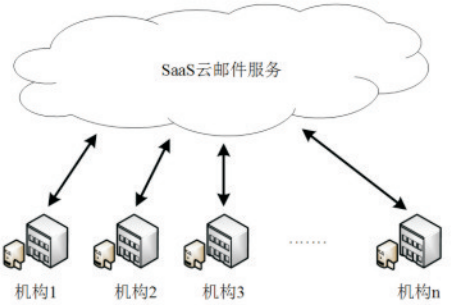


图 2-1 云邮件系统的具体架构

在云邮件系统中，用户租用云邮件服务提供商配置好的服务器来为自己的公司或机构提供邮件服务。在这个过程中，用户只需要架设用于管理的服务器，来配置云邮件系统的基本参数，以及管理用户等。

云邮件系统市场的发展十分迅速，在2017年，Radicati对未来四年中云邮件企业的总营收做了预测，如图2-2所示。可以看到，从2017年开始，所有云邮件厂商的总营收会以接近每年22%的比例高速增长；到2021年，这个数字会增长到43亿美元。可见云邮件市场的火爆。

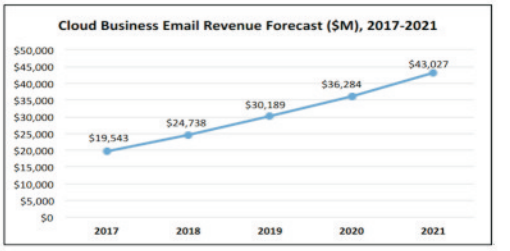


图2-2 Radicati对云邮件企业总营收的预测

随着云邮件系统的发展，其安全问题也越来越受到人们的关注。毕竟通过云邮件系统传输的邮件中包含有公司或机构重要的隐私信息，一旦遭到泄露，造成的损失与影响是巨大的。最近的影响较大的邮件泄露事件就有2016年美国大选期间的“邮件门”事件和2013年沃尔玛集团内部邮件泄漏事件等。这些邮件泄露都对当事方造成了不小的损失。因此，通过加密技术来提高云邮件系统的安全性，并防止邮件内容的外泄是十分重要的。

目前，应用在邮件系统上的加密技术主要有3种，它们分别是SSL协议、PGP协议和IBE协议。

在使用SSL协议发送邮件时，邮件仅在传输链路上是加密的，而在服务器端与客户端中都是以明文形式保存的。可以看出，SSL协议无法保证保存在服务器上的邮件的隐私性。假如服务器遭到攻击，或者服务器本身就是恶意的，那么用户的邮件明文被攻击者获取，导致用户的隐私或者机密信息遭到泄露。因此，就安全性而言，SSL协议并不适合云邮件系统。

而PGP和IBE协议都是公钥加密体制，这意味着攻击者即使获取了用户的加密邮件，在没有私钥的情况下也很难解密出对应的明文。因此，PGP和IBE协议的安全性是非常高的。

但是PGP和IBE协议也有较为严重的问题。首先，用户在群发邮件时，需要使用所有收件人的公钥或ID对要发送的邮件进行多次加密，再将加密后的密文逐一上传至云邮件服务器。在这种情景下，客户端的加密运算与网络通信过程会消耗用户过多的时间，而网络通信过程也会占用带宽，消耗流量，这对移动设备尤其不友好；其次，如果用户想要转发一封已经存在于服务器上的邮件，那么客户端就需要将邮件从服务器上下载下来，将其解密之后，再以上述群发邮件的方式转发给新的收件人。因此PGP与IBE协议的用户体验很糟糕。

从上述介绍可以看出，传统的邮件加密技术无法兼顾安全性和用户使用时的友好性。

为了解决传统邮件系统中加密技术存在的问题，提出了基于CIBPRE加密体制的云邮件系统。CIBPRE加密体制的全称是带条件的基于身份的广播代理重加密(Conditional Identity-Based Broadcast Proxy Re-Encryption)，是徐鹏老师于2016年提出的一种适用于邮件系统的加密体制，它具有三个特点：

1. 基于身份加密，意味着CIBPRE加密体制的运行不需要公钥基础设施（Public Key Infrastructure, PKI）的支持，因为用户的邮件地址就是加密时所需的公钥，只需要密钥生成中心（Key Generation Centre, KGC）为每个用户生成私钥即可。而KGC的运行成本比PKI要低许多，并且客户端在加密时免去了获取公钥证书的开销。因此CIBPRE加密体制运行所需要的成本比较小；

2. 广播加密，意味着用户可以方便地群发邮件。用户在使用广播加密技术群发加密邮件时，只要将明文邮件使用所有收件人的邮箱地址加密一次，产生的定长的密文（定长是指密文总长度不随接收者数量的增长而增长）就可以被所有收件人使用自己的私钥解密；

3. 带条件的代理重加密，意味着用户可以方便地群转发自己保存在云邮件服务器上面的邮件。当用户想要转发一封已经存在于服务器上的邮件给其他一组接收者的时候，他只需要在本地生成一个重加密密钥文件再将其提交给服务器。服务器在收到重加密密钥后就可以对用户指定的邮件进行重加密操作，完成后再将重加密后的邮件转发给用户新指定的一组接收者。新的接收者在收到重加密后的邮件后，便可以使用自己的私钥解密出明文来查阅了。同时，由于使用了带条件的代理重加密技术，云服务器也不可能和某接收者合谋，利用用户的重加密密钥重加密用户的所有邮件，造成用户隐私泄露。

CIBPRE加密体制的核心算法有7个，它们分别为 $Setup_{PRE}$、$Extract_{PRE}$、Enc_{PRE}、$RKExtract_{PRE}$、

$ReEnc_{PRE}$、$Dec\text{-}1_{PRE}$和$Dec\text{-}2_{PRE}$。它们的作用如下：

1. $Setup_{PRE}$：初始化CIBPRE系统并生成系统运行所需要的参数；

2. $Extract_{PRE}$：根据用户ID为用户生成私钥；

3. Enc_{PRE}：对用户的文件进行加密操作，产生的密文可被一组接收者解密；

4. $RKExtract_{PRE}$：生成重加密密钥，用于重加密已有邮件；

5. $ReEnc_{PRE}$：利用重加密密钥对用户的邮件进行重加密操作，产生的重加密邮件可被另一组接收者解密；

6. $Dec\text{-}1_{PRE}$：利用私钥解密Enc_{PRE}算法加密的邮件；

7. $Dec\text{-}2_{PRE}$：利用私钥解密$ReEnc_{PRE}$算法重加密后的邮件。

基于CIBPRE体制的邮件系统运行原理如图2-3所示。其具体的运行步骤为：

1. 密钥生成中心运行$Setup_{PRE}$算法，初始化并生成主公开参数与主秘密参数；

2. 密钥生成中心运行$Extract_{PRE}$算法，为系统中的用户生成私钥；

3. 发送者运行Enc_{PRE}算法，使用第一组接收者的ID加密邮件，并将其上传到邮件服务器；

4. 第一组接收者上线时，从邮件服务器下载加密邮件到本地，并在本地运行$Dec\text{-}1_{PRE}$算法，解密并查看邮件密文；

5. 发送者要将已发送的邮件转发给第二组接收者时，会在本地运行$RKExtract_{PRE}$算法，生成重加密密钥，并上传到云邮件服务器；

6. 云邮件服务器收到发送者的重加密密钥后，利用重加密密钥对指定的邮件运行$ReEnc_{PRE}$算法，并将重加密后的邮件转发给指定的接收者；

7. 第二组接收者上线时，从邮件服务器下载重加密的邮件到本地，并在本地运行$Dec\text{-}2_{PRE}$算法，解密并查看邮件明文。

基于CIBPRE方案的云邮件系统的具体实现中，使用开源软件pbc library作为密码学库来实现CIBPRE密码算法，并使用了OpenSSL中的AES对称加密算法与SHA256哈希算法与CIBPRE密码算法进行组合来共同实现对邮件的加密功能，具体的做法是：使用AES算法加密邮件内容，再用CIBPRE算法加密AES的对称密钥。邮件系统的密钥生成中心使用python与C语言混合编程实现，其中python部分主要实现网络通信功能，而C语言部分实现了CIBPRE的$Setup_{PRE}$和$Extract_{PRE}$算法；客户端修改自开源软件claws-mail，在其中实现了Enc_{PRE}算法、$Dec\text{-}1_{PRE}$算法、$RKExtract_{PRE}$算法和$Dec\text{-}2_{PRE}$算法，并且修改后的客户端具有跨平台能力；邮件服务修改自开源软件iRedMail，并在其中添加了$ReEnc_{PRE}$算法

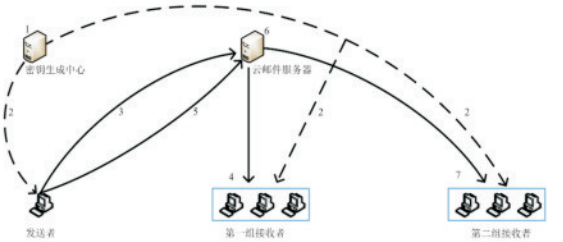


图2-3 基于CIBPRE体制的邮件系统运行步骤

最终的系统实现了图2-3中所有的运行步骤，在保证了用户邮件安全的基础上，保证了用户使用时的友好性。因此相比起传统的邮件加密技术，更适合于云邮件系统使用。

2.3 高密度闪存系统的可靠性优化技术（闪存系统的低密度奇偶校验读性能优化研究）

基于闪存系统的存储介质目前广泛用于手机和个人电脑等终端设备中。虽然闪存系统具有抗震性，非易失性以及高速存取等优势，但高昂的成本是阻碍其广泛应用的一大关键因素。高密度闪存系统（例如：三级存储单元闪存和3D NAND闪存）的出现能有效降低成本，但引起了严重的可靠性问题。为了减轻由技术缩放引起的可靠性问题，低密度奇偶校验码作为一种高级纠错码技术，能为闪存系统提供较高的纠错能力。然而由于闪存错误引起的误码率急剧上升，导致低密度奇偶校验的读性能下

降，极大地削弱了闪存系统的性能优势。如何同时兼顾闪存系统的可靠性和性能，是当前急需解决的一大研究课题。

当前低密度奇偶校验读过程中涉及的高读取延迟主要来自两方面：耗时的闪存采样过程和冗余的读取—重试步骤。低密度奇偶校验读过程中的闪存采样支持软决策采样以应用更多的参考电压来获得较高的纠错能力。参考电压越多则读取等级越高且纠错能力越强，同时引起的读取延迟就越高。因此对具有高错误率数据的读取，为了获得较强的纠错能力而使用高读取等级往往牺牲了读取延迟。当前闪存系统中的高错误率往往在发生在大量磨损的情况下，由于数据保存时间较长而产生较为严重的保留错误。据此提出了缩短数据保存时间以降低读取延迟的轻量级数据刷新方法。对于低密度奇偶校验读过程中引起高读取延迟的第二方面，由于当前页面所需纠错能力未知，无法提前得到适当的读取等级，这使得低密度奇偶校验过程必须从最低的读取等级开始尝试，并且在读取失败时增加读取等级。实际上需要的仅有发生读取成功时的那个等级，其他的读取尝试步骤皆为冗余。为了去除这些冗余延迟，提出了通过利用读取级别的时间局部性来预估需要的读等级和感知延迟的低密度奇偶校验读取方法。

除了上述问题之外，垃圾回收过程中的读性能问题也是当前优化闪存系统性能的一大重要指标。由于在页面迁移过程中写入到新的块之前，受害者块中的有效页面要先通过低密度奇偶校验读取，较高的读取延迟很大程度上影响垃圾回收性能和系统性能。由于垃圾回收中的读取操作不需要向主机返回无差错的数据，因此可以通过直接数据拷贝来迁移页面以避免高读取延迟。然而这种方法引起有效页上的错误累积很容易导致坏块，数据的可靠性无法保证。为了平衡读性能与数据可靠性，提出在垃圾回收中选择性应用低密度奇偶校验读取的方法以避免较高的读取延迟。

针对上述三方面问题提出了如下三个方法来优化闪存读性能：

1）针对闪存采样的高延迟问题，提出了一种基于数据读取特性的轻量级数据刷新方法，通过纠正某些需要高读取级别的页面中的错误以降低其读延迟。该方法主要包含热读数据识别和轻量级页面刷新两个方法。热读数据识别方法应用决策树来决定哪些页面需要被刷新，而轻量级页面刷新方法通过建立多个刷新队列以提高页面刷新的效率。实验结果表明，提出的方法在仅引起平均0.2%的额外编程–擦除循环次数的寿命开销的情况下实现了29%的读性能提升，有效性和轻量性得以验证。

2）针对由迭代读取引起的高延迟问题，提出了具有延迟感知功能的低密度奇偶校验方法来消除当前迭代读取过程中涉及的冗余读取—重试步骤。首先，通过研究读取等级的时间局部性特征估计当前页面所需的读取等级并将其存储在闪存转换层中。作为示例，展示了如何将其集成到基于需求的闪存转换层中。其次，提出了一种新的缓存替换算法以尽可能长地在缓存中保留具有高读取等级的条目，来进一步提升闪存系统的读性能。实验结果表明，与目前的低密度奇偶校验方法相比提出的方法节省了56%的读取—重试延迟并提高了18%的闪存系统读性能。

3）针对垃圾回收中的读性能问题，提出了一种基于可选择低密度奇偶校验读取的垃圾回收方法以实现闪存系统的性能和可靠性之间的平衡。通过选择出具有高错误率的数据块执行低密度奇偶校验读取而对其他块直接读取，可以有效防止高错误率的块变成坏块。由于高错误率块在所有块中占据的比率很低，因此绝大多数块不执行低密度奇偶校验读取，从而保持了低读取延迟的优点。实验结果表明，与传统垃圾回收方法相比，可以减少78% 的读取延迟，实现约2%的系统性能提升。

综上所述，围绕闪存系统的读性能优化问题，通过分析闪存系统所使用的低密度奇偶校验

码读过程，从局部优化和全局优化两个角度提出了上述三个创新点。第一个和第二个创新点侧重于优化闪存采样步骤涉及的延迟和垃圾回收的读延迟，属于局部优化。第三个创新点侧重于优化闪存的整个读取过程，属于全局优化。这些研究创新点对于闪存系统在生产实际中的应用推广有着重要的作用。

