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：www.uhust@hust.edu.cn
网　址：http://grid.hust.edu.cn

服务计算技术与系统教育部重点实验室
集群与网格计算湖北省重点实验室

并行與分布式計算通訊

BING XING YU FEN BU SHI JI SUAN TONG XUN

2018年第3期　总第34期　2018年09月

2017-2018年度实验室各研究方向成果展示

封面人物：金琰琪——精诚所至，金石为开

浅谈AIOps

http://grid.hust.edu.cn

Learning is important
以史为镜
与时俱进
广纳谏言

CONFERENCE CENTER
会演中心

2017-2018年度实验室各研究方向成果展示

又一年，丹桂飘香。华中科技大学“服务计算技术与系统教育部重点实验室”暨“集群与网格计算湖北省重点实验室”在老师和同学们的共同努力下，披荆斩棘，攻克难关，带着累累硕果迎来了又一个崭新学年的开始。值此新学年伊始，本季刊对实验室过去一年的成果进行回顾与总结。

在云计算与移动计算方面，主要致力于虚拟化、数据中心、云端融合等领域研究。研究成果包括：1）在云软件平台层，主要对新型的轻量级虚拟化进行研究。设计了容器镜像快速部署系统和资源视图隔离机制。2）在云基础架构层，聚焦数据中心网络性能。为了加速网络性能同时兼顾灵活性，分别提出了基于FPGA＋CPU以及GPU+CPU两种加速网络功能的平台。3）在云应用层，不仅针对典型流计算应用开展了系统研究，而且面向新兴的区块链技术以及关系国计民生的智慧城市领域也进行了研究探索。在流计算方面，开发了一系列实际系统，包括基于预调度方法的Spark社区第三方开源项目Spark-Lever、基于环形高速缓冲的低延迟流处理系统TurboStream、基于双范式模型的新型流处理系统DO。在区块链应用方面，探索了不同区块链系统上数据建立高效可信连接的问题。在智慧城市方面，提取出了城市各区域中与交通碳排放密切相关的高效特征集，并设计了基于感知神经网络的碳排放预测系统模型。

在系统软件与体系结构方面，主要是利用各种加速器（GPU、FPGA）以及存储器件（MRAM、PCRAM）等新型硬件为上层新型应用提供有效的支撑平台。研究成果主要有四个方面：1）在内存计算方面，提出了一种基于对象的内存分配和迁移机制（OAM）提高访存效率，并在混合内存上实现了一个面向应用内存访问优化和降低检查点开销的预复制检查点子系统Shadow。2）图计算方面，针对图计算性能受限于较高的原子开销问题，研发了基于并行累加思想的图加速器AccuGraph。3）在运行时系统方面，针对基于约束求解的多线程程序并发调试技术可扩展性差的问题，开发基于图计算的超大规模并发调试框架。4）在应用支撑方面，主要研究了异构内存在支撑大数据应用时的访存行为和性能，提出了基于模型层敏感的内存复用方法Layrub以节约GPU显存消耗。

在网络空间安全方面，主要是围绕云计算、软件定义网络以及深度学习等相关技术的安全问题检测与防护展开研究。主要包括四个角度：1）在虚拟化和云安全方面，提出了一种高效且透明的共享库隔离机制和一种函数即服务云计算下的基于SGX的安全保护方法。2）在智能漏洞检测方面，研发了基于深度学习的漏洞检测系统VulDeePecker，在不需要人类专家定义特征的前提下，自动学习漏洞特征，来检测目标程序是否含有与库/API函数相关的漏洞。3）在云计算软件定义安全方面，针对SDN集中控制管理租户网络与云租户自主管理网络需求冲突的问题，提出了一种面向云环境多租户网络的可信控制器架构TNGuard。4）在云数据安全方面，针对云端监控视频隐私泄漏的问题，主要研究了隐私受保护的视频监控，可以在监控视频密文上检测与追踪运动物体。

在大数据方面，主要是通过研究大数据的存储、检索、分析和挖掘等技术，为大数据的广泛高效应用提供技术支持。主要成果有，1）在大数据基础理论方面，针对隐私保护数据聚合问题，设计一个协议，在无可信任第三方情况下，实现保护用户隐私和降低通信开销的目标。2）在大数据处理方面，设计了一个数据并行应用程序基于阶段划分的程序分析优化器，以提高数据并行应用的性能；针对HPC平台上的I/O控制问题提出一个通用的I/O带宽控制方案；研发了分布式流处理容错系统Ares来满足实时流处理应用的低延迟和高容错的需求。3）在大数据管理方面，针对SPARQL查询引擎无法在合理时间内处理复杂查询的问题，提出了共享内存环境下图查询并行处理系统ParTriple。4）在大数据分析方面，为衡量查询语句与文档之间的语义相关性，提出基于实体的语言模型平滑方法。

过去一年的努力取得的丰硕的成果，这将成为我们更高的新起点，也是我们下一学年更努力的动力。又一股新鲜力量的注入，标志着实验室开启了一个新的征程，迎接新时代下的新挑战。在此，衷心祝愿实验室的所有师生能在新的学年快乐科研、勇迎挑战，取得更加丰硕的成果。

黄 宏

二〇一八年九月

主　编：金　海
本期执行主编：黄　宏
编　委：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、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

……1

热点

……3

封面人物

精诚所至，金石为开……金琰祺　10

专栏

云计算与移动计算组典型成果介绍
……樊　浩，陈　飞，王秀秀，鲁向拥，黄　卓，黄　航，顾　琳，肖　江，吴　松　12
系统软件与体系结构组典型成果介绍
……刘海坤，姚鹏程，张　宇，郑　龙，刘　博，廖小飞　22
网络空间安全组典型成果介绍
……马晓静，邹德清，羌卫中，代炜琦，袁　斌　34
大数据组典型成果介绍
……华强胜，胡振宇，钟芷漫，林昌富，赵　峰，郑志高　45

声音

块设备 I/O 虚拟化……刘　伟　56
浅谈 AIOps……陈　琼　58

推荐

DCatch: Automatically Detecting Distributed Concurrency Bugs in Cloud Systems……罗　鹏　推荐　60
Exploring Non-Volatile Main Memory Architectures for Handheld Devices……王孝远　推荐　62

交流

青春的印记……陈　飞　64

具有新内存技术的智能内存架构

（刘 聪 整理）

为了缩小计算和内存之间的差距，现有两类解决方案：内存丰富的处理单元和具有计算能力的内存。

内存丰富的处理。该方法增加了靠近计算单元的片上存储器的容量，从而可以减少计算单元和存储器之间的数据移动。新兴的高容量3D堆叠存储器，如高带宽存储器（HBM）或混合存储器立方体（HMC），已被提议使用基于硅通孔（TSV）的3D集成或插入器与CPU / GPU集成在一起 –基于2.5D集成。基于内插器的2.5D集成已经被业界所接受，用于将3D堆叠存储器与大规模设计集成，因为它在成本和更好的热性能方面具有真正的3D集成的可行性。例如，早在2015年，AMD的Fury X GPU集成了4GB的3D堆叠高带宽内存（HBM）；最新款Nvidia的Tesla V100 GPU将HBM的容量提升至16GB和32GB；英特尔的Knights'Linging CPU还集成了16GB高带宽堆叠DRAM，名为MCDRAM（或多通道DRAM）。Xilinx最近还宣布推出集成了8GB HBM的Virtex Ultrascale + FPGA。我将具有GB的3D堆叠存储器的处理单元（PU）（CPU / GPU或FPGA）定义为存储器丰富的处理单元。

具有计算能力的内存。这种方法试图将计算内部（或更近）移动到内存中。这种方法也可以分为两类：计算和内存的粗粒度耦合，称为近数据计算（NDC）和堆栈内存，以及更精细的合并计算和内存，称为内存处理（PIM）或计算内存（CIM），具体取决于计算的粒度和计算的位置。例如，在HBM或HMC存储器架构中，逻辑层堆叠在多个DRAM裸片的底部，并且可以在该逻辑层上实现各种加速器设计，该逻辑层连接到具有大量硅通孔的DRAM裸片。通孔（TSV），在具有极高带宽和低延迟的相同封装中。这可以显着加速具有高能效的数据密集型工作负载。相反，PIM和CIM架构设想使用内部阵列（位线或每单元）计算来利用电压和电荷属性（用于DRAM / SRAM）或利用基于电阻的模拟计算在新兴的非易失性存储器中（如作为MRAM / PCRAM / ReRAM）以实现简单的批量操作，例如矢量矩阵乘法，乘法累加，布尔逻辑和TCAM或关联处理。人们还可以在全球感测放大器附近放置一些逻辑，具有伯克利智能RAM（IRAM）项目提出的类似概念，在数组边缘启用通用计算 – 最有可能采用向量运算的形式。

PHP 现反序列化漏洞，或使 WordPress 遭远程攻击

（陈肇炫 整理）

英国安全公司 Secarma 的研究主管 Sam Thomas本月在 Black Hat 和 BSides 安全会议上展示了 PHP 编程语言的安全漏洞，并指出该漏洞影响了所有接受用户资料的 PHP 应用程序和库，包括 WordPress 等内容管理系统（CMS），并将允许远程程序攻击。序列化（Serialization）与反序列化（Deserialization）是所有编程语言都具备的功能，序列化将对象转换为字符串，以将数据迁移到不同服务器，服务或应用程序上，然后通过反序列将字符串还原到对象。

安全研究员 Stefan Essar 在 2009 年就透露了PHP中反序列化黑客控制的数据带来的风险，而相关的漏洞不仅存在于 PHP 中，还存在于其他编程语言中。Thomas 公布的是 PHP 的新攻击技术，可用于各种场景，例如 XML External Entity（XEE）漏洞或服务器端伪造请求（SSFR）漏洞等。

Thomas 表示，过去外界认为 XXE 漏洞带来的最大问题是信息外泄，但现在可出发程序执行。相关攻击分为两个阶段。首先，将包含恶意对象的 Phar 存档上传到攻击目标的本地文件系统，然后触发一个基于 phar:// 的文件操作，就可

能导致恶意程序执行。Thomas 已利用 PHP 的反序列化程序成功攻击了 WordPress 与 Typo3 内容管理平台，以及 Contao 所采用的 TCPDF 库。

通用并行计算算法加速系统 Fractal

（甘 鹏 整理）

大多数现代台式计算机中的芯片具有四个“核心”或处理单元，它们可以并行运行不同的计算任务。但未来的芯片可能有几十甚至几百个核心，要想充分利用这些并行核心是一项艰巨的挑战。

麻省理工学院计算机科学与人工智能实验室的研究人员开发了一种新系统，不仅使并行程序的运行效率更高，而且编写程序更容易。在利用该领域的基准测试算法集进行的测试中，研究人员的新系统经常比采用相同并行策略的现有系统实现 10 倍以上的加速，最多可达 88 倍。

例如，用于解决称为最大流的重要问题的算法已被证明非常难以并行化。经过数十年的研究，一个常见的最大流算法的最佳并行实现能取得八倍的加速，然而前提是算法运行在 256 个并行处理器上。使用研究人员的新系统，提升了 322 倍，且程序只需要三分之一的代码。这个名为 Fractal 的新系统通过称为推测执行的并行策略实现加速。

推测执行是一种优化技术，其中计算机系统可能会执行不需要的某些任务。任务会在知道其是否实际需要之前被执行，以防止在知道需要之后才执行任务所带来的延迟。如果结果表明任务根本不需要执行，那么该任务所做的更改都会被恢复，结果会被忽略。

这样做的目的是在有许多可用的额外资源的情况下提供更多的并发机会。这种方法被用于各种领域，包括流水线处理器中的分支预测，利用数据局部性的值预测，预取内存和文件，以及数据库系统中的乐观并发控制。

在传统的并行程序中，你需要将你的工作划分成不同的任务，但是因为这些任务是在共享数据上运行的，所以你需要引入一些同步来确保这些任务存在的数据依赖得到解决。从 90 年代中期到 21 世纪后期，有许多次对推测执行结构进行研究的浪潮。

虽然在完成计算之前不断中止计算不是一种非常有效的并行化策略，但是对于许多应用程序来说，发生中止计算的情况足够少，使得在中止的情况下浪费的时间比在更传统的并行方案中同步任务所需的复杂检查浪费的时间更少。

然而，对推测执行架构的研究经常在原子性问题上搁浅。与所有并行体系结构一样，推测体系结构要求程序员将程序划分为可以同时运行的任务。但是对于推测体系架构，每个这样的任务都必须是“原子的”，这意味着它应该像一个整体一样执行。通常，每个原子任务都分配给一个单独的处理单元进行处理。

OpenAI NLP最新进展：通过无监督学习提升语言理解

（高辰晶 整理）

近日，OpenAI在其官方博客发文介绍了他们最新研究的一个可扩展、与任务无关的自然语言处理系统，就其在文章中展示的对比结果来看，该系统在一系列不同的NLP任务，如语义相似度计算（Semantic Similarity）、阅读理解（Reading Comprehension）、常识性推理（Commonsense Reasoning）、情感分析（Sentiment Analysis）确实取得亮眼的成绩。

该系统的工作流程分为两个阶段：首先，以无监督的方式，使用语言建模作为训练信号，在大量数据上训练一个Transformer模型。然后，在小得多的监督数据集上对该模型进行微调，以帮助它解决特定的任务。

这项研究的灵感来自于OpenAI之前一项关于情绪神经元的研究工作。在工作中，研究人

员发现当训练数据足够多时，无监督学习技术可以产生惊人的判别性特征。在本次研究工作中，研究人员对这个想法进行了进一步的探索：我们可以开发一个模型，以无监督的方式对大量数据进行训练，然后对模型进行微调，以便在不同的任务上获得良好的性能。研究结果表明，这种方法的效果出奇地好。

文章中表明，此次研究工作还包含了对ULMFiT（Universal Language Model Fine-tuning for Text Classification）进行了拓展，说明了如何对单个与数据集无关的LSTM语言模型进行微调以在各个文档分类数据集上获得目前最好的性能。从文章中展示的结果表上看，基于该方法建立的模型在三种关于测试常识推理和阅读理解的数据集（COPA，RACE和ROCStories）上远超第二名，获得了最好的结果。而这些数据集普遍被认为需要借助多语句推理和有意义的世界知识来解决，而该模型主要通过无监督学习来提高这些技能。这也表明了通过无监督技术来开发复杂语言理解功能的可能。

当然，文章中也提到了模型仍存在着一些亟待解决的问题。首当其冲的就是计算时间问题，许多自然语言处理方法一般只需从头开始在单个GPU上训练相对较小的模型。然而该模型预训练的计算需求相当昂贵，需要在8个GPU上训练一个月。幸运的是，这个环节只要进行一次。第二个显著问题是计算资源问题，由于模型比较庞大，因此需要耗费更多的计算和内存，文章中提到该项目使用了37层（12块）Transformer架构，并且在最多可达512个token的序列上训练，并且大多数实验都是在4个和8个GPU的系统上进行的。当然，该模型对于新任务进行微调的速度非常快，稍减轻了额外的资源需求。第三个问题是通过文本学习世界知识的局限性和偏见，由于互联网上的书籍和文本所包含的关于世界的信息并不完整，甚至是并不准确且带有偏见，使得学习出来的模型受到了影响。第四个问题是老生常谈的问题——模型的泛化问题。尽管研究的方法提升了自然语言处理系统在大量任务上的性能，但目前的深度学习自然语言模型仍然难以泛化。

WiFi联盟开启支持WPA3协议

（李道祥 整理）

目前，从负责监管无线网络标准的 Wi-Fi 联盟传来消息，称已经正式开始认证支援第三代“受保护无线连接”（WPA3）加密协议的产品。

现行WiFi网络连接有三种加密方式，安全性的排序为：WEP<WPA<WPA2。WPA全名为是一种保护无线电脑网络（Wi-Fi）安全的系统，它是继承了前一代的系统有线等效加密（WEP）的基本原理而又弥补了WEP的缺点：WPA加强了生成加密密钥的算法，因此即便收集到分组信息并对其进行解析，也几乎无法计算出通用密钥；WPA中还增加了防止数据中途被篡改的功能和认证功能。WPA2是WPA的增强型版本，与WPA相比，WPA2新增了支持AES 的加密方式。

目前，广泛使用的WiFi的WPA2加密模式不断被爆出被破解了，轻者网络被蹭网，严重则用户因而隐私信息被窃取泄露，甚至造成财产损失。鉴于WiFi的WPA2加密协议弊端，无线网络标准的 Wi-Fi 联盟开始认证支持第三代“受保护无线连接”（WPA3）加密协议的产品。

然而，要想获得完整的WPA3安全保护，还需要双方的支持。换言之，除了购买支持WPA3的新款路由器，客户端设备如手机、平板和电脑也必须支持WPA3协议。目前已有许多企业宣布支持WPA3，其中就包括已经开始生产支援802.11ax和WPA3的手机/平板电脑芯片的高通公司。

就如同柯达胶卷被数码相机所取代；CD机被MP3以及手机所取代；功能机被智能机所取代一样。在滚滚向前的科技潮流面前，任何一项技术，都很难避免被淘汰和取代的命运。而

随着5G时代的到来，WIFI技术因为其使用的局限性，也很难逃脱被淘汰和取代的命运。而随着5G时代的到来，WIFI技术因为其使用的局限性，也很可能被淘汰和取代。

推动下一波时序创新的数据中心设计趋势

（郑志高 整理）

流视频、物联网 (IoT)、社交媒体、基于云的企业软件的快速增长，以及即将采用的5G无线网络正在共同推动对经过优化以运行庞杂多样的任务和应用的更高带宽数据中心的需求。

新的软件和服务产品的推出向依赖数据中心部署的新硬件。不久之前，新的软件和服务产品还必须与新的服务器、存储和交换机的推出保持一致，刷新周期通常是2年。为云计算、软件即服务(SaaS)和Web服务推出新服务的速度如今正超越这一固定硬件升级速度，给数据中心运营商和Web服务公司带来了严峻挑战。

数据中心加速往新技术过渡

为了满足当前和未来的需求服务提供商、数据中心运营商和Web服务公司正在迅速向软件定义网络(SDN)模型过渡，从而将软件和服务从基础计算、交换和存储硬件中抽离出来。服务提供商和数据心运营商正在采用新的硬件技术，这种技支持业界向SDN过渡，同时提高数据中心之间和内部的速度和带宽。

服务器、存储系统、脊/叶交换机、合路由器和光转换器全都在经历翻天覆地的技术转变，纷纷采用100G/200G/400G光传输技术、更高速的PCIeGen 4和加速器的缓存一致性互连(CC I X)数据总线、基于NVMExpress(NVMe)的固态存储、针对机器学习和人工智能优化的专业处理技术，以及新的存储器技术，以便满足对更高带宽网络的不断增长的需求。

整个数据中心硬件带宽升级的同之处在于，参考时钟时序要求日趋严格。现在，系统架构师比以往任时候都更加注重硬件设计过程中的时序和时钟树设计。

数据中心互连

数据中心通过高速光纤连接实现彼此之间，以及与底层内核和聚合信网络的连接。相干光学是在数据中心聚合交换机和光转换器中实现的最新技术，在当下和不久将来分别支持以100G及高达600G的速度通过光缆传输更多信息。概括地讲，相干光学技术结合了先进的高速数字信号处理和高速数据转换器来调制各个发射器和接收器之间传输的光的振幅及相位，从而能够通过现有光纤网络发送更多数据。

发射器和接收器中的数据转换器

需要通常超过1.7GHz的超低抖动、高频参考时钟。此外，需要参考时序来支持数字信号处理。初始100G相干光线卡和模块设计使用多个时序IC及振荡器来满足这些时序要求，需要大的电路板空间和成本。SiliconLabs公司推出的Si5342H和i5344H相干光时钟是适用于相干光学的优化单芯片时序解决方案，将所有参考时钟整合在一个解决方案中，从而实现不到100fs的超低抖动性能。

脊交换机和叶交换机

脊交换机和叶交换机创建了服务器机架和存储设备之间的连接网络在整个数据中心均匀分配流量。叶交换机位于每个机架的顶部，提供到服务器的下游连接和到网中各个脊交换机的上游连接。新一代脊交换机和叶交换机设计采用包括28G和56G串行器/解串器(SerDes)的交换机SC，以支持10GbE到25/40GbE的下游端口带宽迁移和到100GbE的上游端口迁移。这些增加的速度需要参考时钟抖动性能显著提高，最高规格低至56G SerDes的12kHz~20MHz掩码中的150fs rms。

这些设计中使用的FPGA、CPU、存储器、CPLD和基板管理控制器(BMC)还需要额外的系统时钟。Silicon Labs的Si5341任意频率时钟发生器和Si5345任意频率抖动衰减时钟可满足这些应用的超低抖动性能要求，同时在单芯片时序解决

方案中提供多达10路独特频率输出，使其成为在100GbE设计中并入28G和56G SerDes的同步或异步叶交换机及脊交换机设计的理想选择。

服务器和存储器

当今数据中心的大多数服务器和存储处理器均基于Intel x86体系结构。越来越多基于IBMPower和ARM体系结构的新产品正在推出。与x86平台相比，基于Power和ARM的平台通常需要额外的时钟来支持处理器和其他I/O功能。无论CU偏好如何，每个体系结构和平台都使用高速数据总线在CP U、存储器、存储设备及附加卡之间传输数据。

PCI-Express(PCIe)是服务器中用的主要数据总线，因为它具有实现成本低、高带宽，以及可用于大部分CPU、FPGA、SoC和ASIC的优势。PCI SpecialInterest Group(PCISI G)最近推出了其第四代P C I e规范，将数据传输速率从8G b/s提高到16Gb/s。

除了在服务器主板上使用之外，随着固态硬盘(SSD)逐渐变得比硬盘介质更受青睐，PCIe正日益在数据中心存储应用中得到广泛采用。由于PC I e数据总线在服务器和存储器应用中的广泛使用，从服务器主板上的CPU一直到各个SSD的整个机架如今需要数量更多、精度更高的PC I e参考时钟。固态存储使用的是NV M e协议而非传统硬盘存储设计中使用的SA S或SATA串行协议。基于NVMe的SSD通过标准P C I e连接器连接到存储系统，这意味着所有基于NV M e的SS D都需要PC I e参考时钟。闪存阵列存储系统通常使用FPGA和定制控制器ASIC来管理服务器和S S D之间的流量，每个FPGA和ASIC都需要自己的高性能参考时钟。

虽然硬盘存储有望成为未来几年内的主流数据中心存储介质，但闪存阵列存储系统部署正在迅速增长行业分析师预计，在2018~2020年期间，闪存阵列存储采用率将急剧上升，主要由Web服务数据中心驱动。Silicon Labs 最近推出了两个新的低抖动时钟发生器系列，专为满足x86、Power、A R M和闪存阵列存储系统提出的时钟树要求。Si5332任意频率时钟发生器系列能够以不到230fsrms的抖动性能为闪存阵列存储系统中的PCIe端点、FPGA、处理器和其他SoC/ASIC器件提供多达12路时钟输出。

Si5332器件能够生成分数和整数相关时钟频率、扩频调制时钟，而且能实现低功耗和频率控制，从而可将存储系统设计中所需的所有时钟集成到单个IC中，节省了印刷电路板(PCB) 面积和系统成本。对于仅需要PC I e时钟源的系统设计和附加卡，PCIeGen 4时钟发生器的Si522xx系列是理想的选择。该系列具有241fsrms 典型相位抖动性能、用于降低EMI的扩频调制和硬件输出使能引脚，能够提供2~12路100MHz 时钟输出。

加速卡

新数据中心设备的设计周期通常为两年。为了适应更快的新软件和Web服务产品推出速度，数据中心架构师已经开始开发专门的处理器附加卡，这些卡可提供额外的CPU处理能力，或者为某些应用（如Web搜索、人工智能或机器学习）优化的替代类型的处理能力。附加卡旨在通过PCIe连接器插入标准服务器主板，立即向现有服务器提供更多功能。附加卡的设计周期可能短至6个月，为运营商和网络公司提供了更多掌控数据中心的能力，使其无须重新构建或重新装配使用新服务器的整个数据中心。

过去几年内，许多类型的附加卡已通过使用FPGA、图形处理单元(GPU)和定制ASIC被部署到了数据中心服务器上。随着针对特定应用进行了优化的新GPU、FPGA和SoC产品上市，预计这一趋势将会加速。

虽然新的替代协议正开始得到采用，但服务器主板和附加卡上的处理器件之间的主要互连仍是PCIe。PCIe、CCIX、NVLink、OpenCAPI 和GenZ正在实现CPU、存储器和加速卡之间的更

快数据传输，使数据传输速率达到16~32Gb/s。鉴于这些数据传输速率，参考时钟必须非常精确，以确保极高的信号完整性并最大限减少误码损失。SiliconLabs的Si5332任意频率时钟发生器为加速卡应用提供理想的单芯片时序解决方案，为FPGA、GPU和/或定制ASIC 以及所用数据总线的参考时钟提供所需的高性能时钟频率，并且全都具有出色的抖动性能。

总结

数据中心正在我们生活的许多方面变得愈发重要，使信息得以存储在广泛的服务，以及云服务和新兴人工智能系统中。为了继续支持在云中运行的新创新和应用的快速发展步伐，架构师和硬件设计者必须继续提高数据中心内的服务器、存储设备和交换网络中的带宽。数据中心互连和叶/脊交换机中到100GbE的迁移、服务器和附加卡中的PCIe Gen4，以及固态存储中的NVME都是为满足更高带宽需求而采用新技术的例证。为了确保发挥这些技术的最大潜力，系统设计者必须更加重视时钟树设计，并在整个数据中心使用超低抖动参考时钟。

下一代云计算中心：Unikernel的兴起
——远超Docker，云计算时代的革新者

（吴 刚 整理）

中国信息通信研究院在2018年8月发布的《云计算发展白皮书》中指出2017年全球公有云市场规模达到1110亿美元，到2021年市场规模将达到2461亿美元。而在这么庞大的市场下，GPU云服务、微服务架构技术以及函数即服务架构（FaaS）成为各大厂商争先发展的三个方向。在微服务架构技术方面，Unikernel的兴起给云计算产业带来了新的机遇。在Wiki上给出了Unikernel的概念：Unikernels are specialised, single-address-space machine images constructed by using library operating systems。通俗的解释就是：利用LibOS，将应用程序代码、依赖的链接库，以及相应的配置文件一起编译构建成一个整体的机器镜像，可以直接运行在Hypervisor 之上。由于Unikernel具有单地址空间，没有内存隔离，没有用户态/内核态的上下文切换，移除了大量不必要的库，因此它具有容量小，启停迅速，部署简易，安全可靠，性能优异等诸多特性。但是，由于Unikernel剥离了其整体设备的流程抽象，导致了灵活性和适用性下降。

当前学术界关于Unikernel的构建存在多种方案：

MirageOS：利用Ocaml语言实现了一个完整的网络协议栈，并实现了其他库和语言扩展，对于特定的应用程序，只需要将其需要库链接进去，就可以生成特定的Unikernel。

ClickOS：将 MiniOS和 Click结合起来，MiniOS提供网络栈功能，Click是一种基于软件控制的模块化路由器。通过两者的结合，能够提高网络的吞吐率，并且启动迅速。

Osv：专为Cloudius Systems的云虚拟机设计的新操作系统。OSv能够在不到一秒的时间内启动，在任何虚拟机管理程序之上执行单个应用程序，从而实现卓越的性能，速度和轻松管理。支持C，JVM，Ruby和Node.js应用程序堆栈。

Rumprun：一个可以将现有的未修改的POSIX软件作为单核运行的软件堆栈。Rumprun支持多种平台，包括裸硬件和虚拟机管理程序，如Xen和KVM。它基于rump内核，提供免费，可移植，组件化，内核质量驱动程序，如文件系统，POSIX系统调用处理程序，PCI设备驱动程序，SCSI协议栈，virtio和TCP / Ip堆栈。

Unik：是一种简化unikernel的编译和编排的工具。与Docker构建和编排容器的方式类似，UniK自动将流行语言（C / C ++，Golang，Java，Node.js.Python）编译为unikernel。UniK将 unikernel部署为 OpenStack，VirtualBox，QEMU，AWS，vSphere，ukvm和Xen 上的虚拟机，整合了Rumprun，OSv，IncludeOS 和MirageOS

项目的工作。

还有Drawbridge, Libra，IncludeOS，Ling，HermitCore， runtime.js， Clive， HaLVM，Kylinx等等诸多构建方案。言而总之，当前学术界在Unikernel方面的研究可谓是百花齐放，百家争鸣，但是Unikernel的研究尚未商业化，仍然存在很大的改进发展空间。

云原生NFV中的容器化VNF部署

(张光锋 整理)

电信应用和IT应用对云原生虚拟网络功能（VNF）有不同的需求，通过微服务和容器部署VNF将能够推动云原生NFV部署的成功。

VNF是嵌入式软件，从网络外围设备中提取出来并作为应用程序驻留在虚拟机上。对VNF进行任何形式的更新都会产生耗时的工作，从而影响整体NFV基础设施的运营。为了向云原生做好准备，捆绑的VNF软件需要基于微服务，其中单片VNF被分解为具有多样化但相关功能不同的较小的协作服务集，能维护其自身状态，具有不同的基础设施资源消耗需求，能使用定义良好的API自动缩放和编排。

