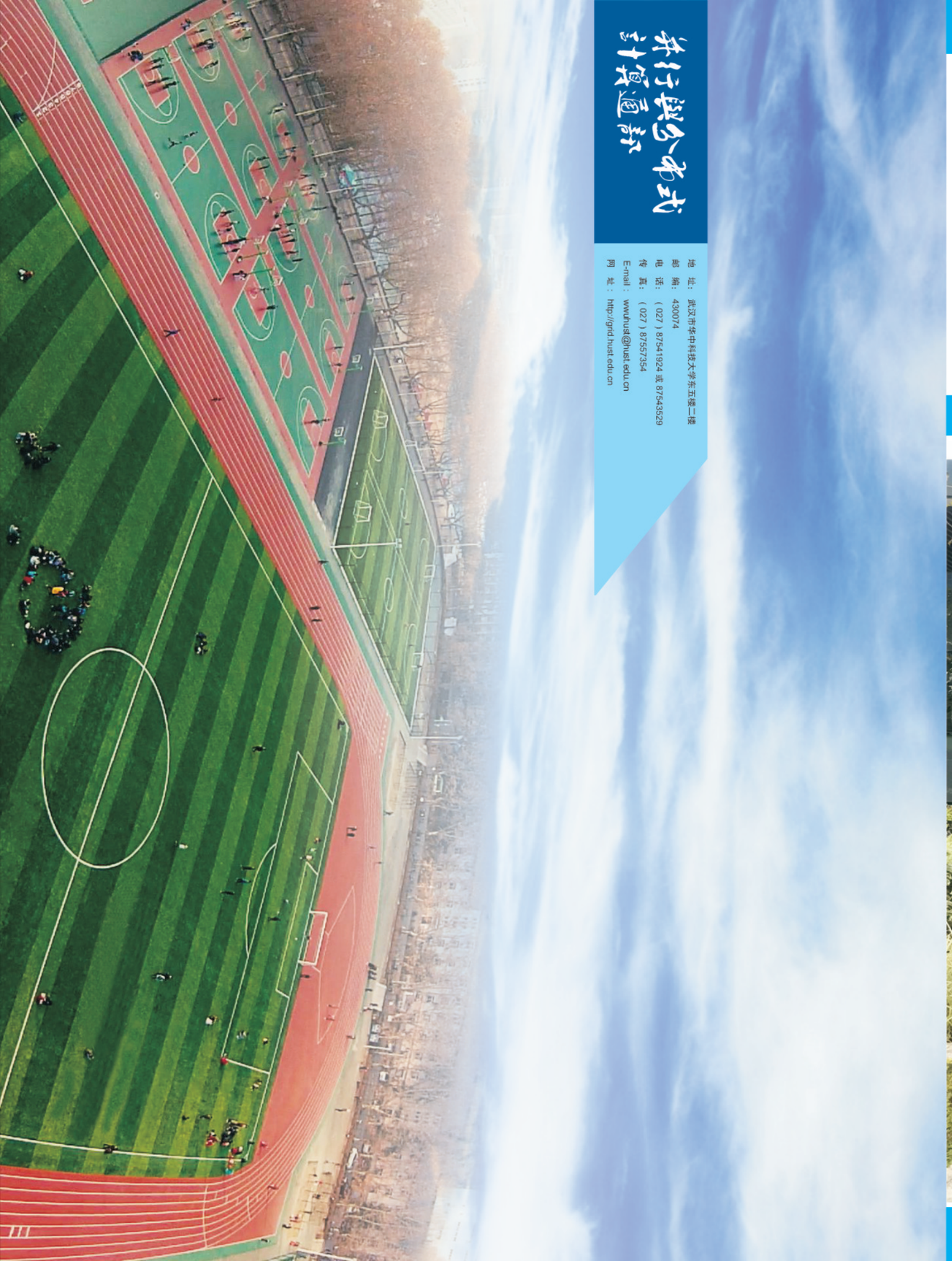


并行與分布式計算通訊

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

SCTS 服务计算技术与系统教育部重点实验室
CGCL 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

2018年第1期　总第32期　2018年03月

封面人物：黎明——积极主动，全情投入

2018年度实验室各研究方向规划

智能运维，云数据中心运维的未来之路

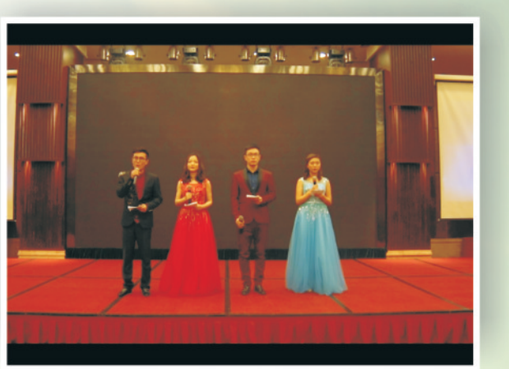
科研的最好时代

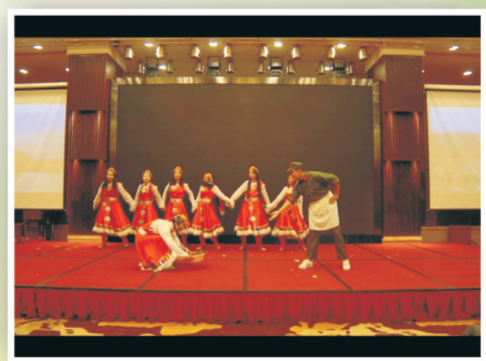
http://grid.hust.edu.cn

实验室第九次学术委员会

2018年实验室新春联欢会

风采





新时代、新挑战、新突破
—2018年度实验室科研方向年度规划

送走硕果累累的2017年，在充满变革意义的戊戌2018年，在习近平总书记提出的四个自信引领的新时代下，实验室面临着如何“把科研论文写在祖国大地上”的新挑战。为迎接这一挑战，实验室年终务虚会议以“新时代、新挑战、新突破”为主题，规划实验室未来五年、十年甚至十五年的发展思路，力争在新时代背景下，迎接新挑战，实现新的科研突破，使实验室的发展迈上更高的台阶。本期季刊作为2018年的开篇之作，将向读者展示实验室体系结构与系统软件、云计算与移动计算、网络空间安全、大数据四个科研课题组为落实务虚会议精神所制订的2018年发展规划。

体系结构与系统软件方向2018年将针对大数据应用给异构计算环境以及云计算需求对异构内存管理带来的挑战，继续延续过去几年制定的研究路线，在面向大数据处理的数据流体系结构、面向虚拟化场景的异构内存系统两个主要方面开展研究，并特别面向图计算需求，开展异构平台上的图计算系统的研究。在面向复杂大数据处理需求的数据流体系结构设计方面，将开展硬件层面的灵活定制、执行层面的高效调度和通信层面的动态划分三个关键问题的研究，提出数据流体系结构的设计规范和设计方法。在面向云计算场景的异构内存资源管理方面，将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特定应用的轻量级虚拟机定制和裁剪机制、基于RDMA的异构内存池化机制等研究工作。在面向图计算的异构平台及系统方面，将开展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等方向的研究。

云计算与移动计算方向2018年将开展面向物联网和国产平台的轻量级虚拟化技术、高效能云计算数据中心资源调度、边缘计算、城市计算、面向移动云服务环境的终端节能、区块链、NFV性能加速等方面的研究。在面向物联网环境的轻量级虚拟化技术方面，将重点研究特型内核构建和应用封装方法、基于特型内核的基础运行环境、分布式超轻量级虚拟化镜像管理机制等关键技术。在面向国产平台的轻量级虚拟化技术方面，将研发面向国产平台的轻量级虚拟机和容器构建与运行管理平台。在高效能云计算数据中心资源调度方面，将研究多维异构云资源的细粒度划分方法及细粒度资源感知和多数据中心调度等关键技术。在边缘计算方面，将基于提出的“终端-边缘-云”协同计算架构，研究相关的基础理论与关键技术。在城市计算方面，将研究交互数据感知、融合和分析技术。在面向移动云服务环境的终端节能方面，将针对移动终端在利用云服务平台性能上存在的瓶颈问题设计节能机制。在区块链方面，将从应用层、共识层、网络层和数据层四个维度研究同构/异构区块链的跨链交互关键技术。在NFV性能加速方面，将研究通过GPU实现对NFV加速的相关难点问题。

网络空间安全方向在2018年将重点围绕云计算安全相关的一系列关键问题开展研究，包括代码漏洞检测、软件定义网络安全与主动防御、区块链安全、云环境隐私追踪取证等。在代码漏洞检测方面，将开展基于深度学习的智能化漏洞检测研究。在软件定义网络安全与主动防御方面，将研究基于软件定义网络、网络功能虚拟化与移动目标防御等技术的主动防御机制。在区块链安全方面，将对数据交易安全、可靠的SPV轻钱包、激励价值创造的POV共识、区块链版本控制等问题展开研究。在云环境隐私追踪取证方面，将集中于研究虚拟计算环境中隐私数据的访问监控、跨虚拟机的隐私流转追踪、虚拟计算环境中隐私计算分析与取证等问题。

大数据方向在2018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，将基于新型体系结构对大图计算问题设计低通信复杂度分布式算法，并证明相应的通信复杂度下界。在大数据处理支撑技术方面，将主要研究面向数据密集型应用的异构内存管理优化、HPC环境下基于软件定义存储的IO调度优化、数据流高效处理、大规模图计算与图挖掘、大规模知识管理与搜索以及抗隐私分析的大数据查询处理优化等问题。在大数据技术应用方面，将基于三角形结构研究符号社交网络中的动态演化机理，对网络的未来状态进行预测，并研究其在实际网络中的应用。最后，大数据方向将继续在医疗、通讯和航空等领域提出基于数据挖掘和数据分析的行业应用解决方案。

预则立，不预则废。科学的科研规划是取得新突破的先期条件。本期介绍的科研规划将为实验室未来一年的发展指明方向，也为未来3-5年的发展夯实基础。祝愿实验室2018年延续辉煌，再创科研佳绩！

于东晓

2018年2月28日

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

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

卷首语

热点

封面人物

专栏

声音

动态

推荐

交流

IBM发布基于内存的人工智能计算架构

（叶源远 整理）

IBM研究人员实现了在内存计算技术上的一次重大突破。他们宣布发明了一种可以运行在100万个相变内存（Phase Change Memory，PCM设备）上的无监督式机器学习算法，并且成功地在一系列未知数据流中发现了时间相关性。与目前最先进的传统计算机相比，这种内存计算原型技术有望在计算速度和能耗利用效率方面提升200倍，非常适合实现人工智能（AI）应用中的高密度、低功耗、大规模的并行计算系统。如果说制造“人工大脑”需要分三步完成，那么甚至可以说，IBM已经来到了第二步。

IBM此次设计的内存计算架构中使用的相变内存设备由两个电极包夹着一层锗锑碲复合材料构成，微弱的电流可以使其加热，复合材料内部状态随着温度上升而发生改变，从无定形态变成晶态，利用了结晶动力学原理进行运算。这与IBM公司2016年8月发布的人造神经元的工作原理相同。

内存计算（In Memory Computing, IMC）或者说可计算储存，是近年来新兴的一个概念，其原理是运用内存设备的物理特性进行资料的储存和处理。不过 IBM 把这点做到更彻底，那就是不采用过去那种内建缓存，并通过总线连接外部内存的阶层式架构，而是省略总线设计，直接把内存和 CPU 核心做在一起，尽最大的可能消除计算过程中因数据迁移所造成的延迟，显著提升计算效率。

在另一方面，相变内存本身就兼具 DRAM 类内存的速度，以及 NAND 的非易失性存储特性，通过将相变内存直接和 CPU 集成，形成一种特殊的计算硬件结构，计算和存储同时可在相同的结构上发生，不需要额外的传输、读取再写入的动作。因此从计算到存储所需要的指令操作、带宽消耗等现象都得以减少。对 AI 计算这种数据迁移量庞大的应用来说，省去了这些操作，自然整体计算效率就可以大大提升，且因为没有总线，也没有额外的读写操作，系统功耗也能大大减少。

如果所有的计算、读写工作都发生在同一个区块上，不用迁移，自然也不会有延迟。这种概念完全不同于目前主流电子系统和设备的运行原理，我们所熟知的桌面计算机、笔记本计算机和手机等等电子设备都采用了冯·诺伊曼结构。使用该构架的设备在运行时，各种数据会在内存和计算单元之间不断穿梭，大量的数据调用会降低计算速度并且增加能量消耗。

这种本身可以计算同时亦可存储的架构看起来好像似曾相识？没错！其实就是一个具体而微的电子大脑结构。虽然这次所使用的计算核心还是一般为 AI 计算优化的 CPU 架构，但 IBM 曾在 2016 年 8 月发布了轰动一时的人造神经元，从这个脉络看来，IBM 的确是想打造一个完完全全的“电子大脑”。

未来，IBM可能继续将人造神经元与内存计算架构相结合，像大脑一样，使逻辑的产生以及信息的存储都在同一处发生，非常适合于人工智能计算。这一成果也是IBM创造“人工大脑”的又一最新进展。

INTEL 推出 XEON D-2100 处理器系列：加快边缘网络及云端运算速度

（蔡晶晶 整理）

数据显示，到2020年，将有超过500亿个物和设备接入网络。这种物与物的连接与通信，将会带来数据洪流。面对这样的吞吐量，移动边缘计算可以在无线网络端增加计算、存储、处理等功能，将传统的无线基站升级为智能化基站，为“物物对话”成为现实提供帮助。英特尔在2018年2月推出了全新Intel Xeon D-2100处理器，该处理器可以确保服务提供商和企业能够在网络边缘或Web层提供最大化的计算智能，并将功耗降至最低。

Intel Xeon D处理器都是采用14nm制程技术，并在核心的互连方式上采用 Scalable Xeon 所用的Mesh架构，取代以前的Ring结构，令存取数据更快速。它还支持Intel AVX-512指令集以及最多1FMA，来加快浮点运算、视频编码、图像识别等功能的速度。在硬件方面，Xeon D-2100 系列最多具备18核心CPU、36线程、512GB四通道DDR4-2666 ECC RAM，单核心加速频率最高可达3.0GHz，最多支持32条PCIe 3.0通道以及4个10Gbps网络端口，平均各型号的TDP均在60–110W之间。

Intel Xeon D-2100处理器在存储方面可用于针对密度优化、轻量级的超大规模云工作负载任务，例如动态Web服务、内存缓存、专用主机和温存储等。在内容分发网络（CDNs）方面，将Xeon D CPU用于CDN的边缘网络，可以为网络边缘的内容分发提供更高的性能，这对于确保视频直播以及处理大规模媒体文件时实现低延迟至关重要。在企业网络方面，该处理器系列还适用于入门级企业SAN和NAS存储、中端路由器、网络设备、安全设备、无线基站和中端嵌入式物联网等应用，亦适用于边缘网络设备。

Intel Xeon D-2100处理器将为网络边缘带来更强的性能和硬件增强的安全性，让计算、分析和数据保护能力更接近于终端设备，以满足日益增长的工作负载需求。随着科技发展一日千里，以及5G流动网络、IoT物联网技术渐趋成熟，网络设备（尤其是边缘网络）的工作负荷就愈来愈重。在今年的平昌冬奥会上，Intel和韩国运营商KT部署了迄今为止规模最大的5G网络，在10个奥运场馆搭建22个5G链路，支持IPTV、虚拟现实、Wi-Fi等应用，还为观众和游客提供千兆级速度服务。作为5G的关键技术Multi-Access Edge Computing（MEC）多接取边缘运算即是把Facebook、Google Map等大型数据库服务置于网络边缘，缩短连接数据中心的路程，让IoT、手机等流动装置更快实现数据存取，但同时也增加了网络边缘设备的负荷。将Intel Xeon D-2100系列用于这些边缘设备，可以实现快速运算、分析大量数据，且最多能缩短29%的MEC指令周期，以达到在有限的空间和功率下最大化计算智能的目的。

下一个可能改变世界的技术领域——量子计算

（胡 蝶 整理）

2018 年 2 月 22 日，中科院量子信息与量子科技创新研究院与阿里云宣布，在超导量子计算方向发布 11 比特的云接入超导量子计算服务，用户可上传测试运行各种量子计算线路代码，并下载相关运行结果；在半导体量子计算方向实现了基于量子点量子芯片的 Deutsch Jozsa 量子算法演示，实现了多量子比特逻辑门。这是继 IBM 之后，全球第二家提供 10 比特以上量子计算云服务的系统。

量子计算机是一种遵循量子力学规律，进行高速运算、存储及处理量子信息的物理装置，不同于传统的电子计算机的运算单元一个比特在特定时刻只有特定的状态， 0 或者 1 ，量子计算机能够利用量子特有的“叠加态”，采用并行计算的方式，一旦实现，意味着计算速度会有数十亿倍的提高。这一计算能力的飞跃，将远远超越从算盘到当代超级计算机的提升。以今天的标准来看，量子计算可以达到的处理速度令人难以想象。假设人们把这种量子计算机运用到监控领域，它可以瞬间在数据库中扫描 60 亿地球人的脸，并实时辨别出一个人的身份。

现在，全球已将量子计算研究作为争夺“人类终极计算能力”的关键入口。仅 2017 年，量子计算能力的竞争在谷歌和 IBM 几大科技公司间渐趋白日化。

阿里巴巴是首个大力投入量子信息科学的中国科技公司，致力于将量子计算从学术带到现实世界。早在 2015 年 7 月，阿里云与中国科学院共同成立亚洲首个量子计算实验室，在量子信息科学领域开展前瞻性研究。阿里云随

后带来全球首个云上量子加密通讯案例。

2017 年 5 月，由中科大、中科院 – 阿里巴巴量子计算实验室、浙江大学共同研制完成的世界首台光量子计算机诞生。从实验室成立到量子计算云平台上线仅用了两年多时间，可见量子计算的发展进程比我们想象的要快的多。 2017 年 9 月，世界知名量子计算科学家施尧耘加入阿里云，担任首席量子技术科学家； 2017 年 10 月，阿里云联合中科院两字创新研究院发布量子计算云平台，推动量子计算产业化。云平台地址为： http://quantumcomputer.ac.cn/index.html 。通过量子计算云平台，用户可以在云端的超导量子处理器上运行自定义的各种量子线路代码，下载相关运行结果。该平台吸引更多人在上运行量子算法，完成初步试验，不仅能用于了解处理器的性能、技术瓶颈等重要特性，还将帮助到下一代处理器开发，为优化应用积累经验。而在云端提供量子计算的创新服务方式，也能从中知悉面临的技术挑战和机会。 2018 年 1 月，两次理论计算机最高奖哥德尔奖得主马里奥·塞格德先后加盟阿里云量子实验室；直到前几天，量子计算云平台成为全球第二家开放 10 比特以上量子计算云服务的系统，上线国内首个超导量子处理器，实现了量子计算领域一个巨大的跨越。

阿里云首席量子技术科学家施尧耘预测，2018 年将上演量子计算年度大戏：几个大公司之间将出现量子霸权混战；量子计算将进入超导和离子阱的两级时代；量子密码和经典密码的竞赛也在白热化，未来量子计算面临着诸多变局。

南安普顿大学安装了20,000个核心超级计算机

（段卓辉 整理）

Iridis 5进入Top500超级计算机榜单第251位

英国南安普顿大学已经开启了其最新的超级计算机Idris 5，其最高性能为1.3 Petaflops。

该20,000核心机器是由高性能计算（HPC）集成商OCF使用Lenovo的Think System SD530服务器设计和制造的。

它已经成为世界上最强大的系统之一，在11月份进入500强的超级计算机排行榜，排名第251位。

“南安普敦大学在运用计算技术产生新知识和洞察力方面有着悠久的传统，可以追溯到1959年，当时我们的研究人员首次在悉尼歌剧院的设计中使用了建模技术，” Oz Parchment，iSolutions学院总监说到。

“使用计算方法对数据进行数据分析是现代科学技术的核心，为了吸引最优秀的世界级研究人员，我们需要世界一流的研究机构。”

南安普敦大学表示，在过去十年里，使用HPC服务的研究项目数量增加了425%。

Iridis的最新版本比其前一版本强四倍，并且有助于加速分析，生物信息学和人工智能领域的研究。它还将支持大学的沃尔夫森单位，该单位侧重于船舶设计（南安普敦是英国最重要的港口城市之一）。

“我们拥有遍布全球的客户群，并与过去参加过三届奥运会的英国自行车队一起工作，并与还参与美洲杯帆船比赛的团队合作，” 南安普敦大学沃尔夫森单位的首席研究工程师Sandy Wright评论道。

“在过去的10年中，计算流体动力学（CFD）已经成为一种重要的商业活动，而不是仅仅对物理实验的需求。 CFD提供了与风洞一样好的解决方案，无需建立模型，因此您可以加快研究速度，同时降低成本。 Iridis 5将使欧胜单位获得更准确的结果，同时能查看更多参数并提出更多关于计算模型的问题。”

Idris 5包含20,000多个Intel Skylake内核，以及40个Nvidia GTX 1080 Ti GPU。使用Lenovo ThinkSystem Sd530服务器提供的计算，而GPU则位于技嘉的服务器中。 OCF策略可使在系统运行时可直接添加Nvidia Volta GPU。

Idris 5还具有联想DSS Spectrum Scale Appliance提供的2 PB存储容量，以及IBM提供的另外5 PB容量的存储容量，并支持最新的15 Tb盒式磁带。

在软件方面，主要的HPC资源将使用OCF的基于xCAT的工具进行管理，Bright Computing的高级Linux集群管理选择提供系统的研究云和数据分析部分。

Griffin：异构处理器在信息检索系统中的协同查询

（康 祥 整理）

面对快速增长的数据集和查询负载，交互式信息检索服务（如企业搜索和文档搜索）必须提供准确，低响应时间的查询结果集。这些不断增长的需求使得研究人员不断进行优化以减少响应延迟，包括查询处理的并行化以及GPU 等协处理器的加速。传统的优化方案是在GPU 或CPU上单独运行某一查询，而忽略了这样一个事实，即给定查询的最佳处理器的选择取决于查询自身的特性，这一特性可能会随着查询处理过程的进行而发生变化。

Griffin根据每一个查询的特性，通过动态组合基于GPU和CPU的并行算法进行处理。Griffin使用先进的基于CPU的查询处理技术，并结合了一种新的基于GPU的查询评估方法。通过利用新的压缩方案并利用先进的基于合并的交叉算法来实现最佳的GPU搜索性能。

信息检索（IR）服务是快速增长数据集的关键，并确保较低的响应延迟。为了提供可伸缩性，当前的IR系统通过在大型计算集群中分发查询来实现大规模、粗粒度的并行。为了达到低延迟要求，他们依靠巧妙的，高度优化的算法，利用单个节点上的内部并行。

传统的方式通过增加CPU线程来处理每个查询，探索查询内部的并行性，现有的研究利用GPU的并行，能够取得比CPU较好的加速效果。由于查询具有两种不同的特性，所以有些查询在GPU上运行得更好，而其他查询在CPU上运行得更好，因此，异构系统通过在适当的处理器上运行不同的查询，能够取得更好的总体性能。然而，查询的特性也会随着查询的执行而改变。虽然查询执行的早期阶段可能在GPU上运行良好，但后期阶段对CPU来说往往更适合，因为随着查询的进行，所需的计算量将逐渐减少。因此，只在CPU或GPU上静态地运行整个查询很有可能达不到最佳性能。这意味着在处理某一查询的时候，通过划分操作动态比给合适的处理器分配相适应的任务将带来更好的效果。

Griffin结合两个创新来提供更好的性能。首先，Griffin联合使用CPU和GPU来处理查询，并随着查询特性的改变，将查询执行从GPU 迁移到CPU。Griffin在没有事先了解查询特性的情况下决定何时何地执行查询操作。为了做出正确的决定，Griffin考虑由于CPU和GPU之间的数据传输，GPU内存管理以及系统负载而产生的开销。Griffin使用动态查询内部调度算法解决了这些挑战，该算法将查询分解为子操作，并根据其运行时特性将它们安排到GPU或CPU中。Griffn使用这种调度算法在基于CPU的最新搜索实现和基于GPU的新搜索内核之间划分工作，该内核称为 Griffin-GPU。

Griffin-GPU是 Griffin的第二个关键创新。GriffinGPU结合了两个组件。 首先是并行的Elias-Fano解压算法，该算法提供了快速解压缩和高压缩比。第二个是基于负载均衡的并行合并列表交集。

Griffin与分别运行高度优化的基于CPU的查询处理系统和 Griffin-GPU相比，Griffin分别将查询处理速度提高了10倍和1.5倍。同时减少了尾部延迟。

