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1期 总第28期 2017年03月

封面人物：张凡——把握时间，全力以赴

2017年度实验室各研究方向规划

Actor模型漫谈

CGCL学习感悟

http://grid.hust.edu.cn

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：wwuhust@hust.edu.cn
网 址：http://grid.hust.edu.cn

实验室第八次学术委员会照片

2017年实验室新春联欢会 风采

不忘初心，开拓进取

——2017年度实验室研究方向科研规划

一元复始、万象更新，2017年伊始，我们本着“预则立”的初心，在取得丰硕成果的2016年基础上，规划2017年的研究方向和研究计划，力争再攀科研高峰，在十三五期间谋划实验室在国内和国际学术界的立足和发展方向。

随着数据量的不断增长以及计算机计算能力的持续提升，计算机的发展已经跨步进入人工智能时代。人工智能的发展冲击着实验室传统的研究方向。2016年年终教师学术年会从“AI by”和“AI for”的角度充分探讨了实验室如何在人工智能时代立足。结合实验室在云计算与移动计算、系统软件与体系结构、网络空间安全、大数据四个主要研究领域的基础和优势，提出了“不忘初心、开拓进取”的指导思想，在发挥实验室传统研究优势基础上，不脱离自身擅长的研究方向，结合人工智能的需求，在并行与分布式计算以及系统结构等领域做出新的有创新性、影响力的科研工作。本期季刊将从实验室四个研究方向出发，向读者展示实验室在新一年的具体科研规划。

在云计算和移动计算方向，2017年将重点研究容器云关键技术、虚拟环境I/O调度、云环境下流计算性能优化、数据中心网络虚拟化加速、跨域数据中心任务调度、公有云容器服务测量、城市计算关键技术、云雾混合环境下性能优化等。针对容器云计算，主要研究容器性能隔离技术和容器故障隔离技术；针对虚拟环境I/O调度，主要研究虚拟环境中SSD友好的I/O调度系统；在云环境下流计算优化方面，主要研究流计算系统在数据倾斜导致的慢结点、负载峰值下资源短缺以及新型器件FPGA下流计算系统优化；在数据中心网络虚拟化加速方面，主要研究协同数据平面开发工具(Data Plane Development Kit, DPDK)和网络功能虚拟化(NFV) 的网络加速分布式系统；在跨域数据中心任务调度方面，主要研究基于Apache Spark 2.0系统实现能够感知广域网复杂性的数据处理框架和任务调度机制；针对公有云容器服务，设计和实现一个跨平台的服务测量系统；针对城市计算，主要研究多源异构数据融合计算、海量时空数据计算优化以及数据关系的分析与推理；最后，针对云雾混合环境，主要研究实时网络流数据包调度策略以及对需要迁移的流数据提出有效缓存和状态迁移策略。

在系统软件和体系结构方向，2017年将重点研究面向大数据处理的异构平台及系统软件，包括异构平台上的图计算系统、面向虚拟化场景的异构内存系统和异构平台上的深度学习系统三个方面。在面向图计算的异构平台及系统方面，主要研究异构图计算环境下的新型运行时系统、异构并行编程框架、异构内存数据管理和典型异构图计算应用示范；在面向虚拟化场景的异构内存系统方面，主要开展虚拟化环境下基于效益的异构内存动态分配机制，异构内存环境下面向NFV应用等云计算需求的虚拟机定制优化和基于RDMA的异构内存池优化机制等研究工作；在面向深度学习的异构平台及系统方面，主要研究异构平台上深度学习应用的执行方式和内存资源管理策略等。

在网络空间安全方向，2017年的研究重点是围绕着云计算安全的一些关键问题开展研究，包括：软件漏洞检测与分析、软件定义网络安全、大数据安全与隐私保护以及区块链技术的研发及产业化。针对软件漏洞检测与分析，主要研究多级漏洞检测机制、降低误报率的漏洞静态检测方法以及基于静态和动态相结合模糊测试的漏洞检测方法；针对软件定义网络安全，主要研究不同网络功能的可编程配置、SDN与NFV融合框架以及网络规则冲突检测和解决方案；针对大数据安全与隐私保护，主要研究基于快速在线身份认证 (Fast IDentity Online, FIDO) 协议的多样认证模式、泛在网的数据安全收集与搜索以及加密机制的耦合与解构；针对区块链技术，主要研究区块链的权限管理、区块链的并发机制、多链交互的扩展机制以及智能合约的可靠、可控机制等。

在大数据方向，2017年的研究重点是大数据基础算法、大数据处理支撑技术、大数据技术应用以及典型大数据行业应用解决方案。在大数据基础算法方面，主要研究大规模图上低复杂度并行和分布式算法，包括静态图中低复杂度图分析分布式算法以及动态图中图性质维护的并行算法等；针对大数据处理支撑技术，主要研究面向数据密集型应用的内存管理优化与I/O优化，大数据流处理技术，大规模图计算与图挖掘，大数据搜索以及抗隐私分析的大数据查询处理优化；在大数据技术应用方面，主要研究基于大数据分析的细粒度动作识别；在典型大数据行业应用领域，主要研究医疗健康大数据智能分析平台、移动通信大数据高效查询处理与挖掘方法以及航空大数据关联分析与价值提取云服务平台。

衷心祝愿实验室在新的一年圆满完成各项科研任务，并在此基础上取得更大的成就！

华强胜

二〇一七年三月

主　编：金　海
本期执行主编：华强胜
编　委：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、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
（按姓氏拼音排序）
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

热点

封面人物

专栏

声音

动态

推荐

交流

谷歌布局大数据：开源平台Apache Beam正式发布

（陈 飞 整理）

2017年1月10日，Apache软件基金会对外宣布，万众期待的Apache Beam在经历了近一年的孵化之后终于毕业。这是大数据处理领域的又一大里程碑事件，谷歌终于完成了对其云端大数据平台Cloud Dataflow开源的承诺。

Apache Beam（原名Google DataFlow）是Google在2016年2月份贡献给Apache基金会的Apache孵化项目，被认为是继MapReduce，GFS和BigQuery等之后，Google在大数据处理领域对开源社区的又一个非常大的贡献。

随着分布式数据处理不断发展，新的分布式数据处理技术也不断被提出，业界涌现出了越来越多的分布式数据处理框架，从最早的Hadoop MapReduce，到Apache Spark，Apache Strom，以及更近的Apache Flink，Apache Apex等。新的分布式处理框架可能带来更高的性能、更强大的功能、更低的延迟，但用户切换到新的分布式处理框架的代价也非常大：需要学习一个新的数据处理框架，并重写所有的业务逻辑。解决这个问题的思路包括两个部分，首先，需要一个编程范式，能够统一，规范分布式数据处理的需求，例如，统一批处理和流处理的需求；其次，生成的分布式数据处理任务应该能够在各个分布式执行引擎上执行，用户可以自由切换分布式数据处理任务的执行引擎与执行环境。Apache Beam正是为了解决以上问题而提出的。

Apache Beam的主要目标是统一批处理和流处理的编程范式，为无限、乱序、web-scale的数据集处理提供简单灵活、功能丰富以及表达能力十分强大的SDK。Apache Beam项目重点在于数据处理的编程范式和接口定义，并不涉及具体执行引擎的实现，Apache Beam希望基于Beam开发的数据处理程序可以执行在任意的分布式计算引擎上。Apache Beam主要由Beam SDK和Beam Runner组成，Beam SDK定义了开发分布式数据处理任务业务逻辑的API接口，生成的分布式数据处理任务Pipeline交给具体的Beam Runner执行引擎。Apache Beam目前支持的API接口是由JAVA语言实现的，Python版本的API正在开发之中。Apache Beam支持的底层执行引擎包括Apache Flink，Apache Spark以及Google Cloud Platform，此外Apache Storm，Apache Hadoop，Apache Gearpump等执行引擎的支持也在讨论或开发当中。

TensorFlow与Apache Spark结合：雅虎开源“TensorFlowOnSpark”

（易宇声 整理）

雅虎于2017年2月13日宣布开源TensorFlow OnSpark。它使得深度学习框架TensorFlow能与Apache Spark中的数据集兼容。对于使用Spark来处理不同类型数据的机构和开发者来说，这无疑是一个好消息。TensorFlowOnSpark的开源代码，已基于Apache 2.0协议在GitHub上发布。

众所周知，深度学习有海量数据需求。许多公司利用Spark对超大规模的数据集进行管理。让深度学习框架直接、方便地获取这部分数据，将为ML开发提供极大助力。

雅虎在官方博客中宣布了这一消息，并解释了此前雅虎Big ML开发团队遇到的问题：“现有的深度学习框架，往往需要设立单独的深度学习数据组。这强迫我们为同一个机器学

习流水线创建多个程序。维护多个独立的数据组，要求我们在它们之间传输海量数据集——这导致不必要的系统复杂性和端到端的学习延迟。”

为解决这一问题，雅虎此前开发了Caffe OnSpark。它使得基于Caffe机器学习框架开发的程序，能与Apache Spark兼容。雅虎已将基于CaffeOnSpark的程序，用于鉴别搜索中的不恰当搜索结果，以及自动探测电子竞技游戏直播视频中的关键看点。

雅虎在2016年开源了CaffeOnSpark，如今它对TensorFlow做了同样的工作。两者的原理几乎相同，只是把机器学习框架换成了TensorFlow。雅虎表示，把TensorFlow程序移植到TensorFlowOnSpark相对方便，并经过反公司内部的反复验证。“这通常只需要修改十行以内的Python代码。许多使用TensorFlow的雅虎开发者已轻松地把TensorFlow程序，移植到TensorFlowOnSpark执行。”

英特尔芯片 BTB 组件漏洞，60 毫秒可绕过 ASLR 防护

（唐晓兰 整理）

来自两所美国大学的信息安全研究人员发现，利用 Intel Haswell CPU 架构中 BTB 组件的漏洞可以快速绕过 ASLR 保护。ASLR 是大多操作系统（Windows、Linux、macOS、iOS 和 Andriod）长期使用的系统防御机制，研究人员已经在 Intel Haswell CPU 的 Linux 设备上利用漏洞做了测试。

研究人员在研究报告中解释道，“攻击所花时间非常短，只需 60 毫秒就足够收集必需的样本数。”

ASLR 保护（地址空间配置随机化）可以让系统免受普通攻击，例如缓冲区溢出、返回导向编程技术（Return-oriented programming，ROP）攻击等。ASLR 通过随机安排计算机内存进程关键数据区的地址空间位置进行保护，使得“劫持式攻击”无法进行。

但在本周发布的研究报告中，专家称，他们发现 Intel 芯片的 BTB 组件中存在漏洞：CPU 使用 BTB 加速操作，运作方式就如同浏览器缓存加速常访问的网页。

研究人员表示，BTB 易遭受碰撞攻击（用随机数据轰炸 BTB，直到它们发现相同的数据已经储存在缓冲区）。研究人员利用特殊的软件程序，绕过了 ASLR 可实现系统级攻击。该技术允许研究人员从包含 ASLR 索引表的 CPU 内核恢复数据，允许攻击者了解具体应用代码执行位置，以便利用漏洞并进行不断调整式破解。从理论上讲，同样的攻击应该应该对其它 OS，甚至 KVM（内核虚拟机）有效。

研究人员称可行的解决方案是鼓励操作系统厂商在代码功能层面实现 ASLR 保护修复漏洞。

生物识别与量子密钥：银行安全的未来

（唐晓兰 整理）

银行业务依赖于保证存款安全，而黑客总能找到更高级的方法来染指客户存款。但世界顶尖的研究机构展现了银行技术的未来，包括量子密钥分发和生物识别技术。

（1）量子密钥分发

当今实时银行交易的密码保护依赖于计算能力的数学方程式，因为这些密钥要破解得花

几个世纪。当前加密密钥通常是间歇性修改的，因为频繁改换对牵涉其中的每个人都是个大麻烦。

量子计算的兴起，颠覆了传统认知。其强大的计算能力，意味着加密可在相对较短的时间内被破解。未来安全团队的帮手就在眼前。BT和东芝联合研发的量子密钥分发，可以通过利用物理定律使数据加密几乎不可破解的方式来缓解该威胁。加密信息以每秒上10Gb的速度由单光子承载着从光纤中飞过。如果需要的话，密钥可每毫秒检查一次，这样一来，量子计算机想要获得有意义的数据量，可能就得破解上万亿密钥。

监听设备会监视分接器上任何溜进传输造成中断的迹象，这在到达时间上会反映出来。量子物理定律决定了，仅需观察，就可改变量子世界中的任何东西。这就是量子分发为什么是密钥传输理想方式的原因所在。想闯进传输偷看数据的人，基本上只是在破坏数据而已。

任何数据泄露都能被当场发现，然后交易立即被取消以避免灾难。由于可以高速交换数据，未被破坏的密钥仍可从储备目录中取用，而新密钥则被毫无中断地自动发送。

（2）生物识别安全

BT的零售银行分行模型的柜台上，装有富士通生物识别手掌读取器，可从手掌接触来验证客户身份。将手掌放到上面，该读取器就会发出对血液中血红蛋白反光敏感的红外光线，扫描皮肤下的静脉。传感器会记录下扫描者的模式，并与数据库中的记录进行比对，然后再决定是否授权。更进一步的生物识别在该中心也有展示。一家名为Iproov的初创公司研发了该面部生物识别工具，可以通过智能手机摄像头来验证身份。把脸放到智能手机屏幕的一个圈里，让一系列彩光扫过面部细节。类似设备只分析一幅图像，但这个工具就更进一步，把一段视频发送给服务器。企图用照片来骗过识别是不起作用的，因为它反映不出立体面部的三维反光。做个一流水准的半身像或许能欺骗该设备，但这半身像的品质怕是世上少有人能做出。

语言智能

（李金朋 整理）

语言智能（Language Intelligence），是语言信息的智能化，是运用计算机信息技术模仿人类的智能，分析和处理人类语言的过程。作为人工智能范畴的专门术语“语言智能”，由首都师范大学教授周建设2013年在北京语言智能协同研究院成立大会上首次正式提出。2016年人工智能得到飞速发展，震惊世界的AlphaGo再度引起世人对人工智能的关注。作为人工智能的重要组成部分及人机交互认知的重要基础和手段，语言智能研究在2016年也取得了令人瞩目的成就。

机器翻译

在“互联网+”时代，资讯在不同语言之间即时翻译转换成为越来越迫切的需求。机器翻译运用计算机程序，快速实现大规模的一对一、一对多、多对一和多对多等不同语言之间的互译。目前，谷歌翻译能支持103种语言之间的互译。2016年9月底，谷歌推出“神经机器翻译系统”，标志着机器翻译领域取得又一重大技术突破，大大提升了翻译的准确率。目前国内有包括百度在内的多家公司和高校科研团队致力于机器翻译研究，并在中文相关翻译上取

得一定的领先优势。深度神经网络（DNN）的运用，极大提升了机器学习的质量。积极探索DNN的软件“神经元”效能，为多语种机器翻译提供策略保障。我国机器翻译当务之急是要建设高质量的以汉语为中间语言的多语言平行语料库，重点建设文化、经济、政治等重点领域的数据库，同时构建相应的句法、语义、语用的模型，加强对模糊结构和语义的识别能力，提高翻译质量。

语音识别与语音合成

语音识别和语音合成是实现人机语音通信、建立有听说能力的口语系统所必需的两项关键技术。科大讯飞公司已经掌握了世界顶级的相关技术，并连续多年在诸多全球顶级专业赛事中证明了自己的过硬实力。语音智能核心技术不仅在中文语音识别方面保持绝对领先，在英语语音识别方面同样达到国际领先水平。目前，科大讯飞已推出多种产品，可以满足从大型电信级应用到小型嵌入式应用、从电信金融等行业到企业和家庭用户、从PC到手机到MP3/MP4/PMP等不同应用环境。智能语音系统的另一个重要组成部分——语音评测技术，是通过机器自动对发音进行评价，对发音偏误进行定位和分析的软件系统。科大讯飞的语音评测系统是目前业界唯一一个通过国家语委鉴定并达到实用水平的语音评测系统，已经运用于普通话水平测试和英语四六级口语测试，对提高汉英口语教学水平发挥了重要作用。

智能批改

20世纪末，基于统计和人工智能的英语作文自动批改技术在自动评改系统中得到应用，其中E-rater技术更是广泛运用于美国的GRE、TOEFL等大型考试以及中小学作文测试，在信度和效度方面取得了长足的进步，在某些方面可以比人工评阅做得更好，效率更高，更客观。北京语言智能协同研究院主导开发的英语作文批改系统，以全国大中小学学生英语作文为评测对象，截至2016年12月已批改英语作文2.83亿篇，服务6000多所学校，1300余万学生。2016年12月11日，中国语言智能研究中心推出了自主研制的汉语作文智能测评系统和汉语写作教学综合智能训练系统。汉语作文智能测评系统具有“打分、评级、纠错、范例”四大功能，坚持“规则+统计”的交互式作文自动批改和反馈，针对国内小学生汉语作文、海外华裔学生汉语作文和留学生汉语作文进行自动批改。汉语写作教学综合智能训练系统高度重视写作教学过程中的师生交互，重视写作过程的管理、监测和指导。自动批改系统客观高效、不受时空限制，既适用于辅助教师教学，也适用于学生自主学习。

智能写作

智能写作是利用人工智能技术，由机器创造性地生成自然语言的过程。从2014年开始，包括美联社在内的美国多家媒体已经开始采用智能机器人自动写作的新闻稿件。2015年9月，腾讯财经、新华社股票新闻等也陆续上线发布“机器人写手”的文章。首都师范大学中国语言智能研究中心于2016年5月发布了自主研发的汉语智能写作系统。该系统能够自动收集大数据文档，并进行整理、提取、过滤、筛选、组装，最后生成相关文章，可以完成NBA体育赛事新闻、运动会开幕词和高校学术活动总结等受限语体写作。未来汉语智能写作将向金融新闻、多种类型应用文书及中小学生辅助写作等领域扩展延伸，成为语言智能领域的“新高地”。

智能问答

智能问答系统是应用信息检索和自然语言处理技术自动分析用户提问、辨识用户意图并提供精准答案的系统，是对使用搜索引擎通过输入关键词进行搜索的传统方式的突破。智能问答系统需要多项自然语言理解技术，从问题分类到需求解析，从网页检索到知识库查询，从信息抽取到答案排序，每一个环节都需要达到很高的精度，才能使得最终搭建起来的问答系统准确地理解问题并给出答案。IBM研发的问答机器人Watson在美国智力竞赛节目Jeopardy中战胜人类选手，苹果公司研发的Siri系统在智能手机中的应用取得了良好效果。近年来，以智能问答技术为核心的智能聊天机器人发展得如火如荼，除了已面世的微软公司的“小冰”、百度公司的“度秘”和华为公司的“小诺”等问答系统外，其延伸产品也研发成功，如陪读机器人、陪聊机器人等。中国语言智能研究中心已完成支撑智能问答的理论和技术的研究并运用于产品研发，2016年10月发布了儿童陪伴教育智能机器人“薄言豆豆”，自主研发的语义解析技术让豆豆能够准确理解对话意图，实现情景对话和知识问答融合的多轮对话。语言智能已经成为人机交互的重要手段，有力地促进了语言教学、语言学习的智能化。语言智能拓展了语言学研究的新领域，在未来科技、教育发展中将发挥越来越重要的作用。

谷歌“即时应用”正式上线 小程序的春天

（谢宏亮 整理）

去年5月的Google I/O大会上，谷歌发布了Android Instant Apps（即时应用），而近日，Google宣布Android Instant Apps正式上线，BuzzFeed、Wish、Periscope以及Viki等少数几款应用已经可以体验了。

据了解，与微信刚刚推出的小程序类似，谷歌给出的Android Instant Apps五大特点：运行Android应用而无需安装、从任意位置访问应用、以Google Play服务为基础、适用于大多数Android设备、升级现有应用。比如朋友给你的安卓手机应用发来一个链接，假设是某摄影用品商店上的商品。而恰好该商店的Android应用也支持了Instant Apps。你点击了这个链接，就直接进入了该商店的Android应用，即便手机并没有安装它。

谷歌解释道，Android Instant Apps本质上其实是一个模块化的APP，将开发者的App模块化了之后，Google Play只会在你点击了链接之后下载所需的那部分功能，这样就相当于在一个简洁的模式下使用该App。

·对于消费者而言，其能在某种程度上节约手机内存，更持久地保持手机的流畅性。

·对于开发者来说，这也可以节约软件开发周期，他们也不必为此专门再去开发一个独立的App，它使用的还是同样的Android API和同样的源代码，开发者们只需要在原有App的基础上进行一下升级，接入Android Instant Apps的功能就可以了。此外，移除了传统APP必须下载安装这一步骤之后，企业也更有机会将“偶遇者”变成用户。

不过，如果想要支持这项功能，开发者的首要任务是去精简自己的应用，并且将其“模块化”，拆分成各不大于4MB的独立模块。官方还给出了一个例子，像是购物应用，就可以拆成浏览，搜索，商品消息等模块。

连谷歌也在做“小程序”，看来未来真属于小程序了？

体系结构顶会HPCA2017英特尔夺魁

（李陈希 整理）

2017年2月8日，HPCA 2017本年度的最佳论文已经公布，来自英特尔的论文《Near-Optimal Access Patitioning for Memory Hierarchies with Multiple Heterogeous Bandwidth Sources》摘下桂冠。近年来，内存技术——比如嵌入式DRAM（eDRAM）和高宽带存储——的进步，使得在CPU上集成大型存储器成为可能。由于容量的限制，这些存储一般都会被当成一个内存侧的缓存。受到传统思路的启发，许多致力于提升系统性能的优化都在尝试将内存侧的缓存的命中率最大化。一个更高的命中率能让人更好地利用缓存，进而被认为能带来更高性能的表现。在此篇论文中，来自英特尔的研究团队对这种传统的思路提出了挑战，并提出了一个动态存取分割（Dynamic Access Partitioning）算法，也称DAP。该算法通过牺牲缓存的命中率来利用主存储器中未被利用的带宽。DAP通过使用一个轻量的学习机制，只需要额外16个字节的存储空间，便能在内存侧缓存和主存之间获得一个近似最优化的带宽。模拟的结果显示，在一个片叠式内存侧DRAM缓存上采用DAP，可以获得13%的性能提升。

Intel公布Stratix 10 MX：单片FPGA实现1 TB/s存储器带宽

（杜 铁 整理）

到2020年，数据中心数据量将达到1ZB，有500亿互联设备，出现8K视频广播和400G网络系统。英特尔的Stratix® 10 MX器件采用了DRAM系统级封装(SiP)，实现了：更高的存储器带宽、更低的功耗、更小的外形封装、使用方便、增强嵌入式SRAM。

Stratix 10 MX结合了灵活的可编程Stratix 10 FPGA和SoC以及3D堆叠宽带存储器2(HBM2)。HyperFlex™ FPGA内核架构的体系结构支持1 GHz内核架构，高效的利用了封装内存储器区块的带宽。DRAM存储器区块使用英特尔的嵌入式多管芯互联桥接(EMIB)技术物理连接至FPGA。

Stratix 10 MX器件含有大量的系统级集成，包括：集成了高达1.5 GHz的四核64位ARM® Cortex®-A53硬核处理器系统；通路数据速率高达56 Gbps的异构3D SiP收发器区块；IEEE 754兼容单精度浮点数字信号处理(DSP)吞吐量达到每秒10万亿次浮点运算(TFLOPS)；安全功能最全面的安全器件管理器(SDM)；90 Mbits嵌入式SRAM (eSRAM)增强了现有模块RAM；突破性的HyperFlex体系结构将内核性能提高了2倍。

Stratix 10 MX器件在内核架构旁集成了HBM Gen2存储器。这种安排显著缩短了内核架构和存储器之间的互联，从而降低了传统上驱动长PCB走线所需的功耗。接口运行的速度也相对较低(2 Gbps)，因此，降低了功耗。走线未进行匹配，容性负载较小，降低了I/O电流。最终结果是更低的系统功耗和更优的每瓦性能。

由于Stratix 10 MX封装集成了存储器组件，因此，PCB设计降低了布线复杂度。这种实现方法减小了外形封装，支持简单使用模型。最终结果是非常灵活的可扩展解决方案，而且使用非常方便。

Stratix 10 MX器件通过嵌入式SRAM (eSRAM)提供快速通路低延时片内存储器。eSRAM进一步完善了已有的模块RAM。

把握时间，全力以赴

2016年11月8日，ICNP 2016（The 24th IEEE International Conference on Network Protocols）在新加坡召开。实验室博士生张凡的论文“Piggyback Game: Efficient Event Stream Dissemination in Online Social Network Systems”被会议录用，并在大会上进行了宣读。ICNP是计算机网络领域顶级国际会议之一。本刊对论文作者张凡博士就科研的经验和感悟方面进行了采访，以下以第一人称来进行陈述。

前序

很荣幸成为本期实验室的受访人物，非常惭愧难以像往期各位学长那样提供详尽的科研“攻略”，只能提供自己两年多的学术训练中形成的一些基本认识给各位同学参考，其中有些我已有切身感受，有些还需要在后面的学业中继续深入实践和体会。

尽快进入状态

记得最早获得保研名额选择陈老师作为导师时，他问过我为什么要选择读博士。我说我觉得知识储备不够想要进一步地学习知识。陈老师认真地对我说“其实读博士的目标不是学习知识，而是要创造知识”。当时我还不怎么理解，现在看来，这话是无比正确的，博士的目标是做有意义的研究，发现问题并予以解决，最终创造出新的知识。

