上海超级计算中心
Shanghai Supercomputer Center

高性能计算发展与应用

Development & Application of High Performance Computing

[总第75期]

4

2022年

第60届TOP500排名

世界前五台高性能计算机

1

Frontier（前沿）
安装地点：美国橡树岭国家实验室（ORNL）
HPE制造
实测速度：1.102EFlops

2

Fugaku（富岳）
安装地点：日本理化研究所（RIKEN）
Fujitsu制造
实测速度：442PFlops

3

LUMI
安装地点：芬兰科学信息技术中心（CSC）
HPE制造
实测速度：309.1PFlops

4

Leonardo
安装地点：意大利CINECA中心
Atos制造
实测速度：174.7PFlops

5

Summit（山峰）
安装地点：美国橡树岭国家实验室（ORNL）
IBM制造
实测速度：148.6PFlops

目 录

从网格到“东数西算”：构建国家算力基础设施

钱德沛 栾钟治 刘轶
北京航空航天大学 计算机学院 北京 100083

摘要：
本文简要回顾了几十年来计算机使用方式的变迁，介绍了基于网络计算技术的国家高性能计算基础设施CNGrid的设计与实现。讨论了在“东数西算”战略工程背景下我国算力发展的新趋势和国家算力基础设施发展面临的新的技术挑战，并对我国未来超算应用生态和算力基础设施建设提出展望。

1. 计算机使用模式的演变

计算机是20世纪人类社会最伟大的发明之一，它的出现，彻底改变了人类生活、工作的面貌。计算与模拟和理论与分析、实验与观测一起，成为人类认识客观世界、开展科学研究的重要手段。对现代科学研究而言，计算的重要性不言而喻。利用计算，可以更清晰地揭示客观世界的发展规律，探索和预测未知的事物。例如，大数据处理分析、人工智能大模型训练和推理、新能源及其利用、新材料设计、工业产品创新设计、创新药物研发、精确天气预报、全球气候变化预测、社会治理和决策支持等等，都依赖计算机的强大算力。因此，算力已经成为一个国家创新能力和综合国力的体现。

另一方面，如何能更容易地使用计算机，便捷地获得所需的算力，也是人们一直追求的目标。计算机诞生70多年来，其使用方式一直在不断变化。早期，人围着计算机转，用户要跑到专门的机房去上机，计算机被一个用户单道程序所独占。随着操作系统的进步，计算机的使用方式逐渐从单道程序的主机模式向多道程序的批作业模式[1]和分时交互模式[2]发展。在分时计算系统[3]中，众多用户可以通过终端，同时使用一台计算机，每个用户在分配给他的时间片内使用计算机，但他感觉上，似乎是在独占这台计算机。80年代个人计算机的出现使计算机进入千家万户，但是个人计算机的性能有限，孤立的个人计算机难以胜任大的计算任务。直到20世纪另一个伟大发明计算机网络的出现和普及，带来网络计算的新变革，计算机的使用方式才发生了影响更深远的变化。所谓网络计算，就是通过网络连接网上分散的计算机，汇聚网络连接的各类硬件和软件资源，形成能力更为强大的计算系统。用户可以通过网络随时随地访问计算机，使用计算资源，完成自身任务，而无需关心计算资源的物理所在。

美国的超算中心联网是网络计算系统的早期范例。20世纪80年代中，美国国防用途的Arpanet进入商用，Internet诞生。在美国自然科学基金会支持下，建设了主干速率56Kbps、运行TCP/IP协议的NSFNET网络。NSFNET将美国加州大学的圣地亚哥超级计算中心SDSC、伊利诺斯大学的美国国立超级计算应用中心NCSA、康奈尔大学的康奈尔国家超级计算机研究室CNSF、匹兹堡超级计算中心PBC、冯·诺依曼国家超级计算中心JVNNSC和美国国立大气研究中心NCAR的科学计算分部连接起来，对大学和科研用户提供可远程使用的计算资源[4]。

上世纪90年代中，网格计算（Grid Computing）的概念在美国兴起[5]。网格（Grid）一词最初是指电力网（Power Grid），网格计算借用电力网的概念，提出要利用高速互联网把分布于不同地理位置的计算、数据、存储和软件等资源连为一体，通过调度、管理和安全保障机制，建立一个像电网一样的计算网格，把算力像电力那样输送给终端用户，支持共享使用和协同工作[6]。在美国国家科学基金会的支持下，分别由NCSA和SDSC牵头，实施了的两个网格计算项目，初步建立了计算网格的雏形[7, 8]。网格计算研究在上世纪末到本世纪最初十年达到高潮。在美国倡导下，成立了全球网格论坛GGF，与此对应，国际IT大公司联合成立了企业网格论坛EGF 。2006年，GGF和EGF合并，成为开放网格论坛OGF。GGF提出了开放网格服务基础设施OGSI和开放网格服务体系架构OGSA等标准[9, 10]，协调全球网格计算的研究和开发力量，研究资源管理，安全、信息服务及数据管理等网格计算基本理论和关键技

术。在Globus项目[11]支持下研发了Globus Toolkit 3.0 (GT 3) 软件，GT3作为OGSI的一个完整的参考实现，成为网格计算的事实标准。

在网格计算热潮中，美国、欧盟、日本、中国都实施了一批网格计算研究计划或项目。部分代表性项目如表1所示。

表1 世界部分网格相关研究计划

国别	网格计算相关研究计划/项目
美国	TeraGrid，XSEDE
欧盟	EGEE，EGI
英国	UK e-Science网格
日本	NAREGI，HPCI
韩国	K*Grid
中国	CNGrid（科技部），ChinaGrid（教育部）

美国是网格计算发源地，在该方向的研究计划持续时间最长，实施的项目数量最多。美国国家科学基金会NSF专门设立Cyberinfrastructure的部门，持续稳定支持网格计算方向的研究。美国的网格项目主要有两类，第一类由NSF支持，在PACI之后，从上世纪末开始实施TeraGrid项目[12]，其主要目标是用网格计算技术推动国家科技进步，保持美国的科技领先地位。2011年，TeraGrid的后继项目XSEDE项目[13]正式启动，该项目旨在连接全球的计算机、数据和研究人员，建立可供科学家共享的计算环境。美国NSF资助的开放科学网格（OSG）在其基础软件HTCondor[14]支持下，实现了众多大学与国家实验室的计算资源共享，为科学家提供了科学计算的环境。第二类网格项目由国防部、能源部等支持，其主要目标是更好地完成本部门的任务。两类研究的应用目标有所不同，但共同点是要发展先进的基于网络的应用基础设施（Cyber Infrastructure），实现应用层面的互联互通、资源共享、协同工作。

欧盟于2000年和2001年分别启动了欧洲网格计划（EuroGrid）[15]和欧洲数据网格计划（European DataGrid）[16]。在欧洲数据网格计划基础上，2004年3月，欧盟框架研究计划启动了EGEE（Enabling Grids for E-science in Europe）项目[17]，其目标是基于网格技术开发欧洲的服务网格基础设施，供科学家全天候使用。2011年，欧盟框架计划又启动了EGEE的后继项目EGI（European Grid Infrastructure）[18]。在这些项目支持下，研发了欧盟的网格中间件gLite[19]，建立了可持续运维的泛欧计算基础设施。

英国的网格研究计划即UK e-Science计划[20]的目标是用网格技术改变科学研究的模式，推动科学技术的进步，其长远目标是影响未来的信息技术基础设施。在UK eScience计划支持下，英国在大学和研究机构建立了一批国家e-Science中心，依托OMII-UK项目研发了英国的开放网格中间件，开发了一批面向e-Science的网格应用系统。

日本文部省MEXT（Ministry of Education, Culture, Sports, Science and Technology）在2003年启动了“国家研究网格基础设施”（National Research Grid Infrastructure）项目，简称NAREGI[21]。NAREGI构建在日本教育科研网SuperSINET之上，旨在研制并部署面向科学研究的网格基础设施，并参与全球开放网格组织OGF的工作，为网格的标准化活动提供支持。在NAREGI之后，日本政府又结合E级超级计算机的研制，启动了日本高性能计算基础设施项目HPCI。HPCI通过SuperSINET连接日本大学和研究机构中的十个大超算中心和两个大数据中心，形成日本的国家级计算基础设施。

中国的网格计算研究起步于上世纪90年代末，科技部是支持网格计算研究的主要政府部门，从1999年起，中国在高性能计算和网格方向连续实施了多个863重大项目和国家重点研发专项，表2列出了科技部在该方向支持的主要项目。在这些项目的持续支持下，研发了国家高性能计算环境系统软件CNGrid GOS和CNGrid Suite，使用环境系统软件，聚合了分布在全国各地近20个超算中心和高性能计算中心的计算资源，实现了资源的互联互通与统一共享、作业的提交与全局调度、数据的全局管理和环境的安全管控，在此基础上，成功构建了基于网格/网络计算技术的国家高性能计算环境-中国国家网格服务环境（China National Grid，简称CNGrid）[22]。CNGrid历经20余年发展，正从“可用”迈向“好用”。目前的聚合计算能力超过50亿亿次，存储容量近500PB，部署了600多个应用软件和工具软件，支撑了数千项国家科技计划项目和重要工程项目的研究工作，用户覆盖基础研究、工业设计、能源环境和信息服务等众多领域，极大促进了我国科技创新能力的提高，已经成为科学研究、技术创新、工程设计中不可或缺的新型信息基础设施。

2006年兴起的云计算是网络计算技术与应用模式的一次大变革。与以往由学术界主导的技术热潮不同，云计算从开始就是由IT公司提出并引领的。2006年3月，亚马逊公司推出弹性计算云EC2[23]，2006年8月，谷歌公司首席执行官埃里克·施密特在搜索引擎大会首次提出“云计算”的概念。此后，微软、戴尔、IBM等国际IT巨头和百度、阿里等国内互联网公司都纷纷跟进。在学术界，美国加州大学

表2 中国科技部的网格和高性能计算项目

项目来源	项目名称	执行周期	项目成果
国家863重大课题	国家高性能计算环境	1999-2000	4000亿次曙光3000；包含5个高性能计算中心的国家高性能计算环境原型
国家863重大专项	高性能计算机及核心软件	2002-2005	11.2万亿次曙光4000，5.36万亿次的联想深腾6800；国家高性能计算环境实验床“中国国家网格CNGrid”，8个结点，18万亿次计算能力；一批网格应用
国家863重大项目	高效能计算机及网格服务环境	2006-2010	4700万亿次的天河1A，3000万亿次的曙光6000，1071万亿次的神威蓝光；具有服务特征的国家网格服务环境CNGrid，11个结点，8000万亿次计算能力；一批网格和高性能计算应用
国家863重大项目	高效能计算机及应用服务环境	2011-2015	12.5亿亿次的神威太湖之光，10亿亿次的天河二号；以服务支持应用的国家高性能计算环境CNGrid，14个节点，20亿亿次计算能力；一批高性能计算应用
国家重点研发专项	高性能计算	2016-2021	E级计算机；初步具备基础设施形态的国家高性能计算环境CNGrid，19个结点，52亿亿次计算能力；一批高性能计算应用

伯克利分校也专门发文，阐述云计算的学术问题[24]。几年之内，云计算已从新兴技术发展成为全球的热点技术。云的资源被虚拟化，可以动态升级，资源被所有云计算用户通过网络方便地使用。云计算的出现改变了IT应用系统部署运行的方式。在传统IT应用模式下，应用部门需要自行采购计算机硬件和软件，在私有的计算系统上安装部署自己的应用软件，运行和维护应用系统。在云计算模式下，用户无须自行采购维护计算机，而是从云服务商那里租赁所需的计算资源，在云中安装特定的应用软件，存放应用的数据，完成应用系统的部署，然后应用系统就能够运行在云端。应用部门本身不需要采购和维护私有的计算机，当应用需求变化时，可以根据需要增加或减少租赁的云计算资源。服务和按用付费是云计算的商业模式，是计算向基础设施形态迈出的一大步。根据所提供的服务内容，云计算可分为IaaS（提供基础资源）、PaaS（提供平台服务）和SaaS（提供应用软件）[25]。根据服务的范围和应用的性质，云计算又可分为公有云、私有云和混合云[26]。按照服务封装部署方式又可分为虚拟机、容器、裸金属服务器等[27]。今天，几乎所有大数据中心都在某种程度上使用云计算技术，提供云服务。云计算技术也被引入传统的高性能计算领域，出现了以云方式运行超级计算中心的“云超算”和提供高性能计算能力的“超算云”[28]。

中国的云计算和国际同步发展。国家863计划在2010年就启动了“中国云”重大项目，支持阿里、百度等互联网公司研发云计算系统。十三五期间实施了“云计算与大数据”重点专项，更加系统全面地推进云计算关键技术和系统的研发与应用。今天，阿里云、华为云、百度云、浪潮云等已经在国内市场举足轻重。

物联网IoT、基于移动互联网应用的蓬勃发展催生了边缘计算[29]。边缘计算的目的是使应用程序、数据和计算能力（服务）更加靠近端用户，而不是更靠近集中的云，这样就能减少数据的移动，降低数据传输的延迟，降低端系统和数据中心之间的传输带宽需求，达到更低的成本和更好的用户体验的效果。随着边缘计算技术的进步，云-边-端融合的IT应用模式也日趋流行，成为渗透更宽广应用领域的网络计算的新形态。

2. 国家高性能计算基础设施CNGrid的实践

建设国家级高性能计算基础设施是创新型国家建设的战略需求。基于网络计算的计算基础设施具有如下特征：（1）动态性。系统的状态和行为动态变化，资源动态接入和退出、设备随时会出故障、网络可能拥塞甚至断开、用户的数量会不断变化等；（2）自治性。地理分散的资源在支持广泛共享的同时，仍能保持原有的隶属和管理属性；（3）开放性。硬件、软件和服务来自不同的厂商，由不同的团队开发，遵循不同的技术规范，兼容并蓄，形成自然生长演化的计算生态环境。这种动态、自治、开放的基础设施不同于资源集中拥有和控制的

云计算环境。

在开放、动态的互联网环境下，聚合网上异构、自治的分散资源，构建在全国范围共享使用的国家高性能计算基础设施，面临重大技术挑战:（1）在动态环境下如何应对系统资源的不确定性，对用户提供稳定的高质量服务；（2）在不改变原有资源隶属关系和管理模式的条件下，如何实现受控共享；（3）在开放异构的环境下，如何高效开发和运行大规模分布并行应用，建立高性能计算应用的生态环境。国家高性能计算基础设施CNGrid通过体系结构、系统软件、应用模式、应用开发与优化技术等创新应对上述挑战，为在我国形成高性能计算资源提供、应用开发和运行服务的完整产业链奠定了技术基础。

2.1 非集中层次虚拟化体系结构及系统软件

针对开放动态环境下分布异构资源的统一管理与受控共享、系统安全以及服务质量保障等重大技术难题，设计并实现了国家高性能计算基础设施“三横两纵”的非集中层次虚拟化体系结构，如图1所示。“三横”是指自底向上的内核系统层、系统服务层和应用层。内核系统层通过资源实体和虚拟组织等抽象，将地理分布、自治的高性能计算物理资源抽象和聚合为可动态划分、申请和调度的虚拟资源，通过运行时虚拟地址空间和自治安全策略，解决资源视图、资源发现及定位、异构资源统一访问等基础性问题。系统服务层通过访问虚拟资源，以服务化形式向上层应用提供作业管理、数据访问与传输、应用编程、用户映射等功能。应用层使用系统服务层提供的功能，实现应用的业务逻辑，服务最终用户，“两纵”是贯穿内核、服务、应用三个层次的环境监控管理和安全机制，保障环境的可管理性和安全性。