大数据在运营商、图计算以及云的应用

（寇志娟 整理）

2017年5月，中国云计算技术大会（CCTC）在京召开。本次会议践行“云先行，智未来”的主题，策划了微服务、人工智能、云核心三大论坛及Spark、Container、区块链、大数据四大技术峰会，众多技术社区骨干、典型行业案例代表齐聚京师，解读本年度国内外云计算技术发展最新趋势，深度剖析云计算与大数据核心技术和架构，聚焦云计算技术在金融、电商、制造、能源等垂直领域的深度实践和应用。

大数据在运营商中的应用

运营商原有大数据平台分为Iaas/Paas/Saas三层，未来运营商大数据平台将往更深层次方向演进，包括应用域、数据域、技术域、基础域、安全域和开放域。经过运营商多年的建设，当前技术主要基于“开源+自主”研发结合，利用大数据核心技术，构建面向业务应用和平台应用的实践。自主大数据核心技术包括Xcloud和Hadoop BEH。XCloud是面向分析型应用领域，基于SQL on Hadoop，结合行列混合存储技术、大规模并行化计算技术、组合数据压缩算法及智能索引等技术构建的新型分布式数据库。东方国信的Hadoop发行版本，是基于开源版本进行增强，兼容开源版本，能随着开源版本的升级而升级。基于开源社区源代码实现二次开发，转换为自己的核心技术，逐渐将应用与生产实践的验证部分代码提交给社区，比如：K8s+Docker底层源代码修订。在运营商的企业运营管理，围绕大数据为核心，面向客户和内部员工，实现企业业务运营和管理。业务应用实践中包括数据、决策、营销、客户、绩效五个方面。

运营商未来发展将具备以下能力：①实时能力：由于数据采集和处理更加实时，对实时数据应用能力提出更高要求；②在线能力：要求数据同时具备OLAP/OLTP能力，并且保证数据一致性，对基于明细数据的在线计算能力越来越高；③学习能力：从模型和算法的分析挖掘逐渐向半机器/机器学习演进；④开放能力：要求从资源到数据逐层对用户开放，对外服务高并发要求和安全性要求越来越多；⑤数据能力：数据管理越来越资产化，逐渐演进到软件驱动数据生产或者软件定义数据生产，提供全工具化的数据生产能力；⑥应用能力：内部应用更精细化和实时化，外部应用以运营商开放数据为基础，面向行业的跨行业应用。

图计算优化技术探索

当前图计算呈现出四大特点：高访存计算比、数据局部性不好、结构不规则和受数据驱动。因此，优化数据载入的速度是重中之重。图计算系统的计算框架，包括计算框架的作用，如便于编程、性能扩展和自动容错；以顶点为中心的计算框架和以边为中心的计算框架等等。图计算由于其应用的广泛以及规模的扩展，现在仍然是热点的研究内容；通过体系结构相关的方法可以加速图计算的运行，如使用体系结构局部性加速图计算、图的三维划分加速计算和外存图计算的加速方法；图计算的不同模式，需要不同的加速方法。

云时代大数据管理引擎HAWQ++

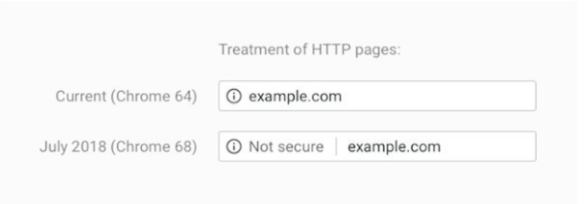
HAWQ++是由Apache HAWQ创始团队打造的HAWQ增强企业版本，采用了MPP和Hadoop结合的创新MPP++技术架构，高可扩展，遵循ANSI-SQL标准，提供PB级数据交互式查询能力，并且提供对主要BI工具的描述性分析支持。兼容Oracle，GPDB和PostgreSQL，原生支持Lava和Kubernetes平台，可以帮助企业无缝迁移到最新的云计算平台。HAWQ资源管理的目标是负责向YARN申请资源和回退资源和为HAWQ用户，查询和操作符分配资源；三级资源管理包括全局资源管理、内部资源管理和操作符级别资源管理；多级资源管理和CPU和memory的管理。在HAWQ++导入导出数据中，可通过hdfs外部表导入导出数据、gpfdist外部表导入导出数

据、COPY命令导入导出数据和hawq load 工具导入数据。HAWQ++新特性包括：①HAWQ++ 2.0.1& 2.1 (2017年Q1)：可插拔存储（文件系统，存储格式等）、容器云平台支持。②HAWQ++ 3.0：高性能执行引擎 (已有原型系统)。

终于来了！Chrome 68将所有HTTP页面标记“不安全”

（汤媛媛 整理）

2018年2月8日，谷歌安全博客发布公告：从2018年7月Chrome 68版本开始，将所有HTTP页面标记“不安全”。



谷歌强推HTTPS加密，取得惊人进展

过去几年，谷歌一直通过大力倡导网站采用HTTPS加密，推动互联网迈向更安全的网络环境。2017年谷歌逐渐加大力度，通过将更多HTTP页面标记“不安全”来帮助用户了解HTTP站点的安全风险。

• 2017年1月(Chrome56)开始，正式将某些HTTP页面标记“不安全”，比如含密码或信用卡信息传输的HTTP页面;

• 2017年10月(Chrome 62)开始，所有需要输入数据的HTTP页面以及“隐身模式”下的所有HTTP页面都将显示“不安全”警告；

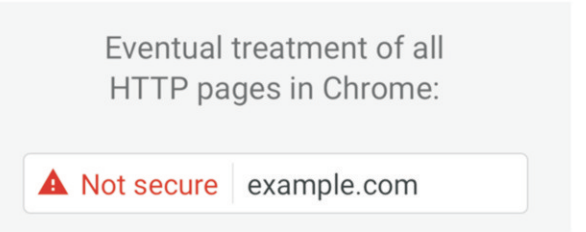
2017年的措施带来了惊人的进展，根据谷歌发布的数据显示：

· 安卓和Windows上，超过68%的Chrome流量已经受到加密保护

· Chrome OS和MAC上，超过78%的Chrome流量已经受到加密保护

· 排名前100的网站中，81个网站已经默认使用HTTPS加密

而此次发布的公告，将进一步推动全球互联网迈向全网HTTPS。目前，Chrome仍显示中性的圆圈“i”标识，未来可能进一步加强警告，最终目标是对HTTP页面显示红色三角警告。



网站没有升级HTTPS，会怎么样?

NetMarketShare公布的2017年8月浏览器市场份额数据显示，Chrome依然是最热门的桌面浏览器，市场占比接近60%。如果您的网站没有升级HTTPS加密，Chrome将标记您的网站“不安全”，警告用户谨慎访问，这可能造成大量用户流失。在Chrome浏览器的引领下，其他浏览器也在逐步实施弃用HTTP的计划，FireFox准备将所有HTTP页面标记“不安全”，Safari也计划添加HTTP安全警告，HTTP页面的警告范围将越来越大。

MONTH	CHROME	INTERNET EXPLORER	FIREFOX	MICROSOFT EDGE	SAFARI	OTHER
October, 2016	54.99%	23.13%	11.14%	5.26%	3.69%	1.79%
November, 2016	55.83%	21.66%	11.91%	5.21%	3.61%	1.78%
December, 2016	56.43%	20.84%	12.22%	5.33%	3.47%	1.70%
January, 2017	57.94%	19.71%	11.77%	5.48%	3.47%	1.64%
February, 2017	58.53%	19.17%	11.68%	5.55%	3.45%	1.62%
March, 2017	58.64%	18.95%	11.79%	5.61%	3.37%	1.65%
April, 2017	59.00%	18.40%	11.80%	5.62%	3.44%	1.75%
May, 2017	59.36%	17.55%	11.98%	5.63%	3.56%	1.91%
June, 2017	59.49%	16.84%	12.02%	5.65%	3.72%	2.29%
July, 2017	59.57%	16.50%	12.32%	5.65%	3.66%	2.29%
August, 2017	59.38%	15.58%	12.28%	5.66%	3.87%	3.23%

此外，安全访问已经成为互联网标准，现代互联网技术都将围绕着更为安全的HTTPS加密而设计的。一些新功能和标准已经被设计为从发布之初就使用HTTPS加密，例如：下一代HTTP/2协议、浏览器最新功能(地理定位、用户媒体等)等等都仅支持HTTPS加密。网站没有升级HTTPS加密，那这些高级功能都和你无缘了。

来源：https://www.wosign.com/news/chrome-68-https.htm

AI点亮MWC2018 旷视联合Qualcomm展示手机智能解决方案

（吴俊芳 整理）

当地时间2月26日，2018年世界移动通信大会(MWC 2018)在西班牙巴塞罗那盛大召开，大会吸引了来自208个国家的通信及移动行业领导者。全球手机行业AI解决方案提供商旷视科技Face++联手Qualcomm在MWC中亮相，向全球展示了终端侧AI的无限可能。

作为一年一度的通信行业顶级盛会以及全球手机及通信行业的风向标，这届MWC依旧看点满满，从旗舰新机发布到5G，再到移动物联网，MWC 2018向世界输出了业内最前沿的应用趋势和最尖端的移动科技成果及产品，随着技术的成熟以及在移动端应用场景的不断深化，AI技术成为了本届大会的重头戏。

MWC 2018上，旷视科技Face++就与Qualcomm联合展示了面向智能手机的AI应用——Animoji。得益于Qualcomm人工智能引擎AI Engine的支持，旷视的算法可以在Qualcomm骁龙移动平台上实现更快的处理速度，并在极低功耗的情况下实现复杂的场景感知计算，完成实时3D人脸建模和面部表情肌追踪等工作。

旷视科技Face++用AI重新定义手机

多年来，旷视科技Face++通过大量的商业实践，积累了丰富的行业经验，已成功在金融、安防、手机等多个行业落地了“云+端”的一体化解决方案。而当人工智能已经成为了老生常谈的话题的时候，旷视希望用AI重新定义手机。

在Qualcomm的MWC展位上，旷视Face++与Qualcomm联合展示了基于Qualcomm 人工智能引擎AI Engine和计算机视觉技术打造的应用产品——3D Animoji。

3D Animoji是旷视的3D影像技术研发上的又一力作。区别于苹果此前所展示的3D表情技术，旷视基于深度学习的3D面部重建与3D表情模拟技术，在单摄上即可实现在卡通形象上模拟超过 46种面部表情肌肉变化，不依赖3D摄像头。同时，旷视的3D Animoji在双摄、TOF、结构光设备上会表现更佳。

和优秀伙伴共创美好未来

科技创新是人类社会进步和提升生活质量的重要驱动力，也给人们的生活带来了无限的美好，而成就手机美好未来的无疑将是人工智能。此次 MWC 2018 以 Creating a Better Future（创造更美好的未来）为主题，这恰与旷视科技“为了人工智能终将创造的所有美好”的使命不谋而合。

旷视科技致力于研发先进的算法并在各个终端建立感知网络，其中最重要的一部分便是移动端计算。旷视希望打造一套适配整个手机平台的智能视觉解决方案，将深度学习的成果部署在移动端平台当中，从而使其更好地适配移动端设备，进而革新智能手机交互模式和体验、满足不同手机厂商在人脸解锁、图像增强、相机增强、智能图像和视频处理上的需求。

2017年，由旷视研究院独创的高性能卷积网络 ShuffleNet在业内引起了广泛关注，与行业中越来越大、越来越重的神经网络不同，ShuffleNet 是一种专为计算能力有限的移动设备而设计的卷积神经网络，它的计算目标在 10-150 MFLOPs， 和之前的AlexNet相比，ShuffleNet 在相同精度下实际运行速度可以提升15-20倍，可以帮助手机厂商在相同甚至更高的精度条件下保持更快的运行速度。不到半年的时间里，ShuffleNet 不仅在旷视自身的硬件研发和手机智能应用中全面投入应用，也在产业届中得到广泛认可。

此次和Qualcomm的联合展示，意味着旷视未来可以更好地帮助手机厂商在计算资源、性能、功耗、成本等多个维度中找到平衡点，满足不同厂商对于移动平台的视觉技术应用和图像增强、相机增强、智能图像、视频处理等需求，从而完成移动端的智慧赋能。

（链接：http://www.kejixun.com/article/180227/415602.shtml）

积极主动，全情投入

2017年4月11号，实验室硕士黎明的论文“SSDUP: A Traffic-Aware SSD Burst Buffer for HPC Systems”被ICS2017年会议录用。ICS是高性能计算和系统领域的顶尖国际学术会议顶级会议之一，主要关注并行应用、体系结构、系统软件、数据中心、网格与云计算等方面的研究，在国内外学术界有着很高的影响力。

论文通过分析科学计算应用常见的三种访问模式，针对高性能计算中的突发写性能瓶颈问题设计了一种基于负载感知的混合写缓冲系统，在有限的SSD资源下能够有效提高系统写性能，不仅能满足大部分应用突发的I/O请求，而且能够很好地节约成本。本刊对论文作者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

作为一名已经毕业的学生能够成为本期实验室的受访人物真的十分荣幸，非常惭愧不能像往期的博士师兄师姐一样提供丰富的科研经验，回想研究生求学时光，感觉自己也是十分平凡，并不能算得上一个牛人，因此只能简单地分享一下过往三年的感悟和写文章的一些经验。

基础知识

首先我们实验室的研究方向是偏系统方向，系统结构的研究需要对计算机的底层原理，如操作系统，网络，数据结构等相关知识有比较完整的认知，这样才会在研究源码的时候有更深的认识和理解。拿我自己的经历来说，之前在参与神威项目时，需要重新设计GlusterFS的一个模块，其中需要记录某段时间内文件操作的历史记录，在多进程的场景下，需要对文件句柄与进程之间的关系有了解才能避免出错。熟悉操作系统与网络知识，对于阅读源码也很有帮助，能更好地帮助我们理解高性能的系统是如何设计的，例如内存池，线程池的设计，索引的设计等等。

正所谓，磨刀不误砍柴工，拥有扎实的基本功能够让我们在学习别人的代码时不会陷入自己臆想的过程，知其然更知其所以然，在修改或者写代码时也会更加地有自信。

读论文

接着我想谈谈自己读论文的一些经验。读研期间的大部分时间里我们都在读论文，这也是一个老生常谈的问题，相信刚来的同学都会要接受这样的培训。这里其实不想赘述读论文的方法，谈一谈自己的爬坑过程。

我觉得我们需要阅读足够多的论文，才能达到训练的目的，我自己在头两年读了很多的文章，好的差的都有。论文看多了就会发现其中有一个阶段性的过程，比如第一个阶段，我是按照英语考试中阅读理解式的方法去了解论文在说什么，特别是摘要和介绍部分，这样我会知道自己的领域是做什么事情的，要解决的是什么样的问题。第二个阶段，在看论文的过程中我会经常问自己，这篇论文为什么能够被录用，进而学会如何提炼创新点，掌握了这种思考的能力，我觉得看论文就算是入门了。第三个阶段，通过别人的论文找idea，在经过前两个阶段的训练之后我们已经阅读了大量的论文，通过总结发现有哪些问题别人可能没有做，或者别人的方法有哪些缺陷，或者结合新的方法去解决一个大家关心的问题。拿我自己的文章举例子，我是通过思考Burst Buffer的缺陷，

认为这个系统无差别缓存的成本过高，另外通过总结他人文章中通过访问模式识别随机方法的局限性，两者结合形成了文章最初的idea。

写论文

在找到靠谱的idea，并实现之后，如何包装自己的成果也是一件不可忽略的事情。虽然不能够投机取巧，但是如何放大自己的优势是需要我们去做和思考的。在我自己写论文的过程中，我有两个经验可以分享。

第一个经验是语言方面，英语作为我们的第二语言，使我们在写文章方面中有很多劣势，我自己写文章的经验是，遇到一些把握不大的描述，我会在脑海中构思中文的表述方式，先把问题说清楚。然后快速地回忆有哪篇文章的哪些句子很接近自己的表达方式，然后去模仿他的写法，包括句式和陈述逻辑，但是注意不要抄袭。这样最后写出来的论文会比较有native speaker的感觉，给老师改的时候也会比较省心省力。

第二个经验是创新点的提炼，这个方面我得到了各位老师（石宣化老师，陈勇老师，何黎刚老师，马晓松老师）的帮助。最早我把文章的核心创新点放在了流水线模块，经过几轮的投稿和修改，石老师认为文章的创新点太过工程性，最后我们经过反复地讨论，将重点的放在了随机访问识别上，并且重写了文章的摘要，介绍和总结，最终文章才被录用。所以不仅仅是要画好看的图，做完美的实验，如何去陈述自己的成果也尤为重要，同样的东西，有时候换一种更加有说服力的呈现方式会有更好的效果。多思考，多讨论，去提炼最有说服力的创新点，才能够打动审稿人。

积极主动，全情投入

最后谈一谈自己做科研的心态吧。我觉得我们对待科研需要有积极主动的心态和全情投入的决心。研究生的三年时间其实很短暂，对一个陌生的领域去深入地理解（对多数人而言），并做出一定的成果，是需要投入很多精力的，也是非常困难的。我自己的经验就是，要积极主动地去与老师交流，不能仅限于组会上，让老师知道自己的进展和遇到的困难，不管多还是少，保持经常交流的状态其实挺重要的，这样可以避免自己少走弯路，也能够对自己的知识积累和研究进展做快速的迭代，我相信出文章的速度与交流的频次是成正比的。

第二个是要舍得花时间，去学习基础知识，阅读论文，编写代码，撰写文章，好的东西都是需要打磨的。而且基础知识并不一定能从论文中学到，所以需要我们花更多的时间去学习。我认识的实验室大牛们，基本上大部分时间都泡在实验室里。在掌握了所有的技能之后，投入的时间越多，得到的收获也会越多，组里的陆路师兄，陈明师兄，梁俊玲师兄，裴成师兄都给我树立了这样的榜样。

结尾

以上就是自己回想研究生三年生涯能够总结的心得体会了，衷心感谢实验室对我的培养，规律的作息制度让我快速地适应了职场，三年的科研训练不仅夯实了基础，也让我的视野变得更加开阔，这些都是无价的财富。离开学校之后，也十分怀念在实验室加班，和小伙伴们一起奋斗一起玩耍的校园时光。最后希望实验室越来越好，老师们收获更多科研成果，同学们论文高中，offer多多。希望大家都能青春无悔。

黎　明
2017级硕士毕业生
研究方向：大数据
Email：limingcs@hust.edu.cn

云计算与移动计算组研究规划

吴 松、刘方明、余 辰、陆 枫、肖 江、顾 琳

本方向在2018年的研究重点包括面向物联网和国产平台的轻量级虚拟化关键技术、高效能云计算数据中心资源调度方法、边缘计算基础理论与关键技术、城市计算中的交互数据感知、融合及分析技术、面向移动云服务环境的终端节能机制、区块链跨链交互关键技术、基于GPU的NFV性能加速技术等。主要研究规划和重点关注的研究内容如下。

（1）面向物联网环境的轻量级虚拟化关键技术

针对云端融合场景及人机物融合应用，研究物联网环境下的轻量级虚拟化技术，主要研究内容包括：1）特型内核构建和应用封装方法。边缘/终端设备的性能和资源都远不如数据中心强大和丰富，传统虚拟化技术过于臃肿、开销庞大，不适用于边缘/终端设备，拟研究体量轻、开销小、隔离性强的超轻量级虚拟化技术，通过结合库操作系统和硬件虚拟化技术，为应用构建定制化的特型内核，支持应用的高效部署和快速启动，提高应用部署密度。2）基于特型内核的基础运行环境。人机物融合系统涉及“云网端”不同的软硬件系统环境，如何构建统一的基础运行环境和平台是需要研究的关键内容，拟基于特型内核技术研究适用于数据中心、边缘系统和物联网终端设备的超轻量级虚拟化技术，以该技术为基础构建标准化、统一化的人机物融合系统基础运行环境。3）分布式超轻量级虚拟化镜像管理机制。在超轻量级虚拟化环境下，各类应用或服务都被统一定制为镜像文件，以方便应用高效分发与快速部署，人机物融合环境具有服务种类多、系统分布广的特点，如何管理数量巨大的各种镜像文件是需要研究的重要内容之一；拟根据人机物融合环境的特点，研究分布式的动态镜像管理机制，实现镜像快速检索、定位、部署和启动。

（2）面向国产平台的轻量级虚拟化关键技术

近年来，以申威、飞腾、龙芯等为代表的国产CPU，在自主可控、信息安全领域的应用取得了长足的进步，然而，受限于国产CPU架构、虚拟化硬件功能支持不充分、单核性能低等影响，复杂庞大的虚拟化软件运行在国产平台存在管理开销大、功能缺失、稳定性不足等问题，生态短板矛盾突出，迫切需要构建多核/众核统一的虚拟化架构和应用 API，构建面向国产处理器虚拟化的生态环境，实现和中国云标准的对接，创造聚心合力的面向国产CPU虚拟化开发平台和环境。拟开展的研究包括基于Hypervisor的轻量级虚拟化方法及其性能优化、容器系统的隔离机制以及轻量级虚拟机和容器仓库构建和管理，研发面向国产平台的轻量级虚拟机和容器构建与运行管理平台。

（3）高效能云计算数据中心资源调度关键技术

云计算数据中心业务持续增长，计算能力和密度要求越来越高，面临四大技术挑战：密度持续增加，设计难度加大；互联带宽受限，网络调整不便；资源利用低效，设备浪费严重；能耗增长过快，绿色节能堪忧。需要突破高密度设计、高速互联、资源利用率、能耗控制等关键技术，拟研发高效能云计算数据中心核心技术满足业务的更高需求。其中，高效能云计算数据中心资源调度关键技术包括：一方面，研究多维异构云资源的细粒度划分方法及细粒度资源感知技术，建立基于强化学习机制的智能化云资源分配策略自主更新学习模型，优化资源分配，从而有效提高数据中心资源利

用率；另一方面，研发多数据中心调度技术，针对多数据中心的服务负载及资源请求特点，建立请求移植和服务迁移的收益/代价评估方法，设计层次化调度机制，实现多数据中心资源的高效整合。

（4）边缘计算基础理论与关键技术

随着物联网技术的蓬勃发展，物联网所连接终端设备的数量正高速攀升，它们源源不断产生的数据也对海量数据处理的实时性提出了新的挑战。然而，当前以云为中心、“云–端”协同、“边缘–端”协同的解决方案仍存在着延迟和成本方面的固有缺陷。为了解决上述问题，依托中德国际合作项目“边缘计算基础理论与关键技术”，拟突破传统的“云–端”或“边缘–端”的双层计算架构，提出“终端–边缘–云”协同的三层计算架构，从而高效融合终端、边缘和云这三个层次在实时性、成本和能效方面的不同优势。为了高效管理边缘层有限的资源，将采用基于容器的轻量级虚拟化技术设计事件驱动的微服务数据处理模型，并构建边缘环境下的微服务性能模型。在此基础上，通过合理的任务分配和迁移策略，充分利用不同层次的异构特性来优化应用执行的延迟、成本与能耗。为了支撑不同层次间的大规模数据交互，将设计高能效以及高性价比的跨层次传输机制。预期通过实现边缘计算原型系统来检验上述策略和机制的性能。

（5）城市计算中的交互数据感知、融合及分析技术

通过对城市数据的收集和清洗，研究城市计算中交互情境数据的感知技术，建立典型应用情境的表达范式及适配模型；研究多维多源情境数据融合技术，为城市中的交互行为和性能态势分析、理解、评估与预测奠定基础；人机物交互行为的统计特征与态势模型；通过统计分析和机器学习等手段，研究数据驱动的人机物融合交互行为的统计特征与态势模型，为智能调度技术提供基础数据与模型支持；以数据驱动为手段，实现资源敏感和时空感知的智能调度模型，通过应用行为分析，利用知识图谱和增强学习技术，实现城市计算情境下的行为特征分析与预测。

（6）面向移动云服务环境的终端节能机制

随着移动云技术的发展，利用移动云服务平台，整合云端服务器资源、cloudlet服务器资源和各类空闲移动终端资源，扩展移动终端的计算和能量性能成为研究热点。然而，目前移动终端在利用云服务平台性能上仍然存在瓶颈问题有待突破：移动APP在移动终端上的自动安装部署困难、工作量大；移动终端的网络连接由于竞争波动性等问题，存在与云端的连接不畅，能耗高等问题；移动终端周围闲散的设备未得到充分利用，浪费资源。拟完成如下研究任务：研制出具有普适性的代码迁移框架，在不同应用场景中将移动APP自动发布成为代码迁移版本，扩展移动终端的计算与能量性能；明确异构网络环境下移动终端在典型应用场景中的数据传输策略，尽可能针对移动APP的数据传输特点，利用典型应用场景中多终端的优势，采用任务整合和多终端聚合的方法，降低网络的竞争性波动，提高移动终端网络传输能力；基于WIFI和用户的分布密度，利用现有成熟的随机模型，明确移动终端在异构网络中的切换策略，提高终端对网络资源的利用率；充分利用典型应用场景中闲置的多终端资源，基于D2D传输技术研制多终端共享计算和数据任务的策略，提高终端的利用率。