这并不是说博士就不需要去学习知识，只是不能像本科阶段那样一味抱着学习已有知识的态度，要尽快明白其最终目标是创造新的知识。博士刚入门会有些迷茫，尤其像我一样的直博生，刚踏出本科的大门，要将学习知识的思维扭转到学术创新的思维需要一定时间和功夫。然而即使开始还不知道如何去创新，容易做到的是先抛开仅从书本、课堂来学习的惯性，多方面地去进行学习和积累，以尽快的完成从本科到研究生的角色过渡。

开始接触科研的第一关是读论文。一开始阅读论文的时候，很容易因为知识的局限难以深入理解，于是囫囵吞枣地只读懂了别人干了什么工作，设计了什么结构或者算法。实际上，读论文并不仅仅是进行阅读理解。一篇好的论文呈现着一个完整的科研工作，也是博士可以去全面学习的对象：例如论文的Introduction部分介绍问题本身的背景、意义、研究动机，也是我们在研究中应首先思索的问题。要做的问题是不是一个重要的问题？这个问题是不是具有挑战性？要解决这个问题的关键点是在哪？Related Work部分则代表着对于问题的详细调研，当前有哪些研究者在这个问题上做过努力？分别从哪些思路进行了思考？他们做到了什么地步又还存在怎样的问题没有解决？我们只有把这些基本的方面都抓住了，才能有效展开后续详细的工作，包括如何设计系统或算法来实现思路，如何用理论或实验来验证自己思路的有效性，如何总结自己的工作成功或不足的地方与未来工作的展望。

所以，一篇好的论文绝对不仅仅是教会你一个小的技巧，一个特定的算法，真正读好一篇文章也没有那么容易。从问题本身来说，你甚至可以通过一篇文章的参考文献不断去追溯

相关问题的历史研究脉络，把握这个领域的发展历程和现状。从其他方面来说，你还可以学习作者如何组织思路、如何设计实验、如何使用简洁精准的语言进行表达等等。

不仅是论文，我们还可以珍惜每一次参加学术会议、学术报告甚至小组研讨会的机会。听别人认真宣讲一篇论文，就省去了你精读一篇论文的宝贵时间；听专家分享一个领域的最新成果，甚至能深入与其探讨，就节省了大量摸索总结的精力。即使是并不完全相关的领域，对一些问题的思考方法或许也可能是相通的，未免不能用来学习借鉴，正所谓他山之石，可以攻玉。

在思考和实践中前行

进入科研后，最为关键的是如何发现问题和解决问题。在前期，对想要研究的领域了解还不是很深入，研究现状把握还不是很准确的时候，可以通过相关领域最新的论文先看看别人在研究什么问题。借鉴别人对问题的详细分析和论述来进一步思考是否还存在尚未解决的问题，自己是否可以在此基础上提出新的想法。一个可以尝试的方法是仅仅精读一篇文章Introduction部分关于问题的描述，然后抛开文章，自己开始思考是否对作者提到的问题完全理解，这个问题到底建立在什么场景下，这个问题有多严重，如果不解决有什么后果，问题的挑战在什么地方，如果自己做这个问题有什么解决思路。经过一番思考后，再继续仔细阅读文章，边读边缜密地思考作者是否完全解决了问题，还存在什么局限，这样读文章就可能产生欣赏、共鸣或质疑。

经过初步的学习和思考的过程，对现有工作有了较为准确的认识，还应再进一步去思考什么是这个方向上最本质的问题，本质的问题往往更需要结合实际应用和系统。在实际应用和系统中，什么才是最关键最亟待解决的问题。实际问题中，往往不会存在各种理想的模型，完美的数据分布，关键问题可能隐藏于系统的某种实际特征之中。这时，要洞察问题往往需要动手对系统或应用数据进行去实践分析。

精心雕琢

在科研中，找到好的问题和idea是你工作的硬实力，是我们必须着重提高的能力。而当有了好的想法后，如何去展现它们则是你的软实力体现，这就包括如何表达你的研究动机和贡献，如何通过严密的理论和可信的实验验证你的idea的可行性和效果，最后如何完整地以论文的形式展现出来等等。从思路成型到完成论文的过程中，反复的思考和修改至关重要。无论是对算法的描述、实验的细节、还是图表的绘制，都要仔细斟酌怎样能最准确简洁地表达出你的想法。在领域顶级会议投稿的激烈的竞争下，即使是自认为完美的论文被拒掉也是常态。你可以用买彩票的心态对不专业的审稿意见一笑了之，但你绝不能以买彩票的心态去投一篇没有充分准备好的论文，投稿前要力求尽善尽美，因为这些软实力不足的原因导致论文被拒是非常可惜的。

与时间赛跑

一篇好的文章被接受，需要投入大量的精力。因此科研中，最宝贵的就是时间。你需要时间去积累经验和知识、发现并解决问题、验证自己的想法、雕琢自己的工作，但这不意味着你可以慢慢来。科研成果具有很强的时效性，尤其是在计算机这门学科上面。即使你很

好地把握节奏，在精细地雕琢你的作品的同时，还需要预防不可控的“撞车”。在一个重要的研究问题上，即使你找到了很好的想法，当你正在论证测试自己的想法的同时，可能类似的甚至更好的想法正在诞生甚至发表。在同一问题上一旦新作问世，你只能选择将你长时间的努力丢弃或者非常艰难地再次提升。因此，瞄准最新的重要问题，与时间赛跑其实是最省力的途径。越是可能取得重要的科研成果，时间可能越是不够用，因此时刻都要保持适度的紧迫感。在发现重要问题时，绝不要吝惜集中投入精力与时间。

尾语

科研过程中，全力投入但久久没有思路时会感到“山穷水尽疑无路”的迷茫，突破困难产生好想法时却会有“柳暗花明又一村”的欣喜，这也许就是科研给我的最大奖赏。科研工作虽然如一场赛跑，但是也不要忘记适当调整节奏。冥思苦想得不到结果的时候不如放松片刻，闲暇之时也不要忘记思考困扰的问题。在紧张和放松之间找好自己的平衡，好好把握和利用时间，将最好的科研结果展现出来，才能不负自己在博士阶段的宝贵时光。

张　凡
2014级博士
研究方向：分布式流计算
Email：zhangf@hust.edu.cn

注意！你电脑的LED灯正在“泄密”（新华社）

台式电脑的主机上一般都有发光二极管（LED）指示灯，这种指示灯通过闪烁显示电脑正在进行指令输入或读取。不过，你能想象得到这种微弱的闪烁也可能泄露大量机密么？

据“今日俄罗斯”电视台网站2月23日报道，以色列本－古里安大学的研究人发现，黑客通过解读LED指示灯闪烁频率，可以获取与网络物理隔离电脑中的机密。物理隔离的电脑可以说是比较安全的电脑，因为它既不和外界的因特网连接，也不和公司或机构内部的网络相连。不过，如果有人把携带病毒软件的移动硬盘或其他存储设备连接在这种电脑上，那么存储设备中的病毒会控制LED指示灯的闪烁。通过闪灭，LED灯可发出二进制信号，从而传递出信息。闪烁信号传输速度最快可达每秒4000比特。

存储设备中的病毒会控制LED指示灯的闪烁。

安有感应装置的微型无人机飞到电脑所在房间的窗外，记录LED指示灯的闪灭，随后，黑客就能破译出电脑所包含的密码、加密密钥等机密信息。

有时，黑客根本不用派出无人机。他们可以入侵某一机构的闭路监控系统，通过监控摄像头直接对LED指示灯的闪烁方式和频率进行收集和解读。

本－古里安大学网络安全研究中心研究与发展部门负责人莫迪凯·古里参与了本次研究。他在接受美国网络杂志《连线》月刊采访时指出：“这种入侵电脑的方式非常隐秘。因为如果电脑感染病毒，通常没人会注意到LED指示灯的闪烁方式有所改变。”

虽然这种入侵电脑的手段很难被察觉，对付它的方法却很简单。研究人员建议，电脑的使用者可以用不透明的胶布粘在LED指示灯上，或直接将其和电脑断连，从而防止LED指示灯向黑客泄密。

云计算与移动计算组研究规划

吴 松、刘方明、余 辰、顾 琳、陆 枫、王多强

本方向在2017年的研究重点包括容器云关键技术、虚拟环境I/O调度、云环境下流计算性能优化、数据中心网络虚拟化加速、跨域数据中心任务调度、公有云容器服务测量、城市计算关键技术、云雾混合环境下性能优化等。主要研究规划和重点关注的研究内容如下。

（1）容器关键技术研究

容器虚拟化技术通过共享宿主机内核创建多个虚拟用户空间，提供相对于传统虚拟机的轻量性，但是宿主机上的资源无法全部进行抽象和隔离，导致容器的虚拟化实现并不彻底，这个问题长期以来严重制约着容器的进一步发展。针对这一现状，拟研究和突破包括容器间性能隔离、容器故障隔离和容器在线迁移等一系列容器虚拟化的关键技术瓶颈。具体内容如下：1）容器性能隔离技术。拟解决现有容器技术中资源限制的不完整性，实现包括磁盘定额和网络带宽在内的更为完善的性能隔离；2）容器故障隔离技术。现有容器技术主要使用namespace机制，一旦某一容器的崩溃破坏了未隔离的信息，将导致故障的级联，甚至引起整个宿主机系统的崩溃。针对这一情况拟对未隔离的信息进行严格的访问控制，从而避免单容器故障传播，提高容器系统的可靠性。3）容器在线迁移技术。由于容器资源隔离的问题，现有的容器在线迁移技术在迁移前后文件系统存在不一致的问题，拟完善容器的在线迁移技术，实现秒级的容器在线迁移。

（2）虚拟环境中SSD友好的I/O调度系统

云服务提供商利用虚拟化技术将物理主机虚拟成多个虚拟机提供给用户，用户根据需求在虚拟机运行不同类型的应用。由于虚拟机内和虚拟机间的两层并发，提高系统性能的关键因素是，实现应用间的I/O带宽公平性的同时尽量提高总带宽。用SSD取代HDD可以大幅提高系统的总带宽，但是应用间的公平性仍然无法保证。拟设计I/O分析模块，在保证透明性的前提下，根据I/O的访存特征对虚拟机内的应用进行识别，为下一步在主机层对虚拟机内应用进行调度做准备。然后，根据I/O分析模块得出的应用特征，分析出带宽侵占性强的应用，通过对此类应用的请求控制和对NCQ队列进行调节来实现应用间带宽的公平性。

（3）云环境下流计算系统性能优化

在大数据时代，有越来越多大数据应用和使用场景比如高频金融交易、故障监测、传感器网络等要求快速的实时响应。如何有效、实时处理大数据，快速提取大数据中的价值成为业界研究的热点。流计算成为大数据实时处理的主要范式。对数据流的实时处理不仅要求流计算系统能够提供低响应延迟，同时需要兼顾高吞吐、高容错、精确一致性执行语义以及在出现数据倾斜和突发峰值负载场景下系统的健壮性和稳定性等。然而，当前主流的流计算系统无论是基于操作符的连续流计算系统还是基于微批处理的离散流计算系统都不能很好的适应这样的需求。针对这一现状，我们主要开展的研究方向包括：1）针对流计算系统在数据倾斜和异构集群环境下慢结点严重影响性能的问题，探索基于流作业运行时特征的慢结点主动避免机制，保证流计算系统的稳定性；2）针对流计算系统在负载峰值和谷值情况下资源短缺和浪费的问题，探索基于容器的流计算系统弹性资源分配策略，提高资源利用率；3）针对流

计算系统在延迟和吞吐量上无法兼顾的矛盾，探索基于流水线和微批混合模式下的新型流处理模型；4）研究在异构环境下利用新型器件（如FPGA）加速流计算系统的机制和方法。

（4）协同DPDK和NFV网络加速的分布式系统

近年来，随着云计算应用用户数量的急剧增长，数据中心分布式应用需要承载越来越多的数据请求服务。据Facebook报道，该公司的memcached服务每秒需处理10亿条以上的请求，其中90%的请求数据量不足32KB，50%的请求小于4KB。然而，当前主流的Linux网络协议栈在处理高并发、小请求的网络流时性能较差。针对当前分布式Key-value Store系统高并发、小请求情况下网络性能较差的问题，拟提出利用网络功能虚拟化（NFV）节点组合客户端请求，使服务端能够批处理大量请求，从而提升服务端网络性能、降低服务端协议栈开销。由于NFV服务基于DPDK快速包处理技术，仅包含一个精简网络协议栈，因此能够高速处理大量服务请求，极大提升分布式系统的性能。包括：通过测量分布式Key-value Store系统在处理不同大小请求、不同并发量时的网络性能，分析Linux网络协议栈的性能瓶颈，设计适用于此类系统的请求组合策略；设计并实现网络层I/O库，包括非阻塞的write、read、epoll_wait等功能，为应用程序提供多线程、功能透明的网络请求组合、拆分服务；在集群上部署NFV服务，以及应用层I/O库，实现分布式请求组合，并测量分布式应用的性能提升比例，以及NFV服务的开销。

（5）感知广域网复杂性的大数据任务调度系统

随着Google、Microsoft等在全球各地部署越来越多的跨域数据中心，大数据正以地理分布式的形式被产生和存储。为了分析这些分布式数据集，传统的方法是将数据跨域传输汇总到一个主数据中心，然后应用单数据中心的处理框架（如Spark、MapReduce）。然而，随着数据量的不断攀升，这种集中式方法正在变得越来越耗时耗带宽成本。为了克服这些性能与成本问题，跨域数据分析应运而生，将计算分布到数据存储的地方，数据在本地预处理，然后仅在数据中心之间传输较少的中间数据。在地理分布式环境下，连接数据中心的广域网异构复杂且带宽昂贵，通常成为跨域数据分析的性能瓶颈。然而，现有数据处理框架的任务调度机制均假设网络同构。因此，拟基于Apache Spark 2.0系统实现能够感知广域网复杂性的数据处理框架和任务调度机制。

（6）公有云容器服务测量平台

随着以Docker为代表的容器技术的日益发展以及微服务应用部署框架的逐渐普及，主流的公有云运营商纷纷在自身的云环境中添加了针对容器的服务。现有的容器云服务均通过在虚拟机实例中运行容器得以实现。然而，由于底层硬件以及所采用的虚拟化技术的多样化，不同运营商所提供的容器云服务在性能和费用上均存在差异。甚至对于同一运营商，其容器服务的性能和费用在不同时间不同地域也存在差异。目前，云用户对主流运营商容器服务在性能和价格上的差异性仍缺乏深入的了解。现有的针对公有云服务的测量工作主要集中在对虚拟机性能的分析以及对竞价实例的价格和可用性的研究，而对于容器云这一新兴的云服务尚缺乏系统和全面的测量与分析。拟针对容器云服务设计和实现一个跨平台的服务测量系统，为选择和使用不同容器云的用户提供性价比量化参考和不同类型应用的部署策略。

（7）城市计算的关键技术研究

利用城市积累的大数据，分布式平台的强大计算能力，和先进数据分析技术，城市计算致力于改善城市中多种问题，例如交通拥堵、空气污染、城市规划不合理、公共安全等等。依托企业合作项目，基于城市多种时空轨迹数据，采用新颖的多源异构数据融合计算方法，

研究时空位置信息的关联关系，解决国家科技发展需求下智慧城市目标的关键技术。本项目拟选择开展以下四个方面的研究：1）多源异构数据的融合计算，需要获取城市多方面的数据，探索新颖的数据融合计算方法，学习互补增强的知识；2）海量时空数据计算的速度优化，主要是将目前单机时空数据挖掘技术和应用进行分布式并行化，主要考虑的是如何进行时空数据的存储和几何计算；3）数据关系的分析与推理，是在挖掘出事件的序列集合基础上，证明事件发生所具有的逻辑并获取相应的知识规则，进一步利用已证明的知识规则指导行动；4）城市计算需要预测精度的提升，在目前统计学+大数据模式下，可以通过数据量的增大即包含更多更详细的情况来提高预测准确度。

（8）云雾混合环境下的NFV性能优化

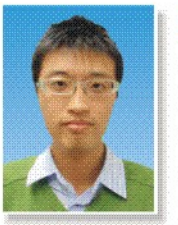
网络功能虚拟化技术消除了网络功能和硬件之间的依赖性，将专用硬件实现的网络功能以虚拟网络功能实例的形式实现，实现了网络服务的灵活调度和扩展。与此同时，随着移动互联网和计算机设备的飞速发展，雾计算端性能得到显著提升，可以有效地分担和缓解云端的服务压力。云端和雾端有着不同的特性，例如云端的资源量巨大但时延较大而雾端的资源有限但时延较小。合理地利用这些特性可以有效地降低网络服务的时延。针对这一现状，拟探索网络服务流的实时调度策略，合理地分配云雾端的物理资源，达到降低网络服务时延（包括处理时延和传输时延）的目的。拟基于云雾混合环境，开展以下几方面的研究：1）提出一个实时的网络流数据包调度策略，以降低各服务的处理和传输时延；2）针对各类网络流变化的动态事件，提出最小化成本的在线实例部署和资源分配算法；3）针对需要迁移的流数据，提出有效缓存和状态迁移策略。

目前，围绕着上述研究目标，研究小组已经部署人力开展相关研究，系统研发也正在开展，一方面拟争取能在2017年的顶级学术会议和期刊上发表研究成果，扩大研究工作的影响，另一方面将努力抓住国家科技计划实施改革的机遇，争取申请各类国家科技计划项目。同时，小组也会持续重视科研成果转化，将科研成果应用到重要云计算企业和实际的生产性环境中。

吴　松

博士，教授

研究方向：主要研究方向是云计算与分布式处理、虚拟化与系统软件、数据中心资源管理等。



刘方明

教授、博导、国家中组部“万人计划”青年拔尖人才

研究方向：云计算与数据中心、绿色计算与通信、移动互联网等

Email：fangminghk@gmail.com

余　辰

副教授

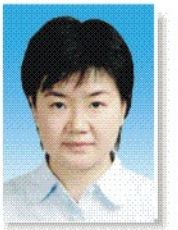
研究方向：普适计算、移动互联网

Email：yujiechen_hust@163.com

顾　琳

博士，讲师

研究方向：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。

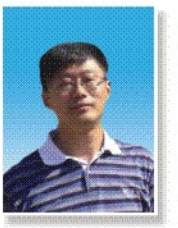


陆　枫

副教授

研究方向：主要从事高性能计算应用、移动互联网等方向的研究

Email：lufeng@hust.edu.cn



王多强

硕士，副教授

主要研究方向为并行与分布式计算，网格计算与云计算。

Email：dqwang@mail.hust.edu.c

面向大数据处理的异构平台及系统软件

——体系结构与系统软件组科研规划

廖小飞、刘海坤、蒋文斌、邵志远、郑 然、郑 龙、张 宇、吕新桥

1 背景与挑战

随着“大数据”时代的来临，计算机系统需要存储和处理的数据量急剧膨胀，使其和底层硬件平台存储处理能力之间的鸿沟日趋巨大，已对现有计算机体系结构和系统软件提出了新的挑战。具体来说，典型大数据应用，如图计算、深度学习等，表现出了与传统以计算为中心的体系结构不相适应的特性，如数据规模大、计算/访存比低、数据局部性差和迭代频繁等，导致现有计算机体系结构设计面临数据访问瓶颈和计算资源利用率低等问题，迫切需要设计面向图计算和深度学习领域的新型计算系统，以如何快速分析和处理这些海量数据，获得它们背后潜在的商业价值或者科学价值。

新型处理器件和新型存储器件的出现，为解决上述矛盾提供了新的可能。在新型处理器件方面，工业界和学术界先后推出了FPGA、GPU、Intel Xeon Phi等加速器芯片，配合CPU支持大数据处理等应用。在新型存储器件方面，工业界研发了多种非易失性存储介质，诸如相变存储器（PCM）、自旋转移力矩磁性随机存储器（STT-MRAM）、磁性随机存储器（MRAM）等，预期将逐渐成为DRAM 的替代品和伴生品，以有效缓解“数据I/O”瓶颈问题，提高大数据处理的时效性。这些新型处理器件和新型存储器件在能耗、并行度、读写速度、存储密度等方面各具优势，催生出了基于这些新型处理器件和存储器件的异构体系结构，以期望利用各器件的优势支持大数据处理，达到高性能、低能耗和高可靠等目的。

然而，这些新型处理器件和存储器件在运作机制、性能和使用方式等方面存在着较大差异。这使得异构平台及其上的虚拟化系统软件在有效管理和利用这些异构计算及存储资源，支持图计算和深度学习等典型大数据应用时，在编程模型、执行方式、数据放置和任务调度等方面分别面临着诸多挑战。

1）以图计算为代表的大数据应用给异构平台带来的挑战

异构化逐渐成为计算机体系结构的一个主流趋势。新型异构平台支持提供了超高的并行性、丰富的带宽资源以及极低的数据传输延时，为更好地支撑图数据的高效处理带来了新的契机。然而，由于各加速设备及存储资源在计算模式与访问接口上存在着明显的差异性，现有图计算系统又多集中在面向单一加速设备及存储构架方面，因此不再适用于新型异构平台，加速效果不明显。如何开展异构平台上高效的图计算系统研究，已成为目前急需解决的重大现实问题。

在异构计算环境下，不同加速设备的并行性表达与程度存在着显著差别。当今图日臻巨量规模和复杂关联的特征使得传统图计算系统的并发能力受到了很大限制，导致底层丰富的片上并行性未能得到充分利用，更是没有充分考虑异构支持（如FPGA等）的技术优势实施有效的协同调度。其次，目前异构并行编程模型对复杂的硬件细节的屏蔽还不够,使得异构平台上多设备间代码的协同运作难以实施，编写和优化异构平台上的图算法仍然具有较高难度。结合图计算的数据处理特征，亟待研究异构图

计算环境下的运行时系统和编程框架以充分发掘并匹配异构计算平台的技术优势。

在异构内存环境下，图计算数据处理的海量性与数据访问的随机性，易于造成异构系统中某一数据通道承载过多的通信量，成为系统的瓶颈。各类新型存储（如NVM和HBM）相较DRAM在结构上存在着较大差异，亦使得传统访存策略效果不明显。如何管理与协调新型异构硬件资源，实施高效的数据同步与一致性维护，并结合异构图计算的访存特性，设计匹配的数据预取与访存优化是一个令人深思的问题。

2）以虚拟化为代表的云计算需求对异构内存管理带来的挑战

首先，在虚拟化环境下，目前虚拟机监控器（VMM）对于内存主要提供地址转换功能，不同介质的内存资源会被抽象成一种资源。客户机和VMM之间存在的语义隔阂使得客户机操作系统不能感知底层的异构内存，因而应用层无法通过编程优化来提高访存效率。此外，在考虑成本效益的云计算环境下，如何确定一个虚拟机中DRAM 和NVM分配的比例关系是一个挑战性问题。

其次，数据中心网络的一个趋势是网络功能虚拟化（NFV）。一个核心问题是如何对执行特定网络功能的虚拟机镜像进行裁决，减少操作系统中其它不相关功能组件造成的资源浪费和性能干扰。并且，如何在虚拟机启动时给出一些启发式信息，指导NFV在异构内存中的数据放置策略。

最后，由于单节点服务器在空间尺寸上的扩展性不足，通过跨节点的内存资源虚拟化和共享可以有效提高内存资源的容量。为此，需要研究低延迟的异构内存池化关键技术，在网络拓扑和协议上实现可扩展和低延迟端到端传输支持，对本地内存和远程内存访问的调度优化、数据放置策略及数据一致性保障机制。

3）以深度学习为代表的智能应用给异构平台带来的挑战

深度学习是进行大数据挖掘分析一类重要方法。深度学习之所以能在大数据处理上取得令全世界震惊的骄人成绩，是与这些年来计算机体系结构的发展密不可分的。正是因为以GPU集群为代表的新的一代计算机体系结构的快速发展才使得深度学习模型训练在时间和资源消耗上变得可以接受，并在实践中得以应用。

然而，也必须看到，异构GPU集群这种模式应用到深度学习当中正面临越来越多的挑战：比如GPU内存的限制问题，多GPU之间以及多节点之间的通信瓶颈问题，又比如GPU的能耗问题、多节点间任务调度的复杂性问题等，这些越来越明显地成为提高深度学习系统执行效率的障碍。

当前，诸如内存计算、FPGA等新的体系结构的出现和发展为深度学习系统更快速高效的执行提供了新的契机，当然，其中也存在着大量的需要研究解决的问题。

2 主要研究内容

基于上述分析，本研究组将在异构平台上的图计算系统、面向虚拟化场景的异构内存系统和异构平台上的深度学习系统三个方面，开展体系结构和系统软件的研究工作。

1）在面向图计算的异构平台及系统方面，研究异构图计算环境下的新型运行时系统、异构并行编程框架、异构内存数据管理和典型异构图计算应用示范。

第一，研究异构图计算环境下的运行时系统。主要研究可充分发挥异构并行性优势的图计算任务的并行性发掘与硬件匹配策略，为多计算模式间（如CPU-FPGA）的协同工作提供运行时调度与优化关键技术支持。

第二，研究异构图计算环境下的并行编程框架。为异构平台上图算法的开发与优化提供统一抽象的异构编程模型、编程接口及关键库支撑等，在降低图算法的异构并行编程开发门槛的同时，又利于运行时系统进行层次化调度与优化。

第三，研究异构图计算环境下的数据管理机

制。针对图计算随机访问的特性，研究异构环境下的高速通信协议，充分合理地利用设备间带宽，保证数据的高速传输；同时结合不同硬件的差异性，重构高效的数据同步与一致性保障机制。

第四，研究异构图计算环境下的运行时访存特性及优化。针对设备内与设备间（如CPU-FPGA、CPU-GPU）不同的结构特性探究并预测图计算应用的访存特性及规律，设计面向图计算的异构内存数据的放置、预取与替换策略等。

第五，异构平台上图计算系统的典型应用示范。开发典型的图计算系统原型集成上述关键技术，挑选若干典型的图计算应用场景（如社交网络、机器学习、数据挖掘等），对关键技术与核心模型进行验证。