图1 国家高性能计算基础设施的非集中层次虚拟化体系结构

体系结构的非集中是指CNGrid的管控采用地理分布模式，即在每个CNGrid结点部署一台运行系统软件的服务器，通过覆盖网络将各个CNGrid结点动态组织成星型、网状或混合结构，以适应国家高性能计算基础设施对资源的分层分域管理的需求。

基于非集中层次虚拟化体系结构，研发了基础设施系统软件CNGrid Suite，其系统架构如图2所示。CNGrid Suite提出并实现了“资源实体”、“虚拟组织”和“运行时虚拟地址空间”等三个系统核心抽象，来表达系统中的各种资源要素、要素间的静态关系和运行时的动态关系，通过这些抽象，将分散、异构、无序的计算机硬件资源、软件资源和用户组织成逻辑有序、可受控共享的虚拟资源，支持资源的动态聚合、调度和安全访问。

图2 CNGrid Suite系统架构

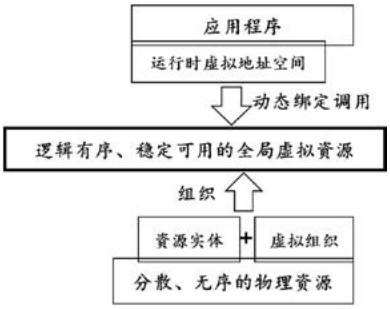


图3 基础设施系统软件核心系统抽象及相互关系

2.2 资源组织与作业调度

针对国家高性能计算基础设施特点和应用需求，提出了“资源实体”、“虚拟组织”和“运行时地址空间”等系统软件创新概念，应对资源描述、组织和访问的挑战，

在CNGrid中，用户、资源和社区都被统一抽象为资源实体。每个资源实体由一个全局id来标识，代表一个可以访问其他实体，也可被其它实体所访问的全局资源要素。系统软件的全局命名管理模块对资源实体实施统一管控，完成资源实体创立与消除、资源定位、资源解耦等功能。

多个相关资源实体可构成一个虚拟组织。虚拟组织描述资源实体之间的静态关系，实现资源实体注册、资源实体接入与剔除、资源实体元信息管理、资源实体权限管理、资源实体访问控制等功能。通过虚拟组织把资源实体组织成可有效管控、相互协同的资源集合。

运行时地址空间描述了资源实体间的动态访问和调用关系，结合动态绑定的安全策略，解决了资源命名、资源视图、资源发现及定位、资源统一安全访问等基础性问题。与传统操作系统的进程概念相对应，CNGrid提出了网程（Grip）概念，网程在运行时地址空间中，代表资源实体动态访问其他资源，实施访问控制，分配、管理和回收资源，实现应用的可控启动和终止。

CNGrid聚合了分布在不同地域、不同组织机构中的各种各样的高性能计算资源，面向用户提供统一的系统映像和透明的作业调度是基本需求。CNGrid的作业调度由服务端、驱动器和客户端三部分组成[30]。面向用户提供统一的访问入口和使用方式。系统软件根据用户作业请求性质为其自动匹配适当的高性能计算资源。另一方面，也提供开放接口，为作业调度模型和调度策略的优化提供了可能。作业调度核心模块由资源收集器、资源匹配器和资源调度器构成，通过引入多种优先级作业队列，细化作业的系统状态，改善了作业调度策略的可配置性[31]。

2.3 监控管理和安全机制

针对资源的分布性、异构性和动态性，服务和应用的多样性以及管理需求的各异性等特征，CNGrid提出了统一实体监控管理、管理功能动态构造、管理功能跨域动态部署及协同工作等创新概念与机制，设计了基于统一实体的监控管理体系架构，研发了一体化的监控管理系统（见图4），实现了CNGrid的资源监控和运行管理，为多层次资源的按需共享和自主协同提供了支撑。

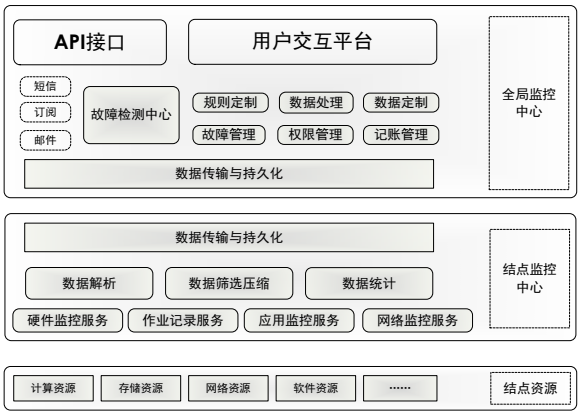


图4 一体化基础设施监控管理体系结构框架

与CNGrid系统软件中的资源实体抽象相对应，设计了基于实体的资源管理信息描述方法，采用统一的“被管对象”抽象建立全局信息模型，对各类资源信息进行有效的建模与表示，形成对不同层次、不同类别资源精确监控和管理的基础。

针对资源的动态性和多样性，提供了监控管理功能的动态生成、部署、运行的能力，支持管理功能的动态扩展和更新，实现监控管理系统的动态构造与演化。针对资源的跨域特点，提出了监控管理功能跨域动态部署的概念。基础设施的监控管理按分布层次式组织，监控管理功能分布在各个监控管理域中，各管理域既局部自治又相互协作。管理域设立自身的监控管理中心，形成多级监控管理中心的协同机制。同时，实现了基于复杂事件处理的监控信息高效获取、传输和控制的机制。这些措施有效减少了与监控管理有关的数据流量，降低了监控管理对基础设施正常应用业务的影响。

针对CNGrid环境下资源种类繁多、数目巨大的现状，设计实现了单维度、多维度以及基于日志等多种故障扫描、识别和应对方法，能够准确定位故障，分析故障根因，及时通告故障事件并推荐应对的策略，为提高CNGrid的可用性、可靠性和可管理性提供了保障。

CNGrid采用基于证书的身份认证和访问权限控制来保证其安全。系统软件采用代理证书实现用户认证和权限代理。首先，通过定义访问控制策略

形成资源共享的操作上下文。一个操作上下文包含用户在CNGrid中的身份信息，即用户的代理证书、用户所属的虚拟组织和所在的组别、以及由虚拟组织签发的访问资源的令牌。一个资源提供者可把资源注册到虚拟组织中，持续管理已注册的资源，把相应的权限分配给用户，控制应用本身占用的资源并进一步支持多个应用之间的协同。在运行时，由网程维护用户身份，实施访问控制。当用户想访问资源时，需要把其操作上下文从用户端传送到资源端，由资源端验证证书，根据证书权限控制访问[32]。CNGrid 中部署了证书认证中心（CA），用户可通过CA 中心提供的 Web 界面申请用户证书。 CNGrid安全机制基于公钥基础设施（PKI），使用标准的X.509证书，提供用户和资源的双向认证。

CNGrid的安全机制在权限控制的前提下，尽可能地支持基础设施资源的共享。受控共享是CNGrid提出的一个重要概念，其概念框架如图5所示。在受控共享机制下，只要访问控制权限允许，非属主用户和属主用户均可完成对资源的操作，此过程称为属主用户和非属主用户对资源的受控共享。

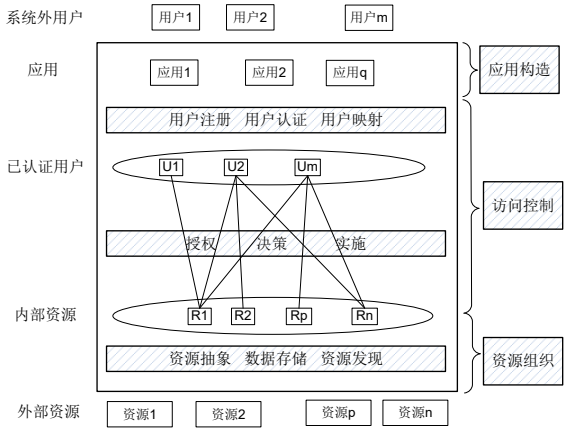


图5 受控共享概念框架

2.4 数据管理和高效传输

高效存储和访问分布、异构、自治的数据资源是CNGrid要解决的另一个关键问题。针对科学研究和行业应用的实际需求，设计并实现了基于虚拟数据空间的数据管理体系，有效集成了环境中的数据资源，构建了统一的数据管理空间，为用户提供了透明统一的数据存储、访问和管理能力。CNGrid的数据资源主要包括文件系统和数据库系统，虚拟数据空间为文件系统的集成共享提供虚拟文件系统，为数据库系统的集成共享提供虚拟数据库系统。在这两者之上，虚拟数据空间提供数据基础服务，简化了存储和数据的使用逻辑，为用户或应用提供便利。

针对CNGrid中数据分布存储和自治管理等特点，虚拟数据空间采用面向服务的分布式层次结构进行构建。设计了基于分布域的联邦数据存储管理机制，在各数据域的自治管理基础上实现全局统一管理，保障数据管理的可扩展性。通过数据域之间的协作来满足应用的分布式存储需求，系统根据用户的访问位置等信息实现数据资源的就近存储和管理，以便提高用户对数据的访问效率。设计实现了异构数据库的整合机制，以统一的接口实现对不同数据库管理系统的数据访问，并且通过并行机制保障在大规模分布式环境下的访问效率。

影响CNGrid中数据传输效率的主要因素包括单次传输的数据量、网络带宽的利用率和传输引入的额外开销，因此，提高效率的关键在于减小单次传输的数据量，充分利用网络带宽，降低传输额外开销。数据传输不可靠的主要原因是网络链路及主机的不稳定，增强可靠性的关键在于克服不稳定因素，减少数据传输错误造成的损失。CNGrid通过多个副本并行传输来提高带宽利用率，通过文件的分块传输来减少每次传输的数据量，通过就近传输来提高传输速度和可靠性，提供断点续传和三方传输来提高数据传输效率，减少额外开销。

CNGrid环境的动态变化特性使数据存储资源难以保证持续的服务。CNGrid引入数据副本管理机制来保证数据服务的可靠性。数据副本的引入带来数据一致性维护问题。为此设计实现了并行化的一致性有限状态机，有效降低了数据一致性维护的代价。数据按照其使用频度被定义为冷热数据，系统根据数据温度动态调整其副本数量，在提高访问效率的同时减少了不必要的开销。此外，还设计实现了位置及网络实时状态感知的数据副本放置策略，在保证一致性的同时提高了数据访问的效率。

CNGrid的数据管理服务通过灵活和自适应的数据访问授权控制，解决了数据安全性与环境复杂动态性之间的矛盾。它采用细粒度的访问控制策略，为不同的资源拥有者和使用者对不同粒度的数据资源的访问，提供个性化的访问控制策略，满足了自治性和个性化的要求。

2.5 基于应用社区的应用新模式

针对以公共计算平台支撑个性化领域应用的需求，CNGrid提出了体现领域应用特点的个性化领域应用社区概念。应用社区具有“批零”结合的资源管控与按需服务机制，既有网格聚合分散资源的能力，又有云计算集中管控、按用付费的特点，成为国家高性能计算基础设施的应用新模式。为了支撑应用社区的构建和运行，发展了领域应用中间件Xfinity，实现了多层次的用户管理机制、按域划分的

资源管理模式、基于模板的应用零开发热部署技术以及资源动态绑定的工作流技术等体系架构和关键技术创新。

按需定制的服务模式体现在服务方式和内容的定制，可为不同用户定制满足其特定需求的专用社区。通过社区动态配置、资源动态绑定与复用、应用按需集成与动态部署等技术，实现了服务的按需定制。按需付费的交易模式贯穿服务交易全过程。资源拥有者在社区发布资源与价格信息，用户通过社区选择能满足其需求且价格合适的服务。社区监督服务交易过程和服务完成情况，保证交易各方的利益。

为实现按需调配的资源管理模式，提出了权属策略灵活配置的社区资源管理技术，将特定资源组织成资源子域授权给不同用户使用。不同资源子域的用户相互隔离，互不干扰。用户对其资源子域拥有完全的支配权，可做更精细的分级授权管理，实现社区内资源的有效调配和充分共享。

社区通过基于角色的权限访问控制、组管理和双层映射等技术实现了多层次、分角色的用户与资源的精细管理。实现了与企业业务系统相容的低开销安全机制，允许独立制定和修改国家高性能计算基础设施、社区、企业这三个管理域的安全机制，在管理域之间建立信任关系和映射机制，消解各管理域不同安全策略间的矛盾。

工业创新设计社区是该新型应用模式的一个实例，其系统框架如图6所示。工业社区将国家高性能计算基础设施的计算服务推送到汽车制造、核电、飞机制造等行业的企业内部，加快了产品设计，降低了研发成本，提高了企业竞争力，取得很好的经济效益。

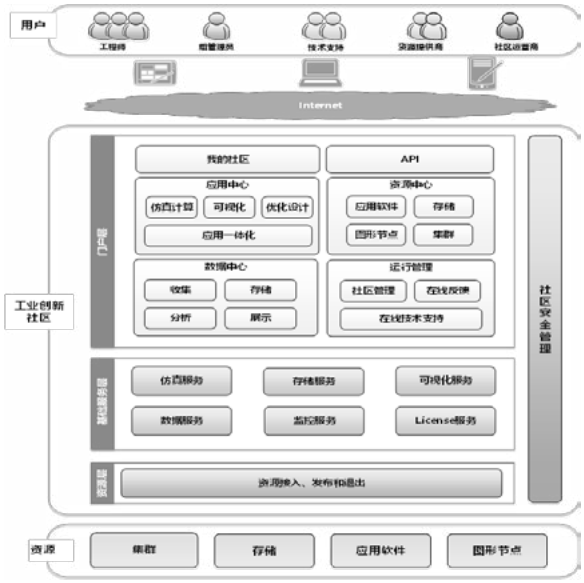


图6 工业社区系统框架

2.6 分布异构环境下应用软件的开发与优化

CNGrid的地理分布、资源异构的特征给大型应用的开发带来新的挑战。针对大规模应用的流程化与跨结点分布特征，CNGrid突破了构件与流程相结合的工作流编排、部署和运行技术，实现了流程在线组装、即时协作、即插即用的服务适配机制，支持分布资源的动态绑定、应用的快速开发和灵活部署执行，允许领域专业人员以低代码方式开发应用。

针对国产超级计算机多级并行和多种异构的特点，提出并实现了一系列并行程序优化方法和技术，例如：提出了节点间MPI、节点内OpenMP、处理器内多核并行的多级混合并行模式。提出了适用于不同异构平台的区域分解和动态负载平衡方法，通过动态可调的区域划分，实现加速器和通用处理器之间的负载均衡，隐藏加速器和通用处理器间的通信开销。提出了定制缓存及计算/访存重叠技术，充分发挥数据在片内核间的最佳重用。提出以定制DMA传输等方式实现计算和访存的最优化重叠，缓解内存带宽对应用整体性能的限制，大幅提升系统效率。研发了应用级断点保护技术，保证了大规模长时间作业的正确执行。发展了屏蔽硬件细节的并行算法库和编程接口，使不熟悉并行计算的应用领域专家能编写高效的并行应用软件。

在上述关键技术突破的基础上，研发了面向国家高性能计算基础设施的应用集成开发环境。集成开发环境包含基础算法库、应用模块库、程序模板库、优化工具库、拖拽式的工作流编排器、适配多种国产处理器的跨异构结点编译环境等，其系统架构如图7所示。开发人员可以使用集成开发环境中基于模板库的开发向导，自动生成程序代码框架，重用基本算法和模块库中的代码，快速构建应用程序，并在国家高性能计算基础设施中交互式地部署、优化和运行。