（7）同构/异构区块链的跨链交互关键技术

基于分布式架构的区块链技术具有公开透明、可靠加密、难以篡改、多方共识的特性，有助于解决物联网发展演进过程中的安全性瓶颈、促进信息的横向流动和多方协作，打破制约物联网发展的数据孤岛桎梏。因此，突破跨链交互技术以支撑海量物联网设备的开放共享和互联融合至关重要。拟提出适用于同构/异构区块链跨链交互的新型框架，并从以下四个维度展开研究：在应用层，采用一套统一的、基

于权限控制的操作原语描述以满足业务逻辑需求；在共识层，优化共识机制以确保执行过程可信、数据透明不可篡改；在网络层，建立同构/异构区块链的通信协议，提升信息传输的效率；在数据层，通过合理设计数据存储结构，保障数据跨链流通过程中的原子性与一致性。

（8）基于GPU的NFV性能加速技术

现有NFV性能提升的多数工作都是在应用层通过NF的部署或者网络流数据包的调度来实现。然而，在应用层对NFV性能的提升是有一定的局限性的。其次，在NFV中大部分的网络功能是无状态的，所以这些网络功能的数据包可以被并行处理。再者，在公有和私有云环境中，越来越多的高性能处理平台和集群被构建起来，如GPU集群。所以，通过GPU等硬件加速的方法提升NFV的性能成为一个研究热点，同时也是一个难点。通过GPU实现对NFV的加速，需要解决以下问题：1）现有的大多GPU平台无法对网络功能的个数和种类进行实时地调整，不满足NFV对隔离性，灵活性和可管理性的要求。所以要提供一个CPU-GPU的混合处理框架来支持NFV技术；2）设计一个资源管理策略来管理CPU和GPU上的资源，并提出一个GPU 的分流算法，根据当前的系统反馈和服务需求确定哪些网络功能可以并且应该被分流到GPU 上执行；3）当一个NF 及其网络流数据被分流到GPU上时，数据批的大小是影响吞吐量和时延的一个重要因素。因此，需要设计一个高效的在线算法根据当前的吞吐量和时延来动态调整GPU上的网络流量。

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2018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重点研发计划的申请机遇，争取获得国家项目支持。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环境中。

吴　松

博士，教授

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。

Email：wusong@hust.edu.cn

刘方明

华中科技大学教授、博导

研究方向：云计算与数据中心、软件定义的网络SDN与虚拟化技术、绿色计算与通信、移动互联网。2011年入选湖北省“楚天学者”计划、2012年入选微软亚洲研究院“铸星计划”、2013年作为国家973计划“青年科学家”项目华中科技大学方课题负责人，获华中科技大学2013年度“学术新人奖”、2014年“华中学者”。

Email：fangminghk@gmail.com

余　辰

副教授

研究方向：普适计算、移动互联网

Email：yuchen@hust.edu.cn

陆　枫

副教授

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

顾　琳

博士，讲师

主要的研究兴趣包括网络功能虚拟化、软件定义网络、云计算等

Email：lingu@hust.edu.cn

肖　江

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

支撑大数据处理的高效体系结构和系统软件
——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

1 背景与挑战

数据作为信息的载体，已经成为包括经济建设、国防安全、社会生活等各个领域最重要、最有价值的资源。为了发现和理解数据背后所隐藏的巨大价值，从商业领域到科学计算领域，大数据处理需求日益增加。总体上来看，大数据处理正日益表现出数据规模的海量性、计算过程的复杂性和内在的强并行性等显著特点。现有计算系统已很难高效支撑大数据处理需求，在特定领域的数据处理加速、内存架构以及处理模式方面都存在着难以逾越的鸿沟。当前，多核/众核等新型计算机体系结构和新型存储器件的出现为更好地支撑大数据处理提供了可能。一方面，新型加速器架构（FPGA以及ASIC芯片等）为大数据处理提供了丰富的片内并行性，在很大程度上减小了依赖分布式所构建的并行计算环境的任务间通信、大数据传输等方面的巨大开销；另一方面，新型存储器件技术（PCM等）的快速发展，也为构建新的低延迟、高吞吐的大数据处理的内存读写环境提供了可能。

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。这使得异构平台及上层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，在处理模型、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、深度学习为代表的大数据应用给异构计算环境带来挑战

现代控制流体系结构的通用性与图计算等典型大数据处理任务的多样化特征之间存在的矛盾。这一矛盾制约了特定领域大数据处理系统的性能，亟待充分考虑大数据处理任务所具有的多样化特征，设计领域通用的数据流体系结构。基于控制流程序执行模型而设计的通用体系结构仍是当前大数据处理平台的主要支撑，尤其在处理复杂计算任务方面具有难以撼动的地位。当前真正根据大数据应用特征来定制的领域通用化体系结构十分稀缺。谷歌的TPU、寒武纪的DianNao系列等人工智能芯片都仅针对极为热门的深度学习领域，而在其它众多的大数据处理领域中，领域通用化体系结构的设计还处于初级阶段。领域通用化体系结构的研发周期长、应用面窄导致了其成本偏高而预期收益偏低，其根本原因在于缺少合适的体系结构定制化理论指导，并且现有的定制硬件架构的方式缺乏灵活性。历经多年发展的数据流模型不仅有成熟的理论支撑，有利于相应体系结构的研发，而且定制软件体系结构相比定制硬件也明显具有更强的灵活性。

然而，要设计基于数据流的大数据高效处理架构，还必须考虑到：软硬件资源的异构、多层属性与数据流计算任务的细粒度诉求之间存在矛盾。这一矛盾制约了大规模数据流计算任务部署和运行的效率，亟待充分考虑新型体系结构异构计算核心和多层次存储环境的资源复杂性，设计数据流计算的执行引擎和软件环境。

硬件资源复杂化是当前计算机体系结构发展的趋势。性能、成本、能效与擅长领域各异的计算和存储设备不断增加，涌现出各类新型

的异构计算核心（如GPU，FPGA，MIC等）与多层次存储环境（非易失性存储介质NVM、3D堆叠缓存等）。这些丰富的硬件资源可能各自适合不同的大数据处理模式在不同处理阶段的需求，因此若能在运行过程中将处理任务动态映射到最合适的资源上，将能有效提升大数据处理的性能。数据流模型的异步并行特点令其天然适合利用异构资源，其细粒度特点也为动态调度提供了潜在的灵活性，然而，也为充分利用复杂资源制造了困难，带来了计算任务–计算资源的实时映射挑战。

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。

2 主要研究内容

基于上述分析，本研究组将继续延续过去几年制定的研究路线，在面向大数据处理的数据流体系结构、面向虚拟化场景的异构内存系统两个主要方面开展研究，并特别面向图计算需求，开展异构平台上的图计算系统的研究。

1）在面向复杂大数据处理需求的数据流体系结构设计方面，开展硬件层面的灵活定制、执行层面的高效调度和通信层面的动态划分三个关键问题的研究，提出数据流体系结构的设计规范和设计方法。

第一，开展面向大数据高效处理的可定制化数据流体系结构的设计方法研究，主要包括数据流抽象机、数据流加速器、存算联动机制三个方面。在数据流抽象机方面，重点突破基于数据流的并行计算理论、基于数据流的高效存储模型、数据流抽象机的形式化验证系统；在数据流加速器设计方面，为满足大数据处理的可定制化需求，研究基于FPGA的数据流加速器，突破面向大数据处理的高并发流水结构、基于FPGA的高吞吐数据流处理模式关键技术，开展数据流加速器的仿真工作，重点研究多级流水功能模块间的关联关系和负载平衡机制，探讨定制化高并发流水模式的可重构方法，提出数据流图抽象模型；在存算联动机制方面，聚合主机和加速器存储空间，实现多模式协同的组合分析与调度，引入HBM（High Bandwidth Memory）硬件支持，研究低通信延迟的数据管理与调度方法，实现高效一体化的集中管理与计算。

第二，开展面向大数据高效处理的数据流执行引擎和软件环境研究。面向大数据处理的定制化新型体系结构的引入在显著提高特定类别应用性能的同时，也造成了计算单元、内存层次结构等硬件资源的高度异构化。如何实现细粒度数据流任务在异构计算单元之间合理映射，以最优化系统性能和能效是关键问题。为解决以上问题，拟开展数据流图实时动态映射策

略、内存层次结构敏感的任务执行机制、用户友好的数据流软件编程与适配环境的相关研究。

第三，开展基于数据流的大数据高效处理的分布式通信优化方法研究。数据流计算任务的细粒度特点造成了任务间的频繁数据传输和消息通信，在大数据应用譬如机器学习和图计算中该问题更加突出。如前所述，目前大数据处理的通信优化方法基本聚焦在垂直通信优化或者水平通信优化，且这些通信优化方法受到具体应用、数据、处理机体系结构、网络拓扑等多方面因素影响，如何在大数据环境下综合考虑这些因素对垂直和水平通信进行联合优化是本项目重要研究内容之一。为解决以上问题，拟研究数据流图分布式动态划分、子数据流图独立任务分布式调度以及联合优化垂直和水平通信方法。

2）在面向云计算场景的异构内存资源管理方面，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特定应用的轻量级虚拟机定制和裁剪机制，基于RDMA的异构内存池化机制等研究工作。

第一，云计算环境下基于效益的虚拟机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，面向特定应用（如网络功能虚拟化（NFV））的轻量级虚拟机定制及裁剪机制，设计适应异构内存环境下的虚拟机定制优化。将从两个方面开展研究：（1）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，基于unikernels选取可用于构建操作系统的最小库集合，随后，把这些库及应用程序一起编译构建轻量级的虚拟机。（2）针对某个特定应用，基于一个功能完备的操作系统内核，如Linux内核，通过迭代反馈机制渐进地裁剪一些不必要的功能模块。首先将分析Linux内核模块之间的依赖关系，构成一个有向无环图DAG，从而把Linux内核模块裁剪问题转化为DAG图分解问题，提高内核模块裁剪的效率。第三，面向异构内存资源池的资源动态抽象、调度和RDMA传输优化机制。考虑到DRAM和NVM内存资源延迟的差异性，在异构内存资源池中，应用数据可以有4种典型的放置方式：本地DRAM、本地NVM、远端DRAM及远端NVM。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，权衡多应用之间资源竞争的公平性问题，设计灵活高效的内存池资源抽象和分配机制，以及运行时数据动态迁移策略。同时，在RDMA访存机制中，由于远端节点上NVM的访问延迟比其他几种数据访问模式更大，因此需要进行特别优化。对于远端节点上NVM的读操作，将采用数据预取机制来先拷贝到应用程序本地的DRAM缓冲区，从而隐藏部分远端NVM数据的读延迟。对于远端节点上NVM的写操作，将在远端节点的DRAM中开辟一个多应用共享的环形缓冲区，从而加速应用对远端节点上NVM的写操作。

3）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深度并行性与硬件匹配策略，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，有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，开展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。

3 小结

目前，围绕上述目标和任务，研究小组已经部署人力开展相关研究。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等方面开展了一定的研究工作，并开始部署数据流体系结构方面的研究。本年度研究方向的部署亟待长期坚持和努力，并通过若干重要应用示范来展现效果。

廖小飞
教 授
研究方向：系统软件
Email：xfliao@hust.edu.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

网络空间安全组研究规划

邹德清、袁 斌、代炜琦、羌卫中、徐 鹏、马晓静

本研究方向在2018年将重点围绕云计算安全相关的一系列关键问题开展研究，包括：代码漏洞检测、软件定义网络安全与主动防御、区块链安全、云环境隐私追踪取证等。主要的研究规划和关注的研究内容如下：

（1）代码漏洞检测

漏洞是网络空间安全最核心的命脉，软件与系统漏洞是网络空间安全威慑和防御的基础，已成为国家网络空间安全的重要战略资源。由于大量新型软件、系统的出现过程中必然产生漏洞，这对软件、系统的安全提出了新的挑战，急需开展深层次、大规模、智能化的漏洞检测研究。传统的源代码漏洞静态检测不仅主观性强，而且难以同时达到较低的误报率和漏报率。深度学习不需要人类专家来定义特征，为解决上述问题提供了一个新思路。然而基于深度学习的漏洞检测研究目前还存在诸多挑战，例如，漏洞检测粒度粗，没有涵盖各种类型漏洞的大规模标注数据集，哪种深度学习模型适合检测哪种类型的漏洞是未知的，缺少除是否含有漏洞外更全面的信息等。针对上述问题，拟开展以下研究：1）研究细粒度候选漏洞代码段生成，基于语法和语义两个层面为深度学习提供数据支撑，有望解决漏洞检测粒度粗的问题；2）研究面向训练程序的候选漏洞代码段标注，基于公开漏洞数据库中的大量数据，对训练程序中的候选漏洞代码段进行自动标注，并实现已标注候选漏洞代码段的数据量扩充，有望构建涵盖各种类型漏洞的大规模标注数据集；3）研究面向训练程序的漏洞模式智能化学习，采用面向漏洞代码数据特性的深度学习模型实现各类型漏洞模式的自动生成，有望给出哪种深度学习模型更适合检测哪种类型的漏洞；4）研究面向目标程序的漏洞检测与模型解释，基于候选漏洞代码段进行“有无漏洞--漏洞类型--漏洞位置--漏洞严重度”四级漏洞检测，有望提供除是否含有漏洞外更全面的漏洞信息，并且基于漏洞检测结果进行深度学习模型的解释，进一步改进模型的有效性。

（2）软件定义网络安全与主动防御

传统网络安全的实现主要依赖于静态的网络安全功能部署与被动的安全策略实施。然而，静态的安全防护手段已远远无法安全需求，尤其在具有强动态特性的云环境中；被动的攻击抵御方式往往要滞后于攻击，具有响应较慢的不足；此外，传统网络安全防护方案不具有可编程性，往往需要人工介入，具有灵活性不佳的缺点。为解决现有防御手段的不足，需要结合新的技术，实现动态的主动的可编程的攻击防御。而近年来，软件定义网络（Software-Defined Networks）、网络功能虚拟化（Network Function Virtualization）与移动目标防御（Moving Target Defense）等技术飞速发展，受到了广泛关注。其中，SDN技术能够提供集中化、自动化的网络管理，其具有的可编程特性使得网络操作的灵活性大大增强；NFV技术能够将高度定制化的硬件网络功能进行虚拟化并运行在X86通用平台上，使得按需的动态网络功能配置成为可能；而MTD技术则能够弥补防御者在时间上滞后于攻击者这一

不足，有效提高攻击的难度。SDN技术与NFV技术的结合能够为软件定义网络安全提供平台基础，使得具有可编程和按需配置的攻击防护成为可能。进一步地，结合MTD技术后，通过变化攻击面，降低攻击的成功率，从而实现对攻击的主动防御。为实现软件定义化的网络安全与主动防御，拟开展如下研究：1）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；2）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；3）研究基于SDN的抗数据窃取的安全网络传输，利用SDN的可编程特性，对数据分发过程进行有效控制，避免关键网络数据被恶意网络节点盗取；4）研究SDN中基于MTD的主动防御方法，利用SDN的多级流表、组表等组件，实现对网络数据的差异化处理，通过增强网络行为的随机性、降低网络的可预见性、增加网络的噪音等方法实现对复杂攻击的主动防御。

（3）区块链安全

区块链被认为是第四次工业革命的关键技术，由于其拥有不可篡改、可追溯，以及在分布式系统中实现去中心化的信任机制的特点，区块链技术的研究与应用呈现出爆发式增长态势。但是目前区块链技术还不完美，本研究方向针对区块链安全上的一些问题开展研究，主要研究内容和规划如下：1）数据交易安全，数据交易产业日益增长，但是缺乏一种安全可靠的数据交易机制来防止数据沉淀等现象。针对此类问题，从以下方面展开工作：研究通过智能合约进行数据交易的新机制，数据买方将自己的数据分析代码以智能合约的形式加密部署在区块链之上，之后数据卖方通过将数据发送给在安全环境下运行的智能合约来完成数据交易。在安全环境下运行的智能合约不会泄露双方的任何隐私，这样即保证了数据卖方的数据不会被窃取，又保证了买方的数据分析代码及结果不会泄露，进而打破现在数据交易中普遍存在的数据沉淀等问题。2）可靠的SPV轻钱包，现有SPV钱包存在一系列安全问题，比如私钥易被窃取或伪造；通过修改生成钱包地址的控制流，篡改生成的钱包地址，导致交易无法正常进行；篡改本地区块头导致钱包验证结果不可靠。为了解决以上问题，研究基于trustzone实现的安全可靠的SPV轻钱包，它能够阻止恶意的操作系统窃取或篡改用户的私钥及钱包地址，同时对钱包的交易流程和本地区块头进行保护，保证验证结果的可靠性。3）激励价值创造的POV共识，已存在的共识机制主要是用于资产交易，但并没有在价值创造这一块做出贡献。为此，研究设计一种基于价值证明的共识机制（POV），它依据商户利润进行价值评估，并在节点处理交易的时候完成价值激励。POV可以实现奖励真正为人类社会创造价值的群体，激发更多的价值创造，以促进价值流通。4）区块链版本控制，由于区块链的不可篡改性及攻击后状态不可逆转，当合约有漏洞或区块链受到攻击时，只能让每个节点定点记录之前状态并在检测到问题后进行自动回滚；或由管理人员介入下在之前某一区块下进行分叉，造成链上节点时间和资源上的损失，同时带来中心化的风险，且难以达成共识。针对这种问题，研究区块链的版本控制方法。当检测到合约存在漏洞，主链节点放弃上次分叉后新挖的区块，侧链成为最长链，主链节点转而在上次分叉形成的侧链上继续挖矿。从而在去中心化的环境下有限地避免合约漏洞可能造成的风险与资源浪费。

（4）云环境隐私追踪取证

云计算是主流的信息系统技术，但是云计算环境下的用户隐私保护和隐私侵犯的追踪取证一直是一个挑战，这主要是由于在云环境下用户隐私窃取具有泄漏痕迹易销毁、用户隐私窃取难追踪、难定位和难溯源的难题。同时，云计算平台的虚拟化特性，例如多租户、迁移等，也为增加了取证难度。当前主流的云计算环境中取证技术体系和标准体系并不成熟和完备。因此，研究云环境下隐私数据的追踪和取证具有十分重要的意义。针对上述问题，拟开展以下研究：1）研究虚拟计算环境中隐私数据的访问监控。主要包括通过相关API定义隐私数据，监控隐私数据在系统中的流转情况，通过生成的可视图来辅助管理员进行取证分析等；2）研究跨虚拟机的隐私流转追踪。分别从面向虚拟机回滚和面向虚拟机迁移行为进行隐私数据的追踪取证；3）研究虚拟计算环境中隐私计算分析与取证。实现内存中隐私攻击痕迹的还原，以及将连续内存镜像和逆向取证进行结合，来完成针对隐私数据的攻击重放和取证。

目前，围绕上述研究内容，研究小组已经部署人力开展相关研究，在区块链安全方面，正在开展区块链共识算法、数据安全、钱包安全和区块链版本控制等方面的研究工作；在代码漏洞检测方面，正在开展面向各种类型漏洞的候选漏洞代码段的提取与数据标注工作；在软件定义网络安全与主动防御方面，正在开展侧信道攻击抵御、数据安全分发机制等方面的研究工作；在云环境隐私追踪取证方面，正在开展基于动态污点追踪的全系统隐私数据监控的研究工作。另一方面，小组也会持续重视科研成果转化，解决关键领域中实际的安全问题。

邹德清
博士，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

袁　斌
2013级博士
研究方向：主要从事云安全、软件定义网络安全等领域的研究。
Email：yuanbin@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、区块链等领域的研究。
Email：wqdai@hust.edu.cn

羌卫中
博士，副教授
研究方向：主要从事系统安全、云安全等领域的研究。
Email：wzqiang@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、云安全等领域的研究。
Email：xupeng@hust.edu.cn

马晓静
博士，副教授
研究方向：主要从事多媒体安全方向的研究。
Email：lindahust@hust.edu.cn

大数据组研究规划

石宣化、陈汉华、赵 峰、袁平鹏、华强胜、谢 夏、丁晓锋、于东晓、黄 宏

本研究方向在2018年的研究重点是大数据基础算法、大数据处理支撑技术（如异构内存计算、流计算、大规模图计算、HPC IO调度等）、大数据技术应用（如社交网络动态演化）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）面向新型体系结构的低通信复杂度分布式算法研究

CPU时钟频率和内存带宽以及网络带宽的发展差距使得主存和缓存之间的数据移动（称为垂直通信）以及处理机之间的数据移动（称为水平通信）相对CPU运算具有更高的延迟和能耗代价，因此设计低通信复杂度（并行垂直和水平通信开销）算法不仅可以提高吞吐率还可以降低能耗，这在大数据处理背景下尤为重要。非易失内存（NVRAM，譬如相变存储PCM 和NVDIMM）由于其具有字节寻址、高存储密度以及读写不对称等特性使得它可以替代传统内存和外存，从而导致计算机体系结构发生变化。和传统垂直通信中只考虑降低读写的总次数不同，由于NVRAM中写比读具有更高的延迟和能耗，因此还要进一步降低写的总次数。

大图计算，譬如求解图的直径、半径，三角形计数，介数中心度求解、K-核分解等是大数据处理的重要分支。国际上高性能计算领域已经开展以线性代数方法来求解大图计算问题的研究，典型的项目是一个由MIT、Berkeley、CMU等美国名校以及Lawrence Berkeley等国家实验室以及Intel、NVDIA等企业参与的GraphBLAS。用代数方法来求解图计算问题的一个主要原因是并行数值计算相对比较成熟，但是图计算主要为稀疏矩阵计算。此外，如前所述，以NVRAM为基础的新型体系结构也对算法设计带来了重大挑战。本项目将以一系列大图计算问题为例，一方面通过把图计算问题转换成相应线性代数求解问题，从而在基于NVRAM的新型体系结构下对这些大图计算问题设计低通信复杂度分布式算法，另一方面还需要证明相应的通信复杂度下界。除了大图计算问题，对于当前机器学习中常见的以迭代计算为主的优化算法，譬如梯度下降法，我们也尝试对这些问题设计低通信复杂度分布式算法。

（2）面向数据密集型应用的异构内存管理优化

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，尤其是对于大规模机器学习等新型数据密集型迭代性应用。另一方面，由于单机DRAM内存容量的增长遇到瓶颈，混合异构内存作为一种新型架构越来越被大数据领域接受和采用。但是异构内存根据其底层硬件实现方式的不同，在持久化存储能力、字节寻址能力、读写带宽差异和随机读写性能等方面都有很大差异。现有大数据处理系统没有针对异构内存硬件特性进行优化，存在数据访问效率低下的缺陷，导致最终内存计算

应用时效性差、性能波动大等问题。为解决数据密集型应用中异构内存对象管理问题，拟开展如下研究：1）研究混合异构内存下的高效对象堆内存存储模型，为运行时系统提供统一的对象管理接口；2）研究面向异构内存的对象编程模型，对用户支持不同层次的抽象程度，为不同特征应用提供灵活的性能和编程易用性选择；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混合内存自动垃圾回收优化机制，主要通过将对回收性能影响较大的对象引用元数据优先调度到DRAM中存储，从而降低回收过程访问NVM带来的额外开销。

（3）HPC环境下基于软件定义存储的IO调度优化

在存储业界，由于存储系统的软硬件紧耦合、管理接口不统一等限制因素，不同存储厂商间的存储设备和解决方案长期存在技术壁垒， 多数现有的存储系统是单一、集成的系统，只支持特定的硬件和软件组合，存储系统缺乏灵活性，无法充分利用不断涌现的新硬件产品的能力和新平台，并且只能进行有限度的扩展，无法满足用户快速增长的数据存储的需要；同时，不同存储系统之间只能实现有限程度的统一管理，由于底层不同厂商的存储设备无法形成统一、可调度的存储资源池，存储系统的管理成本开销会日益庞大。

为了打破传统存储系统软硬件紧耦合所造成的割裂状况，软件定义存储（software defined storage，SDS）理念被提出。SDS将存储的功能从传统存储系统中抽象出来单独部署，通过软件实现，不再是硬件设备上的固件，实现存储对应用的按需配置。目前软件定义存储的研究还处于起步阶段。为提出有效的软件定义存储解决方案，拟开展以下研究：1）高效抽象存储资源，方便系统分配；2）设计高效稳定的控制逻辑，有效实现数据密集型应用对存储资源的按需分配；3）分析软件定义存储下多应用竞争时存在的资源抢夺问题，设计合理的高效调度方案，保障应用的服务质量；4）延续以往研究，分析应用访问模式，结合Burst Buffer 特性深入优化调度方案。