2）在面向虚拟化场景的异构内存系统方面，开展虚拟化环境下基于效益的异构内存动态分配机制，异构内存环境下面向网络功能虚拟化（NFV）应用等云计算需求的虚拟机定制优化，基于RDMA的异构内存池化机制等研究工作。

第一，虚拟化环境下基于效益的异构内存动态分配机制。混合内存中DRAM和NVM两种内存介质在性能（读写延迟）、容量和价格等方面存在差异，因而在云环境下单位容量的DRAM和NVM可以获取的性能效益是不同的。此外，不同虚拟机应用的性能对于内存容量和读写延迟的敏感性是不同的。譬如，有些应用的性能对于内存容量更加敏感，而有些对读写延迟则更敏感。首先根据离线或者在线的访存监测，统计应用访存的读写比例和内存容量需求信息，通过柯布–道格拉斯效用函数确定应用在分配一定容量的DRAM和NVM时的综合效益，然后再综合异构内存容量和价格等因素，得到一个虚拟机内存的最佳分配策略，使应用在成本受限的场景下性能最优。该内存分配策略的重点在于确定不同VM的DRAM和NVM的需求，使用类似博弈论的方法实现内存的有效利用，同时减少NVM的写次数，提升NVM的写寿命，降低系统的整体能耗。

第二，异构内存环境下面向网络功能虚拟化（NFV）的虚拟机定制优化。基于LibOS的思想，以库的形式提供操作系统中的某个功能服务，例如网络功能，根据所要运行的应用，选取一个可用于构建操作系统的最小库集合，随后，通过这些库和相应配置文件组成unikernels，构建轻量级的虚拟机。此外，还要充分考虑在异构内存环境下虚拟机不同介质的内存分配对于虚拟机性能影响，通过静态分析，指导虚拟机在启动时把频繁访问的代码段和数据段放置到DRAM，其他数据放置在NVM，减小NFV虚拟机内页面的迁移开销。

第三，基于RDMA的异构内存资源的池化机制和跨节点共享技术。考虑到DRAM和NVM内存资源延迟的差异性，在异构内存网络结构下，应用数据可以有4种放置方式：local_dram_malloc，local_nvm_malloc，remote_dram_malloc，remote_nvm_malloc。应用程序的内存分配需要根据对访问延迟的敏感性，对数据进行更加精细的划分，综合考虑内存介质的非易失性、耐久性和能耗特征，网络负载均衡，数据并行性等因素，设计更加灵活的数据放置策略。同时，在RDMA访存机制中，对哈希表的遍历会带来较大的时延，在异构内存共享池中，数据流向复杂，标记众多，key-value数据结构的设计和RDMA友好的hash 表设计，将是一个重要的研究问题。需要设计权衡访问延迟（跳数少）和内存数据可靠性（防止全局目录服务器单点失效）的网络拓扑结构，并且具有一定的可扩展性。

3）在面向深度学习的异构平台及系统方面，研究异构平台上深度学习应用的执行方式和内存资源管理策略等。

第一，针对异构集群平台，研究适用于分布式深度神经网络的分组同步优化方法。考虑一种新的以分组为基础的同步方法，克服以往以单节点为同步单元的BSP（严格同步）和ASP（异步）方法存在的不足。具体需要考虑对集群中的节点按性能分组，并结合同步及异步并行机制及学习率的自适应调整来实现更高效的计算；

第二，针对异构平台深度学习系统内存普遍使用效率不高的问题，研究新的内存重用方法和重用逻辑。打破内存重用中各重用内存因大小的差异带来的限制，进一步提高内存重用的灵活度和效率，从而最终达到提高整个系统内存利用率的目的；

第三，针对深度学习系统对于内存计算这一新型体系结构的迫切需求，研究面向深度学习的、高效的内存计算数据调度和管理方法。当前，绝大部分的深度学习系统均架构在异构GPU集群上，如何将内存计算这种新的体系结构与深度学习高效地结合，已经成为一个潜在的非常有研究价值的问题。具体地，研究内存计算中针对深度学习迭代过程的异构内存数据调度方法、数据持久化机制等，并研究高效的深度学习模型数据和训练数据的管理方法。

3 小结

目前，围绕着上述目标和任务，研究小组已经部署人力开展上述研究。小组已经在异构平台上的图计算系统、面向虚拟化场景的异构内存系统和异构平台上的深度学习系统等方面开展了相关的研究工作。

同时，为了在今年全面展开面向应用的异构计算与异构内存平台相关研究工作，本组已经部署人员在CPU+FPGA异构图算法加速方法、面向异构图计算的运行时系统等方面开展了预研和方案设计的相关工作。

再次，对于其他代表性的大数据应用领域，如基于大数据的医疗健康，本组也已部署部分人员开展前期的研究工作，探索大数据应用对异构平台的真实需求以及异构平台在支持真实大数据应用时面临的挑战。

廖小飞
教授
研究方向：系统软件
Email：xfliao@hust.edu.cn .cn

刘海坤
副教授
研究方向：虚拟化，内存计算
Email：hkliu@hust.edu.cn

蒋文斌
副教授
主要从事大数据、分布式计算、媒体计算等领域的研究。
Email：wenbinjiang@hust.edu.cn

邵志远
博士、副教授
研究方向：现从事计算系统虚拟化、高性能计算机体系结构、集群计算等领域的研究
Email：zyshao@mail.hust.edu.cn

郑　然
副教授
研究方向：高性能计算、媒体计算
Email：zhraner@mail.hust.edu.cn

郑　龙
博士后
研究方向：程序分析
Email：longzh@hust.edu.cn

张　宇
博士后
研究方向：大数据处理
Email：zhang_yu9068@163.com

吕新桥
博士，副教授
研究方向：研究兴趣主要包括移动互联网、多核与虚拟化、移动安全等
Email：xqlv@hust.edu.cn

网络空间安全组研究规划

邹德清、徐 鹏、代炜琦、马晓静、羌卫中

本研究方向在2017年的研究重点是围绕着云计算安全的一些关键问题开展研究，包括：软件漏洞检测与分析、软件定义网络安全、大数据安全与隐私保护，另外还将拓展开展区块链技术的研发及产业化工作等，主要的研究规划和关注的研究内容如下：

（1）软件漏洞检测与分析

随着云计算、社交网络、物联网等新技术、新应用的出现，由软件引发的信息安全事件层出不穷，如“维基泄密”事件、“棱镜门”事件等。这些信息安全事件都存在一个共同特点，即软件自身存在可被利用的漏洞。在程序开发阶段，程序的源代码是可见的。基于源码的漏洞检测，能够利用含有丰富语义信息的源码，与软件开发生命周期很好的集成，便于更早地发现并修补漏洞，因此得到了广泛应用。然而，基于源码的漏洞静态检测方法无法准确判断漏洞的真实存在，具有较高的误报率；而漏洞动态检测方法一般开销大、覆盖率低。因此，通过静态检测确定可疑代码位置，并通过动态方法自动验证可疑代码是否能够被利用，具有十分重要的意义。针对上述问题，拟开展以下研究：1）研究多级漏洞检测机制。分别从可疑漏洞、外部输入可达漏洞、可利用漏洞以及可自动生成攻击程序的漏洞4个级别研究漏洞检测方法；2）研究在保证较低漏报率的前提下，尽可能降低误报率的漏洞静态检测方法。主要包括面向漏洞特性的相似漏洞检测、基于深度学习的漏洞模式自动化提取、代码变化对漏洞静态检测的影响等；3）研究基于静态和动态相结合模糊测试的漏洞检测方法。主要包括动态符号执行引导的模糊测试、自动产生输入并验证漏洞、自动生成攻击程序、针对并发漏洞的模糊测试机制等。

（2）软件定义网络安全

传统网络安全的实现主要依靠网络管理员对网络功能部署以及流量转发路径的精心设计，但已远远无法满足云环境中的动态需求，例如用户的加入与退出、网络拓扑变化与虚拟机迁移带来的服务区域变动等。软件定义网络（Software-Defined Networks）受到广泛关注，因为它提供了集中化、自动化的网络管理视角，尤其契合数据中心、云服务的灵活多变的需求。网络功能虚拟化（Network Function Virtualization）则旨在将高度定制化的硬件网络功能虚拟化并运行在x86通用平台上。二者的结合能够为软件定义网络安全提供平台基础，但现有的研究主要集中在设备管理与提高性能上，如何提供软件定义平台、框架与协议，并保证安全性等还有待研究。为实现软件定义化的网络安全，拟开展如下研究：1）研究软件定义网络功能规则的可行方案，实现不同网络功能的可编程配置。主要包括统一规则格式的设计、通讯协议的扩充与设计等；2）研究SDN与NFV融合框架，满足多用户场景下的网络安全可定制化需求。主要包括框架结构设计与验证、网络hypervisor的扩充与功能细化、实现原型管理平台等；3）研究网络规则

冲突检测与解决，同时实现对网络功能与转发设备的规则纠错，保障不同设备间规则的统一与正确执行。主要包括解决单一设备、跨网络功能与整体网络安全策略的规则冲突与错误的解决等。

（3）大数据安全与隐私保护

随着物联网、云计算、5G网络、大数据的发展与融合，传统单一模式下基于密码学的数据安全技术难以解决跨异构网络、跨域、无信任中心实时支持等条件下认证能力的快速搭建与业务随行、加密能力的批量响应与安全耦合等问题。为了解决这些问题，拟开展如下研究：1）基于FIDO协议的多样认证模式。主要包括：群签名/代理签名/指定验证者签名的FIDO化、无样本的生物信息认证等；2）泛在网的数据安全收集与搜索。主要包括：跨异构网络的批密钥分发、异构加密体制下搜索的批处理方法等；3）加密机制的耦合与解构。主要包括：多类加密机制融合下安全性的补偿、面向云平台开放接口的加密机制解构方法。

网络视频监控主要基于IP网进行远程监控，以及视频传输、存储、管理，实现跨区域的统一监控、统一存储、统一管理、资源共享。但视频监控的安全问题，尤其在云计算这种大规模共享环境下的安全问题，显得尤为突出。针对基于云的视频监控应用所面临的隐私泄漏问题，拟研究支持密文内容检测的全区域保护技术。该技术的相关研究尚处于探索起步阶段，将以在多媒体安全领域的相关研究为基础，基于云视频监控的隐私保护实际需求，兼容H.264 和HEVC标准压缩视频，考虑隐私语义的分层特性，研究视频密文内容检测技术。拟开展以下研究：1）压缩标准兼容的视频密文运动物体形状轨迹识别方法；2）压缩标准兼容的视频密文运动存在性判定方法；3）压缩标准兼容的视频运动信息泄漏粒度可控的密文检测方法；4）监控视频运动信息隐私泄漏程度量化评估模型。

（4）区块链技术

伴随着区块链技术如火如荼的发展，区块链的一些问题也日益暴露出来。主要包括：区块链的监管性问题、区块链的并发度问题、多条区块链之间的交互问题、智能合约的安全可控问题等。为解决以上问题，主要拟从以下几方面开展研究开发工作：1）研究区块链的权限管理，在区块链应用中加入注册中心和权限管理功能，使得用户身份和钱包地址绑定，并将绑定信息写入链中，权限管理可以有效限制用户的交易权限，在每一笔交易发生之前首先对用户权限进行验证，进而防止越权交易的发生；2）研究区块链的并发机制，部署“零确认”智能合约，只需要对交易双方的地址进行检测，而无需确认货币的数量以及真实性，从而避免了等待传统公有链6个区块的确认时间，从而达到秒级确认；3）研究多链交互的扩展机制，搭建一个以智能合约为基础的区块链平台，其他区块链可以通过侧链技术进行接入，实现不同区块链之间的信息流通；4）研究智能合约的可靠、可控机制，制定可定制的标准化智能合约模板，既方便应用开发者使用模板编写合约，又可控制合约的安全性。

目前，围绕着上述研究目标，研究小组已经部署人力开展相关研究，在漏洞检测与网络攻防方面，正在开展基于代码相似性和深度学习的研究工作，网络攻防平台已搭建完成；在软件定义网络安全方面，正在开展融合NFV和SDN的软件定义配置的研究工作；在大数据安全与隐私保护方面，也积累了多篇顶级期刊的

研究成果；在区块链方面，正在开发凤链平台，进行产业转化。展望2017年，一方面拟争取能在网络空间安全领域顶级学术会议和期刊上发表研究成果，扩大研究工作的影响，另一方面小组也会持续重视科研成果转化，解决一些关键领域中的安全问题。

邹德清

教授

研究方向：主要从事系统安全、网络攻防等领域的研究。

Email：deqingzou@hust.edu.cn

徐　鹏

博士，副教授

研究方向：主要从事公钥密码学、基于身份密码学、格密码学、云安全等领域研究。

Email：xupeng@mail.hust.edu.cn

代炜琦

博士，讲师

研究方向：主要从事云计算安全、可信计算、虚拟化安全、可信SDN等领域研究。

Email：wqdai@hust.edu.cn

马晓静

华中科技大学计算机学院讲师

研究方向：多媒体安全

Email：lindahust@mail.hust.edu.

羌卫中

华中科技大学计算机学院副教授

研究方向：系统安全，云安全

Email：wzqiang@hust.edu.cn

微软基于ARM芯片开发了套新操作系统（转载）

老牌芯片生产商英特尔该紧张了。

美国时间3月8日，微软展示其新版服务器操作系统，该操作系统将运行高通公司基于ARM技术的处理器，寻求打破英特尔在数据中心芯片领域长年的统治地位。

在此之前，微软与英特尔一直保持着稳定的合作关系，微软的Windows服务器也一直搭载基于英特尔技术的芯片。

彭博社援引微软云计算部门Azure的副总裁Jason Zander的话说，微软已经与高通（Qualcomm）和凯为（Cavium）两家公司合作，为使用ARM处理器开发了一个新版的Windows操作系统。微软目前还在测试这些芯片的搜索、存储、机器学习和大数据任务处理能力，尚未在任何面向客户的网络中运行这一更节能的处理器，公司也没有透露将来会推广到什么程度。

微软公司在一篇官方博客中写道，ARM版Windows处理器是为微软Azure云计算服务的内部使用而开发的，并计划在一个硅谷计算机硬件开放标准会议上展示这个版本的服务器。微软通过测试发现，ARM服务器芯片能够为搜索和机器学习等一些云任务“创造最大价值”。

据《华尔街日报》报道，在这个开放硬件会议上，高通计划宣布针对使用其ARM架构服务器芯片Centriq 2400的服务器的开源设计。Centriq 2400符合微软的云计算系统标准。

Zander在一次采访中表示：“目前这种芯片还没有投入生产，但这将是符合逻辑的下一步。这是对于微软来讲是一个重要的一步，如果我们不把它看做一个专注的项目，或我们路线图的一部分，我们就不会带它去大会上。”

微软计划在开发新云端服务器设计的时候加入ARM架构的芯片，并将在计算机硬件开放标准会议上宣布新的合作伙伴和设计所需组成部分。由于该芯片的设计是开源的，即它可以被自由使用和个性化修改，因此其他公司也可能使用这种芯片的衍生版本。

微软名叫“奥林巴斯项目”（Project Olympus）的服务器设计和ARM为基础的处理器使用，都反映出了微软公司通过硬件创新来削减支出、增强适应性和竞争力的努力，从而与亚马逊、谷歌这两家同样提供强大云服务的企业竞争。

即使大型的云企业已经开始转投不知名的服务器、存储器和网络设备，英特尔芯片仍然被最广泛使用，控制了服务器芯片超过99%的市场份额。此次微软与ARM合作，或许将对英特尔这一地位受到挑战。

去年7月，日本软银以234亿英镑（约合310亿美元）的价格，现金收购了英国芯片巨头ARM。

相比传统的芯片制造商英特尔，ARM更早投身移动设备的芯片设计，赶上了智能手机和平板迅速增长的浪潮。全球逾95%的智能手机都基于ARM的架构设计，所有的iPhone和iPad都使用ARM芯片。

此前，软银集团CEO孙正义下注ARM，外界多分析这是孙正义看好物联网的未来而做的决定。但目前看来，ARM还有挤占英特尔原有市场份额的打算。

大数据组研究规划

——海量数据组规划

石宣化、陈汉华、赵 峰、袁平鹏、华强胜、谢 夏、丁晓峰、于东晓、肖 江

本研究方向在2017年的研究重点是大数据基础算法、大数据处理支撑技术（如内存计算、流计算、大规模图计算、突发缓冲等）、大数据技术应用（如细粒度动作识别）以及典型大数据行业应用解决方案。主要的研究规划和关注的研究内容如下：

（1）低复杂度并行和分布式算法研究

大数据环境下设计低复杂度算法至关重要，这里低复杂度一般指亚线性（sublinear）复杂度或者对数多项式（polylog）复杂度。目前很多常见问题的顺序算法，譬如对多源最短路径问题APSP （All-Pairs-Shortest-Paths）精确求解需要立方时间复杂度，即使近似求解也需要平方时间复杂度。随着并行和分布式系统的广泛使用，并行和分布式算法对设计低复杂度算法提供了更大舞台。譬如，如果采用分布式算法，我们已有研究工作表明无权无向图上APSP问题可以在亚线性时间精确求解。

项目组近期开展的研究主要基于静态大规模图，接下来项目组拟对动态图中图性质维护设计低复杂度并行与分布式算法。目前，动态图中图性质维护算法只能处理单元素变动（插入或删除一条边或者一个节点）。当有多个元素变动时，需顺序处理单元素。显然，这一动态图性质维护方式未充分挖掘图性质维护过程中元素变动处理的并行性，不能充分利用当前多核处理系统和分布式系统带来的计算能力。我们拟处理多元素变动（多条边和多个节点插入或删除），找到可并行处理的元素变动满足的性质，设计并行和分布式算法处理可并行处理的元素变动，从而大大缩短图性质维护时间。

项目组目前设计的分布式算法主要面向同步通信模式下的无权无向图，并且假定处理机网络稳定可靠。但是现实并行分布式系统中节点可能失效，通信可能为异步，且所处理的图可能为有向有权图，这些都给设计低复杂度分布式图算法带来了很大挑战。接下来的一年项目组会集中精力应对这些实际问题。此外，针对当下分布式计算的热点应用区块链（BlockChain）技术，我们也期望能在区块不断增加情况下提出低复杂度分布式一致性（Consensus）算法。

（2）面向数据密集型应用的内存管理优化与I/O优化

随着数据的持续增长，传统以计算为中心的计算机体系结构以及软件架构越来越不适合数据密集型应用需求，尤其是对于新型的数据密集型迭代性应用，如机器学习、深度神经网络等。内存计算作为一种新型架构越来越被大数据领域接受并且变得流行。但现有大数据处理系统依然存在内存数据管理效率低下的问题，导致最终内存计算应用时效性差、性能波动大等问题。为解决内存计算中内存对象管理问题，拟开展如下研究：1）研究高效内存数据对象管理机制，减少内存数据膨胀问题。主要包括基于编程语言特性的对象优化、基于生命期的对象容器管理等；2）研究内存数据与磁盘I/O数据替换策略，减少内存数据与I/O数据的频繁交互。主要包括内存数据Shuffle优化、内存数据压缩等；3）研究混合内存下的动态数据放置策略，利用NVM消除动态磁盘访问。主要包括内存数据映射、基于NVM特性与对象生命期的内存数据放置等；4）研究针对数据密集型应用的突发缓冲技术。

（3）大数据流处理技术

随着互联网、社交网和物联网技术的蓬勃发展，“大数据”往往以“流数据”的形式呈现出来。例如：股票证券的交易数据流、智慧物流系统中的车辆和商品流、电子商务支付数据流、智能电网负荷的实时监测数据流、移动通信网络中用户的通信电话数据流、社交网络用户行为数据流等。如何对纷繁的数据流进行高效处理，进而准确、及时地发现、分析其中隐含的信息，对发掘大数据中蕴含的巨大价值具有重要意义。分布式流处理（distributed stream processing）技术日益成为一种高时效大数据处理的主要手段，为互联网、云计算、物联网中的上层应用提供具有复杂语义的事件发现、监测、预警等基础服务，具有广阔的应用前景。然而流数据具有规模大、泛在性高、实时性强、实时变化快、关联稀疏性明显等特征，给流数据的高效处理带来空间开销高、计算复杂、实时需求与处理质量难以同时保证等严重挑战。现有的流处理系统，如Storm和Spark Streaming等，只针对简单流数据源的快速处理提供了支持，而面向泛在多源数据，在数据处理的高效性、一致性和质量方面均缺乏有效支撑。为解决现有分布式流处理系统的局限性，拟开展如下研究：1）研究分布式流处理平台的任务调度，并行优化及负载均衡机制；2）研究多源异构流数据的实时高效连接技术；3）研究高速动态流数据中的实时热点识别技术；4）研究分布式流处理系统中数据处理一致性保障和关联容错技术；5）研究分布式流处理系统中操作感知的质量控制技术；6）研究基于异构资源的分布式流处理平台及其部署与调度技术。

（4）大规模图计算与图挖掘

图是表达互联数据的有效形式之一。现实世界中，图比比皆是，如社会网络、文献网络等。图数据在不断的增长，如何高效的对大规模图数据进行存储和处理是亟待解决的挑战之一。本研究开展如下探索：1）数据划分是并行分布处理关联大数据的前提。综合图的几何特性、均衡、局部性、数据冗余等多种因素，研究大规模图的划分方法；2）研究大规模图数据及相关操作的构成，研究大规模图数据处理的编程模型。3）研究适应并行分布式计算环境及新型硬件技术（如海量主存、多级缓存、异构众核、GPU等）的图处理任务调度策略。4）研究大规模图数据划分感知的查询分解方法，适应大规模并行分布处理的大图查询计划生成方法；5）研究大规模低开销图数据并行事务处理技术。

数据挖据是指从大量的数据中通过算法搜索隐藏于其中信息的过程。由于大数据的需求，人们越来越多投入到对图数据挖掘的研究工作中。挖掘大数据中的复杂社团结构有助于在社交网络中更清晰的反映出用户之间的亲密度，在社交敏感性搜索中可以为用户推荐更多相关或相似用户；在上下文感知的搜索中，迅速定位相关性更高的网页。对大数据中的复杂社团结构挖掘问题，我们在2017年将探索如何将社团进行有效抽象为动态图，基于内容变化研究动态图挖掘，讨论动态最优路径查询方法以及如何根据用户的喜好设计最优路径；利用张量的多维关联特点，研究基于高阶奇异值分解的社团结构挖掘算法，并提出优化方法；结合张量链分解的优势，提出融合高阶分解和张量链分解的高效计算方法，缓解维度灾难问题。

（5）大数据搜索

数据和内容是大数据网络的核心，随着计算机的处理能力的日益强大，能获得的数据量越大，能挖掘到的价值也就越多。如何对大数据实施快速准确的访问、获取有价值的信息具有重要的现实意义，也是大数据网络信息检索的核心价值所在。近年来，实时搜索和发现用户所需要的信息，理解数据空间和网络空间之间的交互，分析和挖掘网络大数据背后隐含的关联，对其进行可靠管理、准确分析和全面理解，并提个性化的服务，已成为国内外众多学者高度关注的焦点和热点。有鉴于此，我们拟开展如下研究：1）深

层网络搜索：越来越多的Web数据存储在可访问的在线数据库中，这些Web数据不能被传统搜索引擎（如Google, Baidu, Yahoo等）检索到，只能通过查询实时产生的动态页面返回给用户，是“深层”网络（Deep Web）数据。我们主要研究如何搜索和发现用户所需要的信息，挖掘Deep Web背后隐含的知识，如：查询接口集成、数据采样、数据获取等。2）社交网络搜索：随着计算机技术的迅速发展与互联网的快速传播，在线社交网络取得了飞速发展，众多社交网站如Facebook、Twitter、Sina Weibo、Tencent等迅速崛起。我们主要研究如何对这些在线数据实施快速准确的访问、获取有价值的信息，如，实时搜索、垂直搜索、个性化社会推荐等。3）泛在网络信息搜索：云计算环境、移动网络、物联网信息搜索的查全率和查准率低。我们主要研究移动网络、物联网等泛在网络环境下海量数据并行分布式检索的关键技术与核心算法，如：IoT检索、移动搜索等。

（6）抗隐私分析的大数据查询处理优化

在大型科学计算、航空航天、生物制药和物联网等以数据密集型为主要特征的诸多应用领域，数据规模庞大，更是以TB 级甚至于PB级的速度高速增长，引起前所未有的数据存储、管理与查询的复杂度，大数据安全与隐私保护问题更是首当其冲，特别是大数据在存储、访问、查询等过程中带来的个人隐私问题，更是对大数据管理平台提出了全新的挑战。为了满足各种隐私制约型数据查询与处理的高效性，需要解决传统数据库查询处理方法不能应对面向隐私保护、大数据日益复杂的查询处理需求，涵盖了简单数据查询、数据分析、及较为复杂的针对二维或高维数据查询处理等。为解决以上诸多问题，拟开展以下研究：1）建立面向隐私保护的大数据存储、分布、检索与访问的分布式体系架构，并基于此架构，研究满足隐私保护需求的分布式文件系统；2）设计支持隐私预算分配的并行任务调度方法，以及并行查询的隐私预算分配策略；3）研究抗隐私分析的非结构化数据的组织与索引机制,以及满足包括差分隐私等各种标准的查询处理算法；4）研究支持隐私参数设置的高效并行查询语言及用户编程模型，研究使用SQL 语言进行隐私制约型查询的表达方法。