2.4 凤链:基于pov共识机制的区块链应用开发平台

区块链是当下非常热门的一项战略性前沿技术，其去中心化、分布式、防篡改的特性可以保证信息的真实性和安全性，并且可以在无需信任的双方之间直接进行可靠交易，降低交易成本，从而产生价值。区块链中最为核心的模块即为共识机制，它是实现去中心化、保障安全性的基础。但是目前区块链主流共识机制无论POW或POS都没有激励价值创造者。因此凤链希望激励真正为社会贡献价值的群体，以激发更多的价值创造，并在此基础进行了一系列的研究。

凤链平台基于区块链技术并采用多链架构，以POV为核心搭建了一个完整的诚信交易生态圈，生态中的各参与方一方面从生态中持续获利；另一方面又为生态做出卓越贡献，共同促进生态的持续发展。凤链作为第一个具有线下应用场景的区块链，以武汉市园博园为起点，POV等诸多技术为支撑，链接众多线下资源，旨在打造诚信、透明、可评价的服务型旅游新模式。

针对传统数据交易存在的安全问题，比如数据源将数据发送给用户，用户缓存到本地之后再转卖给其他用户损害数据源利益，还有数据源窃取用户代码以及数据处理的正确结果损害用户利益等等，凤链提出了一种区块链技术+SGX技术的数据保护体系（如图4-1）。其中，区块链技术作为第三方接收用户的代码与数据源的数据，SGX技术在节点上为数据交易双方提供可信环境：一方面，通过数据加密传输，在SGX创建的安全执行环境中对数据进行处理、以及添加数据输出控制模块，按照输出收费保护数据价值；另一方面通过对智能合约字节码加密、密钥安全管理还有在安全执行环境中解密并执行智能合约保护智能合约的价值。

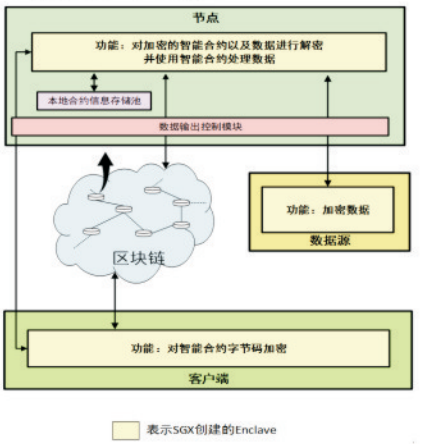


图4-1 基于SGX技术的数据保护体系

不同于中心化系统，区块链上运行的智能合约一旦出错就无法挽回。因此智能合约的安全问题乃为重中之重，而安全最基础、最核心的是访问控制。为了保障智能合约的安全可靠运行，防止恶意调用及潜在的攻击，凤链为智能合约编写者提供了一套安全、可靠、通用的访问控制接口。访问控制框架将访问控制策略存储在凤链上，借助凤链的诸多安全特性防止策略被篡改，抵御恶意或失效节点的攻击，做到可审计、可追溯。

由于具有去中心化，交易过程透明、可追踪、且便于审计等优点，区块链技术在金融领域蓬勃发展。针对目前共识机制存在的资源浪费、趋于中心化等问题，凤链提出了一种基于价值量达成一致的新型共识机制（POV）。它是依据商户利润进行价值评估，并在节点处理交易的时候完成价值激励。商户获得的奖励R可由以下公式得出：其中M为商户的营业额，C为成本，当前奖励系数为∂_i：

$$R=(M-C)*\partial_I$$

奖励系数是会根据额定数量币所消耗的时间进行动态调整的，以此来控制发币速度。在此基础上，凤链独创的基于价值的共识出块（VBBG）将节点分为记账节点和普通节点。记账节点负责维

护全局账本，普通节点不参与共识，但可以看到完整的共识过程，监督记账节点的运行，并能通过报名的方式成为记账节点，大致框架如图4-2。

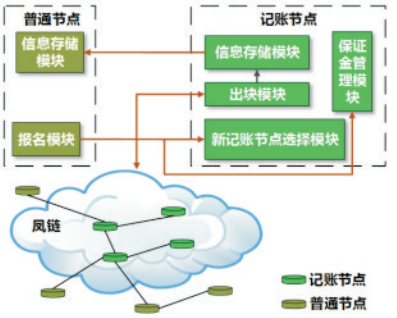


图4-2 一种基于价值量建立共识机制的系统

凤链的另一贡献在于提出了价值分配型合约，它解决了传统的价值分配方法存在的一些问题：合作双方无法完全互信、价值重新分配费时、结算速度慢、多合作方的分配方案不能更精细化控制、分配方案不够公开透明、不能自动化的进行价值分配等等。它通过促进不互信的智能合约间合作共赢，实现精细化按劳分配，达到了提高价值创造效率的目的。

凤链可以在多种应用场景中发挥作用。例如，在园博园景区应用模式中，它可以构建基于景区内部和各相关服务商的立体生态系统和诚信体系，推动旅游共享经济实现更大价值；在AR游戏应用上，玩家获得的奖励情况可查询，做到公平公开，用户行为可追溯、可分析，以及玩家道具等可公平交易，防止欺诈；还可以运用在AR游戏、义务植树等各类场景下。

2.5 Web应用程序的细粒度信息流控制机制

Web应用程序通常会集成来自第三方的代码，并且需要集成在不同站点托管的数据和代码。当前的浏览器缺乏足够安全的机制，来限制不受信任的第三方代码，所以无法保障用户数据的隐私性和应用程序的完整性。该研究通过研究各类Web应用的特点和安全需求，提出了可应用于浏览器的基于标签的细粒度信息流控制模型，兼顾职责分离原则和最小特权原则，能够高效灵活地保障用户数据的隐私性和应用程序的完整性。并且，基于Firefox浏览器实现了一个细粒度信息流控制原型系统JSFfox。

系统总体框架见图5-1。模型引入两种不同粒度的信息流标签，即页面标签和消息标签，并通过跟踪这些动态变化的标签来实现灵活准确的信息流控制策略。页面标签用来关联浏览器组件，代表组件中所有数据的特权，它不仅能够表示机密性和完整性，也能够严格定义授权和降密操作。消息标签用来关联组件间传递的通信消息，表示被标记的消息中包含相关组件的机密信息。模型可以通过消息标签跟踪机密信息，防止其泄露出浏览器。此外，通过证明信息流控制技术的重要属性无干扰性，可以对模型安全性进行分析和验证。 JSFfox系统的标签分发模块，用于为相关浏览器组件分发信息流标签。JSFfox系统提供了两种不同优先级的分发标签方法。系统优先利用开发者通过API 定义的标签信息， 为Web应用开发者提供充足的灵活性和方便性。系统还可以根据应用底层开发者定义的CSP策略的安全信息来自动分发信息流标签，实现安全策略的兼容性。JSFfox 系统的标签检测模块，用于检测相关标签是否满足信息流动的条件。标签检测模块由浏览器内部通信通道标签检测单元、网络通道标签检测单元和浏览器存储通道标签检测单元组成。系统根据跨越组件边界的信息流动的类型，选择相应的标签检测单元根据信息流控制规则允许或者阻止此次消息传递。

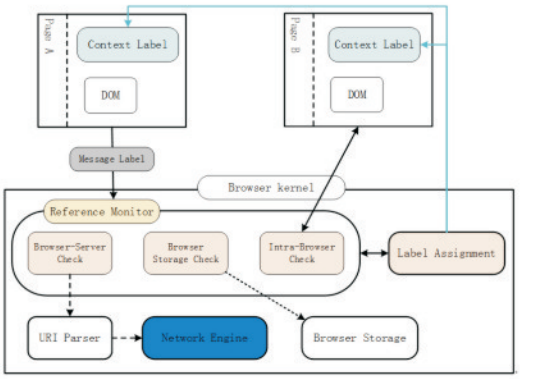


图5-1 系统框架

测试结果表明，在宏观的页面载入延迟中，同等条件下JSFfox系统只增加可忽略不计的性能损耗，并且可以支持大部分流行的网站，兼容性好。在具体典型的Web应用的功能和性能试验中，JSFfox只是增加了微小的性能损耗，就保护了Web应用，防止了隐私泄露，并且能够覆盖绝大部分Web应用。

附成果列表

专利：

- 一种基于系统调用重定向的VMI方法及系统，专利申请号201710248219.7，专利申请日2017年4月14日，金海、羌卫中、徐公平
- 一种基于信息流标签的Web用户隐私保护系统和方法，专利申请号201710323578.4，专利申请日2017年05月10日，金海，羌卫中，郭佳祯，邹德清
- 一种完全上下文敏感的程序控制流完整性保护方法和系统，专利申请号201710321054.1，专利申请日2017年05月09日，金海，羌卫中，黄莹达，邹德清
- 一种基于程序特征树的漏洞检测方法及系统，专利申请号201710242822.4，专利申请日2017年04月14日，金海，邹德清，齐汉超，李珍
- 一种基于攻击过程的漏洞严重度综合评估方法和系统，专利申请号201710243627.3，专利申请日2017年04月14日，金海，邹德清，杨巨，李珍
- 一种基于价值量建立共识机制的方法及其系统,专利申请号201710588193.0，专利申请日2017年07月19日，金海，代炜琦，邹德清，张玮炜，崔长泽（凤链科技）
- 一种基于价值量建立激励机制的方法及其系统，专利申请号201710588192.6，专利申请日2017年07月19日，金海，代炜琦，邹德清，肖德山，崔长泽（凤链科技）
- 一种基于可信环境的智能合约保护方法和系统，专利申请号201710540117.2，专利申请日2017年07月05日，金海，代炜琦，邹德清，代春凯（凤链科技）
- 一种基于区块链的访问控制方法和系统，专利申请号201710540062.5，专利申请日2017年07月05日，金海，代炜琦，邹德清，王晨龙，刘钟泽，柴芳百（凤链科技）
- 一种针对智能合约的价值分配方法和系统，专利申请号201710540049.X，专利申请日2017年07月05日，金海，代炜琦，邹德清，包庆华，李峰（凤链科技）
- 一种基于区块链的旅游景区售票方法和系统，专利申请号201710320678.1，专利申请日2017年05月09日，金海，代炜琦，李峰，邹德清，张舒（凤链科技）
- 一种针对虚拟机回滚的软件补偿方法及系统，专利申请号201710287868.8，专利申请日2017年04月27日，金海，代炜琦，杜玉堃，邹德清
- 面向云环境多租户网络的可信控制器架构及其操作方法， 专利申请号201710273734.0，专利申请日2017年04月25日，金海，代炜琦，万鹏飞，邹德清
- 一种基于硬件虚拟化技术的共享库隔离保护方法及系统，专利申请号201710273274.1，专利申请日2017年04月25日，金海，代炜琦，曹涌，邹德清
- 一种数据存储和共享系统，专利申请号201610990997.9，专利申请日期2016年11月11日，金海，徐鹏，陈天阳
- 一种并行化和结构化公钥可搜索的加密方法，专利申请号201710416111.4，专利申请日期2017年6月6日，金海，徐鹏，唐晓兰
- 一种检测Android权限提升攻击的应用程序重写方法和系统，专利号ZL201410307721.7，专利申请日2014年06月30日，专利授权日2016年10月05日，金海，邹德清，王代斌，徐鹏，羌卫中，陈刚
- Universal serial bus (USB) device access from one or more virtual machines, US 9430424, Aug.30, 2016, Hai Jin, Pei Duan, Deqing Zou

论文：

- Daibin Wang, Haixia Yao, Yingjiu Li, Hai Jin, Deqing Zou, and Robert H. Deng. A Secure, Usable, and Transparent Middleware for Permission Managers on Android. IEEE Transactions on Dependable and Secure Computing, vol. 14, no. 4, pp. 350-362, 2017.
- Bin Yuan, Deqing Zou, Shui Yu, Hai Jin, Weizhong Qiang, Jinan Shen, Defending against Flow Table Overloading Attack in Software-Defined Networks, IEEE Transactions on Services Computing, DOI:10.1109/TSC.2016.2602861.
- Bin Yuan, Deqing Zou, Hai Jin, Shui Yu, Laurence T. Yang. HostWatcher: Protecting Hosts in Cloud Data

Centers through Software-Defined Networking. Future Generation Computer Systems (FGCS).