随着VNF在微服务中的分化、数量的增长，服务提供商可能会面临操作复杂性。为了在生产环境中良好地管理所有微服务，需要使用NFV MANO层和云编排器来实现高层次的自动化。

容器技术恰为此提供了很好的一些思路，容器是操作系统级的一种虚拟化形式，它将应用程序依赖关系、所需的库和配置封装在同一操作系统的其他容器隔离的包中。容器能让应用程序以独立的方式运行，并且可以轻松迁移，随着向云原生的迁移，VNF微服务可以部署在容器中，从而能够连续交付/部署大型复杂的应用程序，但种种方法仍处于云原生NFV的早期阶段。

要在NFV中应用容器，目前还有一定的顾虑亟待解决：与虚拟机相比，容器生态系统仍在发展且不够成熟；容器中涉及安全风险，操作系统中的所有容器共享一个内核，任何违反内核操作系统的行为都会破坏依赖于它的所有容器；使用容器隔离故障并不容易，故障可以复制到其他容器中。

希望在NFV环境中使用容器的服务提供商可能在多租户支持、多网络平面支持、转发吞吐量和有限的编排功能方面面临挑战。在移动边缘计算（MEC）环境中仍然由可能使用容器，这将在未来与5G中的NFV共存。MEC将把用户平面功能接近网络边缘，更接近用户应用程序，以提供非常低的延迟、灵活性并支持IoT、AR/VR这样的实时用例。

容器也可以与NFV环境中的虚拟机一起使用。VNF的部署只能是虚拟机，仅适用于容器以及异构模式，其中一些VNF将在VM中运行，一些在容器中运行，还有的在VM和容器混合环境中运行，服务提供商可以根据NFV基础设施级别的要求评估其部署方法。

云原生NFV方式中容器的优势

拥有一个容器托管微服务可以促进有效的计划和管理来优化资源利用率。容器编排引擎支持为容器提供主机资源、将容器分配给追、实例化和重新调度容器。通过容器，服务提供商可以实现DevOps，从而轻松实现自动化任务。

容器化微服务的另一个主要优点是能够编排容器，以便可以将单独的生命周期管理过程应用于每个服务。这样可以单独对每个服务进行版本控制和升级，而不是升级虚拟机中的整个VNF。在升级整个应用程序或VNF时，容器调度程序将确定哪些单个服务已更改，并仅部署这些特定服务。

容器为NFV基础设施提供云原生能力，为电信特定的应用程序部署和编排提供更高的性能、可迁移性和灵活性优势。要拥有全功能的云原生5G网络，服务提供商必须拥有容器才能部署虚拟机，但是服务提供商将会从ONAP和OPNFV等开源社区寻求进一步的研究和开发。

精诚所至，金石为开

2017年6月5日，ICDCS 2017（The 37th IEEE International Conference on Distributed Computing Systems）在美国亚特兰大召开。CGCL/SCTS实验室硕士生金琰祺的论文“Reducing Cellular Signaling Traffic for Heartbeat Messages via Energy-Efficient D2D Forwarding”被该会议录用，并在会议上做学术报告。该论文应用5G时代下“端到端”多设备智能互联通信（Device-to-Device，D2D）技术开发了一种新型的群体心跳包互助转发系统，以缓解心跳包频繁发送给运营商和移动设备带来的信令与电能的双重消耗。本刊对论文作者进行了有关科研心得的采访，以下以第一人称陈述他的经验总结。

序言

十分荣幸成为本期的受访人物，也很开心有机会和实验室的同学们分享自己三年来的一些收获。与大部分同学们一样平凡的我很难想到有什么特别的经验或方法能够帮助大家取得立竿见影的进步，我所分享的心得也只不过是转述部分刘方明老师和优秀学长们的经验罢了。虽然刘老师并不是一个吝啬表扬的老师，但使我印象最深刻的却是对我的批评。以下便节选部分我曾受到过的“批评”与实验室的小伙伴们分享，共勉。

初来乍到

记得刚加入云数据中心与绿色计算/通信小组的时候，我对学术研究并没有浓烈的兴趣，对于在高水平会议/期刊上发表论文更是从未想过，只是站在自己的角度单纯希望多学习一些技能，积累一些经验。在一次组会上，刘老师严厉批评了我这种“无所谓”的态度，希望我能够以高标准要求自己，来了就要发高水准的论文，先要敢想，再努力去实现。没有努力尝试过又怎么知道自己不喜欢研究呢？在学长们耐心的引导和刘老师多次提醒下，我才渐渐度过了痛苦的磨合期，逐渐接受小组的高标准。

锻炼眼光

刘老师经常教导我们，要时常注意锻炼自己的眼光。这种眼光首先代表了一种标准，即高标准要求自己。在调研阶段，尽量查阅高质量会议或期刊收录的论文。在阅读论文的时候，我们可能无意识间模仿作者的写作风格、做研究的思路、实验的设置等等。所以，选择一个优质的范例对于还没有比较成熟研究风格的科研新兵尤为重要。其次，尽量选择高质量的会议或期刊作为投稿目标。我们实验室通常是在开题阶段制定投稿计划，一个较高的投稿目标自然也会成为论文质量的无形标准。无论最终投稿结果如何，就论文质量而言，一个较高的投稿目标还是有所裨益。除了对于论文本身质量的高要求，参会做学术报告的水准也同样应当受到重视。作为扩大研究影响力的重要途径，刘老师每次都不惜成本为参与论文的同学提供亲赴会议现场交流的机会，每位同学的学术报告都要先在组内进行预演，并要求每位组员都提出问题和改进建议。除此之外，眼光还是辨别质量优劣的能力。在攻读硕士学位期间，印象最为深刻的便是写论文审稿意见。每一篇审稿意见的形成都是一次对其他学者工作质量优劣的判断。虽然一篇审稿意见文字篇幅并不长，但其中的每一句话都需要反复斟酌推

敲。整个过程既是对英文写作的锻炼，更是对思维的锻炼。具备了辨别力再审视自己的工作才能明晰自己的优劣。

最后一课

研究生阶段的最后一个月正好也是论文投稿的时间，在这最后的一段时间里，刘老师也为我上了最后一课。在提交论文之前，刘老师在审阅论文的过程中发现了一个错误，在要求我改正的同时提出多给我一天时间再仔细检查其中的错误。然而在第二天上交的论文中，刘老师又发现了新的错误，于是要求我再花一天时间检查修改，并嘱咐我一定要一个字一个字的看。最终经过三次的返工，才终于做到了挑不出错误。这次历时三四天的文字检查，既是为之前写作时的随意买单，也是文字检查过程中不够耐心认真的结果。表面上，强化训练了定冠词、不定冠词的使用；更深层次的，其实是又一次对于精益求精态度的强化。

精益求精

在小组的三年中，被灌输最多的，也是体会最深的便是对工作精益求精的要求，它贯穿在工作里的每一件小事中。在解决研究中的问题时，精益求精要求解决方案有效且周全。我们在寻求解决方案的时候往往只满足于一个大多数情况下有效的方案，而忽略特殊情况下的变化。很多时候应对这种变化并不困难，难的是在设计方案时便考虑周全。除了研究本身，还有很多时候我们需要展示自己的研究成果，无论是口头表达还是书面表达，例如开题报告，与其他学者交流。在表述的过程中，精益求精要求我们用词精准，逻辑通顺。在表达的过程中，我们常常仅以自己的视角来考虑问题，出现思维跳跃，忽略听众、读者能否跟上，能否轻松理解的现象。除此之外，精益求精还是对细节的较真。例如，小到一个单词的拼写，论文中示意图的大小、排版、批注甚至美观程度都要求仔细修改调整。当每一个细节都用心完善，整个作品的质量也会更上一层楼。

反思

反思自己研究生阶段的工作。研究的过程也并非一帆风顺。从开题阶段对idea实现细节的争议，到做实验时测量手段的艰难探索，再到投稿被拒后大幅的修改，正是在这些不顺利中成长，提高。虽然论文已经发表，但相关的工作由于或主观或客观的原因还有许多地方没能尽善尽美，多少还是有些遗憾。

反思自己研究生阶段的收获。这三年间真正的收获并不是一本学位证、一篇论文或荣誉称号，而是思考方式的转变——更成熟、更严谨、更具有团队意识。对比本科刚毕业的自己，我明白了要多从他人的角度考虑问题；明白了只有对自己高标准、严要求，不放过细节，才能保证工作完成的质量；明白了单靠一个人的力量很难做好一件事情，在团队中“人人为我我为人人”。

尾声

虽然时隔一年多，参会做报告的情景历历在目，刘老师的教诲言犹在耳。非常感谢刘老师领我在学术研究的门口一睹殿内的精彩；感谢小组组员们一直以来对我热心的帮助；希望还在科研道路上奋斗的小伙伴们勇往直前，享受过程，即便过程极为痛苦。精诚所至，金石为开。

金琰祺
2015级硕士毕业生
研究方向：移动计算与绿色计算
Email：richie.jyq@gmail.com

云计算与移动计算组典型成果介绍

樊 浩，陈 飞，王秀秀，鲁向拥，黄 卓，黄 航，顾 琳，肖 江，吴 松

关键词：轻量级虚拟化，容器，流计算，网络功能虚拟化，区块链，智慧城市

1. 介绍

云计算与移动计算小组长期致力于虚拟化、数据中心、云端融合等领域研究。基于本组所承担的“容器虚拟化及新型ICT系统结构”、“高效能云计算数据中心资源调度关键技术研究”、“面向人机物融合的云计算平”、“面向国产服务器的虚拟化关键技术”、“国家高性能计算环境服务化机制与支撑体系”等国家重点研发计划项目，以及“高效能数据中心--数据中心资源的优化组织与调度”、“边缘计算基础理论与关键技术”、“基于无线网络物理层信息的细粒度手势识别方法研究”等国家自然科学基金项目和一批企业合作项目，小组在三个层面开展研究：在云软件平台层针对操作系统虚拟化和容器技术展开研究；在云基础架构层针对数据中心资源管理展开研究：在云应用层针对流计算、区块链和智慧城市展开研究。

1）在云软件平台层，主要对新型的轻量级虚拟化进行研究。设计了容器镜像快速部署系统和资源视图隔离机制。为实现容器的快速部署，提出了文件级缓存架构来充分利用本地的镜像以减少冗余数据的拉取，并通过指令重叠加速构建指令的执行。为了保证容器运行时隔离性，提出了面向容器的系统资源统计方法，并以sysfs的形式提供给容器内应用。另外，还分别设计了基于容器和unikernel构建移动云运行环境的开源系统Android X 和 Libdroid。Android X能够按需生成不同的标准Docker系统镜像，并以hypervisor + 镜像的方式将Android在基于Linux的云平台运行。Libdroid能够以基础镜像+动态加载的机制构建编译迅速、体积小巧的应用镜像，为Android代码卸载提供运行环境。

2）在云基础架构层，聚焦数据中心网络性能。为了加速网络性能同时兼顾灵活性，分别提出了基于FPGA + CPU 以及 GPU + CPU 两种加速网络功能的平台。基于FPGA+CPU的平台DHL能够将网络功能的浅包交给CPU执行，将深包处理卸载到FPGA。通过结合运用无锁缓存队列、用户态I/O、NUMA感知的内存分配、私有输出缓冲队列、部分重配置等一系列技术，实现了CPU与FPGA之间高吞吐率和低延迟的数据传输层、不同网络功能之间的数据隔离、以及网络功能的动态重配置。基于GPU+CPU的平台能够根据GPU和CPU处理数据的不同特性和当前队列负载情况选择最佳的数据包调度粒度和最优队列进行服务，从而达到提高网络吞吐率的目的。

3）在云应用层，不仅针对典型流计算应用开展了系统研究，而且面向新兴的区块链技术以及关系国计民生的智慧城市领域也进行了研究探索。在流计算方面，我们开发了一系列实际系统，包括基于预调度方法的Spark社区第三方开源项目Spark-Lever、基于环形高速缓冲的低延迟流处理系统TurboStream、基于双范式模型的新型流处理系统DO。在区块链应用方面，我们探索了不同区块链系统上数据建立高效可信连接的问题，并探讨了区块链跨链交互的一系列关键挑战和从不同分层解决这些挑战的可能思路。在智慧城市方面，通过获取、整合、分析城市数据来实时预测未来城市各区域中交通碳排放量，提取出了城市各区域中与交通碳

排放密切相关的高效特征集，并设计了基于感知神经网络的碳排放预测系统模型。

2. 云软件平台

云平台层的研究涵盖容器技术和基于轻量级虚拟化的移动云应用运行环境。

2.1 容器技术

容器技术是一种新型轻量级虚拟化技术。相比于传统虚拟机，其具有部署方便、启动快速、服务灵活、资源消耗少等特点。因此，容器正在被越来越广泛地应用于云平台中。我们对大规模容器的快速部署和运行时隔离性两个热点问题进行了研究。

2.1.1 容器镜像构建与快速启动

为了满足云平台下各种需求，容器镜像变得越来越大，已经接近甚至超过了虚拟机镜像的大小，如何快速构建所需的容器镜像就变得非常有必要。云端的Hub只提供了最基础的镜像，用户想要部署一个容器化应用，通常需要自定义自己的镜像，然后通过统一的容器编排工具进行构建管理。这种方法在容器发展初期非常便捷，但随着容器生态飞速发展，容器被应用得越来越广，容器化的应用越来越复杂，容器镜像已经达到GB级，容器的启动变得越来越耗时，严重制约了容器的发展。

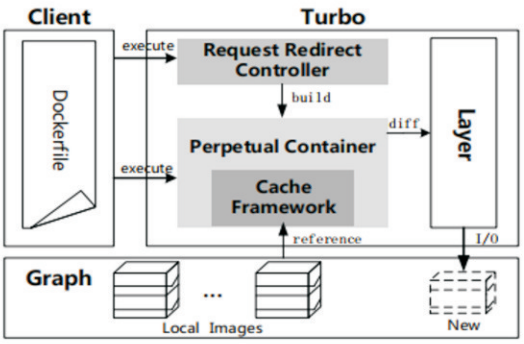


图2.1.1 Turbo系统架构

我们提出并实现一个不改变原生镜像结构并且不需要额外硬件辅助的快速构建容器镜像方法--Turbo。本研究分析了Docker官方Hub中下载次数超过10万次的所有镜像，发现这项镜像存在大量数据冗余，接近60%的数据是重复的，这意味着构建镜像过程中有超过一半的时间浪费在重复数据的下载上，而且还占用大量的主机存储空间。在DevOps场景中，应用会被不断更新，导致本地会存在大量不同版本的镜像，我们在分析了2946个镜像后发现，这种情况下镜像更新会产生大量的冗余数据，更新的数据量是实际变化数据量的12.7倍。发生以上情况的主要原因在于原生Docker的层级共享机制粒度太大，并不能够充分利用到不同镜像的冗余数据。为了解决该问题，Turbo采用了文件级缓存粒度来充分利用本地的镜像数据，并通过指令重叠的方式加速构建指令的执行。在构建镜像的初期，需要从云端拉取一个基础镜像，我们通过修改Docker本身的机制，实现了在本地构建基础镜像，加上文件级缓存和指令重叠，极大加速了镜像构建速度。相对于原生的Docker构建方式，Turbo的构建速度提升了3.1倍，节省数据存储空间41.9%。

图2.1.1显示了Turbo的系统架构。Turbo几乎重新设计了Docker的镜像构建机制，核心工作主要是三个模块：request redirect controller改变基础镜像的构建模式，优化基础镜像的获取方法；perpetual container主要完成指令重叠机制，加速指令的执行；cache framework文件级的缓存利用，避免重复数据的下载。

2.1.2 容器视图隔离机制

大量的容器部署于云服务器时为主机下多容器的隔离带来挑战。根据研究发现，目前部署在容器中的应用主要为多线程应用，而大部分的多线程应用会根据获得的底层系统信息去最大化利用系统资源，如诸多应用（JVM、OpenMP）会根据系统CPU核数来创建相应的线程以提高cpu利用率，会根据系统的Memory总

量来决定应用使用的内存数。但是多容器共享环境下，容器共享主机资源，以Cgroup来限制资源使用，但是由于共享内核，容器获得的仍然是主机的系统资源总量，这使得容器内应用做出错误的资源使用决策，导致多个容器间产生剧烈的资源竞争，使得容器内应用性能损耗严重。通过我们对DockerHub的前143个企业应用仓库的分析，53%的容器镜像内的应用会受到影响，这极大的阻碍了容器技术的发展。

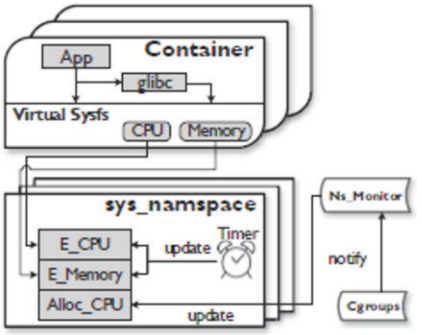


图2.1.2 容器感知的资源隔离视图

容器感知的资源隔离视图如图7所示。首先，通过修改Linux内核，实现一个新的sys_namespace，为容器提供隔离的系统资源。然后，通过kernel sysfs的改进为每个容器提供虚拟sysfs，使得每个容器获取得到的系统信息来自于sys_namespace提供。最后，实现动态实时系统资源计算算法，通过sys_namespace进行容器有效系统资源的定时动态计算与更新。另外，我们提供了两个案例分析，包括JVM和OpenMP的，如何进行应用的修改可以有效的动态使用sysfs提供的资源视图。

2.2 移动云平台

随着手机应用的丰富，其对运行环境以及计算、存储等系统资源的要求越来越高。云平台可以为手机提供丰富的系统资源。但是，传统基于虚拟机的云平台由于启动慢、开销大等问题难以适用于移动云应用。为了给移动云应用提供轻量级、定制化的运行环境，我们分别设计并实现了基于容器和unikernel的移动云平台。

2.2.1 移动云环境下的安卓定制与快速部署系统AndriodX

移动云平台存在系统定制的需求。当前存在的应用运行环境无法适应多样化的云端应用场景需求：例如，对于计算卸载场景来说，其需要云端具有轻量级的可执行环境，因此云端平台应提供尽可能精简的运行环境；而对移动应用测试场景来说，其要求云端测试平台具备完善的测试环境，以此涵盖对应用各项指标，如功能性、兼容性等的测试。

建立一个轻量的移动应用执行环境面临两个挑战：首先，从系统层面来讲，Android与Linux有着诸多区别，例如在内核特性方面，Android依赖于特殊的内核特性，如Binder进程间通信子系统、匿名共享内存等。这些差异的存在使得在移动云平台上运行Android应用是一件极具挑战的工作。其次，不同需求云端应用场景提供统一的系统定制方法。现有的系统定制技术仅仅针对某种特别的硬件或者某些需要特殊驱动程序的应用，缺乏一种统一的定制方法以及一种有效的认证与升级机制，难以适用于云端大规模且多样化的应用场景。

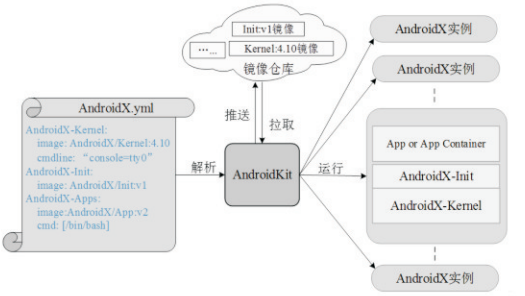


图2.2.1 Android X 系统架构图

为此，我们设计实现了一个可定制的，面向云端大规模、多样化应用场景的Android应用运行系统—Android X。其架构如图2.2.1所示。

首先，我们实现了一个用于定制系统镜像的工具集AndroidKit。该工具集能够通过解析不同的配置文件生成不同的系统镜像，并且这些镜像都是以标准的Docker镜像的方式存在。然后，我们以hypervisor + 镜像的方式将Android集成到基于Linux的云平台。当需要启动一个AndroidX实例时，通过AndroidKit将相关的镜像从镜像仓库中拉取下来并运行。为了避免共享内核所带来的硬件兼容性以及隔离性问题，AndroidX拥有独立的内核。我们采用模板定义的方式来简化内核定制流程，由于通过 AndroidKit 构建的内核是以 Docker 镜像的方式进行维护和存储，为了将最终的Kernel镜像从 Docker 镜像中提取出来，我们设计并实现了AndroidX内核过滤和提取算法，该算法通过识别tar流的头部信息中的name变量来实现内容的过滤和提取。

2.2.2 面向计算卸载的移动边缘云服务系统Libdroid

轻量级虚拟化技术Unikernel启动速度快，同时具有良好的隔离性，非常适合Android代码卸载等轻量级计算任务场景。但是，现有Unikernel都需要经过一个比较缓慢的编译过程，将每个应用编译为一个单独的二进制镜像，并且镜像一旦生成就不可改变，对应用的升级和迭代极为不方便。

为了将Unikernel顺利的应用在边缘计算场景下，我们提出了“Rich Unikernel”的概念。如图2.2.2所示，Rich Unikernel通过将更多的基础代码库打包到镜像中，降低传统Unikernel的定制化程度，从而可以实现更强的通用性，支持在同一个Unikernel镜像中运行多种不同的应用。

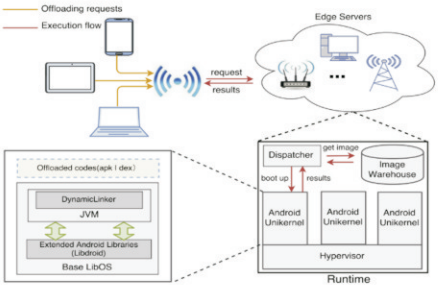


图2.2.2 Rich-Unikernel系统架构图

为了提高系统的响应速度，防止因为缓慢的镜像编译过程导致高响应延迟等问题，我们提出了一种基于预构建基础镜像和应用动态加载机制的新型Unikernel构建方法，通过此方法可以省去应用到Unikernel镜像的缓慢编译过程，实现在毫秒级将应用代码Unikernel化。对比了传统Unikernel生成过程和基于预构建基础镜像的Unikernel生成过程。传统构建方法中需要数秒到数十秒时间才能将应用编译成Unikernel镜像，我们提出的新构建方法中，本质上将传统构建方法编译过程分解为：依赖库编译和应用编译两个部分。其中依赖库编译部分，我们预先将一部分应用公共的依赖库构建到基础镜像中，待请求到达时再实时地将应用代码动态连接到镜像中，实现应用Unikernel化的目的。由于动态链接过程非常快，因此可以充分保证终端请求的响应实时性。

3. 云基础架构

云基础架构层，我们主要关注数据中心资源管理，尤其聚焦数据中心网络功能优化。随着通信网络的快速发展，对于数据中心来说，传统的使用专用硬件的网络架构已经无法满足多样化的网络需求。网络功能虚拟化技术（Network Function Virtualization, NFV）可以将多种多样的网络功能从昂贵固化的专用网元设备解耦到通用服务器上，以软件方式灵活部署与运行。然而，在性能方面，通用服务器却远远比不上专用网络硬件。为此我们分别提出了基于GPU + CPU 以及 FPGA + CPU 两种加速网络功能的平台。

3.1 基于网络功能虚拟化的流处理加速系统

为了解决NFV的性能问题，近年来一些研究工作通过将计算卸载到GPU上来提高网络功能的处理能力，在单个节点上满足了高速I/O的需求。

我们设计了一种基于GPU加速的网络流处

理系统和一种基于上述动态调度算法的网络流处理系统如图3.1所示，其实现基于多线程。该系统主要分为四个模块，IO模块，队列模块，计算模块以及调度模块。IO模块包括一个收包线程和一个发包线程，分别用于接收网卡上缓存的数据包以及将处理完毕的数据包发送到网卡上。队列模块负责维护不同网络流的数据包队列。计算模块包括计算线程和卸载线程，计算线程是从队列中取出数据包并执行CPU上的网络功能代码，处理完毕后将数据包放到下一个队列中。卸载线程从队列中取出数据包并组成一组数据包，之后将数据包发送到GPU端执行GPU上的网络功能代码，一旦GPU处理完毕就从GPU上取回处理完毕的数据包，并将这组数据包依次放入下一个队列中。调度模块实现的是我们的动态调度算法，它每隔一个设定的时间槽负责根据当前队列模块中的所有队列的长度信息，利用我们提出的队列稳定性优化决策决定哪些数据包被处理，传输和接纳。系统启动后，需要进行一些初始化工作，包括1）初始化网络功能：在我们设计的系统中，每一个网络功能实际上是一个具有统一接口的回调函数；2）初始化服务链：将不同的网络功能连在一起形成服务链；3）初始化网络流：将网络流添加到流表中，为每一条网络流分配一条服务链，并根据该服务链分配队列；4）初始化线程：为提高性能，每一个线程都绑定到不同的物理核心上。

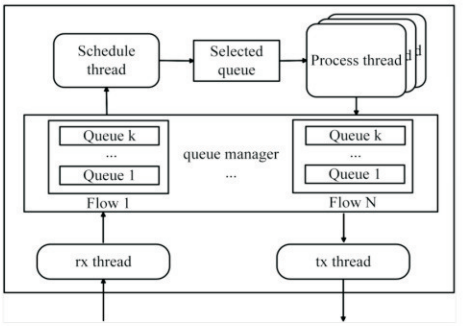


图3.1 网络流处理系统框架图

由于GPU与CPU的硬件架构和特性有很大不同，我们把CPU上的代码迁移到GPU上后，会提供以下的调度策略：1. 批的大小的调整：在CPU上我们每次处理一个数据包，而在GPU上，我们每次处理一批数据包。根据GPU的特性，批越大，越能发挥GPU并行处理的优势，减少数据传输的开销，从而提高吞吐率，然而批越大也会导致组批时间的增加，尤其是在负载较低的时候，这样就间接增加了数据包的延时，对于一些延时要求较高的网络流，可能会有影响。一种可能的解决方案是针对网络流的特性，我们基于当前的时延动态地设置一个合理的批大小，并且根据当前负载情况，动态调整批大小。2. CPU+GPU 共同处理：目前我们使用CPU和GPU协同处理模式，CPU只负责准备，发送和接收数据，不进行计算工作。使用GPU的好处是可以提高吞吐率，然而会相应的增加额外的延时，尤其是在负载较低的时候。因此我们考虑共同处理模式，即在负载低的时候使用CPU进行并行处理，在负载高的时候使用GPU进行处理来提高吞吐率。

3.2 基于FPGA的高效能网络功能加速平台

当前软件网络功能在进行深度包处理时，需要消耗大量的CPU核心和资源才能达到线速度。虽然支持高并发度和可编程性的FPGA具备加速深度包处理的可行性和潜力，但是FPGA中的可编程逻辑十分有限而且成本昂贵，因此若将整个网络功能部署到FPGA上会造成不切实际的资源浪费。此外，当网络功能需要更改时，还需要耗费数小时生成新的FPGA加速程序，阻碍网络功能的快速部署。

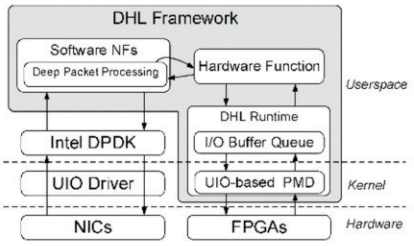


图3.2 DHL（Dynamic Hardware Library）平台的总体架构

针对上述FPGA在NFV性能加速中面临的关键挑战，我们提出和实现了基于动态硬件库（Dynamic Hardware Library, DHL）的FPGA-CPU协同系统架构（如图3.2所示），该框架为网络功能虚拟化平台保证了高性能和高灵活性：（1）该框架将深度包处理实现为FPGA中的加速模块，并将这些加速模块抽象成硬件函数库并为开发人员提供一套DHL编程API，使同类的多个网络功能的复杂逻辑统一部署在FPGA中加速，而简单逻辑仍协同运行在CPU中，从而实现多个网络功能在FPGA-CPU架构中灵活、快速、高性价比的部署，同时具备通用性和易编程的优势；（2）该框架通过结合运用无锁通信队列、用户态I/O、NUMA感知的内存分配、批处理和轮询等一系列实用技术，对DHL整体框架的网络性能进行了优化；（3）实际系统实现和测试验证表明，DHL框架极大地降低了软件开发人员使用FPGA加速网络功能的编程难度，不但拥有比纯CPU方案更高的吞吐率和更低的延迟，而且相比于纯FPGA的方案最大限度地提高了FPGA的资源利用率。

4. 典型云应用与新型应用探索

对于云应用的研究涵盖了传统的流计算系统，新兴的区块链技术以及关系国计民生的智慧城市。

4.1 流计算系统

随着物联网、社交网路、移动互联网等的发展，各种各样的实时流处理框架和系统被提出以满足对流数据低延迟、高吞吐的处理需求。我们的研究范围涵盖流计算平台的系统架构与任务调度关键技术，主要包括操作符通信优化、任务调度、计算范式设计。