（7）基于大数据分析的细粒度动作识别

动作识别技术是联系物理世界和信息世界的重要纽带之一，在物联网的智能人机交互、安全监控、医疗看护等领域具有广阔应用前景。随着无线信号处理技术的快速发展和智能无线终端设备的广泛互联，基于海量感知数据资源，利用低成本、部署方便、覆盖范围广的无线网络基础设施实现动作识别技术，已成为全新思路与研究焦点。目前方法针对大规模无线感知数据的特征分析与刻画尚不够深入，多维度信息未能充分利用，特征提取方法单一，难以反映面向细粒度动作（例如：手指“空中写字”）对无线信号的差异化规律，不利于进一步的深度挖掘与知识发现。有鉴于此，我们拟开展以下研究：1）研究复杂室内多径环境下不同动作方式对无线网络物理层信息的影响，展开深入分析归纳，提取刻画动作与无线信号内在关系的关键特征；2）面向高识别率应用的实际需求，通过设计机器学习算法对处理后的特征数据集进行分类，建立科学的动作识别模型，将目标人员当前的动作信息（位置、轨迹、姿态等）与历史动作样本集进行匹对，计算最佳匹配对象从而实现高精度、高鲁棒性的动作识别；3）研究复杂动态环境下运动识别的自适应性问题，包括量化室内环境动态性，构建环境动态性预测模型，在线环境参数学习与定位算法调整等。

（8）典型行业应用

各行各业均产生大规模数据。行业大数据除开具有大数据的共同特征之外，还具有自身行业特性。行业大数据既是检验和完善所发展的大数据相关理论与方法的重要环境，也是新的科学问题的来源。对此，本研究方向将继续

2016年选取的医疗、通讯、航空领域开展如下研究：1）研发对复杂内联关系的医疗健康大数据进行高效的深度挖掘和分析的医疗健康大数据智能分析平台，为个性化医疗、精准医疗提供信息保障。在此基础上，面向社区医疗、个性化医疗等提供智能化云服务。2）针对移动通信领域巨规模用户呼叫数据的高效存储、管理和处理，构建基于局部性评估的巨规模呼叫数据并行分布处理模型、高效的查询处理及挖掘方法。3）研发航空大数据关联分析与价值提取云服务平台，实现客户行为分析、服务质量分析、旅客流量分析、航班延误分析、航线网络评估、增值服务与交叉销售分析等，解决航空服务质量的全面升级迫切需求问题。

2016年，大数据研究小组从海外引进两名副教授，小组研究实力进一步增强。围绕上述研究目标，研究工作正在有序开展。小组已经研发了Mammoth、Deca、TripleBit、 PathGraph、Frog等一批较有影响力的系统，新的系统研发也正在开展，一方面拟争取能在2017年的大数据领域顶级学术会议和期刊上持续发表研究成果，扩大研究工作的影响，另一方面小组也会持续重视科研成果转化，将科研成果应用到重要大数据处理领域的龙头企业和实际的生产性环境中，并为拥有特种数据的企业提供大数据技术支持，解决行业生产中的大数据管理问题。

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

陈汉华
博士，教授
研究方向：分布式计算与系统，移动互联网，社交网络，对等计算与无线传感器网络。

赵　峰
华中科技大学计算机学院教授
研究方向：信息检索、数据挖据与分布式计算等
Email：zhaof@hust.edu.cn

袁平鹏
华中科技大学计算机学院副教授
研究方向：图数据库、图处理系统分布式计算等
Email：ppyuan@hust.edu.cn

华强胜
副教授
研究方向：网络和分布式算法
Email：qshua@mail.hust.edu.cn

谢　夏
博士，副教授
研究方向：海量数据挖掘，并行计算
Email：shelicy@hust.edu.cn

丁晓锋
博士、华中科技大学副教授
研究方向：分布式计算、海量数据管理、不确定性数据查询处理技术和数据隐私保护技术。
Email：dxfeng.hust@gmail.com

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，图算法，无线网络

肖　江
华中科技大学计算机学院副教授
研究方向：移动计算、无线室内定位与智能感知，大数据分析等
Email：jiangxiao@hust.edu.cn

Linux C标准库IO缓冲区——行缓存实现

（罗 伟 http://luoweiroway.com/blog/2015/06/01/line-buffer-implementation/）

1 glibc中的FILE数据结构实现

```
//<stdio.h>
typedef struct _IO_FILE __FILE;

//<libio.h>
struct _IO_FILE {
int _flags;/* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

/* The following pointers correspond to the C++ streambuf protocol. */
/* Note: Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
char* _IO_read_ptr;/* Current read pointer */
char* _IO_read_end;    /* End of get area. */
char* _IO_read_base;   /* Start of putback+get area. */
char* _IO_write_base;  /* Start of put area. */
char* _IO_write_ptr;   /* Current put pointer. */
char* _IO_write_end;   /* End of put area. */
char* _IO_buf_base;  /* Start of reserve area. */
char* _IO_buf_end;   /* End of reserve area. */
/* The following fields are used to support backing up and undo. */
char *_IO_save_base;
char *_IO_backup_base;
char *_IO_save_end; /* Pointer to end of non-current get area. */
struct _IO_marker *_markers;
struct _IO_FILE *_chain;
int _fileno;
#if 0
int _blksize;
#else
int _flags2;
endif
_IO_off_t _old_offset; /* This used to be _offset but it's too small. */

#define __HAVE_COLUMN /* temporary */
/* 1+column number of pbase(); 0 is unknown. */
unsigned short _cur_column;
signed char _vtable_offset;
char _shortbuf[1];

/* char* _save_gptr; char* _save_egptr; */
_IO_lock_t *_lock;
#ifdef _IO_USE_OLD_IO_FILE
};
```

指针	意义
_IO_buf_base	指向"缓冲区"
_IO_buf_end	指向"缓冲区"的末尾
_IO_read_base	指向"读缓冲区"
_IO_read_end	指向"读缓冲区"的末尾
_IO_read_ptr	读缓冲区的读指针
_IO_write_base	指向"写缓冲区"
_IO_write_end	指向"写缓冲区"的末尾
_IO_write_ptr	"写缓冲区"的写指针

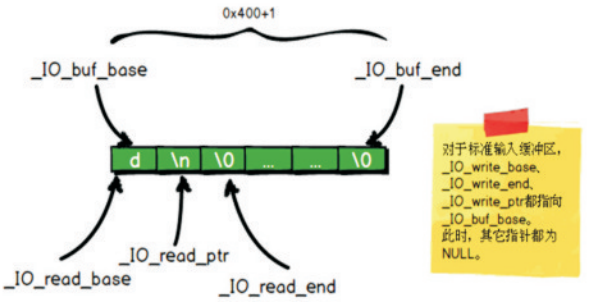
上面的定义看起来给出了3个缓冲区，实际上_IO_read_base、_IO_write_base、_IO_buf_base

都指向了同一个缓冲区。缓冲区在第一次buffered I/O操作时由库函数自动申请空间，最后由相应库函数负责释放

2 一个小现象引起

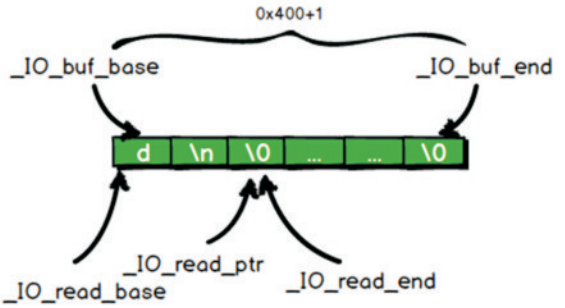
```
int ch1;
int ch2;
ch1 = getchar();
ch2 = getchar();
```

上面这个代码，第一次调用getchar的时候，其会调用读系统调用，以让用户输入，我们输入字符d然后换行表示输入结束，getchar调用成功后，输入缓冲区如下所示：



再次调用getchar时，我们会发现我们没有输入任何东西，getchar就已反回了内容给ch2，这时因为_IO_read_ptr还没有与_IO_read_end重合，表明输入缓冲区中仍有内容可读，而该内容即为第一次输入时的结束换行符，可以发现ch2正好为10（换行符\n的ASCII 码值）。

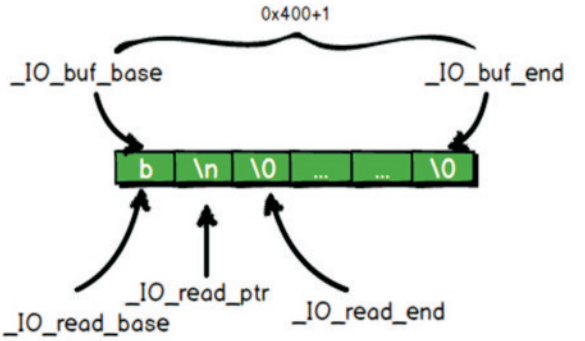
此次调用后，输入缓冲区如下所示：



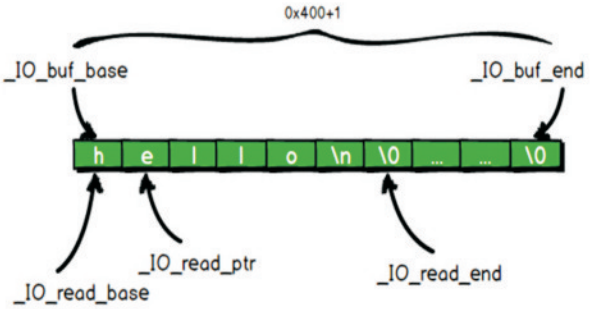
如果我们再加一个getchar，即代码如下：

```
int ch1;
int ch2;
ch1 = getchar();
ch2 = getchar();
ch1 = getchar();
```

则运行到第三次getchar时，会调用读系统调用，新输入的内容会覆盖原内容，假设输入字符b，然后换行结束输入，则输入缓冲区如下所示：



如果输入多个字符，如hello，则他们都会被读到缓冲区，然后之后如果有调用getchar或者其他标准输入函数，则函数会直接从缓冲区中读，不再发起读系统调用。输入缓冲区如下所示：



要想清除输入缓冲区的内容，某些C库(比如MSCRT)实现了fflush(stdin)，而glibc没有实现，调用此函数，stdin缓冲区没有变化。glibc下调用rewind(stdin)也没有效果。

3 行缓冲实现

3-1 stdin输入

假设调用fgets(buf,n,stdin)，意欲读n-1个字符，但是，输入行中的所有数据都会被写入缓冲区（不超过缓冲区大小的前提），包括表示结束输入的换行符(如果输入Ctrl-D表示文件结尾，则不会有换行符)。

如果缓冲区中的内容（即输入的字节数）大于n-1，则_IO_read_ptr向前移动了n位(即指向缓冲区中已被用户读走的字符的下一个)，下次再次调用fgets操作时，就不需要再次调用系统调用read，而是直接从_IO_read_ptr开始拷贝，当_IO_read_ptr等于_IO_read_end时，标准I/O会认为已到达文件末尾或者填满的输入缓冲区已读空，再次读则需要再次调用read。

如果缓冲区中的内容（即输入的字节数）小于n-1，则有多少读多少，读完后_IO_read_ptr指向缓冲区中已被用户读走的字符的下一个。

正常读取完毕后(即_IO_read_ptr与_IO_read_end之间没有内容了)，所有_IO_read_ptr、_IO_read_base、_IO_read_end系列指针全部指向_IO_buf_base。

总结来说，就是：

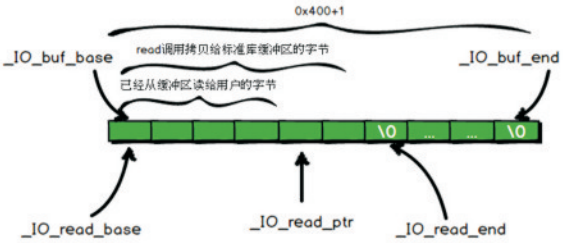
_IO_read_base始终指向缓冲区的开始；

_IO_read_end始终指向已从内核读入缓冲区的字符的下一个；

_IO_read_ptr始终指向缓冲区中已被用户读走的字符的下一个；

(_IO_read_end < _IO_buf_end) && (_IO_read_ptr == _IO_read_end)时则表明已到达文件末尾；

(_IO_read_end = _IO_buf_end) && (_IO_read_ptr == _IO_read_end)时则表明缓冲区已填满且已读完。



3-2 stdout输出

标准库函数先从用户指定的地址出读数据到输出缓冲区中，读多少则_IO_write_ptr往后移多少，满足以下3个条件之一会导致输出缓冲区立即被flush：

- 缓冲区已满;
- 遇到一个换行符;
- 用户主动调用fflush函数;
- 调用标准输入函数时;

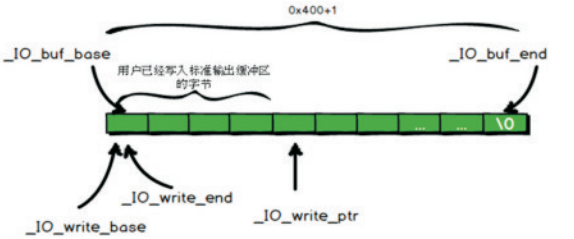
输出缓冲区flush的时候，将_IO_write_base和_IO_write_ptr之间的字符通过系统调用write写入内核，然后令_IO_write_ptr等于_IO_write_base，此时标准IO认为输出缓冲区为空。

行缓冲写的时候：

* _IO_write_base始终指向输出缓冲区的开始；

* _IO_write_end始终指向输出缓冲区的开始；

* _IO_write_ptr始终指向输出缓冲区中已被用户写入的字符的下一个。



4 无缓冲

无缓冲时，标准I/O不对字符进行缓冲存储。典型代表是stderr, glibc中将缓冲区大小(_IO_buf_end-_IO_buf_base)设为1个字节。

对无缓冲的流的每次读写操作都会引起系统调用。

为GlusterFS设计新的xlator (编译及调用过程分析)

（黎 明 https://lidawn.github.io/2016/11/28/glusterfs-xlator/）

GlusterFS 是一个开源的网络分布式文件系统，前一阵子看了一点GlusterFS(Gluster)的代码，修改了部分代码，具体是增加了一个定制的xlator，简单记录一下。

Gluster与xlator

随着计算机技术的发展，不管哪一个领域的数据都呈现出爆炸性增长的趋势，因此产生了大数据处理与存储技术。单机的存储基本不可能满足大量离线数据（文本）的存储需求了，于是在网络分布式文件系统越来越受到重视。开源的分布式文件系统非常多，GlusterFS，Lustre，Ceph，HDFS，FastDFS，关于这些文件系统的分类与区别，可以参考这里，我觉得从块，文件，对象的角度划分比较靠谱。我本是做高性能计算的存储方面研究的，阴差阳错地入了Gluster的坑，具体原因不表了。

Gluster是基于FUSE的用户态文件系统，意味着编译安装Gluster不需要去牵涉内核，关于FUSE，其实Gluster做的比较粗暴，用一个死循环去读取/dev/fuse这个块设备，再丢给客户端或者网络，但是FUSE的原理还是值得去研究的，我也是一知半解。兼容POSIX标准，意味着Linux标准库的read，write等I/O函数不需要经过修改就可以在Gluster上运行。

一个网络分布式文件系统的套路通常是，服务端有多台机子构成一个统一的名字空间，文件以某种分布方式存放在不同的服务器上。而客户端看到的却是一个整体，如一个目录，并且客户端可以被挂载在多个不同的节点，因此可以随时随地访问你的数据。每个文件系统为了实现这一套，都会有各种各样的概念，但本质都是一样的，例如Gluster里面有一些基础概念，其中brick是一个存储节点上的一个输出目录，volume是一系列的brick，代表一个功能子集，translator（xlator）是连接子volume的，xlator本身也是某一个volume的具体实现。

Gluster支持多种数据分布方式：

1. Distributed(默认分布方式)

一个文件分布在一个brick上，不同的文件可能分布在不同的brick上。没有容错。Distributed方式的分配粒度是文件。

2. Replicated

每一个文件都会在每个brick存一个copy，replica数目可以由配置文件指定。Replicated的分配粒度是文件。

3. Distributed Replicated volume

前两者的结合，brick的数量是replia的n倍，假如有N个brick，replica是2，则distribute数目是N/2，相邻的两个brick互为备份。先distribute，再replicate。

4. Striped Volume

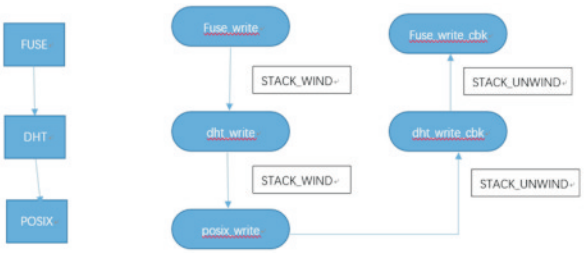
文件被分成固定大小的块，以RR方式分布在不同的服务器上。Striped的分配粒度是文件块。

xlator是Gluster设计的精髓所在，每一个功

能都可以用一个xlator来实现，例如每种分布规则对应一个xlator，另外一些feature可以封装在一个xlator中，如文件加密。并且可以在配置文件中各种xlator混合，嵌套使用，每个xlator编译后会生成一个动态链接库，运行时按需加载。

xlator中的调用(STACK_WIND)与回调(STACK_UNWIND)

Gluster在不同层级的xlator之间的通信有点类似于递归，主要依赖于代码中的两个宏，分别是STACK_WIND和STACK_UNWIND。每一个xlator中的相关函数都有一对，如 write 函数有着对应的 write_cbk 函数，两个函数与两个宏定义配合使用。



我们把xlator的关系简化成三层，FUSE，DHT，POSIX，关系如上图左边所示。假设系统从/dev/fuse中读到了一个write请求，系统将这个请求丢给FUSE xlator，在FUSE xlator中调用fuse_write，通过STACK_WIND将请求传递给他的subvolume，调用subvolume对应的write函数，即dht_write, dht_write同理通过STACK_WIND调用posix_write。在图中posix是最底层的xlator，因此posix_write将不会调用STACK_WIND，而是调用STACK_UNWIND将返回值或者结果返回给父volume，对应着调用父volume中的_cbk函数，即dht_write_cbk，该函数做完相应的处理后继续通过STACK_UNWIND返回到fuse_write_cbk中，这样一个write才算完成。

调用 (STACK_WIND)

接着具体分析一下STACK_WIND是如何工作的。下面是STACK_WIND的宏定义。

```
/* make a call */
#define STACK_WIND(frame, rfn, obj, fn, params ...)                    \
  do {                                                                 \
    call_frame_t *_new = NULL;                                         \
    xlator_t     *old_THIS = NULL;                                     \
                                                                       \
    _new = CALLOC (1, sizeof (call_frame_t));                          \
    ERR_ABORT (_new);                                                  \
    typeof(fn##_cbk) tmp_cbk = rfn;                                    \
    _new->root = frame->root;                                          \
    _new->next = frame->root->frames.next;                             \
    _new->prev = &frame->root->frames;                                 \
    if (frame->root->frames.next)                                      \
    frame->root->frames.next->prev = _new;                             \
    frame->root->frames.next = _new;                                   \
    _new->this = obj;                                                  \
    _new->ret = (ret_fn_t) tmp_cbk;                                    \
    _new->parent = frame;                                              \
    _new->cookie = _new;                                               \
    LOCK_INIT (&_new->lock);                                           \
    frame->ref_count++;                                                \
                                                                       \
    old_THIS = THIS;                                                   \
    THIS = obj;                                                        \
    fn (_new, obj, params);                                            \
    THIS = old_THIS;                                                   \
} while (0)
```

在dht xlator中，dht_write 函数里这样调用STACK_WIND:

```
STACK_WIND (frame, dht_writev_cbk,
            subvol, subvol->fops->writev,
            fd, vector, count, off, iobref);
```

把参数代入到宏定义中，可以按照下面的代码理解：

```
new->parent = frame;
new->this = subvolume;
typeof(fn##_cbk) tmp_cbk = dht_writev_cbk;
new->ret = (ret_fn_t) tmp_cbk;
fn = subvol->fops->writev;
params = {fd, vector, count, off, iobref};
obj = subvol;
//用subvolume->fop->writev
//参数为new，new为新的frame，new的父节点设置为frame，
//obj 就是subvolume
fn (_new, obj, params);
```

可以看到STACK_WIND主要做了三件微小的事，1 传递调用之间的上下文，代码中是frame 这个数据结构，2 记录当前函数的回调函数，一般是对应的cbk函数，也有特例，像dht xlator中逻辑比较复杂的lookup操作（关于Gluster的核心dht xlator的调用分析可以参考这里），3 调用子subvolume的对应函数，将操作向下传递。

因此按照我们简化的xlator关系，即dht的subvolume是posix，那上面的STACK_WIND就调用了posix_writev：

```
posix_writev (call_frame_t *frame, xlator_t *this,
    fd_t *fd,struct iovec *vector, int32_t count,
    off_t offset,struct iobref *iobref))
```

回调 (STACK_UNWIND)

接下来看一下STACK_UNWIND的工作原理。Gluster中有两种回调的宏，一个是STACK_UNWIND，另一个是STACK_ UNWIND _STRICT，两者的差别只是第一个参数，原理是一样的。通常xlator源码里面用的是STACK_UNWIND_STRICT，原因在宏定义的注释里写了，STACK_UNWIND_STRICT是类型安全的。下面是STACK_UNWIND_STRICT的宏定义。

```
/* return from function in type-safe way */
#define  STACK_UNWIND_STRICT(op, frame, params ...)                \
    do {                                                           \
        fop_##op##_cbk_t     fn = NULL;                            \
        call_frame_t *_parent = NULL;                              \
        xlator_t     *old_THIS = NULL;                             \
                                                                   \
        fn = (fop_##op##_cbk_t )frame->ret;                        \
        _parent = frame->parent;                                   \
        _parent->ref_count--;                                      \
        old_THIS = THIS;                                           \
       THIS = _parent->this;                                       \
       frame->complete = _gf_true;                                 \
       fn (_parent, frame->cookie, _parent->this,
           params);                                                \
      THIS = old_THIS;                                             \
    } while (0)
```

前面提到如果你的xlator是最底层的（如客户端的client xlator，服务端的posix xlator），那么这个xlator里不应该存在 xxx_cbk 函数，而是在操作返回之前调用STACK_UNWIND或者STACK_UNWIND_STRICK。posix xlator里面用的是STACK_UNWIND_STRICT 向父volume返回。 STACK_UNWIND_STRICT的第一句是将第一个参数连接成cbk函数，以下是posix的调用：

```
STACK_UNWIND_STRICT (writev, frame, op_ret, op_errno, &preop, &postop);
```

把参数代入到宏定义中，可以按照下面的代码理解：

```
//这里的frame就是STACK_WIND里的obj
//因此 fn = dht_writev_cbk
```

```
fn = (fop_writev_cbk_t)frame->ret;
//parent 就是dht的frame
_parent = frame->parent;
//因此这里调用的是
//dht_writev_cbk(dht_frame,posix_frame,dht,
params);
//实际第二个cokkie参数在dht_write_cbk会
被忽略
fn (_parent, frame->cookie, _parent->this, params);
```

可以看到替换后，首先将上下文换成正确的上下文，即父volume的frame，然后代码的最后一句实际是调用了父volume的 cbk 方法，即dht_write_cbk，在dht_write_cbk里会继续调用STACK_UNWIND_STRICT，这样就会将结果返回到根xlator。

以上就是关于增加xlator所能想起来的基础知识，当然写一个xlator是非常困难的，需要理解和研究的东西的太多。有一个建议就是涉及逻辑的操作最好在cbk函数里面做处理，嵌套调用写在 STACK_UNWIND 前面。另外Gluster在高性能领域用的比较少，所有的文件系统都是有利有弊的，例如Gluster的一大特色就是没有集中的元数据服务器，文件的定位是根据文件名计算hash值来做的，因此Gluster没有分布式文件系统中常见的元数据瓶颈问题，但是在没有指定文件名时候做查询（ls），Gluster性能就不会很好，因为要全盘扫描，关于Gluster性能的探讨，推荐这个系列的博客。

参考

https://www.gluster.org/

http://lustre.org/

http://ceph.com/

https://hadoop.apache.org/docs/r1.0.4/cn/hdfs_design.html

https://github.com/happyfish100/fastdfs

时光会证明你的努力

在学校，可能大部分是凭着一纸的分数来定义一个人，进入社会，我们需要比拼的就更多一些，它给了人们一个展现智商加情商的舞台，是一场不分年龄和性别的较量。学校是一个让我们四肢放松的简单而纯粹的地方，而社会给了我们更多施展拳脚和头脑的赛场，让我们开始学以致用，看清自己有几斤几两的能力，而不是像应付一场考试那样简单。

一想到终将要到来的毕业，对未来既恐慌又充满期待，但不得不承认的是，期待还是大于恐慌的。我们总是会对未来有很多担忧，虽然会有波折，不过经历之后再回过头来看，也不过如此。大多数时候的日子都是很平凡的，就像有人说的，我们的生活总是跌跌撞撞但又四平八稳。这占据我们生命大多数时候的平凡时光，是我们好好沉淀磨练的最佳时机。一个人吃饭，一个人泡图书馆，一个人骑行穿梭于城市，一个人背包去旅行，一个人做着很多梦想清单上的事情，一 步一步靠近梦想。自己要做的就是努力，剩下的都交给时光。

只要足够坚持，足够努力，时光会由日积的量变月累到质变，给出一个自然而然的答复。刘同曾在《你的孤独，虽败犹荣》里提及到“你的脸上云淡风轻，谁也不知道你的牙咬得有多紧。你走路带着风，谁也不知道你膝盖上仍有曾摔伤的淤青。你笑得没心没肺，没人知道你哭起来只能无声落泪。你必须非常努力，才能看起来毫不费力。”有些过程是必须经历的，就好像长大。别人总是看着你变高变壮，却没有人能记录下你脉搏的跳动，也没有人知晓你血液的热度。只有你自己能感受那掷地有声的心跳，也只有你自己能体味你身体的冷暖。而你必须要很用力地活着，别人才能意识到你的存在。