- Weizhong Qiang, Yuehua Liao, Guozhong Sun, Laurence T. Yang, Deqing Zou, Hai Jin, Patch-related Vulnerability Detection Based on Symbolic Execution, IEEE Access, DOI: 10.1109/ACCESS. 2017.2676161
- Weizhong Qiang, Shifan Xin, Hai Jin, Guozhong Sun, DroidAuditor: A framework for auditing covert communication on Android, Concurrency and Computation: Practice and Experience, DOI: 10.1002/cpe.4205
- Peng Luo, Deqing Deqing Zou, Hai Jin, Yajuan Du and Jinan Shen, "A dynamic predictive race detector for C/C++ programs", Journal of Supercomputing (2017):1-21.
- Zhen Li, Deqing Zou, Shouhuai Xu, Hai Jin, Hanchao Qi, Jie Hu. VulPecker: An Automated Vulnerability Detection System Based on Code Similarity Analysis, ACSAC 2016（TOP 80）.
- Yajuan Du, Deqing Zou, Qiao Li, Liang Shi, Hai Jin, and Chun Jason Xue, LaLDPC: Latency aware LDPC for Read Performance Improvement of Solid State Drives, MSST 2017（TOP 80）.
- Yajuan Du, Qiao Li, Liang Shi, Deqing Zou, Hai Jin, and Chun Jason Xue, Reducing LDPC Soft Sensing Latency by Lightweight Data Refresh for Flash Read Performance Improvement, DAC 2017（CCF B类）.
- Peng Xu, Shuai Liang, Wei Wang, Willy Susilo, Qianhong Wu, Hai Jin, Dynamic Searchable Symmetric Encryption with Physical Deletion and Small Leakage, ACISP 2017, 207-226.
- Weizhong Qiang, Jiazhen Guo, Hai Jin, and Weifeng Li, "JSFfox: Run-timely Conning JavaScript for Firefox", Proceedings of the 22nd Australasian Conference on Information Security and Privacy (ACISP 2017), pp 135-150.
- Weizhong Qiang, Yingda Huang, Deqing Zou, Hai Jin, Shizhen Wang, Guozhong Sun, Fully Context-Sensitive CFI for COTS Binaries. ACISP (2) 2017: 435-442.
- Weizhong Qiang, Kang Zhang, Hai Jin, Reducing TCB of Linux Kernel Using User-Space Device Driver. ICA3PP 2016: 572-585.
- Weizhong Qiang, Gongping Xu, Guozhong Sun, Tianqing Zhu, Hai Jin, CloudController: A Writable and Heterogeneous-Adaptive Virtual Machine Introspection for Cloud Management, IEEE TrustCom 2017.
- Zhikun Chen, Weizhong Qiang, ISLUS: An Immediate and Safe Live Update System for C Program, IEEE DSC 2017.
- Deqing Zou, Hanchao Qi, Zhen Li, Song Wu, Hai Jin, Guozhong Sun, Sujuan Wang, and Yuyi Zhong. SCVD: A New Semantics-Based Approach for Cloned Vulnerable Code Detection. DIMVA 2017, 325-344.
- Yajuan Du, Yunpei Jia, Meng Zheng, Jun Zeng, and Chun Jason Xue, "Enhancing SSD Performance with LDPC-aware Garbage Collection", In Proceedings of the 6th IEEE Non-Volatile Memory Systems and Applications Symposium, 2017. yuanbin@hust.edu.cn

袁　斌

博士生

研究方向：云安全、SDN安全

Email：yuanbin@hust.edu.cn

杜亚娟

博士生

研究方向：存储系统的可靠性和性能优化纠错码理论和算法

Email：dyjxtu5@126.com

代炜琦

讲师

研究方向：云安全、可信计算、虚拟化安全、可信SDN等算

Email：wqdai@hust.edu.cn

徐　鹏

副教授

研究方向：密码学、云数据安全

Email：xupeng@mail.hust.edu.cn

羌卫中

副教授

研究方向：系统安全、软件安全

Email：wzqiang@hust.edu.cn

大数据组典型成果介绍

华强胜，于东晓，张 凡，廖良翌，刘 伟，石宣化

关键词：低复杂度算法，高速缓冲系统，内存计算，流计算

1 介绍

大数据具有多样性、快速性和大规模等特点，传统算法与系统面对大数据的复杂应用场景具有很大挑战性。本研究小组旨在研究大数据的存储、检索、分析和挖掘等技术，为大数据的广泛高效应用提供技术支持。基于本组所承担的“面向大数据的高时效并行计算机系统结构与技术”和电力、航空运输等应用处理平台项目，对不同领域大数据问题进行深入研究，从算法、系统等多个维度开展研究，研发了一系列的典型系统、探讨了多个典型大数据领域数据处理问题解决思路：

1）在大数据计算理论方面，针对当前大规模图随机游走中心度顺序计算复杂度高问题，设计了首个线性时间分布式算法，并且通过通信复杂性理论证明该问题分布式计算下界也为线性，从而说明所设计分布式算法为近似最优。此外，针对当前动态图中k-core维护问题，提出了一种可以在多核系统中并行处理的高效算法，实验结果表明所提并行算法比顺序算法快200倍。

2）在大数据计算平台的IO优化方面，针对当前突发性IO写缓冲方案不能合理利用SSD资源的问题，研发了基于负载感知的混合写缓冲系统SSDUP，一方面通过设计新的随机请求识别模型来区分请求类型以充分利用SSD，另一方面使用流水线形式管理数据缓冲和刷新过程，减少系统对计算时间的依赖。SSDUP能充分利用SSD资源，对现有的大数据计算平台的IO写性能有极大提升，具有较大的实用价值。

3）在大数据分布式处理系统的运行时内存管理方面，分布式处理系统中的任务在运行过程中会产生大量持久存活的数据对象，但由于分布式处理系统的内存资源限制，从而导致大量的GC操作严重影响系统执行效率。针对此问题，分析不同的算子API各自的内存使用模型，及其对系统整体内存压力造成的影响。基于内存使用率设计出MURS调度器来调度运行任务以减轻内存压力。该调度器能够大幅减少系统的GC操作，提升了系统的计算性能，具有较大的实际应用价值。

4）在实时流数据处理方面，针对实际大数据应用中，流数据所具有的实时性强、倾斜性高、动态变化大等特征，给分布式处理系统带来了处理时空开销高、吞吐率受限、难以扩展等严重问题，研发了分布式数据区分处理系统DStream，和动态流数据集合管理工具—动态Cuckoo滤波器。此方面的研究显著提升了流数据处理的时效性和吞吐率。

2 低复杂度图处理算法

图作为一种建立在图理论上的抽象数据结构，在实际生活中有非常多的应用。随着图应用的盛行，图数据处理和图性质分析已经成为一个重要的研究领域。由于大规模图数据量大（可达几百亿个节点、几万亿条边）和数据变化快，而以往图处理的方法时间复杂度高（达到平方级甚至立方级时间），急需设计图处理

和图分析的低复杂度算法。本组从分布式算法和并行算法角度出发，得到静态图计算和动态图性质维护的低复杂度算法。

2.1 静态图中心度计算分布式算法

2.1.1 研究问题

通常我们将网络描述为一个图G =(V , E)。其中集合V代表顶点集合，而集合E则代表顶点与顶点之间相连边的集合。使用N和M分别表示顶点数和边数。简单起见，假定所有的图都无向且连通。

为了量化图中节点的相对重要程度，中心度作为一种重要参数被广泛应用在图分析中。迄今为止，已经提出了许多不同种类的中心度，例如紧密中心度(closeness centrality)和介数中心度(betweenness centrality)等。

一个点的介数中心度代表其对网络中信息传播速度的影响程度，它常被定义为经过该点的最短路径数与所有最短路径数之比。一个点的紧密中心度表示其在网络中的重要程度，它常被定义为其到其它所有点距离之和的倒数。然而，网络中信息不仅仅通过最短路径传播，同时也能从任意路径传播，从而提出基于“随机游走”的介数中心度和紧密中心度（简称为随机游走介数中心度和随机游走紧密中心度）。给定一条消息，在一轮中它从网络上一个点任意移动到一个相邻的点上，这被称为“随机游走”。给出两个点s,t ∈ V，一个随机游走从点s出发到达点t，此“随机游走”经过的路径被称为一条“随机游走”路径（每条“随机游走”路径都有一定概率），将从s到t的随机游走定义为从点s出发到达点t的所有“随机游走”路径。一个点的随机游走介数中心度表示该点被所有其它点对间的随机游走所访问的期望次数。一个点的随机游走紧密中心度表示该点到其它所有点的平均随机游走路径长度之和的倒数。

我们主要研究如何在拥塞模型(CONGEST model) 下计算中心度问题。拥塞模型是分布式计算中广泛采用的一种消息传递模型，在该模型下，每个节点每一轮只能向它的邻居发送O(log N)比特的消息。

2.1.2 近似最优分布式算法

从定义看出，基于随机游走的两种中心度完全是基于随机游走，因此一种简单的方法是利用矩阵运算求出相应的概率，之后计算节点的随机游走中心度。但是，这种方法的时间复杂度达到O(M)，即$O(N^2)$，耗时过高。因此需要提出时间复杂度更低的求解两种随机游走中心度的分布式算法。

由于按照矩阵运算的方法无法达到时间复杂度的要求，我们从定义出发，通过分析随机游走介数中心度定义的性质，提出了一个时间复杂度为O(NlogN)、近似比为(1-ε)（其中 ε 为一任意小常数）的分布式算法。我们发现：当每个点上的随机游走个数为O(logN)，随机游走长度为O(N)时，此时一个点的随机游走介数中心度可近似视为所有随机游走经过其次数之和。通过这一发现，我们提出如下算法：初始时使图中每个点拥有O(logN)个随机游走，每个随机游走的长度设置为O(N)；选定图中任意一个节点t，当随机游走到达t或者其走过的路径等于其长度的时候停止；在每一轮中，一个点将一个随机游走通过与它相连的任意一条边发送出去，直到不再有点发送随机游走时算法结束；每个点通过随机游走经过其的次数计算出该点的随机游走介数中心度。由于图中一共有O(NlogN)个随机游走，因此该算法时间复杂度为O(NlogN)。一个点的随机游走紧密中心度也可以被视为所有随机游走经过其次数之和的表达式。因此上述算法同样可以被使用在求解随机游走紧密中心度上（细节有所不同），从而求解一个点的随机游走紧密中心度的时间复杂

度为O(NlogN)。

2.2 动态图核数维护并行算法

2.2.1 研究问题

图分析另一个重要的课题是确认图中的紧密子图。很多指标用于刻画顶点所属的紧密子图，如Clique、k-truss、k-group、k-core等。其中核数（k-core）因其计算复杂度较低，是最为常用的指标。

具体地，给定图G=(V, E)，其中V表示顶点集合，E表示图中顶点之间的边集。图中顶点v的核数C(v)定义为点v所在的最小度最大的子图的最小度。具体地，令H_v是包含v的最小度最大的子图，$\delta(H_v)$表示H_v的最小度。顶点v的核数$C(v)=\delta(H_v)$。

大规模图中，由于图数据的规模较大，在图发生变化以后，尽管计算核数的成本为线性时间，重新计算顶点核数成本仍极高。与之相对的，图数据的变化往往只会改变图中少数节点的核数。因此，动态更新节点核数，而不是重新计算，是更为经济适用的方式。

具体地，动态图核数维护问题定义如下：在图G中添加或者删除顶点集合V'和边集E'，快速更新所得新图G'中顶点的核数。

2.2.2 核数维护并行算法

此前的工作主要考虑了图中单条边、单个顶点变动对于顶点核数维护的影响。如果在图中插入或者删除多条边、多个顶点时，只能通过单个元素变动的线性调整方式来完成，严重影响了核数更新的速度。而图中元素变动（边和顶点的增加和删除）对于核数的影响具有局部性，大规模图中多个元素的变动可能互不影响，可并行处理。此外，线性维护顶点核数未能充分利用现今多核处理系统和分布式系统带来的计算能力。因此，为能在多核处理系统和分布式系统中快速更新顶点核数，应充分挖掘顶点核数更新过程中的并行性，设计并行算法高效处理核数维护问题。

多边更新的k-core 维护问题主要包含以下两个难点，一方面要找到核数发生变化的顶点集合，另一方面要计算出这些顶点的核数变化值并进行更新。为了降低计算复杂度，我们在原有单边处理方法的基础上，提出了superior edge tree 结构，该结构的增删可以保证图中所有顶点的核数增减不大于1。因此在维护过程中只需要判断哪些顶点的核数会发生变化，将原有的增量计算问题转变为0/1问题。进一步地，我们充分利用现有多核系统的并行计算特点以及顶点核数发生变化的规律，将查找核数发生变化的顶点的过程并行化处理。首先我们定义出一条边的核值，并且证明了：核数为k的边的插入和删除只会影响核数为k的顶点核值发生改变。因此我们将superior edge tree结构中的边根据核值划分给不同线程，多个线程可以同时进行计算，进一步提高了计算效率。每个线程执行任务的复杂度和原有单边计算的复杂度相同，但是处理边的数量大大增加。最终实验结果表明，在Intel 双核16线程的服务器上，对于不同的数据集，我们基于superior edge tree 结构的并行处理算法比单边处理算法均能快200倍以上。

3 高性能计算环境下基于负载感知的混合写缓冲系统SSDUP

当高性能计算朝着百亿亿次级发展时，I/O性能仍然是一个主要的瓶颈，尤其是在数据密集型的科学应用中体现得非常明显。科学应用以数据模拟操作为主，其访问特征是阶段性地向并行文件系统输出大量中间结果，这种周期性的写请求会给并行文件系统带来突发性的写性能瓶颈。与此同时，高性能计算集群仍然以机械硬盘（HDD）作为主要存储介质。新型存储设备固态硬盘（SSD）由于其对随机访问不敏感等特性，已经被广泛应用在高性能计算环

境当中。当前大多数的大规模高性能计算集群，使用SSD与HDD结合的混合存储系统，将SSD作为计算节点与HDD之间的高速缓冲，如图3.1所示。以突发性写缓冲（Burst Buffer）为代表的混合存储系统需要使用大量的SSD来缓冲所有的中间数据，并且依赖难以预测的计算阶段进行数据刷新，系统的建设成本仍然很高。

为解决突发性写缓冲存在的缺陷，提出基于负载感知的混合写缓冲系统SSDUP。SSDUP通过分析不同的科学应用访问模式在不同负载情况下引起随机访问的现象及原因，设计一种服务端负载感知的随机识别方法，仅在SSD中缓冲随机请求，顺序请求则被定位到HDD中。SSDUP采用流水线方式管理数据的缓冲与刷新过程，从而减少对计算时间的依赖。SSDUP还使用AVL树管理SSD中的缓存数据，保证数据被顺序写回HDD时的开销最小。实验结果表明，SSDUP相比于突发性写缓冲系统，能够使用更少的SSD空间，并将并行I/O的写性能提升50%以上。

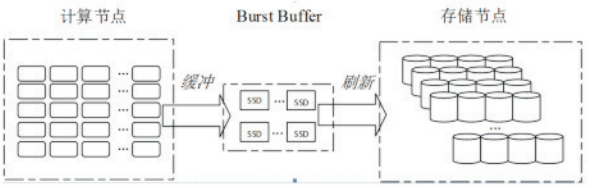


图3.1 一种突发性写缓冲系统架构的实现

3.1 问题分析及解决思路

3.1.1 目前突发性写缓冲存在的问题

以突发性写缓冲为思想构造的系统（以下简称为突发性写缓冲）通常需要保证SSD的容量能足够容纳计算阶段产生的所有中间数据，而这意味着突发性写缓冲所需要的SSD容量要能够容纳应用产生的最大数据量。然而随着高性能领域的应用朝着数据密集型的趋势发展，一个单独的应用所产生的数据量就能达到数百TB甚至数十PB的级别，例如在美国的Mira超级计算机集群上运行的Earth1应用传输的总数据量达到了10PB。不仅如此，突发性写缓冲并不提供数据甄别的功能，将数据无差别地全部写到SSD缓冲中，同样会让SSD的容量很快地消耗殆尽。另一方面，当前的高性能计算集群作业调度器未考虑I/O资源的使用情况，可能会导致新提交的作业I/O性能依然受制于HDD。