4.1.1 基于慢结点预调度机制的批流系统Lever

批流系统将连续的流计算离散化为一系列小时间粒度上的微批作业，并将这些作业周期性提交给批处理引擎处理。典型的代表如Spark Streaming。当前的集群环境下，由于不同批次的硬件更新、资源共享、负载数据倾斜等原因，慢节点在批流系统中普遍存在，严重影响系统性能。传统的批处理引擎处理慢节点的技术是为大而长的批处理作业而设计，是一种反应式的策略，在批流系统中可以归结为post-scheduling方法，这类方法不能很好地适应小而短的微批作业的处理需求。

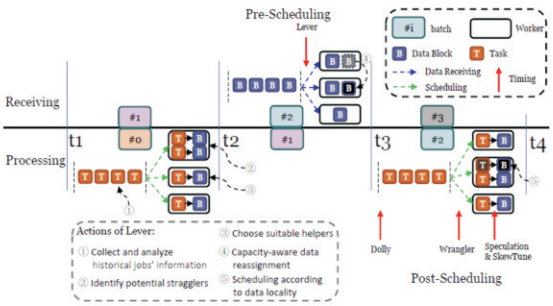


图4.1.1 Pre-scheduling处理流程

我们提出了一个pre-scheduling的策略，并实现了面向Spark的第三方开源系统Spark-Lever，这是目前Spark社区中由大陆高校贡献的唯一流处理开源项目。首先，考虑到微批作业的周期性和慢结点局部性，我们可以通过分析历史作业的运行时信息结合负载的变化识别下一个批次的潜在慢节点；其次，引入ILC（Iterative Learning Control）模型通过多批次的迭代评估节点的计算能力；最后，针对潜在的慢节点，选择合适的帮助者节点，在Batching Module中按照各帮助者节点计算能力提前对潜在的慢节点任务进行划分和分配，在最后任务调度运行时，就可以有效的避免慢结点的产生。

4.1.2 基于环形缓冲的低延迟流处理系统TurboStream

为了能处理这些高吞吐的持续数据流，以Storm为代表的“record-at-a-time”型的分布式数据流处理（DSP）系统得到了业界的广泛关注。这类系统将用户处理数据的计算逻辑封装为操作符，并将这些操作符并行化为多个任务实例分布到集群的各个结点上去执行。上下游操作

符所对应的任务之间就需要通信来传输数据。

通过实验发现操作符间的高延时是由进程间的通信引起的。改进DSP系统的进程间的通信是降低操作符间的延时的一个重要因素。

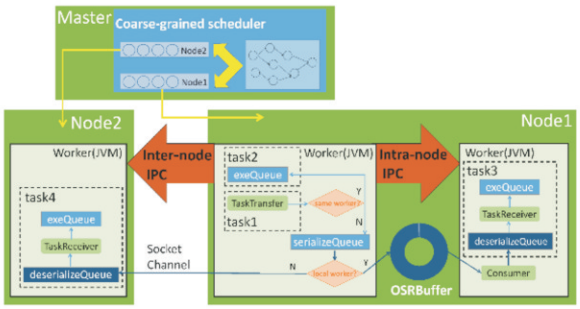


图4.1.2 Turbo Stream架构图

为了解决这一问题，我们设计了Turbo Stream。如图4.1.2所示，该系统主要包括OSRBuffer（堆外流式环形字节码缓冲区）和粗粒度调度器两部分。OSRBuffer主要用于减少在一个节点内部进程间通信的开销。传统的节点内进程间通信需要借助套接字通过虚拟设备回环（loopback device）传送给下游任务，这一过程需要多次的内存数据拷贝以及不必要的排队，增加了通信延迟，OSRBuffer通过在两个进程间共享一个环形缓冲区，并以内存映射的方式传输消息，可以减少不必要的内存拷贝和排队，大大提升节点内进程间通信的性能。粗粒度调度器主要用于降低跨越节点通信的操作符间的事件延时。该调度器可以在调度之前，根据操作符间的数据依赖关系或者通信量来对操作符进行整合，能够将通信密集型的上下游操作符整合在一起作为一个整体进行调度。这样就可以避免通信密集型的上下游操作符频繁的跨越节点进行通信，从而降低系统延迟。

4.1.3 双范式流处理系统Do

连续流系统和批流系统在计算模型、任务部署、性能表征等方面都存在较大差异，各有优劣势。

连续流系统是一种数据流范式。它先将代表计算的操作符固定在各个结点上，然后数据依次流过各个操作符并执行相应的计算逻辑。这样的系统延迟低、但同时吞吐量也低，网络传输是性能的瓶颈。批流系统是一种任务流范式。它先将数据以批量形式固定在节点上，不断分发不同的操作到各个结点上去执行。这样的系统延迟高，但同时吞吐量也高。这样的系统常常都是基于BSP模型，性能瓶颈主要在于不同的任务之间需要进行同步（BSP Barrier）。

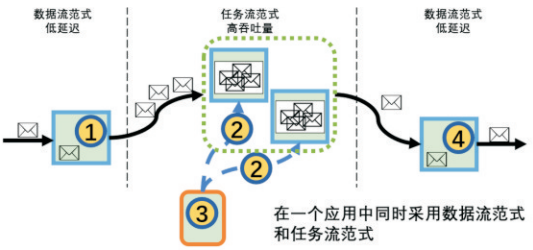


图4.1.3 双范式的流处理架构图

我们提出了一种双范式的流处理方法，并实现了相应系统DO。在一个应用中同时采用数据流范式和任务流范式以达到低延迟和高吞吐的系统性能。比如，可以将非数据洪流影响的操作使用数据流范式部署以获得低延迟，如操作1、4，将要承受数据洪流影响的操作如操作2、3采用任务流范式部署以提供高吞吐，通过调整操作所采用的范式，避免网络传输或同步延迟成为性能瓶颈。

4.2 新型应用探索

我们对区块链和智慧城市等新型应用分别进行了探索。近年来，区块链技术经历了高速发展，越来越多的区块链系统得以部署和应用。然而，跨链交互依然是区块链领域亟待解决的问题。为此我们探索了不同区块链系统上数据建立高效可信连接的问题。智慧城市旨在通过信息和通信手段感测、分析、整合城市运行核心系统的各项关键信息从而对城市各种需求做出智能响应。在这方面，我们结合物联网和云平台技术对城市碳排放量进行了预测。

4.2.1 面向跨链交互的新型区块链架构

现有区块链系统呈现数据流通性差的现象，不同区块链上的数据形成一个个数据孤岛，限制了区块链技术的进一步发展。为了在不同区块链系统上的数据之间建立高效可信的连接，我们展开针对跨链交互技术的研究。

首先，我们对“跨链交互”进行了严格定义并对现有链交互技术展开调研。我们综合考虑安全性、高效性等多方面的因素，明确给出“区块链跨链交互”的定义。在此基础上，我们提出了包括“原子性”、“高效性”、“安全性”、“兼容性”和“易开发”五个关键挑战。根据“跨链交互”定义，我们对现有的主流区块链跨链交互技术进行了全面且深入的分析和总结，将其分为了“主动模式”和“被动模式”两大类，并分析了各自的优缺点。

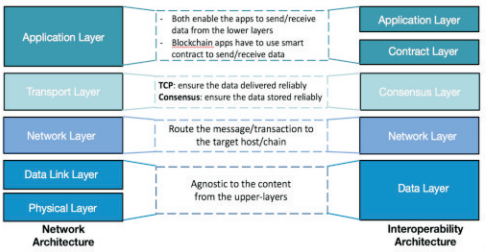


图4.2.1 区块链系统架构的分层设计

然后，我们对区块链的系统架构进行了分层设计来探讨“跨链交互”问题。如图4.2.1所示，分层包括数据层、网络层、共识层、合约层和应用层。从不同层次，我们尝试探讨了解决以上关键挑战的可能思路：1. 为解决“兼容性”的挑战，我们尝试从数据层着手，统一底层的交易格式，使一条链上的交易可以轻易地发往另一条链。对于已部署的区块链系统，难以直接改变其已有的交易格式，并尝试提供一套“交易翻译”工具，将统一的交易格式翻译成针对不同限定条件需求的交易格式。2. 为解决“高效性”的挑战，我们尝试从网络层着手，建立结构化的具有快速数据分发能力的区块链网络拓扑，并实现高效的数据广播算法。针对“被动模式”跨链交互所产生出的轮询开销，我们初步尝试设计了一种“多路复用”的被动交互方法，并通过实验验证了该方法的高效性。3. 为解决“安全性”的挑战，我们尝试从共识层着手。考虑到单链上的共识层用于维护不同节点上状态的一致性，而不同链上保存的数据状态必然是不同的，我们将链间的“共识协议”重新定义为一种特殊的“验证协议”，以验证不同链上不存在矛盾数据，从而保证跨链交互的安全性。4. 为解决“易开发”的挑战，我们尝试从合约层和应用层着手，提供一套实现跨链交互的API。该API可屏蔽底层复杂且易错的实现细节，大大降低跨链应用开发的难度，从而吸引到更多的开发者。

最后，我们尝试将传统互联网上的人物画像分析技术应用到区块链上来。由于区块链上的数据具有匿名性的特点，无法直接进行人物画像分析，我们首先对链上数据进行去匿名的处理。考虑到单条区块链上的数据量有限，我们将综合运用多链数据，以分析并绘制出出更精准的人物画像。

4.2.2 基于城市大数据的交通碳排放预测

随着我国城镇化进程的快速推进和经济的迅速增长，城市中涌现大量人流和车流，产生大规模实时、高密度的数据。针对这些结构多异的城市大数据的分析，有助于从数据中挖掘有价值的潜在城市信息，来解决各城市面临的环境难题：如何实时获取未来某段时间城市各区域里精确的交通碳排放信息。我们借助交通大数据和数据间的关联关系分析，挖掘数据与交通碳排放之间的关联关系，根据挖掘信息的特征设计合适的神经网络预测模型，预测城市各区域未来某段时间的交通碳排放信息。

我们提出一个基于三层感知神经网络（3-layerPNN）的预测系统模型来预测未来某段时间城市各区域的交通碳排放。并基于珠海市现实世界中的数据源，我们对所提出的3-layerPNN 预测系统模型进行了大量的实验和评估。预测系统

架构主要由四部分组成，分别是收集原数据、预处理、特征学习、预测及评估，如图4.2.2所示。

根据训练学习后3-layerPNN神经网络预测模型，我们使用当前某段时间内在城市各个网格区间收集提取的特征数据集作为网络模型的输入，可准确预测未来某段时间各个网格区间的交通碳排放信息。经在珠海市真实城市数据验证，系统对未来1小时内的交通碳排放预测准确度高达90.86%。

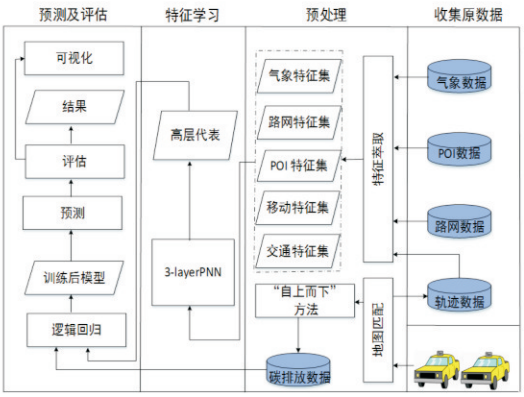


图4.2.2 交通碳排放预测系统架构

5. 总结

云计算与移动计算小组的典型成果涵盖了云软件平台、云基础设施和云应用三个不同层面，尤其是在移动云平台、数据中心资源管理和流计算系统方面有较强的研究特色，形成了一批有代表性的开源系统、论文和专利。小组除了在现有优势领域继续深入研究之外，在边缘计算、区块链等新兴领域也开展了研究探索，并取得了初步的研究成果。

附成果列表：

开源系统

[1] Spark-Lever（Spark社区开源项目）：https://spark-packages.org/package/trueyao/spark-lever

[2] AndroidX（移动云定制与部署系统）：https://github.com/CGCL-codes/AndroidX

[3] Libdroid（基于Unikernel的边缘云系统）：https://github.com/cgcl-codes/Libdroid

专利：

[1] 一种基于FPGA的网络功能加速方法和系统. 专利申请号：201810017804.0

[2] 一种调度和链接虚拟网络功能的方法及系统. 专利申请号：201711262639.7

[3] 一种基于D2D技术的移动终端设备心跳包转发方法. 专利申请号：201711122844.3

[4] 一种面向混合云应用的负载均衡方法及系统. 专利号：ZL201510245754.8. 专利授权日：2017年11月28日

[5] 一种基于Unikernel的云端Android运行环境构建方法，申请号：201711283627.2

[6] 一种面向磁盘故障检测的 SMART 阈值优化方法. 申 请号：201810006503.8

[7] 一种流数据的分布式动态处理方法. 申请号：201810006502.3，专利申请日期：2018.01.04

[8] 一种安卓系统的构建方法及其应用”，专利申请号: 201711113367.4

[9] 一种面向分布式流计算系统的动态任务调度优化方法. 发明专利申请号：201710997546.2

[10] 一种基于城市大数据预测公共汽车出行量的方法和系统，专利申请号：201710324534.3

论文：

[11] Xiaoyao Li, Xiuxiu Wang, Fangming Liu, Hong Xu. DHL: Enabling Flexible Software Network Functions with FPGA Acceleration. ICDCS 2018

[12] Weixiang Jiang, Shaolei Ren, Fangming Liu, Hai Jin. Non-IT Energy Accounting in Virtualized Datacenter. ICDCS 2018

[13] Xincai Fei, Fangming Liu, Hong Xu, Hai Jin. Adaptive VNF Scaling and Flow Routing with Proactive Demand Prediction. INFOCOM 2018

[14] Yipei Niu, Fangming Liu, Zongpeng Li. Load Balancing across Microservices. INFOCOM 2018

[15] Chaobing Zeng, Fangming Liu, Shutong Chen, Weixiang Jiang, Miao Li. Demystifying the Performance Interference of Co-located Virtual Network Functions. INFOCOM 2018

[16] Shutong Chen, Zhi Zhou, Fangming Liu, Zongpeng Li, Shaolei Ren. CloudHeat: An Efficient Online Market Mechanism for Datacenter Heat Harvesting. ACM

Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 2018

[17] Song Wu, Chao Mei, Hai Jin, Duoqiang Wang. Android Unikernel: Gearing Mobile Code Offloading Towards Edge Computing. Future Generation Computer Systems (FGCS), 2018

[18] Jiang Xiao, Zhuang Xiong, Song Wu*, Yusheng Yi, Hai Jin, and Kan Hu . Disk Failure Prediction in Data Centers via Online Learning. ICPP 2018

[19] Song Wu, Zhiyi Liu, Shadi Ibrahim, Lin Gu, Hai Jin, and Fei Chen. Dual Paradigm Stream Processing. ICPP 2018.

[20] Jiahuan Hu, Song Wu, Hai jin, and Hanhua Chen. Container Based Customization Approach for Mobile Environments on Clouds. GPC 2018

[21] Song Wu, Mi Liu, Shadi Ibrahim, Hai Jin, Lin Gu, Fei Chen, and Zhiyi Liu. TurboStream: Towards Low Latency Data Stream Processing. ICDCS 2018

[22] Hai Jin, Fei Chen, Song Wu, Yin Yao, Zhiyi Liu, Lin Gu, Yongluan Zhou. Towards Low-Latency Batched Stream Processing by Pre-Scheduling. TPDS 2018

[23] Fei Chen, Song Wu, Hai Jin, Liwei Lin, and Rui Li. Radar: Reducing Tail Latencies for Batched Stream Processing with Blank Scheduling. HPCC 2018

[24] Fei Chen, Song Wu, and Hai Jin. Network-Aware Grouping in Distributed Stream Processing Systems. ICA3PP 2018

[25] Xiangyong Lu, Kaoru Ota, Mianxiong Dong, Chen Yu, Hai Jin. Predicting Transportation Carbon Emission with Urban Big Data. TPDS 2017.

[26] Hai Jin, Xiaohai Dai, and Jiang Xiao. Towards A Novel Architecture for Enabling Interoperability Amongst Multiple Blockchains. ICDCS 2018

樊　浩

2014级博士

研究方向：I/O虚拟化

Email：u201014452@hust.edu.cn

陈　飞

2013级博士

研究方向：大规模分布式流计算系统

Email：fei_chen_2013@hust.edu.cn

王秀秀

2016级硕士研究生

研究方向：FPGA网络功能加速

Email：742418897@qq.com

鲁向拥

2015级硕士

研究方向：数据挖掘、移动计算

Email：xylu926@sina.com

黄　航

2017级博士

研究方向：容器虚拟化技术

Email：huanghang@hust.edu.cn

黄　卓

2016级博士

研究方向：容器虚拟化技术

Email：huangzhuo_mail@163.com

顾　琳

博士，讲师

研究方向：网络功能虚拟化、软件定义网络、云计算

Email：anheeno@mail.com

肖　江

副教授、博导

研究方向：分布式计算与系统、数据治理与分析、无线网络与移动计算

Email：jiangxiao@hust.edu.cn

吴　松

教授、博导，实验室副主任

研究方向：云计算、虚拟化

Email：wusong@hust.edu.cn

系统软件与体系结构组典型成果介绍

刘海坤，姚鹏程，张　宇，郑　龙，刘　博，廖小飞

关键词：内存计算，图计算，运行时，应用支撑

1 介绍

随着图计算、深度学习等大数据新型应用的普及和各种加速器（GPU、FPGA）以及存储器件（MRAM、PCRAM）不断涌现，本组旨在利用这些新型硬件为上层新型应用提供有效的支撑平台。基于所承担的863项目“内存计算系统软件研究与开发”和其他项目的支持下，在上一年度，本组主要在如下四个方面进行了研究：（1）内存计算，（2）图计算，（3）运行时系统，和（4）应用支撑。下面主要介绍本组在这四个方面研发的一系列典型系统和技术：

1）在内存计算方面，提出了一种基于对象的内存分配和迁移机制（OAM）提高访存效率，并在混合内存上实现了一个面向应用内存访问优化和降低检查点开销的预复制检查点子系统Shadow。

2）在图计算方面，针对图计算性能受限于较高的原子开销问题，研发了基于并行累加思想的图加速器AccuGraph，通过利用图计算中原子操作累加性和单一性，在保证计算结果正确性的同时实现了冲突的原子操作间的并行处理，极大提升了图加速器的性能。此外，本组也实现了一个并发图计算系统用于提高多个图分析任务在图计算平台上的吞吐率。

3）在运行时系统方面，针对基于约束求解的多线程程序并发调试技术可扩展性差的问题，开发基于图计算的超大规模并发调试框架。GraphDebugger并将传统的并发BUG分析问题转化为图遍历分析问题，采取高度优化的图计算技术（如图划分技术）解决并发调试的扩展性问题，从而显著改善多线程程序的并行分析效率。

4）在应用支撑方面，主要研究了异构内存在支撑大数据应用时的访存行为和性能，同时也研究了影响深度学习训练显存消耗的主要因素，提出了基于模型层敏感的内存复用方法Layrub以节约GPU显存消耗。另外，针对现有帧内预测算法并行度较低的问题，研究了基于CUDA的并行帧内编码机制，根据GPU中内存架构的特性以及H.264/AVC中帧内预测公式的特点，通过结合细粒度和粗粒度并行的方式大幅提升并行度，同时降低计算复杂度。该机制在多种平台上对各类视频表现都相当稳定，是一种高效的帧内编码机制。

2 内存计算系统

2.1 异构内存分配器

在平行化异构内存系统中只有将应用程序的数据放置在合适的内存介质中，才能充分利用NVM和DRAM各自的优势。现有的研究主要着重于利用页面迁移的机制来获取更好的性能和更优的能耗效率。然而这些机制都依赖于在线的页面监测，而这种监测机制需付出很大的性能开销，并且以页面为粒度进行数据迁移操作会浪费一定的DRAM带宽，并降低DRAM的利用率。本组提出了一种基于对象的内存分配和迁移机制（OAM）。OAM利用离线访存分析工具获取对象的访存模式，并用性能/能耗模型来指导应用程序的数据分配，并为有需要的数

据对象提供迁移操作。基于OAM中所提供的用于平行化异构内存系统内存分配和数据迁移的编程接口，应用程序的源码可通过OAM提供的静态代码工具自动修改。当程序开始执行后，操作系统根据OAM提供的分配建议以及当前系统中DRAM的可用量共同决定对象的放置。如图1所示，OAM主要包括离线分析（offline instrumentation，OFI）模块和动态分配（online allocation，OLA）模块两个模块，并且通过提供适用于本系统的编程接口以及基于底层虚拟机（low level virtual machine，LLVM）实现的自动化修改程序源代码的机制，将这两部分有效地结合。

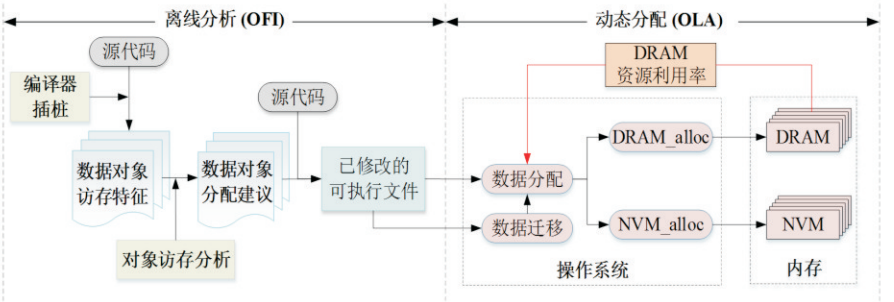


图1. 基于对象的内存分配和迁移机制总体架构

2.2 异构内存全系统检查点技术

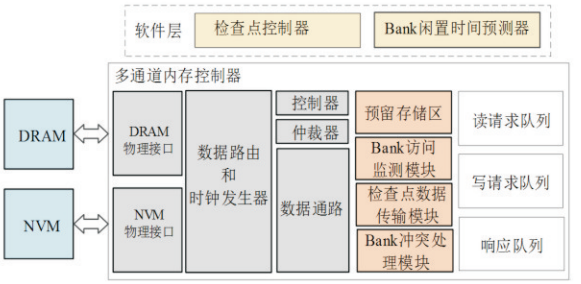


图2. Shadow系统架构

在异构内存环境下，为了减轻预复制对其他应用的内存访问干扰、降低检查点的开销，笔者对不同应用在混合内存上的访存行为进行了研究，发现多数应用访问内存Bank（一组存储颗粒的集合）的频繁程度随着时间变化有着较大的变化，并且即使在同一时间段内，不同Bank的访存频繁程度也有较大差异。根据这个特征，可以利用应用访问Bank的空闲时间完成预复制，减少检查点对应用访存的干扰，降低检查点操作的开销。基于上述思想，在混合内存上实现了一个面向应用内存访问优化、降低检查点开销的预复制检查点系统——Shadow。Shadow利用预复制检查点数据复制顺序灵活可变且对内存访问时延不敏感的特征，优化了预复制检查点的访存行为。Shadow系统需要通过软件和硬件的协同工作来实现，其总体设计原则为将复杂、高开销的预复制检查点流程控制、Bank访问频繁程度预测等功能放在软件层实现；将简单且需要底层支持的应用Bank访问信息采集、单个预复制事件数据复制、冲突预复制操作处理等功能，通过在传统内存控制器上添加新的部件，在硬件层实现。其整体结构如图2所示。具体来讲，分别在软件与硬件层面实现了以下主要功能。

在系统软件层面，主要实现了以下功能：

· 预复制检查点流程控制：该模块用于实现预复制检查点的基本功能，包括跟踪应用数据的修改、生成检查点数据复制事件、控制检查点操作的起止、维护检查点操作元数据等。

· Bank访问频繁程度预测：该模块根据硬件收集的应用Bank访问信息，通过预测算法计算得出未来应用对各个Bank访问的频繁程度，为预复制检查点流程控制模块生成新的检查点复制事件提供参考。

在硬件层面，在内存控制器上添加了以下功能部件：

· 应用Bank访问信息采集：该模块用于监测到达内存控制器的应用访存请求，记录应用对各个Bank的访问情况，为软件中的预测模块提供输入数据。

· 预复制事件数据复制：该模块从软件中的检查点控制模块接收单个预复制事件的源地址、目的地址，在内存控制器中完成检查点数据的复制，使检查点数据无须经过CPU。

· 冲突预复制操作处理：该模块监测到达内存控制器的应用访存请求，判断应用是否与预复制检查点发生Bank冲突，并通知预复制事件数据复制模块暂停或重启复制过程。

· 检查点数据寄存器：该模块用于存储检查点执行过程中产生的关键数据，包括Bank访问记录、检查点事件信息、缓存的检查点数据请求，软硬件各功能模块通过寄存器中的数据进行数据交流。

3 图计算系统软件

3.1 基于FPGA的图加速器

图结构因为其灵活的表达力被广泛的应用在各类真实的应用场景中，例如网页排序，社交网络分析等。近年来，随着图规模的逐渐增大，图计算本身一些独有的特性（包括随机访存和计算/通讯比低等特性）使得其在通用处理器上的执行效率远低于传统应用。为了解决通用处理器带来的局限性，许多研究工作致力于设计专用的图加速器以从硬件的层面为图计算提供更高的执行效率。相比于通用处理器，专用的图加速器往往能够更高效地处理图计算中的访存请求。然而在处理图计算中的计算请求时，现有的图加速器往往使用原子结构以保证计算结果的正确性。如图3所示，受限于严重的原子开销，这些图加速器依旧难以提供理想的性能。

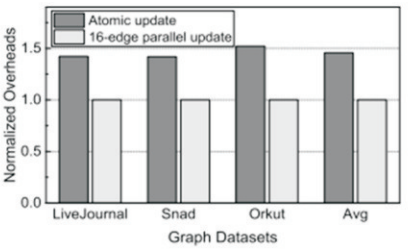


图3. 现有图加速器的原子开销

为了降低原子开销，我们提出了基于并行累加思想的图加速器AccuGraph。AccuGraph通过同时执行多条原子操作以并行化处理冲突的节点更新操作，并通过并行地累加执行结果保证最终结果的正确性。考虑到现实图结构中的节点度数往往遵循幂律分布，我们设计了一个专用的累加器在保证并行处理原子操作的同时区分处理不同度数节点。累加器通过同时执行多个低度数节点保证高效的边级并行度，并通过限制高度数节点的点级并行度避免频繁的同步操作和降低。最后，为了保证高效的数据访问，我们为AccuGraph设计了一个高吞吐的片上缓存架构以降低乱序访存对累加器效率的影响。

3.1.1 高效并行累加器

现实图结构中的节点度数往往遵循幂律分布，导致了大部分节点度数较低而少部分节点占据了绝大多数的度数。在处理低度数节点时，传统累加器架构往往在输入和输出间建立固定的映射关系以实现对输入数据的累加。然而在处理图数据时，同一轮迭代中不同节点的度数差距较大。这种固定的映射关系导致传统的累加器架构只能同时处理一个低度数节点的边数据，严重的限制了图数据的处理效率。因此，面向图计算设计的累加器需要能够感知节点度数的变化，并能够动态的调整输入输出的映射关系保证多个低度数节点的并行处理。而在处理高度数节点时，传统累加器架构在对其更新操作进行累加时需要频繁地读取和写回该节点的临时值，导致了严重的同步开销。此外，当累加器同时处理多个节点时，高度

数节点的边数据往往和其他节点的边数据往往不在一个cacheline上，产生了较多的乱序访存。

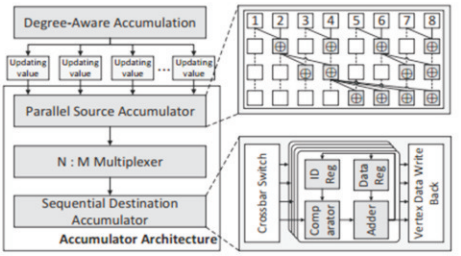


图4. 高效并行累加器架构

针对上述问题我们设计了如图4所示的并行累加器架构。其中parallel source accumulator在传统的前缀和网络的基础上针对图计算遍历的局部性特点设计了相应的断点识别机制。该机制通过在加法节点中比较输入数据的节点编号实现对累加操作的动态调整，保证了累加器可以同时处理多个任意度数的低度数节点。Multiplexer通过读取断点位置对应的数据保证对累加器产生的无用数据的高效过滤。而degree–aware accumulation和sequential destination accumuator则分别通过度数感知的节点任务调度和延迟写回机制保证了高度数节点的高效执行。