而我们那么努力，也不过是为了更从容不迫地面对将来。只要用心做好该做的事，那么在放松的时候，才能更心安理得；在竭尽全力过后，才敢坦然的向时光索要答案。

每一个自己讨厌的现在，都有一个不努力的曾经。人的一生，犹如一场爬山涉水的旅行，其中酸甜苦辣五味尽全，既然已经决定好上路，就不要因为太多细枝末节而犹豫，锲而不舍的为梦想努力，只求不负光阴。

（黄钰瑶

http://user.qzone.qq.com/2634303336/blog/1487684495?ptlang=2052）

栈溢出利用之Return to dl-resolve payload构造原理

（董泽照 http://blog.csdn.net/guiguzi5512407/article/details/53012878 ）

Return to dl-resolve是一种新的rop攻击方式，出自USENIX Security 2015上的一篇论文How the ELF Ruined Christmas，下面介绍一下Return to dl-resolve payload的构造原理。

1 dl-resolve解析原理

Return to dl-resolve主要是利用Linux glibc的延迟绑定技术(Lazy binding)。Linux下glibc库函数在第一次被调用的时候，才会去寻找函数的真正地址然后进行绑定。在这一过程中，主要由过程链接表(PLT)提供跳转到解析函数的胶水代码，然后将函数的真正地址回填到函数的全局偏移表中，再将控制权交给需要解析的函数。

首先，了解一下在动态链接过程中需要的辅助信息：重定位表(.rel.plt和.rel.dyn)、全局偏移表(.got和.got.plt)、动态链接符号表(.dyn.sym)、动态链接字符串表(.dyn.str)。

1.1 重定位表

重定位表中.rel.plt用于函数重定位，.rel.dyn用于变量重定位。函数重定位表的主要作用是提供函数在动态链接符号表以及全局偏移表中的位置。具体的，函数重定位表表项为地址解析函数提供一个参数(r_info的前24位)，定位需要解析的函数；为重定位入口提供一个偏移地址，定位函数地址的保存位置(r_offset)，即函数的全局偏移表值(.got.plt)。重定位表的定义如下：

```
typedef struct
{
  Elf32_Addr    r_offset;       /* Address */
  Elf32_Word    r_info;         /* Relocation type and symbol index */
} Elf32_Rel;
typedef struct
{
  Elf64_Addr    r_offset;       /* Address */
  Elf64_Xword   r_info;         /* Relocation type and symbol index */
  Elf64_Sxword  r_addend;       /* Addend */
} Elf64_Rela;
```

1.2 全局偏移表

全局偏移表中.got保存全局变量偏移，.got.plt保存全局函数偏移。全局函数偏移表主要保存了函数在内存中的实际地址，刚开始全局函数偏移表中存放的是：过程链接表PLT中该函数胶水代码中的第二条指令地址。在函数解析之后，才存放了函数的真正地址。以XMAN中32位ELF level4中的read@plt为例：

```
~/workspace/pwn/jarvisoj/xman$ objdump -d -j.plt level4
level4:     file format elf32-i386
Disassembly of section .plt:
08048300 <read@plt-0x10>:
8048300:   ff 35 04 a0 04 08       pushl  0x804a004
 8048306:   ff 25 08 a0 04 08       jmp    *0x804a008
 804830c:   00 00                   add    %al,(%eax)
   ...
 08048310 <read@plt>:
 8048310:   ff 25 0c a0 04 08       jmp    *0x804a00c
 8048316:   68 00 00 00 00          push   $0x0
 804831b:   e9 e0 ff ff ff          jmp    8048300 <_init+0x30>
   ...
~/workspace/pwn/jarvisoj/xman$ objdump -R level4
level4:     file format elf32-i386
DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
08049ffc R_386_GLOB_DAT    __gmon_start__
0804a00c R_386_JUMP_SLOT   read
   ...
~/workspace/pwn/jarvisoj/xman$ gdb level4
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.2) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
(gdb) x/wx 0x804a00c
0x804a00c <read@got.plt>:   0x08048316
(gdb) x/wi 0x804a00c
0x804a00c <read@got.plt>:   push   %ss
```

调用函数read时，先调用read@plt(本例是0x8048310)。在read@plt中先跳转到.got.plt，如上图所示，第一次调用时.got.plt值为read@plt的第二条指令地址。然后reloc_arg参数进栈，跳转到过程链接表开始（PLT[0]），执行胶水代码。

read@plt - 0x10(本例是0x8048300)是整个过程链接表的开始（PLT[0]），该处的胶水代码是进入_dl_runtime_resolve(link_map, reloc_arg)的入口。pushl 0x804a004，将_dl_runtime_resolve函数的第一个参数link_map入栈，该link_map函数地址位于全局函数偏移表(.got.plt) + 4的地方（64位应该是+8）。第二个参数reloc_arg前面已经通过read@plt中的第二条指令（本例是0x8048316）入栈。

在32位应用中reloc_arg为重定位项在重定位表的偏移值，而64位应用中reloc_arg为重定位项在重定位表中的索引，构造payload的时候需要注意一下。下面的宏定义说明了原因。

```
#ifndef reloc_offset
  #define reloc_offset reloc_arg
  #define reloc_index  reloc_arg / sizeof (PLTREL)
#endif
```

1.3 动态链接符号表和字符串表

动态链接符号表是一个结构体数组，保存了每个函数解析时需要的信息，比如函数名在动态链接字符串表中的偏移等等。动态链接字符串表中保存了函数的名称，并且以\x00作为开始和结尾。动态链接符号表的定义如下：

```
/* Symbol table entry.  */
typedef struct
{
  Elf32_Word    st_name;        /* Symbol name (string tbl index) */
  Elf32_Addr    st_value;       /* Symbol value */
  Elf32_Word    st_size;        /* Symbol size */
  unsigned char st_info;        /* Symbol type and binding */
  unsigned char st_other;       /* Symbol visibility */
  Elf32_Section st_shndx;       /* Section index */
} Elf32_Sym;

typedef struct
{
  Elf64_Word    st_name;        /* Symbol name (string tbl index) */
  unsigned char st_info;        /* Symbol type and binding */
  unsigned char st_other;       /* Symbol visibility */
  Elf64_Section st_shndx;       /* Section index */
  Elf64_Addr    st_value;       /* Symbol value */
  Elf64_Xword   st_size;        /* Symbol size */
} Elf64_Sym;

#define ELF32_R_SYM(val)        ((val) >> 8)
#define ELF32_R_TYPE(val)       ((val) & 0xff)
#define ELF32_R_INFO(sym, type) (((sym) << 8) + ((type) & 0xff))

#define ELF64_R_SYM(i)          ((i) >> 32)
#define ELF64_R_TYPE(i)         ((i) & 0xffffffff)
#define ELF64_R_INFO(sym,type)  ((((Elf64_Xword) (sym)) << 32) + (type))
```

注意：32位动态链接符号表和64位动态链接符号表中表项的顺序。

1.4 延迟绑定的过程

正如前面提到的，从PLT[0]进入_dl_runtime_resolve函数。

32位应用中：该函数位于glibc-2.24/sysdeps/i386/dl-trampoline.S中，用汇编语言写的，先保存寄存器值，然后通过栈传递参数，调用_dl_fixup函数。

```
glibc-2.24/sysdeps/i386/dl-trampoline.S _dl_runtime_resolve(link_map, reloc_arg)函数:

        .text
        .globl _dl_runtime_resolve
        .type _dl_runtime_resolve, @function
        cfi_startproc
        .align 16
_dl_runtime_resolve:
        cfi_adjust_cfa_offset (8)
        pushl %eax              # Preserve registers otherwise clobbered.
        cfi_adjust_cfa_offset (4)
        pushl %ecx
        cfi_adjust_cfa_offset (4)
        pushl %edx
        cfi_adjust_cfa_offset (4)
        movl 16(%esp), %edx     # Copy args pushed by PLT in register.  Note
        movl 12(%esp), %eax     # that `fixup' takes its parameters in regs.
        call _dl_fixup          # Call resolver.
        popl %edx               # Get register content back.
        cfi_adjust_cfa_offset (-4)
        movl (%esp), %ecx
        movl %eax, (%esp)       # Store the function address.
        movl 4(%esp), %eax
        ret $12                 # Jump to function address.
        cfi_endproc
        .size _dl_runtime_resolve, .-_dl_runtime_resolve
```

```
(1) 分别获取动态链接符号表和动态链接字符串表的基址
const ElfW(Sym) *const symtab = (const void *) D_PTR (l, l_info[DT_SYMTAB]);
const char *strtab = (const void *) D_PTR (l, l_info[DT_STRTAB]);

(2) 通过参数reloc_arg计算重定位入口，这里的DT_JMPREL即.rel.plt, reloc_offset即reloc_arg
const PLTREL *const reloc = (const void *) (D_PTR (l, l_info[DT_JMPREL]) + reloc_offset);

(3) 根据函数重定位表中的动态链接符号表索引，即r_info字段，获取函数在动态链接符号表中对应的条目。
const ElfW(Sym) *sym = &symtab[ELFW(R_SYM) (reloc->r_info)];

(4) 根据strtab+sym->st_name在字符串表中找到函数名，然后进行符号查找获取libc基地址result
result = _dl_lookup_symbol_x (strtab + sym->st_name, l, &sym, l->l_scope, version, ELF_RTYPE_CLASS_PLT, flags, NULL);

(5) 将要解析的函数的偏移地址加上libc基址，就可以获取函数的实际地址
value = DL_FIXUP_MAKE_VALUE (result, sym ? (LOOKUP_VALUE_ADDRESS (result) + sym->st_value) : 0);

(6) 将已经解析完的函数地址写入相应的GOT表中
if (sym != NULL && __builtin_expect (ELFW(ST_TYPE) (sym->st_info) == STT_GNU_IFUNC, 0))
  value = elf_ifunc_invoke (DL_FIXUP_VALUE_ADDR (value));
```

函数_dl_fixup位于glibc-2.24/elf/dl-runtime.c中。由于源码太长，现只介绍一下其中主要步骤：

64位应用中：_dl_runtime_resolve函数和_dl_fixup函数在有些方面与32位不同，后面payload构造时介绍。

2 payload构造思路

前面已经介绍过dl-resolve如何进行函数地址的解析，下面介绍一下payload构造思路。

主要步骤：通过函数的function@plt，将reloc_arg参数压栈；再跳转到过程链接表的开始（PLT[0]），将link_map地址压栈。然后调用_dl_runtime_resolve(link_map, reloc_arg)解析函数的地址，并写回到函数的全局偏移表（.got.plt）中，最后返回到需要解析的函数中。

payload构造：如果伪造reloc_arg，使该函数的重定位表表项位于可控制的位置；再伪造重定位表项（即r_offset和r_info），可使该函数的动态链接符号表表项也位于可控制的位置；然后，伪造动态链接符号表表项（即st_name、st_value、st_size、st_info、st_other、st_shndx），主要是st_name的值，使该函数动态链接字符串表表项位于可控制的位置；最后，伪造动态链接字符串表表项为我们想要解析的函数名，就可以了。

3 32位应用的payload构造

下面以XMAN level4中的32位ELF为例，介绍32位应用payload的构造过程。该ELF有个明显的栈溢出漏洞，首先，通过栈溢出向bss段写入我们精心构造的payload2，一般选择bss段偏移为0x400的位置(.bss+0x400)写入。该payload包含解析函数的地址入口、返回值、需要解析函数的参数、伪造的重定位表项、动态链接符号表表项、动态链接字符串表表项、以及函数的参数值。然后，通过平衡堆栈，将程序的执行交给解析函数。

32位应用 payload的主要构成如下：

payload1：利用漏洞进行读操作，将数据写入bss区。

'A' * offset	read_plt	p_p_p_ret	0	base_stage	100	p_ebp_ret	base_stage	leave_ret
溢出填充	返回地址覆盖为read的plt地址	返回地址为gadget平衡堆栈	arg1	arg2,写地址.bss+0x400	arg3	以写入地址恢复ebp，构造假的栈帧，返回		

payload2：构造假的重定位表项、动态链接符号表项、动态链接字符串表项

	"BBBB"	栈顶地址
	PLT[0]	PLT表基址，即调用_dl_runtime_resolve函数的入口
	reloc_offset	重定位偏移，即相对于重定位表(.rel.plt)的偏移
	ret	返回地址，一般随便填充。
	arg1	函数参数1，arg1
	arg2	函数参数2，arg2
	arg3	函数参数3，arg3 //不足3个参数，随便填充其他值
80Bytes	r_offset	假的重定位表项：r_offset ,即函数的got
	r_info	假的重定位表项：r_info
	align	由于动态链接符号表字节对齐，因此需要对齐字节
	st_name	假的动态链接符号表表项：st_name
	st_value	假的动态链接符号表表项：st_value
	st_size	假的动态链接符号表表项：st_size
	st_info....	假的动态链接符号表表项：st_info、st_other、st_shndx
	"system"	假的动态链接字符串表表相：字符串如："system"、"write"等等
	'AAAA....'	填充字符：'A'，使长度达到80字节
	"/bin/sh"	写入的参数字符串："/bin/sh"
20Bytes	'AAAA....'	填充字符：'A'

4 64位应用的payload构造

前面提到过64位应用和32位应用在解析函数地址的时候有一些区别：

1）前面提到过_dl_runtime_resolve函数的参数reloc_arg在32位应用中是偏移值，在64位应用中是索引值；

2）在_dl_fixup中，在伪造sym之后，会造成解析过程中VERSYM取值超出范围，造成segment fault，需要将link_map + 0x1c8地址上的值清0；

3）动态链接符号表表项中的成员顺序有变化（即ELF64_Sym结构体成员顺序有变化）。

在构造payload的时候需要注意以上3点不同，并且64位应用函数通过寄存器传参。在构造payload的时候长度不要太长。因为构造的payload过长，传输的过程会截断，造成无法利用成功。

64位应用 payload的主要构成如下：

payload1: 泄漏link_map地址

'A' * offset	com_gadget	addr_vulfun
溢出填充	x86_64 通用的gadget	漏洞函数的地址

payload2：覆盖link_map 地址 + 0x1c8 的地方为0，并读入数据

'A' * offset	com_gadget	com_gadget	p_rbp_ret	base_stage	leave_ret
溢出填充	将link_map地址偏移0x1c8的地方置0	读入数据写入bss区偏移0x400位置	以写入地址恢复rbp，构造假的栈帧，返回		

payload3：构造假的表项，注意函数通过寄存器传参

	"BBBBBBBB"	栈顶地址
	pop_rdi_ret	函数的参数入rdi寄存器
	addr_shell	"/bin/sh" 字符串的地址
	PLT[0]	PLT表基址，即调用_dl_runtime_resolve函数的入口
	reloc_index	重定位偏移，即相对于重定位表(.rel.plt)的索引
	reloc_align	reloc_arg值为重定位表的索引，需要对齐
180Bytes	r_offset	假的重定位表项：r_offset ,即函数的got
	r_info	假的重定位表项：r_info
	sym_align	由于动态链接符号表字节对齐，因此需要对齐字节
	st_name,st_info...	假的动态链接符号表表项：st_name、st_info、st_other、st_shnd
	st_size	假的动态链接符号表表项：st_size(64bits)
	st_value	假的动态链接符号表表项：st_value(64bits)
	"system"	假的动态链接字符串表表相：字符串如："system"、"write"等等
	'AAAA....'	填充字符：'A'，使长度达到180字节
	"/bin/sh"	写入的参数字符串："/bin/sh"
200Bytes	'AAAA....'	填充字符：'A'

Actor模型漫谈

（史瑜良 http://www.zenlife.tk/actor.md）

这篇文章，从go语言的channel，讲到我自己写的线程库task，从分发/订阅模式，讲到actor模型。算是一个漫谈吧，都是围绕actor模型相关的东西，目前我的一些模糊的理解。

go语言中的actor

go语言中，通过channel + goroutine配合使用，就能达到actor模型效果了Actor行为:

```
var c = make(chan bool)
func Actor() {
  <-c
  //begin to do the thing
}
func Main() {
  go Actor() //观察者在后台开始准备
  c <- true //通知观察者
}
```

一个goroutine就是一个actor，通信是通过语言提供的管道完成的。go语言的goroutine是非常轻量级的，又可以充分发挥多核的优势。actor模式的核心就在这里，无锁+充分利用多核，actor之间通过消息通信共同工作。

actor模型分类

Actor模型的任务调度方式分为“基于线程（thread-based）的调度”以及“基于事件（event-based）的调度”两种。

基于线程的

基于线程的调度为每个Actor分配一个线程，在接受一个消息（如在Scala Actor中使用receive）时，如果当前Actor的“邮箱（mail box）”为空，则会阻塞当前线程直到获得消息为止。

基于线程的调度实现起来较为简单，例如在.NET中可以通过Monitor.Wait/Pulse来轻松实现这样的生产/消费逻辑。

不过基于线程的调度缺点也是非常明显的，由于线程数量受到操作系统的限制，把线程和Actor捆绑起来势必影响到系统中可以同时的Actor数量。

而线程数量一多也会影响到系统资源占用以及调度，而在某些情况下大部分的Actor会处于空闲状态，而大量阻塞线程既是系统的负担，也是资源的浪费。

因此基于线程的调度是一个拥有重大缺陷的实现，现有的Actor Model大都不会采取这种方式。

基于事件的

于是另一种Actor模型的任务调度方式便是基于事件的调度。“事件”在这里可以简单理解为“消息到达”事件，而此时才会为Actor的任务分配线程并执行。

很容易理解，我们现在便可以使用少量的线程来执行大量Actor产生的任务，既保证了运算资源的充分占用，也不会让系统在同时进行的太多任务中“疲惫不堪”，这样系统便可以得到很好的伸缩性。

在Scala Actor中也可以选择使用“react”而不是“recive”方法来使用基于事件的方式来执行任务。

线程级actor模型的实现

go语言中的goroutine和channel自是不用说。goroutine是非常轻量，而且调度器会使用多核。这个语言的设计就是打算这么干的。

lua语言中的coroutine协程，在某种程度上也可以完成actor模型的工作，就是不太好用。是手动的yield和resume的，没有channel而是通过yield和resume的参数来传递消息。

我自己写过一个[[https://github.com/tiancaiamao/task][线程库]]，类似coroutine做的事情，有点像lua和go中的coroutine的一个结合。像lua一样单线程又像go一样支持channel。可以像线程级actor模型这么用。

当然，只是玩具代码，通过swapcontext保存上下文实现的，轻量还算是轻量。这么用还是有问题，主要二个：

· 不像go语言，我写的这个东东不能线程的栈自动扩张

· 用户级的线程库，完全没有用到多核

第二点太要命了，actor模型的出发点就是充分利用多核优势的。当然出发点不一样，我本来也不是写个actor框架的。

要是真的用C语言搞一个这种东西，搞到最后就是go语言的一个拙劣的模仿品。消息队列和一些数据结构倒是蛮有用。后面再考虑自己写一个actor的框架玩玩吧...等有时间了。

RPC与分发/订阅

本文是actor模型漫谈，为什么会谈到RPC和分发/订阅呢？

还是因为中间有相似的东西。先看RPC，想一想RPC的流程，client做一个RPC调用，其实是向server发一个消息，让server做某事，server做完后回client一条消息。

整个过程就像client在调用某个函数一样，只管给个输入，调用一个函数，返回后得到一个结果。它不用在乎过程如何实现的，即这个函数发生在本地还是远程。

换一个角度看，其实RPC就是一种actor模型。client和server都是一个actor，前者通知后者去做某事，不去管后者是如何做的，两个actor之间通过消息通信进行协作。

当然真正的RPC实现中会复杂的多。首先涉及到序列化和反序列化的工作，专业术语叫marshall和unmarshall。

这个解决的是调用参数的问题。比如参数传一个结构体，或是传一个int又或是string，要序列化为一条消息才能通过网络上传输到server那边去。marshall就做这个事。

完了server收到消息，又需要unmarshall出client发过来的参数，再调用自己对应的函数。

server如何知道要调用哪个函数？client又如何知道哪个server提供了哪些服务函数呢？这里就要讲到分发/订阅了。

首先要有一个中心server，它做两件事情：

1. 接受注册事件。某个server提供什么函数，就向中心server注册一下。这就是发布过程。

2. 响应查询事件。某个client可以向中心server查到谁谁谁，提供了什么样的RPC调用。

当然中间有点不同的地方。分发/订阅是

client消息直接发到中心server，由中心server再发到其它server去执行，其它server充当的是worker的角色。

而RPC的中心server只是提供一个查询，返回结果中包含了比如server的端口等一些信息，后面是由client去发消息的。

actor模型

前面废话了这么久，最后，想一想怎么设计一个actor模型的编程框架。

其实云风的[[https://github.com/cloudwu/skynet][skynet]]从某种意义上看，就是一个这种框架。把actor如何通过消息通信弄出来由框架实现的，它上面每个skynet_context就是一个actor，完成某种特定的服务。

横看成岭侧成峰，你可以说它是一个消息通信的框架，每个服务完成自己的任务，它就是actor，等待消息来然后做对应的处理。至于消息怎么来它不管，由框架做了。

框架做的事情也很专一：分发消息，调用对应actor的回调函数。有个线程池，actor不等于线程。这是一个基于事件的actor模型。

不管是基于线程，或者是基于事件，总结规律，看它们的共同特点：

1. 都有一个回调函数
2. 都有消息队列

线程自然是有回调函数，这个就是线程启动后执行的函数。至于基于事件的，它的回调函数就很像RPC中server注册回调函数那样了。回调函数只是一个入口，消息会有类型，然后再解包消息，像protol buffer什么的。

消息队列是都有的。明显，因为actor处理消息的速度可能达不到消息来的速度，这样就必须搞个消息队列排队，并阻塞发送者。

再看不同点。最核心的就是“消息队列”是隐式的还是显式的。这个决定了和使用上的不同。

显式比如通道这东西。actor之间不是直接传消息而是一个传给actor，而是一个写通道，另一个读通道，这个通道就是消息队列。

隐式的就没有把消息队列暴露出来，而是actor自己私有的。比如skynet中是由框架调度发给具体的actor的消息队列。

这个不同之处有多大影响呢？主要影响了编程的模式。

像skynet感觉这么搞编程模式就有一点麻烦。自己写模块，向框架注册回调函数，有消息到来的回调函数就会被执行。

言　论

时间是一切财富中最宝贵的财富。

（董泽照 http://2283296285.qzone.qq.com/main）

自律的人生才有自由。

（唐晓兰 http://user.qzone.qq.com/2279777383/main）

如果我们可以想得少一点，如果我们想到什么就去尝试一下，或许人生不会因此而变得格外圆满，但至少能够少一点点遗憾。

（朱亮 http://user.qzone.qq.com/75745014/311）

生活是由一连串的细节组成的。

（郑龙 http://user.qzone.qq.com/306740552/1）

性能优化之-SSE指令心得

（石 翔 http://glatue.com/2015/12/21/%E6%80%A7%E8%83%BD%E4%BC%98%E5%8C%96%E4%B9%8B-sse%E6%8C%87%E4%BB%A4%E5%BF%83%E5%BE%97/）

一、介绍

SSE（Streaming SIMD Extensions）是英特尔公司开为SIMD（单指令流多数据流）的实现开发的一套指令集。使用SSE指令，能够让CPU在运行一条指令同时运算多个人数据。至于AVX（Advanced Vector Extensions）是Intel与AMD在X86指令集的SSE延伸架构。

二、主流的使用-向量化

主流的方法是编写向量化程序。

OpenMp的向量化

要编写向量化程序的方法很多，比如在openmp中，加入ivdep关键字，就能够自动的向量化。虽然很方便，但是并不能使性能达到最优，因为它的优化由编译器来完成，你不知道它具体干了什么。

SSE指令向量化

本人在Intel Xeno Phi上亲测，如果你对SSE指令比较熟练，你用SSE指令设计出来的程序比omp的自动向量化要快很多。

来看看一个简单的程序，摘自《MIC高性能变成》P218:

SSE的函数参考在此：http://jgsj.iteye.com/blog/2050902

更完整的使用方法在此：https://www.ibm.com/developerworks/cn/linux/l-gccsimd/

三、多线程优化

在多线程下，使用SSE指令中的_mm_stream_si32 系列，让程序绕过cache写内存，在某些情况下，能够起到很好的效果。

场景

多线程共享一个全局变量，或是线程分别拥有的变量在内存中靠得很近的时候（比如每个线程分别读写一个数组中与线程对应的元素，由于cpu的cache会缓存内存中被操作变量周围64 bytes的数据，所以这种情况也可视为共享一个全局变量）。除掉全局变量同步的开销，还存在着cache更新的开销。每个线程都对应一个cpu，每个cpu拥有各自的cache。一旦一个线程要修改全局变量，其余的cpu中的cache都需要进行复杂的协议操作来使得数据一致（参看写回法&MESI协议）。这样会存在很大的开销。

解决方法

存在开销的原因是一个cpu写全局的数据只是写到了自己的cache中，而没有写到真正的内存中，更没有更新到其它cpu的cache中，等到其它cpu需要使用全局变量时，复杂的一致性操作就发生了。若能够让全局变量的数据直接将数据写入内存，就能够大大的减少一致性的操作。

这样直接写内存的操作有很多：

1. SSE指令的方法：采用_mm_stream_si32系列的函数直接写入。实测能起到很好的效果。

2. 使用汇编指令：movntdq 进行内存直写。太复杂，要将ASM嵌入C，没测过，目测可行。

3. 使用volatile修饰全局变量：这种方法去除了所有对全局变量的优化，读和写都不经过cache，直接从内存中读写，比较暴力。本人测试没有方法1效果好。

2016年度实验室十大新闻揭晓

吴　未

2017年2月25日晚，在一年一度的实验室新春联欢会上，2016年度实验室十大新闻揭晓，“石宣化教授指导的华中科技大学代表队获全球大学生超级计算比赛ISC SCC16最高计算性能奖。”等十条新闻入选（详见附件）。