（4）大数据流处理技术

随着互联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。分布式流处理技术日益成为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。然而流数据具有泛在性高、实时性强、连续不断、动态性高、关联稀疏等特征，对处理系统的时效性、可靠性、空间开销等提出了苛刻的要求。本方向未来一年拟重点开展如下研究：1）时序流式大数据关联处理融合计算系统；2）泛在多源异构流数据的高时效流连接技术；3）时序流式大数据近似存储和处理技术；4）大规模分布式流处理系统关联容错技术；5）面向异构资源的流处理系统细粒度资源管理和系统智能配置技术。

（5）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。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）图划分是并行分布处理大图的前提。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）下的图处理任务调度策略。4）研究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）研究大规模并发事务环境下，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。

（6）大规模知识管理与搜索

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，获取有价值的知识，并有效管理这些知识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。有鉴于此，我们主要研究：（1）大数据环境下的知识管理，从知识生成、存储、组织、自治、导航、检索等层次出发，研究数据驱动的知识发现基本理论和关键技术。如：研究知识库的构建技术，研究针对元数据、知识实体及语义关联信息的自动抽取技术，智能构建知识库；研究知识库的弹性存储模型，支持大规模知识库的增量式更新，研究弹性存储模型以及访问策略等。（2）大数据的实时搜索，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，主要内容包括深层网络搜索、社交网络搜索、泛在网络信息搜索、数据流链路预测等。

（7）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。5）基于新型的计算设备和机器环境下的隐私保护。新型的计算任务依托于新型的计算设备和机器环境，在CPU/GPU、SSD/HD、RAM、FPGA、RDMA等硬件设备混合的物理环境下，数据的存储、传输、计算过程中的隐私安全问题显得更为复杂化和困难化。因此，研究计算平台硬件层面的可信机制、安全机制、检测机制、防范机制具有重要意义。

（8）基于三角形结构的网络动态演化

面临大数据时代的来临，各式各样的社交网络的应用不断普及并深入人心，对社交网络进行挖掘最重要的课题之一就在于对网络结构的理解以及网络演化规律的掌握。目前对网络的动态演化的研究主要分为两类：基于边、点等基本元素变化的研究和基于群体结构的研究。前者从边、点等基本元素出发，考虑边、点状态的改变，例如，对网络中边的建立建模成链路预测的问题，对网络中点状态的改变建模成社会影响力扩散的问题。后者从宏观视角出发，考虑网络中的群体结构的演化，这一问题可以建模成社群发现的问题。然而，从宏观的群体结构的角度出发研究网络的动态演化规律很难了解网络内部的深层机理，而从边、点基本元素出发的研究只能抓住简单的边、点自身的属性，从而忽略了边、点之间可能存在的耦合关系，不能推演网络演化的真实过程。因此，我们拟引入三角形结构，从一个微结构角度——介于宏观与微观之间的一个视角，对符号社交网络中的动态演化机理进行探究，期望在考虑网络中边、点自身属性的同时考虑边、点之间的耦合关系，从而更好地研究网络演化的深层机理，深入理解动态网络。具体拟开展以下研究：1）利用数据挖掘技术，找出网络演化过程中三角形结构的演化规律；2）构建基于三角形结构的网络动态演化模型，对网络的未来状态进行预测；3）考察基于三角形结构的网络动态演化模型的有效性及可扩展性，研究其在实际网络中的应用。

（9）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，本研究方向将继续在医疗、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。

2017年，大数据研究小组从德国引进博士一名，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。一方面拟争取能在2018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。
Email：chen@hust.edu.cn

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
副教授
主要研究领域为并行分布式算法、无线网络、图数据挖掘。
Email：dxyu@hust.edu.cn

黄　宏
博士，讲师
主要研究领域：数据挖掘；社交网络分析；社会计算；大数据
Email：honghuang@hust.edu.cn

智能运维，云数据中心运维的未来之路

（熊 壮 http://blog.csdn.net/cloud_architect/article/details/74517037）

在云计算时代，IT系统建设越来越成为企业发展至关重要的一环。业务系统，以及支撑业务系统运行的基础设施通常是企业关注的首要目标；然而，保障业务健康运行的背后“功臣”——运维系统同样至关重要，因为每一次IT系统的转型，运维系统和业务保障都是最艰难的部分。在当前企业IT系统向云架构转型的时刻，运维系统再一次面临着新的挑战。

云架构对运维系统的新需求和新挑战

随着越来越多的企业拥抱云计算，为了支持业务系统的快速上线、灵活伸缩以及更高的SLA要求，再加上有限的IT运维成本，运维人员将面临比以往更大的运维压力。在运维拥有海量设备且高度复杂的云数据中心环境时，如何提供99.95%或以上的高质量IT服务，提升效率并降低成本，是运维团队当前面临的最大挑战。

“保障高运维质量”云数据中心的设备规模从几十/几百向几万/几百万数量级演进时，海量硬件设备的使用对硬件故障的快速定位和隔离将带来巨大挑战；同时，采用虚拟化和分布式弹性技术也加剧了云数据中心的复杂度。这些都会导致运维难度增加，小概率故障成为常态且影响加大，用户级的99.95%或以上的服务质量承诺（SLA）很难保障。

“提高运维效率”虚拟化技术和众多开源技术的引入使得运维变得越来越复杂，传统人工运维模式处理速度慢、出错概率高。此外，传统人均50～100台设备的维护效率，在大规模云化环境下，需要投入大量人力。

“保持低运营成本”传统IT的资源使用率通常小于20%，在云化后资源使用率有所提升，但是个性化、按需弹性需求导致资源碎片化、负载不平衡以及扩容规划不精准，可能会造成整体资源利用率并没有达到规划目标，运维成本居高不下。

为了提升资源的利用率，云架构下资源是共享的，而非独占，这与传统IT完全不同。云计算通过自动的弹性伸缩策略来实现资源共享与用户体验及业务可用性之间的平衡，这是云计算的核心优势之一。但这也带来了运维的新需求和新挑战，即运维人员往往并不知道业务系统具体运行在哪个硬件上，故障定位变得非常困难，解决这种不可知性要求运维系统要做到“更加全面的系统监控”，从而实现“可知性”。

分布式架构的云计算系统，其资源调度、业务伸缩、故障隔离和故障修复等都是自动化的，不可能基于人工来完成，这已经完全颠覆了传统IT的软件安装部署、业务使用和管理维护模式。因此，运维的工作不再是传统的运维管理，而是构建自动化运维模型和运维工具，这不但对运维人员、更对运维系统提出了新的要求。

智能化的故障预防、发现与自愈

传统模式下，运维人员的工作模式是被动等待问题发生，然后再进行故障处理。根据有关数据统计，运维人员平均每天计划内的工作只占50%左右，剩下的时间都是在到处救火。随着云数据中心规模快速增长，运维人员需要处理的事件量越来越大，人工救火将力不从心。这就需要一个智能的运维平台，利用大数据关联分析与机器学习技术为运维系统赋予人工智能，提供从故障预防到故障定位、再到故障闭环的智能保障能力。

“主动故障预防”故障处理再迅速也不如不产生故障，尤其是在大规模云数据中心场景下，即便很低的故障率也会产生一定规模的故障，为了避免到处救火，最好的方法是做好防火工作。

“及时故障发现”云数据中心由于技术堆栈层次多、技术架构复杂，如何识别故障是个很大的难点。构建一个从资源到租户体验端到端的监控体系，全面掌握系统运行状态数据，有助于准确识别出业务系统响应慢、查询速度慢、产品质量差（问题多、交易失败率高）和用户数量少/资源利用率低等问题的根源，推动技术团队不断改进，达到持续优化的运维管理目的。

“智能故障定位”云时代由于分布式和微服务化软件架构的流行，业务调用关系愈发复杂，出现故障后，对故障的快速定位形成了很大的挑战。

“自动故障修复”云数据中心规模的扩大带来了一个很大的问题——故障数量的提升。根据华为自己的数据中心运维经验，一个较大规模的云数据中心，如果不进行故障的自动化归类和处理，每日各种级别的故障单可能超过上千个。因此，迫切需要运维系统能够识别常见的故障，并有相关的故障自愈策略进行匹配。当故障发生时自动执行闭环策略，对于常见故障无需人工干预即可自动闭环解决。

云运维作为云计算必不可少的组成部分，会越来越展示出其重要性，成为云计算的核心竞争力之一。下一步华为将加大人工智能在云运维的投入与实践，让数据中心机器人融入更多的运维业务场景，替代传统的手工操作，提供高度自动化和智能化的“无人值守”式云数据中心运维解决方案。

言 论（转载）

当你开始厌倦无聊的代码，烦人的数据，琐碎的步骤，开始动摇时，那么恭喜你，只要你再坚持一下，那么成功就不远了。

王 华

（https://m.weibo.cn/status/4212719164606630）

如果你有过迷茫，仰望天空，阳光灿烂般微笑，你为何要如此困苦？

如果你有过彷徨，回想最初，既然决定和选择，你为何要轻易言败？

如果你有过放弃，看看别人，为了圆梦而努力，你为何要因此退出？

二十五岁的危机，不要慌。坚定，前进，付出的努力都会开出花来。

（汤媛媛 https://user.qzone.qq.com/384530689）

科研的道路不是一条平坦大道，过程中有汗水也有泪水，不论结果如何，一路上我们终将会有收获；漫漫科研之路，愿我保持一颗坚韧的心，不气馁，不放弃，一路拼搏。

（吴俊芳 https://user.qzone.qq.com/1395600475）

做实验不能放过任何一个细节，该有的处理过程一步也不能少，否则将付出超过原来几倍的力气去重复这个过程。没有谁做实验会一帆风顺，不要气馁，牛人很多都是从困难的境况中走出的。

（马 阳 http://grid.hust.edu.cn/graduate/space.php?uid=6344&do=doing&doid=7134）

从虚函数的实现，到虚表劫持攻击

（刘本熙 http://sec-lbx.tk/2017/06/27/%E8%99%9A%E5%87%BD%E6%95%B0%EF%BC%8C%E5%8E%9F%E7%90%86%E5%92%8C%E6%94%BB%E5%87%BB%E6%96%B9%E5%BC%8F/）

虚函数与多态

继承和多态，是面向对象中老生常谈的话题。C++中，我们也可以经常看到virtaul、override这样的关键字；这正是虚函数的标志。虚函数就是为了解决多态的问题：如果要使用一个基类的指针，根据对象的不同类型去调用相应的函数，就需要使用虚函数了。通俗的说也就是同一个入口，却能够调用不同的方法。

通常，对于虚函数的调用，往往在运行时才能确定调用哪个版本的函数。这是由于基类的指针或者引用，其动态类型必须在运行的时候才能确定（它具体指向了什么类型）。而“动态绑定”就指的在运行时，根据对象的类型，调用具体方法的过程，这个过程正是通过虚函数表实现的。

抽象基类指的是有纯虚函数的类。纯虚函数，指的是没有函数体的函数，通常通过在函数体的位置写上= θ 来表示。对于抽象基类，是不能直接创建一个对象的；但是可以创建它它们的派生类的对象：只要它们覆盖了纯虚函数。纯虚函数表示这个函数的具体实现全部交给派生类去做。

虚函数表与虚函数的调用

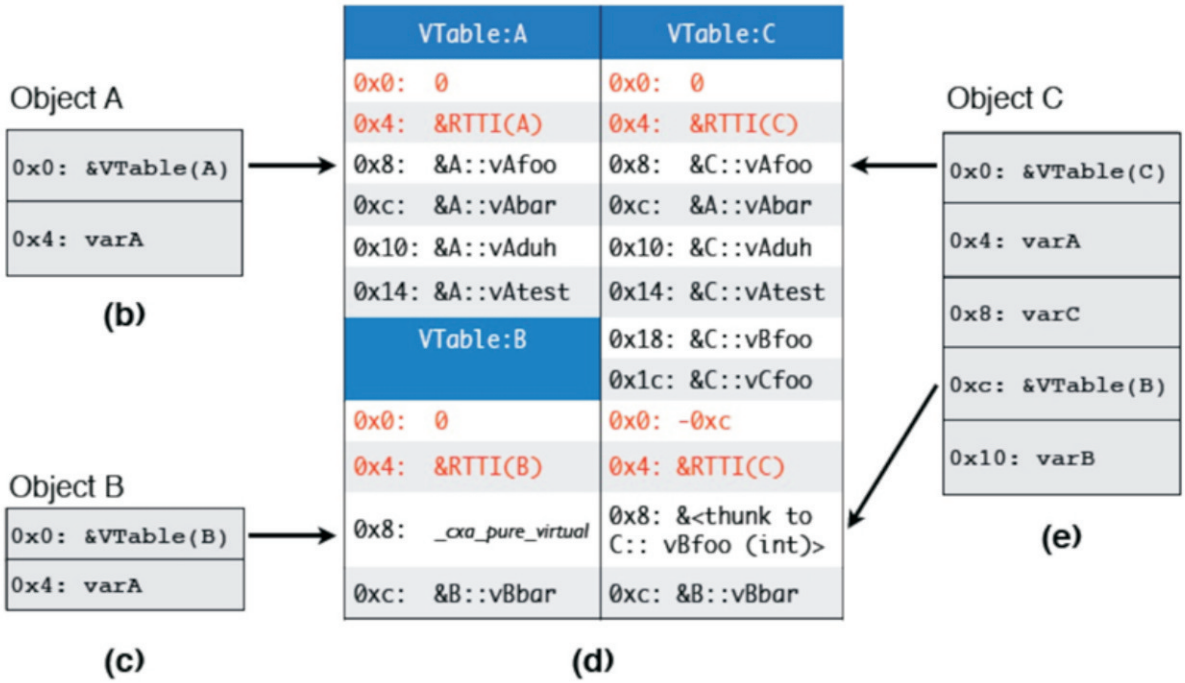
那么，“动态绑定”是如何实现的呢？这便是借助于虚函数表来实现。对于每个具有虚函数的类，都会有一个对应的虚函数表vtable，其代码和对应内存结构如下所示：

```
class A {
    int varA;
    public:
    virtual int vAfoo(int a, int *b){
        return a + (*b);
    }
    virtual int vAbar(int a){
        return a + 1;
    }
    virtual bool vAduh(){
        return true;
    }
    virtual int vAtest(int a){
        return 0;
    }
    void Afoo(){
        this->vAduh();
    }
};

class B {
    int varB;
    public:
    virtual int vBfoo(int a) = 0;
    virtual bool vBbar(int b){
        return b == 0;
    }
    char *Bfoo(char *c){
        return c;
    }
};

class C : public A, public B {
    int varC;
```

```
public:
int vAtest(int a){
    return -(a);
}
int vAfoo(int a, int *b){
    return *b;
}
int vBfoo(int a){
    return a - 1;
}
virtual void vCfoo(){}
bool vAduh(){
    return false;
}
};
```



这个表中的每一项，都是一个虚函数的地址，也就是虚函数的指针。而每个对象的第一个值都是虚标指针，它指向了所对应虚函数表的第一个表项（也就是虚函数表的基址）。每次调用虚函数时，都会首先通过这个虚表指针，找到虚函数表，然后再在虚函数表中，找到真正的虚函数的地址，并进行调用。假设存在有多继承的情况，那么就会有多个vptr，分别放在对应的基类对象的开头位置。

虚继承

对于“菱形继承”情况（也即两个子类继承同一个父类，而新的子类又同时继承这两个子类），则可能产生二义性问题。例如下面的情况，那么D中就会保存两次A中的变量和函数，并且在使用时也会很不方便，必须利用域作用符来使用变量和函数。

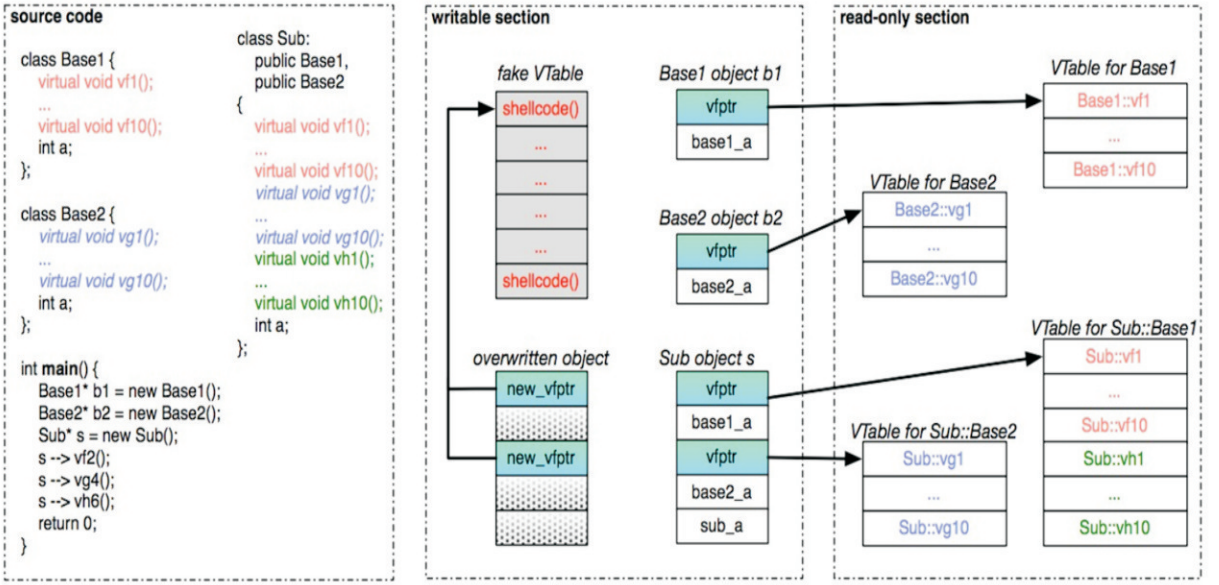
```
    A
   / \
 B1   B2
   \ /
    D
```

虚继承是在继承时，在基类类型前面加

上virtual关键字。虚继承能够解决基类多副本的问题：在任何派生类当中，虚基类都是通过一个共享对象来表示的，它们通过指针去访问这个基类中的内容；它不用去保存多份基类的拷贝，而是只需要多出一个指向基类子对象的指针。从内存布局上来说，在虚表的负offset位置，会保存一个指针指向虚基类对象。

也就是说继承自A的虚函数和对象，全部只保存一份在D自身的子对象中，相比不使用虚继承，它删除了B1和B2当中的（2份）基类成员；它自己则需要保存一份基类成员和偏移指针；而如果要用B1和B2的指针或者引用去访问一个D对象时，那么访问A的成员则需要通过间接引用来访问；也就是说子对象需要有一个偏移量，指示在内存中，基类的位置。其内存布局一般如下：

内 存
B1的虚表指针
B1的偏移指针
B1的数据成员
B2的虚表指针
B2的偏移指针
B2的数据成员
D的虚表指针
D的偏移指针
D的数据成员
A的虚表指针
A的数据成员

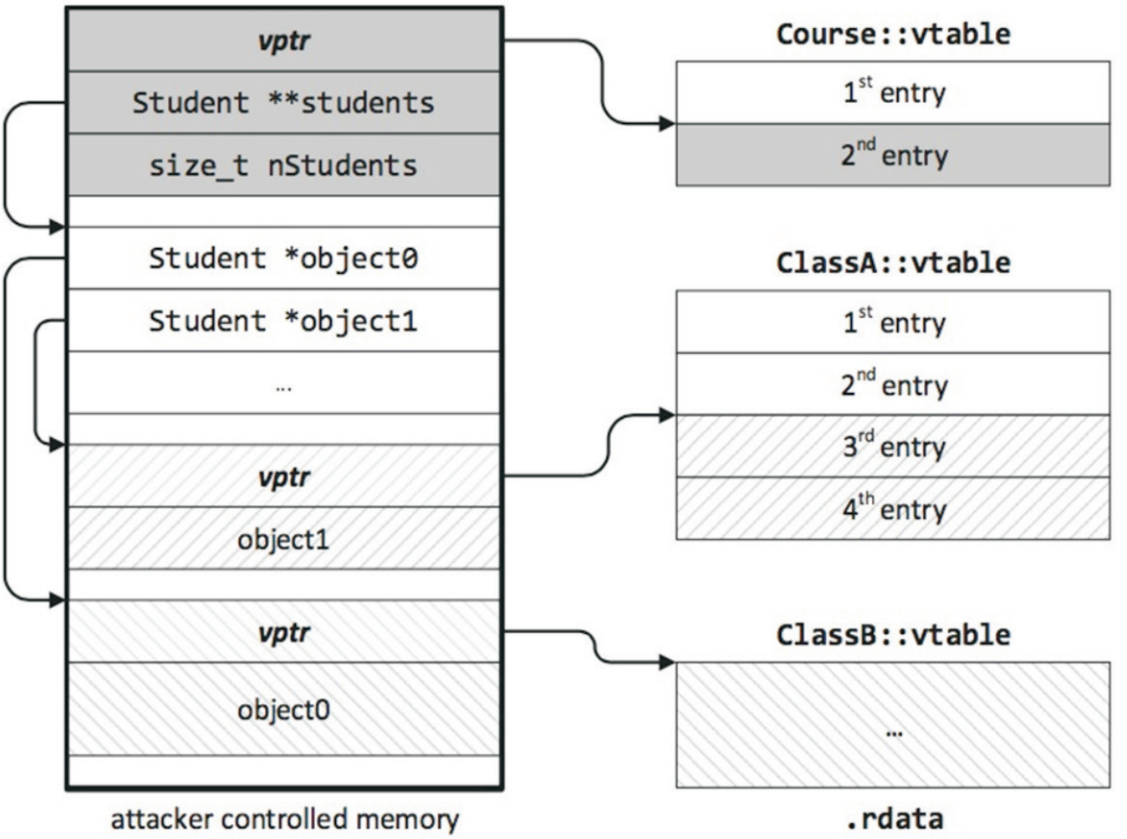


(a) source code (b) object layouts (c) vtables

虚表与劫持攻击

在C++程序中，%90以上的间接调用都是vcall。篡改程序中的虚函数调用，是劫持C++程序的一种常见手段。这里简单说说常见手段。

一种方法是虚表注入。众所周知，虚表保存在程序的.rodata段中，它是可读，不可写的；



而对象当中的虚表指针却是可读写的状态；因此篡改虚表指针是较为直接的方式。

如图，如果利用漏洞（overflow、use-after-free等）在内存中构造一个虚假的虚表，并且将对象中的虚函数指针指向注入的虚假的虚表，那么在虚函数调用时，就会调用虚假的虚函数。甚至只需要一次虚函数调用就能够通过shellcode完成攻击。

当然，如果程序进行了一定程度的保护，例如检查虚表指针是否属于.rodata段，攻击就只能依赖于现有的虚表来构造了。Counterfeit Object-oriented Programming就提出了这样一种方法。

可以看到，这种方法没有注入新的虚表，而是将vptr的值，指向了虚表中的不同位置（而不是虚表的起始地址）。如果能够构造一系列的虚假对象，那么就可以在一次循环中（比如某个对象数组的依次析构），在调用同一个虚函数时，实际上调用不同的函数，从而构造一个虚假的执行链。看到这里，也许你会有疑问：仅仅用有限的虚函数，能够构造图灵计算的攻击吗？答案是肯定的：有兴趣的话可以阅读一下原文，通过拼凑虚函数，是能够组合出各种语义的。

小结

可见，虚函数是面向对象语言中，十分巧妙而又必不可少的设计；但它的特点也使得它成为黑客滥用、攻击的目标。指的庆幸的是，目前已经有一些开销较小的方法，能够保护虚表和虚函数了。

一种高性能无锁队列设计

（胥清清 http://blog.csdn.net/u011305688/article/details/79407001）

分布式与多核处理器在共享资源的情况下均要求在线程安全环境下完成提交的任务，在多线程并发处理大量数据任务情况下为解决多生产者多消费者，任务队列线程安全设计查找到一种高性能无锁队列，进行学习、研究。

主要问题—资源竞争

一个通用的无锁队列似乎相当容易实现。问题的根源在于相同的变量必然需要与多个线程共享。例如，采取一种基于链表的通用方法，至少需要共享列表的头部和尾部，因为消费者都需要能够读取和更新头部，而生产者都需要能够更新尾部。当多个线程需要更新队列同时要求保持队列处于一致的状态下肯定会导致资源的互斥访问。例如，如果消费者从队列中读取最后一个任务节点并且更新头部，尾指针不应该指向它，因为该任务节点很快就会被释放。但是消费者可能被操作系统打断，如在更新尾部之前暂停几毫秒，在这段时间内，尾指针可以被另一个生产者更新，然后第一个消费者将其设置为null就太晚了。