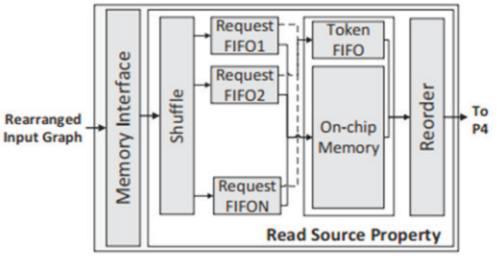


图5. 高吞吐片上缓存架构

3.1.2 高吞吐片上缓存

尽管上述累加器架构可以提供较为理想的计算性能，图计算自身不规则的访存特性依旧是限制性能的潜在因素。在处理现实图数据时，目标点的编号往往较为随机，严重降低了节点数据访问的效率，从而极大的限制了累加器的性能。为了保证累加器的吞吐量，我们设计了如图5所示的高吞吐片上缓存架构以提升节点访问的效率。为了保证每个源点所属的目标点在原始数据中分布的均衡性，我们提出了一套数据重排机制。该机制通过将目标点按照其编号模N（片上缓存划分数）的结果对原始边数据进行重排，保证其分布的均衡性。在此基础上，为了降低目标点编号本身数值的不均衡性带来的影响，我们设计一套乱序机制处理点数据访问请求。该机制通过乱序的处理不同调度轮次的点数据访问请求提升片上缓存的吞吐量，并在访存结束前按照调度轮次进行重排保证累加器的正确执行。

3.2 面向并发图分析的图计算系统

大量的图算法，或称作并发图分析任务，会以服务的形式并发地运行在同一平台上对底层图数据进行处理，以对平台上的图数据进行多方位的分析处理，获得各种目的性的分析结果，提供视频推荐、道路选择和安全审计等服务。由于并发图分析任务常需要并发地对同一图数据集进行处理。因此，并发图分析任务之间存在数据访问的时空关联性。首先，并发图分析任务们需要遍历大部分它们共享的图结构数据以对图顶点状态进行相应的处理来达到各算法目的。这意味着，多个图分析任务需要访问和处理的图数据存在大量的交集，即空间关联性。同时，多个图分析任务可能需要同时访问同一个图划分块，即时间关联性。然而，在现有图计算系统中，各图分析任务互不感知其余图分析任务和自己在数据访问需求方面的空间/时间关联性，分别按照自己需求对这些共享的图结构数据进行访问和处理。因此，使用目前方法，并发图分析任务在执行过程中会存在大量冗余数据访问，并且不能有效共用底层数据访问相关资源。由于底层硬件平台的数据访问相关资源（cache、内存和内存带宽等）有限，这些图计算系统在支持高并发图分析任务运行时将面临严重的内存墙和缓存污染等问题，致使其系统吞吐率低。因此，本课题设计了

关联性感知的并发图计算系统，以充分利用图分析任务之间数据访问空间/时间关联性和图顶点之间依赖关联性，有效地支持并发图分析任务在底层硬件平台上的执行。

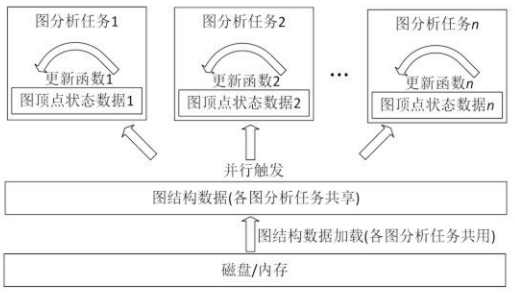


图6. 基于LTP的多任务执行模型解释

如图6所示，本系统采用基于LTP（Load，Trigger，Push）的多任务执行模型支持多图分析任务的有效执行。LTP模型将图数据分解为图结构数据和图顶点状态数据。各图分析任务都会拥有一个私有图顶点状态数据集，而图结构数据为各图分析任务所共享。然后，LTP模型将根据各图分析任务的数据访问和处理需要，反复载入各任务共享的待处理的图结构数据，触发各图分析任务根据此图结构数据和各自的更新函数（用于完成各算法目的），更新各自私有的图顶点状态直到收敛。这样，它允许各图分析任务利用它们之间的数据访问空间/时间关联性，充分共享图结构数据及其访问，提高系统吞吐率。具体而言，每轮图处理包含图加载（Load）、图并行触发处理（Trigger）和图结果推送（Push）三个阶段。

（1）图加载：在图加载阶段，它首先实时挖掘各图分析任务目前有哪些图顶点未收敛，并通过剖析各图分析任务待处理的图顶点交集得知它们之间数据访问空间关联性。然后，它以块为单位以数据流的形式顺序载入待处理的共享的图结构数据，以充分利用顺序访问带宽。每当加载一个图结构数据块（简称图划分块）到cache中，各任务中与此块对应的私有状态数据块（载入的图划分块中图顶点及其邻居图顶点对应的私有状态）也会加载进去。注意，加载过程会跳过各图分析任务已经收敛的私有状态数据块的加载和处理。当某图划分块不需要被任何图分析任务访问时，允许标记此图划分块以跳过加载。这样，它不但可以减少数据加载开销，也能避免无效数据块对cache的竞争，提高cache命中率。这里，图划分块的状态信息（包括：需要处理它的图分析任务数量和相关图分析任务ID）存储在内存的一个状态表中。

（2）图并行触发处理：当图划分块载入cache之后，它则触发需要处理此图划分块的图分析任务们对其进行处理，避免各图分析任务分别加载它。注意，一个图划分块的载入和处理需要在前一个块被对应所有任务处理完之后才能进行，否则将会导致大量独立的数据访问。然而，处理同一个图划分块的各任务所需时间可能是倾斜的，导致负载轻的图分析任务等待和计算单元空闲。为解决此问题，本课题拟采用一种动态负载均衡策略，自适应地迁移各计算单元上的任务。具体而言，当有计算单元空闲时，它允许图分析任务将其私有状态数据块进一步划分为多个片，用于在多个核上并行处理此数据块。也就是说，有些图分析任务在一个核上处理，而有些图分析任务在多个核上处理。

（3）图结果推送：在获得新状态之后，各图顶点需要在图结果推送阶段，将其值推送给其邻居图顶点用于邻居图顶点状态更新。然而，这将引起大量块外邻居图顶点所在数据块的加载，导致严重数据访问开销和缓存污染问题。为避免此问题，在此过程中，各图顶点将根据邻居图顶点*ID*在私有邻居图顶点状态数据块中累积此推送的状态值。只有当这些邻居图顶点所在的数据块载入cache用于处理时，这些累积的状态值才作为私有邻居图顶点状态数据块载入cache，用于它们的状态更新。需要注意的是，这对于图划分块内的邻居图顶点更新不

会有任何影响，因为对应的私有邻居图顶点状态数据块在cache中。

在LTP模型中，各图分析任务分别拥有私有空间存储各算法自身图顶点状态数据，而图结构数据作为全局数据被它们所共享。因此，它保证了执行过程的正确性。由于对同一个图进行分析的图算法们需要遍历它们共享的图结构数据以对各图顶点进行相应处理达到各自目标，因此它们存在数据访问空间/时间关联性。LTP模型将最需要处理的图划分块加载到内存，从而能利用时间关联性减少数据访问量。此外，LTP模型以块为单位并行触发各算法的更新函数对此块进行处理，以利用各任务数据访问空间关联性，尽量让它们共用图结构数据（实践显示其占到了整个图分析任务所需访问数据的大部分）访问。这将带来多个好处：（1）减少数据访问量；（2）避免多个图分析任务对内存带宽和I/O带宽的竞争；（3）减少cache需求，缓解cache竞争导致的cache污染问题。因此，它能最终减少各图分析任务平均访问时间，获得高吞吐率。实验显示本系统的吞吐率相对于目前系统平均能提高2.3倍以上。

4 面向并发程序的性能调优系统

多线程编程有助于充分利用多核处理器上性能，然而，由于线程的不确定性，并发程序容易出现并发错误，这可能导致严重后果。同时因为并发错误只在特定交错下表现出来，在正常执行中很少触发，所以很难重现并发错误。因此一系列研究致力于并发调试，其中，以CLAP为代表的基于约束求解的多线程程序并发调试技术能够重现隐藏的并发错误。然而，基于约束求解方法的复杂度通常随着程序执行时间的长短呈指数级增长，严重制约了并发调试的效率。特别是对于长时间运行的程序执行实例，CLAP需要几天甚至更长的时间才能重现并发错误。虽然H3减轻了这种复杂性，但从长远来看，基于约束求解的方法在可扩展性上仍然面临极大的挑战。

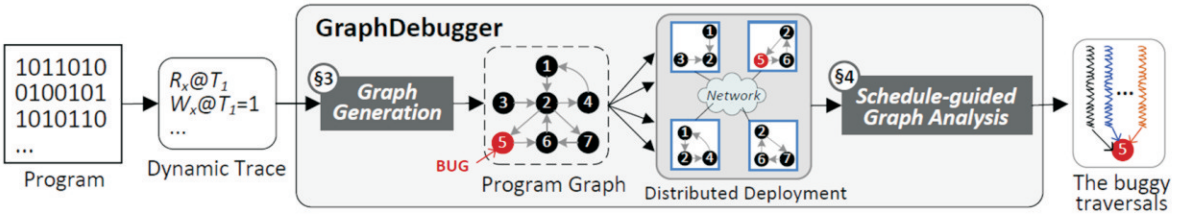


图7. GraphDebugger架构

针对上述的可扩展性问题，我们提出了一种基于图计算的超大规模并发调试框架GraphDebugger（如图7所示），其核心理念是传统的并发BUG分析问题可以转化为图遍历分析问题。由于图遍历问题在现有图计算系统中得到广泛关注与研究，因此许多高度优化的图计算技术（如图划分技术）可直接用以解决并发调试的扩展性问题。为了使程序并发分析与分布式图处理相契合，GraphDebugger实现了两个主要的部件。第一个部件将动态程序执行转换为程序图，该程序图可用于表示所有可能的线程内和线程间依赖性。第二个部件在生成的程序图上执行基于程序执行语义的图遍历算法，以发掘最大化的可行调度路径，重现并发BUG。

4.1 程序图生成技术

我们首先定义了程序图（Program Graph）是一个有向图，表示数据传播和一系列导致这些传播的算术运算。考虑到程序语句可以对一个或多个存储位置（例如，y=x和y=x+1）进行操作，为了统一表达所有情况，我们在程序图上定义该边仅仅负责表示从源顶点到目标节点的数据流，没有权值。每个顶点都包含一对属

性<关联数据，操作>。对于给定顶点，入边表示写入事件（用于更新新数据），而出边表示读取事件（用于传递其关联数据）。

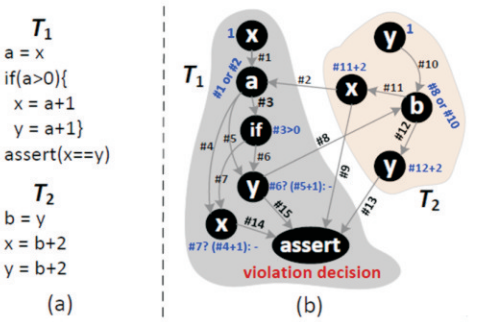


图8. 程序图示例

程序图生成需要考虑两个构造部分：顶点构造和边构造。基于动态追踪（Dynamic Trace），顶点构造相对简单。程序员只需要提取在程序执行期间访问过的所有内存位置，并进一步将它们隔离为独立的顶点。关键是如何构建适当的边以连接这些顶点。这是探索程序的线程依赖性的衍生问题，线程相关可以表现出很多种方式，①程序路径（Program Path），对于每个程序路径，我们构建一个带有数据类型是布尔值的关联数据的顶点，用于判断路径表达式是否为真，该顶点的入边来自存在数据依赖的顶点。②锁同步，对于每个lock顶点，我们从unlock顶点或lock初始化顶点构建指向它的边。③Read/Write匹配，我们为程序图中的每个可能的Read/Write对建立了从Read顶点到Write顶点的有向边。④内存顺序模型，内存顺序模型给出了如何执行线程中事件的规范。顺序一致（SC）的内存顺序强制执行由程序顺序指定的事件的执行顺序。放松内存模型（例如，PSO）允许重新排序操纵不同内存位置的事件。⑤错误显示（Failure Manifest），通常因为断言条件被违反而发生错误。与处理程序路径类似，我们构建一个布尔值的顶点，其入边来自那些存在数据依赖的顶点。如果布尔表达式的结果值更新为false，则表示发生了并发错误。基于上述的线程依赖，图8举例说明了如何生成程序图。

4.2 面向可行调度的图遍历分析

我们采用基于Push的计算模型执行图计算，该模型使用以下三个面向顶点的算子来处理每个顶点：Check：检查收到的信息是否足以执行更新。Update：执行操作以更新给定中心顶点的值。Push：中心顶点的新更新值被推送到其所有相邻顶点。

在程序图上，我们遵循基本的图迭代执行原理来模拟程序执行。也就是说，一旦其检查状态有效，程序的语句执行由顶点的更新操作进行。之后，这个更新的数据立即传播到所有相邻的顶点，重复上述过程，直到没有任何顶点未经处理。

为了捕获最大的可行调度路径集，我们使用一个简单而有效的优先级队列概念，为每个运行变量维护多个队列。在Check不满意的情况下，我们将新到达值推入相应的队列，如果条件有效，则将它们拉出。为了确保调度的可行性，还需要实施大量的顺序一致性。因此我们提出了顶点可以被更新的前提，即与待更新顶点位于同一线程中的所有相邻顶点都已完成其CUP运算，该顶点才能被更新。特别是对于SC内存模型，执行范围应该进一步扩展到同一个线程中的所有前置顶点。

```
Input  :Program Graph PG = {V, E}
V_active ← ActiveVertexInitialization(PG)
while V_active ≠ ∅ do
    foreach v ∈ V_active do
        foreach arrived data D_u do
            if sequential consistency is guaranteed then
                if Check(D_u) is satisfied then
                    v ←Update(D_u, Dequeue())
                    V_active ←Push(D_v)
                else
                    Enqueue(D_u)
            else
                Enqueue(D_u)
```

图9. 基于可行程序调度的图遍历算法

通过上述考虑，我们提出来了基于Check-Update-Push编程模型的最大图并行遍历算法（如图9所示）。首先初始化一些被push的数据，通过监视从其相邻顶点到达的每个数据来激活一组顶点。如果此数据到达满足顺序一致性要求，则执行CUP模型以并行化不同顶点之间的计算。

5 应用支撑

5.1 面向大数据的异构内存应用

大数据应用需要越来越大的内存，而新兴的NVM内存，因其优异的特性可成为内存扩展更好的选择。相比于DRAM，NVM具有较低的带宽和较高的时延，但也具有大容量和成本低廉的优势。笔者量化分析了相同成本下，异构内存环境带来的性能提升。由先前的工作可知，DRAM内存的成本约为NVM内存成本的5倍，所以使用Twitter的数据集测试和对比了同种成本情况下多种图计算应用的性能。如图10所示，相同成本情况下，异构内存系统中程序的性能明显优于纯DRAM系统，其中宽度优先搜索（breadth first search，BFS）算法、网页排序（page rank）算法和单源最短路径（single-source shortest path，SSSP）算法分别获得了3.47倍、4.52倍、6.96倍的性能提升。这是因为16 GB的DRAM内存，图数据无法全部加载到内存中处理，部分数据需要存储在磁盘中，而相同的成本，采用异构内存技术，可以拥有更大的内存，有机会使数据全部在内存中处理，而无需与磁盘进行数据交换。

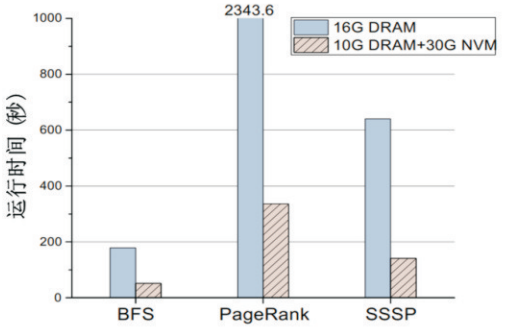


图10. 各种图计算应用下性能对比

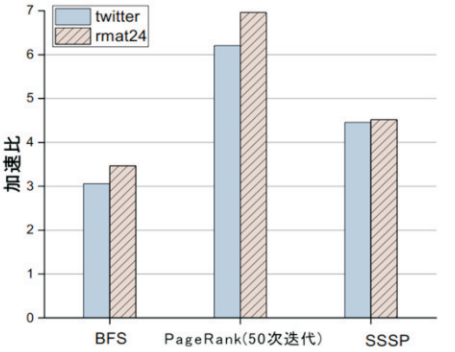


图11. 不同数据集各种图计算应用下性能对比

此外，我们测试了在不同的数据集下，不同的图算法在两种内存环境下的性能提升。如图11所示，不同的应用程序，不同的数据集，采用异构内存获得的性能各不相同，但是总得来说，相同成本下，采用异构内存可以获得一定的性能提升。笔者未考虑固态硬盘（solid state drive，SSD）和磁盘的开销，因为有了足够大的NVM内存，可以不用SSD和磁盘来存放数据。把SSD或者磁盘开销考虑进去，程序性能提升会更大。

5.2 面向深度学习的应用支撑

深度学习是以大规模的训练数据以及复杂模型作为支撑的。基于GPU的深度学习模型训练过程需要大容量的GPU内存才能有效执行。然而，目前GPU上DRAM 的容量非常有限，无法满足其训练过程中内存消耗日益增长的要求。已有大量的研究工作提出各种方法对深度神经网络训练过程中的中间数据进行剪枝、量化及压缩，能够有效地减少模型训练过程中的数据占用及内存消耗，但其对模型训练精度有比较大的影响，并增加了模型训练过程的困难。现有的一些训练框架（如TensorFlow）考虑使用模型并行的方法，将单个模型划分到多张GPU卡上，以减轻单个GPU的内存压力。然而，由于模型的复杂性，现有技术手段很难对模型进行有效划分。如何合理平衡多卡的计算负载、有效减少模型并行的通信负载都是模型并行相关工作中尚未很好解决的

难题。鉴于此，我们提出一种模型层敏感的GPU内存复用方法Layrub，以节约深度学习系统训练过程的GPU显存消耗。

为了充分理解影响训练过程内存消耗的主要因素，我们来观察一个深度学习训练过程的访存行为特征（如图12所示），我们注意到：第一，在传统的训练框架中，会在训练初始过程默认分配整个模型所需的内存空间，即为图12中的各个区域，这会导致巨大的内存开销，因此我们考虑在训练初始时不分配全部内存给模型；第二，模型激活值及其梯度值的内存占用量完全一致，如图12中的Conv和∇Conv，FC和∇FC，因此训练前向过程中为激活值所分配的内存空间可以方便地被对应的梯度值复用；第三，不同层之间的内存占用情况是相互独立的，因此我们考虑以层为中心按需分配内存空间。根据上述观察，我们提出了一种基于模型层敏感的内存复用方法Layrub。

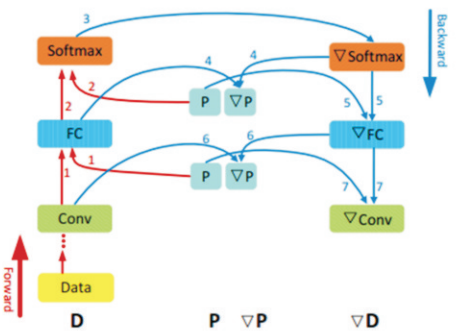


图12. 深度学习训练过程访存行为特征

（P及∇P分别为参数及其参数梯度）

首先，在不违背正常计算逻辑顺序的情况下对训练过程的数据访问顺序进行重新调整，并确保每层的参数梯度计算发生在激活值梯度计算之前。然后，根据上述数据访问重排规则，挖掘出训练过程的激活值及其梯度数据的复用可能性，提出针对模型层内中间数据复用的策略，使得训练过程的激活值梯度数据能够复用其对应激活值的内存空间，具体如图13（a）所示。该策略对于以宽度见长的神经网络效果明显。

其次，提出了针对模型跨层数据复用的策略，以层为中心按需分配内存空间，并且利用远端CPU的内存作为数据迁移复用的缓冲区域。我们提出一种内存访问计数追踪器，记录模型所有层对内存访问的次数，模型所有的层则共同复用同一片内存区域，将其他层的中间数据暂时迁移到CPU内存中，待计算到当前层时，提前将CPU内存中的数据迁回到GPU内存中，具体如图13（b）所示。在采用该策略使用内存复用后，对于任意深度的网络模型，GPU 内存分配可以只保留常数级的空间，即网络模型中的三层的空间，从而显著地节约内存开销。该策略对于以深度见长的神经网络效果明显。

最后，考虑到两种策略具有很好的兼容性，我们将两者混合使用，能够达到更好的内存节约效果，并兼顾多种形态的神经网络。具体如图14所示。

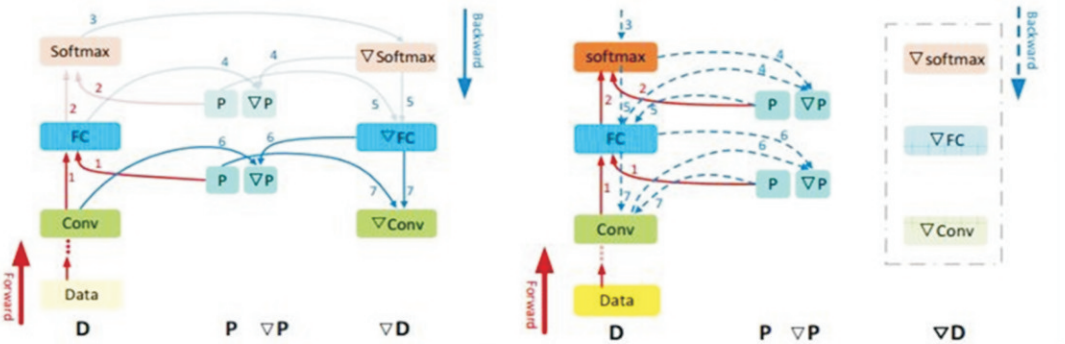


图13.（a）层内中间数据复用；（b）跨层中间数据复用

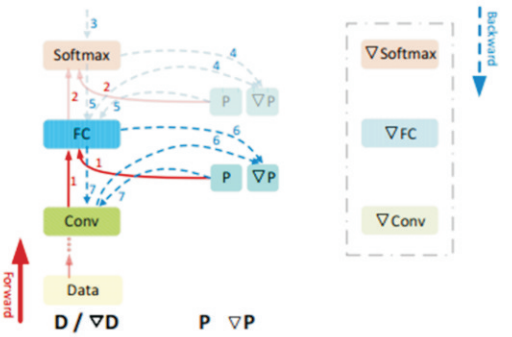


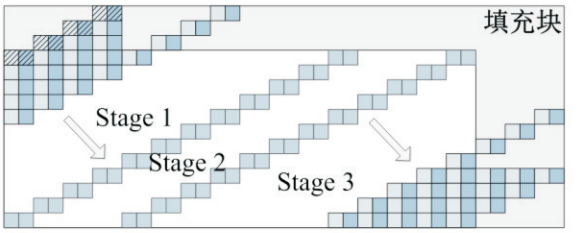
图14. 混合中间数据复用

5.3 面向H.264/AVC帧内编码的应用支撑

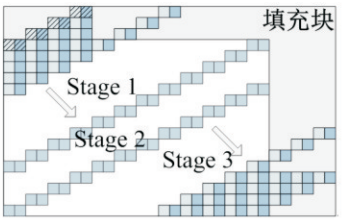
现有的帧内预测相关工作不能有效地利用GPU设备强大的计算能力，其主要原因在于算法并行度不高。这些工作大多都只是停留在帧内预测算法的粗粒度并行，如块级并行和片级并行等，对于像素级和模式级的细粒度并行很少涉及。实现基于GPU的并行帧内编码系统时主要存在以下问题，并行度不高问题、线程同步问题和内存组织问题。并行度不高会导致系统对GPU设备的利用率偏低，进而使得整体编码效率偏低。而导致并行度不高的主要原因为：块间数据依赖、块编码顺序执行、以及细粒度并行帧内预测逻辑复杂。为了实现帧内预测的高度并行，需要在不破坏块间数据依赖的情况下，最大限度地提高并行度。在线程同步问题上，线程间同步机制的具体实现对系统整体性能的影响相当大，特别是在CUDA架构上。在内存组织问题上，GPU设备上的多级内存保证了核函数的高效运行，但是这需要程序员能合理分配各级内存上所存放的数据。帧内预测过程中大量的初始数据和中间结果由于程序并行度相当高，很容易把GPU设备上的共享内存和寄存器资源消耗完。

针对这些问题，我们设计一套基于CUDA架构的帧内预测细粒度并行机制，该机制改进了现有帧内预测的执行过程。该执行过程共由四个部分组成，包括并行帧内预测、变换量化、模式决策和像素重建。为解决并行度不高的问题，提出一种基于查表的细粒度并行方法，实现了像素级和模式级的并行，将由细粒度并行产生的大量分支语句转化成为了对内存的不规则访问，避免了在GPU设备上执行过多的分支语句，提升编码效率。细粒度帧内预测算法相对于现有基于GPU的帧内预测算法最大的优势就是支持细粒度的并行。通过将细粒度并行带来的大量分支语句转换成对GPU设备共享内存的不规则访问，避免了CUDA的最小调度单元warp不善于处理分支指令的问题，极大地提高了GPU设备的有效利用率。

为进一步提高编码效率，在模式选择决策阶段，我们采用并行快速模式决策降低计算复杂度，将基于查表的并行帧内预测算法和并行快速模式决策算法配合使用，从降低计算复杂度和并行处理的两个方面减少了编码所需的时间，提升了GPU设备的有效利用率。我们对已有快速模式决策算法做出改进并使之并行化。首先将视频帧的像素矩阵从全局内存读取至共享内存中，然后计算各个方向上的密度梯度值，将8种模式的密度梯度值进一步处理，采用5步判断法，找出值最小的3个方向，将值最小的3个方向对应的模式存储至候选模式数组。



(a) Width >= 2 * Height



(b) Width < 2 * Height

图15. 改进的编码顺序

要充分利用GPU设备的性能，只靠细粒度并行是不够的，必须要把多种并行粒度结合起来使用。我们的并行帧内预测算法在实现细粒度并行的同时也实现了粗粒度并行，包括帧级并行和子宏块级并行。图15是改进后编码顺序的示意图，改进的编码顺序是基于10步法对细粒度帧内预测算法优化过后的。图15（a）中斜条纹标识的8个方框为一个处理单元，一个处理单元将由一个线程块进行处理。之所以这样安排，是因为细粒度并行帧内预测算法在结合并行快速模式决策算法之后，一个子宏块只需要64个线程，而一个线程块通常需要分配256个线程以保证GPU设备中的内存资源的充分利用以及掩盖访问内存的延迟。除了可以保证用相同数量的线程块来处理编码单元这个优势以外，采用改进的编码顺序可以有效地保证数据之间的同步。

本系统在三种配置不同级别GPU以及CPU的机器上都进行了测试。根据测试结果，采用了上述所有优化方法的系统明显优于其它帧内编码系统，相对于现有并行帧内编码系统实现了2至6倍的加速比，相对于现有串行帧内编码系统甚至可以达到7至14倍的加速比。

附成果列表:

主要专利：

[1] 一种基于深度卷积生成对抗网络的图结构数据生成方法，专利申请号201711261769.9，专利申请日2017年12月04日，邵志远，廖小飞，金海，李永强.

[2] 一种适用于核外图处理系统的异步IO执行方法及系统，专利申请号201711115579.6，专利申请日2017年11月13日，邵志远，廖小飞，金海，李浪.

[3] 一种图结构感知的图处理方法及装置，专利申请号201810199490.0，专利申请日2018年03月09日，金海，司贝贝，廖小飞，张宇，杨荡，李志威.

[4] CPU-GPU异构环境中对GPU应用的性能评估的方法及系统，专利申请号201711146155.6，专利申请日2017年11月17日，廖小飞，郑然，胡清月，金海.

代表性论文：

[1] Long Zheng, Xiaofei Liao, Hai Jin, Jieshan Zhao, and Qinggang Wang. Scalable Concurrency Debugging with Distributed Graph Processing. CGO 2018. 188-199.

[2] Zhuohui Duan, Haikun Liu, Xiaofei Liao, Hai Jin. HME: A lightweight emulator for hybrid memory. DATE 2018. 1375-1380.

[3] Long Zheng, Xiaofei Liao, Hai Jin, Bingsheng He, Jingling Xue, and Haikun Liu. Towards Concurrency Race Debugging: An Integrated Approach for Constraint Solving and Dynamic Slicing. PACT 2018, accepted.

[4] Pengcheng Yao, Long Zheng, Xiaofei Liao, Hai Jin, and Bingsheng He. An Efficient Graph Accelerator with Parallel Data Conflict Management. PACT 2018, accepted.

[5] Yu Zhang, Xiaofei Liao, Hai Jin, Lin Gu, Haikun Liu, Bingsheng He, Ligang He. CGraph: A Correlations-aware Approach for Efficient Concurrent Iterative Graph Processing. USENIX ATC 2018. 441-452.

[6] Bo Liu, Wenbin Jiang, Hai Jin, Xuanhua Shi, Yang Ma.Layrub: layer-centric GPU memory reuse and data migration in extreme-scale deep learning systems. PPOPP 2018. 405-406 (Poster).

[7] Chencheng Ye, Chen Ding, Hao Luo, Jacob Brock, Dong Chen, Hai Jin. Cache Exclusivity and Sharing: Theory and Optimization. TACO. 14, 4 (2017), 34:1-34:26.