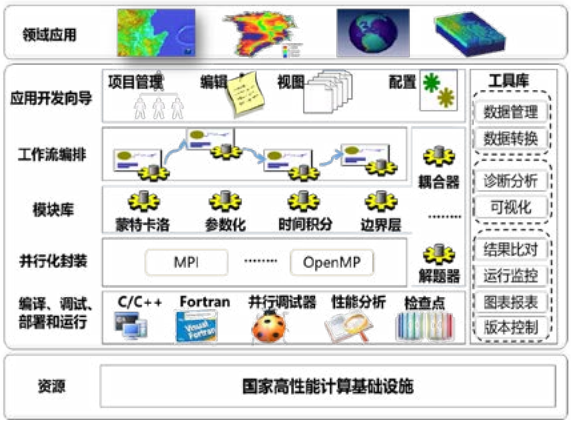


图7 高性能计算应用集成开发环境系统架构

3. 算力基础设施发展趋势与展望

3.1 新兴应用和技术趋势

3.1.1 新兴应用及算力需求

近年来，一系列新兴技术与应用的快速发展对算力基础设施提出了更高的要求。其中最具代表性包括人工智能、大数据和云计算等。

新一代人工智能的核心驱动力来自于深度学习技术。通过对多层大规模人工神经网络进行训练并用于推理，促进了计算机视觉、自然语言处理等领域的突破性进展。虽然人工神经网络概念的出现和应用已有数十年，但之所以近年来才取得快速发展，离不开算力的支持。深度学习是一种计算和数据驱动的技术，由于深度神经网络规模庞大，且通常需要使用大量的数据进行训练，这带来了巨大的计算量。表3[33]给出了几种典型神经网络模型的训练计算量，庞大的计算量使得神经网络训练通常需要借助加速部件进行，即使这样，一次模型训练也需要花费数小时到数天时间。

表3 典型神经网络模型的训练计算量

神经网络模型	应用领域	训练计算量 (浮点运算次数)
AlexNet	图像分类	4.7×10^{17}或470PFLOPs
VGG16	图像分类	8.5×10^{18}或8.5EFLOPs
YOLOv3	图像目标检测	5.1×10^{19}或51EFLOPs
Transformer	自然语言处理	7.4×10^{18}或7.4EFLOPs
GPT-3	自然语言处理	3.1×10^{23}或310ZFLOPs

注：数据来源于https://docs.google.com/spreadsheets/d/1AAIebjNsnJj_uKALHbXNfn3_YsT6sHXtCU0q7OIPuc4

大数据和云计算是另一种推动算力设施发展的新兴技术。在虚拟机和容器技术的支持下，人们可以在硬件平台上实现计算资源的灵活划分和隔离，以及软件环境的快速部署，这使得用户可以从云平台获得按需分配、可动态伸缩、易于部署且稳定可靠的硬软件平台和算力服务，这一特性吸引越来越多的用户将网站、业务平台和信息系统等迁移托管到云数据中心(国内俗称为“上云”)。由于企事业单位数量众多，此类应用的需求聚合到一起形成了对算力资源的庞大需求。

3.1.2 体系结构及算力设施的发展趋势

随着集成电路延续数十年的“摩尔定律（Moore's law）”减缓并走向停滞，计算机体系结构进入了变革期，多样化的体系结构不断涌现。为了在集成电路规模和性能增长减缓的背景下持续提升应用性能，定制化（customization）成为近年来体系结构发展的一大特点，即通过设计面向不同应用的加速部件/处理器，持续提升应用性能，典型代表有GPU（Graphic Processing Unit）和深度学习处理器/加速器。在GPU方面，除了Nvidia GPU外，传统处理器厂商AMD和Intel也陆续推出自己的GPU，国内也已研发出多款自主GPU芯片，为人工智能、科学计算、图形/图像处理等应用提供了高性能计算平台；在深度学习处理器/加速器方面，比较有代表性包括寒武纪、Google TPU、华为昇腾等，这些处理器/加速器专为神经网络计算而设计，其性价比和能效均优于通用CPU和GPU。

在多种新兴应用的推动下，各种处理器/加速器被应用于算力设施中，这带来了以下两方面的变化：

（1）算力中心内的异构化

在算力中心内部，异构已成为主流架构。表4给出了TOP500超级计算机排行榜中前十位高性能计算机的体系结构，可以看出，10台机器中仅有1台(富岳)采用同构架构，其他9台均为异构架构，除CPU+GPU结构外，我国的神威·太湖之光采用片内异构众核处理器，天河2A采用CPU+加速器结构。算力中心异构化的另一个体现是面向人工智能应用的异构体系结构，除了CPU+GPU结构外，CPU+深度学习处理器/加速器结构也被智算中心广泛采用，如CPU+Google TPU、CPU+寒武纪、CPU+华为昇腾等。

表4 TOP500排名前十的高性能计算机(2022年6月)[34]

排名	系统	处理器/加速器	Linpack性能(PetaFlops)
1	Frontier	AMD 64C+AMD MI250X	1102
2	Fugaku(富岳)	A64FX 48C	442.01
3	LUMI	AMD 64C+AMD MI250X	151.90
4	Summit(顶点)	IBM Power+Nvidia V100	148.60
5	Sierra(山脊)	IBM Power+Nvidia V100	94.64
6	Sunway TaihuLight(神威·太湖之光)	Sunway SW26010	93.01
7	Perlmutter	AMD 64C+Nvidia A100	70.87
8	Selene	AMD 64C+Nvidia A100	63.46
9	Tianhe-2A(天河2A)	Intel Xeon + Matrix2000	61.44
10	Adastra	AMD 64C+AMD MI250X	46.10

注：数据来源于http://www.top500.org

（2）算力中心的多样化(算力中心间的异构化)

传统的超算中心主要面向科学/工程计算，应用类型以并行数值模拟为主，主要特征是以双精度浮点运算为核心的计算密集型应用。与之相比，人工智能应用的计算类型主要是单精度/半精度浮点和定点运算，而大数据和云计算则以数据密集型应用为主。为了适应这些新兴应用的需求，算力中心的硬件配置也开始出现变化，出现了配置深度学习处理器/加速器、主要面向人工智能应用的智算中心，以及配置大容量内存和网络虚拟化设备、主要面向大数据和云计算应用的云算中心，同时，超算中心也开始支持人工智能和大数据应用。

3.2 算力基础设施面临的技术挑战

随着“东数西算”国家战略的实施，西部多个算力枢纽将建设算力中心，并面向东部经济发达地区提供算力服务。在这一背景下，如果由各个算力中心单打独斗，分散运营，则算力中心需投入人力物力自行发展用户，容易出现算力中心间的负载不均衡，导致算力碎片化和算力资源浪费；而在用户侧，由于各算力平台的硬件配置、软件资源、服务接口存在差异，也将给用户的软件开发和资源使用带来诸多不便。因此，通过将多个算力中心互连，向用户提供一站式、集成化的算力服务，形成覆盖全国的算力基础设施，对于提升算力资源利用效率和服务水平，促进国产软件和应用生态发展，支撑“东数西算”国家战略具有重要意义。

为了构建算力基础设施，需要解决算力中心异构化和多样化带来的诸多技术挑战，主要体现在以下几方面：

（1）计算任务在异构算力中心间的透明调度

算力基础设施必须具备的一项功能是：用户通过算力基础设施的服务平台提交一个计算任务后，可以直接得到计算结果，而无需关心这个计算任务在哪个算力中心上运行。这需要按照计算任务的类型和需求确定其所需资源，并根据各个算力中心的硬软件配置及可用资源数量进行任务分配和调度。例如，如果用户提交的是一个使用CUDA编写的可执行程序，那么就要在各算力中心寻找配置Nvidia GPU的节点并获取其当前使用状态，在此基础上进行调度；而如果用户只是希望在指定的数据集上完成深度神经网络模型训练，或者只是希望对所设计零件进行结构强度分析，那么计算任务的分派调度就不但要考虑硬件资源，还要考虑是否具备所需的软件资源。在以上基本调度功能的基础上，如果进一步考虑用户的服务质量和计费需求，如限定任务完成时间，限定资费水平等，任务调度要考虑的因素就更多。

（2）如何提供多层次、多样化的算力服务

算力服务的层次与云计算服务相似，也可分为基础设施(IaaS)、平台(PaaS)、应用软件(SaaS)三个层次。在基础设施层，算力服务以处理器/加速器/计算节点的形式提供；在平台层，算力服务在基础设施之上还提供系统软件和支撑软件，如高性能计算用户需要的MPI环境和基础算法库，人工智能用户需要的深度学习框架，大数据用户需要的分布式处理框架，等等；在应用软件层次，用户则可以直接使用不同种类的应用软件和服务。在算力基础设施中，算力中心的异构化和多样化使得算力服务变得更加复杂。以平台层为例，即使是高性能计算所需的MPI环境和基础算法库，在不同的算力中心中也会有不小的差异，用户在不同算力中心中使用这些服务时往往需要进行适配和修改(如修改作业脚本等)。由于算力基础设施需要向用户提供透明、一致的算力服务，如何屏蔽这些差异，就是需要着力研究和解决的问题。

（3）多算力中心分布式协同计算与虚拟超级计算机

在过去，由于算力中心间的网络带宽有限，计算任务通常分配在单个算力中心上完成，即使进行跨中心计算，一般也仅限于工作流中的不同计算阶段。近年来，随着网络基础设施和算网融合技术的发展，算力中心间的网络互连带宽不断提升，传输延迟显著下降。国内已有超算中心间网络互连达到传输带宽10Gbps、延迟接近1ms的水平，这给多个算力中心进行分布式协同计算提供了可能性。这一方面需要算力中心基础支撑软件互连互通构成分布式计算环境，另一方面，还需要研究在这种环境中的任务划分、调度和迁移等技术。更进一步，在互连带宽和延迟满足要求的情况下，是否可能将多个超算中心互连成为统一管理、统一调度的单一计算系统，形成可完成数倍于原有规模并行计算的“虚拟超级计算机”，也是值得研究和探讨的问题。

（4）多样化算力中心和异构化体系结构的编程问题

异构体系结构显著增加了并行编程的复杂性，这个问题在多样化算力中心场景下更加突出。为了支持异构处理器/加速器编程，厂商都推出了相应的编程语言/接口，如用于Nvidia GPU的CUDA、用于申威众核处理器的Athread、用于AMD GPU的ROCm/HIP等，并基于这些编程接口开发/移植了各种基础算法库、求解器、深度学习框架等，但异构平台编程的复杂度仍然远高于传统的CPU平台。在多样化算力中心中，异构硬件平台的种类更多，为

了使程序具有更好的平台适应性，在软件编程模型和语言方面还需要更多的工作。虽然近年来已经出现了一些独立于厂商的加速器编程接口，如OpenCL、OpenAcc、SYCL等，但这些编程接口在不同硬件平台上的实现仍然有差异，为一种平台编写的程序通常难以不加修改地在另一种平台上编译和运行。有鉴于此，仍然有必要提出独立于硬件平台且可屏蔽硬件细节的编程模型/语言，与此同时，研究开发异构程序转换工具，实现并行程序在不同硬件平台间的透明转换和自动编译，进而支持并行程序在多样化算力中心的透明调度和运行，也将是一项很有价值的工作。

（5）数据在分布式算力中心间的放置问题

无论是科学工程计算，还是人工智能或大数据应用，其数据规模都较为庞大。由于算力中心间的数据传输和访问开销较大，在算力中心间进行计算任务调度和迁移时，数据放置就成为必须考虑的一个重要因素。

（6）公共算力中心的数据安全和隐私问题

在人工智能领域，为了满足数据隐私和安全需求，人们提出了联邦学习技术(Federated learning)[35]，通过多个数据拥有者协同完成训练，避免了数据向其他实体公开。这种数据隐私和安全性需求同样存在于大数据分析、科学与工程计算等领域。为了使用公共算力中心的计算服务，用户往往需要将数据上传至算力中心，虽然通过VPN等技术可以保证数据在网络中传输时的安全性，但在多用户共享的算力中心中，数据在外存中的存放，以及计算过程中数据在内存的存放，仍然存在数据外泄的可能性。为满足对数据安全性要求较高用户的需求，如何在大数据分析和科学工程计算领域提供类似于联邦学习的机制，或实现“可计算但不可读写”，是值得深入研究的问题。

3.3 我国超算应用生态和算力基础设施未来展望

3.3.1 我国超算应用生态存在的问题

近年来我国高性能计算技术水平取得了长足进步，超级计算机研制水平已处于国际前列，拥有性能排名前列的超级计算机，生产和部署的高性能计算系统数量也世界领先。在高性能计算应用软件方面，面向国产超算系统研发了一批重点行业/领域应用软件，取得了众多的应用成果，大规模并行算法及应用也两次获得代表国际超算应用最高水平的Gordon Bell奖，但总体上，高性能计算软件与应用的发展相对不足，应用生态也不够丰富。产生这种现象的原因有多个方面：

首先，软件和应用研发投入不足。我国科研领域和产业界长期存在着“重硬轻软”的现象，在高性能计算领域，国家投入经费的大部分都用于高性能计算机系统研制，软件和应用处于配合和支撑地位，与发达国家超算研究计划中硬软件投入接近1:1相比，我国的软件和应用研发投入明显不足。

其次，软件和应用种类多、研发持续时间长。软件和应用种类繁多，应用生态的建立需要长期持续的努力。一种新型处理器/加速器推出后，初期往往只具备操作系统和编译等核心软件，而单靠硬件研发单位完成多种编程语言编译器、调试及性能分析工具、基础算法库、求解器、各种领域应用软件的研发非常困难，需要多方参与，经过若干年的持续努力，逐步研发和完善。

第三，用户使用习惯不利于国产软件的推广。我国超算应用软件研发起步相对较晚，西方国家在很多行业和领域已推出了商业化软件，用户已形成了商业软件使用习惯和对商业软件的认知度，某些行业甚至只认可某种软件的仿真结果，这给国内应用软件的自主研发和推广应用带来了很大困难。一种软件研发完成后，需要通过推广应用来支撑驱动软件维护和持续升级，通过软件持续升级，不断增强功能并改善用户体验，进而吸引更多用户使用，然而，目前国产超算软件研发还未能形成这种良性循环。

软件和应用生态存在的不足使得我国高性能计算领域存在着“大而不强”的现象，与此同时，高性能基础和应用软件大量依赖国外软件，也存在“卡脖子”的风险。

3.3.2 我国算力基础设施的发展展望

“东数西算”战略的实施将形成算力中心建设的新高潮，为了构建国家算力基础设施，需要在研发突破关键技术的基础上，补足我国超算软件与应用的短板，并通过运营模式和机制创新，建立起丰富且自我发展的国产软件应用生态。为此，需要在以下方面开展重点工作：

（1）研究突破关键技术，支撑算力基础设施发展

围绕算力中心异构化和多样化带来的技术挑战，解决算力基础设施面临的技术难题，研发核心软件和服务平台，实现多样化算力中心的互连互通、资源共享和服务提供，为国家算力基础设施的构建和发展提供技术支撑。

（2）强化算力软件研发，补足国产软件与应用短板

以国产处理器/加速器的兴起为契机，加强基于国产硬件的工具链、算法库、求解器、领域应用等基础和应用软件研发，通过若干年的持续努力，建

立较为完备的国产硬件支撑和应用软件栈，形成可自我发展的国产软件与应用生态。

（3）改变单一机时服务方式，推动算力中心能力建设

国内已建成的超算中心为了弥补运行经费的不足，普遍以提供机时服务为主，即俗称的“卖机时”。这种低层次的算力服务消耗了较多的人力和精力，制约了算力中心向更高水平发展。通过建立国家算力基础设施，可以推动算力中心从机时提供者向应用研发者和解决方案提供者转变。同时，各超算中心通过研发建立领域应用平台，可以突出自己的技术特色，进而形成算力中心各有所长的态势，也可以避免算力中心发展同质化。

（4）创新算力运营模式和机制，打造多方共赢的应用生态

算力中心涉及地方政府、投资方、设备提供方和运营方，在构建国家算力基础设施的过程中，需要以多方共赢为目标，通过运营模式和机制创新，鼓励多方参与，通过竞争促进技术进步和服务水平提升，与此同时，以应用商店（App store）等模式打造研发、服务、运营等多方共赢的软件和应用生态，进而推动国家算力基础设施做大做强。

4. 结论

CNGird在国家科技计划支持下历经20余年发展，已经成为不可或缺的国家高性能计算基础设施，并为“东数西算”背景下国家算力基础设施的建设积累了宝贵经验，奠定了技术基础。温故而知新。面对“东数西算”的新任务，要认真总结CNGrid的历史经验，分析应用和技术发展的新趋势，定位亟待解决的瓶颈问题，探索新的应用模式，更高效地构建新一代国家算力基础设施，实现“东数西算”的国家战略，完成国家创新发展赋予的历史使命。

参考文献：

[1] Jack B. Dennis. Segmentation and the Design of Multiprogrammed Computer Systems. [J] Journal of the ACM (JACM), Volume 12, Issue 4. Oct. 1965, pp 589 - 602.