解决共享数据安全访问是无锁编程的关键。通常最好的方法是设想一种算法，它不需要同时更新多个变量以保持一致性，或者增量更新仍然使数据结构保持一致状态。现已有不同的技巧可以使用，如在指针的最后两位存储额外的状态，指针和引用计数等，但像这些方法只能越走越远，真正的解决办法是开发算法本身。

设计思路

更少的线程在相同的数据上进行斗争，必然降低开销。因此，不能使用一个单一的数据结构，进行所有的操作。如果每一个生产者线程对应一个队列，这意味着不同的线程是完全平行的，彼此独立。

当然，消费者获取任务稍微复杂一些，任务出队时我们必须检查每个任务队列。从不同队列中获取任务的顺序并不重要，而同一生产者产生的任务的相对顺序至始至终是一致的，尽管任务队列之间获取任务的相对顺序可能交错，但并无不可，因为即使在单队列模型中，不同生产者产生的任务入队的顺序也是不确定的（因为不同生产者之间有一个竞争条件）。这种方法的唯一缺点是，如果队列是空的，必须检查每个子队列以确定这一点（同时，在检查一个子队列的时候，以前空的队列可能已经变成非空），但实际上这不会造成问题。在非空的情况下，由于子队列可以与消费者“配对”，所以总体上的争用更少，数据共享达到接近最优的水平（每个消费者都与一个生产者匹配），这个“配对”是使用一种启发式的方法，必须在任务出队期间保持某种状态——这是通过用户指定的“令牌”来完成的。注意，令牌是完全可选的——队列只返回到搜索每个队列中任务，只是在涉及多个线程时稍微慢一点。

每个队列中使用的核心算法不是基于节点链表（这意味着不断地分配和释放或重用元素，并且通常依赖于一个比较和交换的循环，在激烈的争用过程中可能会比较慢），每个队列基于数组模型，没有链接单个元素，而是多个元素一个“块”。队列的逻辑头指针和尾指针使用原子表示。在这些逻辑指针和“块”之间，存在一个将每个指针映射到该块中索引的

方案。入队操作只是增加了尾部标记（每个子队列只有一个生产者线程）。出队操作增加了头部标记，如果头部小于尾部，然后检查是否意外地增加了头尾的头部（这可能发生在争用的情况下——每个子队列有多个消费线程）。如果头部超过了尾部，那么就会增加一个校正计数器（使队列最终保持一致），如果不增加，它就会继续前进，并增加另一个整数，从而使它成为实际的最终逻辑索引。这个最终索引的增量总是在实际队列中生成一个有效的索引，不管其他线程在做什么或已经做了什么。这是有效的，因为只有当保证至少有一个元素被删除时，最终索引才会增加（当第一个索引被增加时，它会被检查）。

入队操作是通过单个原子增量完成的，而一个出队是快速地完成两个原子增量。当然，这需要对所有块分配、重用、引用计数、块映射的实现，这一点很重要，大多数的成本分摊到“块”上。这个设计允许极其高效的批量操作——原子的指令（这往往是一个瓶颈），一个块上入队n个任务的开销几乎和入队1个任务的开销相同，出队同理。这是真正的性能提升的原因。

生产者队列列表

该设计需要维护所有生产者队列（LIFO）列表，这个列表是使用尾指针和每个生产者的入队next指针来实现的。尾指针最初指向空；当一个新的生产者被创建时，它首先读取尾部，然后使用尾部的设置，然后使用CAS操作将尾部（如果它没有改变）设置到新的生产者。生产者不会从列表中删除，仅标记为不活动。当一个消费者想要出列时，它只需寻找一个带有元素的SPMC单生产者多消费者队列的生产者列表。

出队策略

消费者可以将令牌传递给出队操作。这个令牌的目的是加速选择一个合适的内部生产者队列，从中尝试出列队列。每个消费者都被分配一个自动递增的偏移量，代表它应该从中执行出队操作的生产者队列的索引。以这种方式，消费者尽可能公平地分配生产者队列。然而，并不是所有的生产者都有相同数量的可用元素，并且一些消费者可能消耗得比其他消费者快。为了解决这个问题，第一个消费者从同一个内部生产者队列中消耗了窗口大小个任务，所有消费者将在下一个出队操作时开始从下一个生产者队列出队。同时，如果消费者的指定队列中没有可用的元素，它将转到具有可用元素的下一个队列。这种简单的启发式算法非常高效，能够将消费者与生产者进行之间的资源竞争尽可能减少，从而带来出队性能提升。

2018人工智能八大趋势展望（转载）

趋势一：大公司先发优势，势在必赢。亚马逊、谷歌、Facebook和IBM将引领人工智能的发展。作为大公司，他们有合适的资源来收集数据，因此有更多的数据可用。

趋势二：趣事儿算法和技术的整合将会发生。行业的参与者之间将发生数据交易，算法和技术将得到巩固。数据的交易以及算法和技术的整合将使人工智能更加有效。

趋势三：众包数据将无比巨大。所有人工智能公司都将追求巨大的数据集，以寻找方法和手段来实现他们对人工智能的雄心。

趋势四：并购将越来越多。2018年，机器学习/人工智能领域的所有小公司都将被大公司收购，主要有两个原因：没有数据集，人工智能就不能独立工作；没有数据的算法毫无用处。

趋势五：AI工具走向民主化，以增加市场份额。

趋势六：人机交互将会改善。虽然机器已经被编程用于语音分析和面部识别，但它还将能够基于你的声音音调识别你的情绪，也就是情感分析，制造业自动化和非消费者方面的解决方案将首先得到改善。

趋势七：人工智能将影响更多垂直领域。2018年保险、法务、公关&媒体、教育、健康将会逐步受到人工智能的影响，为产业发展注入新动力。

趋势八：安全、隐私、伦理和道德问题。人工智能保护伞下的一切，如机器学习和大数据，都很容易遭遇新兴的安全和隐私方面的问题，人工智能的伦理将是2018年的一个主要问题。

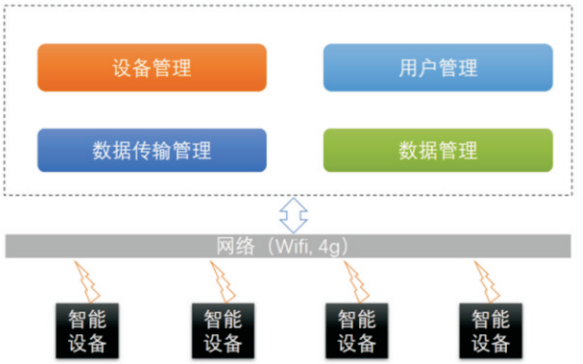
物联网平台架构

（赵智慧 http://blog.csdn.net/u012822903/article/details/79441735）

物联网很久之前就提出了这个概念，现在也是在继续加速发展的过程中，物联网名叫IOT（Internet of Things），学术点来说是一个基于互联网、传统电信网等信息载体，让所有能够被独立寻址的普通物理对象实现互联互通的网络，白话点来说就是一张万物互联的网。它区别于我们熟悉的互联网：互联网连接人与人、人与物、人与信息，而物联网连接物与物。现从学习中总结出来的建构在云端的物联网平台基本架构进行分享。

物联网平台，应该是基于现在的互联网，通讯技术来建构，而不依赖与特定的硬件模块，用户可以基于自身的设备技术架构，简单轻松接入物联网。下图是物联网的核心架构：

在物联网中存在4大核心模块，那就是设备管理，用户管理，数据传输管理，数据管理，只有具备了这四大核心模块，才能认为是一个完整的物联网平台，而所有其他的功能模块都是基于此四大功能模块的延展。



1. 设备管理

设备类型管理：定义设备的类型，此功能一般由设备的制造商来定义，一种设备类型最重要的是关联到一套独有的数据解析方法，数据的存储方法，已经设备规格等数据，也只有设备的制造商才可以编辑有关设备类型的数据，而设备的使用者只能浏览设备类型的相关信息。

设备管理：设备管理定义设备相关信息，每个设备必须定义其设备类型，设备类型有使用者属性，设备在完成销售，并被使用者激活后设备就属于设备使用者了，这时候设备使用者对设备有完全的控制权，可以控制设备的哪些数据可以被制造商查看，可以被哪些用户查看等权限。

2. 用户管理

组织管理：在物联网平台中一个很重要的观念就是组织，所有的设备，用户，数据都是基于组织的管理的，设备制造商是一个组织，设备的使用者是一个组织，家庭都可以是一个组织。

用户管理：用户是基于一个组织下的人员构成，每个组织下面都有管理员角色，管理员可以为其服务的组织添加不通的用户，并分配每个用户不同的权限。一个用户也可以属于多个不同的组织，并且扮演不同的组织。

用户组：一组用户，也是基于组织的用户组管理，同一用户组的用户拥有相同的权限

权限管理：同样是基于组织的权限管理，主要是针对对象级别的权限细分，如设备的浏览权限，可以控制每个用户是否看到这个设备；设备数据浏览权限定义是否可以查看设备的运行数据。

3. 数据传输管理

3.1 基本格式

数据传输管理，定义针对一类型设备的数

据传输协议，基本格式是：

88888888@01@322t12543222213555

每一个设备有厂商唯一的序列号，因为每个制造商有自己的编码格式，固此序列号没有固定格式。

命令码，为此条数据的作用，比如是上传数据，或者服务器下发给设备的命令等，一般采用2位数字编码00~99。

数据，此部分是此条报文，所包含的数据部分，每个协议可以定义不同的解析方式，比如服务器在收到数据包后，会根据预先定义好的解析方式解析数据字段，并按照规则存储。

3.2 数据解析定义

每种设备类型可以定义多条命令，每个命令都有自己不同的解析方式，组织的管理员可以为自己的设备类型定义解析方式

服务器接收到数据后，会自动根据预先定义的解析方式解析数据字段

设备开发者要根据在IOT平台定义的数据格式，自行开发自己设备的解析代码

数据字段都按照HEX方式收发

3.3 数据的存储

存储要支持分布式架构，可以为每个设备定义不同的存储位置，在diego iot中数据存储使用mysql数据库，实现不同的设备存储在不同的mysql数据库中

每条数据定义生命周期，在生命结束后，系统将自动删除

4. 数据的管理

权限管理，数据的权限在物联网平台中是至关重要，数据属于谁是一个非常重要的概念，只有设备的拥有者才能定义数据可以给谁看

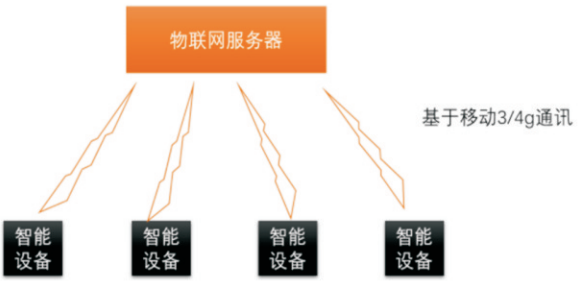
大数据，物联网数据本身就是海量的数据，我们可以借助一些开源的大数据平台来实现数据的可视化分析，只有经过分析的数据才是有价值的数据

数据的导出，用户可以导出数据到本地做分析

5. 网络通讯

现在所有的云端的物联网平台和设备之间的通讯，本质上都是建构在TCP/IP协议之上的，只是对数据包的再封装而已，基于此我们可以是用wifi，4g来实现设备和云平台的通讯，不过设备与设备之间的通讯，可以有wifi，Bluetooth，zigbee等，下面介绍几种常用的通讯架构。

5.1 基于移动3/4G通讯



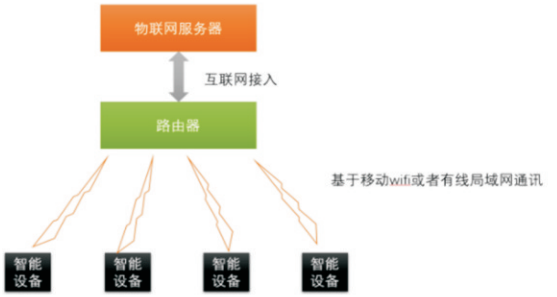
此架构是最简单的架构，设备就如同我们的手机，基于移动通讯来上网，其主要需要考虑如下几点：

每个设备都需要一个SIM卡，可以到移动服务器商办理专门针对物联网的SIM卡。

数据流量问题，这种架构完全是走数据流量，如果有视频数据，将会产生比较大的流量费用，这都是要考虑的。

通讯质量问题，这完全依赖于移动服务商的网络覆盖状况，就如同我们手机一样，在有些环境下是没有信号的，也就没办法收发数据。

5.2 基于WiFi局域网



此架构，适合于所有的物联网设备都是运行在一个局部环境中，设备通过wifi或者有线连接到路由器，而由路由器统一连接的物联网服务器，就如同我们家中装一个wifi路由器上网一样的架构，需要注意的事项：

局域网内的智能设备，是没有公网独立的ip的，只有一个局域网内的ip，带来的问题就是，设备可以直接给物联网服务器发送数据包，而物联网服务器是不能直接给设备发送数据包，就因为设备没有公网独立ip。

功耗问题，对于使用wifi接入的设备，最好不是电池供电，因为wifi的功耗比较大。

干扰问题，如果在大型的厂房部署这种架构，一定要考虑，厂房内是否有强干扰源，如电磁干扰，可以考虑采用工业级的无线路由器，一般抗干扰能力比较强。

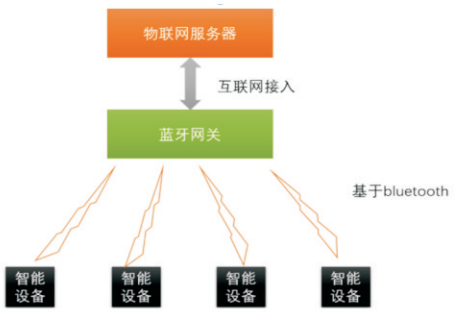
5.3 基于蓝牙通讯

一般的基于蓝牙的物联网，会考虑通过蓝牙网关来部署。

蓝牙由于其点对点的通讯方式，所以要考虑如下问题：

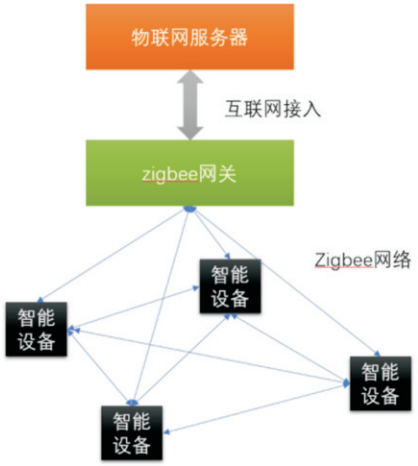
蓝牙网关的容量问题，也就是一个蓝牙网关能接入几个蓝牙设备，这取决于蓝牙网关中使用了多少个蓝牙设备。

蓝牙的配对问题，蓝牙设备直接的通讯都首先配对才能通讯，如果实现自动配对，如果不能自动配对，大规模部署，将是一个很麻烦的事情。



5.4 基于zigbee

ZigBee也是一种流行的组网模式，zigbee本身设计是针对传感器之间的联网，具有非常强的低功耗能力。



zigbee接入网络也依赖于zigbee网关，网关本身也是一个zigbee设备，zigbee设备是自组网的，在使用过程中注意的问题有：

数据量的问题，设备能力和功耗本身是自相矛盾的，由于ZigBee是超低功耗方案，固在通信能力上也是打折扣的，很适合一些传感器数据的采集，如温度湿度，但如果对大数据量的视频类的就不适用了。

6. 总结

本科的时候也曾学习过物联网相关的课程及知识，虽然没能坚持下来往这个方向走，但是一直关注着物联网的发展，特别是现在智能家居，智能城市，智能交通等一系列智能应用的发展，物联网又落地有了新鲜的生命。

比ls快8倍？百万级文件快速遍历的技巧

（转载自（https://mp.weixin.qq.com/s?__biz=MzI5NzkyOTA1Mw==&mid=2247483717&idx=1&sn=faed52d2d0eabe40e98cfa49ee79d100&chksm=ecacd177dbdb58613e46d26cbdad70f18fc9a770d2039244d49e21d785b4f3b11be2f9787b25&mpshare=1&scene=24&srcid=02262DCjORN5XPd9P0x4vHil#rd））

问题背景

在Linux下当我们操作一个文件数较少的目录时，例如执行ls列出当前目录下所有的文件，这个命令可能会瞬间执行完毕，但是当一个目录下有上百万个文件时，执行ls命令会发生什么呢？带着疑问，我们做了如下实验(实验中使用的存储设备为NVMe接口的SSD)：

```
[root@localhost /data1/test_ls]# for i in {1..1000000}; do echo 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' > $i.txt ; done
[root@localhost /data1/test_ls]# time ls -l | wc -l
1000001

real    0m5.802s
user    0m2.544s
sys     0m3.328s
```

可以看到，统计一个包含1000000个小文件的目录下的文件个数花费了将近6秒的时间，那么文件个数多造成ls缓慢的原因是什么呢？且听我们详细分析

原理分析

众所周知，strace是分析系统调用的利器，所以我们用strace来分析在大目录下执行ls命令的结果，其中这样的输出引起了我们的注意：

```
getdents(3, /* 1024 entries */, 32768)  = 32768
getdents(3, /* 1024 entries */, 32768)  = 32768
getdents(3, /* 1024 entries */, 32768)  = 32768
getdents(3, /* 1024 entries */, 32768)  = 32768
brk(0)                                  = 0x12e8000
brk(0x1309000)                          = 0x1309000
getdents(3, /* 1024 entries */, 32768)  = 32768
mremap(0x7f93b6246000, 2461696, 4919296, MREMAP_MAYMOVE) = 0x7f93b5d95000
getdents(3, /* 1024 entries */, 32768)  = 32768
getdents(3, /* 1024 entries */, 32768)  = 32768
getdents(3, /* 1024 entries */, 32768)  = 32768
brk(0)                                  = 0x1309000
brk(0x132a000)                          = 0x132a000
...
```

可以看到，在大目录下执行ls命令会频繁调用getdents这一系统调用，实际上我们通过查看coreutils的ls.c源码可以发现：

```
static void
 print_dir (const char *name, const char *realname)
 {
  register DIR *dirp;
  register struct dirent *next;
  register uintmax_t total_blocks = 0;
  static int first = 1;

  errno = 0;
  dirp = opendir (name);
  ...
  while (1)
   {
      /* Set errno to zero so we can distinguish between a readdir failure
       and when readdir simply finds that there are no more entries.  */
      errno = 0;
      if ((next = readdir (dirp)) == NULL)
     {
      if (errno)
       {
```

```
            /* Save/restore errno across closedir call.  */
            int e = errno;
            closedir (dirp);
            errno = e;

          /* Arrange to give a diagnostic after exiting this loop.  */
            dirp = NULL;
          }
        break;
      }
...
```

ls会首先调用opendir打开一个目录，然后循环调用readdir这个glibc中的函数，直到遇到目录流的结尾，也即读完所有的目录项(dentry)为止。我们首先看一下man page里面对于readdir的定义：

```
struct dirent *readdir(DIR *dirp);
```

readdir返回一个指向dirent结构体的指针，指向目录流dirp中的下一个目录项，所以在print_dir的循环中，每次从目录流中取出一个目录项并赋值给next变量。既然说到目录流(directory stream)，我们顺便看一下glibc中对它的定义：

```
#define __dirstream DIR

struct __dirstream
  {
   int fd;          /* File descriptor.  */

    __libc_lock_define (, lock) /* Mutex lock for this structure.  */

   size_t allocation;      /* Space allocated for the block.  */
   size_t size;        /* Total valid data in the block.  */
   size_t offset;       /* Current offset into the block.  */

   off_t filepos;       /* Position of next entry to read.  */

   /* Directory block.  */
  char data[0] __attribute__ ((aligned (__alignof__ (void*))));
  };
```

从上面的定义中可以看到，目录流实则维护一个buffer，这个buffer的大小由allocation来确定，那么问题来了，allocation值什么时候确定，其实是在opendir过程中确定下来的。opendir的调用路径如下所示：

```
__opendir-->__opendirat-->__alloc_dir
DIR *
internal_function
__alloc_dir (int fd, bool close_fd, int flags, const struct stat64 *statp)
{
  ...
   const size_t default_allocation = (4 * BUFSIZ < sizeof (struct dirent64)
                ? sizeof (struct dirent64) : 4 * BUFSIZ);
  size_t allocation = default_allocation;
  ...
  DIR *dirp = (DIR *) malloc (sizeof (DIR) + allocation);
  ...

  dirp->fd = fd;
  ...
  dirp->allocation = allocation;
  dirp->size = 0;
  dirp->offset = 0;
  dirp->filepos = 0;

  return dirp;
}
```

在__alloc_dir中，会分配sizeof(DIR) + allocation大小的内存空间，最后将allocation赋值给目录流dirp的allocation变量。allocation的默认值通过比较4*BUFSIZ的大小和dirent64结构体的大小(<32768)来确定，BUFSIZ的大小在以下几个头文件中定义：

```
stdio.h:      #define BUFSIZ _IO_BUFSIZ
libio.h:      #define _IO_BUFSIZ _G_BUFSIZ
_G_config.h:  #define _G_BUFSIZ 8192
```

回看一下strace中的输出，getdents第三个参数以及返回值32768就是这么来的。讲完目录流的buffer大小是怎么确定的之后，让我们回到readdir的glibc实现：

```
DIRENT_TYPE *
__READDIR (DIR *dirp)
```

```
{
  DIRENT_TYPE *dp;
  ...
  do
  {
    size_t reclen;
    if (dirp->offset >= dirp->size)
    {
      /* We've emptied out our buffer.  Refill it.  */
      size_t maxread;
      ssize_t bytes;
#ifndef _DIRENT_HAVE_D_RECLEN
    /* Fixed-size struct; must read one at a time (see below). */
      maxread = sizeof *dp;
#else
      maxread = dirp->allocation;
#endif
     bytes = __GETDENTS (dirp->fd, dirp->data, maxread);
      ...
      dirp->size = (size_t) bytes;

      /* Reset the offset into the buffer.  */
      dirp->offset = 0;
    }

   dp = (DIRENT_TYPE *) &dirp->data[dirp->offset];

#ifdef _DIRENT_HAVE_D_RECLEN
    reclen = dp->d_reclen;
#else
    assert (sizeof dp->d_name > 1);
    reclen = sizeof *dp;
    dp->d_name[sizeof dp->d_name] = '\0';
#endif
    dirp->offset += reclen;

#ifdef _DIRENT_HAVE_D_OFF
    dirp->filepos = dp->d_off;
#else
    dirp->filepos += reclen;
#endif

   /* Skip deleted files.  */
    } while (dp->d_ino == 0);
    ...
    return dp;
}
```

这段代码的逻辑还是比较清晰的，首先判断目录流的偏移量有没有超过buffer的大小，如果超过，则说明已经读完缓冲区中的所有内容，需要重新调用getdents读取，getdents一次最多读取32768个字节(有_DIRENT_HAVE_D_ RECLEN定义时为dirp->allocation)，并将读取到的buffer返回给dirp->data，读取到的字节数返回给dirp->size，然后重置偏移量为0。如果没有超过buffer大小，则从dirp->offset开始读，然后将偏移量增加reclen个字节作为下次读取的起点，reclen记录在目录项结构体dirent的d_reclen变量中，表示当前目录项的长度，dirent(DIRENT_ TYPE)这个结构体的定义如下所示：

```
struct dirent
{
  __ino_t d_ino;  /* inode number */
  __off_t d_off;  /* offset to the next dirent */
  unsigned short int d_reclen;  /* length of this record */
  unsigned char d_type;  /* type of file */
  char d_name[256];  /* filename */
};
```

总结一下以上整个过程就是，ls命令每次调用readdir都会从目录流中读取一个目录项，如果目录流的buffer读完，就会重新调用getdents填充这一buffer，下次从新buffer的开头开始读，buffer的默认大小为32K，这也就意味着如果一个目录下有大量的目录项（目录项的总大小可以通过ls -dl查看），则执行ls命令时将会频繁地调用getdents，导致目录下的文件数越多时ls的执行时间越长