[8] Long Zheng, Xiaofei Liao, and Hai Jin. Efficient and Scalable Graph Parallel Processing With Symbolic Execution. TACO. 15, 1 (2018), 3:1-25.

[9] Hai Jin, Bo Liu, Wenbin Jiang, Yang Ma, Xuanhua Shi, Bingsheng He, Shaofeng Zhao. Layer-Centric Memory Reuse and Data Migration for Extreme-Scale Deep Learning on Many-Core Architectures. TACO, accepted.

[10] Yu Zhang, Xiaofei Liao, Hai Jin, Lin Gu, Bing Bing Zhou. FBSGraph: Accelerating Asynchronous Graph Processing via Forward and Backward

Sweeping. TKDE. 30, 5 (2018), 895-907.
[11] Wei Zhang, Xiaofei Liao, Peng Li, Hai Jin, Li Lin, Bing Bing Zhou. Fine-Grained Scheduling in Cloud Gaming on Heterogeneous CPU-GPU Clusters. IEEE Network. 32, 1 (2018), 172-178.
[12] Long Zheng, Xiaofei Liao, Hai Jin, and Haikun Liu. Exploiting the Parallelism between Conflicting Critical Sections with Partial Reversion. TPDS. 28, 12 (2017), 3443-3457.
[13] Yu Zhang, Xiaofei Liao, Xiang Shi, Hai Jin, Bingsheng He. Efficient Disk-based Directed Graph Processing: A Strongly Connected Component Approach. TPDS. 29, 4 (2018), 830-842.
[14] Xiaofei Liao, Minhao Lin, Long Zheng, Hai Jin, and Zhiyuan Shao. Scalable Data Race Detection for Lock-Intensive Programs with Pending Period Representation. TPDS, accepted.
[15] Ran Zheng, Qingyue Hu, Hai Jin. GPUPerfML: A Performance Analytical Model based on Decision Tree for GPU Architectures. HPCC 2018. 602-609.
[16] Wenbin Jiang, Yangsong Zhang, Pai Liu, Geyan Ye, and Hai Jin. FiLayer: A Novel Fine-grained Layer-wise Parallelism Strategy for Deep Neural Networks. ICANN 2018, accepted.
[17] Qiongjie Yao, Xiaofei Liao, Hai Jin. Training deep neural network on multiple GPUs with a model averaging method. Peer-to-Peer Networking and Applications. 11, 5 (2018), 1012-1021.
[18] Wenbin Jiang, Min Long, Laurence T. Yang, Xiaobai Liu, Hai Jin, Alan L. Yuille, Ye Chi. FIPIP: A Novel Fine-Grained Parallel Partition Based Intra-Frame Prediction on Heterogeneous Many-Core Systems. FGCS. 78, 1 (2018), 316-329.
[19] Xiaofei Liao, Long Zheng, Bingsheng Zhang, Yu Zhang, Hai Jin, Xuanhua Shi, Yi Lin. Dynamic Cluster Strategy for Hierarchical Rollback-recovery Protocols in MPI HPC Applications. CCPE, accepted.
[20] Xinqiao Lv, Wei Xiao, Xiaofei Liao, Hai Jin, Yu Zhang, Qiangsheng Hua. An Effective Framework for Asynchronous Incremental Graph Processing. FCS, accepted.
[21] Fang Zhang, Xiaofei Liao, Yu Zhang, Hai Jin. PNPFI: An Efficient Parallel Frequent Itemsets Mining Algorithm. CSCWD 2018, accepted.
[22] Hai Jin, Saqib Qamar, Ran Zheng, and Parvez Ahmad. Single Binding of Data and Model Parallelisms to Parallelize Convolutional Neural Networks through Multiple Machines. Journal of Intelligent and Fuzzy Systems, accepted.

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

姚鹏程
博士生
研究方向：图计算
Email：hustyaopc@gmail.com

张　宇
博士后
研究方向：运行时系统及其优化
Email：zhyu@hust.edu.cn

郑　龙
博士后
研究方向：可重构计算机体系结构及其运行环境
Email：longzh@hust.edu.cn

刘　博
博士生
研究方向：深度学习系统
Email：borenaliu@hust.edu.cn

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

网络空间安全组典型成果介绍

马晓静，邹德清，羌卫中，代炜琦，袁　斌

关键词：云计算安全，智能漏洞检测，云计算软件定义安全，云数据安全

1 介绍

随着云计算、软件定义网络以及深度学习等技术的发展与应用，安全威胁日益严重，本组研究了相关技术的安全问题检测与防护。基于所承担的973项目“云计算安全基础理论与方法研究”和其他项目的支持下，在上一年度，本组主要在如下四个方面进行了研究：（1）云计算安全，（2）智能漏洞检测，（3）云计算软件定义安全，和（4）云数据安全。下面主要介绍本组在这四个方面研发的一系列典型系统和技术：

1）在虚拟化和云安全方面，提出了一种高效且透明的共享库隔离机制，攻击者无法通过应用程序漏洞直接执行共享库代码，可以有效的防止利用共享库的攻击行为；提出了一种函数即服务云计算下的基于SGX的安全保护方法，可以防止服务提供商非法获取计算过程中隐私数据，并且保护运行时不被用户的恶意代码破坏。在云基础设施安全方面，针对云平台虚拟机回滚后导致的软件状态不一致的问题，研发了一种针对虚拟机回滚的软件补偿系统，在预处理阶段采用代码分析软件的回滚敏感变量和控制流路径，生成回滚敏感状态表和回滚敏感控制流路径表，在软件运行时，从域外虚拟机对目标软件中的回滚敏感状态进行监控，监控到虚拟机回滚事件时对回滚敏感状态进行补偿，由此解决虚拟机回滚引起的安全问题。

2）在智能漏洞检测方面，针对传统基于专家人工定义漏洞模式的方法很难准确刻画一类漏洞特征的问题，研发了基于深度学习的漏洞检测系统VulDeePecker，在不需要人类专家定义特征的前提下，自动学习漏洞特征，来检测目标程序是否含有与库/API函数相关的漏洞，明显降低漏洞检测的漏报率和误报率。此外，提出了基于多特征分析的漏洞自动化识别方案，来降低面向缺陷追踪系统漏洞识别的漏报率和误报率。

3）在云计算软件定义安全方面，针对SDN集中控制管理租户网络与云租户自主管理网络需求冲突的问题，提出了一种面向云环境多租户网络的可信控制器架构TNGuard，设计了新的网络权限模型，剥夺了云管理员不必要的网络权限，使得云管理员无法滥用网络划分、路由路径以及网络带宽权限。基于可信计算技术，TNGuard实例的完整性管理机制保证了整个系统的启动顺序，并允许TN管理员远程验证相关网络状态的的完整性。

4）在云数据安全方面，针对云端监控视频隐私泄漏的问题，主要研究了隐私受保护的视频监控，可以在监控视频密文上检测与追踪运动物体。另外研究了同态加密应用，考虑到算法同态域实现的局限之处，提出将上下文相关的JPEG 图片解压缩算法改写为上下文无关的无分枝算法，在同态域实现了JPEG压缩图片的解压。

2 云计算安全

2.1 基于硬件虚拟化的共享库安全隔离

在现代操作系统中，应用程序与共享库属

于同一个地址空间且在调用过程中无须权限切换。这种机制为库函数调用提供了便捷性的同时也为用户安全造成了潜在的威胁。在代码复用攻击中，共享库是攻击者查找恶意指令或者函数接口的首选目标。尽管对共享库进行内存隔离是一种有效的应对措施，但是现有的方法都需要对应用程序或共享库进行重新设计并破坏共享库的共享特性，因此它们大多数都不能高效的应用于现代系统中。

为了解决共享库的安全问题，共享库隔离保护系统Libsec提出了一种高效且透明的共享库隔离机制。借助于Intel硬件虚拟化技术的可拓展页表以及相关拓展指令VMfunc，首先，Libsec 将共享库从原有的应用程序地址空间中隔离并创建出单独的共享库地址空间，避免了应用程序代码直接跳转到共享库中执行。然后，Libsec 提供了对应的地址空间切换接口，来保证隔离后共享库与其他组件之间的正常交互。通过静态插桩以及动态处理结合的方式，Libsec 避免了对应用程序的修改，并维护了共享库的共享特性。最后，由于攻击行为并未使用特定的地址空间切换接口，Libsec可通过虚拟机管理器截获的异常事件来判断共享库运行时行为的合法性。

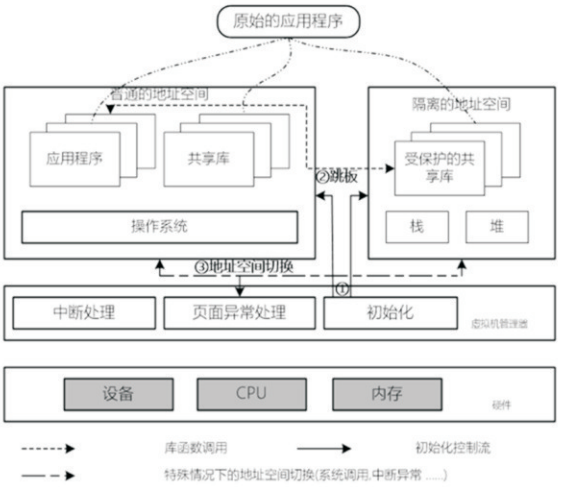


图1. Libsec架构图

Libsec被用于对Glibc标准库、OpenSSL库等共享库的保护中，实验结果表明，攻击者无法通过应用程序漏洞直接执行共享库代码，因此有效的防止了利用共享库的攻击行为。同时，性能测试结果表明将Libsec用于对底层共享库进行隔离保护时仅具有15%左右的性能开销和较小内存开销。

2.2 函数即服务云计算中基于SGX的安全保护

函数即服务（FaaS，Function as a Service）是当前新兴的一种云计算模式，具有无需维护、规模自动伸缩以及事件驱动等特点。一方面，FaaS云计算面临着与传统云计算模式相同的安全问题，如服务提供商非法获取计算过程中隐私数据、攻击者恶意篡改应用代码等。另一方面，由于FaaS云计算平台允许用户部署任意代码以及服务运行时容器环境的弱隔离性，用户的恶意代码会破坏服务运行时，从而导致云端环境面临的安全挑战更加严重。因此，需要设计一种针对FaaS云计算的安全保护机制。

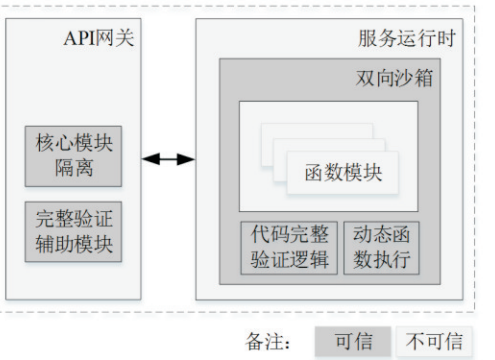


图2. Se-Lambda的架构

基于SGX保护的FaaS云计算框架，利用Intel 软件保护扩展SGX保护云服务运行时和API网关等重要组件。通过在服务运行时中引入基于SGX 的双向沙箱机制，可以防止服务提供商非法获取计算过程中隐私数据，并且保护运行时不被用户的恶意代码破坏。在双向沙箱的基础上，增强了对沙箱内函数模块的权限监控

以防止越权执行，并且优化了双向沙箱的执行周期以提高服务运行时性能。针对应用代码被攻击者恶意篡改的威胁，设计并实现了基于SGX远程认证的应用代码完整性验证方法。最后，为防止API网关计算过程中隐私数据被窃取，利用SGX隔离API网关中的重要模块。

结果表明，基于双向沙箱的服务运行时可以保护计算过程中隐私数据不被泄露并且可以防止用户的恶意代码进行破坏。应用代码完整性验证可以有效地检测到代码是否被篡改。基于SGX的API网关隔离保护机制可以防止攻击者窃取网关计算过程中隐私数据。双向沙箱对服务运行时执行时间的影响较小。

2.3 一种针对虚拟机回滚的软件补偿系统

虚拟机回滚作为一项重要的虚拟化技术在虚拟化平台虚拟机管理方面起到了至关重要的作用。然而在虚拟机回滚带来的旧状态造成安全问题的方面却鲜有关注。软件的状态补偿保护技术能够在回滚发生后，针对回滚对软件的变量状态以及控制流状态产生的不良影响进行状态补偿，保护软件在回滚后的安全。现有的软件状态补偿保护的方案存在如下不足：过度的程序员的参与要求使得现存系统的应用推广存在障碍。采取允许重放用户请求的方式恢复状态导致只能够应用于确定性的软件。其次被动的恢复状态使得软件在状态恢复连续之前，都将处于不接受新请求的状态，影响了程序的可用性。

该发明提供了一种针对虚拟机回滚的软件补偿方法及系统，其目的在于线下的预处理阶段采用代码分析软件的回滚敏感变量和控制流路径，生成回滚敏感状态表和回滚敏感控制流路径表，在软件运行时，从域外虚拟机对目标软件中的回滚敏感状态进行监控，监控到虚拟机回滚事件时对回滚敏感状态进行补偿，由此解决现有回滚补充技术存在的对使用者技术要求高和会中断保护软件运行等问题。

该方法包括：（1）由目标软件源码生成代码信息，代码信息包括控制流信息，程序依赖信息以及抽象语法信息；（2）分析目标软件代码信息，抽取目标软件中的非确定性分支条件变量，将其作为回滚敏感变量加入回滚敏感变量表；（3）分析目标软件代码信息，抽取目标软件中每个控制流节点和最近的支配节点，支配节点必须是回滚敏感变量所控制的分支节点且入度小于1，将当前控制流节点和支配节点的对应关系加入回滚敏感控制流表；（4）目标软件运行过程中，从虚拟机外部持续监控虚拟机中目标软件内存空间中回滚敏感变量的变化并记录；（5）监控回滚事件，发现虚拟机回滚时，回滚敏感变量表中具有随机性的回滚敏感变量保持随机，其他回滚敏感变量恢复为回滚发生前的最新状态;若被恢复的回滚点在回滚敏感控制流表中，则将软件控制流调整到回滚点在回滚敏感控制流表中对应点处。

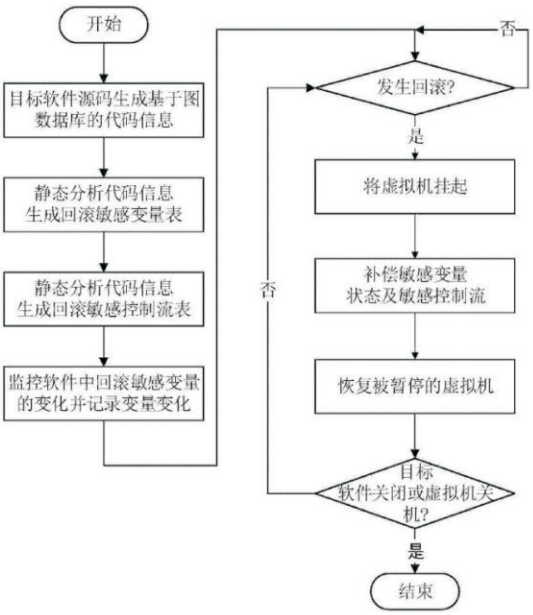


图3. 针对虚拟机回滚的软件补偿方法流程图

2.4 面向云环境多租户的网络权限隔离与划分

随着云技术的不断发展和日渐完善，许多

企业纷纷把自己的IT基础设施外包给IaaS云提供商以节省成本。为了实现网络虚拟化以及方便云租户网络自主管理，有人提出了虚拟软件定义网络 (Virtualized SDN, 简称vSDN) 技术，然而，现有的vSDN技术存在以下的技术问题：首先，违背最小特权原则，即SDN管理器为云管理员提供了过大的网络特权，这意味着无论是云管理员本身成为内部攻击者，还是他所使用的管理终端被恶意入侵，都会威胁所有租户网络的安全；其次，违背职责分离原则，即vSDN 控制器本来由TN管理员使用以管理租户网络，而却由云管理员分配，使得vSDN控制器的完整性易遭到恶意云管理员的破坏。

针对现有技术的以上缺陷或改进需求，课题提供了一种面向云环境多租户网络的可信控制器架构及其操作方法。如图14，面向云环境多租户网络的可信控制器架构包括云管理员域、多个TN管理员域、以及控制域，云管理员域与外部的云管理员通信连接，TN管理员域与外部的TN管理员通信连接，控制域与云管理员域、TN管理员域、以及数据层通信连接。控制域包括网络控制器和TN管理器。

面向云环境多租户网络的可信控制器架构的操作方法，包括以下步骤：（1）云管理员域和控制域启动；（2）云管理员域在接收到来自于新租户的租户网络创建请求以后创建TN管理员域，同时控制域根据该租户网络创建请求中的网络需求创建对应的租户网络，并将租户网络中的虚拟资源与数据层中的实际资源之间的映射关系记录在其TN管理器的虚实映射表中；（3）TN管理员域在TN管理员登陆后运行来自于该TN管理员的TN管理应用程序，以生成网络命令，并将该网络命令发送到控制域；（4）控制域使用其网络监视器判断接收到的网络命令是否是符合权限，如果符合权限则转入步骤（5）否则拒绝执行该网络命令；（5）控制域将网络命令中的虚拟资源转换为实际资源，并将转换后的网络命令发送到数据层；（6）云管理员域在接收到来自于TN管理员的TN释放请求后回收租户网络中的虚拟资源，并销毁TN管理员域；（7）控制域在TN管理器的虚实映射表中删除租户网络中的虚拟资源与数据层中的实际资源之间的映射关系，并从数据层中删除对应的网络命令。

总体而言，课题所构思的以上技术方案与现有技术相比，能够取得下列有益效果：控制器架构能解决现有vSDN网络架构中存在的违背最小特权原则和职责分离原则题，其次，通用性好，开销低。

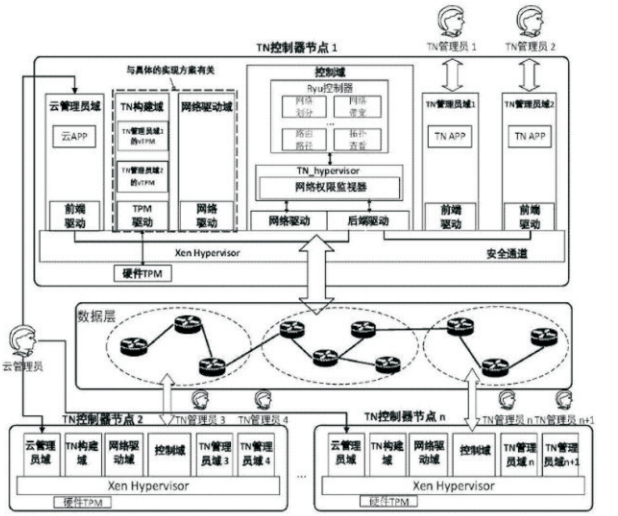


图4. 面向云环境多租户网络的可信控制器架构图

3 智能漏洞检测

3.1 基于深度学习的软件漏洞检测

软件漏洞的自动检测是一个重要的研究问题。现有的漏洞静态分析方法存在两个问题：第一，依赖人类专家定义漏洞特征。由于漏洞特征复杂，即使对专家而言也是一个冗长乏味、主观性强、且易出错的工作。不同专家定

义的漏洞特征可能不同，漏洞特征的质量决定了漏洞检测系统的有效性。第二，现有的漏洞检测方法漏报较高。一个具有高误报的漏洞检测系统是不可用的，而具有高漏报的漏洞检测系统是无用的。理想的漏洞检测系统是同时满足低误报和低漏报，但通常二者很难同时满足，更好的处理方法是强调低漏报，只要误报在可接受的范围内。

为了解决上述问题，首次开展基于深度学习的漏洞检测研究。该方法具有很大潜力，因为深度学习不需要人类专家定义特征，但同时也具有挑战，因为深度学习不是为漏洞检测这种应用而产生的。主要探讨了将深度学习用于漏洞检测的指导原则，包括将深度学习用于漏洞检测的程序表征、代码粒度以及神经网络的选择。具体来说，采用code gadget来表征程序，其中code gadget是语义相关的多行代码（可以不连续），通过编码为向量作为深度学习模型的输入。

以候选代码段code gadget为粒度，提出了基于深度学习的漏洞检测系统VulDeePecker，整体结构如图5所示。基于双向长短期记忆网络（BLSTM）自动学习生成漏洞模式，在不需人类专家定义特征的前提下，自动检测目标程序是否含有漏洞，并给出漏洞代码的位置。包括两个阶段：学习阶段和检测阶段。学习阶段针对训练程序，包括以下4个步骤：

· Step I：生成code gadget。首先提取库/API 函数调用，然后针对库/API函数调用的每个参数提取一个或多个程序切片，最后将针对同一个库/API函数调用的多个切片组合成为一个code gadget。

· Step II：为code gadget构建ground truth标签。根据已知的漏洞信息及漏洞位置，给生成的code gadget加漏洞标签或无漏洞标签。

· Step III：将code gadget转换为向量。首先将code gadget转换为符号表征来容纳更多的语义信息，然后将符号表征转换为向量，作为BLSTM神经网络的输入。

· Step IV：训练BLSTM神经网络。将转换为向量的code gadget及其标签输入到标准的BLSTM 神经网络进行训练。

检测阶段针对目标程序，包括3个步骤Step V-VII，其中Step V与Step I类似，Step VI与Step II类似，Step VII如下：

· Step VII：分类code gadget。采用学习阶段训练好的BLSTM神经网络，对目标程序的code gadget进行分类，若分类为1，则为有漏洞，否则为无漏洞。

针对能否同时处理多类漏洞、人类经验能否改进有效性、与其他静态检测方法的有效性

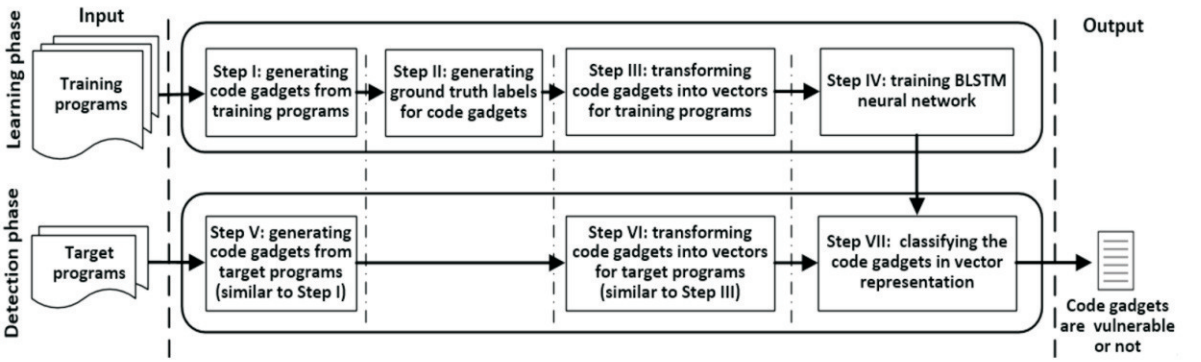


图5. VulDeePecker系统结构

比较3个方面对VulDeePecker的有效性进行了评价。实验结果表明，VulDeePecker可以应用到多类漏洞，其有效性与安全相关库/API函数的个数有关；人类经验可用于选择和安全有关的库/API函数，能够改进VulDeePecker的有效性；VulDeePecker比人工定义规则的静态分析工具（开源工具和商业工具）更有效，比基于代码相似性的漏洞检测方法具有更低的漏报，其有效性受数据量的影响。VulDeePecker在Xen, Seamonkey和Libav 3个软件中检测到4个在美国国家漏洞库NVD中未公布的漏洞，这些漏洞在相应软件的后续版本中进行了默默修补。而这些漏洞几乎未能被其他漏洞检测系统检测到。更准确地说，一个漏洞检测系统检测出了4个漏洞中的1个，漏掉了3个，而其他漏洞检测系统漏掉了全部。

基于美国国家标准与技术研究院的National Vulnerability Database（NVD）和Software Assurance Reference Dataset（SARD），构建了第一个用于深度学习的漏洞检测系统的数据集，共61,638个code gadget，包括17,725个漏洞code gadget和43,913个无漏洞code gadget。其中，漏洞code gadget包括10,440个缓冲区错误漏洞code gadget和7,285个资源管理错误漏洞code gadget。

3.2 基于多特征分析的漏洞自动化识别

很多软件开发商借助缺陷追踪系统完善其软件开发过程，通过审核缺陷追踪系统中的缺陷报告可以高效地检测漏洞。然而，现有的面向缺陷追踪系统的漏洞识别方案大多只考虑了缺陷报告的单一特性，导致其漏报和误报严重。

为了解决上述问题，提出了基于多特征分析的漏洞自动化识别方案SBRer，自动识别出缺陷报告是否与安全漏洞相关（即漏洞报告或非漏洞报告）。SBRer提取了缺陷报告中多种有效信息，包括预定义非文本字段（元特征），缺陷报告的文本内容（文本特征）和缺陷补丁文件的代码属性（代码特征）。基于这些特征，通过自然语言处理技术和机器学习算法构建自动化漏洞识别模型。与现有的自动化识别方案相比，SBRer在保持高精确率的同时，将召回率提高了22.9%~175.5%。

SBRer的整体结构如图6所示，包括两个阶段：学习阶段和识别阶段。在学习阶段，SBRer从缺陷追踪系统中收集一系列的缺陷报告作为输入，其中一部分缺陷报告与安全漏洞有关（即漏洞报告），另一部分则与安全漏洞无关（即非漏洞报告）；学习阶段的输出是漏洞自动化识别模型。首先从缺陷追踪系统中收集缺陷报告（包括漏洞报告和非漏洞报告）作为训练集，然后提取训练集中缺陷报告的各类特征，并生成对应的特征向量空间。基于特征向量空间，SBRer通过训练机器学习模型，最终得到一个漏洞自动化识别模型。具体包括3个步骤。

· Step 1：待训练缺陷报告特征提取。特征是多个可量化的缺陷报告特性，这些特性可以用于区分缺陷报告的类型。涉及3种类型的特征，即元特征，文本特征和代码特征，从待训练缺陷报告中提取得到。

· Step 2：待训练缺陷报告特征向量生成。Step 1中提取的多类特征用于生成每个缺陷报告的特征向量。每一个特征向量有一个对应的标签表示缺陷报告的类型（“1”对应漏洞报告，“0”对应非漏洞报告）。

· Step 3：自动化识别模型构建。基于训练集中带标签的缺陷报告特征向量构建漏洞自动化识别模型。识别模型是一个机器学习模型，其训练过程是标准的。

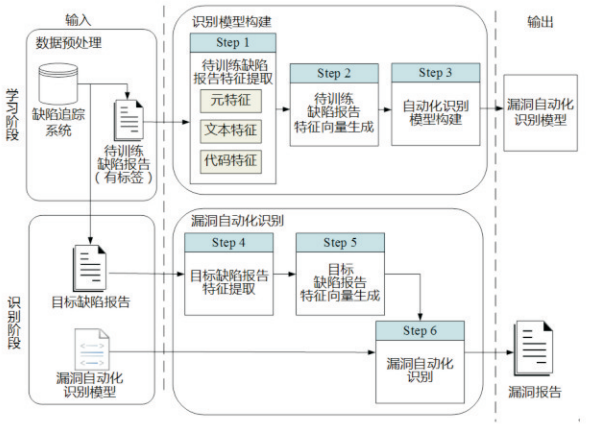


图6. SBRer整体架构

在识别阶段，将目标缺陷报告输入SBRer，然后SBRer提取目标缺陷报告的特征并生成对应的特征向量，漏洞自动化识别模型经过分类识别目标缺陷报告的类型，最终输出已识别的漏洞报告。具体包括3个步骤。

· Step 4：目标缺陷报告特征提取。从目标缺陷报告中提取出三种类型的特征，即元特征、文本特征和代码特征（类似于Step 1）。

· Step 5：目标缺陷报告特征向量生成。将Step 4中提取的特征被转换成向量表示（类似于Step 2）。

· Step 6：漏洞自动化识别。通过在学习阶段中训练好的机器学习模型，来识别与缺陷报告相对应的特征向量类型。当特征向量被识别为“1”时，这意味着相应的缺陷报告是一个漏洞报告。否则，相应的缺陷报告被标识为“0”（即非漏洞报告）。

4 云计算软件定义安全

4.1 软件定义网络数据层中内部错误的自动容忍机制

由于集中式的控制方式，在诸多影响软件定义网络的可靠性因素中，控制器产生的网络决策的正确与否直接影响整个网络的可靠性。在软件定义网络中，网络决策的生成依赖于控制器的输入信息，即数据层中交换机提供的各类网络状态数据。然而，数据层中的软硬件故障、网络攻击、掉电以及误操作等都可能导致数据层内部出现错误，而错误的内部交换机会导致控制器得到错误的输入信息，从而导致错误的网络决策。然而，现有工作都忽略了该问题，大都假设数据层是可信的，不存在错误节点。在实际软件定义网络系统中，控制器完全有可能被错误交换机误导，产生错误的网络策略。一方面，错误交换机严重威胁着软件定义网络的可靠性；另一方面，现今的软件定义网络系统的规模往往很大，人工管理方式的难度和成本都极大。因此，亟需解决错误交换机导致控制器做出错误网络决策的问题。