[2] H. Sackman. Time - sharing versus batch processing: the experimental evidence. [C] AFIPS ‘68 (Spring): Proceedings of the April 30 - - May 2, 1968, spring joint computer conference. April 1968, pp 1 - 10.

[3] Jules I. Schwartz, Edward G. Coffman, Clark Weissman. A general - purpose time - sharing system. [C] AFIPS ‘64 (Spring): Proceedings of the April 21 - 23, 1964, spring joint computer conference. April 1964, pp 397 - 411.

[4] D. L. Mills, H. Braun. The NSFNET backbone network. [C] SIGCOMM ‘87: Proceedings of the ACM workshop on Frontiers in computer communications technology. August 1987, pp 191 - 196.

[5] Ian T. Foster, Carl Kesselman. The Grid: Blueprint for a New Computing Infrastructure. [B] Morgan Kaufman Publishers, 1998.

[6] Rick Stevens, Paul Woodward, Tom DeFanti, Charlie Catlett. From the I - WAY to the National Technology Grid. [J] Communications of the ACM (CACM), Volume 40, Issue 11. Nov. 1997, pp 50 - 60

[7] National Center for Supercomputing Applications (NCSA). [EB/OL] at https://www.ncsa.illinois.edu/

[8] M. Thomas, J. Boisseau, M. Dahan, et.al. Development of NPACI Grid Application Portals and Portal Web Services[J]. Cluster computing.2003, 6(3).177 - 188.

[9] I. Foster, K. Czajkowski, D.E. Ferguson, J. Frey, S. Graham, T. Maguire, D. Snelling, S. Tuecke. Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF. [J] PROCEEDINGS OF THE IEEE, VOL. 93, NO. 3, MARCH 2005. Pp604 - 612.

[10] Domenico Talia. The Open Grid Services Architecture: Where the Grid Meets the Web. [J] IEEE Internet Computing, Volume 6, Issue 6, November 2002. pp 67 - 71.

[11] Ian Foster, Carl Kesselman. Globus: A metacomputing infrastructure toolkit. [J] International Journal of Supercomputer Application. 11(2), pp115 - 129, 1998.

[12] Daniel A. Reed. Grids, the TeraGrid, and Beyond. [J] IEEE Computer, Jan. 2003, pp62 - 68.

[13] XSEDE website. [EB/OL] at https://www.xsede.org/.

[14] HTCondor website. [EB/OL] at https://htcondor.org/.

[15] European computational grid testbed. [EB/OL] at https://www.eurogrid.org/

[16] P.Kunszt. European DataGrid project: status and plans. [J] Nuclear Instruments and Methods in Physics Research Section A:

Accelerators, Spectrometers, Detectors and Associated Equipment
Volume 502, Issues 2－3, 21 April 2003, Pages 376-381
[17] F. Gagliardi, B. Jones, F. Grey, M. E. Bégin, and M. Heikkurinen. Building an infrastructure for scientific Grid computing: status and goals of the EGEE project. [J] Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 363, no. 1833, pp. 1729-1742, 2005.
[18] EGI: Open Ecosystem for Research and Innovation. [EB/OL] at https://www.egi.eu/.
[19] gLite, "Lightweight middleware for Grid Computing." [EB/OL] at http://glite.cern.ch/.
[20] Tony Hey, Anne E.Trefethen. The UK e-Science Core Programme and the Grid. [J] Future Generation Computer Systems. Volume 18, Issue 8, October 2002, Pages 1017-1031.
[21] S. Matsuoka, S. Shinjo, M. Aoyagi, S. Sekiguchi, H. Usami, K. Miura. Japanese Computational Grid Research Project: NAREGI. [J] PROCEEDINGS OF THE IEEE, VOL. 93, NO. 3, MARCH 2005
[22] 中国国家网格(CNGrid). [EB/OL] at http://www.cngrid.org.
[23] Amazon EC2 website. [EB/OL] at http://aws.amazon.com/ec2.
[24] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. [R] EECS Department University of California, Berkeley Technical Report No. UCB/EECS-2009-28, February 10, 2009
[25] Manish Saraswat, R.C. Tripathi. Cloud Computing: Analysis of Top 5 CSPs in SaaS, PaaS and IaaS Platforms. [C] 9th International Conference on System Modeling & Advancement in Research Trends, 4th－5th, December, 2020, Moradabad, India.
[26] Borja Sotomayor, Rubén S. Montero, Ignacio M. Llorente, Ian Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. [J] IEEE INTERNET COMPUTING, Volume: 13, Issue: 5, 2009. Pp14-22.
[27] Rabindra K. Barik, Rakesh K. Lenka, K. Rahul Rao, Devam Ghose. Performance analysis of virtual machines and containers in cloud computing. [C] 2016 International Conference on Computing, Communication and Automation (ICCCA), 4.29-4.30, 2016, Greater Noida, India.
[28] Josh Simons. HPC Cloud Bad; HPC in the Cloud Good. [C] 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, 20-24 May 2013, Cambridge, MA, USA.
[29] Nitesh Mor. Edge Computing: Scaling resources within multiple administrative domains. [J] Queue (QUEUE), Volume 16, Issue 6November-December 2018, Pages: 60, pp 106－116.
[30] 乔健，查礼. 中国国家网格作业管理设计与实现. [J] 计算机应用. 第28卷，第8期，2008年8月，pp2003-2009.
[31] 王小宁，肖海力，曹荣强. 面向高性能计算环境的作业优化调度模型的设计与实现. [J] 计算机工程与科学. 第39卷，第4期，2017年4月，pp 619-626.
[32] 喻林，邹永强，查礼. CNGrid GOS安全:设计与实现. [J] 华中科技大学学报(自然科学版). 第38卷，第S1期，2010年6月，pp 6-10.
[33] J Sevilla, P Villalobos, J Cerón, et.al., Parameter, compute and data trends in machine learning [EB/OL]. [2022-5-30]. https://docs.google.com/spreadsheets/d/1AAIebjNsnJj_uKALHbXNfn3_YsT6sHXtCU0q7OIPuc4
[34] TOP500 List [EB/OL]. June 2022 [2022-6-20]. https://top500.org/lists/top500/2022/06/
[35] K Bonawitz, H Eichner, W Grieskamp, et.al., Towards federated learning at scale: system design [C] //Proceedings of the Conference on Machine Learning and Systems (MLSys2019), Palo Alto, CA, IEEE Press, 2019:1-15

第60届全球超级计算机TOP500排名及分析

陈家慧 上海超级计算中心 上海 201203 jhchen@ssc.net.cn

摘要：

在2022年11月，第60届全球超级计算机TOP500榜单揭晓。本文介绍了TOP10情况及TOP500排名分析，并介绍了GREEN500等其他榜单排名情况。

关键词：超级计算机、TOP500、GREEN500

1. 综述

TOP500榜单的第一个版本出现在1993年6月于德国举行的一个小型会议上。在1993年11月，作者出于好奇重新审视了这个榜单以了解排名发生了怎样的变化。也正是在那个时候，他们决定继续编辑这个榜单，如此形成了现在这个一年两次备受期待、备受关注的活动。2022年11月，TOP500步入了第60届。

2022年11月14日，SC（全球超级计算大会）公布了第60届全球超级计算机TOP500榜单，美国橡树岭国家实验室（ORNL）的Frontier继续占据榜首，并且仍然是唯一一台HPL计算性能达到1Exaflop/s的超级计算机。在上一届TOP500榜单中，Frontier以1.102Exaflop/s的HPL计算性能使得美国重回领头羊地位。

虽然Frontier此次没能将其在2022年6月份达到的HPL计算性能进行提高，不过它还是以三倍于第二名的性能获得了冠军，这是计算机科学的重大胜利。此外，Frontier获得了7.94EFlop/s的HPL-MxP性能，HPL-MxP基准测试用于测量混合精度的计算性能。Frontier基于HPE Cray EX235a架构，采用AMD EPYC 64C 2GHz处理器，拥有8,730,112个内核，能效为52.23Gflops/watt，依靠千兆ethernet进行数据传输。

在TOP500榜单中，日本理化学研究所（RIKEN）的Fugaku以442Pflop/s排名第二，欧洲高性能计算联盟芬兰科学信息技术中心的LUMI排名第三。LUMI之所以能够排名第三要归功于系统的升级，使得机器规模得以翻倍，HPL性能从151.9Pflop/s提高到309.1Pflop/s，目前该机器仍然是欧洲最大的系统。LUMI拥有一个绿色中心，其废热用于满足约20%的城市供热需求。

进入TOP500榜单前10的唯一一台新机器是欧洲高性能计算联盟意大利CINECA的Leonardo系统，排名第四。该机器由3,000多个Nvidia HGX节点构成，每个节点有4个Nvidia A100 GPU和1个Intel Lake CPU，HPL计算性能达到0.174Exaflop/s。这台具有1,463,616个内核的水冷系统，其节点间通过Nvidia Mellanox HDR 200Gb/s InfiniBand进行互联。

本届TOP500榜单有41个新系统，新系统中有27个Lenovo系统，其中大部分是云/超大规模系统。此次，中国有2台Lenovo制造的新系统进入TOP500，。

目前，TOP500系统的综合性能从6个月前的4.40Eexaflop/s上升到了现在的4.86Exaflop/s，每台系统的平均并行规模由6个月前的182,864内核上升到了现在的189,586内核。

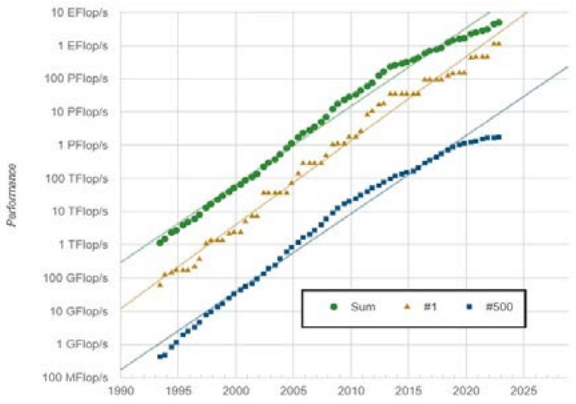


图1 历年TOP500性能发展情况

2. TOP10情况介绍

#1：Frontier，部署于美国田纳西州橡树岭国家实验室。

这台HPE Cray EX系统是美国第一台在HPL基准测试中计算性能超过1Exaflop/s的超级计算机，由DOE（Department of Energy，美国能源部）运营。

目前，Frontier拥有8,730,112个内核，性能达到1.102Exaflop/s。该系统基于HPE Cray EX架构，配置有经过优化的适用于HPC和AI的第三代AMD EPYC处理器和AMD Instinct 250X加速器，节点之间通过Slingshot-10互联。

#2：Fugaku，部署于日本理化学研究所计算科学中心。

Fugaku拥有7,630,848个内核，HPL基准测试性能达到442Pflop/s。

#3：升级后的LUMI，部署于欧洲高性能计算联盟芬兰科学信息技术中心。

目前，LUMI性能为309.1Pflop/s，同样采用HPE Cray EX235a架构。欧洲高性能计算联盟汇聚欧洲资源，以开发用于处理大数据的E级计算机。

#4：Leonardo，部署于欧洲高性能计算联盟意大利CINECA中心。

Leonardo是一台Atos BullSequana XH2000系统，以Xeon Platinum 8358 32C 2.6Hz为主处理器，NVIDIA A100 SXM4 40 GB作为加速器，Quad-rail NVIDIA HDR100 Infiniband为互联网络，性能为174.7Pflop/s。

#5：Summit，部署于美国田纳西州橡树岭国家实验室。

Summit由IBM构建，性能为148.8Pflop/s。Summit拥有4,356个节点，每个节点包含两个Power9 CPU，每个CPU有22个内核和六个NVIDIA Tesla V100 GPU，每个GPU具有80个流式多处理器（SM，streaming multiprocessor）。这些节点通过 Mellanox 双轨EDR InfiniBand网络连接在一起。

#6：Sierra，部署于美国加利福尼亚州劳伦斯利弗莫尔国家实验室。

Sieera架构与No.4系统Summit非常相似。Sieera由 4,320个节点构成，每个节点具有两个Power9 CPU和四个 NVIDIA Tesla V100 GPU。Sieera性能达到了94.6Pflop/s。

#7：Sunway TaihuLight，部署于中国江苏省国家超级计算无锡中心。

Sunway TaihuLight由中国国家并行计算机工程与技术研究中心研制，性能达到93Pflop/s。

#8：Perlmutter，部署于美国劳伦斯伯克利国家实验室。

Perlmutter是基于HPE Cray“Shasta”平台的异构系统，其配置了AMD EPYC节点和NVIDIA A100加速节点，性能达到了64.6Pflop/s。

#9：Selene，部署于美国NVIDIA公司内部。

Selene是NVIDIA DGX A100 SuperPOD系统，基于AMD EPYC处理器，采用NVIDIA A100加速，Mellanox HDR InfiniBand作为互联网络，性能达到63.4 Pflop/s。

#10：Tianhe-2A（Milky Way-2A），部署于中国广东省国家超级计算广州中心。

Tianhe-2A由中国国防科技大学研制，性能达到61.4 Pflop/s。

表1 TOP10系统及配置信息

排名	国家	计算机名	系统配置	核心数（个）	Rmax (TFlos/s)	Rpeak (TFlop/s)	power (kW)	制造商
1	美国	Frontier	HPE Cray EX235a、优化的第三代AMD EPYC 64C 2GHz、AMD Instinct MI250X、Slingshot-11	8,730,112	1,102	1,685.65	21,100	HPE
2	日本	Fugaku	A64FX 48C 2.2GHz, Tofu interconnect D	7,630,848	442.01	537.21	29,899	Fujitsu
3	芬兰	LUMI	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11	2,220,288	309.10	428.70	6,016	HPE
4	意大利	Leonardo	BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 40 GB, Quad-rail NVIDIA HDR100 Infiniband	1,463,616	174.70	255.75	5,610	Atos
5	美国	Summit	IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband	2,414,592	148.60	200.79	10,096	IBM
6	美国	Sierra	IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband	1,572,480	94.64	125.71	7,438	IBM / NVIDIA / Mellanox
7	中国	Sunway TaihuLight	Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway	10,649,600	93.01	125.44	15,371	NRCPC