3.1.2 不同访问模式对系统性能的影响

随机访问在机械硬盘中会引起巨大的访问延迟。高性能计算领域科学应用的I/O行为和性能会受到很多因素的影响，例如进程访问数据的方式，每一个I/O调用的请求大小，参与I/O的进程数目，数据服务器的数目，Unix内核的I/O子系统等等。所有的这些因素均可能导致严重的随机访问问题。

3.2 基于负载感知的混合写缓冲

针对系统建设成本过高，依赖不确定的计算时间以及现有随机访问判别方法的局限性，SSDUP提出一种基于负载感知的写缓冲系统，目标是使用成本更低的方案，即更少的SSD容量来达到良好的写性能。整体的系统架构与功能模块如图3.2所示。

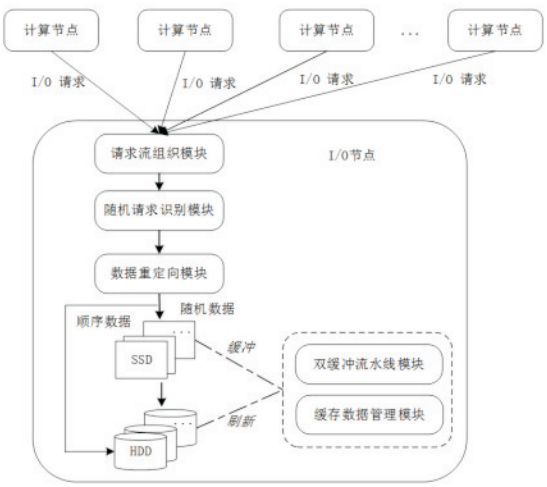


图3.2 系统架构与功能模块

3.2.1 写请求流组织与负载感知的随机性评估

本系统通过一段写请求而不是单个写请求来判断随机性。当不同进程的写请求到达I/O节点后，会被记录元数据信息并按照到达顺序组织成“写请求流”，且后续模块对请求的处理都是以“写请求流为单位”。通过三种访问模式的访问踪迹（trace）来分析不同访问模式在服务端的寻址表现。经过逻辑地址排序，我们发现了各种访问模式的地址分布变得有一定的规律，通过这种规律，我们定义访问模式的随机因子来定量评估写请求流地址分布的随机性。

3.2.2 写请求流的重定向机制

“写请求流”重定向模块主要是根据“写请求流”的随机性将写请求定向到不同的设备，以充分发挥SSD和HDD的优势。由于数据的流向会以某一个状态维持一段时间，因此数据的重定向过程还需要考虑当前的数据流向，以避免数据流向的频繁变化，保护数据的空间局部性，例如只有足够的随机数据被缓冲到SSD中时，其刷新到HDD时的空间局部性和性能才会更好。

3.2.3 双缓冲流水线机制与缓存元数据管理

为了减少系统对计算时间的依赖，同时利用更少的SSD空间提升系统性能，SSDUP系统使用了流水线技术将缓冲与刷新过程并行化。首先，双缓冲流水线模块将SSD空间划分为两个缓冲区，每个区初始都为空且大小相同。在请求需要被缓冲时，SSDUP首先挑选一块空缓冲区来缓冲随机请求，当一块填满之后，开始挑选第二块缓冲区进行写操作，与此同时，后台开启刷新进程，将已填满的缓冲区中的数据刷到HDD中，以此来保证总有空余的空间来缓存随机请求。在缓存数据过程中，由于系统采用追加写的方式，原来数据顺序被打乱，因此需要采用AVL树来管理缓存数据。两个缓冲区分别对应两颗AVL树，分别负责记录元数据和为刷新SSD提供元数据。AVL树在数据开始缓存时被建立，并在缓存过程中进行插入和调整。在数据刷新时，AVL树提供刷新过程所需要的元数据，并在刷新过程结束后清空所有元数据。

4 MURS:减轻面向服务的数据处理系统的内存压力

以MapReduce为计算模型的分布式数据处理系统的诞生为业界处理大型数据提供了很大的便利，其中基于内存计算的分布式系统如Spark，Flink可以通过减少中间数据的磁盘IO操作来提升数据处理和计算的速度，在迭代计算领域很有优势。然而这也带来一些弊端，内存的限制和压力会给用户在计算平台上提交的job带来很大的性能影响。内存的压力来自于正在运行的任务在有限的内存容量中产生了大量持久存活的数据对象，导致GC变得非常频繁。我们发现每个任务都会包含这个分布式框架所提供的算子API，大部分算子API都会基于特定的模型来产生持久的数据对象，一些需要恒定大小的内存空间，一些需要线性增长的内存空间。不同的模型会对内存压力造成不同的影响，我们提出了内存使用率这个概念用于将一个任务划分到不同的模型中。基于内存使用率我们设计了MURS调度器来调度所有运行任务以减轻内存压力。最终的效果与Spark比较提交Job的运行时间减少至65.8%；将垃圾回收时间减少至81%；避免了91%spill任务的磁盘IO操作。

4.1 任务的内存使用模型

一些任务使用少量的内存就可以执行完成，但是有的任务需要花费大量的内存空间去产生持久数据对象。其中一个主要原因就是处理管道中的算子API，我们建立了模型描述这些算子API的内存使用特性，并且依靠内存使用率决定一个任务属于何种模型。

4.1.1 API的内存使用模型

分布式框架所提供的算子API都是基于KV对操作的，必要的内存空间是为了算子API存储存活的结果数据对象，因为临时的数据对象很快会被内存管理系统回收。因此，我们基于算子API处理单元数据的内存使用规律建立了四种API内存使用模型，模型是基于处理数据的大小，而不是数据记录的个数（KV对的数量）。四种内存使用模型和满足的条件总结如下：

（1）Constant：算子API不区分K，结果数据不会在内存中缓存。

（2）Sub_Linear：算子API区分K，算子API会聚合V，K在输入数据集中随机出现。K值相同，shuffle buffer大小不增加，K值不同，shuffle buffer大小会增加。

（3）Linear：算子API区分K，算子API不聚合V。

（4）Super-Linear：对相同K值的V做非聚合操作，每处理一个KV，shuffle buffer大小会增加

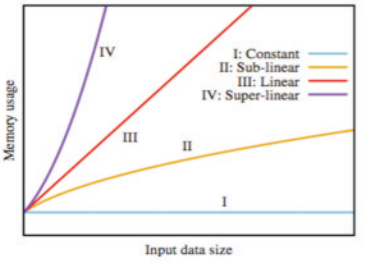


图4.1 算子API的四种内存使用模型

如图4.1所示，算子API的四种内存使用模型都对应着图中不同的斜率，斜率越大，说明造成的内存压力更严重。

4.1.2 任务的内存使用模型

绝大多数分布式处理系统提供多种算子API，例如Spark和Dryad，通常一个任务由若干个算子API构成。两个shuffle操作之间形成的任务大致分为三个阶段，读入，处理和写出。一个任务的读入和写出部分有着独立的内存使用模型，剩下的处理部分不同的任务会有不同的内存使用模型，但这部分操作总是Constant模型，他们从不区分K，且不需要计算所有临时数据。我们定义任务的内存使用模型的依据为：当一个任务处理单位大小的输入数据时，所产生的持久数据对象所使用的内存空间大小。

4.2 MURS调度策略

对象的拆解已经解决了内存膨胀的问题，并且极大的简化了对象的模型，这与JVM原生的内存管理机制有较大差异，因此需要更合适的内存管理机制。

数据对象在JVM中都具有自己的生命周期，我们发现：在以Spark为代表的新一代通用数据并行系统中，作业执行时通常有几类数据容器会持有数据对象，而数据对象的生命周期和持有它的数据容器本身的生命周期由很强的关联性：

策略的根本机制在于当内存压力出现时，暂停重型任务，等到轻量型任务完成时或者内存压力减轻时，再恢复暂停的任务。

MURS定义了两个阈值，黄色值代表内存压力，当内存堆中的占用空间比值达到黄色值时，代表full GC将要发生。还有一个红色值，用来避免spill操作，达到红色值意味着可能会发生OOM或者会发生spill操作。MURS设计了一个记录当前运行任务监控信息的采样器，以周期性更新当前内存使用率、处理数据记录数和结果数据大小等任务信息。当内存压力达到红色阈值，MURS会进行避免spill的操作。当达到黄色阈值且没有超过红色阈值，会根据内存使用模型按照constant，sub-linear，linear，和super-linear的顺序处理。

其中，重型任务（heavy-task）包括super-linear，linear，或者处理特别大数据的非constant模型的任务。通过MURS计算暂停重型任务的算法来减少当前的内存压力，当内存压

力降到阈值以下时再逐步释放暂停的任务。经过MURS的调度策略能通过将任务的执行划分到不同的时间段来大大减少full Gc的操作，同时削弱内存的压力，以此提高数据处理的效率。

5 高时效分布式流处理系统

实际大数据应用中，数据往往以实时流数据的形式呈现，其具有的实时性强、倾斜性高、动态变化大等特征，给分布式处理系统带来了处理时空开销高、吞吐率受限、难以扩展等严重问题。围绕大规模流数据的高时效处理，研发了分布式数据区分处理系统DStream（http://github.com/CGCL-codes/DStream），和流数据动态集合管理工具—动态Cuckoo滤波器（http://github.com/CGCL-codes/DCF）。这方面的研究显著提升了流数据处理的时效性和吞吐率。

5.1 分布式流数据区分处理系统DStream

为了对实时海量的流数据进行高效的处理，现有的分布式流处理系统往往采用流水线并行的方式，将复杂的流应用抽象为若干简单的处理阶段，分别由分布式集群中不同的处理单元进行执行。数据在不同的处理单元间以流水线的方式被快速地处理和传递。为了进一步提升系统的吞吐率，各处理单元可以生成多个相同的实例，以将数据划分进行高效并行的处理。现有的分布式流处理系统在对数据进行划分时，常采用轮询或哈希等一刀切的策略。哈希策略根据数据的不同键值进行划分，然而在面向高倾斜分布的数据时，哈希策略会导致各个实例间严重的负载不均问题，尤其是对热点数据的处理成为系统的瓶颈，造成系统计算资源的浪费，降低系统的吞吐率；轮询策略保证了数据划分的均衡性，然而产生线性的存储开销，使得系统面临扩展性问题。

此项研究指出高效的分布式流处理的关键是能够对不同热度的数据进行区分处理。为此，研发了分布式流数据区分处理系统DStream。DStream对热点数据项使用轮询调度方法来保证系统的负载均衡；对非热点数据项采用哈希的调度方法，以避免不必要的空间开销。为了进一步解决热点流数据项实时动态变化难以预测的问题，DStream设计了一种轻量级基于概率计数的技术，高效的对流数据热度进行统计和热门数据项识别。相对于目前最新流处理系统，DStream将系统总体吞吐率提升至2.3倍，将平均处理延时降低了64%。

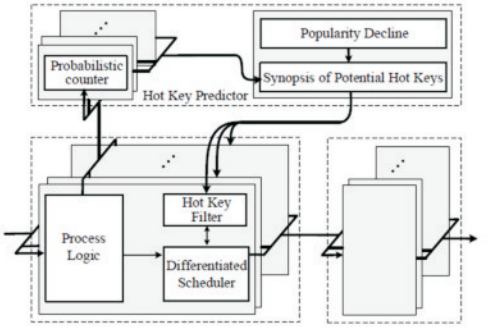


图5.1 Dstream系统架构

图5.1显示了DStream的系统架构。DStream的架构主要分为两层：预测器和调度器。预测器从每个处理单元实例获取当前处理数据，时空高效地识别出热门数据项并进行存储和更新。调度器嵌入每个流处理实例，并根据预测器的信息进行调度策略的选择。

5.2 流数据动态集管理工具—动态Cuckoo滤波器

流数据表现出的连续和动态性特征，为实际大数据应用中的数据集表示和管理带来了严重的挑战。例如，云存储系统中需要利用近似集合成员判定技术进行数据的轻量快速去重操作。在面向大规模动态变化的流式数据中，数据集合大小随着时间不断发生动态变化，并且存在数据的插入、删除等多种操作。现有的近似集合成员判定技术只能单一的实现数据集合存储结构动态扩展或集合元素可靠删除等功能，并且在时间和空间上都做出了较大牺牲，难以处理持续到来的大规模流数据，不具可扩展性。

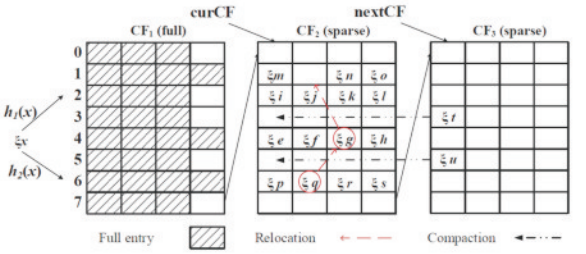


图5.2 动态可扩展Cuckoo滤波器

本研究设计了一种基于指纹判定和重定位技术的动态可扩展数据结构—动态Cuckoo滤波器（DCF），对大规模数据集合进行组织和存储，并且设计了一套高效的算法对已存储的数据集合进行插入，查询，删除和空间整理操作。DCF通过动态增加building block实现了存储结构动态扩展，通过基于“The power of two choice”思想的重定位技术解决了哈希碰撞带来的低空间使用率问题，实现了高效的流处理系统动态数据集合组织和存储，而使用独占的指纹来标识数据的存在性则实现了集合元素的可靠删除操作。结合随机数据测试和真实数据集测试，证明了DCF可以高效地支持数据结构容量动态伸缩以及可靠删除操作，相比于当前的DBF方法减少了75%的空间开销，并且在集合元素插入和近似集合成员判定操作上速度提高了50%~80%。

图5.2给出了动态Cuckoo滤波器基本操作示意图。初始情况下，DCF由一个building block CF_1组成，当CF_1存储的集合成员达到容量之后，一个新的building block CF_2将会被分配并且链接在DCF尾部。DCF使用精简的指纹来代替原始数据进行集合成员的表示，为了提高DCF结构的空间使用率，DCF为每个指纹分配两个候选的插入位置。当指纹的两个候选位置都满的情况下，将会触发重定位操作。重定位操作通过随机选择并移动已存储的指纹到另一个候选位置中，为新插入的元素腾出存储槽位。如图5.2所示，由于x对应的候选位置2，6已满，插入x的指纹将会触发元素q和元素g的指纹重定位到各自另一个候选位置中，从而为x的指纹腾出存储槽位。元素的查询和删除操作则相应的需要遍历每个building block并且到相应的候选位置中做指纹匹配和删除操作。通过基于贪心算法的指纹整理操作，DCF将较稀疏building block中的指纹搬移到其它building block对应的位置中，并且释放空building block，以提升DCF空间效率。