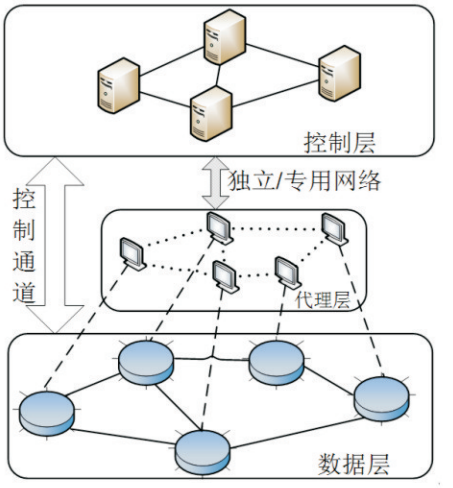


图7. 基于逻辑代理的错误交换机容忍框架

为解决上述问题， 在分析内部错误的诱因和后果的基础上，提出了基于拜占庭模型的恶意交换机容忍方法；进而，结合软件定义网络和拜占庭容错模型的特点，对原生拜占庭容错模型进行修改，设计了如图7所示的基于逻辑代理的错误交换机容忍框架，在软件定义网络架构中引入一个代理层。代理层是位于控制层和数据层之间的一个逻辑层。当控制器发出网络流量统计信息查询请求时，代理层从相应

的交换机上获取统计结果，执行拜占庭容错协议，最后把正确结果返回给控制器。使用逻辑代理层来实现复杂的拜占庭容错协议的设计使得系统不会对交换机引入额外的工作负载，以实现适用于软件定义网络的高效拜占庭容错方案。通过与其他方案的详尽实验对比与结果分析，证明了所提出的系统能有效保证控制器输入信息的正确性，保障网络决策的可靠性，从而增强软件定义网络的可靠性。

4.2 面向多租户的细粒度网络设备权限管理

尽管软件定义网络SDN（Software-Defined Networking）从提出至今已经被广泛认可多年，但是安全性问题阻止了SDN在真实网络环境中的推广与部署，其中控制器的权限管理问题尤为关键。已有工作针对控制器的权限进行了API级的划分，利用静态的权限表定义租户的应用可以访问控制器的某些API或者API中的某些字段，从而控制用户对整个网络中的SDN交换机的访问权限。然而已有工作未考虑多租户场景，静态权限表无法支持租户在动态加入与离开时的动态权限分配；并且不同租户实际可管理的网络可能只是整体网络的某个分片，甚至不同用户对同一SDN交换机的访问权限也存在差异。除此之外，SDN与网络功能虚拟化NFV（Network Functions Virtualization）的融合趋势意味着网络中将同时存在着SDN交换机与网络功能，因此对这些设备的访问权限进行统一、便捷的管理不可或缺。

针对以上不足与需求，如图8所示，提出了一种SDN和NFV环境下的细粒度的多租户权限管理框架。由管理员通过细粒度的三级权限（设备级、API级、参数级）脚本语言来配置不同租户的初始应用权限，同时支持SDN交换机以及多种网络功能如防火墙、代理网关、入侵检测系统等设备的权限列表定制。用户参考该初始权限列表进行应用编写，在用户应用运行时将自动进行权限监控，并基于不同租户实际可管理的网络分片进行权限隔离。当用户权限需要调整时，管理员通过动态修改权限配置可实现不同租户的动态权限调整，并且自动检测调整后的权限是否与已有配置产生冲突，保证了多租户环境下的SDN交换机、网络功能权限管理的细粒度、有效性和安全性。

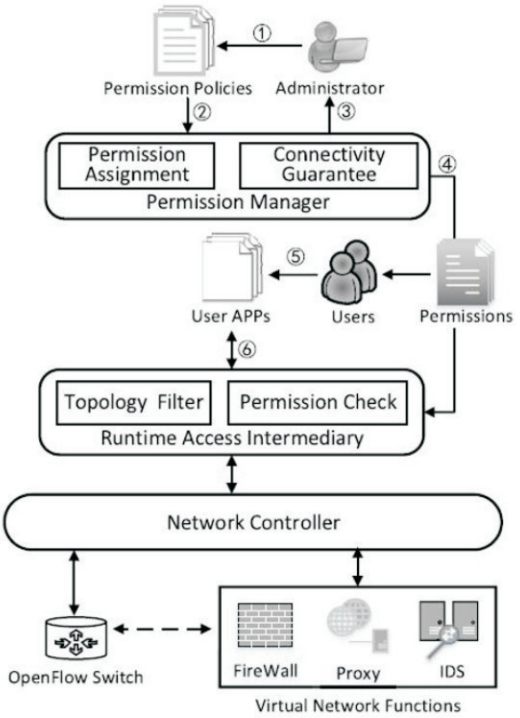


图8. 面向多租户的细粒度网络设备权限管理框架

5 云数据安全

5.1 隐私受保护的监控视频密文运动检测

云视频监控系统因其普遍性、可扩展性和稳健性而受到越来越多的关注。但是，云端存储的监控视频包含很多个人或者组织的隐私信息，这些信息面临着被窃取的危险，因此云端应该保护监控视频免受隐私泄露，目前常采用的做法是在监控视频上传到云端之前对其进行加密；另一方面，利用云的计算能力来处理和分析监控视频是一种趋势，其中运动检测和追

踪是云视频监控系统重要的任务。

总之，云视频监控系统需要在保护视频隐私的同时仍允许云端进行运动检测和追踪。现有技术允许在加密视频进行运动检测和追踪，其中一种解决方案在像素域进行运动检测，但是过高的计算复杂度很难满足当前应用的即时性。近来一些解决方案提出在压缩域上进行加密和运动检测，但是在加密视频中检测的运动轨迹所有细节完全被暴露。监控视频密文运动物体检测研究在最新的H.265/HEVC视频编码技术基础上提出了一种视频监控系统，该系统支持粒度可调整的运动检测和跟踪，用于对隐私保护和运动检测精度之间的冲突要求进行灵活的粒度控制。视频加密技术如图9所示，在常规格式兼容的码流选择性加密算法的基础上添加了扰乱层次的加密，以确保只能检测到给定的粒度下的运动信息。同时该算法可逆，以确保授权用户能够从加密视频中解密还原原始视频。运动检测方法如图10所示预测局部运动和全局运动，最终推断运动物体的轨迹。实验结果系统在运动检测和跟踪方面具有很高的精度，同时加密算法对底层视频压缩的比特率和处理速度影响非常小。

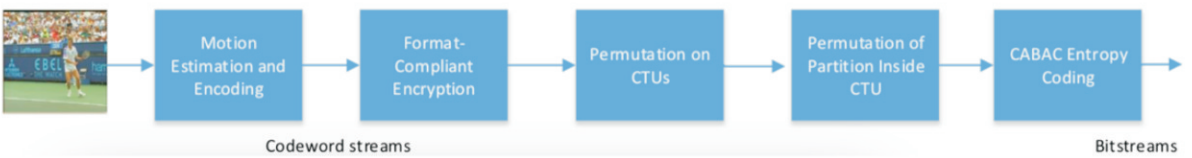


图9. 监控视频加密流程

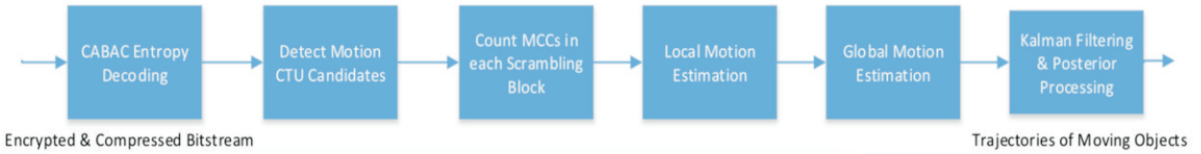


图10. 监控视频密文运动物体检测与追踪流程

5.2 同态域JPEG图片解压缩

云计算时代，传统计算模式发生转变，数据拥有者与数据使用者分离，且智能终端在日常生活中的普及导致数据内容渐趋私人化，导致隐私泄漏风险增大。同态加密支持密文上的运算，JPEG图片需在同态域解压缩后进行处理。

由于JPEG解压缩过程上下文紧密相关，难以直接在同态域实现，我们提出将JPEG图片基于上下文内容的解压缩算法重写为无分枝的算法，如图11所示，并最终在同态密文域实现。

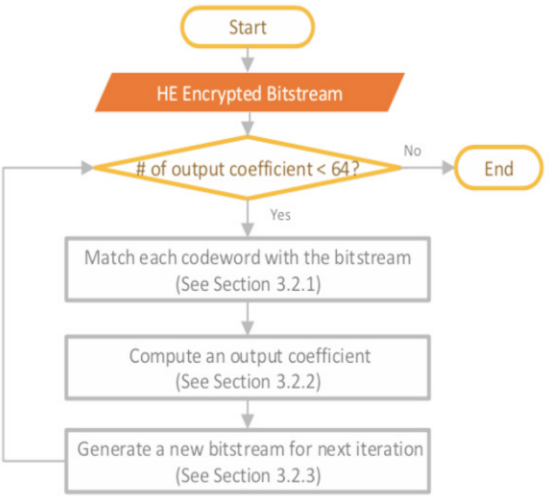


图11. JPEG图片8*8块同态域解压缩流程

附成果列表

主要专利：

[1] 一种基于TrustZone的共享库安全隔离方法及系统，专利申请号201810249221.0，专利申请日2018年03月22日，金海，代炜琦，邓俊

[2] 一种面向云租户的隐私数据保护方法及系统，专利申请号201810182419.1，专利申请日2018年03月06日，金海，代炜琦，夏妍，邹德清

[3] 一种可搜索公钥加密方法、采用该方法的系统和服务器，专利申请号201810007241.7，专利申请日2018年01月04日，徐鹏，金海，贺双洪，邹德清

[4] 一种基于数据流监控的代码复用攻击防御方法及系统，专利申请号201711257780.8，专利申请日2017年12月04日，金海，邹德清，刘本熙

[5] 一种基于扩展vSDN的网络功能共享方法及系统，专利申请号201711164841.6，专利申请日2017年11月21日，金海，邹德清，卢宇

[6] 一种软件漏洞检测方法、分级响应方法及软件漏洞检测系统，专利申请号201711114150.5，专利申请日期2017年11月13日，金海，邹德清，赵健，吴月明

[7] 一种基于深度学习的漏洞检测方法及系统，专利申请号201711089352.9，专利申请日2017年11月8日，金海，邹德清，李珍，王苏娟

[8] 一种基于多特征的安全缺陷识别系统和方法，专利申请号201711037584.X，专利申请日2017年10月26日，金海，邹德清，邓之珺

[9] 一种Intel SGX状态一致保护方法和系统，专利申请号201810058723.5，专利申请日2018年01月22日，羌卫中，金海，董泽照

[10] 一种基于中间语言分析的控制流完整性保护方法及系统，专利申请号201711236825.3，专利申请日2017年11月30日，金海，羌卫中，王世振

[11] 一种Linux内核数据攻击的检测方法、防御方法及系统，专利申请号201711205897.1，专利申请日2017年11月27日，金海，羌卫中，杨嘉玮

[12] 一种混合型可搜索加密方法和系统，专利申请号201810051562.7，专利申请日2018年01月19日，金海，徐鹏，陈天阳，陶冶

[13] 一种HEVC/H.265视频加密方法和视频密文运动检测方法，专利申请号201711314499.3，专利申请日2017年12月12日，金海，马晓静，彭欢

[14] 一种隐私保护下的视频运动物体轨迹跟踪方法及系统，专利申请号201711184418.2，专利申请日2017年11月23日，金海，马晓静，曹思行

代表性论文：

[1] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, Yuyi Zhong, VulDeePecker: A Deep Learning-Based System for Vulnerability Detection, NDSS 2018 (Top 80)

[2] Xiaojing Ma, Changming Liu, Bin Zhu, Sixing Cao, JPEG Decompression in the Homomorphic Encryption Domain, MM 2018 (Top 80).

[3] Weizhong Qiang, Zezhao Dong and Hai Jin: Se-Lambda: Securing Privacy-Sensitive Serverless Applications using SGX Enclave, SecureComm 2018.

[4] Xiaojing Ma, Bin Zhu, Tao Zhang, Sixing Cao and Hai Jin, Efficient Privacy-Preserving Motion Detection for HEVC Compressed Video in Cloud Video Surveillance, 2018 IEEE Conference on Computer Communications Workshops (INFOCOM Workshop): BigSecurity 18: The Fourth International Workshop on Security and Privacy in Big Data.

[5] Xiaojing Ma, Huan Peng, Hai Jin and Bin Zhu, Privacy-Preserving Cloud-based Video Surveillance with Adjustable Granularity of Privacy Protection, ICIP 2018.

[6] Deqing Zou, Zhijun Deng, Zhen Li, Hai Jin, Automatically Identifying Security Bug Reports via Multitype Features Analysis, ACISP 2018.

[7] Weiqi Dai, Yukun Du, Hai Jin, Weizhong Qiang, Deqing Zou, Shouhuai Xu, Zhongze Liu, RollSec:

Automatically Secure Software States Against General Rollback, NPC 2017.

[8] Weizhong Qiang, Yong Cao, Weiqi Dai, Deqing Zou, Hai Jin, Benxi Liu, Libsec: A Hardware Virtualization-Based Isolation for Shared Library, HPCC 2017.

[9] Bin Yuan, Hai Jin, Deqing Zou, Laurence Tianruo Yang, Shui Yu: A Practical Byzantine-Based Approach for Faulty Switch Tolerance in Software-Defined Networks. IEEE Trans. Network and Service Management 15(2): 825-839 (2018)

[10] Xiaojing Ma, Laurence T. Yang, Yang Xiang, Wenjun Kevin Zeng, Deqing Zou, Hai Jin: Fully Reversible Privacy Region Protection for Cloud Video Surveillance. IEEE Trans. Cloud Computing 5(3): 510-522 (2017)

[11] Weizhong Qiang, Gongping Xu, Weiqi Dai, Deqing Zou, Hai Jin, CloudVMI: A Cloud-Oriented Writable Virtual Machine Introspection. IEEE Access 5: 21962-21976 (2017)

[12] Weiqi Dai, Pengfei Wan, Weizhong Qiang, Laurence T. Yang, Deqing Zou, Hai Jin, Shouhuai Xu, Zirong Huang, TNGuard: Securing IoT Oriented Tenant Networks Based on SDN. IEEE IoT Journal 5(3): 1411-1423 (2018).

[13] Peng Xu, Xiaolan Tang, Wei Wang, Hai Jin, Laurence T. Yang: Fast and Parallel Keyword Search Over Public-Key Ciphertexts for Cloud-Assisted IoT. IEEE Access 5: 24775-24784(2017)

[14] Deqing Zou, Yu Lu, Bin Yuan, Haoyu Chen, Hai Jin: A Fine-Grained Multi-Tenant Permission Management Framework for SDN and NFV. IEEE Access 6: 25562-25572 (2018)

[15] Peng Luo, Deqing Zou, Hai Jin, Yajuan Du, Long Zheng, Jinan Shen: DigHR: precise dynamic detection of hidden races with weak causal relation analysis. The Journal of Supercomputing 74(6): 2684-2704 (2018)

[16] Ihsan H. Abdulqadder, Deqing Zou, Israa T. Aziz, Bin Yuan, Weiming Li: SecSDN-Cloud: Defeating Vulnerable Attacks Through Secure Software-Defined Networks. IEEE Access 6: 8292-8301 (2018)

[17] Ihsan H. Abdulqadder, Deqing Zou, Israa T. Aziz, Bin Yuan: Validating User Flows to Protect Software Defined Network Environments. Security and Communication Networks 2018: 1308678:1-1308678:14 (2018)

[18] Weiqi Dai, Jun Deng, Qinyuan Wang, Changze Cui, Deqing Zou, Hai Jin, SBLWT: A Secure Blockchain Lightweight Wallet based on Trustzone, IEEE Access, DOI: 10.1109/ACCESS.2018.2856864

马晓静
副教授
研究方向：多媒体安全，密文处理
Email：lindahust@hust.edu.cn

邹德清
教　授
研究方向：云安全，软件安全与软件定义安全
Email：deqingzou@hust.edu.cn

羌卫中
副教授
研究方向：系统安全
Email：wzqiang@hust.edu.cn

代炜琦
讲　师
研究方向：云安全、可信计算、虚拟化安全、可信SDN等
Email：wqdai@hust.edu.cn

袁　斌
博士后
研究方向：软件定义网络安全
Email：yuanbin@hust.edu.cn

大数据组典型成果介绍

华强胜，胡振宇，钟芷漫，林昌富，赵　峰，郑志高

关键词：低复杂度算法，数据并行，QoS机制技术，图数据查询，语言模型

1 介绍

大数据具有多样性、快速性和大规模等特点，传统算法与系统面对大数据的复杂应用场景具有很大挑战性。本研究小组旨在研究大数据的存储、检索、分析和挖掘等技术，为大数据的广泛高效应用提供技术支持。基于本组所承担的“大数据快速存储与计算环境”、“高通量计算与近似计算的理论与算法”等国家重点研发计划项目；“面向大数据的高时效并行计算机系统结构与技术”、“大规模图中图性质求解的低复杂度分布式算法研究”等国家自然科学基金项目以及一批面向电子政务、航空运输等等应用处理平台项目，对不同领域大数据问题进行深入研究，从算法、系统等多个维度开展研究，研发了一系列的典型系统、探讨了多个典型大数据领域数据处理问题解决思路：

1）在大数据理论方面，针对隐私保护数据聚合问题，大部分已有协议只能针对某一种或者几种统计函数，同时需要一个可信任的第三方。我们设计一种协议，在无可信任第三方情况下，聚合中心获取所有用户数据，同时去除可信任的第三方的参与和实现较低的通信复杂度。

2）在大数据处理方面，通过对数据并行应用进行观察和分析，得出数据并行应用的运行过程具有阶段性的结论。根据每个阶段中任务的数据和数据分区的特点，研发了基于阶段划分的程序分析优化器SDPA。同时，针对I/O请求具有突发性并且在不同服务器上分配不均以及由于应用的带宽访问模式的限制使得I/O带宽分配不合理导致系统整体性能下降的问题，提出了一个通用的I/O带宽控制方案。

3）在大数据管理方面，针对SPARQL查询语言复杂难懂以及无法在合理时间内处理复杂查询的特点，提出以自然语言为访问接口，设计了基于异构图模式近似匹配方法研究的自然语言问答系统。同时还设计了共享内存环境下图查询并行处理系统ParTriple。

4）在大数据分析方面，针对实际流处理应用中，上下流任务在数据处理和任务恢复两方面所具有的强依赖特性，给分布式流处理系统带来了处理延迟高、吞吐率受限、恢复延时长等严重问题，研发了分布式流处理容错系统Ares。另一方面，为了衡量查询语句与文档之间的语义相关性，本组提出基于实体的语言模型平滑方法。提出了符合文档语义主题下单词概率分布的实体语义语言模型，以及两层次的平滑方法。

2 低复杂度隐私保护数据聚合算法研究

隐私保护数据聚合问题一直是应用密码学领域的研究热点。在实际生活中很多应用，用户需要提供他们敏感的数据到数据中心，数据中心根据收到的用户的数据计算进行相应的统计函数，如最大值、最小值和平均数等等。基于计算的结果，数据中心为用户提供相应的服务。对于这个问题，通常有两个核心的挑战需要解决，即通信效用与数据隐私。如何降低计算过程中通信开销和保护用户数据隐私是相关研究学者一直不懈探索的目标。

2.1 研究问题

对于我们的研究问题中，我们采用One Aggregator模型，即在这个模型中，有一个不可信的聚合中心和n个用户。通过双向的通信链路，聚合中心与每一个用户可以互相发送消息。每个用户定期会产生隐私的数据。根据用户的隐私数据，聚合中心主要负责计算一些统计函数。我们假设聚合中心与用户都是半诚实的。他们忠诚的执行协议，同时他们对其他用户的隐私数据是好奇的。他们彼此之间可以相互同谋，以便获取其余参与方的隐私数据。我们假设聚合中心和k个用户同谋。我们的研究问题是在无可信任第三方情况下，聚合中心计算任意统计函数，同时保护用户隐私和实现较低通信开销。

2.2 研究策略

从问题中，我们可以看出为了使聚合中心可以计算任意统计函数，其中一个可行的方法就是获取所有用户数据，然而用户数据是隐私的，显然用户直接发送隐私数据到聚合中心的方式是禁止的，为此我们设计一个协议，在无可信任第三方情况下，聚合中心获取所有用户数据，同时保护隐私((*n*-*k*)-源匿名：对于未同谋用户的数据，聚合中心不知道所收到的数据与用户之间所对应的关系)和实现较低的通信开销。

为了保护用户数据隐私，我们采用DH密钥交换协议，使得每个用户产生共享的密钥集。根据所得密钥集，每个用户生成相应的伪随机函数。利用生成的伪随机函数每个用户对数据进行加密。

为了降低通信开销，我们设计随机抽样与划分技术使得每个用户获得唯一序列号。具体来说，每个用户在区间 [1, n^5] 随机抽样一个随机数，并将此区间划分为n等长的子区间。当某个子区间所含随机值数量是1时，聚合中心保存此子区间；当某个子区间所含随机值数量超过1但是小于 $a=\frac{\log n}{\log\log n}$ 时，聚合中心将此区间均匀划分a^2份；当某个子区间所含随机值数量大于$a=\frac{\log n}{\log\log n}$ 时，聚合中心将此区间均匀划分($log\ n$)^3份。聚合中心与用户递归此过程，直到聚合中心保存的子区间的数量等于n。然后，聚合中心发送这n个子区间到所有用户。每个用户根据这n个子区间获得自己的唯一序列号。最后根据唯一序列号，聚合中心获取所有的用户隐私数据。通过严格的理论证明，对于隐私保护问题，我们所设计协议实现(n-k)-源匿名；对于通信开销，我们协议通信复杂度是$O(n\frac{\log n}{\log\log n})^2$。

3 面向数据并行的大数据处理技术

3.1 SDPA：数据并行的应用程序的程序分析的优化器

在大数据处理系统中，数据并行应用的性能变的越来越重要，这些应用的性能和代码逻辑与运行时的资源利用有关。程序分析技术是一种常用的优化应用的方法，而且这项技术已经被运用到数据并行应用中，但是目前还没有一个能够高效的分析整体数据并行应用的方法。

通过对数据并行应用进行观察和分析，我们得到数据并行应用的特点，即数据并行应用的运行过程具有阶段性，每个数据并行应用的作业会划分为阶段。在每个阶段中会运行多个独立的任务，任务的数目和数据分区的数目相同，每个任务会计算一个分区的数据，且每个任务对分区中的数据进行同样的操作。根据这个特点，提出了基于阶段划分的程序分析(Stage-Divided Program Analysis)，称之为SDPA，来简化和加速数据并行应用的程序分析。对于每个阶段，对其进行重新划分为一个或多个运行周期。对于每个周期内的用户定义函数，将其抽取出来并融合为一个方便分析的函数。这样避

免了过程间分析且规避开不必要分析的复杂数据处理系统框架代码。

3.1.1 问题分析及解决思路

性能对于数据并行应用程序至关重要，通过优化代码来提高应用程序的性能是一种好方法。程序分析是一种常用的方法，它分析程序的行为，并获得程序的正确性，安全性和活跃性等属性来优化程序。这些大数据处理系统的编程接口支持数据并行运算符，它使得编写大数据应用程序变得很容易，只需要少量代码就能完成编写工作。但是，这些应用程序的每个运算符都会调用大量的框架代码。因此，一个简短的应用程序可能会引用很多类和方法，并且分析这些并行数据应用程序会花费大量时间。

3.1.2 SDPA的设计与实现

数据并行应用有一个特点，就是在运行阶段会被划分为多个Stage，每个Stage的任务是相对独立的，所有的任务都对不同的的数据分区进行相同的计算并且具有相同的shuffle依赖。基于这些发现，我们提出了SDPA来简化加速数据并行应用的程序分析。SDPA的处理流程如图3.1所示。

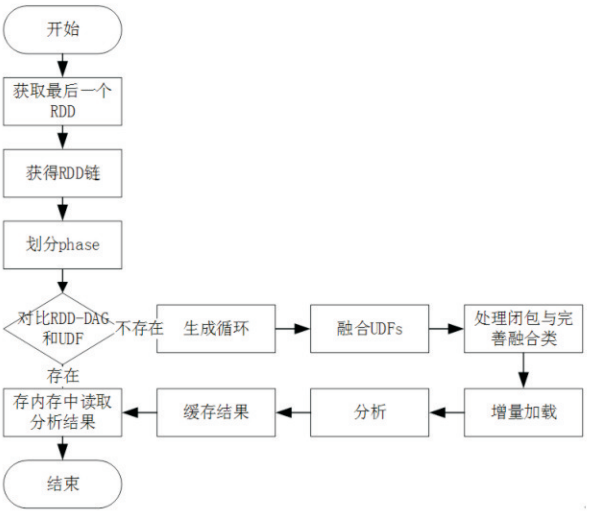


图3.1 SDPA处理流程图

1. Phase划分

在Spark中，每个Job可以划分为一个或多个Phase。每个Phase可以分为三个部分：（1）从数据源读取数据对象，例如缓存RDD或Shuffle缓冲区；（2）对每个数据对象执行一系列的用户自定义函数进行计算；（3）计算结果写入新的数据收集器。Phase的划分如图3.2所示。按照Phase的定义，Phase与Spark划分的Stage的区别在于，Stage中间可能有缓存操作，缓存操作会对Stage进行隔断，因此一个Stage会被划分为一个或多个Phase。

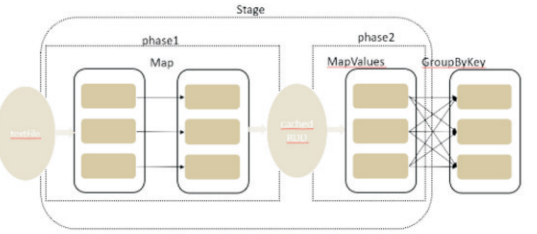


图3.2 带有Cache的Phase划分

2. 用户自定义函数融合

每个phase，都是对数据源中的数据对象应用一系列用户定义函数的过程。因此，将每个phase中各个算子的用户定义函数抽取出来，然后将这些函数融合成为一个大的函数，这样可以跳出算子的束缚，更加容易的分析一系列的算子代码。Phase中对象的操作如图3.3所示。

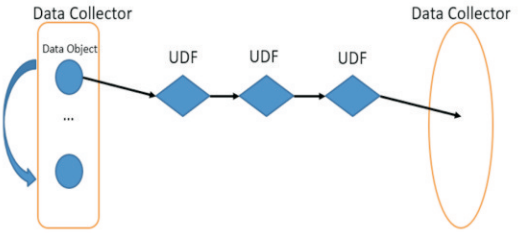


图3.3 phase中对象操作示意图

融合过程使用融合工具Soot来实现，然后操作Soot生成的中间语言Jimple代码。对每个Phase的迭代融合机制可概括为以下四个步骤：

1）获取当前Phase的最后一个RDD

2）获得完整的RDD链

3）通过Java的反射机制抽取出每个RDD的UDF

4）将每个UDF的输入输出相连接

融合过程中，需要针对每个算子的语义进行处理。将一个UDF的输出作为接下来相邻UDF的输入，这样就可以将一系列的UDF融合成为一个大的函数，然后使用Soot声明一个类，将生成的循环放入这个类中。在后面的程序分析过程中，就可以直接分析这个融合生成的类。

3. 代码处理与分析

在SDPA中，使用Soot-SPARK来对融合之后的代码进行程序分析。Soot-SPARK是一个用Java实现指针分析的框架，它支持基于子集的和基于等价的分析以及两者之间的任何分析。SPARK是作为Soot框架的一部分提供的，可以在soot.jimple.spark.*包中找到。SPARK的一部分提供指针分析，并且可以使用选项来控制分析的多个方面。由于使用Soot-SPARK分析代码需要一个主类与入口方法，而在对用户定义函数进行融合之后只是得到一个大的函数，因此需要对融合之后的函数进行处理。SDPA将融合之后的函数放到一个Phase-Function-Class里面，然后添加入口的apply方法，最后将这个类和前面的Phase-Function-Class链接起来。这就可以使用Soot-SPARK进行程序分析。

4. 闭包处理与分析过程加速

在SDPA中，闭包处理通过Soot对Jimple代码分析为基础。针对不同的RDD，每个RDD都会对应一个算子，对每个算子来说，它生成的RDD都是固定类型的，比如Map算子，生成的是MapParitionRDD，因此，对于这些RDD，首先进行强制类型转换，得到精确类型的RDD。得到精确类型之后，通过观察Jimple代码，遍历其字段，通过判断字段的name是否包含“$outer”来判断这些是否是闭包变量，针对这些闭包变量，使用Soot将其注入到最后融合的Phase-Function-Class中。