8	美国	Perlmutter	HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10	761,856	70.87	93.75	2,589	HPE
9	美国	Selene	NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband	555,520	63.46	79.22	2,646	Nvidia
10	中国	Tianhe-2A	- TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000	4,981,760	61.44	100.68	18,482	NUDT

3. TOP500排名分析

3.1 装机总量

中国和美国在TOP500中占据的席位最多，其中美国有126台机器，中国则从上一届的173台减少为162台。在TOP500榜单中，中国和美国的装机量接近2/3，但很明显，其他国家正在努力实现自己的HPC创新。事实上，从整个大陆来看，欧洲在TOP500中占据131席，而在上一届榜单中他们的席位数是118台。

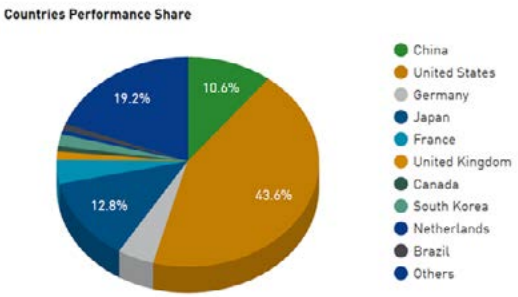


图1 TOP500各国装机量份额图

3.2 处理器

Intel仍然是TOP500系统中最大的处理器厂商，占有率达到75.80%，相较于6个月前的77.60%有所下降。Xeon芯片在TOP500系统中应用普遍，最新入榜的Leornardo采用的就是Xeon Platinum处理器。

AMD处理器也是受欢迎的HPC系统芯片，其占有率从6个月前的18.60%上升到如今的20.20%。TOP500系统中有101个系统采用AMD处理器，例如排名第一的Frontier和排名第三的LUMI。

3.3 加速器/协处理器

在TOP500榜单中，总计有179个系统使用了加速器/协处理器技术，相较于6个月前的169个系统有所上升。这些系统中的84个系统采用NVIDIA Volta芯片，64个系统采用NVIDIA Ampere。

3.4 互联网络

与上一届榜单相比，互联网络略有变化。采用Ethernet互联网络的机器从226台增加到233台，采用Infiniband的机器从196台减少为194台，采用Omnipath的机器从40台减少为36台，采用专有网络的机器从6台减少为4台。

3.5 榜单准入门槛

TOP500的准入门槛提升到了1.73Pflop/s HPL计算性能，本届第500名系统在上一期TOP500排名中位列第460名。TOP100的准入门槛更是提升到了5.78Pflop/s HPL计算性能。

4. 其他排名介绍

4.1 Green500排名

Green500榜单也是每年发布2次，基于LinPack基准测试模型计算出的系统能效，评选出最节能的超级计算机系统。相比TOP500，Green500更加重视超级计算机的能耗问题，而不仅仅追求运算速度。

Henri在Green500排名中位列第一。Henri部署于美国Flatiron研究所，基于Lenovo ThinkSystem SR670, 配备Intel Xeon Platimun、Nvidia H100和InfiniBand HDR。尽管在TOP500中排名第405位，但是Henri在能效方面表现出色，其能效为65.09GFlops/Watts，拥有5,920核心，HPL基准测试性能为2.04PFlop/s，理论峰值为5.42PFlop/s。

Frontier TDS（Frontier Test & Development System）在Green500排名中获得了第二名。Frontier TDS部署于美国ORNL，其能效为62.68GFlops/Watts，总计有120,832核心，HPL基准测试性能为19.2PFlops/s，在TOP500中排名第32位。鉴于Frontier TDS只是一个机架，与实际Frontier系统使用的机架相同，因此有理由相信该机器要比排名第一的Henri系统强大。

Adastra系统获得了Green500的第三名。Adastra部署于法国GENCI-CINES，是一台HPE Cray Ex235a系统，配备AMD EPYC和AMD Instinct MI250X芯片，其能效为58.02GFlops/Watts，HPL基准测试性能为

46.1PFlops/s，在TOP500中排名第11位。

Frontier风格系统在Green500中表现出色。在上一届榜单中，该类系统Frontier TDS、Frontier、LUMI和Adastra占据了前四的位置，它们都是由HPE构建，同样配备 AMD Epyc“Milan”CPU、AMD Instinct MI250X GPU和Slingshot 11网络。今年，这类系统中加入了采用类似架构的 Pawsey's Setonix GPU分区和KTH's Dardel GPU分区。在本届榜单中，Frontier TDS、Adastra、Setonix GPU、Dardel GPU、Frontier和LUMI分列第2到第7位。

表2 Green10系统及配置信息

排名	国家	计算机名	系统配置及制造商	核心数（个）	Rmax (TFlos/s)	Power (kW)	能效 (GFlops/watts)	TOP500排名
1	美国	Henri	Lenovo ThinkSystem SR670 V2, Intel Xeon Platinum 8362 2800Mhz (32C), NVIDIA H100 80GB PCIe, Infiniband HDR Lenovo	5,920	2.04	31	65.091	405
2	美国	Frontier TDS	HPE Cray EX235a、优化的第三代AMD EPYC 64C 2GHz、AMD Instinct MI250X、Slingshot-11 HPE	120,832	19,20	309	62.684	32
3	法国	Adastra	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11 HPE	319,072	46.10	921	58.021	11
4	澳大利亚	Setonix- GPU	GPU - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11 HPE	181,248	27.16	477	56.983	15
5	瑞典	Dardel GPU	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11 HPE	52,864	8.26	146	56.491	68
6	美国	Frontier	HPE Cray EX235a、优化的第三代AMD EPYC 64C 2GHz、AMD Instinct MI250X、Slingshot-11 HPE	8,730,112	1,102.00	21,100	52,277	1
7	芬兰	LUMI	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11 HPE	2,220,288	309.10	6,016	51.382	3
8	法国	ATOS THX-.A.B	BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 40 GB, Quad-rail NVIDIA HDR100 Infiniband Atos	25,056	3.50	86	41.411	159
9	日本	MN-3	MN-Core Server, Xeon Platinum 8260M 24C 2.4GHz, Preferred Networks MN-Core, MN-Core DirectConnect Preferred Networks	1,664	2.18	53	40.901	359
10	法国	Champollion	Apollo 6500, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 80 GB, Mellanox HDR Infiniband HPE	19,840	2.32	60	38.555	331

4.2 HPCG500排名

HPCG（High Performance Conjugate Gradient，高度共轭梯度基准测试）采用共轭梯度法求解大型稀疏矩阵方程组，用于衡量超级计算机解决大规模稀疏线性系统等实际问题的能力。众所周知，HPL是计算密集型的，而HPCG是数据传输基准测试，对系统

内存、网络延迟要求相对更高。HPCG旨在补充HPL基准测试，提供评估超级计算机性能的替代指标，帮助人们更全面的了解机器。

Fugaku在HPCG500排名中继续位列第一位，其HPCG基准测试性能为16.0PFlop/s，比峰值的3%略多。Frontier以14.054PFlop/s HPCG性能（峰值的0.8%）排名第二，升级后的LUMI以3.408PFlop/s排名第三。

表3 HPCG10系统及配置信息

排名	国家	计算机名	系统配置及制造商	核心数（个）	Rmax (TFlos/s)	HPCG (TFlops/s)	TOP500 排名
1	日本	Fugaku	A64FX 48C 2.2GHz, Tofu interconnect D Fujitsu	7,630,848	442.01	16004.50	2
2	美国	Frontier	HPE Cray EX235a、优化的第三代AMD EPYC 64C 2GHz、AMD Instinct MI250X、Slingshot-11 HPE	8,730,112	1,102.00	14054.00	1
3	芬兰	LUMI	HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11 HPE	2,220,288	309.10	3408.47	3
4	美国	Summit	IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband IBM	2,414,592	148.60	2925.75	5
5	意大利	Leonardo	BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 40 GB, Quad-rail NVIDIA HDR100 Infiniband Atos	1,463,616	174.70	2566.75	4
6	美国	Perlmutter	HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10 HPE	761,856	70.87	1905.44	8
7	美国	Sierra	IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband BM / NVIDIA / Mellanox	1,572,480	94.64	1795.67	6
8	美国	Selene	NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband Nvidia	555,520	63.46	1622.51	9
9	德国	JUWELS Booster Module	Bull Sequana XH2000 , AMD EPYC 7402 24C 2.8GHz, NVIDIA A100, Mellanox HDR InfiniBand/ParTec ParaStation ClusterSuite Atos	449,280	44.12	1275.36	12
10	沙特阿拉伯	Dammam-7	Cray CS-Storm, Xeon Gold 6248 20C 2.5GHz, NVIDIA Tesla V100 SXM2, InfiniBand HDR 100 , HPE	672,520	22.40	881.40	21

4.3 HPL-MxP排名

HPL-MxP（以前称为HPL-AI）用于评测超级计算机的混合精度计算性能，利用现代硬件的新型混合精度算法求解线性方程组，强调高性能计算（HPC）和基于机器学习及深度学习的人工智能（AI）工作负载的融合。传统HPC使用64位浮点运算，如今的硬件则提供多级别的浮点运算精度，例如32位、16位甚至是8位。HPL-MxP基准测试表明在计算期间使用混合精度可以获得更高的性能，并且与直接64位精度相比，混合精度技术通过数学方法

表4 HPL-MxP排名前5系统及配置信息

排名	国家	计算机名	核心数（个）	HPL-MxP (Eflop/s)	HPL Rmax (Eflops/s)	HPL-Mxp相对HPL加速	TOP500排名
1	美国	Frontier	8,730,112	7.942	1.1020	7.2	1
2	芬兰	LUMI	2,174,976	2.168	0.3091	7.0	3
3	日本	Fugaku	7,630,848	2.000	0.4420	4.5	2
4	意大利	Leonardo	1,463,616	1.842	0.1682	11.0	4
5	美国	Summit	2,414,592	1.411	0.1486	9.5	5

可以计算相同的精度。

在HPL-MxP基准测试中，Frontier性能达到了7.9EFlop/s，排名第一。LUMI以2.2Eflop/s排名第二，Fugaku以2.0EFlop/s排名第三。HPL-MxP排名前五的系统如表4。

5. 小结

TOP500榜单始于1993年，是对全球已安装超级计算机进行排名的知名榜单，每半年发布一次。在2022年11月发布的榜单中，Frontier继续位于榜首，中国则有2台超级计算机进入前10。

发展至今，TOP500对超级计算机的评价更为全面。TOP500最初使用Linpack程序对超级计算机进行基准测试，取前500个最优秀系统进行排名并对外公布。如今的榜单则包括了高性能共轭梯度（HPCG）等基准测试结果，并且Green500项目的数据收集和管理已与TOP500项目整合。相比TOP500，Green500更加重视超级计算机的能耗问题，而不仅仅追求运算速度。在云和超大规模的发展趋势下，超级计算机需要计算性能和绿色节能兼备，才能更好地掌握HPC领域的主动权。

从TOP500看，由CPU和加速器共同组成的异构算力是上榜计算机的主流选项，加速器/协处理器技术的应用越来越普遍。异构多元算力是超级计算机在从传统的高性能计算向HPC与AI并重转变的具体体现，揭示了当今超级计算机的应用领域不仅仅是高性能计算，还涵盖了人工智能以及其他更为广泛的领域，并且具有研究与商业兼具的特点。可见，应用领域的拓展影响着超级计算机的发展，二者相辅相成。

参考文献：
[1] TOP500，https://www.top500.org
[2] HPCwire，https://www.hpcwire.com

浅谈流体力学与机器学习融合的现状与发展前景

徐 莹[编译] 上海超级计算中心 上海 201203 yxu@ssc.net.cn

随着近些年机器学习、深度学习等技术的发展，学界开始关注机器学习与流体力学融合的可能性，包括提高模拟的速度，开发具有不同保真度的湍流模型，并经典方法精度更高的降阶模型[1-3]。几位作者通过机器学习改善流体力学的实验技术、控制应用等[3, 4]，在CFD的机器学习中，具体的应用例如湍流模型[5, 6]以及空气动力学对传热的优化[15]。本文将结合NASA CFD 2030计划[1]以及Burnton在Nature Comp. Sci.[2]及ARFM[4]的综述论文，介绍机器学习与流体力学结合的情况及发展方向。

1. 流体动力学中机器学习的挑战与机遇

科研及研发中积累了大量的数据，从海量数据中抽取、挖掘有用的信息已经成为一种新的科学研究模式，也成为一种新的商业模式。我们正在经历：1）海量数据的不断增加；2）计算硬件的进步以及计算、数据存储和传输成本的降低；3）强大的算法；4）大量的开源软件和基准测试；5）对数据驱动。随着科学研究越来越多地从第一原理转向数据驱动研究方法，当前机器学习的研究力度和发展前景可以与20世纪40年代和20世纪50年代数值方法的发展以解决流体动力学方程相提并论。

流体动力学所面临的挑战不同于机器学习的许多应用，如图像识别和自然语言处理等。流体动力学涉及复杂的多尺度现象，非定常流场需要能够解决非线性和多个时空尺度的算法，这些在流行的机器学习算法中实现都有困难。此外，机器学习的许多重要应用，如图形分类等，依赖于系统评估和学习过程的详尽分类，这些实验方法通常可复制或自动化。在流体中，实验可能难以重复或自动化，模拟可能需要超级计算机长时间运行。

流体动力学广泛应用在交通、健康、工程、化工等系统中，机器学习解决方案必须是可解释和可推广的。目前的许多机器学习算法无法确保结果的收敛。将机器学习和第一原理模型相结合，将是科研发展的沃土。

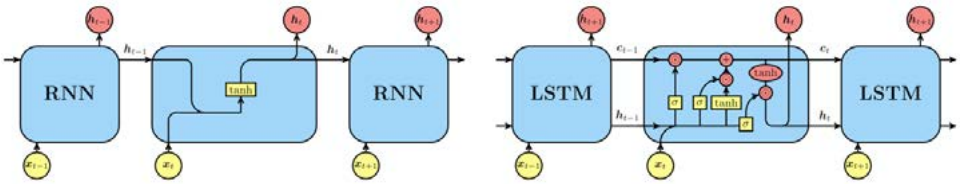


图1 用于时间序列预测和长短期记忆的递归神经网络[4]

图1所示的递归神经网络（RNN）在流体力学中应用有广阔的前景。它们对数据序列（例如，来自视频的图像、时间序列等）进行操作，其权重通过时间反向传播（BPTT）获得。RNN在自然语言处理和语音识别方面已经非常成功，RNN架构考虑了数据的固有顺序，经典神经网络在信号处理方面有一些开创性应用RNN的有效性受到训练过程中出现的梯度减小或爆炸的影响。Graves[7]提出了一种称为多维LSTM网络（MD-LSTM）的扩展架构，可以有效地处理高维时空数据。

强化学习（RL）是一个用于解决问题的数学框架[8]，它暗示了主体与其环境的目标导向交互。强化学习与动态规划密切相关[9]，将与环境的交互建模为马尔可夫决策过程。和动态编程不同，RL不需要动态模型，例如马尔可夫转移模型，而是通过反复试验和错误和环境交互来进行。强化学习的这种近似使得它非常适合于流体动力学中的复杂问题。深度学习在游戏中取得了成功，例如Go[10]和AI gym[10, 11]，