附成果列表：

开源系统

- 高性能计算机高速缓冲系统SSDUP：https://github.com/CGCL-codes/SSDUP
- Spark数据Cache替换策略LCS：https://github.com/CGCL-codes/LCS
- 深度学习系统TensorFlow-RDMA：https://github.com/CGCL-codes/Tensorflow-RDMA
- 内存计算GC压力缓解调度器MURS：https://github.com/CGCL-codes/MURS
- 分布式大规模动态图K核维护方法ParaCoM：https://github.com/CGCL-codes/ParaCoM
- 分布式流数据区分处理系统DStream, http://github.com/CGCL-codes/DStream, 2017
- 流数据集管理工具包—动态Cuckoo滤波器, http://github.com/CGCL-codes/DCF, 2017

专利：

- 一种基于Storm实时流计算框架的消息可靠处理保障方法，专利号ZL201310682070.5，专利授权日2016年10月5日，谢夏，金海，胡亚军，柯西江
- 一种基于四维索引的大规模图的可达查询方法和系统，专利号:201710366030.8,专利授权日2017年05月23日 袁平鹏，金海，周双
- 一种基于簇的图数据划分方法，专利号：201710319629.6，专利授权日2017年5月9日，袁平鹏，金海，龙浩
- 一种基于Skyline的数据泛化方法，专利号：201710339575.X，专利申请日2017年05月15日，丁晓峰，金海，王丽
- 一种基于多线程的MapReduce执行系统，专利号：ZL201310602222.6，专利授权日2017年2月8日，石宣化，金海，陈明，吴松，陆路
- 一种隐私保护的随机遍历方法和系统，专利申

请号201710290787.3，专利申请日2017年04月28日，丁晓峰，金海，刘朋

- 一种支持差分隐私的频繁项集挖掘方法和系统，专利申请号201710273748.2，专利申请日2017年04月25日，丁晓峰，金海，陈龙
- 一种高效的分布式大规模动态图K核维护方法，专利申请号201610837582.8，专利申请日2016年09月21日，金海，王娜，谢夏，钱辰，柯西江，付煜
- 一种基于多级缓存结构的流式数据实时处理方法及系统，201710176167.7，专利申请日2017年03月23日，赵峰，李少锋，金海，肖洋
- 一种随机访问识别方法及系统，专利申请号201710281734.5，专利申请日2017年04月26日，石宣化，金海，黎明，刘伟
- 分布式数据处理系统中缓解内存压力的调度方法和系统，专利申请号201710273273.7，专利申请日2017年04月25日，石宣化，金海，张雄，柯志祥
- 一种基于多GPU的图数据处理系统及方法，专利申请号201710276951.5，专利申请日2017年04月25日，石宣化，金海，罗璇，赵鹏
- 一种解决大数据处理系统中内存资源激烈竞争的系统，专利申请号201610189095.5，专利申请日2016年03月31日，石宣化，金海，耿元振，王斐
- 一种大规模RDF图的Theta Join查询处理方法，专利申请号201610247087.1 专利申请日2016年04月20日，袁平鹏，金海，王涛
- 一种基于团的大规模图最短距离索引方法，专利申请号201610188131.6专利申请日2016年03月29日，谢夏，李沛洋，金海
- 一种社交网络用户关系搜索方法，专利申请号201610257499.3，专利申请日2016年04月25日，谢夏，杨晓冬，金海，王多强
- 一种众包数据库下的双向K-匿名方法，专利申请号201510611209.6，专利申请日2015年09月23日，丁晓峰，金海，张凡

论文：

- Xuanhua Shi, Ming Li, Wei Liu, Hai Jin, Chen Yu, and Yong Chen, "SSDUP: A Traffic-Aware SSD Burst Buffer for HPC Systems". in Proceedings of the ACM International Conference on Supercomputing (ICS), Chicago, Illinois, USA, June 13-16, 2017.
- Xuanhua Shi, Xiong Zhang, Ligang He, Hai Jin, Zhixiang Ke, and Song Wu, "MURS: Mitigating Memory Pressure In Service-Oriented Data Processing Systems". in Proceedings of the 24th IEEE International Conference on Web Services (ICWS), Honolulu, Hawaii, USA, June 25-30, 2017.
- Qiang-Sheng Hua, Ming Ai, Hai Jin, Dongxiao Yu, Xuanhua Shi, Distributively Computing Random Walk Betweenness Centrality in Linear Time. The 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017), June 5-8, 2017, Atlanta, GA, USA.
- Na Wang, Dongxiao Yu, Hai Jin, Chen Qian, Xia Xie, Qiang-Sheng Hua, Parallel Algorithms for Core Maintenance in Dynamic Graphs. The 37th IEEE International Conference on Distributed Computing Systems (ICDCS 2017), June 5-8, 2017, Atlanta, GA, USA. (Short paper)
- Hanhua Chen, Hai Jin, Xiaolong Cui, Hybrid followee recommendation in microblogging systems. SCIENCE CHINA Information Sciences 60(1): 12102, 2017
- Yong Chen, Juncheng Yao, Hai Jin, Chunjiang He, Hanhua Chen, Exploring the Evolution of New Mobile Services. Scientific Programming 2017: 5159690:1-5159690:9, 2017
- Yong Chen, Juncheng Yao, Chunjiang He, Hanhua Chen, Hai Jin, Adaptive Traffic Signal Control with Network-Wide Coordination. ICA3PP 2017: 180-194, 2017
- Dongxiao Yu, Li Ning, Yifei Zou, Jiguo Yu, Xiuzhen Cheng, Francis C. M. Lau. Distributed Spanner Construction With Physical Interference: Constant Stretch and Linear Sparseness. IEEE/ACM Transactions on Networking(ToN), 2017.
- Lu Lu, Xuanhua Shi, Yongluan Zhou, Xiong Zhang, Hai Jin, Cheng Pei, Ligang He, Yuanzhen Geng. Lifetime-Based Memory Management for Distributed Data Processing Systems. PVLDB9(12):936-947(2016)
- Dongxiao Yu, Yuexuan Wang, Tigran Tonoyan, Magnús M. Halldórsson. Dynamic Adaptation in Wireless Networks Under Comprehensive Interference via Carrier Sense. IPDPS 2017: 337-346
- Fan Zhang, Hanhua Chen, Hai Jin. Piggyback game: Efficient event stream dissemination in Online Social Network systems. ICNP, 2016
- Ming Li, Xuanhua Shi, Wei Liu, Hai Jin, Yong Chen. SSDUP: An Efficient SSD Write Buffer Using Pipeline. CLUSTER 2016:166-167
- Dongxiao Yu, Li Ning, Yong Zhang, Hai Jin, Yuexuan Wang, Francis C. M. Lau, Shengzhong Feng. Uniform Information Exchange in Multi-channel Wireless Ad Hoc Networks. AAMAS 2017: 1026-1034

- Peiyang Li, Xia Xie, Hai Jin, Hanhua Chen, and Xijiang Ke. Optimizational Methods for Index Construction on Big Graphs, Proceedings of IEEE 10th Asia-Pacific Services Computing Conference (APSCC 2016), Zhangjiajie, China, Nov. 16-18, pp:292-305
- Kai Wang, Xia Xie, Hai Jin, Pingpeng Yuan, Feng Lu, and Xijiang Ke. Frequent Subgraph Mining in Graph Databases Based on MapReduce, Proceedings of IEEE 10th Asia-Pacific Services Computing Conference (APSCC 2016), Zhangjiajie, China, Nov. 16-18, pp: 464-476
- Dongxiao Yu, Li Ning, Yifei Zou, Jiguo Yu, Xiuzhen Cheng, Francis C. M. Lau. Distributed Spanner Construction With Physical Interference: Constant Stretch and Linear Sparseness. IEEE/ACM Transactions on Networking(ToN), 2017.
- Xiaofeng Ding, Li Wang, Zhiyuan Shao, Hai Jin. Efficient Recommendation of De-identification Policies using MapReduce. IEEE Transactions on Big Data(TBD), 2017
- Yuan Liu, Xuanhua Shi, and Hai Jin. "Runtime-aware adaptive scheduling in stream processing." CCPE, 2016
- Yunjie Du, Xuanhua Shi, Hai Jin, Song Wu, Laurence T. Yang. FITDOC: fast virtual machines checkpointing with delta memory compression. The Journal of Supercomputing 72(9):3328-3347(2016)
- Yuanzhen Geng, Xuanhua Shi, Cheng Pei, Hai Jin, Wenbing Jiang, "LCS: An Efficient Data Eviction Strategy for Spark", International Journal of Parallel Programming, doi:10.1007/s10766-016-0470-1, 2016
- Xia Xie, Xiaodong Yang, Xiaokang Wang, Hai Jin, Duoqiang Wang, Xijiang Ke. BFSI-B: An Improved K-hop Graph Reachability Queries for Cyber-physical Systems, Information Fusion, Vol.38, pp:35-42, 2017
- Yun Hao, Gaofeng Li, Pingpeng Yuan, Hai Jin, Xiaofeng Ding. An Association-Oriented Partitioning Approach for Streaming Graph Query. Scientific Programming 2017.
- Feng Zhao, Yajun Zhu, Hai Jin, Laurence T. Yang. A personalized hashtag recommendation approach using LDA-based topic model in microblog environment. Future Generation Computer Systems. 65: 196-206 (2016)
- Hanhua Chen, Hai Jin, Xiaolong Cui: Hybrid followee recommendation in microblogging systems. SCIENCE CHINA Information Sciences 60(1): 12102, 2017
- Yong Chen, Juncheng Yao, Hai Jin, Chunjiang He, Hanhua Chen: Exploring the Evolution of New Mobile Services. Scientific Programming 2017: 5159690:1-5159690:9, 2017

华强胜

华中科技大学计算机学院副教授

研究方向：分布式算法

Email：qshua@hust.edu.cn

于东晓

华中科技大学计算机学院副教授

研究方向：分布式计算、无线网络、图算法和机制设计

Email：dxyu@hust.edu.cn

张　凡

2014级博士

研究方向：分布式流计算，大规模图计算

Email：zhangf@hust.edu.cn

廖良翌

2015级硕士

研究方向：近似集合成员判定技术，分布式流计算

Email：liaoliangyi@hust.edu.cn

刘　伟

2016级硕士

研究方向：高性能计算

Email：cccloude@hust.edu.cn

石宣化

华中科技大学计算机学院教授

研究方向：云计算与大数据处理、异构并行计算等

Email：xhshi@hust.edu.cn

Js渲染引擎比较 HtmlUnit/Selenium/PhantomJs

（王 摧 http://blog.csdn.net/w305172521/article/details/74853089）

现如今的爬虫再也不是简单的爬取静态页面、解析Html文本这么简单，许多单页面应用、异步请求调用、页面初始化js渲染等技术的使用，使得传统的通过发起http请求获得的Document无法直接使用。因此，基于实际业务需求，在爬取某电商平台数据时，发现其页面特定位置为js渲染，因此，有此一文，基于实际代码测试，分析HtmlUnit/Selenium/PhantomJs三类流行的js渲染引擎。

–HtmlUnit

1）内置Rhinojs浏览器引擎，没有哪一款浏览器使用该内核

2）解析速度一般

3）解析JS/CSS差

4）无浏览器界面

– Selenium

1）Seleninum 1+ WebDriver = Selenium

2）基于本地安装的浏览器，需打开浏览器

3）需要引用相应的WebDriver，正确配置webdriver的路径参数

4）在爬取大量js渲染页面时明显不合适

– PhantomJs

1）神器，短小精悍

2）可本地化运行，也可作为服务端运行

3）基于webkit内核，性能及表现良好

4）完美解析绝大部分页面

注：基于实测结果，在爬取大量任务时，推荐将PhantomJs作为服务端使用，此处，分别介绍本地及远程服务端使用的例子（也可查看官网example）

本地

需要构造目标执行的js文件，利用命令行调用PhantomJS 。

示例：

window平台下

PhantomJs.exe target.js param1

对应的本地target.js可参考如下示例：

```
"use strict";
var page = require('webpage').create();
var system = require('system');
if (system.args.length !== 2) {
    console.log('Usage: server.js <some port>');
    phantom.exit(1);
} else {
    var url = system.args[1];
    page.open(url, function (status) {
            console.log(page.content);
            phantom.exit();
            }
        });
```

在Java程序中，通过调用控制台执行命令

```
        Runtime runtime = Runtime.getRuntime();
        Process p = runtime.exec("D:/phantomjs.exe target.js url);
        InputStream is = p.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(is));
        StringBuffer sb = new StringBuffer();
        String tmp = "";
        while((tmp = br.readLine())!=null){
            sb.append(tmp);
        }
        return sb.toString();
```

搭建远程服务器

保证远程服务器指定端口开启；

示例：

在阿里ecs上开启指定端口，如3003；打开控制台，在安全组中添加自定义TCP连接，可访问的ip组设置为0.0.0.0/0，同时配置入网和出网端口。

操作步骤

1）官网下载exe文件至指定位置（Linux平台同理）

2）新建一个server.js文件

3）命令行运行PhantomJS server.js即可开启服务

4）本地通过在浏览器或者java代码中提交http请求，即可获得响应，url为 http://远程服务器ip地址：端口号/http://自定义url

注：此处server.js为关键，其设置了服务器的监听端口及响应请求逻辑 。

server.js示例代码：

```
"use strict";
var page = require('webpage').create();
var server = require('webserver').create();
var system = require('system');
var host, port;

if (system.args.length !== 2) {
    console.log('Usage: server.js <some port>');
    phantom.exit(1);
} else {
    port = system.args[1];
    var listening = server.listen(port, function (request, response) {

        console.log("GOT HTTP REQUEST");
        console.log(JSON.stringify(request, null, 4));

        // we set the headers here
        response.statusCode = 200;
        response.headers = {"Cache": "no-cache", "Content-Type": "text/html"
        // this is also possible:
        response.setHeader("databee", "databee");
        // now we write the body
        // note: the headers above will now be sent implictly
        //response.write("<html><head><title>YES!</title></head>");
        // note: writeBody can be called multiple times
       // var url = "http://www.baidu.com";
        var url = request.url;
        url = url.substring(1);//获得的url较为奇怪，根据request的内容进行url改造成
        page.open(url, function (status) {
            if (status !== 'success') {
                response.statusCode = 403;
                 response.headers = {
                    'Cache': 'no-cache',
                    'Content-Type': 'text/html'
                };
                response.write("FAIL");
                response.close();
                console.log('FAIL to load the address');
            } else {
                response.statusCode = 200;
                response.headers = {
                    'Cache': 'no-cache',
                    'Content-Type': 'text/html'
                };
                //console.log(page.content)
                response.write(page.content);
                response.close();//response.close()表明响应结束，必须加入
                console.log('Send success');
            }
        });

       //response.close();
    });
    if (!listening) {
        console.log("could not create web server listening on port " + port)
        phantom.exit();//代表退出phantom
    }

}
```