此外，在完成迭代器融合之后需要对融合生成的Phase-Function-Class进行程序分析，首先需要对Phase-Function-Class需要的类和方法进行加载，而Soot-SPARK的加载方式是全量加载，也就是说会加载很多不必要的方法，因此为了加快分析过程，SDPA实现了增量加载。针对迭代应用，SDPA实现了cache code，以方便复用分析结果。

3.2 海量存储系统中的软件定义QoS机制技术

随着数据信息的快速发展，对大容量存储系统的需求日益增长，海量存储系统逐渐成为研究热点。海量存储系统由客户端、管理节点、存储节点等三种节点组成，结构图如图3.4所示，管理节点存储命名空间和系统的元数据，如文件名、其对应的对象号、以及该对象保存在哪个数据节点上等信息；数据节点保存文件数据（对象），对象以对象号标识应用服务器通过定制协议或 HTTP 访问存储系统。海量存储系统的核心是分布式文件系统，文件系统允许从一个计算机网络中的多台主机上访问文件，这使得多个用户可以通过多台主机共享文件及存储资源。

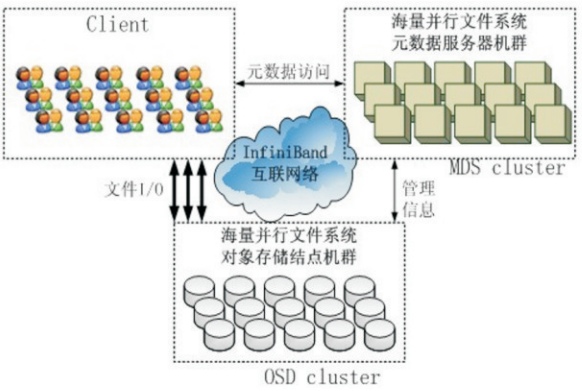


图3.4 海量存储系统

海量存储系统的发展也使得应用越来越倾向于数据密集型，导致高度密集的海量数据I/O吞吐需求日益增长，同时半导体技术的飞速发展，使得固态硬盘（Solid State Drives）得到迅速普及和广泛应用。尽管新的高速存储设备的出现和部署使得HPC系统的整体I/O性能稳步提高，相比于计算性能的增长，I/O资源仍处于不足的状态。这使得多应用之间对于存储资源的竞争和干扰极大程度上影响应用的性能，比如一个规模庞大的作业因为等待缓慢的I/O而被阻塞，又或者一个小的作业在运行时产生了大量I/O阻塞了其他优先级较高的应用。可以得出I/O的分配不合理不光是存储资源的低利用率，也是对计算资源的浪费。所以，使得每个应用占据与它的计算资源匹配的I/O资带宽对于HPC存储系统来说是很有必要的。

3.2.1 问题分析及解决思路

在HPC平台上实现I/O控制面临很多挑战：第一，HPC应用可以轻易达到上千的计算节点的规模，每秒产生百万级别的I/O请求。这使得全局上监控应用的I/O行为十分困难。同时搜集到应用所有线程的I/O行为会造成监测规模过于庞大的问题。第二，I/O请求具有突发性并且在不同服务器上分配不均。这些特性要求I/O控制做到细粒度（应对I/O突发性）和全局优化（应对I/O分配不均）并且这两点在实现上是有较大的难度的。同时要求服务器去学习应用的I/O变化规律也是十分困难的。第三，保障应用的带宽并不是简单的设置一个硬性的标准，同时需要使得整个系统的I/O性能达到最优，例如当应用的带宽被自身的访问模式所限制无法达到理想带宽时，余下的带宽就应当分配给其他应用使用以达到整体性能最优。

为此，我们提出了一个通用的I/O带宽控制方案，基于OrangeFs并行文件系统实现，核心的思想是在HPC存储系统中加入Data Plane和Control Plane组件去控制应用的I/O行为，同时使用令牌桶技术达到多应用间的公平性和整体性能最优，架构如图3.5所示。

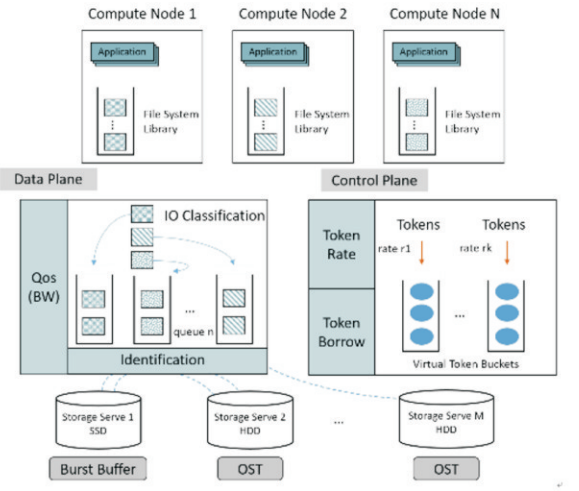


图3.5 SDQoS框架

3.2.2 SDQoS设计与实现

1. 数据模块和控制模块

每一个服务端维持一个Data Plane，是监测和控制应用I/O行为的核心组件，同时这一系列操作会在实际I/O行为发生之前进行处理。该组件的实现在不同的文件系统上会有不同的实现方式，例如在lustre文件系统上，data plane可以实现在OST层，在pvfs文件系统中，我们实现在服务端。

Data Plane首先与Control Plane进行连接，获取到应用的I/O需求，并为每个应用维护一个单独的队列。然后，每个到来的I/O请求从令牌桶中获得令牌，根据获得令牌的情况被服务。

Control Plane在整个系统运行过程中是全局且唯一的，它并不是单独运行在某个节点上，而是逻辑上存在的一个分布式调度组件，用于接收应用的特性并基于这些特性计算出应用应该被分配的带宽（也可以直接指定带宽）。需要指出的是，这些被计算出的带宽并

不是应用强制需要达到的带宽。由于实际I/O过程的不均衡以及服务器能提供带宽的物理限制，应用实际得到的带宽也会进行调整。

2. 本地I/O控制

在SDQoS中，我们利用令牌桶去限制单个应用的I/O带宽。

令牌桶策略在网络中被广泛用于管理和整形网络数据包，它能够应对突发流量同时保障了常规流量。因为在HPC中，I/O同样存在突发现象，所以我们也可以在HPC环境下的 I/O场景中使用令牌桶策略。

令牌桶策略在HPC中的使用和在网络应用中的使用有很大的区别。在网络中，超出限制的数据包可以被丢弃重发，而在HPC中，I/O不可以被丢弃，需要被缓存起来，这样就会造成延迟和阻塞。并且，我们有时需要允许应用得到超过它得到的令牌数的I/O资源以达到整个系统I/O性能最优。

在具体实现上，我们为每个应用都分配了单独的令牌桶。令牌桶根据应用的需求设定令牌产生速率以及令牌桶容量。其中，令牌产生速率代表了应用稳定情况下的带宽，令牌桶容量代表了应用在突发情况下所能分配的最大带宽。

4 RDF图数据的查询与匹配

由于图数据模型具有描述能力强，适用范围广等优点，越来越多的数据使用图数据模型发布或存储。资源描述框架（Resource Description Framework，RDF）数据是图数据中的一种。由于RDF数据模型有着诸多优点，比如方便简洁，模块化程度高等，越来越多的机构或组织加入了RDF用户社区。

SPARQL语言是一种常用的图查询语言。SPARQL查询语句通常构成一个查询图，表示在对应RDF图数据上的子图匹配查询。SPARQL是知识图的固有访问接口，但复杂难懂，我们提出以自然语言为访问接口，设计了基于异构图模式近似匹配方法研究的自然语言问答系统。此外，SPARQL查询引擎无法在合理时间内处理复杂查询，我们提出了共享内存环境下图查询并行处理系统ParTriple。

4.1 异构图模式近似匹配方法

4.1.1 研究问题

随着互联网的飞速发展，大量的知识库可供公众访问，例如DBpedia，YAGO，Freebase。不同于传统的信息检索中使用的文档库，知识库以RDF形式，即三元组（主语，谓语，宾语）的形式整合细粒度的信息，因此它在整合信息，加强智能搜索方面担任非常重要的角色。

然而，由于知识库采用普通用户不熟悉的结构化的方式来存储信息，所以访问使用这些大规模知识库中的信息成为当前困扰普通用户的问题。另外，因为知识库以图的结构来组织信息，所以，问题的本质也就是图模式的近似匹配问题。但是为了连接用户和知识图，需要有一个访问接口来方便用户使用。

虽然知识图提供了固有的访问接口，例如SPARQL，但是复杂的语法和知识图结构使得它不能直接为普通用户所用。因此，为知识图设计一个高效且方便用户使用的访问接口成为人们亟待解决的问题。由于以自然语言为访问接口，方便用户直接从知识库中获取信息，所以我们提出了基于异构图模式近似匹配方法，在此基础上实现了自然语言问答系统Svega。

4.1.2 基于语义向量的图模式匹配与相似度计算

给定一个查询图$Q=(V, E, R_E)$，其中$V=v_1, v_2, \ldots, v_n$. 节点集合V中的每一个节点v_i在知识库中都有一个匹配的候选结合S_{vi}，则当且仅当满足以下两个条件时，知识库中的一个子图sub-KG才是查询图Q的匹配子图：

1. 对于查询图Q中的任何一个节点v_i，一定有一个节点$u_i \in S_{vi}$，并且该节点被包含在子图sub-KG中；

2. 如果查询图Q中的两个节点v_i和v_j之间存在一条边，则在子图sub-KG中一定存在一条路径$P(u_i, u_j)$，并且节点u_i和u_j分别是查询图中节点v_i和v_j对应的匹配节点。

异构图模式近似匹配方法研究，提出了一种以图模式近似匹配的方法来访问知识图的方法。为了更加精准的挖掘出查询目的，提出了一种实体驱动的策略来构建查询图。首先基于查询图中节点特征的识别出自然语言问句中的实体，然后根据问句中单词之间的语法关系分析挖掘出实体之间的谓语关系，从而构建出查询图。在进行图模式匹配过程，其本质就是消歧处理，所以本系统首先基于查询图的结构特征进行消歧，提出了基于节点匹配来从知识库中得到查询图的异构子图，同时挖掘匹配的路径，过滤歧义信息，然后基于查询图的语义特征进行消歧，提出了合成词向量的方法来表示查询图中边和匹配子图中路径的语义信息，从而简化了相似度的计算，同时可以根据语义相似度来过滤歧义信息。

4.2 共享内存环境下图查询并行处理系统ParTriple

4.2.1 研究问题

随着RDF数据的爆炸式增长，现有的一些SPARQL查询处理系统已无法在合理时间内处理复杂查询。为此，SPARQL查询引擎应当使用并行处理技术来实现对复杂查询的高效处理。然而，当前的一些SPARQL查询并行处理技术有着查询优化效率低下且生成的查询计划不利于并行，系统并行程度较低等不足。在这种背景下，将SPARQL图查询处理与并行技术相结合，开发SPARQL图查询并行处理系统，即共享内存环境下的图查询并行处理系统，显得尤为必要。

开发图查询并行处理系统主要有两方面的挑战。一方面，查询优化对查询处理的性能有很大影响，其生成的查询计划不可避免地影响查询执行的并行处理。另一方面，由于现代计算机上配备的处理核心数目越来越多，如何在单机上最大限度地利用这些计算能力来进行SPARQL 查询处理本身也有很多难点，比如并行性开发不够，资源竞争过多等等。为了应对这些挑战，我们在高可扩展的RDF数据存储查询系统TripleBit的基础上设计并实现了ParTriple系统。

4.2.2 并行处理系统ParTriple

ParTriple旨在为多核计算机环境下的SPARQL图查询提供并行处理。我们提出了一种基于自底向上树的查询计划生成方法。另一方面，提出了一种两级并行模型来执行查询计划，以充分利用多核架构的计算能力，提高查询处理性能。

ParTriple系统主要由数据导入模块、数据结构模块、查询处理模块三部分组成。ParTriple通过数据导入模块，对RDF数据进行解析、字典映射，并进行排序。在此期间，数据结构模块与数据导入模块进行交互，建立数据字典，存储三元组数据，并建立相应的索引。查询处理模块用于处理用户提交的SPARQL查询语句。可分成SPARQL语句解析、查询优化器、任务管理、并行操作符等子模块。

首先，该系统提出了一种高效，易于并行的查询计划生成算法。这种算法生成的查询计划可以使用流水线处理技术，且不同的流水线之间不需要相互等待。进一步地，这种查询计划可以更有效地削减中间结果，提高查询处理性能。接着，提出了一种数据块级与模式级的两级并行执行框架，将查询执行分解成基于数据存储分块的子任务与基于中间结果分块的子任务，从操作符间并行与操作符内并行两个层面开发了系统的并行性。最后，开发了三种不同的并行操作符，分

别针对数据扫描，连接处理，数据洗牌等操作，进一步提升了系统查询处理的性能。

5 大数据分析技术

5.1 分布式流处理容错系统Ares

为了满足实时流处理应用的低延迟和高容错的需求，分布式流处理系统需要保障系统能够以低延迟的方式处理海量的实时数据流，同时当系统中的节点发生故障时，系统能够在较短的时间内恢复正常。现有的分布式流处理系统通常通过优化任务调度来实现海量实时数据流的低延迟处理。典型的分布式流处理系统任务调度策略通过将流处理应用拓扑中的上下流任务进行绑定并分配到同一节点的方式，降低上下流任务之间的数据传输时间，从而保障低延迟处理性能。然而，该任务调度策略忽视了上下流任务之间的依赖关系对系统容错性能的影响。即当系统中的节点发生故障时，由于失效节点上存在大量的上下流任务，这些任务之间的依赖关系会导致在任务恢复期间发生任务级联等待的现象，从而使得系统无法在较短的时间内恢复正常。

本研究指出实现一个低延迟和高容错的分布式流处理系统的关键在于能否在任务调度过程中充分挖掘流处理应用拓扑中上下流任务之间的依赖关系。为此，研发了分布式流处理容错系统Ares。Ares在系统任务调度过程中兼顾上下流任务之间的依赖关系对处理性能和容错性能两者的影响，并提出了一种容错调度问题来同时优化处理延迟和恢复时间。为了最大化系统效用，Ares利用单一任务的最佳放置策略往往取决于该任务的上下流任务的放置策略的特性，设计了一种基于最佳应对动态算法Nirvana来自动优化任务放置策略。相对于目前最新流处理系统，Ares将系统总体吞吐率提高了3.6倍，将平均处理延迟降低了50.2%，将平均恢复时间降低了52.5%。

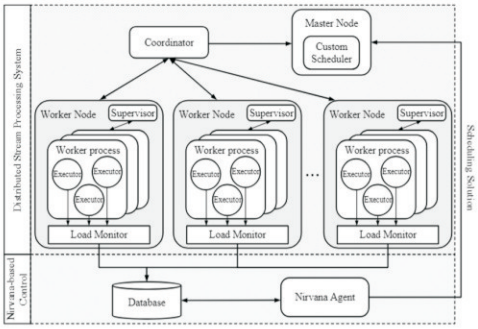


图5.1 Ares系统架构

图5.1显示了Ares的系统架构。Ares的架构在传统分布式流处理系统架构的基础上，增加了一个负载监视器和一个Nirvana控制器。负载监视器负责实时监控所有计算节点上的任务的资源使用情况。Nirvana控制器根据当前系统的资源使用情况，通过Nirvana算法计算得到满足低延迟和高容错需求的最佳任务调度方案。

5.2 面向文档过滤的语言模型平滑方法

丰富的网络资源中蕴含着海量的数据信息，帮助用户从中快速、准确的找到所需的信息是一项极具价值的任务。但是海量的数据规模以及自然语言表达带来的语义歧义性和多样性，给信息系检索带来了巨大的挑战。我们以实体为桥梁，以维基百科中的实体信息为内容，提出了符合文档语义主题下单词概率分布的实体语义语言模型，并且提出两层次的平滑方法，结合文档无关的全局语料库信息源和文档主题相关的实体语义语言模型信息源来对原始文档语言模型进行平滑，使得平滑后的语言模型能够很好的衡量查询语句和文档之间的语义相关性，提高了检索系统的性能。

5.2.1 实体语义语言模型的构造

文档中的实体被链接到维基百科中。但是由于实体链接工具不是完美的，存在错误的可能，不能保证百分百的正确率。为了处理这种不确定性，提出了硬合并（Hard-Fused Method）和软合并（Soft-Fused Method）两种方法。硬合并

方法只考虑文档中置信度比较高的实体，同时忽略实体的频率信息。软合并方法将置信度作为实体的权重，同时考虑实体的频率信息。

硬合并方法仅考虑链接置信度大于特定阈值的实体，并且忽略实体在文档中出现的次数。实体链接工具的精度有限，不可避免的会产生错误。直接忽略低置信度的实体可以预防引入实体链接工具的错误，只使用那些质量高的、信得过的链接实体，提高模型的健壮性。硬合并方法同时将所有高置信度的实体当作一样来对待，分配同样的权重，即忽略它们在文档中的频率信息。

另外一种构建实体语义语言模型的方法是软合并方法，这种方法考虑实体链接结果中的置信度等级和实体在文档中频率信息。文档中实体的重要程度是不一样的。重要实体的知识库信息与文档的语义主题更加相关。用SF-ESLM来表示软合并方法生成的实体语义语言模型：

$$P_{SF-ESLM}(w|d) = \frac{1}{\sum_{m' \in M(d)} p(m')} \sum_{m \in M(d)} p(m) * P_{MLE}\left(w \middle| Wiki\left(e(m)\right)\right)$$

其中$P_{SF-ESLM}(w \mid d)$表示单词w在文档d软合并方法得到实体语义语言模型中的概率大小，$M(d)$ 表示文档d中所有实体提及名字组成的集合。

5.2.2 两层次的平滑方法

基于实体的语言模型平滑框架从文档语义主题的角度出发，利用实体作为桥梁，构建实体语义语言模型。在这个方法中，文档中的实体被实体链接工具链接到外部实体知识库中，如维基百科，然后使用软合并方法和硬合并方法将实体在知识库中的信息合并在一起组成实体语义语言模型。最后提出了两层次的平滑方法，将上述语言模型结合在一起，可以很好的区分单词与文档主题的相关性。

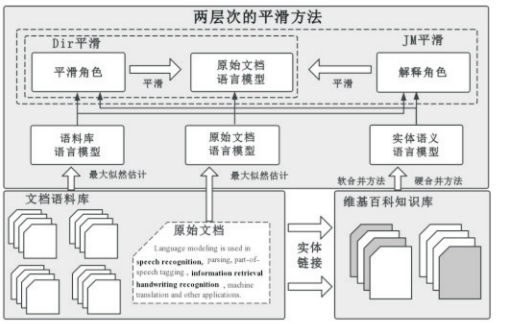


图5.2 基于实体的语义平滑方法示意图

如图5.2，整个框架包含原始文档语言模型、实体语义语言模型和语料库语言模型三个语言模型。原始文档语言模型通过统计原始文档中词频信息来计算得到。实体语义语言模型通过文档中实体指向的维基百科主页信息来计算得到。全局语料库语言模型通过统计语料库中的全局词频信息来计算得到。实体语义语言模型是整个框架的核心。它引入与文档主题相关的语义信息，使得平滑后的语义模型更加接近文档主题下的单词概率分布。

附成果列表：

开源系统

[1] 高性能计算机高速缓冲系统SSDUP，https://github.com/CGCL-codes/SSDUP

[2] GPU异步图计算系统Frog，https://github.com/CGCL-codes/Frog

[3] Spark数据Cache替换策略LCS，https://github.com/CGCL-codes/LCS

[4] 高性能实时分布式流处理系统DStream，https://github.com/CGCL-codes/DStream

[5] 动态可扩展近似集合元素判定技术DCF，https://github.com/CGCL-codes/DCF

专利：

[1] Mixed cache management. 专利号：US 9934281 B2. 专利授权日：2017年10月10日

[2] 一种基于减少进程间通信开销的并行I/O优化方法与系统.专利号:ZL201510181574.8. 专利授权

日：2018年03月27日
[3] 一种基于GPU的异步图数据处理系统.专利号: ZL201510176312.2. 专利授权日2017年12月22日
[4] 一种面向微博实时搜索的自适应索引方法.专利号: ZL201510242074.0. 专利授权日2017年12月08日
[5] 一种可扩展的面向关联的流式图数据划分方法.专利号:ZL201510242491.5. 专利授权日2017年11月10日
[6] 一种基于签到特征的用户位置分类方法及系统.专利号:ZL201510151827.7. 专利授权日2017年10月31日
[7] 一种实时流计算流速感知弹性执行容错系统.专利号:ZL201510180431.5.专利授权日2017年8月18日
[8] 一种阶梯接近中心度确定方法及系统，专利申请号201711349361.7
[9] 一种基于实体知识库的语言模型平滑方法，专利申请号201711265786.X
[10] 一种分布式SQL处理方法及系统，专利申请号201711260747.0
[11] 一种分布式计算有向图围长的方法，专利申请号201711237056.9
[12] 一种面向会话式实体搜索的查询扩展方法，专利申请号201711162771.0
[13] 一种GPU间通信的方法及系统，专利申请号201711115570.5
[14] 一种基于k-core集散结构的社交网络事件流通信优化方法，专利申请号201711096683.5
[15] 一种分布式博弈中心度求解方法，专利申请号201711069407.X
[16] 一种多GPU环境下的数据通信性能优化方法，专利申请号201711045712.5
[17] 一种面向大规模图数据发布的隐私保护方法及系统，专利申请号201711050486.X

论文：
[1] Dongxiao Yu, Yong Zhang, Yuyao Huang, Hai Jin, Jiguo Yu and Qiang-Sheng Hua. Exact Implementation of Abstract MAC Layer via Carrier Sensing. INFOCOM 2018
[2] Xuhui Gong, Qiang-Sheng Hua, Lixiang Qian, Dongxiao Yu, Hai Jin. Communication-Efficient and Privacy-Preserving Data Aggregation without Trusted Authority. INFOCOM 2018
[3] Hong Huang, Bo Zhao, Hao Zhao, Zhou Zhuang, Zhenxuan Wang, Xiaoming Yao, Xinggang Wang, Hai Jin, Xiaoming Fu. A Cross-Platform Consumer Behavior Analysis of Large-Scale Mobile Shopping Data. WWW 2018
[4] Changfu Lin, Jingjing Zhan, Hanhua Chen, Jie Tan, Hai Jin. Ares: A High Performance and Fault-tolerant Distributed Stream Processing System. ICNP 2018
[5] Hanhua Chen, Fan Zhang, Hai Jin, Popularity-aware Differentiated Distributed Stream Processing on Skewed Streams, ICNP 2017
[6] Hanhua Chen, Liangyi Liao, Hai Jin, Jie Wu, The Dynamic Cuckoo Filter. ICNP 2017
[7] Fei Wang, Xuanhua Shi, Dongxiao Yu, Zhixiang Ke, Hai Jin, Song Wu. SDPA: An Optimizer for Program Analysis of Data-Parallel Applications. HPCC 2018
[8] Gaofeng Li, Pingpeng Yuan, and Hai Jin. Svega: Answering Natural Language Question over Knowledge Base with Semantic Matching. SEKE 2018
[9] Tao Zhao, Hong Huang, and Xiaoming Fu. Identifying Topical Opinion Leaders in Social Community Question Answering. DASFAA 2018
[10] Fangjia Xing, Liming Gui, Hanhua Chen, Changfu Lin, Hai Jin. Efficient Event Stream Dissemination in Online Social Networks Based on Community Detection. ICC 2018
[11] Sijie Wu, Hanhua Chen, Changfu Lin, Hai Jin. Shadow: Exploiting the Power of Choice for Efficient Shuffling in MapReduce. ICPADS 2017
[12] Syed Muhammad Abrar Akber, Changfu Lin, Hanhua Chen, Fan Zhang, Hai Jin. Exploring the Impact of Processing Guarantees on Performance of Stream Data Processing. ICCT 2017
[13] Hao Long, Pingpeng Yuan and Hai Jin. Distributed PathGraph: A Cluster Centric Framework for Distributed Processing Graph. SOCA 2017
[14] Peng Zhao, Xuan Luo, Jiang Xiao, Xuanhua Shi and Hai Jin. Purin: Graph Processing System on Multi-GPUs. SOCA 2017
[15] Fangjia Xing, Fan Zhang, Xuxiang Tian, Wenyao Li, Hanhua Chen, Hai Jin. Identifying the Most Recent Heavy Hitters in Large-Scale Streams Using Block-wise Counting. SOCA 2017
[16] Dongxiao Yu, Yifei Zou, Jiguo Yu, Xiuzhen Cheng, Qiang-Sheng Hua, Hai Jin, Francis C.M.

Lau. Stable Local Broadcast in Multihop Wireless Networks Under SINR. IEEE/ACM Transactions on Networking (ToN), 2018.

[17] Xuanhua Shi, Xuan Luo, Junling Liang, Peng Zhao, Sheng Di, Bingsheng He, Hai Jin, "Frog: Asynchronous Graph Processing on GPU with Hybrid Coloring Model", IEEE Transactions on Knowledge and Data Engineering (TKDE), 2018.

[18] Xuanhua Shi, Zhigao Zheng, Yongluan Zhou, Hai Jin, Ligang He, Bo Liu, Qiang-Sheng Hua, Graph Processing on GPUs: A Survey, ACM Computing Surveys. Vol.50, No.6, pp.1-35, 2018.

[19] Hong Huang, Yuxiao Dong, Jie Tang, Nitesh Chawla, Xiaoming Fu. Will Triadic Closure Strengthen Ties in Social Networks? ACM Transactions on Knowledge Discovery from Data (TKDD), 2018.

[20] Hai Jin, Na Wang, Dongxiao Yu, Qiang-Sheng Hua, Xuanhua Shi, Xia Xie. Core Maintenance in Dynamic Graphs: A Parallel Approach based on Matching. IEEE Transactions on Parallel and Distributed Systems (TPDS), 2018.

[21] Fan Zhang, Hanhua Chen, Hai Jin, Piggyback Game: Efficient Event Stream Dissemination in Online Social Network Systems. IEEE Transactions on Parallel and Distributed Systems (TPDS), 2018.

[22] Hai Jin,Chen Qian,Dongxiao Yu,Qiang-Sheng Hua,Xuanhua Shi,Xia Xie. Parallel computation of hierarchical closeness centrality and applications. World Wide Web, 2018.

[23] Feng Zhao, Zeliang Tian, Hai Jin. Entity-based language model smoothing approach for smart search, IEEE ACCESS, Vol.6, pp.9991-10002, 2018

[24] Qiong Hu, Hanhua Chen, Hai Jin, Chen Tian, Aobing Sun, Tongkai Ji. IBCube: An Economical and Incremental Datacenter Network. International Journal of Web Services Research. Vol.15, No.1, pp.27-46, 2018.

[25] Xiaofeng Ding, Yangling Ou, Jianhong Jia, Hai Jin, Jixue Liu. Efficient subgraph search on large anonymized graphs. Concurrency and Computation: Practice and Experience. 2018.

[26] Xiaofeng Ding, Xiaodong Zhang, Zhifeng Bao, Hai Jin. Privacy-Preserving Triangle Counting in Large Graphs. CIKM 2018

[27] Magnús M. Halldórsson, Yuexuan Wang, Dongxiao Yu. Leveraging multiple channels in ad hoc networks. Distributed Computing, 2018.

[28] Xia Xie, Xiaodong Yang, Xiaokang Wang, Hai Jin, Duoqiang Wang, Xijiang Ke. BFSI-B: An Improved K-hop Graph Reachability Queries for Cyber-Physical Systems, Information Fusion, Vol.38, pp.35-42, 2017.