强化学习需要大量的计算资源，需要大量的事件来正确描述代理和环境的交互。对于游戏来说，这一成本可能较小，但在实验和流场模拟中，计算资源需求会变得巨大。

2. 流体模拟与机器学习

在过去的几个世纪里，第一性原理是流体建模的基础，对于高雷诺数，使用流体力学中的Navier-Stokes方程进行全尺度模拟对计算资源要求极高。另一种方法是根据这些方程的近似值或在实验室进行实验模拟。对于迭代来说，模拟和实验是代价高昂，对于实时控制模拟无法满足时效性要求。因此，通过降阶模型模型的大量研究，在保证精度的情况下，可以降低试验和模拟的成本，以捕捉流场的基本属性。这一领域大致可以分为两个方向：降维模拟和降阶建模。降维包括关键特征和主要模式的提取，这些特征和模式用作简化坐标，对流体进行有效地描述。降阶建模将流场的时空演化描述为参数化的动力学系统。

机器学习构成了用于数据驱动系统识别和建模的模块化算法的快速增长体。数据驱动的流场建模需要用到部分先验知识（主要是第一性原理）包括控制方程、约束和对称性的。随着模拟能力和实验技术的进步，流体动力学正成为一个数据丰富的领域，机器学习算法加入和融合日益成为潮流。模式识别和数据挖掘是机器学习的核心优势，大量技术手段可以用于流场时空数据。

深度学习通常需要大量可用的训练数据，这些数据量远超网络参数量。当新输入数据的概率分布与训练数据不同时，所得到的模型通常适用于插值，但可能不适用于外推。在机器学习应用中，例如图像分类，训练数据庞大，可以很自然地可以预期大多数分类任务将落在训练数据的插值内。尽管有大量的实验和模拟数据，流体力学界的数据工作范式与机器学习的数据规范相差很大。在未来几年中，可能会出现足够大、标记过的和完整流体数据库，以促进深度学习算法的应用。

聚类和分类是机器学习基础应用之一。分类在流体动力学中也被广泛用于区分各种典型流场和动态区域。近几年，有学者使用神经网络从局部涡度测量中研究了俯仰翼型后面尾流拓扑的分类[12]。神经网络已与动态系统模型相结合，以检测流动扰动并估计其参数[13]。

许多机器学习都集中在图像科学上，提供了基于统计推断的提高分辨率、消除噪声和腐败的强大方法。这些超分辨率和去噪算法有可能提高流体模拟和实验的质量。PIV实验数利用小区域内的超高分辨数据来提高大成像域上的分辨率。Fukami等人[14]开发了一种基于CNN的超分辨率算法，证明了该算法在湍流重建中的有效性，并确保能量谱的准确性。深度卷积神经网络已用于从PIV图像对构建速度场[15]。

使用机器学习研发湍流闭合模型是一个活跃的研究领域[16]。湍流中的时空尺度范围极广，数值模拟中解析所有尺度的计算量极大，解析工业生产的配置中的所有尺度在短期内无法实现。截断小尺度并用闭合模型模拟小尺度流场对大尺度的影响，常见的方法包括雷诺平均Navier-Stokes(RANS)和大涡模拟(LES)。这些模型需要仔细调整以匹配并直接模拟或实验的数据。已经报道的研究包括，用随机森林建立雷诺应力张量差异的监督模型[17]，用贝叶斯框架中的稀疏在线速度测量来推断雷诺应力张量差异等。

3. 在流体优化与控制中使用机器学习

学习算法非常适合于涉及“黑箱”或多模态成本函数的流程优化和控制问题。这些算法是迭代的，成本函数的计算成本比基于梯度的算法多几个数量级[18]，学习算法也不能保证结果的收敛。同时，强化学习等技术已被证明优于甚至最优的流量控制策略[19]。对不同的流体控制和优化问题，有多种机器学习算法可选。流场的优化或控制问题描述为学习参数的概率分布，并实现成本函数的最小化。这些概率分布是根据优化过程中获得的成本函数样本构建的。此外，目前用于训练非线性学习机的高维和非凸优化程序非常适合于流控制中的高维、非线性优化问题。

流量反馈控制根据传感器的测量值带动传动机构，修改流体动力学系统。反馈对于非稳态系统的稳定、传感器噪声的衰减、外部干扰和模型不确定性的补偿是必须的。流量控制的挑战包括高维系统、非线性、潜在变量和时间延迟等。机器学习算法已广泛应用于控制、系统识别和传感器布置。Lee等人[20]率先将神经网络应用于湍流控制，用很少的表面阻力传感器，使用局部壁面正吹和吸力来减小湍流边界层的表面摩擦阻力。对具有复杂高维非线性，众多传感器和致动器的系统，神经网络控制可能需要较高的计算或实验资源。自这些应用以来，神经网络的训练时间已经提高了几个数量级，这需要进一步研究这些神经网络在流量控制方面的潜力。

遗传算法已经用于解决流量控制问题，遗传算法要求控制律的结构是预先指定的，并且只包含几个可调整的参数，一个流体控制设计的案例是遗传算法用于后向台阶[21]。但是，大多数控制律需要经过1000次以上的测试评估获得，而在风洞实验中只需要几秒钟。

4. 结论与展望

本文编译了Burnton等研究者发表的流体力学与机器学习相结合的综述文章，汇总了机器学习在关键流体力学任务中的一些成功案例，如降维、特征提取、PIV处理、超分、降阶建模、湍流闭合模型、形状优化和流量控制等。机器学习包括基于数据驱动的优化及应用回归，这些技术非常适合于流体动力学这样高维非线性问题。在大数据时代，流体力学知识和几个世纪以来建立原理如守恒定律等，可以帮助将问题描述的更精准，并有助于减少机器学习算法在流控制和优化中相关的大量计算成本。

机器学习需要强大的算法，这些算法与流体流动的建模、优化和控制相关，研究者需要具备机器学习和流体力学的专业知识。流体力学传统上关注大数据。几十年来，它一直使用机器学习来理解、预测、优化和控制流程。目前，机器学习能力正在以前所未有的速度发展，流体力学开始充分挖掘这些强大方法的潜力。流体力学中的许多任务，例如降阶建模、形状优化和反馈控制，可以被提出为优化和回归任务。机器学习可以显著提高优化性能并缩短收敛时间。机器学习也用于降维，识别低维流形和离散流动状态。流量控制策略传统上基于精确的顺序：从理解开始，然后建模，然后控制。机器学习范式表明，数据驱动和第一原则方法之间的序列和迭代更具灵活性。

参考文献：

[1] J. Slotnick, A.Khodadoust, Juan Alonso, D. Darmofal, W. Gropp, E. Lurie, D. Mavriplis “ CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences, ” NASA/CR–2014 - 218178, 2014.

[2] Vinuesa, R., Brunton, S.L. Enhancing computational fluid dynamics with machine learning. Nat Comput Sci 2, 358–366 (2022). https://doi.org/10.1038/s43588 - 022 - 00264 - 7

[3] M. Brenner, J. Eldredge, and J. Freund. Perspective on machine learning for advancing fluid mechanics, Physical Review Fluids, Vol. 4, No. 10, 100501 (2019).

[4] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine learning for fluid mechanics. Annual Review of Fluid Mechanics, Vol. 52, pp. 477–508 (2020).

[5] K. Duraisamy, G. Iaccarino, and H. Xiao, “ Turbulence modeling in the age of data, ” Annual Review of Fluid Mechanics, vol. 51, pp. 357–377, 2019.

[6] S. E. Ahmed, S. Pawar, O. San, A. Rasheed, T. Iliescu, and B. R. Noack, “ On closures for reduced order models—a spectrum of first - principle to machine - learned avenues, ” Physics of Fluids, vol. 33, no. 9, p. 091301, 2021.

[7] Graves A, Fern´andez S, Schmidhuber J. 2007. Multi - dimensional recurrent neural networks. Artifi - cial Neural Networks–ICANN :549 - 558.

[8] Sutton RS, Barto AG. 2018. Reinforcement learning: An introduction, vol. 2nd Edi - tion. MIT Press.

[9] Benard N, Pons - Prats J, Periaux J, Bugeda G, Braud P, et al. 2016. Turbulent separated shear flow control by surface plasma actuator: experimental optimization by genetic algorithm approach. Exp. Fluids 57:22:1–17.

[10] Mnih V, Kavukcuoglu K, Silver D, Rusu Aa, Veness J, et al. 2015. Human - level control through deep reinforcement learning. Nature 518:529–533

[11] Silver D, Huang A, Maddison CJ, Guez A, Sifre L, et al. 2016. Mastering the game of Go with deep neural networks and tree search. Nature 529:484–489

[12] Colvert B, Alsalman M, Kanso E. 2018. Classifying vortex wakes using neural networks. Bioinspi - ration Biomim. 13:025003

[13] Hou W, Darakananda D, Eldredge J. 2019. Machine learning based detection of flow disturbances using surface pressure measurements, In AIAA Scitech

[14] Fukami K, Fukagata K, Taira K. 2018. Super - resolution reconstruction of turbulent flows with machine learning. arXiv preprint arXiv:1811.11328

[15] Lee Y, Yang H, Yin Z. 2017. PIV - DCNN: cascaded deep convolutional neural networks for particle image velocimetry. Exp. Fluids 58:171

[16] Duraisamy K, Iaccarino G, Xiao H. 2019. Turbulence modeling in the age of data. Annu. Rev. Fluid Mech.

[17] Wang JX, Wu JL, Xiao H. 2017. Physics - informed machine learning approach for reconstructing Reynolds stress modeling

discrepancies based on DNS data. Phys. Rev. Fluids 2:034603
[18] Bewley TR, Moin P, Temam R. 2001. DNS-based predictive control of turbulence: an optimal benchmark for feedback algorithms. J. Fluid Mech. 447:179–225
[19] Novati G, Mahadevan L, Koumoutsakos P. 2019. Controlled gliding and perching through deep reinforcement learning. Physical Review Fluids
[20] Lee C, Kim J, Babcock D, Goodman R. 1997. Application of neural networks to turbulence control for drag reduction. Phys. Fluids 9:1740–1747
[21] Benard N, Pons-Prats J, Periaux J, Bugeda G, Braud P, et al. 2016. Turbulent separated shear flow control by surface plasma actuator: experimental optimization by genetic algorithm approach. Exp. Fluids 57:22:1–17

要闻集锦

美政府再追加6千万美元，发展Aurora超级计算机

为应对气候变迁及绿色能源的开发需求，美国能源部近日获得美国政府拨款，其中有将近6,000万美元将投入在超级计算机Aurora的洁净能源研究项目。

美能源部上周获得拜登政府在《降低通胀法案》（Inflation Reduction Act of 2022，IRA）中获得15亿美元补助。该法案预计投入3,700亿美元于温室气体减排及能源安全，被视为是美国史上单一最大的气候与能源投资。这15亿美元将用于推进能源部辖下国家实验室的相关研究。

在这15亿中，有5,400万美元将用于打造Aurora exascale等级的超级计算机，这计算机是位于能源部下的阿贡国家实验室（Argonne National Laboratory，ANL）的先进运算设施（Leadership Computing Facility）。Aurora预计明年完成构建，将成为全球最快的超级计算机，它将协助多项高运算任务，像是有效洁净能源、新药开发，以及环保电子与国家安全加密技术的研发。

这也是Aurora近年获得最大笔的经费。能源部于2015年投入2亿美元，于2018年启动结合三座国家实验室，包括橡树岭（Oak Ridge）、ANL及劳伦斯利弗莫尔（Lawrence Livermore）国家实验室之力打造Aurora的计划。去年秋天ANL也在Aurora计划上获得了120万美元的拨款，投入《能源创新高性能计算》（High Performance Computing for Energy Innovation，HPC4EI）计划，启动4项能源制程及材料开发的研究。

ANL上个月表示，Aurora预定将推动高分辨率大脑连接体（connectome）构建、可视化癌症细胞扩散、核融合能源电浆扰动预测、以及寻找宇宙构成元素的研究。

Aurora连同橡树岭国家实验室领先运算机构正在打造的Frontier，美国将拥有两座运算速度达exascale的超级计算机。美能源部在去年宣布了第三座Exascale超级计算机Polaris的打造计划。

（卢永捷）

非MDS码存储系统的通用可靠性模型

聂世强 郑旭达 刘钊华 伍卫国 董小社 张兴军
西安交通大学 西安 710049 nsqiang@gmail.com

摘要：

为了量化基于非最大距离可分码的分布式存储系统可靠性，从非最大距离可分码的构造矩阵入手，提出了一种求解采用非最大距离可分码编码的数据对象在丢失若干块后数据对象的可修复概率算法，该算法穷举丢失若干块的所有可能组合，并在生成矩阵中判断每种组合相对应的矩阵是否可逆以计算可恢复的概率，随后采用马尔科夫理论，针对此类系统建立较为通用的度量存储系统可靠性的理论模型，该模型能够量化非最大距离可分码容错配置、存储规模、修复带宽、单节点可靠性、单节点容量对存储系统可靠性的影响，最后采用数值分析的方法以局部修复码为例验证了模型的正确性和比较了不同因素对存储系统可靠性的影响。本模型为采用非最大距离可分码的存储系统的设计和实现提供理论基础。

关键词：存储系统、可靠性、非最大距离可分码、马尔科夫模型、平均数据丢失时间

容错机制在大规模分布式存储系统是不可或缺的。大规模存储系统由成千上万台服务器组成，诸多研究报告指出节点失效成为常态[1][2][3]。近年来谷歌等大型数据中心的统计数据表明，平均每天都会有1%-2%的节点失效[1]。服务器失效引起数据丢失造成的损失是无法估量的。目前存储系统常用的数据冗余方法有多副本、纠删码等，多副本将数据复制多份分别存放在不同的存储节点，只要数据的副本所在节点不同时失效，数据便不会丢失[19]，纠删码是将数据分割为相等的数据块，采用编码策略生成校验块，部分数据块丢失后，可以通过编码恢复[4]，不同于多副本对存储空间的大量需求，纠删码可以显著降低存储开销因此被广泛使用，如云存储系统：Giza[6]、Hybris[8]等。然而目前纠删码在可靠性、存储利用率等方面都不同程度地存在着缺陷，难以同时达到理想的状态[7][10]。

可靠性可以判断系统或设备是否具备持续有效提供正确数据服务的能力，因此在存储系统中，可靠性是与性能和费用等指标重要性相当的一个评价标准。为了探究存储系统可靠性与诸多因素的关系，很多学者都对存储系统可靠性展开了广泛的研究。早期如Patterson采用马尔科夫模型分析磁盘矩阵系统可靠性，并以平均数据丢失时间（Mean Time To Data Loss，MTTDL）作为可靠性评价指标[11]，当前研究的方向有考虑数据放置算法[12]、数据中心物理拓扑结构[20]等因素对系统可靠性的影响，穆飞等针对大规模副本存储系统建立了马尔科夫模型度量了系统规模、副本阶数、节点容量等对可靠性的影响[13]，黄成等提出了在Windows Azure存储系统使用局部修复码（Locally Repairable Codes，LRC）以降低数据修复过程中的网络传输数据量，并建立了马尔科夫模型对LRC码与里德-所罗门码（Reed-Solomon codes，RS）进行可靠性的比较[14]。同时也有大量对于纠删码的研究[9][16][17]，胡玉鹏等人根据存储介质的可靠性提出变长的UFP-LRC码[18]，并对其可靠性进行数值分析，郝晓慧等人也对LRC码的构造进行了研究[5]。Hafner等对低密度奇偶校验码（Low-density Parity-check，LDPC）和WEAVER码两种非最大距离可分码（Maximum Distance Separable code，MDS）建立了可靠性模型，但是并未针对所有非MDS码进行研究，方法不具通用性[15]。