提供本地发起请求Java代码示例：

```
        URL url = new URL(finalUrl);//finalUrl此时为get请求url
        HttpURLConnection conn = (HttpURLConnection)url.openConnec
        InputStream is = null;
        BufferedReader br = null;
        if (conn.getResponseCode() == 200) {
            is = conn.getInputStream();
        } else {
            is = conn.getErrorStream();
        }
        br = new BufferedReader(new InputStreamReader(is));
        String line = "";
        StringBuilder sb = new StringBuilder();

        while ((line = br.readLine()) != null) {
            sb.append(line);
        }
        return sb.toString();
```

总 结

先了解，后使用。在未深入了解三款js渲染引擎时，走了许多弯路，尤其是在HtmlUnit的使用上，发现经常报内存溢出，js/css渲染错误等各种问题。

先官网，后博客。很多问题往往在官网上能够得到解答，尤其是官网上提供了响应的demo或者example说明，仔细阅读提供的示例代码，即使在没有注释说明的前提下，通过阅读代码，也能知其大致用法。

先业务，后工具。工具往往只是解决问题的一个手段和方法，如果对自身业务不了解，盲目使用工具，势必得不偿失。许多工具有其自身特点，如何正确使用，往往取决于自身的业务场景与需求。因此，凡事无绝对，没有最好，只有更好；没有最差，只有更差。

浅谈中断虚拟化

（原文链接: http://sec-lbx.tk/2017/07/30/%E8%AF%A6%E8%A7%A3%E4%B8%AD%E6%96%AD%E8%99%9A%E6%8B%9F%E5%8C%96/）

中断虚拟化做了什么?

我们首先回忆一下中断的过程：中断由设备发送给了I/O APIC，随后交给CPU本地的LAPIC进行处理。LAPIC把它交给CPU，然后由操作系统根据相应的IDT表进行处理。

很明显，如果虚拟机内的操作系统，要正常的去处理中断，就必须要由VMM来完成这两个部分：

（1）虚拟化LAPIC，并虚拟化客户机对APIC寄存器的访问

（2）虚拟化到达客户机的所有中断

APIC的虚拟化

当虚拟机访问local APIC寄存器时，最终同样会对物理上的local APIC进行读写。那么VMM该如何对它进行控制，防止其干扰宿主机的环境呢？VMM有两种方式，来对local APIC进行限制：

（1）通过EPT来进行保护，处理对local APIC的读写，但这样每次都会触发EPT violation，无疑会引入开销；

（2）利用Virtual-APIC机制，guest对local APIC的访问实际上是对virtual-APIC页进行访问。

通过设置VMCS中的Virtualize APIC accesses字段为1，会开启第二种方式：当guest访问APIC-access page时，会根据APIC-register virtualization当中的设置，来决定这次访问的方式：

（1）产生APIC access VM EXIT，或者APIC write VM EXIT

（2）通过virtual-APIC机制

这里，我们详细讨论一下virtual-APIC方式。

APIC-access page和Virtual APIC accesses

当Virtualize APIC accesses开启时，处理器会启用APIC-access page。此时APIC-access page对应着物理的local APIC页面，其地址保存在VMCS的APIC-access address当中。从硬件上，CPU能够帮助我们去监控这个页，这样一来，只要guest尝试线性访问local APIC，就会访问APIC-access page。那么此时，会根据“APIC-register virtualization”位的设置，进行不同的处理：

1. 发生APIC access VM-exit，或者APIC write VM-exit直接访问

2. 访问virtual-APIC page中的数据（某些寄存器）

virtual-APIC page是什么？当“use TPR shadow”为1时，会引入virtual-APIC page，它是local APIC的影子页面，其地址保存在VMCS的virtual-APIC address当中。它保存了一份TPR数据，也即virtual TPR寄存器。实际上，virtual-APIC page的唯一用处就是对寄存器进行虚拟化，这样每个VCPU都能够有不同的TPR、VEOI、VLDR等寄存器，其中TPR寄存器保存在virtual-APIC page的80H位置。

那么APIC-access page和virtual-APIC page之间是什么关系呢？实际上是一种类似映射的关系，当guest尝试线性访问（注意，只有线性访问）APIC-access page页面时，会实际上访问vritual-APIC page页面中的数据。

在虚拟机访问APIC access page时：

（1）如果线性访问了没有虚拟化支持的offset位置，会产生APIC-access VM-exit

（2）采用guest-physical方式访问APIC-access page页中的任何offset位置，都会发生

APIC-access VM-exit

（3）采用physical方式访问APIC-access page，可能发生APIC-access VM-exit

这里我们主要关注线性访问的情况，来具体说明APIC-access page和virtual-APIC page的工作原理。

读APIC-access page

在如下情况时，访问APIC-access page时，都会产生APIC-access VM-exit:

（1）“use TPR shadow”为0

（2）尝试执行APIC-access page

（3）访问数据超过32位

（4）访问APIC-access page的动作发生在虚拟化写APIC-access page过程中

（5）尝试进行跨过local APIC寄存器边界的访问

在除上述原因之外的情况下，读取时，会根据APIC-register virtualizaion位的设置，来进行读取：

（1）如果APIC-register virtualization为0，则如果线性读取TPR，直接从virtual-APIC page返回；否则直接产生APIC-access VM-exit

（2）如果APIC-register virtualization为1，如果访问的是某些特定的偏移量，那么就会返回virtual-APIC page中的数据；否则直接产生APIC-access VM-exit

写APIC-access page

在如下情况时，访问APIC-access page时，都会产生APIC access VM-exit：

（1）“use TPR shadow”为0

（2）访问数据超过32位

（3）访问APIC-access page的动作发生在虚拟化写APIC-access page过程中

（4）尝试进行跨过local APIC寄存器边界的访问

在除上述原因之外的情况下，读取时，会根据APIC-register virtualizaion和virtual-interrupt delivery位的设置，来进行写入：

（1）如果APIC-register virtualization和virtual-interrupt deliverty都为0，那么如果线性写TPR时，会写入VTPR中（virtual-APIC page），其他位置都会产生APIC-access VM-exit

（2）如果APIC-register virtualization为0，但virtual-interrupt deliverty为1，那么线性写TPR、EOI、ICR时，会写入virtual-APIC page中对应的虚拟寄存器当中，其他位置都会产生APIC-access VM-exit

（3）如果APIC-register virtualization为1，则会允许更多的虚拟寄存器写入virtual APIC，其他位置则产生APIC-access VM-exit

但是写入是一个更为复杂的过程，在写入对应的虚拟寄存器之后，处理器还要进行后续的处理，也即APIC-write emulation：根据virtual-interrupt deliverty的设置，产生APIC-write VM-exit，或者对local APIC进行虚拟化操作（TPR、EOI、Self-IPI等）。

小 结

APIC虚拟化的过程，实际上也是借助了硬件的力量。根据CPU的设置，可以把不同VCPU的虚拟寄存器保存在不同的页中。这样不仅避免了虚拟机对素主机的污染，还能够减少VM-exit的发生，并且能够让每个VCPU拥有独立的寄存器，对性能是有极大提高的。

中断的evaluation和delivery

现在，我们来看看VMM是如何判断一个中断是否应该交给虚拟机进行处理，并且将它转交给虚拟机的。

VMCS中有一个External-interrupt exiting位。当这个位设置为1时，如果CPU处于non-root模式，那么所有中断都会产生VM-exit；如果设置为0，则所有中断都会直接交给虚拟机进行处理。这无疑是很不安全的，所以一般情况下外部中断都会造成VM-exit，并且在虚拟化层处理完返回虚拟机（VM-entry）时，需要做中断的

evaluation（判断是否有中断要交给虚拟机处理）和delivery（将中断交给IDT处理）。

在这几种情况下，会引发中断的evaluation和delivery：

（1）TPR虚拟化、EOI虚拟化、Self-IPI虚拟化。

（2）在VM-entry操作时引发。

（3）在利用“post-interrupt processing”机制处理外部中断时引发。

中断的注入，是通过由VMM将中断写入VMCS对应的信息位来实现的，当中断完成后，通过读取中断的返回信息，来分析中断是否正确。在中断注入之后，处理器会完成evaluation和delivery的工作（还有另一种方式是posted interrupt处理）

那么evalutaion会根据什么来判断一个虚拟中断是否应该pending呢？处理器首先根据虚拟中断的优先级别进行判断，只有在RVI[7:4]大于VPPR[7:4]时，处理器才会组织一次虚拟中断pending。随后处理器还要检查虚拟中断是否被阻塞，只有不被阻塞的时候，会通过guest-IDT来进行deliver执行。

在evaluation之后，虚拟中断就会通过guest-IDT 进行delivery。这时处理器将会更新virtual-APIC的状态，包括VIRR、VISR、RVI、SVI状态等，随后处理器根据虚拟中断向量号，通过guest-IDT表进行deliver执行。

posted interrupt处理

前面我们也提到了，外部中断通常都会造成VM-exit的下陷。posted interrupt处理，能够直接把中断注入到guest当中去，而不需要产生VM-exit。

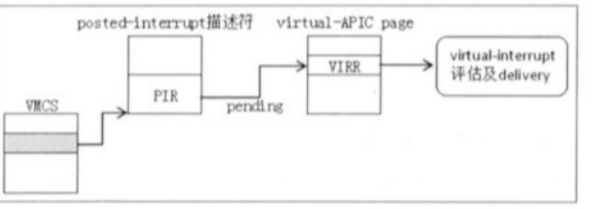
假如启用了posted interrupt处理，那么处理器对外部中断的处理就不同了：

（1）一般情况下，如果external-interrupt exiting为1，那么外部中断产生VM-exit；否则直接通过guest IDT表处理；

（2）在posted interrupt机制下，如果处理器接收到的外部中断向量号是预先定义的通知向量号时，不会产生VM-exit，而是通过PIR的pending处理。

那么这个不产生VM-exit的过程是如何发生的呢？

VMM会在posted-interrupt notification vector当中设置通知向量号，一旦local APIC接到对应的外部中断，那么就会通知guest进行post-interrupt处理。此时posted interrupt描述符内的PIR请求列表会复制到VIRR（virtual-APIC页的另一个用途），形成新的虚拟中断请求列表，然后再对这个虚拟中断请求表中的中断请求进行evaluation和delivery。

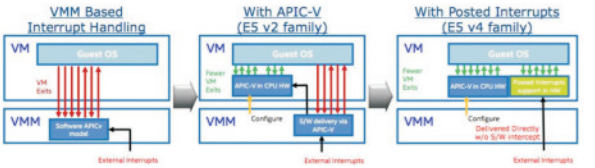


而如果外部中断向量号不等于通告的向量，那么依然会触发VM-exit，处理器获取外部中断的信息，通过VMCS中的字段来注入中断。

小 结

可以看到，中断虚拟化方式的改变，是随着硬件特性的改变而改变的。硬件的支持，主要就是为了降低虚拟化的开销。我们知道，VM-exit无疑会给虚拟机的性能带来损耗，而中断的特殊性，又使得它极有可能造成VM-exit。所以intel的工作，一直都致力于减少VM-exit的发生，提升虚拟机的性能。让虚拟机直接接管硬件一直是虚拟化中极为重要的一种思想，不论是IO还是中断，或者是内存，都是如此。

Ps：《处理器虚拟化技术》这本书极烂，不建议阅读。



教育部科技委信息学部2017年度工作会议在武汉召开

吴　未

7月23日，教育部科技委信息学部2017年度学部工作会议在武汉召开。本次会议由华中科技大学和武汉大学联合承办。华中科技大学骆清铭副校长、武汉大学李建成副校长，国家自然科学基金委政策局郑永和副局长，国家自然科学基金委政策局董超副研究员，教育部科技司高新处刘法磊，教育部科技委副秘书长朱小萍、科技委魏纯辉，教育部科技委信息学部主任吴建平院士，常务副主任尤肖虎教授，学部副主任鲍虎军教授、金海教授和其他学部委员、专家等共30余人参加了会议。

开幕式上，骆清铭校长代表华中科技大学致欢迎辞，向与会领导、专家的到来表示热烈欢迎和诚挚感谢。骆校长介绍了华中科技大学以及学校开展双一流建设及学校信息学科的发展状况。李建成副校长介绍了武汉大学信息学科的发展现状。

教育部科技委朱小萍副秘书长通报了教育部科技司和教育部科技委学部2017年的工作要点。国家自然科学基金委政策局郑永和副局长作了题为《推进教育基础研究资助工作的若干思考》的报告，希望通过自然科学基金项目资助部署，广泛吸引不同领域的科学家开展多学科交叉的基础研究，来解决教育创新发展中亟待解决的科学问题。随后，信息学部主任吴建平院士对2016年学部工作进行了总结，通报了信息学部2017年工作计划及进展，规划了下半年工作要点，提出了2017年七项重点工作。华中科技大学金海教授作了题为《科技创新2030大数据重大专项》的报告。

根据科技委工作部署和要求，20多位学部委员分别从国家发展战略高度出发，研讨信息领域前瞻性重大战略问题等重要议题，并经过充分讨论达成共识，落实了重大战略研究报告、重大项目建议、专家建议的撰写任务。会议期间，与会专家还参观、考察了华中科技大学服务计算技术与系统教育部重点实验室和武汉大学遥感信息工程学院，学部副主任、浙江大学鲍虎军教授为华中科技大学服务计算技术与系统教育部重点实验室的师生做了一场学风与科普宣传的讲座。