华强胜
华中科技大学计算机学院副教授
研究方向：分布式算法
Email：qshua@ hust.edu.cn

胡振宇
2017级硕士
研究方向：内存计算、分布式计算系统
Email：cszhenyuhu@hust.edu.cn

钟芷漫
2018级硕士
研究方向：自然语言处理
Email：15927923213@163.com

林昌富
2015级博士
研究方向：分布式流处理系统，大规模图处理系统
Email：lcf@hust.edu.cn

赵　峰
华中科技大学计算机学院教授
研究方向：知识发现，信息检索
Email：zhaof@hust.edu.cn

郑志高
2015级博士生
研究方向：GPU图处理
Email：zhengzhigao@hust.edu.cn

块设备I/O虚拟化

（刘　伟　https://www.cnblogs.com/Bozh/p/5788402.html）

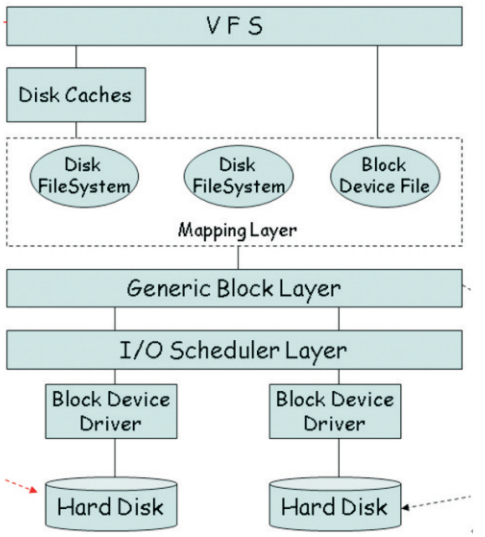
在计算机技术中，虚拟化（技术）或虚拟技术（英语：Virtualization）是一种资源管理技术，是将计算机的各种实体资源（CPU、内存、磁盘空间、网络适配器等），予以抽象、转换后呈现出来并可供分区、组合为一个或多个电脑配置环境。KVM，基于内核的虚拟机（英语：Kernel-based Virtual Machine，缩写为KVM），是一种用于Linux内核中的虚拟化基础设施，可以将Linux内核转化为一个hypervisor。KVM在2007年2月被导入Linux 2.6.20核心中，以可加载核心模块的方式被移植到FreeBSD及illumos上。KVM在具备Intel VT或AMD-V功能的x86平台上运行。它也被移植到S/390，PowerPC与IA-64平台上。在Linux内核3.9版中，加入ARM架构的支持。KVM目前由Red Hat 等厂商开发，对CentOS/ Fedora/RHEL等Red Hat系发行版支持极佳。

块设备IO虚拟化简介

作为KVM虚拟化中的一个模块，块设备IO的虚拟化非常重要。同网络IO虚拟化类似，块设备IO也有全虚拟化和virtio的虚拟化方式（virtio-blk）。现代块设备的工作模式都是基于DMA的方式，所以全虚拟化的方式和网络设备的方式接近，同样的virtio-blk的虚拟化方式和virtio-net的设计方式也是一样，只是在virtio backend端有差别。

块设备IO协议栈

如图所示：



Page cache层，这里如果是非直接IO，写操作如果在内存够用的情况下，都是写到这一级后就返回。在IO流程里面，属于writeback模式。 需要持久化的时候有两种选择，一种是显示的调用flush操作，这样此文件（以文件为单位）的cache就会同步刷到磁盘，另一种是等待系统自动flush。VFS，也就是我们通常所说的虚拟文件系统层，这一层给我们上层提供了统一的系统调用，我们常用的create，open，read，write，close转化为系统调用后，都与VFS层交互。VFS不仅为上层系统调用提供了统一的接口，还组织了文件系统结构，定义了文件的数据结构，比如根据inode查找dentry并找到对应文件信息，并找到描述一个文件的数据结构struct file。下层的文件系统需要实现file_operation的几个接口，做具体的数据的读写操作等。再往下就是针对不同

的文件系统层，比如我们的ext3，ext4等等。我们在VFS层所说的几个文件系统需要实现的接口，都在这一层做真正的实现。位于文件系统层下面的是通用块设备层，这一层在程序设计里面是属于接口层，用于屏蔽底层不同的块设备做的抽象，为上层的文件系统提供统一的接口。块设备驱动层就是针对不同的块设备的真实驱动层了，块设备驱动层完成块设备的内存映射并处理块设备的中断，完成块设备的读写。

块设备IO虚拟化

块设备IO的虚拟化流程和网络IO的流程基本一致，差别在于virtio-backend一段，virtio-net是写入到tap设备，virtio-blk是写入到镜像文件中。块设备IO的流程需要经过两次IO协议栈，一次位于Guest，一次位于HV。当我们指定virtio的cache模式的时候，实际上指定的是virtio-backend（下面简称v-backend）写入HV块设备的方式。在虚拟化层次来看，Guest对于这几种Cache模式是没有感知的，也就是无论Cache模式是怎样，Guest都不会有所谓的绕过Guest的Page cache等操作，Virtio-front模拟的是驱动层的操作，不会涉及到更上层的IO协议栈。

如上图所示，蓝色表示 writethrough，黄色表示 none，红色表示 writeback。其中虚线表示写到哪一个层次后write调用返回。

cache=writethrough（蓝色线）

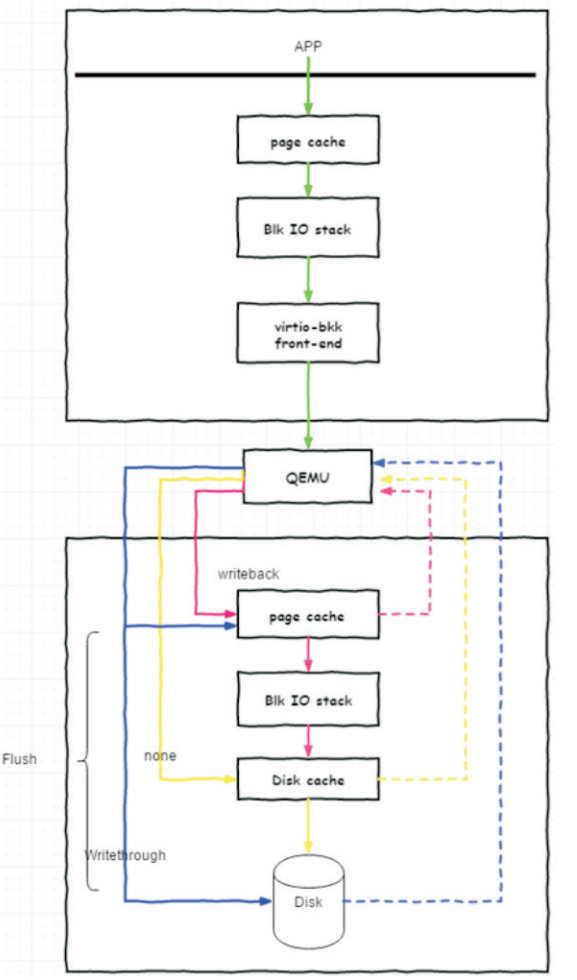
表示v-backend打开镜像文件并写入时候采用非直接IO+flush操作，也就是说每次写入到Page cache并flush一次，直到数据被真实写入到磁盘后write调用返回，这样必然会导致数据写入变慢，但是好处就是安全性较高。

cache=none（黄色线）

cache为none模式表示了v-backend写入文件时候使用到了DIRECT_IO，将会绕过HV的Page cache，直接写入磁盘，如果磁盘有Disk cache的话，写入Disk cache就返回，此模式的好处在于保证性能的前提下，也能保证数据的安全性，在使用了Disk cache电池的情况下。但是对于读操作，因为没有写入HV Page cache，所以会有一定性能影响。

cache=writeback（红色线）

此模式表示v-backend使用了非直接IO，写入到HV的Page后就返回，有可能会导致数据丢失。



浅谈AIOps

（陈　琼　https://1drv.ms/b/s!AgaWExwdZFs0hmQ2NP72bYsejgI4）

1. 什么是AIOps？

AIOps 即（Artificial Intelligence for IT Operations），智能运维。它将人工智能应用于运维领域，其目标是，基于已有的运维数据（日志、监控信息、应用信息等），通过机器学习的方式来进一步解决自动化运维所未能解决的问题，提高系统的预判能力、稳定性、降低 IT 成本，并提高企业的产品竞争力。

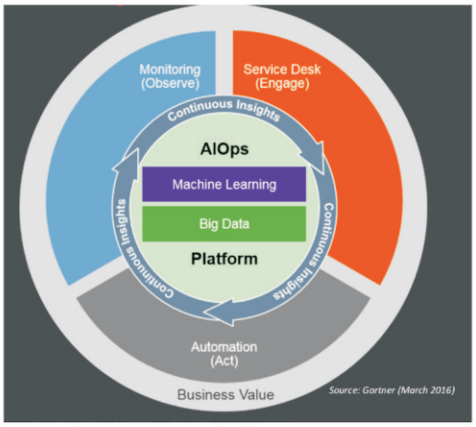


Figure 1: Gartner's visualization of the AIOPS platform

Gartner 在2016年时最早提出 AIOps 的概念，并预测到2020年，AIOps 的采用率将会达到50%。同时，Garther也详细解释了 AIOps 平台如何利用图1中的相关部件来工作，总的来说，其主要由两个部分组成：大数据和机器学习。大数据是基础，每一个机器学习算法都离不开数据的支撑，尤其就目前最流行的深度学习而言，普通体量的数据完全不足以支撑训练出理想的模型，需要海量的数据。机器学习是关键，不同机器学习算法挖掘出的数据价值是不同的，即不同的机器学习算法有不同的适应场景。因此，只有两者有机结合，相辅相成，AIOps才能为企业带来最直接最深远的价值。

AIOps 目前在国内外领先的互联网企业（如百度、腾讯、阿里等）开始被逐渐应用，也是近年来国内外被普遍看好的新技术之一。

2. 什么驱动AIOps的诞生？

回顾一下整个运维的发展历史，如下图所示：

我们可以发现，早期的手工运维大部分依靠运维人员手工完成，在互联网业务快速扩张、人力成本高企的时代，难以维系。于是自动化运维应运而生，其基于可被自动触发的、预定义规则的脚本，来执行常见的、重复性的运维工作，减少了人力成本，提高了运维效率，但随着互联网业务急剧膨胀，以及服务类型的复杂多样，“基于人为指定规则”的自动化运维的不足，日益凸显。随着人工智能时代的来临，AIOps的概念被提出，不依赖于人为指定规则，主张由机器学习算法自动地从海量运维数据（如日志数据，性能指标数据，网络抓

包数据，用户行为数据等）中不断学习，不断提炼并总结规则。

综上所述，基于机器学习的AIOps将是必然趋势，原因如下：

1. 庞大、复杂、多变的软硬件系统故障难以避免

2. 在保障业务可靠、高速、高效、安全运转的前提下，如何快速准确决策

3. 运维规则复杂、多变、无法人力维护

4. 海量、高速、多样、高价值的监控数据

3. AIOps核心要素

为了实现成本管理、效率提升、质量保障的场景，根据Gartner的定义，AIOps应包含下图所示的基本要素：

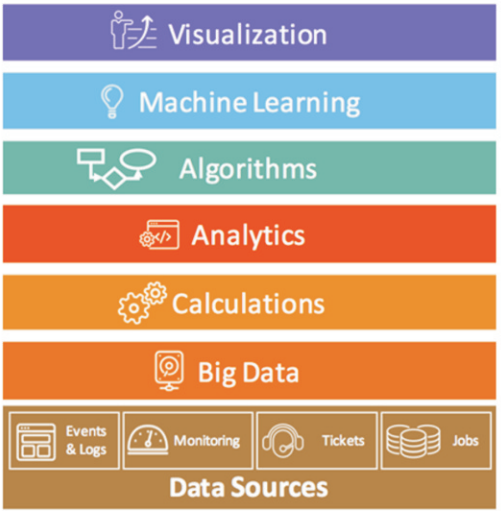


Figure 2: The technologies that make up an AIOps platform

数据源：大量并且种类繁多的IT基础设施（如事件、度量、日志、工作数据等）

大数据平台：用于处理历史和实时的数据

计算与分析：通过已有的IT数据产生新的数据（如数据清洗、去除噪声等）

算法：用于计算和分析，以产生IT运维场景所需要的结果

机器学习：一般指无监督学习，可根据算法的分析结果来产生新的算法

可视化：以一种易耗的方式提供对IT运营的洞察力和建议，以促进理解和行动

如上所述，AIOps通过分析机器学习产生的结果自动创建和应用响应或改进已识别的问题，最终实现运维的智能化。但为了将AIOps完全落地，还需解决诸如：数据采集、数据处理、数据存储、离线和在线计算以及面向AIOps的算法技术等一系列流程中所遇到的各种问题，目前一些大厂（如百度、腾讯、阿里等）已经开始了相关的落地实践，并取得了瞩目的效果，相信在未来的几年里，AIOps将大放异彩，为更多的企业带来最直接最深远的价值。

言 论（转载）

人生总有坎坷崎岖，风霜雪雨不断磨砺。痛苦快乐轮番交替，悲欢离合缠绕不息。雾里赏花微风柳絮，金秋硕果冬雪红梅。得失成败别挂心里，谋事在人成事天意。只要你是尽心努力，就能做到问心无愧，得不忘形失不气馁，耐心积累以待良机

（李道祥 http://dwz.cn/hY8KxeIX）

生活总有不如意，接受这个事实;自己总会犯错误，接受这个事实，因为它们都是最正常的规律。活着，就是要努力让生活更好，让自己更好，努力放大生活中的光明面，放大优化发挥自己的光明面。而不是做徒劳的事——阻止一切不好的事情发生。

（陈肇炫 https://weibo.com/u/6533399742?from=feed&loc=avatar）

往前走，别站在远远的地方观望，你就可以找到你的方向。

（公绪辉 http://dy.163.com/v2/article/detail/DP395VP10521F9BP.html）

DCatch: Automatically Detecting Distributed Concurrency Bugs in Cloud Systems

罗 鹏 推荐

“Dcatch: Automatically Detecting Distributed Concurrency Bugs in Cloud Systems”是国际顶级系统结构会议ASPLOS2017录用的文章，这次会议于2017年4月在中国陕西西安召开。本文提出了一个采用发生序（HB）模型查找分布式系统中分布式并发错误的方法，名为DCatch。

在大数据和云计算领域，分布式系统的可靠性极其重要。然而，分布式系统容易出现分布式并发错误。分布式并发错误隐藏于分布式云系统的巨大状态空间中，发生具有随机性，受分布式系统计算和通讯时间影响。已有研究发现，分布式并发错误广泛存在于现有分布式系统中，可导致各种系统问题，如数据破坏，系统崩溃和任务悬挂等。因此，查找分布式并发错误是一个亟需解决的问题。

由于分布式并发错误和传统多线程程序中的并发错误具有诸多类似之处，如都具有随机性，受运行环境影响，都由多个并发进程/线程交互引发。因此，查找分布式并发错误可采用查找传统并发错误类似方法，如经典且高效的查找并发错误模型，发生序模式。接下来，DCatch的工作是将分布式系统中的事件映射到发生序模型，也是查找分布式并发错误的核心。以下是对分布式系统中的事件进行模型映射分析，其中使用箭头（→）表示发生序（HB）。

节点间的并发与通信

1. 远程调用（RPC），其中一个节点可通过RPC远程调用另一节点分配任务，并可等待其完成：

新建任务 → 任务开始

任务结束 → 任务等待

2. 网络套接字（Socket）同步，套接字通过发送和接收消息进行通信：

发送消息 → 接收消息

3. 基于推送的同步协议，一个节点更新了状态可推送更新变化信息给其他节点：

更新状态 → 信息推送

4. 基于拉取的同步协议，一个节点可不停拉取另一个节点的状态变化信息：

更新状态 → 信息拉取

节点内的并发与通信

1. 多线程的并发同步，多线程程序通过新建线程和等待线程进行多线程同步：

新建线程 → 线程开始

线程结束 → 等待线程

2. 异步事件驱动并发，事件驱动进程中，事件可发生在任何线程：

新建事件 → 事件开始

事件1结束 → 事件2开始，如果事件1和事件2同属于一个单容器的FIFO队列，并且事件1先于事件2开始

3. 顺序程序序列，事件按发生顺序同步：

事件1 → 事件2，如果事件1和事件2同属一个线程，并且事件1先于事件2发生

事件1 → 事件2，如果事件1先于事件2发生，并且它们发生在一个事件处理，

或消息处理，或远程调用处理期间

接下来，DCatch基于以上规则建立的模型查找分布式并发错误。DCatch的工作分为四步，监视运行–分析trace-优化精简–触发错误，对应于四个组件：监视器、分析器、优化器和触发器。

监视器 DCatch需要追踪监视分布式系统的运行信息，以便进行分析查找分布式并发错误。监视的内容分为三个部分。第一个部分是内存访问信息，其中包括远程调用函数、网络套接字操作和事件处理函数。第二部分是与HB相关的操作，包括节点间和节点内并发与通信对应的事件。第三部分是同步操作原语，如加锁与解锁。

分析器 DCatch根据监视得到的trace信息，使用HB模型生成HB关系图，对关系图进行分析查找可能的分布式并发错误。为了生成HB关系图，DCatch遍历所有收集到的trace信息，其中包括各个节点上所有的进程和线程，并将每个记录都对应到关系图上一个顶点。然后，DCatch根据前面介绍的规则生成HB关系。最后，对关系图进行分析，定位所有可能的分布式并发错误。

优化器 DCatch静态分析可能的分布式并发错误，排除那些不会发生的分布式并发错误。DCatch进行本地分析和分布式分析。本地分析使用进程间分析和进程内分析，加入本地因素，排除那些不会导致错误的候选点。分布式分析主要分析远程调用函数以理解对内存访问的远程影响。DCatch不分析节点间套接字的影响，因为套接字通信不像远程调用那样可结构化。

触发器 Dcatch验证每一个查找到的分布式并发错误，并对其危害性进行评估。该工作包括两部分：（1）分布式系统中时间同步操作的基础架构；以及（2）使用基础架构来触发可能的分布式并发错误的分析工具。这两个特性对于触发分布式并发错误具有独特性。DCatch使用分布式的时间延迟来进行时间同步。危害性评估则根据结果进行分析。

BugID	LoC	Workload	Symptom	Error	Root
CA-1011	61K	startup	Data backup failure	DE	AV
HB-4539	188K	split table & alter table	System Master Crash	DE	OV
HB-4729	213K	enable table & expire server	System Master Crash	DE	AV
MR-3274	1,266K	startup + wordcount	Hang	DH	OV
MR-4637	1,388K	startup + wordcount	Job Master Crash	LE	OV
ZK-1144	102K	startup	Service unavailable	LH	OV
ZK-1270	110K	startup	Service unavailable	LH	OV

BugID	Detected?	#Static Ins. Pair			#CallStack Pair		
		Bug	Benign	Serial	**Bug**	Benign	Serial
CA-1011	✓	$\mathbf{3}_1$	0	0	$\mathbf{5}_1$	2	0
HB-4539	✓	$\mathbf{3}_3$	0	1	$\mathbf{3}_3$	0	1
HB-4729	✓	$\mathbf{4}_4$	1	0	$\mathbf{5}_5$	5	0
MR-3274	✓	$\mathbf{2}_1$	0	4	$\mathbf{2}_1$	0	9
MR-4637	✓	$\mathbf{1}_1$	2	4	$\mathbf{1}_1$	3	9
ZK-1144	✓	$\mathbf{5}_1$	1	1	$\mathbf{5}_1$	1	1
ZK-1270	✓	$\mathbf{6}_1$	2	0	$\mathbf{6}_1$	2	0
Total*		$\mathbf{20}_{12}$	5	7	$\mathbf{23}_{13}$	12	12

Dcatch使用了 WALA v1.3.5和 Javassist v3.20.0 实现，代码有12K行。DCatch分析查找的错误包括：本地显式错误（LE）、本地悬挂（LH）、分布式显式错误（DE）和分布式挂起错误（DH）。查找结果如上图所示。实验结果表明，DCatch对于分布式并发错误查找效果良好。

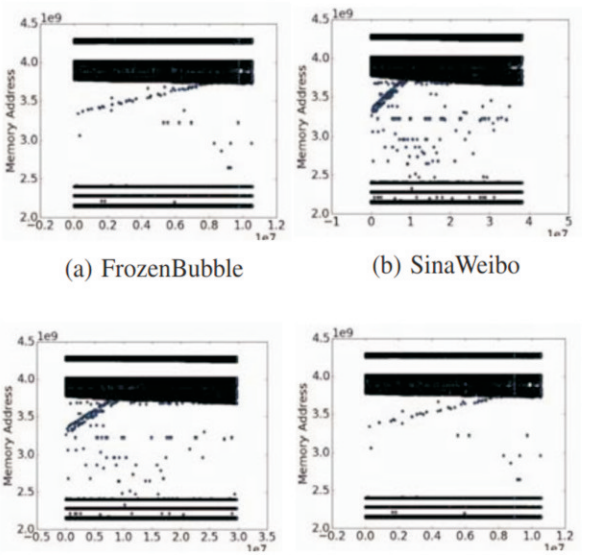
罗　鹏
2013级博士研究生
研究方向：系统安全
Email：pluo@hust.edu.cn

Exploring Non-Volatile Main Memory Architectures for Handheld Devices

王孝远　推荐

本次我推荐的文章是发表在DATE 2018 上的文章，文章主要论述的是筛选合适的系统内存架构可以在不明显降低系统性能的前提下降低系统功耗。

随着GPU、DRAM等专用设备的嵌入集成到多核CPU中，现在的片上系统（SoC）结构极其复杂，并且功耗极高。为了是能耗控制在一个合理的水平内，手机片上系统实际用的是一种低功耗的DRAM版本——LPDDR DRAM（low-power double data rate DRAM）。LPDDR 通过除去能效低效的电路，使得它非常适合手机端设备。但是随着手机端应用工作集（working set）的不断增加，传统的LPDDR DRAM 已经无法应对持续增加的巨大能耗问题。通过使用镁光的LPDDR2设备能耗计算工具，发现当系统内存从1G到2G的情况下，对于row-buffer 命中率较高的（85%）会增加平均12.9%的功耗，而对于row-buffer命中率较低的（15%），相关能耗增加为26.7%。并且DRAM部件能耗的增加会显著增加内存子系统功耗占系统功耗的比例。以上发现说明，受限于系统能耗，盲目的增加系统内存对于现有移动端发展是不可行的。文章展示了一种智能内存架构，通过分析工作负载的内存访问模式，在不影响用户体验的情况下提供大量的节能，文章指出使用该方式，可以在仅有2.8%性能损失的前提下，设计出具有显著节能效果的架构。



(a) FrozenBubble　(b) SinaWeibo

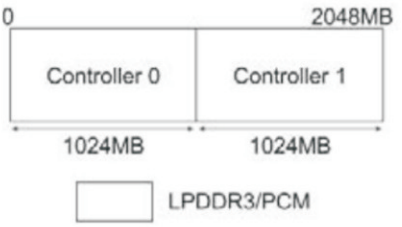
(c) Netease　(d) Baidu

图1. 内存访问分布图

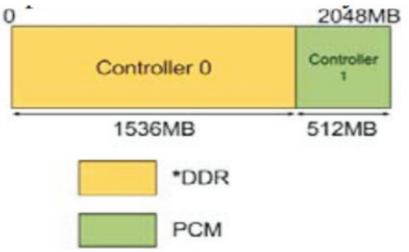
文章通过对Moby suite测试集中的测试用例进行测试，（如图1所示）发现绝大多数应用的工作集集中在有限的小部分地址空间，其中，Y轴代表物理地址，X轴表示程序运行期间特定地址空间被访问次数，访问越多，图中点越密集，颜色越深。文章发现，超过75%的访问集中在25%的特定地址空间内。所以，如果可以把频繁访问的数据放入LPDDR等较快的内存中，而把非频繁访问的数据放入PCM中，可以有效地改善系统能耗情况，并且不会显著降低系统性能。

为此，文章利用PCM 和 LPDDR3设计了多种异构内存系统，如图2所示，通过设计不同的

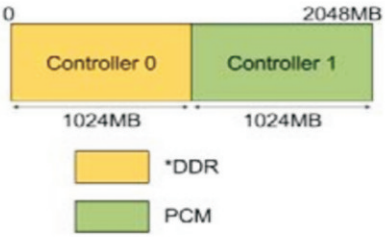
通道（channel）和内存控制器数目，文章实现了单通道单内存控制器系统（图2.a所示）、双通道双内存控制器系统（图2.b、图2.c所示）以及四通道四内存控制器系统。为了更好的研究内存容量对系统的影响，文章研究了不同配比情况下系统的性能差异（如图2.b和图2.c所示）。



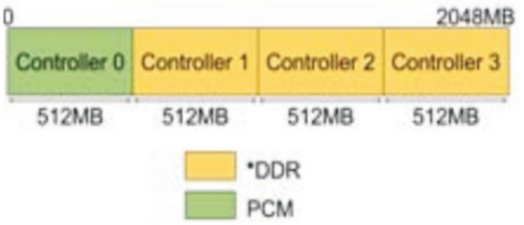
a)传统内存架构，单控制器纯LPDDR3/NVM



b)1/2LPDDR和1/2PCM双控制器内存系统

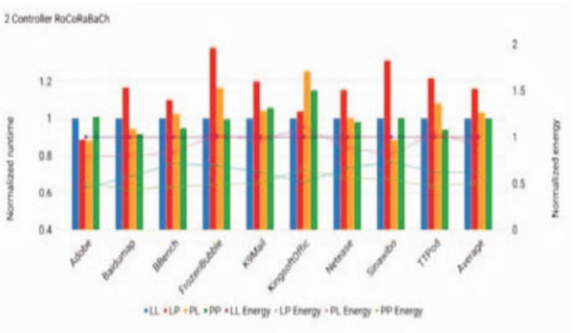


c)½ LPDDR和1/2 PCM双控制器异构内存系统

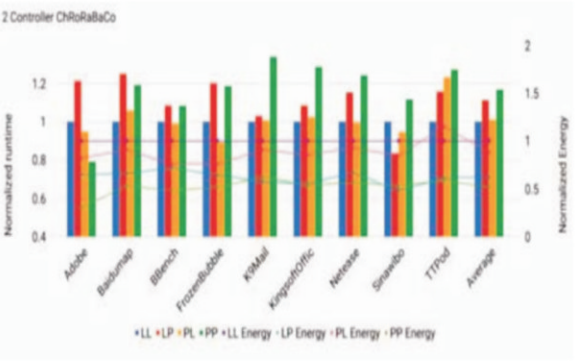


d)3/4LPDDR和1/4PC四控制器内存系统

图2. 多控制器的异构内存系统架构



(a) Default Address Mapping



(b) New/Proposed address mapping (ChRoRaBaCo)

实验表明，合理的内存架构可以在2.8%的系统性能降低情况下，节约多达61%的系统功耗。

王孝远
2014级博士研究生
研究方向：内存计算，操作系统
Email：xiaoyuanw@hust.edu.cn

青春的印记

陈　飞

时光真如白驹过隙，一转眼已在东五楼度过了整整五个春秋，五年的时光承载着在这里的美好回忆悄然散去，青春的印记里留下的是SCTS/CGCL永恒的足迹。我们在这里学习，我们在这里成长，我们从这里起航。

从一个学了点专业基础就恃才傲物的本科生到掌握一门领域的专业技术、对未知的知识世界充满敬畏的研究生，从一个懵懵懂懂、涉世未深的少年到一个能够独立思考、思想成熟稳重的青年大叔，从一个满脑子理论、bug面前唉声叹气的书生到一个能够直面挑战、从容应对各种bug的IT工程师，在实验室的经历和成长带给了我们太多的变化。

回首往昔的种种依旧历历在目，刚保完研决定留在本校SCTS/CGCL读研时，我们意气风发、自信满满；新生杯篮球赛场上，我们挥洒汗水、奋力冲刺；开题前夕，我们焦虑彷徨、挑灯夜战；文章投稿前，我们废寝忘食、反复检查；文章被拒稿后，我们鼓起勇气、拂去尘土；文章被录用了，我们欢呼雀跃、再接再厉……

在SCTS/CGCL的这些年，科研占据了生活的大部分时间。科研中的酸甜苦辣可以让我们浅尝人生的味道。从泛读论文、方向调研时的迷茫到idea初具雏形时的激动甚至辗转难眠，从面对海量源码、系统开发不知如何下手时的急切到系统调通、bug消除时的兴奋，从论文反复修改时的小心翼翼到文章被录用后的柳暗花明，凡此种种都深深嵌入到科研的每个环节之中。一份高质量的科研成果凝聚了作者大量的心血。阅读一篇高质量的美文不仅能学到专业知识，更是一种精神上的享受。

于我而言，我一直都很相信时间的力量，把时间当做最好的朋友。科研本来就是一个厚积薄发的过程，前期的积累是必不可少的。对于领域内知识的全面掌握需要花大量的时间去做调研，对于系统细节的深刻理解需要花大量的时间去阅读源码，对于文章的撰写和修订也需要花大量的时间去琢磨和思考。其间可能会遇到各种各样的困难，很多的困难都是可以被克服的。当我们站在一个长的时间尺度上来看的时候，我们往往会发现此时此刻遇到的困惑和障碍都是微不足道的，也许十年之后这段经历能够在你内心留下印记的恰恰是你当初为之苦恼而废寝忘食探索答案的过程。

往事已成追忆，需要感恩的人还有很多。清晰地记得在学术沙龙上，金老师广阔的视野和高瞻远瞩的大局观常常会让我在陷入具体的技术细节之后，能够及时抽身出来站在更高层次上旁观自己的工作，真有如醍醐灌顶；清晰地记得在晚上九点之后，灯火通明的办公室内和吴老师细致地讨论文章的整体结构、每一个章节、每一个段落、每一张图，每次的讨论都能让自己获益匪浅；清晰地记得刚进入实验室时，师兄师姐在科研入门和选题上的帮助和指导；清晰地记得小组例会上，和学弟学妹们一起总结各自的工作进展并互相提问、互相解惑，他们活跃的思维每次都能带给我审视问题不一样的视角。

金老师常常教导我们，要立足于当前国家和社会经济发展的需求来解决本领域内关键性的技术挑战，做有价值的科研。这不仅是一名老师对学生的谆谆教诲，更是一名计算机科学领域的先驱长者对后来人的殷切期望。扪心自问，自己的学术成果离这个要求确实还差得很远，希望后来人能够做得更好。

陈　飞
2013级博士研究生
研究方向：大规模分布式流计算系统
Email：fei_chen_2013@hust.edu.cn