解决方法

既然glibc中readdir的buffer大小我们没法控制，何不绕过readdir直接调用getdents，在这个

系统调用中我们可以直接控制buffer的大小，以下就是一个简单的例子listdir.c：

```
#define _GNU_SOURCE
#include <dirent.h>     /* Defines DT_* constants */
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/syscall.h>

#define handle_error(msg) \
      do { perror(msg); exit(EXIT_FAILURE); } while (0)

struct linux_dirent {
   long          d_ino;
   off_t         d_off;
   unsigned short d_reclen;
   char          d_name[];
};

#define BUF_SIZE 1024*1024*5

int
main(int argc, char *argv[])
{
   int fd, nread;
   char buf[BUF_SIZE];
   struct linux_dirent *d;
   int bpos;
   char d_type;

fd=open(argc>1?argv[1]:".",O_RDONLY|O_DIRECTORY);
   if (fd == -1)
      handle_error("open");

   for ( ; ; ) {
      nread = syscall(SYS_getdents, fd, buf, BUF_SIZE);
      if (nread == -1)
         handle_error("getdents");
      if (nread == 0)
         break;

      printf("-------------- nread=%d --------------\n", nread);
      printf("inode#   file type  d_reclen  d_off  d_name\n");
      for (bpos = 0; bpos < nread;) {
         d = (struct linux_dirent *) (buf + bpos);
         printf("%8ld  ", d->d_ino);
         d_type = *(buf + bpos + d->d_reclen - 1);
         printf("%-10s ", (d_type == DT_REG) ?  "regular" :
                          (d_type == DT_DIR) ?  "directory"
:
                          (d_type == DT_FIFO) ? "FIFO" :
                          (d_type == DT_SOCK) ? "socket" :
                          (d_type == DT_LNK) ?  "symlink"
:
                          (d_type == DT_BLK) ?  "block dev" :
                          (d_type == DT_CHR) ?  "char dev" : "???");
         printf("%4d %10lld  %s\n", d->d_reclen,
                 (long long) d->d_off, d->d_name);
         bpos += d->d_reclen;
      }
   }

   exit(EXIT_SUCCESS);
}
```

在这段代码中，我们将getdents的buffer大小设置为5M，编译执行这代码，我们得到如下结果：

```
[root@localhost /data1]# time ./listdir test_rm | wc -l
1000016

real    0m0.755s
user    0m0.432s
sys     0m0.320s
```

统计目录中的文件数由默认的5.802s缩短为0.755s，可以看到提升还是较为明显的。

总结

其实不止是ls命令，其他一些命令如rm-r等的实现中都会用到glibc中的readdir函数，所以如果遇到操作百万级文件的大目录这种场景(当然实践中不提倡一个目录下放这么多文件)，不妨直接调用getdents并加上自己的一些逻辑，这样就可以在实现标准命令功能的基础上，还能获得其不具备的性能提升。

浅谈GCC编译优化

（刘本熙 http://sec-lbx.tk/2017/07/15/%E6%B5%85%E8%B0%88GCC%E7%BC%96%E8%AF%91%E4%BC%98%E5%8C%96/）

GCC Pass

在GCC完成词法、语法分析，并获得源代码对应的抽象语法树AST之后，会将其转换为对应的GIMPLE序列。随后，GCC会对GIMPLE中间表示进行一系列的处理，包括GIMPLE的低级化、优化、RTL生成等、RTL优化等。为了方便管理，GCC采用了一种称为Pass的组织形式，把它们分成一个个的处理过程，把每个输出结果作为下一个处理过程的输入。

GCC中，Pass可以分为4类，GIMPLE_PASS，RTL_PASS，SIMPLE_IPA_PASS，IPA_PASS。其中除了RTL_PASS之外，处理对象都是用GIMPLE中间语言表示的。名字包含IPA的两类Pass的功能主要是过程分析，也即函数间调用和传递。Pass的执行是以链表的形式组织的，每个Pass还可以包含子Pass，并且以函数为执行的基本单元。

如果把GCC中的Pass全部打印出来，可以看到数量有数百个，因此这里只选择几个代表性的pass研究一下。值得说明的是，在GCC 4.6之后，增加了插件功能，它同样是通过Pass的形式来进行管理，这使得我们自定义处理过程变得很容易。

去除无用表达式

该Pass从GIMPLE序列中进行搜索，从中删除死代码。这些死代码主要包括：

（1）空语句

（2）无意义的语句块、条件表达式

（3）目标地址就是下一条段GOTO表达式

（4）无意义的malloc、free

（5）…

在GCC 4.4版本中，这个Pass是pass_remove_useless_stmts，现在这个函数已经被移除了；其功能被拆分到了别的函数中。但gcc的文档中还是说明了这个Pass。

类似这样的结构，在这个Pass中都会被优化掉：

```
if(1)      //无意义的条件表达式

goto next;   //无意义的goto
next:
  //do sth
```

那么该Pass是如何进行判断的呢？我们来看看GCC 5.4版本中，类似函数的实现。在gcc/tree-ssa-dce.c当中，有这样一个函数eliminate_unnecessary_stmts()。这个函数会(逆序的)逐个遍历表达式，防止某些定义或名字被释放：

```
for(gsi = gsi_last_bb (bb); !gsi_end_p (gsi); gsi = psi){
  stmt = gsi_stmt (gsi);
  ...
  if (gimple_call_with_bounds_p (stmt)) {
   //如果有必要删除，那么会首先利用这个函数设置
    gimple_set_plt(...)
  }
  //对于需要删除的表达式进行删除
  if (!gimple_plf (stmt, STMT_NECESSARY))
  ...
}
```

如果进行了删除，那么这个函数还会对一部分CFG进行修改，并删除不可达的基本块。

控制流图的构造

在GCC中，控制流图指的是函数内的控制流图，这和我平时在文章中看到的“CFG”是不同的。GCC中，CFG的节点为基本块，而边则是基本块之间的跳转关系。pass_build_cfg对函数对

GIMPLE序列进行分析，完成基本块的划分，并且根据GIMPLE语义来构造基本块之间的跳转关系。

控制流图的构造，主要是在build_gimple_cfg()当中完成的。

```
static void
build_gimgple_cfg (gimple_seq seq)
{
  gimple_register_cfg_hooks();//注册这个函数，要构造cfg了
  init_empty_tree_cfg();//先构造一个空的cfg
  make_blocks(seq);//具体的基本块构造过程
  ...
  make_edges();//创建基本块的边
}

static void
make_blocks (gimple_seq seq)
{
  while (!gsi_end_p (i))//遍历seq中的每一条
  {
    ...
     gimple_set_bb (stmt, bb);//把当前的语句添加到基本块
    if(stmt_ends_bb_p(stmt))
    {
    //如果stmt是基本块的终结，就进行处理，并且在下一次创建一个新的基本块
    ...
    }
  }
}
```

在基本块创建之后，就可以根据基本块，构造基本块中的边，对于函数中的不同情况，生成GCC会不同的边，并且创建对应的label。

```
static void
make_edges (basic_block min, basic_block max, int updata_p)
{
  ...
  //处理min和max之间的基本块
  FOR_BB_BETWEEN (bb, min, max->next_bb, next_bb)
  {
    //在5.4版本中，是直接对RTL进行处理
    rtx_insn *insn;
    insn = BB_END(bb);
    //如果为JUMP指令，还包括了条件跳转
    if(code == JUMP_INSN)
    ...
    //如果为CALL指令，还包括了异常处理
        else if(code == CALL_INSN || cfun->can_throw_non_call_exceptions)
    ...
  }
}
```

指令调度

前面介绍了GCC当中，GIMPLE表达式的部分优化过程，现在我们来说说RTL中的优化过程。指令调度是对当前函数中的insn序列进行重新排列，从而更充分地利用目标机器的硬件资源，提高指令执行的效率。指令调度主要考虑有数据的依赖关系、控制相关性、结构相关、指令延迟和指令代价等，通常指令调度与目标架构上的流水线有关。在GCC中，指令调度分为两个部分：寄存器分配之前的pass_sched 和寄存器分配之后的pass_sched2。

不过，在这里应该先阐述一下：为什么要进行指令调度？这是由硬件的特性决定的：

（1）指令拥有不同的执行时间。对于指令来说，其实latency（指令开始执行后，多少时钟数能够为其他指令提供数据）、throughtput（指令完成计算需要的时钟数）、μ ops（指令包含的微操作数）都是不同的。

（2）指令流水线能够并行执行指令的微操作，从而在等待时完成一部分工作。

（3）超标量处理器，比如现在的i7处理器，能够同时运行多条指令，虽然硬件完成了一部分调度工作，但存在一个窗口的问题，依然需要编译器进行调度。

正因如此，编译器需要对指令进行调度，来提高运行时的效率。指令在调度时，存在着以下限制：

（1）数据依赖关系。包括flow（一条指令定

义的值和寄存器在后面的指令用到）、anti（一条指令用的值或者寄存器会被后面的指令修改）、output（一条指令定义的值或寄存器在后面会被改；

（2）循环限制，多次迭代之间的关系，循环内部的依赖关系等；

（3）硬件资源的限制，例如能够同时并行的指令数；

这些限制必须作为指令调度的输入被考虑到，通过这些依赖关系，可以把所有的指令，构成一个依赖关系图。

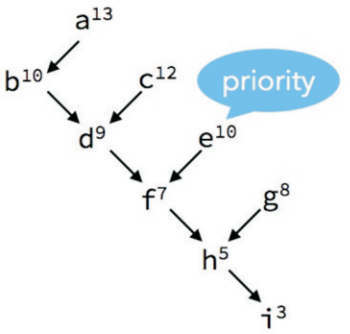
指令调度是一个典型的NP-hard问题，而表调度算法就是一种典型的启发式算法。通常其调度的单元在基本块的内部，并把数据依赖关系图作为输入，并计算出指令的优先级。优先级的计算涉及：依赖图的根到节点的路径长度、某个节点的后继数量、节点的latency等。

随后，保持两个表：ready表中保存了能够不延时执行的指令，它们根据优先级从高到低排列；active表包括了所有正在执行的指令。在算法的每一步：

（1）从ready表中取出优先级最高的指令进行调度，将它移到active队列当中，它会在队列中呆上一个时延的时间；

（2）更新指令的依赖关系，把新的就绪指令插入ready表中。这里我们用一个例子来说明：

在每一轮，都取了ready队列中，优先级最高的指令，如果active中的指令没有执行完，导致依赖关系无法满足，那么ready可能是空的，此时就继续下一轮调度。



cycle	ready	active
1	a13,c12,e10,g8	a
2	c12,e10,g8	a,c
3	e10,g8	a,c,e
4	b10,g8	b,c,e
5	d9,g8	d,e
6	g8	d,g
7	f7	f,g
8		f,g
9	h5	h
10		h
11	i3	I

GCC中默认的指令调度算法是表调度算法。其处理过程在/gcc/sched-rgn.c中定义。schedule_insns()是调度的主要函数，它会执行两次，分别在：

（1）寄存器分配之前，在pass_sched中调用，实现以区域为调度范围的指令调度；

（2）寄存器分配之后，在pass_sched2中调用，通常只在每个基本块内部进行指令调度；

```
schedule_insns()
{
  int rgn;
  //如果没有包含代码的基本块，直接退出
    if (n_basic_blocks_for_fn (cfun) == NUM_
FIXED_BLOCKS)
    return;
  //初始化指令调度信息
  rgn_setup_common_sched_info ();
  //初始表调度的基本信息
   rgn_setup_sched_infos ();
  //haifa表调度的数据结构初始化，进行数据流分析
   haifa_sched_init ();
   //初始化区域信息，将CFG的区域提取出来
   sched_rgn_init (reload_completed);

   bitmap_initialize (&not_in_df, 0);
   bitmap_clear (&not_in_df);

   //对每个区域内的基本块进行调度
   for (rgn = 0; rgn < nr_regions; rgn++)
  if (dbg_cnt (sched_region))
```

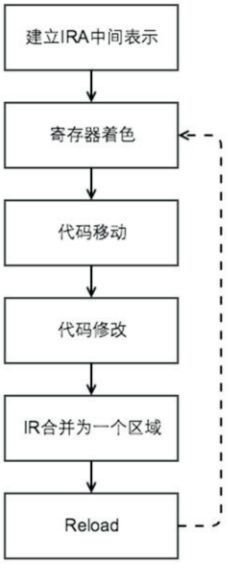
```
        schedule_region (rgn);

    //清除
    sched_rgn_finish ();
    bitmap_clear (&not_in_df);
    haifa_sched_finish ();
}
```

统一寄存器分配

我们知道，RTL在生成和处理的过程中，大量地使用了虚拟寄存器，那么这些虚拟寄存器在转换为目标机器的汇编码之前，需要映射到目标机器中的物理寄存器上，该过程即为寄存器分配。那么合理地分配、使用物理寄存器，是GCC后端及其重要的一个任务。

GCC中采用的是一种把寄存器合并、寄存器生存范围划分、寄存器优选、代码生成和染色整合在一起的算法，所以被称为统一寄存器分配。整个统一寄存器分配是这样一个过程：



第一步是建立IRA的中间表示，其代码在ira-build.c中，入口函数为ira-build()；

第二步是寄存器的着色。这一步按照自顶向下的顺序，遍历每个区域，对于区域中的分配元进行着色，由ira_color()函数完成；

第三步是代码移动，解决父子区域中，寄存器被spill/store的问题，如果子区域需要spill到内存，那么父区域也可以直接spill到内存，这样就不用多次进行移入内存/从内存取值到寄存器的过程；

第四步是代码修改，在着色处理后，父子区域中，相同虚拟寄存器可能会分配到不同的物理寄存器或者存储位置，而某个区域内部也可能会对同一个虚拟寄存器分配不同的物理寄存器。此时IRA就分配新的虚拟寄存器进行取代，并且在区域边界实现新的分配元并进行交换；

第五步：将所有区域的分配元合并到一个区域中；

第六步：尝试对spill操作的分配元分配物理寄存器；

第七步：Reload Pass。前面的过程可能会引入新的代码、虚拟寄存器，产生不能满足模版约束的情况，reload处理这些情况，因此要重新进行第二步开始的操作，直到不再产生新的代码。

这里只说明一下寄存器着色的过程。在第一步中，我们能够根据每个虚拟寄存器的生存范围，能够确定它们的冲突关系：生存范围有冲突的虚拟寄存器，不能被分配到同一个硬件寄存器当中去（ira_build_conflicts()的工作）。而寄存器着色，就是根据虚拟寄存器之间的冲突关系图，进行图的着色处理。在这个图中，每个节点代表需要着色的虚拟寄存器，而每条边都定义了冲突关系，也就是这两个虚拟寄存器不能着相同的颜色。那么如果有N个寄存器可以分配，就相当于用N个颜色来着色；如果N不够大，那么就需要物理内存来保存虚拟寄存器的值。

小结

这里我们从几个点出发，简单介绍了几个有代表性的编译优化过程，实际上编译器的优化会涉及到数以百计的pass，它的过程是及其复杂的。

2017年度实验室十大新闻揭晓

吴　未

2017年3月3日晚，在一年一度的实验室新春联欢会上，2017年度实验室十大新闻揭晓，“金海教授获2017年度‘网络安全优秀教师奖’”等十条新闻入选：

1、金海教授获2017年度“网络安全优秀教师奖”。

2、刘方明教授获国家自然科学基金委优秀青年科学基金奖。

3、廖小飞教授获中国计算机学会–IEEE Cs青年科学家奖。

4、实验室成功举办首届华中科技大学网络空间安全喻园青年科学家论坛。

5、金海教授等完成的教学成果“以并行计算系统能力培养为导向的本科课程体系的创新与实践”获湖北省教学成果二等奖。

6、学院党委书记吴涛为实验室全体师生上了一堂以“学术理想与家国情怀”为主题的专题党课。

7、实验室引进德国哥廷根大学毕业的黄宏博士。

8、实验室今年在学术论文方面继续进步，共发表论文总数突破100篇。

9、张伟、牛轶佩，艾明等11位同学获得国家奖学金，张振，张晓冬，杨莹等 72人次获得校级以上各种奖励。

10、2017年实验室获得了3项国际专利授权。

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.con

实验室老师参加计算机学院举办的首期“TIME”教授沙龙（转载）

3月15日下午，计算机学院在八号楼二楼多功能厅举办首期“TIME教授沙龙”，共邀请到学院9名教授作PPT分享报告。沙龙由青年千人计划入选者陈敏教授主持，近60名教师参加。

“TIME教授沙龙”是计算机学院打造的以教授为主导的交流互动平台，拟每年按季度举办多期，每位教授都将走上讲台分享。该沙龙紧紧围绕Teamwork（协作）、Innovation（创新）、Morality（修德）、Exploration（开拓）的学院“TIME”文化核心精神，旨在发挥教授引领、示范、启发的作用，指导和带领全院教师共同干事创业，共同推进一流学科建设。

沙龙的首位分享者金海教授以“我看TIME”为题，从Teamwork、Innovation、Morality、Exploration四个方面，分享了团队在科研创新、人才培养、文化建设、管理经验、未来发展思路等方面的情况；千人计划入选者宋恩民教授分享了团队在医学图像处理领域的最新进展和未来发展思路；陈敏

实验室学术委员会第九次会议召开

郑　然

2018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九次会议在北京召开。实验室学术委员会成员深圳大学陈国良院士、北京邮电大学方滨兴院士、北京理工大学梅宏院士、清华大学郑纬民教授、中科院软件所戴国忠研究员、中科院信工所孟丹研究员、西安电子科技大学杨宗凯教授、国防科技大学王怀民教授、西北工业大学周兴社教授、浙江大学鲍虎军教授、武汉大学何炎祥教授等参会指导工作。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老师二十余人参加了会议。

本次会议由学术委员会委员深圳大学陈国良院士主持，金海教授作了实验室2017年度工作汇报，就实验室2017年度工作进展、主要研究方向与研究成果、师资队伍建设、人才培养、开放交流、运行管理等情况进行了详细介绍。华强胜副教授作了题为“分布式系统中低复杂度图算法研究”的学术报告。

会上，学术委员会的专家们对实验室的工作情况和学术报告内容进行了充分的交流和讨论，充分肯定了实验室这一年以来的工作，一致认为实验室发展稳定，各项工作取得了可喜的进展。同时，专家们对突出科研成果影响力和辐射面、服务社会、人才培养与引进、成果转化等提出了宝贵的建议和殷切的希望。

郑　然
博士，副教授
主要从事分布式计算、云计算、高性能计算及应用等的研究。
E-mail：zhraner@hust.edu.cn

接上页

分享了自己入职以来的工作进展情况和学术论文撰写经验；李国徽教授、李瑞轩教授分享了教书育人、投身科研的成长发展经历和经验；朱虹教授分享了数据库相关领域的研究工作和团队建设中的压力与困惑；王芳教授结合科研论文成果分享了开展科学研究的经历和经验；青年教授石宣化教授和余辰教授分享了围绕学生特点和兴趣进行特色培养方面的心得体会。教授们的报告融合学科特点和个人经历体验，有的庞大宏观，有的细腻亲切，赢得了参加沙龙活动的教师的阵阵掌声。

学院党委书记吴涛在总结讲话中指出，“TIME”文化寓意深刻，“TIME”文化既是学院精神的历史传承，也体现了学科的时代特征。计算机科学与技术日新月异的进步与发展，要求我们以“只争朝夕”的紧迫感追求卓越，与“时间”竞赛跑；要求我们深刻理解计算机科学与技术与信息时代发展的紧密联系和重要地位，为“新时代”做贡献。学院发展、学科发展、教师成长发展都离不开教授的引领，特别是教授治学精神、研究方法、育人理念的引领，教授沙龙提供了交流互动的平台，希望老师们加强学习，在良好的学术氛围中快速成长，与时代同步伐，与祖国共命运，加快建成世界一流计算机学科，为支撑我国信息产业自主可控，实现中华民族伟大复兴中国梦贡献力量。

（转自 http://cs.hust.edu.cn/info/1094/1762.htm）

LightNVM: The Linux Open-Channel SSD Subsystem

樊 浩 推荐

“LightNVM: The Linux Open-Channel SSD Subsystem”是国际顶级会议FAST2017录用的一篇文章。该会议于2014年的2月在美国加州圣克拉拉召开。这篇文章由CNEX实验室的Matias Bjørling, Javier Gonzalez以及来自来哥本哈根大学的Philippe Bonnet合作完成。文章为open-channel SSD设计了内核模块LightNVM。LightNVM是业界第一个开源的open-channel SSD项目，它可以实现在系统层对SSD的管理。实验表明，LgihtNVM控制的open-channel SSD相对于最先进的NVMe SSD有更好的性能，并且可以更好的实现根据应用实现定制化的FTL。

文章主要进行了三方面的工作，下面是详细介绍：

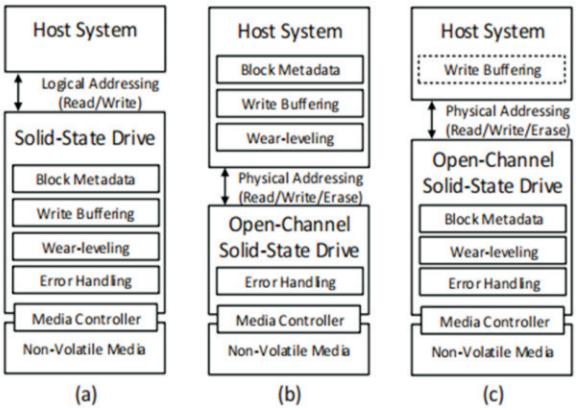


图1 不同类型SSD管理模块分布。
(a)为传统的块设备级SSD，(b)为本文采用的结构，
(c)为将来SSD的设计架构

文章的第一项工作是讨论了open-channel SSD管理的特点。首先文章讨论了NAND Flash的特点并分析了open-channel SSD管理NAND Flash时缓存、容错的时机以及这些功能实现的位置（主机上，SSD上）。如图1，文章讨论了三种SSD管理模式，并选择图1.b作为实现方案。

文章的第二项工作是设计了PPA(Physical Page Address) I/O接口。来自系统层的逻辑地址（LBA）将转换成PPA地址进行读、写操作，以方便访问SSD。另外，PPA可以排除SSD中的坏块。

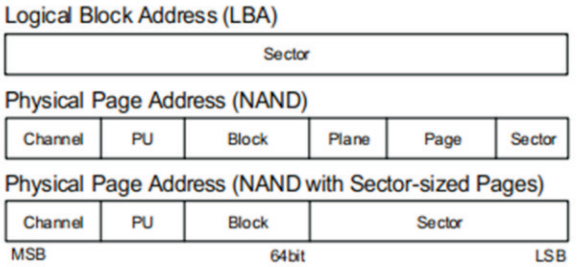


图2. LBA和PPA的对应关系

文章的第三项工作是在内核层为open-channel SSD设计了子系统LightNVM。LightNVM的层次结构如图3所示。

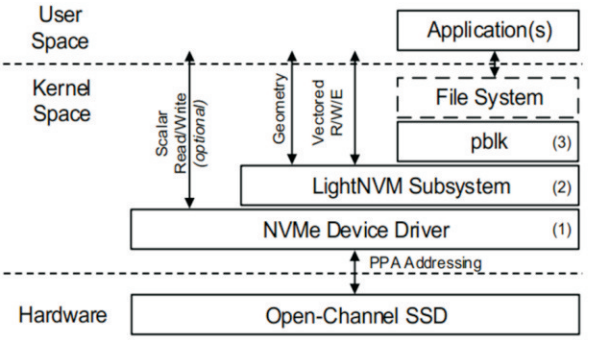


图3. LightNVM的层次机构

NVEe Device Driver 使内核模块通过PPA接口访问open-channel SSD。这层驱动可以将SSD设备作为一个普通的设备提供给用户态。应用可以通过I/O控制码与设备交互。LightNVM Subsystem 是位于NVEe Device Driver之上的实例，这个实例使得内核能够得通过sysfs和nvm_fs得到设备的信息。另外，该实例还利用blk-mq实现了矢量I/O。Pblk是一个高层的I/O接口，文件系统层的I/O和应用定义的标准化I/O都可以实现交互。Pblk在内核层为SSD提供了地址映射、容错以及垃圾回收。

文章通过三方面的实验验证了LightNVM的效果：

首先，在open-channel SSD上部署pblk并计算其带来的开销。pblk的部署会给open-channel SSD的读请求额外带来1%的CPU开销和18%的延迟，给写请求带来额外3%的CPU开销和45% 的延迟。额外开销和延迟的原因是L2P表和缓存的查询。

然后，将pblk接口的open-channel SSD与最先进的NVMe SSD进行比较。如图4-5及表1，相比于NVMe SSD, open-channel SSD能够提供更低的延迟和更高的带宽。

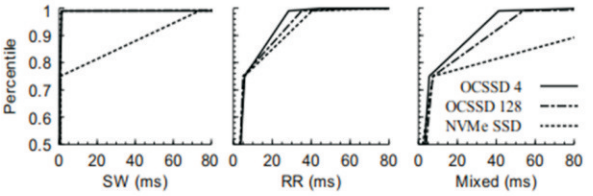


图4. RocksDB中顺序、随机以及混合负载下OCSSD以及NVMeSSD的延迟

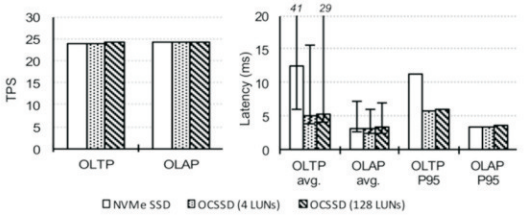


图5. OLTP和OLAP中OCSSD和NVMe SSD情况下每秒钟处理事件数

表1. Rocks下顺序写、随机读以及混合负载在OCSSD和NVMe上的带宽

	NVMe SSD	OCSSD 128	OCSSD 4
SW	276	396	80
RR	5064	5819	5319
Mixed	2208	3897	4825

最后，文章验证pblk精确到SSD存储颗粒的写可以用来优化I/O调度，实现满足预期延迟。如图6所示，通过测试有背景写请求存在时，读请求的延迟来观察open-channel SSD 能否保证I/O延迟。当背景写请求比例提高时，open-channel SSD的读请求受的影响很小。而NVMe接口的SSD请求大量读请求出现高延迟。