第七届教育部科技委员会共设13个学部，其中信息学部是国家和教育部在信息领域的思想库和智囊团。本届信息学部共包含专家40人，其中学部委员31人及科技委委员9人，挂靠单位为东南大学。

吴　未
硕　士
主要负责实验室宣传、项目管理等工作。
E-mail：wwuhust@hust.edu.cn

实验室2017年暑期年会召开

吴　未

2017年8月3日至5日，实验室2017年暑期年会隆重召开，包括教师、博士生、硕士生在内的260余人参会。自实验室2001年成立至今，暑期年会每年召开一次，今年是第十七届。

大会开幕式由实验室主任金海教授主持，并首先为本次年会做了致辞，回顾了实验室的发展历程，介绍了实验室的基本情况。然后，受大会邀请，来自加拿大多伦多大学的李葆春教授以及英国华威大学的何黎刚副教授分别作了题为“Optimizing Job Performance within and across Datacenters”、“WolfGraph: Minimizing the graph pre-processing time on GPU”的大会特邀报告。

开幕式后，实验室四个研发组的代表老师依次作了科研方向前瞻研究报告。肖江副教授作了题为“区块链+大数据：开启‘可信价值互联网’时代”的报告，博士后郑龙作了题为“量子计算系统研究：我的一点思考”的报告，邹德清教授作了题为“软件安全关键技术研究”的报告，余辰教授作了题为“普适计算的前世今生”的报告。之后，各研发组组长报告了过去一年来的项目进展和成果情况，以及下一步工作计划和方向规划等，并分别进行了系统演示。

针对各研发组的报告，师生踊跃发言，对其中的理论、方法和技术展开了认真的讨论。金海教授从各方向的科研发展前沿提出了问题，为相关研究工作给予了指导。其他老师针对各研发组的工作也提出了宝贵的意见和建议。通过深入而广泛的交流和探讨，各研发组进一步明确了研究方向和工作重点、优化了研究计划，确保今后研究工作能够按计划稳步推进。

8月5日，王新猴等8名博士生分别汇报了各自的博士毕业论文撰写提纲和论文答辩计划，老师们针对汇报内容与博士生一一交流并提出了修改意见。大会最后由金海教授进行了总结，公布了经教师工作会议讨论形成的教师和研究生分组方案。

本次年会之前，8月1日–2日召开了实验室科研发展研讨会，实验室全体教师参加。会上，各位教师汇报了个人发展规划。最后金海教授指出要改变做科研的方式，方法，作风，“要把论文写在祖国的大地上，把科技成果应用在实现现代化的伟大事业中”，真正做有利于社会，有利于发展。

吴　未
硕　士
主要负责实验室宣传、项目管理等工作。
E-mail：wwuhust@hust.edu.cn

What Makes a Link Successful on Wikipedia?

吴 尧 推荐

"*What Makes a Link Successful on Wikipedia?*"是被计算机国际顶级会议WWW17录用的一篇文章。虽然Web上存在大量超文本链接，但只有少量的链接被经常点击。基于以上观察，文章对大规模的维基百科点击数据进行研究，以了解链接成功被点击的原因。文章研究成果有助于理解用户浏览行为，利用这些发现能改进链接结构和相应算法。

文章系统地分析了链接不同特征是如何影响链接的流行程度。通过分析发现用户偏好点击通向网络外围的链接，语义相似度高的链接文章也更易被点击，而位于屏幕顶端和左侧的链接点击率更高。在研究用户浏览行为时，一阶马尔科夫链模型被广泛的采用的随机模型，因此文章将这些结果作为假设改进该模型，并成功地改善模型的精度；同时把得到的结论作为权值改进了PageRank算法，提高算法精确度。

文章主要关注链接特征对链接流行程度的影响，研究了三种不同特征类型：网络特征，语义相似性特征、视觉特征。网络特征指某一链接中源页面和目标页面的在维基百科链接网络中的中心性，如入度、出度、PageRank、k-core等。语义相似特征指某一链接中源页面和目标页面的相似程度。文章测量了文本相似性和话题相似性，通过利用亚线性缩放的tf-idf加权向量空间模型来捕捉维基百科文章的词类以计算文本相似度，两个页面之间的文本相似度由相应向量的余弦相似度确定；同时根据分配给源页面和目标页面的类别，利用余弦计算了一个主题相似性得分。第三类视觉特征指目标页面的链接放在源页面的位置特征，文章将页面分为6个部分，并将链接第一次出现的位置作为该连接的视觉特征值。

Feature	# Links	# Transitions	Mean Trans. per Link
trg_degree < src_degree	462,147	3,944,747	8.54
trg_degree ≥ src_degree	566,557	2,741,834	4.84
trg_in_degree < src_in_degree	412,653	3,831,100	9.28
trg_in_degree ≥ src_in_degree	616,051	2,855,481	4.64
trg_out_degree < src_out_degree	545,980	3,912,604	7.17
trg_out_degree ≥ src_out_degree	482,724	2,773,977	5.75
trg_kcore < src_kcore	283,668	3,477,828	12.26
trg_kcore ≥ src_kcore	745,036	3,208,753	4.31
trg_page_rank < src_page_rank	414,124	3,833,596	9.26
trg_page_rank ≥ src_page_rank	614,580	2,852,985	4.64
text_sim > median of article	511,871	4,335,445	8.47
text_sim ≤ median of article	516,833	2,351,136	4.55
topic_sim > median of article	392,955	3,246,766	8.26
topic_sim ≤ median of article	635,749	3,439,815	5.41
position = lead	93,546	2,489,342	26.61
position = body	373,407	3,442,017	9.22
position = left-body	64,320	934,766	14.53
position = right-body	309,087	2,507,251	8.11
position = navbox	502,079	99,498	0.2
position = infobox	59,672	655,724	10.99
screen_x_coord: left third	211,549	2,358,899	11.15
screen_x_coord: middle third	291,260	1,422,183	4.88
screen_x_coord: right third	525,847	2,905,499	5.53
screen_y_coord: top half	221,980	3,991,912	17.98
screen_y_coord: bottom half	174,500	998,287	5.72
screen_y_coord: scroll needed	632,176	1,696,382	2.68
Overall	1,028,704	6,686,581	6.5

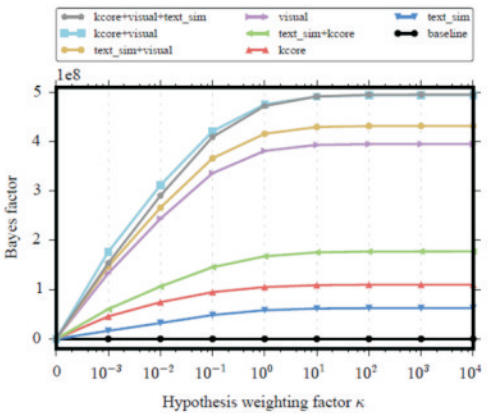
图一：描述性统计结果

为了研究链接特征对链接成功被点击的影响，文章对数据进行描述性统计，并结合"mixed-effects hurdle models"来分析。图一显示了描述性统计的结果，显示了每个链接特征的链接数，链接的累积转换次数，以及每个链接的平均转换次数。结果表明通向网络周边的链接，连接语义上类似的文章和位于上方和左边区域的链接积累相对较多的转换。图二显示的实验结果进一步验证了描述性统计得到的用户偏好结果。

Feature	Transformation	Binomial Model		
		Fixed effect	LRT	Pr(>Chi)
trg_degree	scale	-9.0704	9441	< 1.0 e-300 ***
trg_in_degree	scale	-9.2925	9456	< 1.0 e-300 ***
trg_out_degree	scale	-0.6425	4052	< 1.0 e-300 ***
trg_kcore	scale	-9.4873	9470	< 1.0 e-300 ***
trg_pagerank	scale	-8.8103	9384	< 1.0 e-300 ***
text_sim	scale	0.2944	617	3.4 e-136 ***
topic_sim	scale	0.2052	482	8.6 e-107 ***
position = lead	none	1.9682	6146	< 1.0 e-300 ***
position = body	none	0.2351	93	5.9 e-22 ***
position = left-body	none	1.106	739	9.4 e-163 ***
position = right-body	none	-0.3631	231	4.4 e-52 ***
position = infobox	none	-0.1955	25	7.2 e-07 ***
position = navbox	none	-6.3237	9226	< 1.0 e-300 ***
screen_x_coord	scale	-0.2974	1081	4.4 e-237 ***
screen_y_coord	scale	-4.258	10588	< 1.0 e-300 ***

图二：Mixed-effects hurdle model建模结果

文章在得到了影响链接成功的因素后，继而考虑了如何将这些结果整合到现有的用户浏览模型中。为此，文章研究了用户浏览模型中链接特性的影响。文章选取了一阶马尔科夫链模型，该模型被广泛应用于研究用户浏览行为。首先，文章将得到的用户浏览行为偏好的结果作为假设，并将其整合到贝叶斯推断过程中。通过观察给出假设后贝叶斯因子的变化，发现给出用户浏览行为偏好的假设后，可以更好地解释页面转换行为，提高了马尔可夫链模型拟合程度；与原有的基础模型对比实验，也间接验证了影响链接成功被点击的链接特性。实验结果如图三所示，在不同的假设权重因子的影响下，加入不同用户浏览偏好假设与原有随机浏览假设相比得到的贝叶斯因子均为正数，表明改进后马尔科夫模型拟合度提高，其中整合视觉和网络特征的模型优化效果明显。



图三：贝叶斯因子结果

Damping factor α/Hypothesis M	0.80	0.85	0.90
baseline	*0.421*	*0.428*	*0.436*
kcore	**0.434**	**0.440**	**0.447**
visual	**0.507**	**0.516**	**0.526**
text_sim	0.400	0.407	0.415
text_sim+kcore	0.407	0.412	0.417
text_sim+visual	**0.489**	**0.500**	**0.513**
kcore+visual	**0.530**	**0.538**	**0.545**
kcore+visual+text_sim	**0.494**	**0.505**	**0.517**

图四：PageRank算法对比实验结果

同时，文章提出了加权的PageRank算法，并与观察到的访问维基百科文章数量对比验证算法的精确度。传统的PageRank算法假设链接网络中某个链接跳转至相邻链接的概率是相等的。文章中利用之前得到的用户浏览偏好的结果，根据不同的链接特征赋予了链接网络中边不同的权值。在不同的链接特征假设的影响下，文章将加权PageRank算法其与传统算法进行对比，实验结果（如图四）显示：对比传统算法，基于网络特征和视觉特征改进的加权PageRank 算法得到的结果更接近实际数据中的维基百科文章访问数量。

文章对数据进行描述性统计并结合“Mixed-effects hurdle model”研究链接流行度的影响因素，发现维基百科的用户喜欢浏览维基百科链接网络的边缘页面、语义相似的页面，以及页面链接在源页面的顶部或最左侧页面。根据这些结论，文章将用户浏览偏好整合到贝叶斯推理过程中，提高了马尔可夫链模型的拟合度。同时，文章给链接网络的边赋予权值以显示浏览偏好来改进PageRank算法，得到PageRank值对比未加权基线算法更符合实际的观察统计结果。

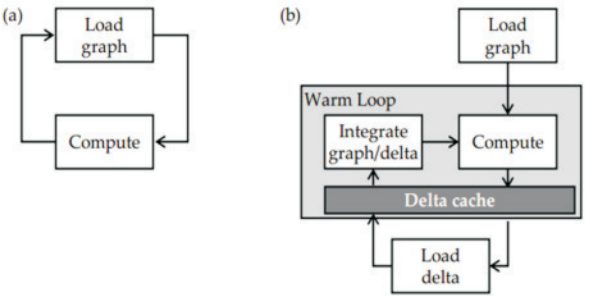
吴　尧
2016级博士研究生
研究方向：大数据分析
Email：yaowu@hust.edu.cn

Version Traveler: Fast and Memory-Efficient Version Switching in Graph Processing Systems

武斯杰 推荐

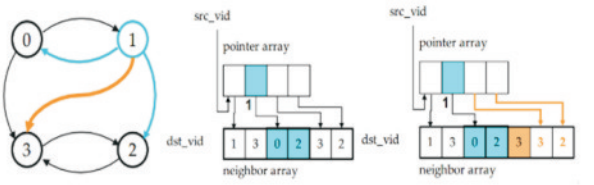
“Version Traveler: Fast and Memory-Efficient Version Switching in Graph Processing Systems”是被计算机国际顶级会议ATC 2016录用的一篇文章。本文作者观察到现有图处理系统在处理多版本图的过程中版本切换的时间和内存开销很大，针对这一问题设计了一种新的图数据的数据结构，可以快速地在不同版本的图数据之间进行切换。在此基础上，本文作者实现了一个面向多版本图的图处理系统Version Traveler。实验表明，与传统的图处理系统相比，Version Traveler实现了最多23倍的处理速度提升，而且降低了内存的使用。

随着时间的推移，图数据往往是在不断改变的。如Facebook的社交网络图，每一个新用户，每一种新的用户之间关系的增加和删除，都意味着图数据的更新。传统的图处理系统在处理多个版本图数据的过程中，采用的是不断的load-compute的机制，如图一(a)所示，每次要处理新版本图的时候，先把现有图从内存中删除，再读取新的图到内存中。现有研究表明，图的读取是图处理的瓶颈，会占用大量时间。而多版本图处理过程中这种不断读取新版本的图的操作更是会产生大量的时间开销，影响了图处理的效率。现有的一些系统提出，可以用delta存储不同版本图之间的差异信息，在切换不同版本的图的时候，通过现有版本和目标版本之间的delta来计算出目标版本，避免了不断读取新图的时间开销，如图一(b)所示。