综上所述，当前对存储系统可靠性研究涉及了

基金项目：国家重点研发计划项目（2016YFB1000300）；国家自然科学基金项目（61972311）；山东省自然科学基金（ZR2019LZH007）

诸多方面，对特定的非MDS码与存储系统可靠性分析也有相关研究，然而对非MDS码的数据失效若干块后剩余块的可修复概率求解仍是尚未解决的问题，并且针对采用非MDS码的存储系统也缺乏较为通用的可靠性模型。因此，笔者对采用非MDS码的存储系统可靠性进行了研究，与现有研究基于特定校验码以抽样概率计算方法不同，从非MDS码的构造矩阵入手提出一种求解非MDS码丢失若干块后的可修复概率计算方法，并提出了一种采用面向非MDS码存储系统的通用可靠性模型，最后数值分析以LRC码为例验证了模型的正确性，并且比较了不同因素对存储系统可靠性的影响。

1. 存储系统可靠性模型

1.1 背景知识

为了描述的统一，文中所提及的分布式存储系统由多个对象存储设备（Object-Based Storage Device，OSD）对象存储设备组成，存储系统对外提供的数据读写单元以对象为粒度，客户端的对象会在存储系统内部分割为个数据块，并且按照不同的纠删码规则生成个校验块，对象的数据块和校验块被随机分布到所有的存储节点中，同一对象的数据块和校验块不会在相同存储节点存储。

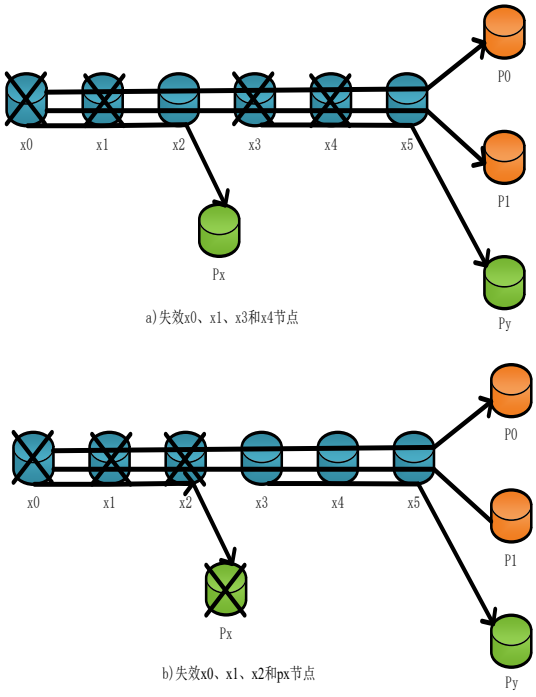


图1（6，2，2)LRC码失效修复示意图

若采用多副本或MDS码存储系统建模非MDS码存储系统可靠性，其结果是不准确的，因为多副本和纠删码中的MDS码的容错能力是确定的，而非MDS的容错能力具有一定概率，以（6，2，2）LRC码为例说明，对象X被分割为6个数据块，并生成局部效验块和全局校验块，由于LRC码是非MDS码，它是无法容忍任意4个块失效的，如图1(a)和(b)所示分别表示（6，2，2）LRC码失效四个块的两种不同情况，对于(a)来说，失效和节点，可以使用和四个校验块修复对象；对于(b)来说，失效和节点以后，校验块对于修复过程无任何意义，校验块和只能修复两个数据块，因此对象无法恢复，对于（6，2，2）LRC码中丢失任意4个块只有87%的概率可以恢复[14]。

表1 非MDS码可恢复概率求解算法伪码

算法名：非MDS码可恢复概率求解算法
输入： M：非MDS的m*n生成子矩阵 lost：丢失的块数(任意的数据块和校验块)
输出： P:可恢复概率值

过程：
recovery=0;//可恢复组合数
total=0;//所有丢失块的可能组合数
block=[i for i in range(m)];

for (i=0;i++;i<= lost) //i表示丢失的校验块数
all_combin+= combinations(m,i)]
while(i<length(all_combin)):
//遍历丢失的校验块数
parity=length(all_combin[i])
if(lost - parity = =0)://未丢失数据块
recovery+=1;
total+=1;
else:
//丢失数据块的所有组合
data += combinations(lost - parity);
for(j=0;j<length(data);j++):
matrix_A=M [all_combin[i]，block,:];
matrix_B= matrix_A[:,data[j]];
row,column=matrix_B.shape;
if(row =column):
recovery+=1 if det(matrix_B)= =1 else 0
else(row >column):
for sub_matrix in matrix_B:
if det(sub_matrix)= =1:
recovery+=1;
break;
total+=1;
p=recovery/total;
return P;//遍历求解可恢复的概率值

针对MDS码和多副本存储系统的可靠性模型大都基于马尔科夫理论，通常只考虑数据丢失的情况，不考虑其他诸如内核升级、临时性停电导致的数据临时性无法访问，模型中仅考虑硬盘的失效和修复，笔者所建立的模型也遵循此前提。

1.2 非MDS码可恢复概率求解算法

为了建立针对采用非MDS码的存储系统的较为通用的可靠性理论模型，笔者提出了一种基于矩阵

运算的求解任意非MDS码失效若干块后可恢复概率求解算法。这里非MDS码指的是不满足Singleton边界条件的编码方式的纠删码，如LRC码、LDPC码和WEAVER码等，纠删码的不同之处在于编码和译码过程，编码过程中将对象分割成多个数据块，不同的纠删码生成特定的生成矩阵，对数据块和生成矩阵进行矩阵运算生成校验块，在可容忍的范围内丢失若干块后，剩余的块和生成矩阵逆运算能恢复出丢失的块。

如表1所示为非MDS码可恢复概率求解算法伪码，已知对象的若干块丢失，设丢失块数目为，那么对象的数据块和校验块中随机选取个块共有种可能性，需要依次遍历完所有组合判断其是否可恢复，设可恢复的组合共有种，则2.2节中可恢复概率。算法判断对象可修复的过程分为二步。

1）去除生成矩阵上的单位矩阵留下行列的子矩阵，子矩阵即是生成校验块的矩阵；设丢失的个块包括了个数据块和个校验块，则首先从子矩阵行中随机去除行，表示有个校验块丢失，所有丢失块的组合有种。

2）经过步骤1)子矩阵变成行列的子矩阵，子矩阵表示目前可以用来恢复数据块的校验块组合，随后从子矩阵中抽取列组成子矩阵，抽取的可能性共种，并根据子矩阵的行列关系分别判断，若子矩阵的行大于列，即子矩阵包含了多个行列相等的行列式，如果存在某个行列式对应的子矩阵可逆，则表示此次的组合是可以恢复的；若子矩阵的行小于列时，则表示此次的组合无法恢复；若子矩阵的行等于列，则判断子矩阵是否可逆，如果可逆则此次组合是可以恢复的，否则相反。通过对所有可能性进行计算求出任意非MDS码失效若干块后可恢复的概率。

表2 模型中包含的变量和含义

变量名	含义
C	系统存储容量
n	OSD节点数目
r	单OSD节点失效速率
b	单OSD节点修复带宽
d	非MDS码的数据块数目
p	非MDS码的校验块数目
k	非MDS码临界可容错值

1.3 模型描述

文中使用MTTDL平均数据丢失时间作为非MDS码存储系统的可靠性评价指标。首先定义了如表2中的变量和其含义，特别指出的是表2中的含义是指非MDS码中失效任意块都能恢复的最大值。

非MDS码存储系统马尔科夫模型如图2所示，模型中包含了等状态值，不同的状态表示系统所处的阶段，状态为系统初始状态，状态表示系统中一个节点发生失效，其余除状态外的状态都表示节点失效但并未发生数据丢失的情况。状态为吸收态，表示系统发生数据丢失。因为存储系统存储海量数据，这里可以认为对象的数目是无穷多个，无穷多的对象随机分布到所有OSD中，则存放对象的数据块和校验块的OSD元组的所有可能为，即当失效OSD数目超过时一定会发生数据丢失，因此本模型中最终状态为。

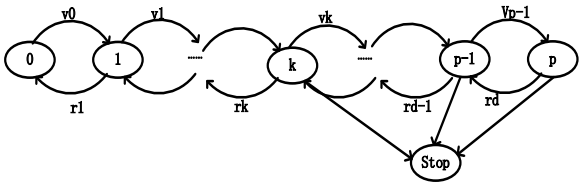


图2 非MDS码存储系统可靠性模型

首先计算非MDS码存储系统状态转移图中的各个状态间的转换速率，即状态到状态的失效速率，状态 到状态的修复速率，状态到 吸收态的速率。

状态到状态的失效速率计算需要分为二个阶段，第一个阶段是失效个OSD节点以内的情况，包括状态到状态的阶段，因为对象的数据块和校验块是随机分布到所有OSD节点的，则失效节点数目小于的情况下系统失效任何节点数目都不会造成数据丢失，状态转换到状态的失效速率为。第二阶段是失效节点数目大于的情况，在状态到状态的阶段，此阶段中失效个节点后其下一个状态可能是状态或状态。

若状态的下一状态是，不发生数据丢失情况的失效速率：

$$P_i=(\sum_{j=k}^{i}\frac{C_i^jC_{n-i}^{d+p-j}}{C_n^{d+p}}t_j)(n-i)r \quad (k\le i\le p) \qquad (1)(1)$$

的推导如下：当失效节点数目时，因为所有对象的数据块和校验块是随机完全分布的，则存放对象的OSD节点所有可能的组合有种，存在若干对象其数据块和校验块在失效OSD节点，剩余块在未失效的OSD节点。这里仅考虑对象可能发生数据丢失的情况，设在失效节点中存在对象的块在失效OSD节点中为，那么存放对象的OSD所有组合为，因此对象的个块在失效OSD节点概率为，并且当对象失效个块时，可恢复的概率用表示，文中1.2节已说明概率的求解过程，因此当失效个节点后，对象的丢失概率为，且当前系统的节点失效速率为。因此第二阶段状态转换到状态的失效速率为。

若状态的下一状态是，则发生数据丢失情况的概率:

$$(1-\sum_{j=k}^{i}\frac{C_i^j C_{n-i}^{d+p-j}}{C_n^{d+p}}t_j)(n-i)r \quad (k\le i\le p) \qquad (2)$$

对象存储系统平均每个存储节点的存储数据量为，节点失效后数据的平均修复时间为，则状态到状态的修复速率为。

通过以上分析能够得到非MDS存储系统马尔科夫模型的状态转移矩阵，状态转移矩阵的元素表达式如下:

$$q_{ij}=\begin{cases}1-nr & i=j=0\\ 1-(n-i)r-inb/C & i=j\neq 0\\ inb/C & i\neq 0\,\&\,i=j+1\\ (n-i)r & 0\le i<k\,\&\,i=j-1\\ (\sum_{j=k}^{i}\frac{C_i^j C_{n-i}^{d+p-j}}{C_n^{d+p}}t_j)(n-i)r & k\le i\le p\,\&\,i=j-1\\ 0 & \text{其他}\end{cases} \qquad (3)$$

计算系统的平均数据丢失时间MTTDL的计算矩阵[11]，记矩阵第1行的元素为，则系统的平均数据丢失时间。需要指出的是文中采用MTTDL指标只是为了量化分析采用非DMS码的存储系统的可靠性，其他指标也可以用类似的方法得出[20]。

2. 模型分析

非MDS码存储系统可靠性影响因素众多，笔者利用建立的可靠性模型量化不同因素对可靠性的影响。非MDS存储系统可靠性数值计算程序采用python语言实现，设单个硬盘容量为2T，数据修复带宽只占系统总带宽的一部分，这里假设系统分给OSD节点恢复的带宽30MB/S，单存储节点的平均失效时间为5年。 首先比较了在相同配置下常见的RS码(MDS码)和LRC码(非MDS码)对可靠性的影响。如图3所示，比较(6，4)RS码、(6，2，2)LRC码和(6，3，2)LRC码的可靠性，从图中可以看出在相同系统容量下(6，3，2)LRC码的可靠性最高，(6，4)RS码次之，(6，2，2)LRC码最低，并且在系统容量较低时采用不同配置的纠删码存储系统MTTDL较为明显，呈现数倍的差异，随着存储系统容量的增大，不同配置的纠删码存储系统可靠性都随之下降，实验结果较为符合预期期望的，因为(6，3，2)LRC码最低可容忍3个块丢失，最高可容忍5个块丢失，而(6，4)RS码仅能容忍4个块丢失，(6，2，2)LRC码最高可以容忍4个块丢失，实验仿真结果与理论分析较为一致，可以一定程度说明模型是符合真实情况的。

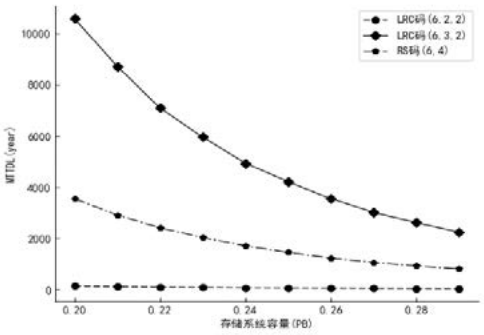


图3 纠删码配置与存储系统可靠性的关系

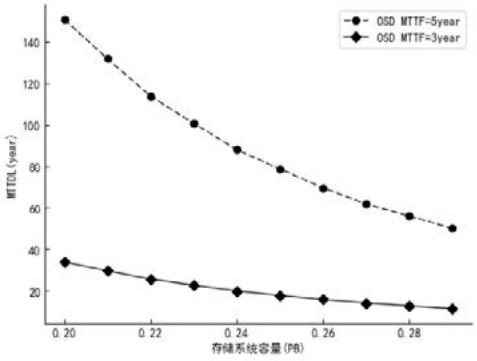


图4 硬盘寿命与存储系统可靠性的关系

随后量化OSD节点的可靠性对系统的影响，实验计算分析针对PB级存储系统，固定其他参数不变，仅改变硬盘的MTTDL比较系统MTTDL变化趋势，计算结果如图4所示，可以发现系统的平均数据丢失时间随着存储系统规模增加而不断降低，虽然采用了纠删码容错，系统可以容忍一定数量的硬盘失效而不造成数据丢失，但是在相同存储规模下，硬盘失效时间为5年时系统可靠性更大，主要是因为当OSD节点的可靠性较低时，很可能出现某些硬盘失效后系统进行修复过程中又出现其他硬盘失效，频繁失效造成的数据恢复需求超出系统的修复能力，因此系统的可靠性将会大大降低。

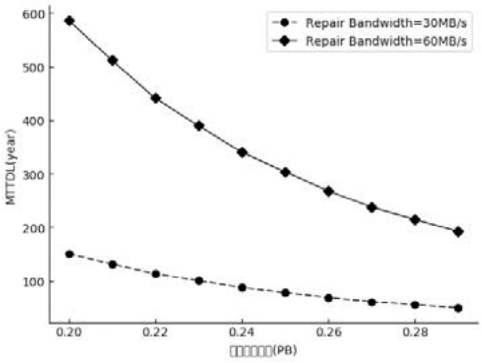


图5 带宽与存储系统可靠性的关系

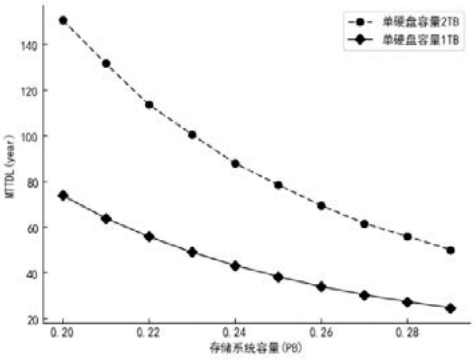


图6 硬盘容量与存储系统可靠性的关系

存储系统的网络带宽不仅仅影响系统的性能，也影响了节点失效后系统的修复过程，如图5所示，比较不同的修复带宽对可靠性的影响，从图上中可以看出修复带宽为60MB/S时系统的可靠性比修复带宽为30MB/S时更高，并且随着存储规模增大系统可靠性降低。主要是因为系统存储节点失效后其修复时间受失效OSD节点数目、OSD节点容量和修复带宽的影响，修复带宽越大，修复的时间越短，在修复时间内其他存储节点发生失效的概率越低，因此数据丢失的概率越低，存储系统更加可靠。

最后考察存储系统可靠性与存储节点容量的关系，如图6所示，在相同存储规模下，存储节点容量越大，可靠性性越高，因为系统存储节点失效后其修复时间与节点容量成正比，容量越大修复时间越长；而存储容量越大，在相同存储规模的条件下，其存储节点数目也会相应减少，因此可以得出在相同存储规模的条件下，存储节点数目对可靠性的影响比存储节点容量对可靠性的影响更大。随后在OSD节点数目相等的条件下，对具有不同存储节点容量的存储系统比较，图中存储节点容量为1T时其存储规模为0.2PB，相对于的是存储节点容量2T时存储规模为0.4T，从图中单硬盘容量2T的可靠性趋势变化中可以看出在存储节点数相等的前提下，单硬盘容量2T组成的存储系统比单硬盘容量1T组成的存储系统可靠性更低的。这也是与实际相契合的。

3. 结论

由于对大规模存储系统的可靠性实测需要消耗数年的时间，开销巨大，因此建立度量分布式存储系统可靠性理论模型显得尤为重要。面向大规模存储系统，首先提出了通用的非MDS码失效若干块后可恢复概率求解算法，随后建立了可度量非MDS码存储系统可靠性的理论模型。最后通过数值分析以LRC码为例验证了模型的正确性，比较分析了存储系统中不同纠删码配置参数、节点容量和节点失效时间、修复带宽对系统可靠性的影响，为搭建真实的存储系统提供了相应的参考，具有一定的理论价值。

参考文献：

[1] FORD D, LABELLE F, POPOVICI F I, et al. Availability in globally distributed storage systems[C]//Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation, Berkeley,CA: USENIX Association, 2010: 1–7.