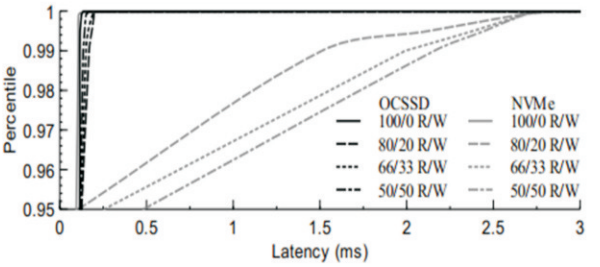


图6. 当存在大量背景写请求时读请求的延迟

文章介绍的LightNVM是业界首个openchannel-ssd的开源内核子系统，不仅极大地挖掘了SSD的潜在性能，也为针对应用定制FTL层提供了可能。

樊　浩
2014级博士研究生
研究方向：I/O虚拟化
Email：u201014452@hust.edu.cn

NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories

周 放 推荐

“NOVA: A Log-structured File System for Hybrid Volatile/Non-volatile Main Memories”是国际顶级会议FAST'16上的一篇论文，其作者Jian Xu来自加州大学圣地亚哥分校。NOVA是一个基于混合易失和非易失内存的日志结构、POSIX文件系统，它扩展LFS并充分利用NVMM的优势。设计NOVA基于以下三个观察。第一，日志支持原子更新容易在NVMM上实现，但是对于查找操作不高效；第二，日志清理的复杂性主要来自需要提供连续空闲的存储空间，但是在NVMM中没有必要，因为NVMM中随机访问是便宜的；第三，对于磁盘而言，使用单个日志是合理的，但是这个限制了并发性。由于NVMMs支持快速高并发随机访问，使用多个日志并不影响性能。

因此NOVA保存日志和文件数据到NVMM中，并且通过在DRAM中构建radix树结构保存索引以便加速查找操作。NOVA中每个索引结点（inode）均有自己的日志（log），允许跨文件并发更新并且无需同步化操作。该结构使得文件访问高并发，并且在故障恢复期间，可以同时重新执行多个日志。NOVA使用logging和轻量级的journaling来执行复杂的原子更新。为了原子地写数据到一个log，NOVA首先将数据追加到log，然后原子地更新该log tail指针以便提交该更新，这避免了journaling文件系统的双倍写开销以及shadow paging系统的迭代更新问题。因为NVMM支持快速并发随机访问，所以采用链表机制是可行的。因此NOVA使用4KB NVMM 页的单链表实现索引结点log。

NOVA采用非顺序日志（non-sequential log）存储，提供如下三个好处：1. 分配日志空间容易，原因是不需要分配大量的连续区域用于log；2. NOVA可以执行细粒度，页大小粒度的日志清理；3. 回收仅包含过时条目的日志页只需要几个指针分配。NOVA中索引节点的日志不包含文件数据，相反NOVA采用写时复制技术用于修改页面（modified pages），结合追加元数据的写到日志（log）中。对于文件数据，采用写时复制（COW）原因如下：1. 这样日志可以更短，从而加快恢复过程；2. 使得垃圾回收更简单和高效，因为NOVA从不从日志中拷贝文件数据以便回收日志页（log page）；3. 回收过时页以及分配新的数据页均是简单的，因为只需要从DRAM空闲列表（free lists）中添加和移除页；4. 能够立刻回收过时的数据页，即使是在高的写负载和高NVMM使用情况下。

Jian Xu在设计NOVA时，面临如下挑战：1. 实现硬件潜在的性能；2. 写重新排序和一致性的影响；3. 提供原子性

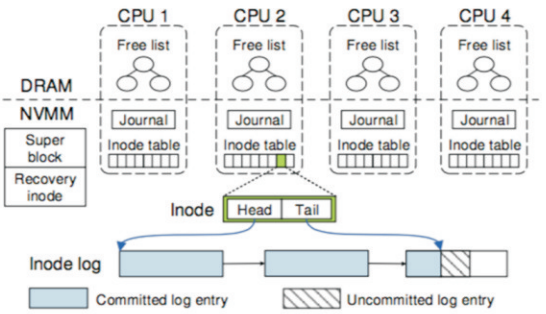


图1 NOVA数据结构布局

如上图所示，NOVA将NVMM分成四个部分：超级块（superblock）和恢复结点（recovery node），journals以及日志/数据页（log/data pages）。该超级块包含文件系统的全局信息，恢复结点存储恢复信息以便加速NOVA在正常关机情况下的重新挂载（remount），索引节点表（inode tables）包含索引节点，journals提供目录操作的原子性并且剩余区域包含NVMM日志和数据页。为了获得好的扩展性，NOVA在每个CPU中保持一个索引结点表、journal和NVMM空闲页列表（free page list）以避免全局锁和扩展性瓶颈。

NOVA采用了Filebench工作负载包括fileserver、webproxy、webserver和varmail去评估NOVA的应用程序级性能。经实验验证NOVA获得了高性能并且有强数据一致性保证，如图2所示。

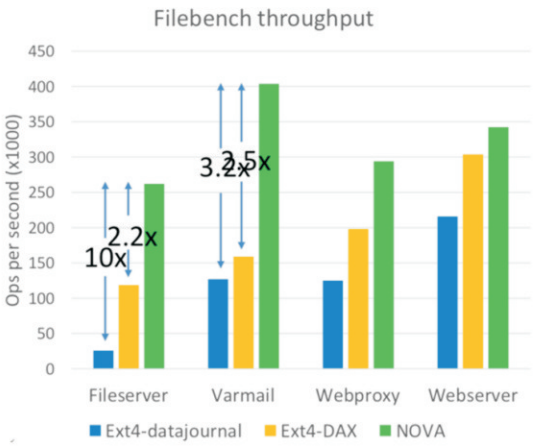


图2

NOVA的性能随着运行时间的增加保持稳定，原因是在长期的运行过程中，Fast GC回收了主要的过时页（stale pages），如图3所示。

NOVA使用DRAM去维持NVMM空闲页列表（free page lists），所以在文件系统挂载时必须重建。NOVA通过延迟地重建索引节点信息，保持短的日志并且执行并行的日志扫描来加速恢复过程。为了测量恢复开销，采用了表4中的三种工作负载。每种工作负载代表了文件系统的一种不同的使用情况：Videoserver包含少量的大尺寸请求的大文件访问，mailserver包含了大量的小文件并且请求大小比较小，而Filesever介于两者之间。

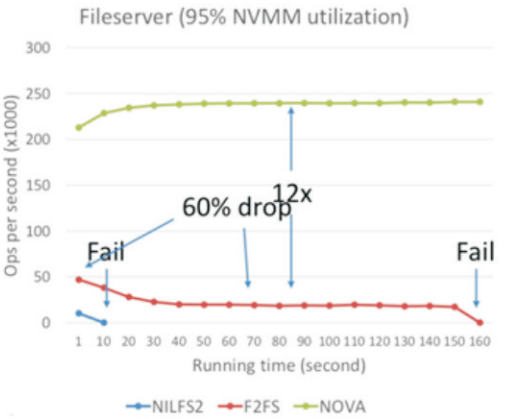


图3

正常关机，NOVA在1.2ms时间恢复文件系统，原因是NOVA不需要扫描索引结点日志。在系统故障后，NOVA的恢复时间随着索引结点数量的增大而增长，文件越多，文件日志越多并且随着IO操作的进行文件越来越碎片化以致文件日志变得越来越长。基于STT-RAM上比基于PCM的恢复快是因为NOVA需要读日志以便重构NVMM空闲页列表，而PCM有比STT-RAM更高的读延迟。在PCM和STT-RAM中，NOVA能够在116ms时间内恢复50GB数据，恢复带宽超过400GB/s。

周　放
2014级博士生
研究方向：异构内存管理
Email：flame@hust.edu.cn

Designing scalable FPGA architectures using high-level synthesis

陈绍鹏　推荐

“Designing scalable FPGA architectures using high-level synthesis”是在计算机编程语言国际会议PPoPP 2018研讨会上发表的一篇文章，该会议于2018年2月在奥地利维也纳市（Vienna）举行。文章设计并提出一种高效可扩展的FPGA体系结构，通过高级综合（high-level synthesis, HLS）软件方法实现FPGA程序算法的高度可并行化，使之能很好地适用于大规模高性能计算（high performance computing, HPC）领域。

FPGA的高性能及低功耗特性使其极有潜力成为大规模高性能计算系统核心组件。虽然FPGAs已经开始部署到一些云及数据中心场景（如微软的AI云，亚马逊云等），但是其在HPC领域的应用依然受限。其主要原因有两点：（1）FPGA硬件描述语言（hardware description languages, HDLs）对HPC研究者而言，学习门槛过高，编程过于繁琐；（2）FPGA相较于GPU或多核等体系结构有着天然的低带宽，逻辑单元结构单一（如无浮点运算单元）等劣势。随着HPC系统规模不断扩大，其功耗开销成为重要瓶颈，这使得FPGA的低功耗特性尤为可贵。人们希望能够寻求方法弥补FPGA的不足，使之应用于大规模HPC系统，提升性能的同时极大地降低功耗。针对HDLs不足，FPGA提供商和第三方尝试将其抽象层级由传统的寄存器传输级（register transfer level, RTL）提升到由高级语言（如OpenCL，C/C++等）实现的算法级（algorithmic level），进而开发出一系列的HLS工具。针对硬件支持不足，我们需要（a）优化FPGA资源空间利用率，避免可用逻辑单元的闲置浪费；（b）提升FPGA吞吐率，保证在单位时钟周期内尽可能多的逻辑单元处于工作状态。

```
1 void StreamingDataflow(Data_t *...) {
2   #pragma HLS PIPELINE DATAFLOW
3   Stream<Data_t> pipes[D+1];
4   ReadMemory(memory_in, pipes[0]);   // Writes head
5   for (int d = 0; d < D; ++d) {
6     #pragma HLS UNROLL
7     ProcessingElement(pipes[d], pipes[d+1]);
8   }
9   WriteMemory(pipes[D], memory_out); // Reads tail
10 }
```

图 1. streaming dataflow核心算法

文章提出软件方法在HLS过程中优化RTL层的FPGA资源分配及流水吞吐，充分发挥FPGA资源空间高度并行化特性，提高FPGA工作性能。文章提出4个关键优化方法：（1）流水（pipelining）优化，对传统HLS生成的流水调度进行细粒度划分；（2）缓冲（buffering）优化，使用片上缓冲区（on-chip buffering）来减轻内存访问带宽压力，规则化访存形式；（3）平铺（tiling）优化，通过unrolling技术生成平铺调度（tiling scheme），并以此来确定需要划分片上内存大小；（4）流（streaming）优化，使用FIFO streaming方法对生成的展开调度优化处理，以解决内存瓶颈及访问延时问题，其核心算法streaming dataflow实现方法如图1所示。

An Empirical Comparison of Compiler Testing Techniques

黄 冠 推荐

“An Empirical Comparison of Compiler Testing Techniques”是国际顶级会议ICSE'16收录的论文，该论文对目前常用的编译器测试方法进行了总结和对比，并指出了它们的适用环境和使用效果。

编译器测试技术发展至今已经有30年的历史，目前主要通用的编译器测试方法主要有三类，分别是Randomized Differential Testing (随机差异测试)、Different Optimization Levels (不同优化级别测试)、Equivalence Modulo Inputs (等效模式输入测试)。其中随机差异测试是使用两个或者多个规格相同的对比编译器，这些编译器在处理相同输出测试集的时候，应该产生相同的结果，若结果出现差异，利用投票机制，很容易的就可以发现哪些编译器存在漏洞。不同优化级别测试是对比不同优化级别下

接上页

	Type	Stencil	Source	Device	Performance	Frequency	Power	Power efficiency
TPDS'17 [1]	CPU	Heat 2D	C	Xeon E5645	54 GOp/s	2400 MHz	(2 × 80 W)[a]	(0.34 GOp/J)[a]
PPoPP'17 [4]	GPU	Jacobi 2D	CUDA	Titan X	490 GOp/s	1417-1531 MHz	(250 W)[a]	(1.96 GOp/J)[a]
FPT'13 [3]	FPGA	RTM (3D)	Maxeler	MaxGenFD	131 GOp/s	100 MHz	142 W	0.92 GOp/J
TPDS'14 [5]	FPGA	Jacobi 2D	HDL	9× EP3SL150	260 GOp/s	133 MHz	201 W	1.29 GOp/J
TPDS'17 [6]	FPGA	Jacobi 2D	OpenCL	395-D8	238 GOp/s	230 MHz	(75 W)[a]	(3.17 GOp/J)[a]
Ours	FPGA	Jacobi 2D	C++	TUL KU115	228 GOp/s	160 MHz	31 W	7.36 GOp/J

表 1. 功效评估对照表

文章采用2D Jacobi标准迭代算法对自己提出的方法进行了功效评估，并与当前一些主流方法进行了比较，如表1所示。文章提出的方法相较于GPU方法虽然性能下降约50%却有着接近3.8倍的能效提升，同时相较于当前主流FPGA方法（如TPDS'17 OpenCL方法）在性能相当情况下，功耗则缩减约一半。文章还对FPGA各资源的使用率及性能进行比较，如图2所示。从图中我们可以看出，随着程序计算量级的逐渐增大，FPGA各资源组件（LUT，DSP，RAM）的使用率趋于均衡。由此可以看出文章提出并设计的方法对于HPC系统具有较好的适应性。

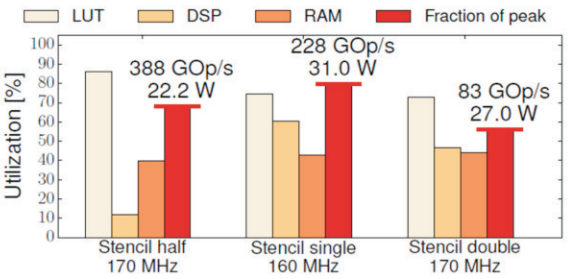


图 2. 资源使用率与性能比较图

陈绍鹏
2015级博士研究生
研究方向：编程语言
Email：shpchen@hust.edu.cn

的编译器编译结果，例如GCC编译器有–O0, -O1, -Os, -O2, -O3五个不同的优化级别，当不同优化级别下的编译器产生了不同的结果，我们就可以发现编译器中存在的漏洞。而等效模式输入测试是指给定特定输入，利用动态分析识别测试程序未执行区域，对未执行区域进行删除或是篡改，产生变体。由于测试程序和它的变体程序在该特定输入的前提下是等效的，通过对比测试程序和它的变体程序在同一编译器下的编译结果的差异可以发现编译器中存在的漏洞。

之前的工作通过触发编译器错误行为的测试程序数目来判定编译器测试方法的效果。这种做法很不准确，因为不同的测试程序可能触发相同的编译器错误，因此本文提出了新的评判标准---修正补丁数目。当测试程序在一个编译器的早期版本中触发了错误，检查新版本中的编译器补丁，利用补丁数目来近似的估计编译器漏洞的数目。

文中对GCC和LLVM两种最通用的编译器进行的测试，并通过三个维度对测试效果进行判定：Efficiency（效率）：在同一时间能进行测试的程序数目。Strength of test oracles（强度）：给定相同测试集，哪种测试方法能检测出更多漏洞。Effectiveness of generated test programs（有效性）：哪种产生测试程序的方法更有效。

Table 5: Number of Test Programs Used During the Experimental Period

Compilers	RDT	EMI	DOL
GCC	27,990	4,794	62,552
LLVM	27,990	5,040	64,385

图1 编译器测试方法效率对比

通过图一我们可以看出DOL是效率最高的检测方法，EMI是效率最低的检测方法。检测方法的效率越高，能检测出的编译器漏洞数目越多。

Table 6: Strength of Test Oracles

Compilers	Detected Bugs			Unique Bugs		
	RDT	EMI	DOL	RDT	EMI	DOL
GCC	18	12	18	0	0	0
LLVM	16	4	10	6	0	0
Total	34	16	28	6	0	0

图2 编译器测试方法强度对比

通过图2我们可以看出在给定相同测试集的前提下，RDT能检测出最多的漏洞以及独特漏洞。EMI能检测出的漏洞最少。

Table 7: Effectiveness of Generated Test Programs

Compilers	RDT		DOL	
	Random	Variant	Random	Variant
GCC Bugs	11	8	11	8
LLVM Bugs	9	6	4	2
GCC Unique Bugs	6	3	6	3
LLVM Unique Bugs	5	2	3	1

图3 编译器测试方法效用对比

通过图3我们可以看出由EMI产生的变体程序，检测效率低于由Csmith产生的随机程序。文中还指出EMI产生的变体程序能检测出的漏洞，能够作为随机生成程序的补充。

通过这篇文我们可以看出，选择编译器测试的优化级别很大程度的影响测试的效率，近一步影响测试结果，通过结合不同优化级别的测试方法提高效率。对于今后的工作，尝试去平衡三种影响编译器测试效果的因素，改进生成测试程序的方法，更进一步的提高测试的效率及准确度。

以上内容只是论文的一个简单介绍，其中还有更多的细节由于篇幅的原因没有办法进一步介绍，感兴趣的同学可以对这篇论文进行更加详尽的阅读。

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

TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning

刘 博 推荐

“TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning”是机器学习与深度学习领域的国际顶级会议NIPS 2017录用的一篇口头报告（oral）文章，该会议于2017年12月在美国加利福尼亚州长滩举行。该次会议共收到3240篇论文投稿，有678篇论文被选中作为大会论文，其中有40篇被选中进行口头报告（oral），112篇选为亮点海报（spotlight）进行展示。

随着深度学习神经网络规模越来越大，训练一个深度神经网络（Deep Neural Networks, DNNs）往往需要几天甚至几周的时间。为了加快学习速度，需要分布式的CPU/GPU集群来完成整个训练。如何提高分布式深度学习的速度和可扩展性有着非常实际的意义。如图1，在主流的基于数据并行的分布式深度学习中，各个计算单元（使用不同的训练数据）（worker）并发地训练同一个DNN，每一次迭代结束后，各个计算单元里的DNN参数或梯度($g_t^{(i)}$)会通过网络（如以太网，InfiniBand等）发送到参数服务器进行同步再下发($\overline{g_t}$)。训练时间主要包括计算时间和通信时间。计算时间可以通过增加workers线性地减少，然而，通信时间却随着workers的增加而增加。因此，在大规模分布式训练中，通信时间成为了新的瓶颈，如何降低通信时间成为很重要的研究课题。

作者提出将网络模型的参数梯度量化到低精度，即在保证新梯度的均值还跟原先梯度一致的基础上，对梯度随机地量化到三元值，即为0和±1。理论上，TernGrad可以把通信量至少减少到1/20；实际应用中，即使对0和±1采用简单的2比特编码（浪费掉一个可用值），相对于传统的32比特的浮点型梯度，通信量也可以减少到1/16。这可以大大克服通信瓶颈的约束，提升分布式训练的可扩展性。

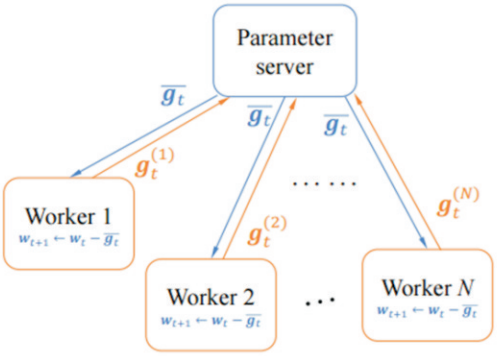


图1 基于数据并行的分布式训练

分布式深度学习中，降低梯度的精度会严重影响DNN训练效果。在基于量化的深度模型压缩算法中，即使可以将网络权重量化到低精度，但是训练过程仍然需要浮点精度的梯度，以保证训练的收敛性。本文使用梯度三元化方法将梯度量化到只有三个值，不影响最后识别率，这是由于在普遍采样的随机梯度下降（SGD）训练方法中，梯度是随机的，而且这种随机性甚至可以有助于DNNs跳出很差的局部最小值。考虑到梯度本身就是随机的，且在训练过程中学习率往往较小，在梯度形成的优化路径上，即使TernGrad偶尔偏离了原来的路径，由于均值是一样的，后续的随机过程能够将偏离弥补回来。

Making Smart Contracts Smarter

王泽丽　推荐

“Making Smart Contracts Smarters”是国际顶级安全会议CCS2016年录用的一篇关于区块链智能合约代码安全性的文章。文中调研了以太坊中智能合约运行的安全问题，主要是代码漏洞。由于智能合约可以处理很多数字货币，一旦这些漏洞被攻击者利用，会造成巨大的财产损失；且因为区块链不可更改的特性，智能合约一旦发布到上面便无法再更改，导致发现漏洞也不能打补丁。针对此种问题，文章设计实现了一个称为Oyente的符号执行工具，其可以在合约发布

接上页

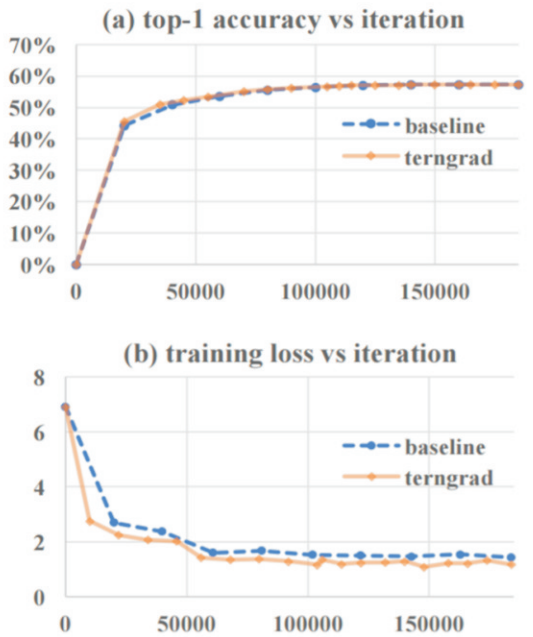


图2 基于 TernGrad 训练 AlexNet 的收敛性

具体来说，作者基于伯努利分布，类似于扔硬币的形式，把梯度随机量化到0或±1。在合理假设下，本文从理论上证明了TernGrad以趋近于1的概率收敛到最优点。相对于标准SGD对梯度的上界约束，TernGrad对梯度有更强的上界约束，为了提升收敛性，作者提出使用逐层三元化和梯度修剪技术，使得 TernGrad的梯度上界约束接近标准 SGD 的上界约束，从而大大改善了TernGrad的收敛性。实验结果表明，在分布式训练AlexNet时，TernGrad有时甚至会提高最后的识别率；在GoogleNet上，识别率损失也小于2%。图 2 为分布式训练AlexNet的结果，相对于标准SGD基线，TernGrad具有同样的收敛速度和最终识别率。

工作充分考虑了通用性和透明性（几乎不影响训练的精度），使得TernGrad得以更广泛更简便的应用。TernGrad通过提升通信速度和效率让分布式深度学习能更有效地利用计算资源。这样不但可以让最新的硬件（比如GPU，TPU等）更好地发挥计算潜能，也能让老的计算集群大大提高性能（比如用低速以太网连接的集群），节省资源和成本。更重要的是对于网络带宽有限但计算资源几乎可以无限扩展的云计算，TernGrad可以大大提高分布式深度学习的延展性。

刘　博
2015级博士生
研究方向：深度学习系统
Email：80937591@qq.com

之前帮助合约开发者查找潜在的安全漏洞。此外，通过使用Oyente工具发现，现有的19366个智能合约有8833个存在漏洞。

文章主要的创新点和意义有以下四点：（1）描述了以太坊智能合约中的几类新安全漏洞；（2）将以太坊智能合约的语义形式化，并提出推荐方案以解决描述的漏洞；（3）构建了一个符号执行工具Oyente，其可以分析以太坊智能合约，检测出漏洞；（4）在真实的以太坊智能合约上运行了Oyente，检测出以太坊网络中存在的攻击。