依据实验室十大新闻评选办法，对选中条目最多的师生给予了奖励：有13名师生选中7条新闻，经过抽签，李高峰获得一等奖，王明亮、夏妍获得二等奖，董泽照、胡侃、黄卓、王多强、王世振、郑龙、郑然7人获得三等奖。

附件：2016年SCTS&CGCL十大新闻（排名不分先后）

1、石宣化教授指导的华中科技大学代表队获全球大学生超级计算比赛ISC SCC16最高计算性能奖。

2、廖小飞教授当选为青年长江学者。

3、实验室承办的并行与分布式计算的国际会议（ICPADS 2016）在武汉召开。

4、实验室引进于东晓、肖江两位青年教师，张宇、郑龙两名博士后。

5、2016年9月20日，2016年国家网络安全宣传周“智慧城市建设及安全保障”分论坛在湖北武汉召开。实验室金海老师做大会主持；实验室邹德清老师、徐鹏老师为程序委员会成员。

6、实验室蒋文斌、刘海坤、袁平鹏、赵峰、邹德清、顾琳、于东晓、代炜琦8位老师获得国家自然科学基金。

7、实验室近200余名师生赴太原参加CNCC2016，这是实验室第一次资助硕士生参加CNCC。

8、周知等6位同学获得国家奖学金，实验室学生还积极参加各类活动，有55人次获得校级以上各种奖励。

9、实验室入选科技部重点领域推进计划创新团队。

10、2016年实验室获得了4项国际专利。

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

金海教授团队论文入选《大数据》高被引论文TOP10（转自华中大新闻网）

2月10日，大数据领域的中文权威期刊《大数据》公布了创刊以来的TOP10高被引论文列表，计算机学院金海教授团队的论文入选，排名第六。论文题为《医疗健康大数据：应用实例与系统分析》，作者是董诚（硕士生）、林立（博士生）、金海、廖小飞。

文章聚焦医疗健康大数据的技术现状，指出，目前医疗健康大数据还处于初期发展阶段，但是它已经展现了改变医疗服务的潜力。医疗健康服务提供商利用大数据分析技术可以从临床数

实验室学术委员会第八次会议召开

吴　未

2017年3月11日，服务计算技术与系统教育部重点实验室学术委员会第八次会议召开。

本次会议由学术委员会主任委员深圳大学陈国良院士主持，学术委员会成员浙江大学鲍虎军教授、北京邮电大学方滨兴院士、上海交通大学过敏意教授、武汉大学何炎祥教授、北京理工大学梅宏院士、中科院信工所孟丹研究员、国防科技大学王怀民教授、华中师范大学杨宗凯教授、清华大学郑纬民教授、西北工业大学周兴社教授等委员参加会议。实验室主任金海教授，副主任吴松教授、廖小飞教授以及实验室近20位老师参加了会议。

金海教授作了实验室2016年度工作汇报，就实验室年度整体状况、主要研究方向与研究成果、队伍建设与人才培养、开放交流与运行管理、发展思路等情况进行了详细介绍。吴松教授作了题为“应用驱动的云平台设计与优化——面向云端融合的容器云平台”的学术报告。

学术委员会委员们对2016年实验室的建设、发展和科研工作中取得的研究成果给予了充分肯定，并对在实验室发展现状的基础上，对实验室定位、战略规划、发展路线等方面提出了中肯、具体、可行性强的指导意见，进一步明确了实验室发展方向。

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

接上页

据、研究数据、个人健康数据、公共健康数据中挖掘潜在的关系为临床决策、公共卫生、个人健康提供帮助。将来，医疗健康大数据将会快速地发展。目前，医疗健康大数据还面临着诸多挑战，隐私问题关系到用户的数据不会被用作恶意用途，数据安全和标准化需要成立专门的机构来管理。然而，随着技术的发展，医疗技术和大数据技术的结合将更好的为人类健康提供服务。

健康大数据一直是学校医工交叉领域的重点研究方向，学校自2014年起，就对该研究方向提供了诸多支持。在学校各个部门和医学单位（附属协和医院、附属同济医院等）的直接支持下，金海教授牵头的团队于2016年获批了湖北省发改委“医疗大数据智能分析与服务平台（1期）”的重大项目支持。目前，计算机学院为了大力推进研究工作，已经为该团队提供了近200平米的科研用房，用于兴建华中科技大学医疗健康大数据中心。

目前，医疗健康大数据的研究和建设工作正在快速推进。中心基础设施建设正在推进中，研究工作已经取得了诸多进展。如，已经积累了一批用于大数据，特别是健康大数据分析的技术平台，包括了内存计算系统、图数据处理系统、云计算平台等；结合图像分析、深度学习等技术，研发了用于尘肺病的自动智能诊断系统，准确率高达90%以上；为关爱空巢老人所研发的智慧医疗看护系统；人体生理特征监测的智慧衣等。

据介绍，《大数据》是中华人民共和国国家新闻出版广电总局审批通过的大数据领域的第一本科技期刊。办刊宗旨是“以开放、创新姿态，推动大数据技术的研究与应用，促进技术交流，推广创新成果，服务大数据社会”。

Finding deep compiler bugs via guided stochastic program mutation

黄 冠 推荐

“Finding deep compiler bugs via guided stochastic program mutation”是国际顶级会议OOPSLA'15收录的论文，该论文提出了一种引导程序突变来发现编译器漏洞的方法。从实验结果上来看该方法能够发现大量的未曾发现的编译器深层漏洞。

测试是发掘编译器漏洞的一类重要方法，传统的编译器测试方法是基于差异测试，通过随机生成C程序，然后对比不同编译器或不同版本的同一编译器对该C程序的编译结果，如有所差异就可以发现编译器漏洞。这样的测试方法需要使用多个编译器或多个版本的编译器，该文作者在2014年就提出了结合EMI（等效模式输入）来进行编译器测试的方法，所谓EMI（等效模式输入）是指针对某一具体输入I，形式上不同，但含义等价的代码段，如图1所示。需要注意的是，EMI一定是针对具体的输入而定的，对于输入I，两段代码P、Q是EMI，但对于另一输入I'，P、Q就不一定是EMI。

利用EMI可以只用一个编译器，和一段源码以及由它产生的EMI变体，来进行编译器测试。具体实现上，针对一个确定的输入I，我们可以利用控制流的检测工具记录程序的运行流程，并对源程序不执行的代码段进行随意的删除，以得到源程序针对输入I的EMI变体。由于部分代码的删除，源代码控制流和数据流的结构发生了改变，而编译器正是依据程序的控制流和数据流来进行优化处理。因此源码和源码的EMI变体经过编译优化后所得到的代码结构会有一定差异。由于源码和源码的EMI是针对输入I等价的，如果在对比最后编译结果时发现了差异，我们就能发现编译器的漏洞。

```
struct com{char a; char b;};
f(struct com x, struct com y)
{
    if (x.a != 10) abort();
    if (x.b != 20) abort();
    if (y.a != 11) abort();
    if (y.b != 21) abort();
}
main()
{
    struct tiny x[2];
    x[0].a = 10;
    x[1].a = 11;
    x[0].b = 20;
    x[1].b = 21;
    f(x[0], x[1]);
    exit(0);
}
```

```
struct com{char a; char b;};
f(struct com x, struct com y)
{
    if (x.a != 10) abort();
    if (x.b != 20) /* deleted */;
    if (y.a != 11) abort();
    if (y.b != 21) /* deleted */;
}
main()
{
    struct tiny x[2];
    x[0].a = 10;
    x[1].a = 11;
    x[0].b = 20;
    x[1].b = 21;
    f(x[0], x[1]);
    exit(0);
}
```

图一：源程序与原程序的EMI

在EMI的概念基础上，为发掘的更多编译器深的层漏洞，本文在提出了一种引导程序突变的方法。由于之前的EMI方法生成的变体数量有限，生成的变体的控制流和数据流的多样性也有限，使我们很难发现一些深层的编译器漏洞。之前方法存在这种问题的主要原因是由于它生成EMI的方法太过简单单一，仅仅只对未执行代码进行随机删除，所得结果没有指向性，数量也有限。而本文所提出的方法除了能删除未执行代码，还能在不执行区域内添加代码。这样可以得到无限的变体数量和多样性的控制流和数据流结构。程序的突变策略基于贝叶斯最优化定理，突变的目标是使得变体程序

Scalable Graph-based Bug Search for Firmware Images

石　建　推荐

“Scalable Graph-based Bug Search for Firmware Images”是国际顶级安全会议CCS 2016年录用的一篇文章。这篇文章利用计算机视觉中成熟的技术来解决基于控制流图的bug搜索方法中的扩展性问题。

随着物联网技术的发展，越来越多的物联网设备进入我们的生活中，这些设备中的漏洞会严重的影响我们的正常生活，所以发现物联网设备中的漏洞比以往更加重要。因为许多集成供应商的产品会依赖于相同的供应商，或者使用相同的SDK进行开发。不同的产品可能会拥有相似的固件。因此，这些设备通常会受到同一个漏洞的影响，检测不同物联网生态系统中的相同漏洞就变得尤为重要。

基于控制流图的bug搜索技术的瓶颈不是图的匹配算法，而是搜索策略。在每次搜索时，都要对图进行两两匹配，这就限制了这个技术不能应用于大规模的数据集。文章利用了计算机视觉中相似图片的搜索策略，提出了一种用于检索物联网设备中bug的搜索方法Genius。

接上页

和源程序的控制流和数据流结构差异达到最大化，然后他们利用MCMC（马尔可夫链蒙特卡罗）技术来实现这个目标。

MCMC———马尔科夫链蒙特卡罗方法产生于19世纪50年代早期，是在贝叶斯理论框架下，通过计算机进行模拟MonteCarlo方法，该方法将Markov过程引入到MonteCarlo模拟中，实现抽样分布随模拟的进行而改变的动态模拟，弥补了传统的蒙特卡罗积分只能静态模拟的缺陷，是近年来广泛应用的统计计算方法从理论上说。贝叶斯推断和分析在实践中，鉴于未知参数的后验分布多为高维、复杂的非常见分布，对这些高维积分进行计算十分困难，MCMC方法通过模拟的方式对高维积分进行计算，进而使原本异常复杂的高维积分计算问题迎刃而解，使贝叶斯方法仅适用于解决简单低维问题的状况大有改观。

利用这个方法他们发现了GCC与LLVM中许多未曾察觉的深层错误。具体来说在19个月的时间内Athena一共发掘出了83个新的编译器漏洞，其中被确认的有72个，并且有68个已经得到了修复。其余是一些开发人员还未来得及考证或是重复的漏洞，具体如表1所示。

	GCC	LLVM	TOTAL
Fixed	38	30	68
Not-Yet-Fixed	2	2	4
WorksForMe	0	3	3
Duplicate	3	4	7
Invalid	1	0	1
TOTAL	44	39	83

表1：Athena 发现的编译器漏洞

黄　冠
2015级博士
研究方向：可信编译与编译安全
Email：2271567680@qq.com

方案：

受到图片检索技术的启发，Genius分为四个步骤：

1）原始特征提取

第一步的目标是提取Attributed Control Flow Graph(ACFG)，ACFG是包含了基本块属性的控制流图。基本块的属性反应基本块的两类特征：统计和结构。表1列出了6个特征，统计特征描述了基本块内的统计信息，结构特征描述了基本块在CFG中的位置特性。

表格 1 基本块的特征

类型	特征	权值
统计特征	字符串常量	10.82
	数字常量	14.47
	转移指令的数目	6.54
	Call 的数量	66.22
	指令的数目	41.37
	算术运算指令的数目	55.65
结构特征	后代的数量	198.67
	中心度	30.66

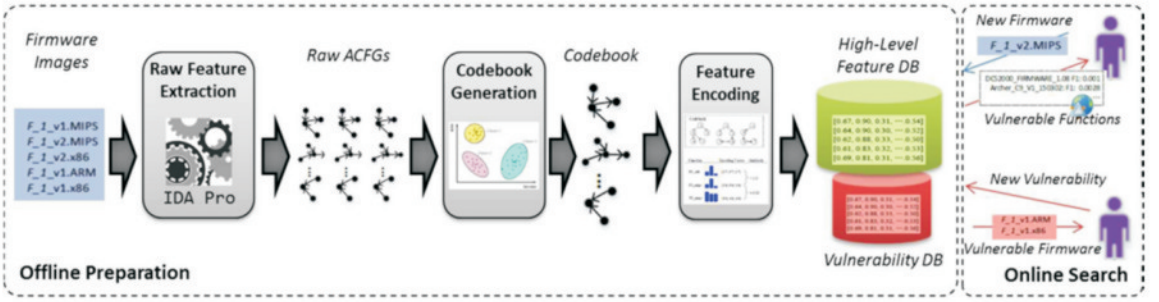


图1 Genius的实现

2）Codebook生成

要生成codebook，首先要进行相似性度量计算。利用二分图匹配算法计算两个图匹配的代价来量化ACFG的相似性。二分图由两个ACFG G1和G2构造，边的权值由两个基本块之间的特征向量计算得出。ACFG的相似性得分是两个图匹配的最小代价。

定义了ACFG之间的相似性度量标准，下一步就是要产生codebook，这个过程可以看做在ACFG数据上的聚类处理。将ACFG测试集聚类到n个集合中，$s=\{s_1, s_2, ..., s_n\}$，c_i是s_i的聚类中心，使得所有ACFG到它的聚类中心的距离之和最小。$C=\{c_1, c_2, ..., c_n\}$就是codebook。

3）编码特征

得到codebook后，特征编码的过程就是将原始特征ACFG映射成为更高等级的数字向量。文章中采用VLAD编码，向量的每个元素是ACFG和codebook中一个集合的相似度距离，即ACFG和c_i的距离。编码后的特征向量能更好的应用于bug搜索系统中。

4）在线搜索

虽然编码后的特征向量可以直接用于搜索，但直接搜索在面对上百万的数据时并不高效，所以Genius采用局部敏感哈希对特征向量进行索引。

当查询时，先计算函数的特征向量，然后在数据集合中搜索与该函数距离最近的那个函数。

测试：

实验结果表明Genius在速度和精度方面优于最先进的bug搜索方法。Genius在跨x86，arm和MIPS三个架构的4.2亿个函数集上进行测试，平均能在1秒内完成搜索。在50个搜索结果中，确认了23个含有漏洞的固件。

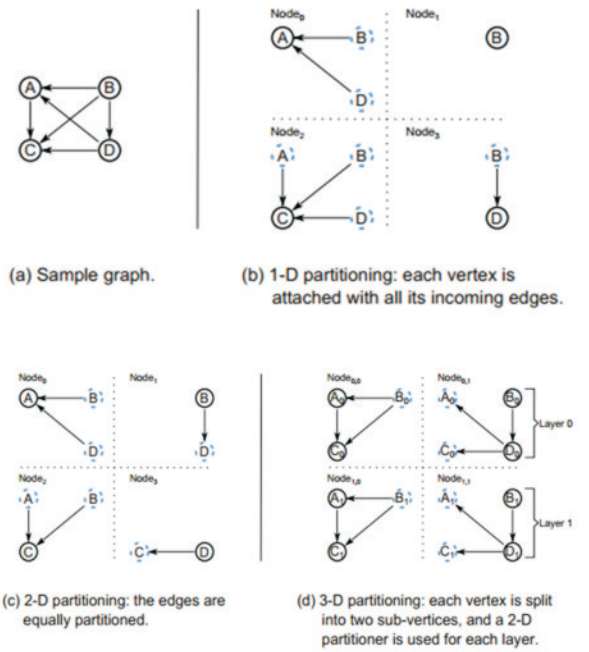
石　建
2015级博士
研究方向：二进制程序分析
Email：shijianwh@hust.edu.cn

Exploring the Hidden Dimension in Graph Processing

唐训祝　推荐

“Exploring the Hidden Dimension in Graph Processing”是被计算机国际顶级会议OSDI 2016录用的一篇文章。本文从一个新的维度提出一种新的图数据划分方法，提高分布式图计算系统的并行度，减少通信开销，提高系统计算效率。由于传统分布式图计算系统认为顶点以及边是不可分割的，所以把任务划分等价于图划分，但本文发现机器学习及数据挖掘应用可转化为顶点和边均为向量的二分图。在此类图中，顶点是可分割的，因此，可利用顶点向量元素之间的并行性进行任务划分，提高计算的并行度和效率。基于此任务划分方法，本文还设计了新的分布式图计算框架CUBE，可以极大地减少通信开销，提高计算的并行度和效率。

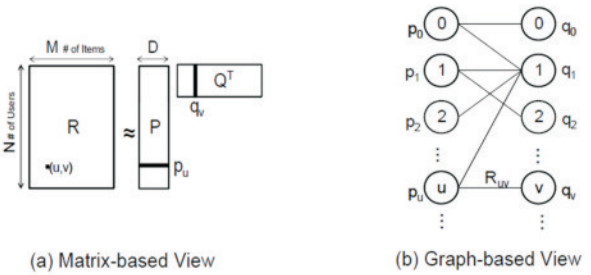
分布式图计算系统的任务划分策略决定了系统负载均衡程度以及通信开销。当前的分布式图计算系统假设每个顶点和边是不可分割的，因此，将节点的任务划分等价于对图切割。现有的图切割策略包括1D划分和2D划分，如图一所示。1）1D划分：将图中的所有顶点均匀分配到集群的各个结点上，这对于随机图比较高效。然而在实际图数据中，不同顶点的度差距较大，往往具有幂律分布。1D划分策略会导致负载不均衡。Google的Pregel框架就是典型的1D划分。2）2D划分：为避免1D划分导致的负载不均，2D划分实现边的切割，将所有边均匀分配给每个节点，解决了由于节点度数差距较大导致的负载不均衡问题。



图一：图划分策略

本文发现，机器学习与数据挖掘应用可转化为顶点/边为向量的二分图，由于向量是可分割的，因此在此类图中，任务划分不等价于图切割。例如协同过滤应用 (如图二)。对于给出的(user,item)数据集矩阵R(N*M大小)，需将找到低秩矩阵P(N*D)和Q(M*D) (D为特征向量的维度)，使其满足$R=P \times Q^T$。而矩阵相乘可转化为二分图，图二(b)中p(i)为图二(a)矩阵P的第i行，图二(b)中q(j)为图二(a)矩阵Q的第j列，

p(i)与q(j)之间的边是矩阵R的(i,j)元素。基于机器学习应用的顶点向量化特征，本文提出了新的划分方法，3D划分，如图一(d)所示。将集群的所有结点分成多层，每层结点数相同，不同的层处理顶点向量的不同维度。在每层内部的所有结点之间按照2D划分对图拓扑结构进行切割和计算更新。图一(d)将所有集群结点分为两层，每层内部利用2D切割将边均匀分配到结点上。



(a) Matrix-based View (b) Graph-based View

图二：协同过滤应用

本文基于3D划分设计了新的图计算系统CUBE。CUBE系统扩展了传统的GAS编程模型，提出了一种新的UPPS编程模型 (Update, Pull, Push, Sink)。Update：该操作需要所有的顶点/边的值来计算新值，粗略地从竖直方向上(向量元素之间)更新元素的顶点和边进行操作。由于顶点和边可能会被划分为几层，每一个节点更新都需要和其它层同时进行；Push, Pull, Sink：这三个操作是进行水平上(顶点与边)的更新，对于每一条边(u, v)，push操作使用顶点u和边(u, v)数据去更新顶点v，pull操作则是使用顶点v和边(u, v)数据去更新顶点u，而sink操作则是使用u和v数据去更新边(u, v)。

实验结果如图三所示，当D(顶点向量维度)一定，层数(表中的括号内表示层数)一定，CUBE系统明显优于PowerGraph及PowerLyra。

总结：本文提出在传统的分布式图计算框架中“任务划分=图划分”的假设是不正确的，在机器学习及数据挖掘中，顶点/边的属性是向量而不是单个值。本文作者发现这个特点并提出3D划分算法，将顶点/边的向量元素分配到不同的节点，提高计算的并行度。基于3D划分，本文作者还设计了新的图计算系统CUBE，实验结果表明基于3D划分的CUBE框架比基于2D划分的PowerLyra提高到4.7倍(是PowerGraph的7.3倍)。

D	# of workers	Libimseti GD	Libimseti ALS
64	8	9.78 / 9.56 / 2.04 (2)	70.8 / 70.4 / 46.7 (8)
	16	8.04 / 8.16 / 1.95 (4)	72.6 / 71.5 / 37.6 (16)
	64	6.82 / 6.89 / 2.59 (4)	87.0 / 86.8 / 28.7 (64)
128	8	14.99 / 14.94 / 3.87 (2)	261 / 258 / 193 (8)
	16	12.81 / 12.91 / 2.62 (4)	270 / 270 / 135 (16)
	64	11.64 / 11.62 / 3.33 (8)	331 / 331 / 109 (64)

D	# of workers	LastFM GD	LastFM ALS
64	8	12.0 / 8.98 / 3.45 (2)	124 / 73.5 / 70.9 (8)
	16	10.5 / 8.22 / 2.59 (2)	128 / 69.5 / 61.6 (16)
	64	10.4 / 9.86 / 2.48 (4)	158 / 111 / 57.6 (64)
128	8	19.0 / 13.8 / 4.74 (2)	465 / 263 / 270 (4)
	16	17.6 / 13.5 / 3.35 (4)	490 / 253 / 200 (16)
	64	18.6 / 17.8 / 3.47 (8)	Failed / Failed / 230 (64)

D	# of workers	Netflix GD	Netflix ALS
64	8	34.4 / 27.7 / 6.03 (1)	256 / 204 / 110 (2)
	16	26.7 / 17.3 / 3.97 (1)	186 / 107 / 60.4 (2)
	64	18.3 / 7.42 / 4.16 (1)	179 / 66.0 / 42.5 (8)
128	8	51.8 / 38.6 / 9.65 (1)	865 / 657 / 463 (1)
	16	41.9 / 23.0 / 6.59 (1)	669 / 340 / 258 (2)
	64	30.6 / 11.3 / 6.55 (2)	Failed / 239 / 118 (8)

图三：PowerGraph/PowerLyra/CUBE 在三个数据集下执行时间比较

唐训祝
2016级博士生
研究方向：图计算
Email：tangxz@hust.edu.cn

Large-scale cluster management at Google with Borg

黄 航 推荐

“Large-scale cluster management at Google with Borg”是Google在EuroSys 2015上发表的论文，这篇文章主要介绍了Borg系统架构和特点，重要的设计决策，并定量分析它了的一些策略以及作者十年以来的运维经验和学到的东西，披露了在容器集群编排上过去极少提及的一些技术细节。

文章介绍的Borg就是一个集群管理器，它负责对来自于几千个应用程序所提交的job进行接收、调试、启动、停止、重启和监控，这些job将用于不同的服务，运行在不同数量的集群中。

文章指出Borg系统提供了四个主要的好处：

（1）通过权限控制、高效的任务包装、超售、和进程级别性能隔离实现了集群资源的高利用率，做到了跨多个数据中心的资源利用率最大化；

（2）高可靠性和高可用性的操作，并支持应用程序做到高可靠高可用；

（3）让我们在数以万计的机器上有效运行，并且通过减少相关故障概率的调度策略最大限度地减少故障恢复时间；

（4）隐藏资源管理和故障处理细节，使其用户可以专注于应用开发；

而Brog的主要系统架构是：一个Borg的Cell包括一堆机器，一个逻辑的中心控制服务叫做Borgmaster，和在每台机器上跑的Borglet的agent进程(见图1)。所有Borg的组件都是用C++写的。

这套架构中包含了以下组件：

· 单元（Cell）——将多个机器的集合视为一个单元。单元通常包括1万台服务器，但如果有必要的话也可以增加这个数字，它们各自具有不同的CPU、内存、磁盘容量等等。

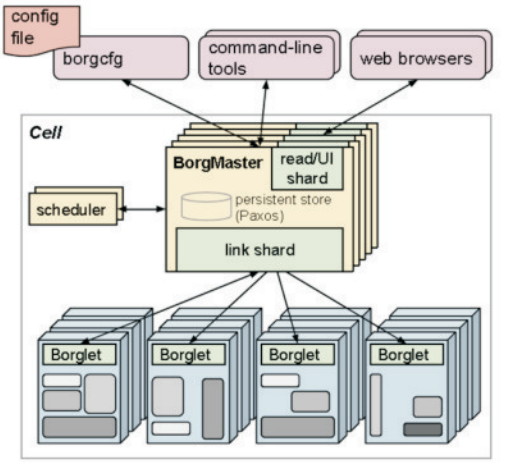


图1：Borg的高级别架构图

· 集群 —— 一般来说包含了一个大型单元，有时也会包含一些用于特定目的的小单元，其中有些单元可以用做测试。一个集群通常来说限制在一个数据中心大楼里，集群中的所有机器都是通过高性能的网络进行连接的。一个网站可以包含多个大楼和集群。

· Job —— 是一种在单元的边界之内进行执行的活动。这些job可以附加一些需求信息，例如CPU、OS、公开的IP等等。Job之间可以互相通信，用户或监控job也可以通过RPC方式向某个job发送命令。

· Task —— 一个job可以一个或多个任务组成，这些任务在同一个可执行进行中运行。这些任务通常是直接在硬件上运行的，而不是在虚拟环境中运行，以避免虚拟化的成本。任务的程序

ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems

周 放 推荐

MICRO 2015 (The 48th International Symposium on Microarchitecture) 于2015年10月15-18日在美国夏威夷怀基基海滩召开，是系统结构领域五大顶级会议之一。本文作者Jinglei Ren与Onur Mutlu来自美国卡内基梅隆大学SAFARI实验室，其在内存与NVM技术领域处