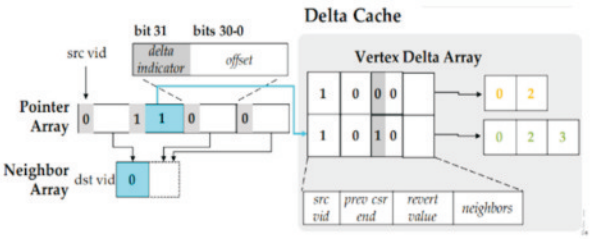
图一：多版本图处理过程

现有的图处理系统大多采用CSR模型来存储图数据，如图二所示。每个点的邻居节点都连续存放在neighbor array中，这样在图处理过程中读取邻居节点时就是连续的内存访问，读取速度很快。然而，在CSR模型上进行图数据的更新操作则是十分复杂的。当节点1和3之间有一条新的边连接时，就要把节点3插入到neighbor array中节点1对应的邻居节点的位置，插入位置之后的所有点都要后移一位，产生了极大的开销，特别是更新操作十分频繁时，这种开销就更大。所以通过delta推导目标版本的方法在CSR模型中并不适用。



图二：CSR存储模型

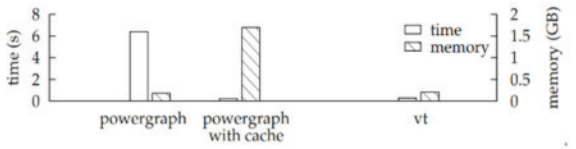
本文设计了一种新的图数据存储模型，实现了支持图数据更新操作和快速版本切换的目标，如图三所示。



图三：VT图存储模型

Version Traveler把不同版本之间的差异也存在内存中。当一个点的邻居节点有更新的时候，先改变这个点在pointer array里对应的delta indicator位，然后把更新后的数据存在delta cache中。在需要版本切换的时候，就改变CSR模型中pointer array里节点对应的offset，指向需要的版本。如图三中青色箭头所指，说明第三个版本中节点1有3个邻居节点，分别是0, 2, 3。

这种方法在切换版本的时候，只需要改变指针所指的位置即可，所以切换速度很快。但是一个点无论有多大的更新，每更新一次，这个点的所有邻居节点信息都需要存一遍，这种内存开销是很大的。本文在delta cache存放neighbors的过程中做了一些优化。通过使用指针，把不同版本的邻居节点串联起来，使不用版本的同一个点可以共享邻居节点的数据，降低了同一个点邻居节点数据存放的次数，极大地降低了内存的开销。



图四：与powergraph对比

图四展示了VT与powergraph的对比，分别从处理时间和内存占用两个方面进行了比较。Powergraph有两种配置，分别是：1）每次都重新读取新的图。2）把所有版本的图都预先cache进内存中。可以看出当把所有版本的图都放进内存中时，powergraph的处理时间很低，然而其内存占用却十分大，这种方式在真实的图处理中并不实际。与第一种配置相比，VT实现了23倍的处理速度的提升。与第二种配置相比，VT仅占用了其15%的内存。所以说Version Traveler通过不cache图的方式，达到了与传统图处理系统cache所有版本图相近的效果。

总结：本文针对现有图处理系统在处理多版本图过程中版本切换不高效的问题，设计了一种基于CSR模型的新的图数据存储结构，并实现了一个新的多版本图处理系统Version Traveler，利用较少的内存，实现了快速的不同版本图的切换和处理。与powergraph相比，处理速度提升了23倍。

武斯杰
2016级博士研究生
研究方向：图计算
Email：wsj@hust.edu.cn

Developing the Graph-based Methods for Optimizing Job Scheduling on Multicore Computers

报告人：何黎刚　副教授
英国华威大学
时　间：2017年4月20日
整　理：刘长鸣

何黎刚博士介绍了作业调度在作业处理过程中，尤其是在多核计算机上的重要性。何黎刚博士先从现有工作入题，分析了现有工作在最优调度解决方案中的不足，然后引出并介绍了基于图的串行作业的最优调度方法，之后把该方法扩展到对并行作业的处理，并介绍了一系列的优化方法来加速求解过程。

何博士的报告内容丰富，深入浅出地介绍了基于图的作业调度方法，拓展了同学们的视野。在交流过程中，实验室的老师和同学与何博士就作业相似度、作业类型划分、作业图构造、优化求解、加速求解等方面的问题进行了深入探讨，同学们启发颇多，对科研有了更多新的研究思路。

何黎刚，博士，英国华威大学计算机系副教授，湖南大学“青年千人”。研究领域为高性能计算、并行分布式处理和大数据处理。在IPDS、JPDC、JCSS等国际学术期刊和IPDPS、ICPP、ICSOC等国际学术会议上(例如IPDPS, ICPP, ICSOC等)发表论文90余篇。何黎刚博士为多家国际期刊的编委或专刊客座编委，并经常担任国际会议的程序委员会成员及会议主席。

Memory Centric Optimization: Keep the memory hierarchy but nothing else

报告人：孙贤和　教授
美国伊利诺伊理工大学
时　间：2017年6月28日
整　理：夏　妍

孙贤和教授结合自身的实践经验指出存储性能对数据密集型计算的影响。孙教授首先简要说明存储墙问题对高性能数据分析的挑战及其研究意义。随后，针对传统存储性能测试模型存在的不足，提出C-AMAT（Concurrent-Average Memory Access Time）模型，该模型将传统模型推广到可同时考虑存储访问的局部性和并发性。随后又介绍了搏动数据传输方法，并基于搏动数据传输方法提出分层性能匹配全局控制算法，结合C-AMAT的分析结果减小存储墙效应，优化内存系统性能。

孙教授的报告内容丰富，深入浅出地介绍了一系列存储系统性能优化的研究成果，生动形象地呈现给现场在座的老师和同学，引人入胜。在报告的提问环节，孙教授和与会者就纯粹缺失的计算方式等问题进行了深入的交流与讨论，同学们启发颇多，对科研有了更多新的研究思路。

Optimizing MapReduce Framework through Joint Scheduling of Overlapping Phases

报告人：吴　杰　教授
美国天普大学
时　间：2017年7月17日
整　理：吴　尧

吴杰教授在报告中主要介绍了如何通过联合调度来优化MapReduce框架，以达到最大限度地减少平均工作量的目标。吴杰教授首先简要说明MapReduce框架中Map和shuffle阶段分别属于CPU密集型和I/O密集型，可以通过联合调度并行执行这两个阶段以减少平均工作量。然而，其中的挑战在于Map和shuffle两个阶段存在的依赖关系。为此，吴杰教授介绍了“perfect pair”的概念，证明了在整套作业可以完全分解为满足“perfect pair”性质的成对作业时，最优的调度就是并行执行可以形成“perfect pair”的成对作业。在此基础上，提出了联合调度的优化算法：将所有任务分为一定数量的组，要求每个组内最大的工作量差之和最小，然后在组内将任务进行配对执行。最后，吴杰教授用实验数据验证了该调度策略的可行性和有效性。

吴杰教授的报告通俗易懂、深入浅出，从具体的应用场景引入，生动形象地呈现了如何从具体问题抽象出模型。在讲座的提问环节，吴杰教授和与会者就联合调度改进策略、科研方法等问题进行了深入的交流与讨论，为同学们提供了新的研究思路和方法。

吴杰教授现任天普大学副教务长（主管外事工作）、网络计算中心主任、Laura H. Carnell荣誉教授。曾任美国科学基金会项目主任、美国Florida Atlantic大学杰出教授。吴杰教授是IEEE会士、IEEE分布式处理技术委员会（TCDP）主席，任IEEE Transactions on Service Computing和Journal of Paralleland Distributed Computing等国际期刊编委。研究领域包括移动计算和无线网络、云计算和绿色计算、网络安全和社交网络等。吴杰教授曾担任IEEE MASS 2006、IEEE IPDPS 2008、IEEE ICDCS2013、ACM MobiHoc 2014等国际会议主席，也曾任IEEE INFOCOM 2011和CCF CNCC 2013的程序委员会主席。吴杰教授是IEEE计算机学会杰出访问学者、ACM杰出演讲嘉宾、CCF杰出演讲人，并于2011年获得中国计算机学会海外杰出贡献奖。

接上页

孙贤和教授现任伊利诺伊理工大学可扩展计算软件实验室主任，同时也是美国阿贡国家实验室数学与计算机科学系客座教授，曾就职于美国艾姆斯实验室、NASA兰利研究中心、路易斯安那州立大学和海军研究实验室。孙贤和教授是IEEE会士，以研究内存制约加速比模型而闻名，该模型也称为Sun-Ni定律。孙贤和教授的研究包括并行和分布式计算，内存和I/O系统，大数据系统软件，计算系统性能评估和优化等，在这些领域发表学术论文250余篇，获5项发明专利。孙贤和教授曾任IEEE CS杰出演讲嘉宾、IEEE可扩展计算技术委员会副主席、伊利诺伊理工大学计算机科学系主任，是并行与分布式计算领域多个顶级期刊的编委。

感　恩

杜亚娟

记得大约五年前曾给自己设立过一个五年计划，希望能够拿到计算机博士学位、一次出国旅行和一次马拉松。虽然那时候生活忙碌单调，学业迷茫无所适从，精力也大不如前，却依然憧憬着更高山峰上更美的风景。如今五年即将到来，计划得以顺利完成，欣慰之余，我不得不感恩所经历的一切：感恩老师们，也感恩朋友和家人！

首先要衷心地感谢我的导师邹德清教授。老师兢兢业业的工作态度，与人为善的处事态度和积极健康的生活态度深深地影响着我，犹如一盏指路的明灯。在工作上，无论多忙多累，老师都坚持每周召开博士会议跟进每位同学的进展，并在科研和生活上答疑解惑，提供无私的帮助，让我敬佩不已。在为人处事上，老师处处为学生着想。有幸成为老师的第一个博士生，我倍感荣幸。那年我因怀孕无法到实验室上班，老师二话不说同意了我的申请，还时常询问我的情况，让我感动至今。老师热爱跑步锻炼，时常在工作之余一起到郊外跑步，教导我们身体是科研的本钱。感谢老师带给我的这些宝贵的生活态度和习惯。我性格急躁，做事情只凭一时热情却很难坚持，老师多次指出我的不足，使我时常反省这个缺点，提醒自己要耐心和坚持。感谢老师的谆谆教诲，让我在未来的工作和生活中逐步改正不良习惯，成为更好的自己。

感谢在香港城市大学的导师薛春教授。感谢老师对我耐心细致的指导和谆谆教诲。在读博的初期，因转专业我在计算机系统结构方面的知识比较薄弱，老师时常鼓励我别怕困难，保证每天的学习时间并养成良好的学习习惯。老师每周对我进行一对一的指导，为我的研究进度指明方向并教会我很多做人做事的道理。感谢老师资助我在香港期间的学习。虽然前面几年并没有出什么成果，但是老师仍然资助我。感谢老师没有放弃那时的我，并且成就了今天的我。感谢老师带领我进入科研的殿堂，让我在这个美好的世界里遨游畅想。期待未来与老师有更多的合作，创造出更多绚烂的研究成果。

感谢实验室主任金海教授。老师高屋建瓴，对计算机领域的前沿有着十分独到的把握和见解。在老师的引领下，实验室拿下多项国家和教育部重大项目，实验室的老师和同学多次在国际顶级会议和期刊上发表论文，实验室在国内外的科研界有着十分重要的地位。这些都离不开老师的辛苦奔波，感谢老师对实验室的奉献。能够在这样的实验室学习，我感到非常荣幸。

感谢科研上的合作伙伴重庆大学石亮副教授和李乔同学对我给予的帮助。在研究闪存系统性能初期，我对系统结构的知识匮乏，时常对当前的科研问题和实验方法充满困惑。感谢他们对我毫无保留的解答和帮助，使我逐步深入理解课题，并在这个领域

完成博士论文的工作。感谢他们和我并肩作战，一起经历那些通宵熬夜做论文的艰苦时光。感谢科研上的合作伙伴山东大学赵梦莹副教授，作为我在香港城市大学时曾经的师姐，感谢她对我写作的第一篇文章的精心指点，让我在后来的科研工作中受益匪浅。感谢与我合作的几个本科生贾云培、曾俊、朱玉、胡一公。在每周的小组讨论中，他们刻苦的努力和快速的学习能力使得课题研究得以顺利进展。感谢他们对我的配合和支持，给予我最珍贵的学生指导经历。

感谢实验室廖小飞老师，徐鹏老师，羌卫忠老师，马晓静老师和代炜琦老师等。感谢他们对我的支持。感谢实验室办公室刘英书老师、王小兰老师、吴未老师、耿聪老师对我的关照。感谢她们不辞辛劳地管理实验室各项事务，为我们营造良好的学习环境。感谢曾和我在一个课题组的全体同学们。感谢实验室同学王代斌、罗鹏、袁斌、李珍、张盼、夏阳、陈浩宇、何木青、石建、吴月明和李志等。感谢香港城市大学“香江学者”李建华，香港城市大学博士生傅忱忱、梁宇、纪程、李富洋、吴超、徐倩、周梓梦、盛建中和孙轩，重庆大学博士生高聪明等。和这么多优秀的同学共同学习和讨论，我倍感荣幸，受益匪浅。

感谢用勤劳的双手让我从落后的农村走出来的父母，是他们含辛茹苦的付出，让我能够在这漫漫二十多年的求学路中衣食无忧，专心钻研知识。每当看到他们爬满皱纹的脸颊，我潸然泪下。感谢他们为我所做的无私奉献，让我经历这般美好的人生。感谢我的宁先生，用他的辛勤工作撑起一个家。感谢这五年时间有他的陪伴。当我遇到任何困难时，他总是第一个为我分忧解难。感谢他带给我的爱情和亲情以及温暖的家庭港湾，为我遮风避雨。感谢公婆任劳任怨地照顾家庭和孩子，让我在攻读博士的五年生涯里，倾心倾力在科研工作上，不为家庭琐事分心。感谢可爱的沁沁小朋友。没能陪伴她的左右，是我最大的遗憾。感谢她的善解人意和活泼可爱，让我能够安心读书。她是我生命中的天使，有她甜美的笑容，我无论何时都不惧怕任何困难，一定坚持到底！

五年的时间转瞬即逝。在即将毕业之际，虽有喜悦，更多的却是不舍。生命中有这么多珍贵的感情：师生情、友情、亲情和爱情，我多么感激所经历的这一切。这五年的经历将永远刻在我的心中，鼓励着我继续前行。在未来的日子里，在工作上我将继续努力从事科研工作，希望能够做出更多更好的成绩。

杜亚娟
2012级博士生
研究方向：系统及软件容错，依赖关系，耦合及容错代价评估
Email：dyjxtu5@126.com