文中介绍了合约中存在的几种安全漏洞，比如（1）依赖交易顺序合约，会存在类似于买方购买商品的同时，卖方抬高价格，两个交易处理顺序不同，会导致结果不同，见图1；（2）依赖时间戳合约，由于产生哈希值的输入参数除了时间戳之外，其他的参数都是知道的，攻击者可以通过预计算和选择时间戳使其产生一个更容易被接受的哈希值，从而有选择性的将奖励发放给特定的人；（3）误操作异常合约，主要是指在调用合约时不验证返回值，合约中止导致的安全问题；（4）重入漏洞，其典型代表便是Dao攻击，其主要是利用合约调用时，调用者等待被调合约执行完毕的间隔时间，发起攻击。这些漏洞直接关系到智能合约相关者的资产，解决这些问题迫不及待。

```
contract MarketPlace{
 uint public price;
 uint public stock;
 /.../
 function updatePrice(uint _price){
   if (msg.sender == owner)
      price = _price;
 }
 function buy (uint quant) returns (uint){
   if (msg.value < quant * price || quant > stock)
      throw;
   stock -= quant;
   /.../
 }}
```

图1. 一个用户可以买卖代币的交易所合约

Oyente工具是基于符号执行方法实现的，符号执行是指将程序变量的值表示为输入符号值的符号表达式。每个符号路径都有一个路径条件，它是通过累加这些输入必须满足的约束，建立的符号输入公式，以便执行检查该路径。如果路径条件不可满足，则路径不可达。否则，路径可达。而且，其他分析工具也可以作为独立的插件实施，而不会干扰该工具现有的功能。其体系结构见图2，其中虚线框内的是其主要的组件，阴影框内的是可以被公开访问的。

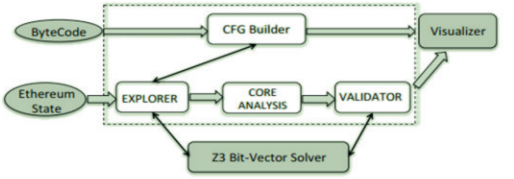


图2. Oyente体系结构概览

文章通过定量和定性分析衡量了Oyente 工具，其目标分为三层，首先衡量前面描述的四类安全漏洞在真实以太坊合约中的普遍性；其次，强调 Oyente的设计和实施选择是由现实生活中的智能合约的特征驱动的，Oyente在处理分析时足够强健；最后，文章提出了几个案例研究，表明许多合同开发者对以太坊微妙语义的误解。图3是利用Oyente工具测出来的每类安全漏洞中合约的数量。

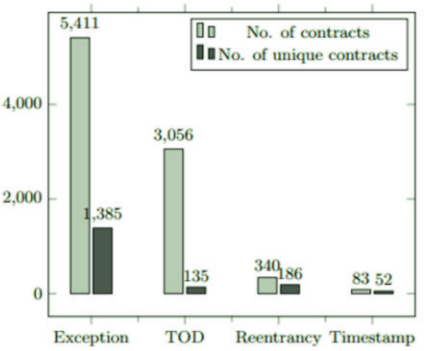


图3. 每类安全漏洞中合约的数量

王泽丽
2017级博士
研究方向：区块链安全
Email：zeliwang@hust.edu.cn

Private, yet Practical, Multiparty Deep Learning

公绪辉　推荐

“Private, yet Practical, Multiparty Deep Learning”是计算机领域的国际顶级会议ICDCS 2017录用的一篇文章。本届会议于2017年6月在美国亚特兰大召开。本文由浙江大学纪守领老师与理海大学Ting Wang老师的团队合作完成。

深度学习（Deep Learning）是一个近些年备受关注的研究领域，在机器学习中起着重要的作用。它的前提是建立在大量可获得的训练数据上。大部分深度学习系统是需要集中式数据存储库。但这存在隐私泄露风险：对于数据拥有者来说，数据已经不在掌控之中，也不知道该系统如何使用他们的敏感数据。此外，在一些领域中，尤其是与医疗相关的，个人敏感信息共享是法律禁止的。因此，这种集中式训练系统只能执行在他们自己拥有的数据上，这会产生次最优化的模型。在这种情况下，隐私性和机密性限制极大地阻碍了对这些拥有者的数据充分的利用。因此，对多方深度学习（MDL）的需求至关重要，在这种情况下，数据拥有者可以联合训练深层神经网络（DNN）模型，而不需要共享他们的敏感数据，而这些数据却从其他拥有者的数据中受益。

关于MDL，现已提出的解决方法是基于模型共享范式：各方使用自己的数据训练模型，然后共享它的局部模型，最后聚合所有局部模型获得全局模型。然而通过逆向工程，局部模型可能会泄露隐私数据。关于这个问题，有些研究工作是通过在局部模型中添加随机化因子，从而保护隐私数据。但是这会显著地导致全局模型的效用损失。

基于这些问题，本文提出∝MDL，不仅提供强的隐私保障而且保留所期望的模型效用。

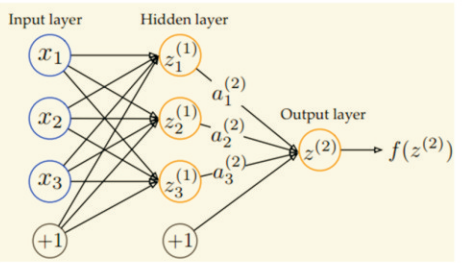


图1：神经网络的一个示意图

训练一个DNN模型（参见图1）就是最优配置各个参数。由于模型的复杂性，随机梯度下降和它的相关衍生方法经常被用到。DNN模型的每个参数ω，通过迭代公式 $\omega := \omega - \lambda \frac{\sum_{\mathcal{A}_i \in \mathcal{A}} \nabla_\omega^i}{|\mathrm{A}|}$ 计算出来，其中λ表示学习率，$\mathcal{A}$表示参与者集合，$\mathcal{A}_i$表示第i个参与者，∇_ω^i是第i个参与者（worker）计算出关于ω的梯度。

对于攻击模型，本文考虑的是半诚实模型，即每一方严格执行预定协议同样好奇其他参与者的隐私信息。进一步，不同的参与者会同谋，以便获取其他参与者的隐私信息。

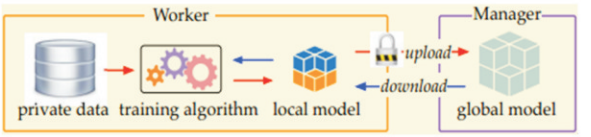


图2 多方深度学习架构图

如图2所示，多方深度学习架构图。Worker与Manager之间有通信链路，Worker既可以上传数据到Manager，又可以从Manager下载数据。

在本文研究中，假设有n个Worker和一个Manager。直觉上，越多的局部梯度聚合，获得更好的全局梯度估计值。本文正式描述这个直觉通过：ρ –能见度。即，对模型每个参数，只有至少m= ρ n 局部参数聚合，才能获得全局参数估计值（O≤P≤1）。

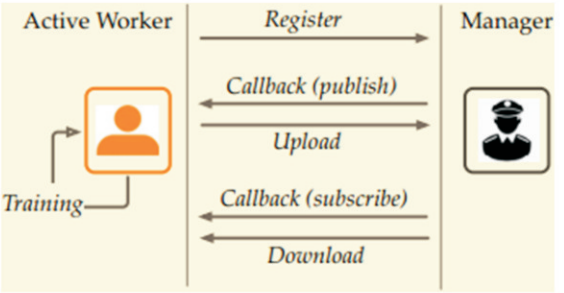


图3 Manager 与 Worker 协议交互过程

如图3所示，协议的简略过程。Worker使用自己局部训练集训练模型参数，然后随机选取一些参数并发送给Manager（Register）。对于一个参数ω，如果至少m个Worker向Manager发送过，则Manager向Worker发送调用消息（Callback）。Worker收到调用消息后，通过隐私保护的方式发送他们的相应参数ω的局部梯度（Upload）。Manager收到Worker发送到局部梯度后，通过相应计算，获得全局梯度并通知所有Worker（Callback）。Worker收到通知后，下载全局梯度，并更新自己的相应参数（Download）。

具体来说，首先通过可信任第三方完成初始化过程，Manager和Worker会收到相应的公钥与秘钥。

当完成初始化过程后，Manager处理所有请求和调用。它记录所有Worker发送到参数，并使用Shamir秘钥共享协议（如果至少有m个Worker加密的局部梯度，则全局梯度能解密出来）和ElGamal加密系统对消息进行加密操作。

当完成初始化后，Worker进入训练过程。在每次训练周期中，它决定发送到Manager的参数。然后Worker使用随机梯度下降法更新局部参数。接下来它上传加密的局部参数 到Manager。当收到Callback后，Worker下载更新后的参数。

通信复杂度分析：作者假设每个Worker 注册（Register）最小数量的参数k= ⌊mt/n⌋（*t*表示在DNN中所有参数的数量），则在每次更新参数周期，每个Worker通信代价为O(mt/n)。相应的，Manager通信代价为O(mt)。

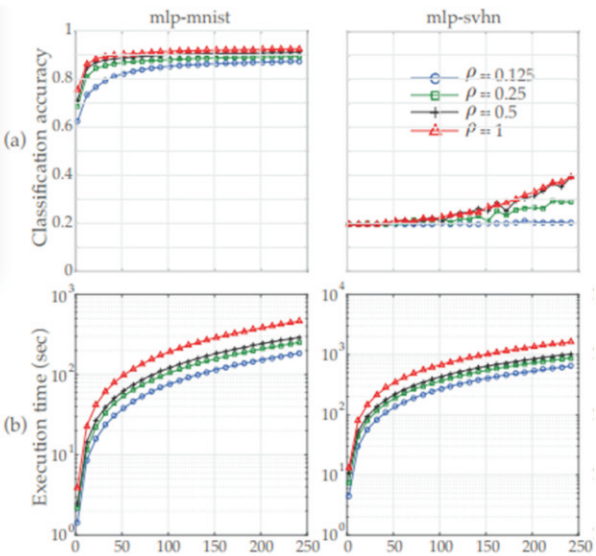


图4: 模型效用和训练效率间的权衡

最后，作者使用标准数据集MNIST、SVHN和DNN架构测试所提出的算法。图4所示在不同ρ情况下，作者测试所提算法∝MDL训练效率，其中横坐标表示训练周期。正如所直觉分析一样，在图4（a），越大的ρ，暗示更高的模型精度。在图4（b），越大的ρ，暗示更高的训练代价，协议执行时间越长。

公绪辉
2015级博士生
研究方向：网络安全、深度学习
Email：xuhuigong@hust.edu.cn

Complete Event Trend Detection in High-Rate Event Streams

于水英　推荐

“Complete Event Trend Detection in High-Rate Event Streams”是被计算机国际顶级会议SIGMOD 2017录用的一篇论文。本文针对现有的完整事件趋势（Complete Event Trends，CETs）检测产生重复计算及内存需求问题，定义了CET图来压缩需要匹配的所有CETs，并提出了CPU和内存优化的CET检测算法。

复杂事件处理已经成为支持流应用如财务欺诈检测、健康分析等的一项重要技术。复杂事件处理系统使用高速事件流检测事件序列，以实时预测事件趋势，如实时检测循环空头支票和不规则心率。这些事件序列将是随机的、statically unknown、潜在不定长度的，因此被称为完整事件趋势检测（CETs）。

对于时间严格CET检测的挑战主要有以下三个：

不确定长度且指数级的CETs 不仅每一个CET不确定长度，且经证明CETs的数目在最坏情况下为指数级。

CPU vs 内存 在CETs中有许多共有事件子序列发生，要么调用重复计算，要么在CET检测时需要大量的内存存储部分CETs中间结果。分别导致的结果为，系统可能低延迟，或者运行时由于内存不足而失败。

Np难流划分问题 分而治之的原则是对上述CPU和内存之间平衡问题通常的解决方案。也就是说，把流划分，内存对每个划分进行计算和复用，从而得到最终结果。在指数级增长的检索空间中，轻量级流划分算法必须保证系统的响应能力。

为了更清楚的表达CET检测的过程，从而进行优化，这里设计了Baseline算法：

Baseline CET检测

对于每一条事件趋势（含多个事件）分别存储每个所涉及的事件，如图1：

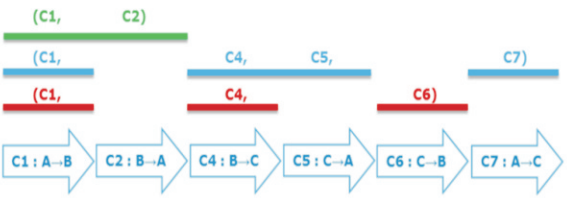


图1 Base-Line CET检测

基于图的CET检测

以上Baseline算法的空间和时间复杂度都比较大，因为这个算法没有考虑在CETs中的共有事件序列，一个单独事件会被分配到几个CETs中，如图1中，事件c1是所有CETs的一部分。为解决这个问题，本文提出了压缩的数据结构，CET图（如图2，为相应图1中事件的CET图的构造），其中图的节点代表流中的事件，边代表CET中相邻事件的联系。CET图防止了每个相关事件趋势对事件的重复存储，避免了由于每个新事件进行比较的重复计算。

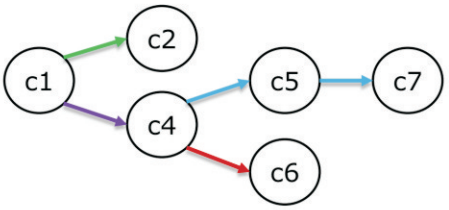


图2 CET图的构造

Dhalion: Self-Regulating Stream Processing in Heron

林昌富　推荐

“Dhalion: Self-Regulating Stream Processing in Heron”是被计算机数据库领域的国际顶级会议VLDB 2017录用的一篇论文。本文首先指出现有流处理系统面临的两个关键挑战：1）为了达到服务等级目标（service level objective（SLO）），需要人工调整各种相应的

接上页

对整个CET图的检测，这里区分了对CPU和内存消耗的两种极端情况下的算法。有两种CET图遍历算法：优化时间的基于BFS算法；优化空间（内存）的基于DFS算法。为了达到目标“最小CPU消耗，且内存不溢出” 这里将CET图划分成小graphlets，提出了hybrid CET算法（H-CET），方法如下：

1）每个graphlets内使用BFS算法，提取和存储每个graphlets的CETs；

2）使用DFS算法来缝合graphlets之间的CETs。

划分计划搜索空间中不可分割的graphlets数量使指数级的，为了找到最优的CET图划分策略，这里利用开销的单调性定义了branch-and-bound CET图划分算法（B&B）。B&B划分的原则有：1）每个graphlet中的节点数量尽量平衡；2）内存开销不要太大，即graphlets的数量不要过小，防止内存溢出；3）CPU的开销不要过大，即graphlets的数量不要过大，减小处理延迟。

文章采用了多种数据集，和相应的不同的事件模式和预测对不同的检测方法进行比较。多个实验结果证明CET检测方法在处理时间和内存使用率上较现有工作都有较大提高。如图3所示，当流速率为每秒5万事件时，CET的处理时间较前人工作中最优的SASE++快42倍。

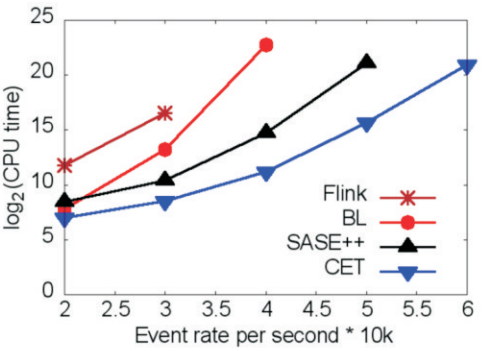


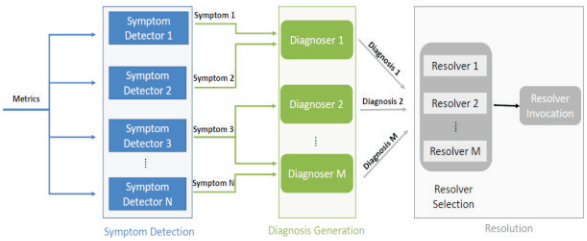
图3 事件速率变化时CPU时间的情况

本文是第一篇在给定的内存中优化CPU时间来处理高速事件流，利用图划分和开销的单调性使CPU和内存之间的开销达到平衡。并结合实验说明了该方法在处理时间和内存使用率上都优于现有工作。

于水英
2015级博士生
研究方向：大数据处理
Email：shuiying@hust.edu.cn

配置参数，该过程耗时且易错；2）为了应对各种突发的、不可预测的负载变化以及软硬件性能下降，需要实时维护服务等级目标。为了解决上述问题，本文首次提出了一个能够自我管理（self-regulating）的流处理系统Dhalion。

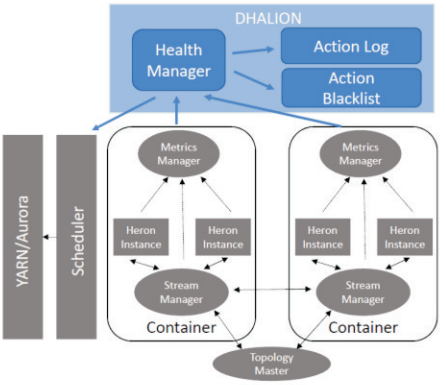
Dhalion的自我管理能力包括以下三方面：自我调整（self-tuning）、自我稳定（self-stabilizing）和自我恢复（self-healing）。自我调整是为了减轻系统用户配置参数的任务量。针对给定的服务等级目标，自我调整指的是系统能够自动调整相应的参数来达到该服务等级目标。自我稳定是指当系统碰到外部激变，例如不可预测的负载变化，系统能够适当地重新调整相应配置参数，使得系统能够稳定地运行，同时能够保证服务等级目标。自我恢复指的是当系统发生故障或者软硬件服务质量下降时，系统能够诊断出问题的根源，并采取适当的措施来恢复正常运行。



图一 Dhalion policy各个阶段

为了实现自我管理，Dhalion的核心是设计定义良好的policy，其中policy指的是用户和系统管理者对系统的需求。例如，用户可能需要一个policy使得系统在不过度使用资源的情况下最大化系统吞吐。图一给出了一个Dhalion policy 的各个阶段。一个典型的Dhalion policy通常包括三部分：症状检测（symptom detection）、诊断生成（Diagnosis Generation）和解决方案（resolution）。其中症状检测阶段通过搜集的若干系统指标来判断是否又该症状，并将检测结果发给诊断生成；诊断生成阶段通过接受的症状检测结果来生成包含若干原因的诊断结果，并将该结果发送给解决方案；解决方案阶段根据相应的问题，选择最合适的解决方案。

在Dhalion policy的基础上，作者给出了Dhalion在Heron上的体系结构，如图二所示。Dhalion主要包含3个组件：health manager、action log和action blacklist。其中health manager包含用户定义的policy，health manager定期的搜集系统指标来检测应用的状态，并针对各种问题，采取相应的解决措施。Action log记录了系统针对特定问题，所采取的解决方案。Action blacklist记录了一些问题及相应的不能达到预期效果的解决方案，这样做的目的是为了避免系统再次采取错误的解决方案。



图二 Dhalion体系结构

本文最大的亮点在于提出了一个能够自我管理的流处理系统Dhalion，Dhalion显著降低了用户或者系统管理调节合理配置参数达到各种需求的工作量，使得流处理系统更健壮。然而Dhalion只提供给用户少量的policy，还需要设计更多policy来满足更多应用的需求。

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

科研的最好时代

张 凡

当曾经熟悉的同学们都已毕业，又迎来了新一批同学时，才发现时间已经过去了这么长。从2013年冬季作为大四实习生进入CGCL实验室进行实习到现在，已经不知不觉进入到第五个年头了。回头看看，这些年里在实验室经历了很多，收获了很多，也成长了很多。

陈汉华老师常对我们提到，现在是科研的最好时代。通过这些年在实验室的学习与经历，才慢慢切实能够体会到这一点。感谢实验室为我们提供了良好的科研条件和许多难得的资源，给了我们一个能够静心进行科研和成长，不被杂思杂虑困扰的净土。在实验室进行学习和科研这些年里，给我留下最深的感触就是实验室给了我们一个非常单纯而自由的科研环境，在这个环境下，我们可以进行多层次多维度的交流。

首先，当然是和导师之间的一对一交流。最开始进入实验室时，当时的自己对科研还没有一个完整的概念，没有明确的研究方向，基本上从头开始。回想起来当初的自己，比起现在刚入学的学弟学妹来在很多方面的能力都有些自愧不如。然而幸运的是有陈老师全方位悉心的指导，我很快就参与到了第一个科研工作中。从如何进行论文阅读，到如何对问题进行分析和思考；从如何准确的表达自己的思想，到如何进行严谨的实验设计；大到如何遵守科研里最重要的各个基本原则，小到如何优化每句话的每个符号，每张图里每根线的颜色和粗细，陈老师都亲自认真的进行指导。尽管当时的我还不能完全理解其中的每一要点，但是陈老师的一丝不苟的科研态度还有正确的科研方法给我留下了十分深刻的印象，在往后的科研上少走了很多的弯路。除了科研方法上，还有生活上，思想上等等方面，一次次交流和谈话都让我受益匪浅。

其次，是小组间的激烈研讨与交流。从最开始两三个人在白板前写写画画，到一小组的人聚在会议室里你一言我一语的讨论，每周的小组讨论会也是我在这些年里珍贵的回忆。最初害怕自己学识不够，了解太少，怕讲错，怕出丑，每次自己报告之前都反复认真的阅读论文，查找资料，尽力去理解，想尽方法去优化ppt，讲稿，使得能让大家听明白。这促使自己对相关的文献仔细地研读，研究工作的背景与思路和它的来龙去脉。同时也害怕对其他同学提问答不上来，自己反复思考哪些地方自己可能讲不清楚会被提问，然后不断追问自己直到弄懂为止。这样一来相比于自己一个人去读论文，反而准备的更加充分，对文章理解的更加深刻。尽管如此，老师和同学们还是会从自己想当然认为没有问题的地方，提出新的问题来迫使自己更深入的去理解和思考。在这样不断的训练之下，也可以看到自己明显的提高。正是因为每个人都有不同的视角和想法，这样通过激烈的交流和碰撞才能更好的发现自己的不足，并诞生出新的想法。当前我的一些

工作想法，或直接出自于研讨之中，在研讨之中产生灵感。因此，也要非常感谢老师和各位参与的同学们的付出。

然后，是实验室这个大家庭内部的交流了。感谢金老师将整个实验室组成这样一个大家庭，每个人分工明确却又不完全相互独立。虽然平时大家都忙于自己的科研工作，课题各不相同，但是大家仍然有许多可以相互进行交流和沟通的平台，如实验室提供的实验室之家、季刊等。通过阅读其他组优秀同学的经验、阅历，对照自身，也对自己有很强的激励和警醒作用。而对我来说印象更深的是每年实验室进行的开题，以及在每年暑期年会最后的博士毕业论文开题汇报，无论是向做的优秀的博士学习，还是听取各个老师们对做的还不够好的博士们的十分认真中肯的点评，都是难以多得的宝贵经验。

最后，是和实验室外的专家和博士们交流的经历。最初在对科研并没有太多了解，也没有太明确方向时，并没有太看重这一点。然而逐渐对科研入门，也经历过多次学术报告之后，渐渐也喜欢上了听成功的学者分享他们的成果与经验，并且思考自身哪些做得到而哪些还值得改进。于是从被集体拉去听学术讲座，变成了针对性地主动通知同学去听讲座，并积极思考自己从报告中得到的收获。除了实验室多次举办的学术报告会外，实验室也给了我们能够无忧参加国内和国际会议的宝贵机会，例如近两年实验室都资助全体同学参加CNCC这一计算机年度盛会，这实在是非常难得的。参加如此大型的会议，除了听取精彩的报告和与专家进行直接对话外，我更喜欢的是会下能够与其他各个高校的同学进行自由的交流。而在读博期间我有幸出国参加了三次国际会议，期间曾游历过新加坡国立大学和多伦多大学，并和在那边留学的博士生们进行了长聊，在分享他们的科研经历的同时，好好自己思考自己的平时的工作态度与习惯，以自警，自省与自勉。事实上，实验室提供给我们的资源、待遇甚至更优于这些国外大学，而能更加成功的同学们优于我们的更多是来自自身对科研的积极主动性，以及对实验室和团队的粘合性。

实验室的要求很严格，但正是这么严格的管理和规定才能创建好这么强大的家庭，带来了这么好的科研氛围，给大家带来这么多的良好交流机会。机遇与挑战共存，这是一个科研的最好时代，也是一个逆水行舟不进则退充满挑战的时代。科研终究还是对自己的磨砺与战斗。再好的机会不自己好好把握也只是一场空。寄语同学们珍惜在实验室共同度过的大好时光，抓住机遇，全力以赴，迅速成长。为实验室的发展添砖加瓦，让这最好的时代一直持续下去！

张　凡
2014级博士生
研究方向：分布式实时流处理
Email：zhangf@hust.edu.cn