接上页

是静态链接的，以避免在运行时进行动态链接。

· 分配额（alloc）——专门为一个或多个任务所保留的机器资源集。分配额能够与运行于其上的任务一起被转移到一个不同的机器上。一个分配额集表示为某个job保留的资源，并且分布在多台机器上。

· Borglet ——Borglet是部署在cell的每台机器上的本地Borg代理程序。它启动停止task；如果task失败就重启；通过修改OS内核设置来管理本地资源；滚动debug日志；把本机的状态上报给Borgmaster 和其他监控系统。Borgmaster 每过几秒就会轮询所有的Borglet来获取机器当前的状态还有发送任何请求。这让Borgmaster能控制交流频率，避免一个显式的流控机制，而且防止了恢复风暴。

· Borgmaster —— 一个控制器进程，它在单元的级别上运行，并保存着所有Borglet上的状态数据。Borgmaster将job发送到某个队列中以执行。Borgmaster和它的数据将会进行五次复制，数据将被持久化在一个Paxos存储系统中。所有的Borgmaster中有一个领导者。

· 调度器 ——对队列进行监控，并根据每个机器的可用资源情况对job进行调度。当一个job被提交的时候，Borgmaster会把它持久化的存储在Paxos存储上，然后把这个job的task放到等待(pending)的队列里面去。这个队列会被scheduler异步的扫描，然后分发task到有充足资源的机器上。scheduler主要是处理task的，不是job。扫描从高优先级到低优先级，在同个优先级上用round-robin的方式处理，以保证用户之间的公平性和避免头上的大job阻塞住。调度算法有2个部分：可行性检查(feasibility checking)，找到一台能跑task的机器，和打分(scoring)，找个一个最合适的机器。

文章最后，作者介绍了十年以来在生产环境操作Borg得到的一些定性经验教训。教训主要有：Jobs是唯一的task分组的机制、一台机器只有一个IP把事情弄复杂了、给资深用户优化而忽略了初级用户。经验是：Allocs是有用的、集群管理比task管理要做更多的事、反观自省是至关重要的、master是分布式系统的核心。

黄 航
2016级硕博
研究方向：容器虚拟化
Email：huanghang@hust.edu.cn

于最前沿水平。

按位寻址NVM器件技术的发展，其结合了主存和外存二者的好处，打破传统存储层级之间的界限。这为应用带来了巨大的好处：允许应用使用load/store指令直接操作主存中的Persistent数据，而不需要像以前一样需要经过storage设备，转换数据格式，执行开销巨大的系统调用。Persistent Memory对内存系统提出了一项很重要的要求：当system failures发生时，如何保证系统中的数据完整性，不受写顺序和部分更新的影响。Persistent memory需要保证存储在NVM中的数据，在经历系统crash之后，能够在系统重启和恢复过程中，恢复到一个一致性的版本。

大部分现有保证crash一致性的设计都需要编程人员的参与。应用开发人员需要遵循一定的编程模型和软件接口，清楚地操纵数据在Persistent Memory中存储和移动，以保证数据不受重新排序和部分更新的影响。这种方式允许编程人员对数据进行全权控制，但却存在许多的局限性：（1）程序员需要付出极大工作量来编写新程序，清楚地定义Persistent 和Transient数据结构。（2）旧代码编写的应用无法享有新Persistent Memory 带来的好处。（3）现有大部分Persistent Memory设计，为保证一致性，都要遵循事务性语义进行版本控制和写顺序控制。这种做法面临严重的扩展性挑战。（4）Persistent Memory应用的部署基于特殊的软件接口和运行的系统，不同的计算机系统在这些方面有很大的不同，这减弱了应用的可移植性。

传统的Logging-based Systems和Copy-on-Write based Systems能够提供软件透明地系统一致性支持，但这两个系统都存在巨大的开销。Logging会在每一次的数据更新时，将数据和元数据都存在log表项中。Log回放过程会增加system failures的恢复时间，抵消了NVM替换缓慢块设备所带来的快速恢复的好处。C-O-W则有两个缺点：其一，Copy操作时间很长并且需要较长的暂停时间。其二，C-O-W会不可避免地将copy未修改的数据，因此会消耗额外的NVM带宽。

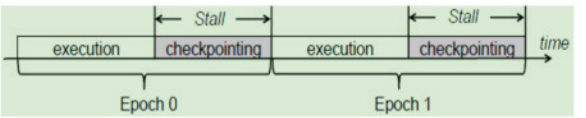
因此，我们设计了一个有效地，具有软件透明性的系统Transparent hybrid Non-volatile Memory(ThyNVM)，用以保证Persistent Memory系统中的Crash Consistency。ThyNVM 使用了一种新的双重粒度的checkpoint机制，能够有效地覆盖checkpointing时间和应用执行时间。经过测试，我们的系统在只有DRAM的系统上仅有4.9%的开销即可提供crash consistency 保证。

在Checkpoint系统中，Checkpointing性能和元数据开销是最为关注的两个因素。他们之间存在复杂的tradeoff关系：首先，Checkpoint机制的元数据开销由其Checkpointing数据的粒度所决定，其次，Checkpointing延迟受工作数据数据位置的影响。经过对两种粒度优劣势的分析，我们决定使用多粒度的Checkpointing：（1）Cacheline粒度的Checkpointing将数据复制保存在NVM中；（2）Page粒度的Checkpointing 将工作内存保存在DRAM中，并将更新写回NVM。

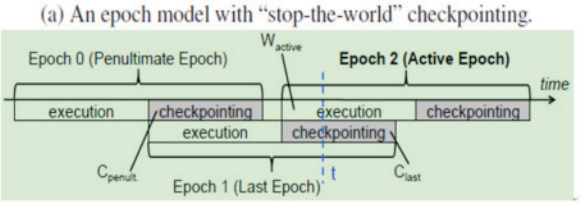
为了加快Checkpoint机制的执行，我们设计了一个新的执行模型，如图1(a)，能够覆盖应用运行阶段和Checkpointing阶段，使得二者能够并行执行，因此减少了Checkpointing阶段所需的应用暂停时间。这种做法在保证crash consistency的时候面临着两项挑战：（1）将运行阶段和Checkpointing阶段并行执行，可能会引起两个阶段的数据相互覆盖。我们设计了合理的数据隔离机制，成功地避免了这一问题。（2）在发生system failures的时候，会出现数据丢失的情况，需要始终保证系统中存在一份安全可用的一致性数据。

我们设计将不同的数据存在不同的内存区

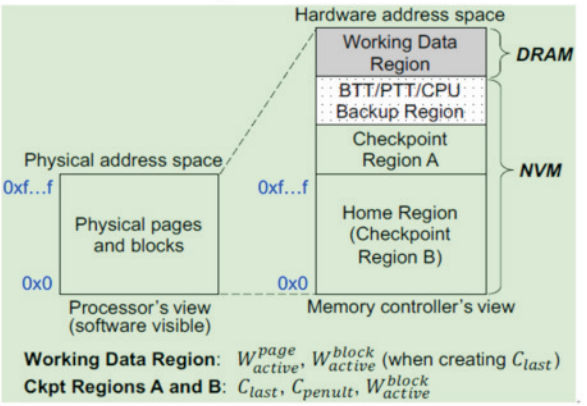
域中，使得唯一的一份一致性的数据能够在系统恢复之后被CPU使用。如图1(b)，在系统中存在多种类型的数据。Home Region(Checkpoint Region B)中，存放只读类型的数据，CPU可以直接通过其物理地址进行操作，这部分数据不进行Checkpointing。Working Data Region中存放的是需要进行Checkpointing的工作数据，其对应Checkpointing数据存放在Checkpoin Region A中。BTT/PPT/CPU Backup Region区域中，存放的是系统中元数据BTT(block translation table)，PTT(page translation table)和CPU中寄存器备份数据。



(a) An epoch model with "stop-the-world" checkpointing.



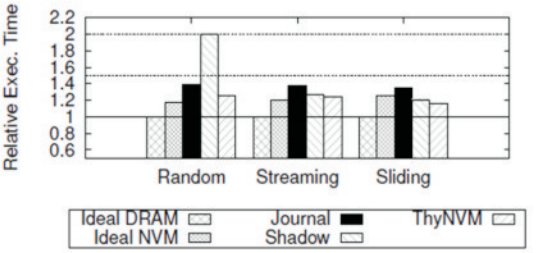
(a) 执行阶段图



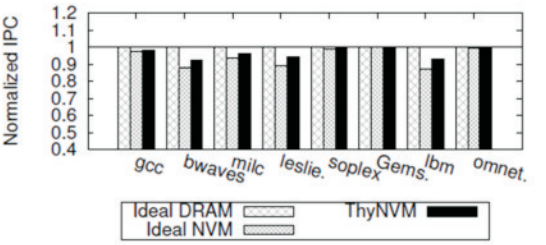
(b) 数据放置图

图1

在发生system failures之后，系统恢复将遵循三个主要步骤进行恢复。（1）内存控制器重新加载保存在NVM中的BTT和PTT，恢复元数据管理信息。（2）内存控制器通过页写回的方式利用Checkpointing数据恢复Working Region的数据。（3）在元数据和内存数据都恢复好了之后，系统将重新加载保存在NVM相应区域中的Processor寄存器中的数据。



(a) Micro-benchmarks



(b) SPEC CPU2006

图3

文中使用了多组benchmark进行测试，这里仅列出两组，如图2。图2(a)中是使用Micro-benchmarks所得结果可以看出，我们的系统性能领先journaling和shadow paging分别为10.2%和14.8%。图2(b)中是使用SPEC CPU2006测试所得出的结果，可以看出我们的系统与理想DRAM系统比，性能仅落后3.4%。可以认为ThyNVM系统在保证数据一致性的同时，与理想系统相比所带来的开销基本没有。

周　放
2014级博士
研究方向：内存计算
Email：364558434@qq.com

ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware

张启夏　推荐

“ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware”是国际顶级会议SIGCOMM'16收录的一篇论文，该论文提出了一个基于FPGA加速网络功能（Network Functions，NFs）平台的ClickNP，它具有高度灵活性和模块化架构，具有较高的性能和较低的延迟，并且使用高级语言编程，使软件程序员可以方便地开发。

由于基于硬件的传统网络设备不够灵活，几乎所有现有的云服务提供商，例如微软、亚马逊和VMWare都已经在服务器上部署了软件网络功能。目前，高度灵活的软件网络功能是实现云中多租户的关键组件。然而，商用服务器上的软件包处理过程面临两点限制：一是往往受服务器容量限制，二是等待延迟变化大且平均较长。虽然软件NF可以使用更多的服务器扩容，但这样做也会显著增加成本。

本文作者指出，为了克服软件包处理的限制同时保持灵活性，最新的研究已经提出使用图形处理单元（Graphics Processing Units，GPU），网络处理器（Network Processors，NP）或可重配置硬件，如现场可编程门阵列（Field Programmable Gate Arrays，FPGA）来为NF加速。与GPU相比，FPGA更节能；与专用NP相比，FPGA具有更多的灵活性，能为任何服务虚拟地重配置任何硬件逻辑。FPGA因为价格便宜，目前已经在数据中心大规模部署。然而，FPGA主要使用低级硬件描述语言（HDL）编程，它们难以编码和调试，并且对于大多数软件程序员来说，HDL几乎是不可访问的。

为了克服这些限制条件，并充分利用FPGA的优势，本文作者提出了一个基于FPGA加速平台——ClickNP，它用于加速在商用服务器上的高度灵活和高性能的NF。ClickNP分三个步骤解决FPGA的编程难题：

（1）ClickNP提供了一个模块化架构，类似于著名的Click模型，如图1所示。

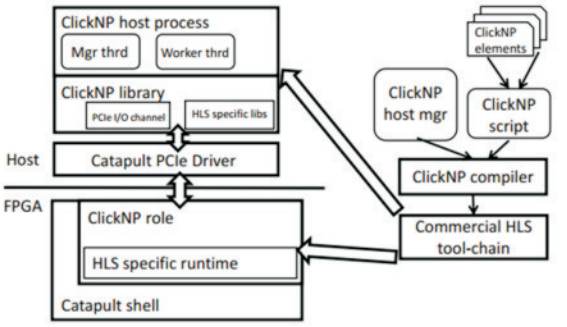


图1 ClickNP系统架构图

（2）ClickNP使用高级语言，如C/C++语言，并且是跨平台的。利用商业高级综合（HLS）工具，它可以在CPU上编译为或FPGA使用的低级硬件描述语言（HDL）。

（3）本文作者开发了一个高性能的PCIE I/O通道。它可以为在CPU和FPGA上运行的元件之间提供高吞吐量和低延迟的通信。这个PCIE I / O通道不仅支持联合CPU-FPGA处理——允许程序员自由分配其处理过程，而且对调试有很大帮助，因为程序员使用熟悉的诊断工具来调试。

ClickNP在元件级和元件内都利用了FPGA并行性。作者还采用一系列优化技术来有效利用FPGA中的大规模并行性。

为了评估ClickNP的灵活性，作者基于ClickNP实现了5个通用的NF，包括：（1）高速流量捕获器和发生器（2）Openflow防火墙（3）IPSec网关（4）第4层负载平衡器（L4LB）（5）pFabric调度器。

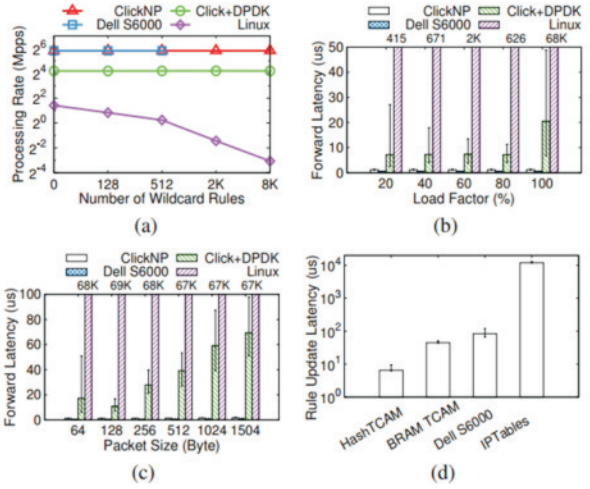


图2 防火墙性能

通过实验评估表明，这些优化使ClickNP实现了高达2亿个数据包/秒的高数据包处理吞吐量和超低延迟，在大多数应用中任何数据包大小的延迟均小于2μs。与CPU和GPU加速的CPU相比，ClickNP把软件NF的吞吐量上提高了10倍和2.5倍，同时分别将延迟降低10倍和100倍。如图2显示了Openflow防火墙的在吞吐量和延迟上的性能测试结果，图3显示了IPsec网关的性能。

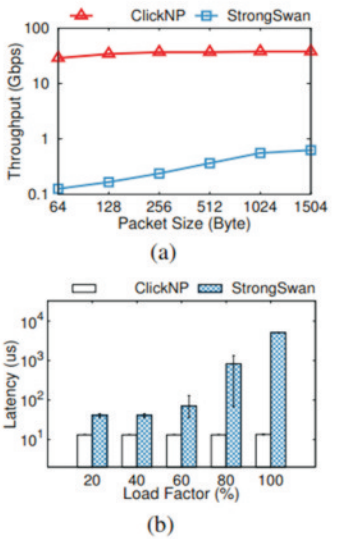


图3 Ipsec网关性能

作为首次应用于NF的可重构硬件，这项工作有这很大的贡献，特别是克服FPGA编程的困难。基于此，可以逐步调度一些增强或进一步的工作，诸如：

（1）通过深度优化HDL综合工具来提高FPGA编译速度。

（2）开发更多的通用模块，实现更多的通用NF，以进一步降低程序员的负担。

（3）将FPGA应用于网络的前沿技术，例如SDN，NFV等。

张启夏
2016级博士
研究方向：NFV、SDN
Email：zhangqixia427@hust.edu.cn

Data Tiering in Heterogeneous Memory Systems

吴文超　推荐

“Data Tiering in Heterogeneous Memory Systems”是被计算机国际顶级会议EuroSys 2016年录用的一篇论文。本文主要针对混合异构内存环境下的数据中心应用，通过大量的量化分析和实验证明：合适的数据分类与放置，可以有效屏蔽将NVM引入内存所带来的性能下降。在此基础上，本文设计和实现了一个自动识别数据结构访问模式的工具，以及一种NVM内存管理框架X-Mem，实现了内存最优化映射。X-Mem框架实现了在混合异构内存下的大容量数据的高效放置和处理，达到了高性能/经济比。

随着大数据时代的到来，数据中心的各种应用面对的数据量呈现爆炸式增长，一方面。需要处理越来越大的数据集，另外一方面，传统DRAM由于工艺和物理限制，其拓展性有限。一种替代方案是：将数据频繁溢出到磁盘，但是其带来的性能下降又难以忍受。在内存计算的需求下，新型存储介质的出现为数据中心应用解决内存扩张难题带来了新的方案。

Parameter	DDR-DRAM	NVM
Capacity per CPU	100s of GBs	Terabytes
Read Latency	1×	2× to 4×
Write bandwidth	1×	$\frac{1}{8}$× to $\frac{1}{4}$×
Estimated cost	5×	1×
Endurance	10^{16}	10^{6} to 10^{8}

表1 NVM和DRAM性能对比

新型非易失性存储器（NVM）主要有相变存储器（PCM），自旋转移矩磁随机存储器（STT-RAM），阻变随机存储器（RRAM）等，这些新型存储器一般可以达到DRAM的读速度，以及接近外存的容量。相对于DRAM具有密度高，价格低，非易失性，读延迟低的优点。表 1 列出了主流的NVM的性能参数。同时，NVM也具有一些固有缺点，以PCM为例，具有读写不对称性，读速度快而写速度慢，写的功耗大，写次数严重受限的特点。图1为PCM读写流程图。

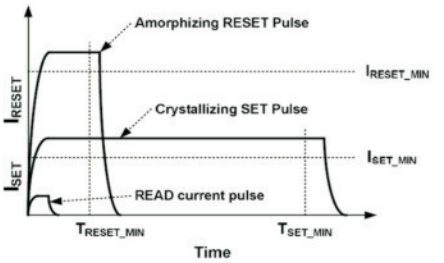


图1 PCM读写流程图

目前NVM应用到系统里面主要有几种方式，1. 作为CPU可以直接字节寻址的内存,与DRAM处于同级的平行架构。2. DRAM作为NVM的cache的垂直架构，3. NVM作为外存来持久化数据。4. NVM作为内存和外存的混合方式以及垂直与水平架构的混合等方式。

虽然NVM可以极大缓解内存紧张问题，但是由于NVM在访问速度和带宽等方面的缺陷，直接将传统应用运行于NVM上会带来严重的性能下降。基于NVM的读写特性，本文提出了一种数据访问模式的量化模型，和基于量化结果的数据放置策略，并实现了一种NVM内存管理框架X-Mem。在将大部分DRAM替换成NVM的情况下，有效的减少性能损失，极大降低存储成本，获得很高的性能／经济比。

X-Mem框架的分析流程如下：

1）：应用程序员识别应用程序中的主要数据结构。

2）：使用X-Mem提供的API对数据结构进行人工标记。

3）：分析数据结构的访问频度以及访问模式，得到数据结构的存储优先级（优先级越高，越优先分配到速度快的DRAM）。

4）：应用代码运行。采用第三步分析的结果，将数据分配到合适的内存区域（DRAM或NVM），在运行过程中实现数据的动态迁移和管理。

	Dependent	Independent
Sequential	NA	Streaming
Non-sequential	Pointer chasing	Random

表2 数据的访问模式

X-Mem框架的核心是分析和发现数据结构的访问模式，并对此建模。和以前的论文中仅仅考虑数据的访问频度不同，本文基于多种因素对内存冷热数据进行划分，从应用层出发，针对不同的应用和不同的数据结构，将数据结构从访问依赖性和顺序性2个维度进行划分（如表2），对随机访问，指针依赖，以及流式访问这3种主流的访问模式进行分析，分别测试每种数据结构在三种访问模式下对应的频度分布和平均访问延迟。

$$A=\frac{\sum_{i=1}^{n} C_i S_i}{\sum_{i=1}^{n} C_i}$$

同时计算数据放到混合内存中的总延迟：

$$\hat{A}=\sum_{r\in Regions}^{n}\ \sum_{p\in Patterns}^{n} F_{D(r)}(P)\ L(p,T(r))$$

该问题等价于在DRAM容量有限的情况下，上述延迟值最小的问题。可转化为选取部分数据，从NVM迁移放到DRAM中，使之得到的收益最大化的问题。可利用贪心算法求解此问题，将每个数据按照从NVM迁移到DRAM中受益值从大到小进行排序，优先迁移受益值大的数据。

本文选取了三种带宽和延迟敏感度不同的应用作为测试案例，分别是图分析架构GraphMAT，内存数据库VoltDB，以及键值对存储MemC3。分别对比了在纯NVM内存，纯DRAM内存，以及DRAM和NVM不同混合比例，不同性能参数（带宽和延迟）下的对比实验。实验结果表明，在X-Mem框架下，在内存总量不变的基础上，GraphMat仅仅需要1/16的DRAM，以及在VoltDB和MemC3下只需要1/4的DRAM，应用的性能下降只有13%到40%。考虑DRAM和NVM的不同价格，这些应用具有2到2.8倍的性能/价格加速比。

本文也存在一些需要完善的地方，首先，在数据放置时，只考虑减少延迟这一项指标，而没有考虑到NVM的寿命问题。其次，X-Mem框架需要人工对应用程序的数据结构进行识别和标记，增加了编程人员的额外负担，要求编程者必须要对数据结构有着深刻的理解。最后，X-Mem的预处理阶段，大量的数据访问跟踪时间开销较大，只适合进行离线操作，无法进行运行时操作。

吴文超
2016级博士生
研究方向：混合异构内存，内存计算，机器学习
Email：1366199330@qq.com

GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server

姚琼杰　推荐

GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server是计算机系统领域顶级会议EuroSys2016录用的文章。本文是来自于CMU的邢波团队开发的基于GPU的参数服务器结构的深度学习训练系统。

大规模的深度学习在图片识别，语音和自然语言处理方面取得了很好的性能，然而训练一个深度学习模型耗费很长的时间，可能几个星期或几个月，这对于实际深度学习应用的部署是不能容忍的。目前，GPU成为训练深度学习模型的标准计算设备，但单GPU 计算能力仍显不足，而使用基于CPU的参数服务器的分布式深度学习系统，虽然可以利用多个GPU，但性能较低，基于这一现状，本文开发GPU指定的参数服务器结构的深度学习系统GeePS。

本文主要有三个贡献：

1）第一个GPU指定的参数服务器系统。传统利用多GPU训练的方式主要利用现有基于CPU的参数服务器系统，如图1。这种多GPU训练方式存在严重的CPU与GPU间的数据传输开销，GPU只负责梯度计算，最终的梯度值传回到CPU端进行参数更新。本文开发的系统解决了这一问题，并用计算时间来掩盖数据传输开销，如图2。虽然本文的设计有其独特之处，但此处的说法有点一家之言，因为MXNet等系统可在GPU端进行参数更新。

2）GeePS的扩展性能好。GeePS设计了分布式共享内存的key-value数据库，将现有的GPU代码简单修改，数据存放于GeePS的数据库，即可部署到多GPU上并取得良好的系统扩展性。与单GPU相比，16台机器可以取得13倍的加速。

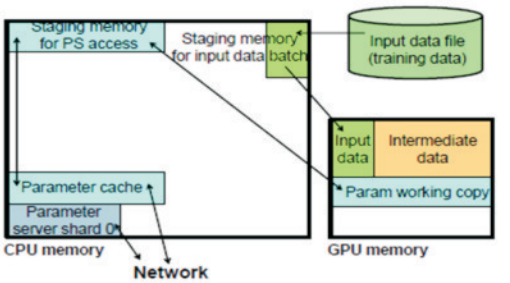


图1 基于CPU参数服务器系统

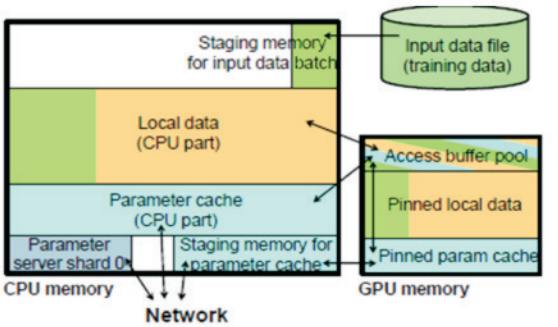


图2 基于GPU参数服务器系统

3）突破了GPU的内存限制，能训练比GPU内存更大的模型。这是GeePS最出色之处。GeePS采用的并行方式是数据并行，支持三种策略：同步数据并行，有限异步数据并行和完全异步数据并行。由图2系统结构图可知，GPU内存中存入训练数据，模型参数和模型中间计算值，因此，这些数据的大小限制了GPU所能训练模型的大小。这是当前多GPU训练系统面临的主要问题，本文解决了这一问题，精心部署内存，利用

Going Deeper with Embedded FPGA Platform for Convolutional Neural Network

张 伟 推荐

“Going Deeper with Embedded FPGA Platform for Convolutional Neural Network”录用于FPGA'16，是清华大学汪玉教授团队在使用FPGA加速CNN网络的优秀工作。

近年来，卷积神经网络（CNN）大量应用于计算机视觉并取得巨大成功。但由于CNN网络属于计算密集型和资源消耗型计算，所以很难被集成到嵌入式平台中。FPGA是加速CNN网络的理想平台之一，但是内存带宽和片上存储限制了其加速性能。

本文深入分析使用嵌入式FPGA平台加速CNN网络，并针对Image-Net大规模图片分类设计了CNN加速器。首先，本文深入分析了最新的CNN模型，揭示了卷积层以计算为主，而全连接层以内存访问为主。本文采用动态精度的数据量化，并设计了高效的卷积计算器以提高带宽和资源的利用率。实验表明，对于VGG16 网络，本文采用的8/4-bit量化方法仅仅有0.4%的准确率损失。此外，本文针对设计了数据重排策略以提升外部存储带宽。最终，本文实现VGG16-SVD 模型最为案例分析。该系统在Xilinx Zynq ZC706平台上采用16-bit数据量化实现了帧速率为4.45fps，top-5准确率为86.66%。在150MHz工作频率下，卷积层平均性能为187.8GOP/s，整个模型的综合性能为137.0GOP/s。这一性能相较于此前的研究有显著提升！

接上页

神经网络训练的迭代性质，预测每层网络的数据访问模式，将暂时不参与计算的数据存放在CPU内存中。若要使用CPU端的数据，则由服务器端库将数据从CPU端传输到GPU端，这个传输过程在后端进行，并与计算过程重叠，对训练总时间影响不大。但令读者比较疑惑的是作者在系统实现章节却未祥述细节，虽然可利用训练算法的迭代性能预测将使用的数据，但何时传输数据才能做到计算与数据传输的良好重叠呢？设置时间断点还是统计每轮迭代中主要步骤的计算时间，绘制函数启动时间表？

除以上贡献外，本文发现在到达相同的分类精度，同步数据并行比其它两种并行方式速度更快，如图3。这一发现与许多论文的观察相反，这说明选择哪种并行方式是分布式深度学习系统设计中的一个折中选择。

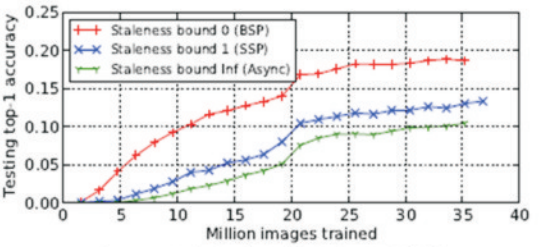


图3 不同数据并行策略的测试精度

姚琼杰
2012级博士
研究方向：深度学习并行训练
Email：yaoqiongjie@gmail.com

如图 1所示，典型的CNN网络中，图片首先经历若干CONV（卷积层），再接若干FC（全连接层），这些层的计算都是顺序执行。CONV层属于计算密集操作，所需参数并不多，但是FC层则包含上亿个权重值，因此属于访存密集型。此前的一些研究工作，只针对CONV层进行计算优化，而不是针对整个CNN 网络；也有些工作针对模型压缩。本文通过深入分析，发现CONV所需的计算操作占了整个CNN网络的绝大部分；FC层则占用了大部分的权重参数。本文从CNN模型的计算和访存着手，应用多种优化方法并挖掘并行性，整体考虑CNN网络的优化问题。

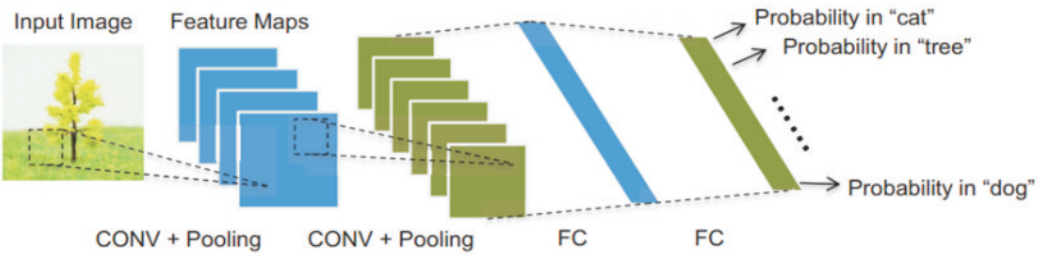


图 1 CNN网络

针对FPGA平台，通常短的定点数代替长浮点数能够显著减小内存占用和带宽需求。如图2所示，Fractional部分bit长度对于不同的层和feature map动态变化着并在同一层内保持不变，这样能够将每层的截断误差降到最小。

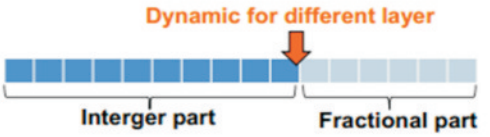


图2 动态精度数据量化

如图 3所示，系统分为PS和PL两部分，PS部分使用通用处理器，负责设置DMA，数据预处理以及结果输出。PL部分包含片上缓存、控制器以及计算处理器。针对计算，该系统设计了多个处理单元（PE）。该系统实现了多种级别的并行，包括采用2D卷积计算器实现操作级并行，通过多个卷积计算器的同时运算实现内部输出级并行，并通过布置多个PE实现外部输出并行。

最后，通过各层计算中的数据存储模式分析总结出了适合卷积和全连接层的最佳数据重排方式，极大地减少了传输数据所需的DMA事务数目。

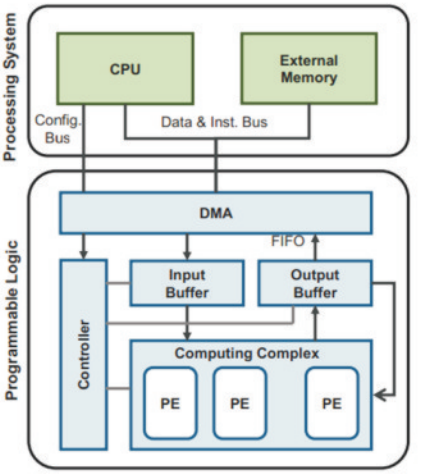


图3 系统架构

该工作细致深入地分析CNN网络中的计算模式以及数据量化对准确率的影响，对使用FPGA高效地实现CNN加速有莫大借鉴意义。同时通过实现VGG16-SVD，给出了细致的优化方式和依据，对于厘清诸多优化选项意义非凡。

张　伟
2013级博士生
研究方向：云端融合
Email：alanzw@163.com

R-Storm: Resource-Aware Scheduling in Storm

林昌富　推荐

“*R-Storm*: Resource-Aware Scheduling in Storm”是被计算机国际顶级会议Middleware 2015录用的一篇文章。本文主要针对分布式实时流处理系统Storm中默认的轮询调度策略由于忽略资源的需求和可用性所导致的资源调度不高效的问题，在Storm的基础上提出了一种资源感知的调度策略，这种调度策略通过最大化系统资源利用率的同时最小化网络延迟来增加系统整体吞吐量。

随着大数据时代的到来，每天都会产生海量的数据，IBM的一个分析报告指出，2014年每天大约有2.3泽字节（$2.3*10^{21}$）的数据被生成。越来越多的应用需要能够实时地处理流式到达的数据，快速响应用户的查询，从而及时发现并利用数据潜在的价值。Storm是一个用来分布式实时处理大规模流数据的系统，被广泛应用于工业界，具有很好的性能。

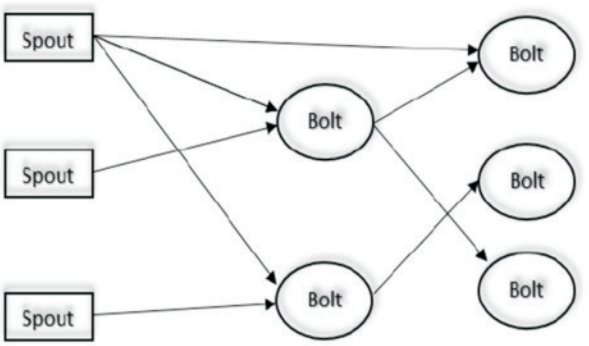


图1 Storm拓扑图示例图

Storm将应用的处理逻辑抽象成一个Storm拓扑图的方式来处理实时数据。如图1所示，一个Storm拓扑图包含如干个Spout和Bolt组件。每一个Spout组件通过从源头不断地读取数据，并将数据分发给不同的Bolt进行进一步的处理，每一个Blot组件将读取的数据执行用户定义的功能函数并将结果传给下流节点。每个Spout和Blot组件可以通过多个任务（task）并行执行，从而提高系统的并行化程度。

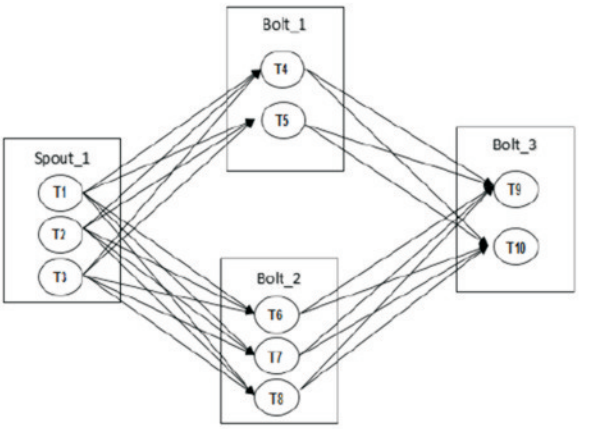


图2 Storm拓扑图内部通信

图2是Spout和Blot中task内部通信的一个示例图。Storm的资源调度就是要将所有的任务调度到不同的计算节点上使系统具有很高的吞吐量。Storm采用轮询调度策略将任务依次分发到不同的节点，如图3所示，任务T1, T2, … , T10按编号从小到大依次被分发到计算节点Machine

1, 2, 3。这种调度策略很容易导致计算节点的过度使用（over-utilized）和轻度使用（under-utilized），过度使用和轻度使用都会对系统执行性能产生很大的影响。例如，过度使用内存资源会给系统带来不能恢复的失效。轻度使用则会给系统带来不必要的支出。因此，这种简单的调度往往会忽视资源的需求和可用性，导致系统资源利用率低，吞吐性能差。

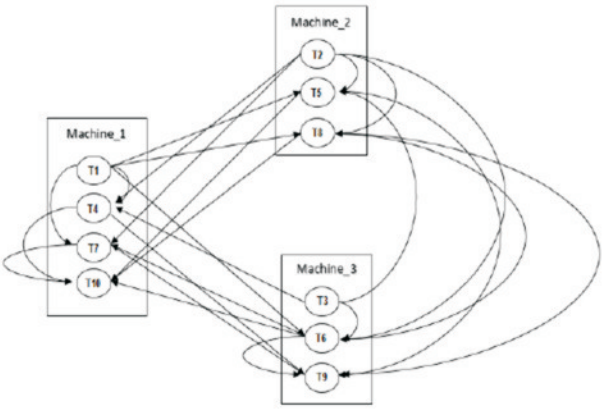


图3 Storm轮询调度示例图

$$\text{Maximize}_{\{\mathcal{N}'\subseteq\mathcal{N}\}} \sum_{\theta\in\mathcal{N}'} Q_{\theta_i} \quad \text{subject to}$$

$$\sum_{\tau_i\in\mathcal{N}'} c_{\tau_i} \le W_1,\ \sum_{\tau_i\in\mathcal{N}'} b_{\tau_i} \le W_2,\ \sum_{\tau_i\in\mathcal{N}'} m_{\tau_i} \le W_3.$$

针对上述问题，本文考虑资源的需求和可用性，主要对CPU使用，内存使用和带宽使用引入到资源调度中，将资源调度定义成如下一个二次多3维背包问题。这个问题就是要求在不超过给定CPU、内存和带宽资源的情况下使系统吞吐量最大。这个问题是一个求解复杂度极高的难题，本文作者提出了一个近似算法。这个近似算法的主要思想就是尽量将Strom拓扑图中相邻节点的任务放置在同一计算节点来降低任务之间的通信开销。具体来说，这个近似算法首先将Storm拓扑图中的节点按宽度优先算法得到一个节点的偏序关系使得相邻节点在偏序关系中尽量相邻，然后按这个偏序关系将拓扑图中的节点中的任务进一步排序，最后将排好序的任务依次按任务节点相邻性放置当前任务到一个最适合的计算节点，这里的任务节点相邻性用一个任务和一个节点在CPU、内存和带宽三者的加权距离衡量。

在实验性能分析中，本文作者将新的资源调度策略部署到Storm，并在一系列基准测试集和雅虎的实际应用中进行测试。实验结果表明在基准测试集上，本文的资源调度策略比Storm默认的资源调度策略提高了30-47%的吞吐量和69-350%的CPU利用率。在雅虎应用中，本文的资源调度策略比Storm默认的资源调度策略提高了将近50%的整体吞吐量。

本文的主要贡献在于利用将Storm拓扑图中相邻节点任务放置于同一计算节点来降低通信开销，并结合CPU、内存和带宽资源的需求和可用性，提出了一种资源感知的调度策略。

林昌富
2015级博士生
研究方向：分布式流处理
Email：lcf@hust.edu.cn

HOTL: a Higher Order Theory of Locality

池 也 推荐

“HOTL: a Higher Order Theory of Locality”被体系结构领域的国际顶级会议之一的ASPLOS 2013录用，该会议于2013年3月在美国德克萨斯州休斯顿市(Houston, Texas)举行。

内存和缓存的分配一直是计算机的一个重要问题。当多个程序在同时执行的时候，给不同程序的内存cache分配往往是静态的，这就可能导致分配给某个程序的内存cache过多或者不足的情况，导致内存浪费影响程序执行的性能。本文提出了一个新的理论HOTL以及一系列的公式，使得程序运行中个5个局部性参数可以相互推导，能够通过简单的方式预测出程序在不同cache大小下的丢失率，以便于针对不同的程序分配不同大小的cache。

Cache的局部性参数有很多，本文中选区其中最重要也最具代表性的5个，分别是footprint, inter-miss time, volume fill time, miss ratio和reuse distance。footprint是给定窗口大小下的平均数据访问次数；inter-miss time是给定cache大小下两次cache丢失之间的时间间隔；volume fill time是程序数据访问达到一定数量的平均时间；miss ratio是cache未命中比例；reuse distance是同一数据被2次访问之间的数据访问次数。其中，最主要的就是footprint和miss ratio。本文通过转换公式，由便于测量出的footprint推导出预测的miss ratio，通过miss ratio来判定需要给程序分配多大的cache。

首先，文中说明了footprint的含义公式如下：

$$fp(l) = \frac{\sum fp_w}{n - l + 1}$$

l表示窗口大小，w表示所有的窗口，n表示数据痕迹长度，所以平均footprint就是所有窗口的footprint的总和除以窗口的总数$n-l+1$。例如一个数据访问为“abbb”，拥有3个长度为2的窗口：“ab”，“bb”，“bb”。它们三个的footprint分别为2，1，1；因此平均footprint为（2+1+1）/3=4/3。

然后文中给出了volume fill time和footprint之间的关系如下：

$$vt(c) = \begin{cases} fp^{-1}(c) & if\ 0 \le c \le m \\ \infty & if\ c > m \end{cases}$$

其中m是程序访问的不同的数据量，c是cache大小，$vt(c)$表示填充满c大小的cache所需时间。如果$c>m$，说明cache大小比程序访问的不同的数据量还多，那就可以将所有的访问的不同数据全部存储在cache中，因此cache填充满的时间是无穷大。

在全相联LRU置换策略下，一段程序运行$vt(c)$时间，程序访问c个数据，正好填充完一个

大小为c的cache。如果继续运行到$vt(c+1)$，当新的数据访问进来的时候，就会产生cache丢失情况，因此平均丢失时间inter-miss time与volume fill time的关系也就十分清楚了。

$$\mathrm{im}(c)=\begin{cases} vt(c+1)-vt(c) & if\ 0\le c\le m \\ \dfrac{n}{m} & if\ c\ge m \end{cases}$$

这里的n表示是程序访问的总数据量。平均inter-miss time和miss ratio之间是倒数关系。miss ratio和footprint之间的计算公式如下：

$$mr(c)=mr\big(fp(x)\big)=\frac{fp(x+\Delta x)-fp(x)}{\Delta x}$$

假设$x=vt(c)$，所以 $fp(x)=fp\big(vt(c)\big)=fp(fp^{-1}(c))=c$，也就是说刚好 $vt(c)$ 时间将一个大小为c的cache填充完，那么再有新的数据访问时候就会出现 cache 丢失的情况发生。

volume fill time，footprint，inter-miss time和miss ratio之间关系如下图图1所示。

根据一系列的转换公式，可以比较简单的预测程序的miss ratio 从而为程序分配一个相对合理的cache大小，即能满足程序需要，不会产生性能上的降低，又能节约cache 的数量。

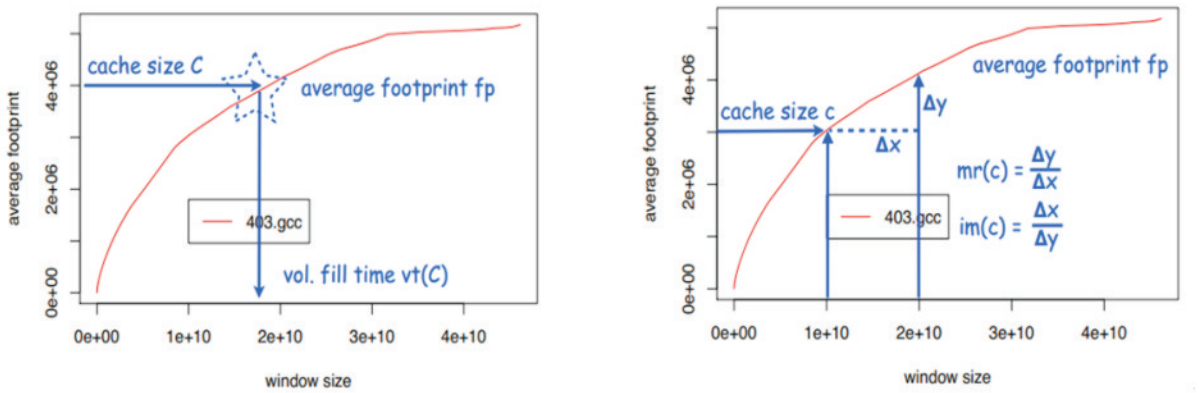


图1 volume fill time，footprint，inter-miss time和miss ratio关系转换图

最后，本文根据所提出的理论和方法，在SPEC 2006和PARSEC v2.1 suite中共挑选出了37个基准数据集进行了模拟测试。在模拟测试中所有的并行代码都是共享同一单级的cache。在测试中，HOTL针对于3000种不同大小的cache，预测miss ratio比只有单个大小的cache速度提升39%。达到相同的预测率，局部采样预测方式只需要原程序0.9%的执行时间同时也只会增加不到0.5%的额外时间开销。不仅如此，在用于预测多个程序同时执行时候所引起的cache冲突情况时候，本文提出的新机制只会用到相对于穷举测试的0.5%时间同时达到99.5%的预测准确率。

池　也
2016级博士研究生
研究方向：程序性能分析
Email：450246195@qq.com

立大志、入主流、上大舞台、成大事业
——CGCL学习感悟

周 知

2011年10月，当时还是大四本科生的我保研到CGCL实验室，提前开始体验科研生活。虽然当时与实验室以及导师结缘的过程无比偶然和梦幻，但如今让我更加难忘和眷恋的则是在CGCL实验室六年学习的点点滴滴。这六年，是我最为快乐、成长最快、收获最多的人生阶段，作为对这六年的一个阶段性总结，以下是我的几点感悟与体会。

首先，第一篇论文投稿时高标准要求自己。虽然与实验室以及导师刘方明老师结缘的过程很梦幻，但此后初入研究的路途却是充满挑战的。尽管当时我的身份还是一位本科生，刘老师却在我身上倾注了大量的时间和精力。正是由于刘老师的精心指导，我相对迅速地进入了研究的角色。然而真正提升研究能力的，则是2012年7月和刘老师在东五楼220室近10天备战INFOCOM投稿的高强度配合。刘老师带领着我在实验室一遍又一遍地润色写作、梳理逻辑、丰富内容、强化实验，以及与合作者邮件沟通。对于此前从来没有如此高强度工作过并且注意力不集中、喜欢开小差的我而言，那一周是我最为“痛苦”的一周，长时间的“烧脑”使我随时随地都想躺下来小憩一下，经常在最为疲乏的时刻，我还需要一遍又一遍地将没想明白的问题想明白，将没有写清楚的语句写清楚。做研究永远都是充满困难与挑战的，就在投稿前的第三天，我们发现故事有一个致命的漏洞，在冥思苦想了一天却仍无头绪的情况下，此前从没遇到过这种挫折的我甚至都有了放弃的打算。然而，在刘老师不断的引导与启发下，以及我们连续的思维火花碰撞之后，漏洞终于在最后关头被成功规避，论文顺利投出。连续10天的高强度工作，除了留下了9页纸的论文和80余封沟通邮件，更重要的是给我带来了极大的成长，使我具备了专注的做事态度、严谨的逻辑思维、清晰的表达能力和锲而不舍的执行力。

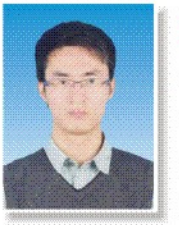
其次，积极参加国际交流活动。研究生学习期间，我先后5次前往欧美以及香港等地区参加国际学术会议或者访问学习。虽然做研究是一件相对乏味和辛苦的事情，但参加国际学术交流活动却是无比快乐有趣的。首先，国际学术会议的举办地都是国际大都会或是历史名城，一次学术会议附带一次海外旅行，有益于拓宽我们的视野，提升科技与人文素养。其次，通过积极地与高水平的教授或者参会的博士生交流，我们不仅能获取前沿的研究资讯、高效的研究方法和珍贵的科研经验，而且还能加深对他人以及自己工作的理解，锻炼自己的沟通交流能力。很有可能，你的下一个研究点或是合作者就来自你参加的上一次国际学术会议。最后，参加国际交流活动能够极大地增强我们的自信心。16年我在德国访学的时候遇到了国内最好的计算机系的一位教授，我告诉他实验室全额资助我们两百多名研究生参加CNCC，他听了后非常震惊，说了一句“金老师那边家大业大，我们不能比”。同样在德国访学期间，我还与来自清华、复旦、南大等学校的同学们交流过我们实验室奖励高层次论文的政策，他们听说后同样也是惊呆了，无比羡

慕。后来有一天我的导师刘方明老师去复旦那位同学所在实验室作报告，那位同学收到报告通知后对我说，让刘老师在他们那边多讲讲我们这边土豪般的论文奖励政策。此外，15年在香港参加INFOCOM的时候，有次向几名香港和北美的博士生做完自我介绍后，其中一位对我说，你们小组做得很好，我知道你们组有一位很厉害的博士生，叫郭鉴。看来，不仅我们CGCL的老师们享有盛誉，就连学生也是声名在外。如果大家都能体会到这样强大的自信心，那么科研上就不会有太大的困难了。

此外，勇于走出自己的舒适区并追求更高的目标。在研究生的前两年半，得益于实验室的大力支持与导师刘方明老师的精心指导，我发表了2篇INFOCOM，1篇TPDS和1篇ICDCS。此后导师觉得我已经有了比较好的积累，就鼓励我向更高的目标发起冲击。由于我的研究兴趣主要在于系统建模与性能评估，因此刘老师一直鼓励我全力冲击这一领域最富盛誉、难度最高的旗舰会议ACM SIGMETRICS。然而，虽然我此后向该会议发起了两次冲击，但遗憾的是一次铩羽而归，一次只中了poster。我曾想，如果我没有听从导师的建议，而是继续停留在自己的舒适区发自己比较熟悉的会议，那么我现在应该文章更多津贴更高吧？然而我发现，暂时的失败并没有挫败我的热情，反而让我愈战愈勇，研究品味越来越高。而且，每当和别人讲起自己的研究时，往往令我更自豪的，不是我中过什么会议，而是我投过什么会议。或许，就像那句广告那样，“所谓的光辉岁月，并不是波澜闪耀的日子，而是无人问津时，你对梦想的偏执”。

最后，做“顶天立地”的研究，做“五能”研究生。所谓顶天，是指研究要天马行空，有创新性、学术价值高。而所谓立地，是指研究要接地气，面向实际问题，面向祖国与人民需求。所谓“五能”，即“能做、能写、能讲、能用、能卖”是指研究生培养的五个层次，分别为：研究能力强、写作能力强、演讲能力强、做的研究有人用或是跟进、做的研究成果能被卖出去。越到后面，研究生能力越强，研究成果影响力越大。随着实验室科研经费的飞速增长和软硬件条件的不断改善，近年来我们小组在导师刘方明老师的带领下先后做了几项“顶天立地”的研究，例如在年会上系统演示环节被金老师鼓励可以创业的融合网盘系统、在安卓市场了吸引了4万下载量的节能微博客户端以及吸引了华为技术高层注意力的数据中心能耗监测系统。对于我自己而言，我前几年的工作都是基于系统建模而缺少实现，因此足够“顶天”而不够“立地”。后来导师鼓励我将自己的研究做得更加接地气，结合实验室优渥的硬件条件，通过真实系统测量或是原型系统实现来激发理论研究的必要性或是验证理论成果的有效性。目前，我也照着这个方向在转换自己的研究风格，转换风格后的第一篇论文虽然到现在还没有被接收，但我一直认为这是我最好的一篇文章。

以上就是我读研以来的一点点感悟，希望能对大家起到一点点作用。六年来，在实验室文化的熏陶下，在老师们的指导教育下，我在各个方面都感受到了飞速的成长。实验室见证了我的成才，我也见证了实验室六年来的飞速发展，国家奖、重大项目、人才计划、高层次论文……每一项沉甸甸的成果都让我们无比激动。作为一名即将毕业的博士生，我将和所有的CGCL学子们一起牢记老师们的教诲，努力践行“立大志、入主流、上大舞台、成大事业”。



周　知

2014级博士毕业生

研究方向：绿色云计算、大数据处理

Email：zhiz@hust.edu.cn