[2] XU E, ZHENG M, QIN F, et al. Lessons and actions: What we learned from 10k ssd - related storage system failures[C]//2019 USENIX Annual Technical Conference. Berkeley,CA: USENIX Association, 2019: 961 - 976.

[3] RASHMI K V, SHAH N B, GU D, et al. A solution to the network challenges of data recovery in erasure - coded distributed storage systems: A study on the Facebook warehouse cluster[C]//Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems. Berkeley,CA: USENIX Association,2013：8 - 8.

[4] 王意洁 许方亮 裴晓强. 分布式存储中的纠删码容错技术研究[J]. 计算机学报, 2017, 40(1) :236 - 255.

[5] 郝晓慧,车书玲,张欣瑜.LRC码最小距离限的深入分析[J].西安电子科技大学学报,2018,45(05):75 - 79+135.

[6] CHEN Yulin, MU Shuai, LI Jinyang, et al. Giza: Erasure coding objects across global data centers[C]//Proc of the 2017 USENIX Conf on Annual Technical Conf(ATC 17).Berkeley,CA:USENIX Association,2017:539 - 551.

[7] ULUYOL M, HUANG A, GOEL A, et al. Near - Optimal Latency Versus Cost Tradeoffs in Geo - Distributed Storage[C]//17th USENIX Symposium on Networked Systems Design and Implementation. Berkeley,CA:USENIX Association,2020: 157 - 180.

[8] DOBRE D,VIOTTI P,VUKOLI M.. Hybris:Robust hybrid cloudstorage[C]//Proc of the ACM Symp on Cloud Computing. New York:ACM,2014:1 - 14.

[9] SHAHABINEJAD M,KHABBAZIAN M,ARDAKANI M. A Class of Binary Locally Repairable Codes[J].IEEE Transactions on Communications,2016,64(8):3182 - 3193.

[10] 李静,王刚,刘晓光,等.存储系统可靠性预测综述[J].计算机科学与探索,2017,11(3):341 - 354.

[11] PATTERSON D A, GIBSON G, KATZ R H. A case for redundant arrays of inexpensive disks (RAID)[C]//Proceedings of

the 1988 ACM SIGMOD international conference on Management of data. New York, NY: ACM,1988: 109 - 116.
[12] CARNS P, HARMS K, JENKINS J, et al. Impact of data placement on resilience in large - scale object storage systems[C]//2016 32nd Symposium on Mass Storage Systems and Technologies (MSST). Piscataway:IEEE, 2016: 1 - 12.
[13] 穆飞 薛巍 舒继武 郑纬民. 一种面向大规模副本存储系统的可靠性模型[J]. 计算机研究与发展, 2009, 46(5): 756–761.
[14] HUANG C, SIMITCI H, XU Y, et al. Erasure coding in windows azure storage[C]//Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12). Berkeley,CA: USENIX Association,2012: 15 - 26.
[15] HAFNER J L, RAO K K. Notes on Reliability Models for Non - MDS Erasure Codes[J]. IBM Research Report, 2006.
[16] TAMO I, BARG A, FROLOV A. Bounds on the Parameters of Locally Recoverable Codes[J]. IEEE Transactions on Information Theory, 2016, 62(6): 3070–3083.
[17] HAFNER J L. WEAVER codes: highly fault tolerant erasure codes for storage systems[C]//Proceedings of the 4th conference on USENIX Conference on File and Storage Technologies. Berkeley,CA: USENIX Association,2005, 5:16 - 16.
[18] HU Y, LIU Y, LI W, et al. Unequal Failure Protection Coding Technique for Distributed Cloud Storage Systems[J]. IEEE Transactions on Cloud Computing, 2017: 1 - 1.
[19] KADEKODI S, RASHMI K V, GANGER G R, et al. Cluster storage systems gotta have HeART: improving storage efficiency by exploiting disk - reliability heterogeneity[C]. 17th USENIX Conference on File and Storage Technologies . Berkeley,CA:USENIX Association,2019 : 345 - 358.
[20] ZHANG M, HAN S, et al. SimEDC: A Simulator for the Reliability Analysis of Erasure - Coded Data Centers[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(12): 2836 - 2848.

要闻集锦

赋能药物发现领域探索创新 亚马逊云科技发布量子计算解决方案

日前，亚马逊云科技宣布推出开源解决方案“量子计算探索之药物发现方案”，帮助客户借助量子计算技术进行药物发现研究，进一步降低量子计算的探索门槛。该解决方案提供了一键部署的量子计算/经典计算混合架构，通过Amazon Braket调用量子计算资源进行实验，并与调用经典计算资源进行对比，为量子计算在药物发现方面的应用探索新思路。

量子计算技术未来很有可能解决经典计算机无法解决的计算问题，给科学和技术带来颠覆性变革，但目前仍处于早期探索阶段。其中，药物发现是研究人员致力于评测量子计算发展阶段的领域之一。生物制药是一个高风险且耗时的流程，通常从药物发现到得到众多的候选药物再到一款药物最终被药品监督管理局批准上市，往往需要数年的时间以及数十亿美元的研发成本。

研究人员目前正在采用计算机辅助药物设计(CADD)来提升药物发现的效率。随着量子计算的发展，研究人员希望将量子计算应用到CADD的一些具体环节，并针对相同的运算问题与经典计算进行实验比较，以查看量子计算的效果并不断改进量子算法。

亚马逊云科技的“量子计算探索之药物发现方案”可以帮助研究人员设计和运行药物发现领域的计算研究。此解决方案提供了一种混合架构，可以让客户灵活地使用量子计算或经典计算资源或同时使用两者，使用这些资源对相同的问题进行测试，进而评估和比较实验价值，并进行可视化展现。该解决方案还内置了针对某些药物发现问题(例如：分子展开)的代码，供用户在相关领域快速开启量子计算之旅。亚马逊云科技计划未来增加更多的药物发现领域的代码。

通过该解决方案，用户可以通过Amazon Braket一站式的调用IonQ、Rigetti、OQC、Xanadu等量子计算机，试验超导量子、离子阱量子、光量子等不同的技术路线。Amazon Braket服务的推出降低了量子计算的使用门槛，让量子计算从只有少数研究机构可以使用的精密实验系统，发展为世界各地的研究人员、开发人员甚至是量子计算爱好者都可以通过云服务来使用，探索量子计算的无限可能。

（陈继军）

一种基于六边形循环分块的Jacobi计算优化方法

屈彬 刘松 张增源 马洁 伍卫国
西安交通大学 计算机科学与技术学院 西安 710049 wgwu@mail.xjtu.edu.cn

摘要：

Jacobi计算是一种模板计算，在科学计算领域具有广泛的应用。围绕Jacobi计算的性能优化是一个经典的课题，其中循环分块是一种较有效的优化策略。现有的循环分块策略主要关注分块对并行通信和程序局部性的影响，缺少对负载均衡和向量化等其他因素的考虑。本文面向多核架构选择一种先进的六边形分块作为加速Jacobi计算的主要方法。在分块大小选择上，综合分析分块对程序负载均衡、向量化效率和局部性等多方面的影响，一种六边形分块大小选择算法Hexagon_TSS。实验表明本文的算法可将L1数据缓存失效率降低至原始串行程序的5.46%，相对原始串行程序最大加速比达24.48，并且具有良好的可扩展性。

关键词：Jacobi计算、六边形分块、性能优化、多核架构、分块大小选择

模板计算（stencil computation）是一种常见的循环计算模式，其基本特征是遍历计算区域，每个位置均执行相同的计算操作，所有参与计算的数组元素共享同一套指令模板。模板计算在医学成像、数值方法或机器学习等领域都有广泛的应用[1]。其中，Jacobi计算是一种具有高度优化潜力和研究价值的模板计算，对Jacobi计算的优化往往能取得良好的加速效果，但至今仍没有一套公认的针对Jacobi计算的最佳优化策略。Jacobi计算在众多科学计算领域也有着广泛的应用，例如大规模线性和偏微分方程组求解、计算流体力学[2]等。

大多数科学计算及相关应用的性能热点主要位于嵌套循环，因此采用循环优化技术可以有效提高计算性能[3]。循环优化包括循环展开（loop unrolling）、循环融合（loop fusion）、循环偏斜（loop skewing）和循环分块（loop tiling）技术，它可以通过改变原始循环的执行顺序，来优化循环程序的局部性和并行性[4][5]。由于目前的计算机体系结构普遍采用分层存储架构，在运行大规模科学计算时往往存在“存储墙”（memory wall）问题[6]。在循环优化技术中，循环分块作为一种十分有效的访存优化技术，在缓解存储墙问题上能够发挥关键的作用[7]。

现有的分块方法在进行分块时往往只考虑到了分块局部性的优化，然而在实际运行过程中，分块形状和大小还会对负载均衡和向量化效率产生影响，从而影响到实际的计算速率。因此本文综合考虑了负载均衡、分块向量化效率和分块局部性的影响，提出了一种新的六边形分块大小选择方法。

本文的主要工作包括以下内容:

第一、采用一种先进的六边形分块作为Jacobi计算循环分块优化方法。为使分块内的访存地址具有强连续性，选择时间维与空间维的最外维构成的平面作为分块平面。

第二、分析了分块因子对运行时负载均衡、向量化效率和局部性的影响，为六边形分块设计实现了分块大小选择算法Hexagon_TSS。

第三、对Jacobi，Heat等经典模版计算程序进行测试，通过实验验证六边形分块策略在降低缓存失效率和提升程序计算性能方面的效果明显，并具有良好的可扩展性。

1. 相关研究

本章对本文中使用的关键技术进行介绍，首

基金项目：国防基础科研核科学挑战专题（TZZT2019-B2.2）；国家自然科学基金青年科学基金项目（6200050052）；陕西省自然科学基础研究计划一般项目(青年)（2020JQ-077）

先，介绍Jacobi计算的特征；其次，介绍循环分块技术，分别从分块形状和大小、局部性量化和多面体模型三个角度详细说明；最后，介绍本文所采用的波阵面并行方式。

1.1 Jacobi计算的研究现状

Jacobi计算来源于求解线性方程组常用的雅克比迭代法（Jacobi Iteration）的离散化形式[8][9]。除雅克比迭代法外，Jacobi计算在数值模拟领域具有广泛的应用，包括偏微分方程组求解、Lanczos图像插值和快速傅里叶变换在内的大量科学计算方法的离散化形式都可归类为Jacobi计算。

Jacobi计算的核心循环代码通常包含一层时间维循环，并在时间维循环内嵌若干层空间维循环，最内层循环执行Jacobi内核，根据空间维的层数不同可分为Jacobi-1d，Jacobi-2d，Jacobi-3d等，以Jacobi-1d计算程序为例，其核心循环代码如图1所示。程序在空间维循环中遍历数据，在遍历过程更新数组元素，完成一次遍历后进入下一个时间步继续从头开始遍历。在Jacobi内核中，任意一数组元素的更新依赖于其周围若干个数组元素的数据。

```
for (t = 0; t < tsteps; ++t)
  for (i = 1; i < size - 1; ++i)
    A[(t + 1) % 2][i] = 0.5 * (A[t % 2][i - 1] + A[t % 2][i + 1]);
```

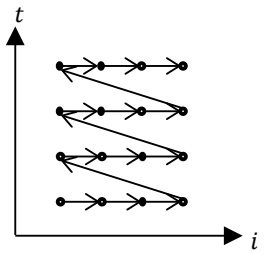
图1 Jacobi-1d程序核心循环代码

Jacobi计算主要通过奇偶复制方式实现，即分配两个大小相同的数组，每个时间步遍历一个数组，并将计算结果写入另一个数组中;进入下一个时间步后，程序遍历另一个数组，并将计算结果写入到原数组中，依此类推。奇偶复制方式使得Jacobi计算中同一个时间步内对空间维的遍历是完全可并行的，因此Jacobi计算适合并行化运算。

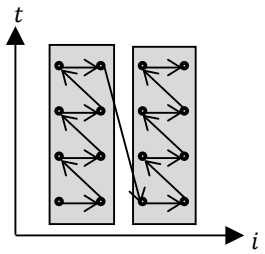
1.2 循环分块技术

循环分块是指通过增加循环嵌套的维度来提升数据局部性的循环变换技术[10]，在程序并行任务划分、通信优化、局部性优化等领域发挥着重要作用。其中，局部性优化特别是时间维局部性优化，是循环分块能够提升程序性能的主要原因。在局部性优化方面，如图2所示，循环分块能够改变程序执行顺序，进而改变数据重用距离，使程序的重用发生在更小的访存地址跨度内，以降低基于LRU替换算法的高速缓存失效率，并减少访存时间。

针对不同类型的计算模板，循环分块可以有不同的形状，包括矩形、平行四边形、梯形、菱形、六边形等。在循环分块中，矩形、平行四边形、梯形被看作是基本四边形，而六边形可看作是一个正梯形和一个倒梯形的组合，菱形则是特殊的六边形。矩形分块具有实现简单的特点，并且在所有分块形状中能够达到最高的分块内数据重用率。在条件允许的情况下，实现循环分块应尽可能地采用矩形。矩形分块中不允许存在跨空间维的数据依赖，所以矩形分块适用于Point-wise模板计算和矩阵乘类型计算，但不适用于Jacobi计算。



(a) 未分块循环执行顺序



(b) 分块后循环执行顺序

图2 循环分块对循环执行顺序的影响

针对Jacobi计算的数据流特点，Uday等人[11]提出和实现了平行四边形分块（parallelogram tiling）方法，通过倾斜分块表面使分块内的数据依赖得到保持。平行四边形分块之间存在空间维超平面内的数据依赖，因此分块间不能完全并行执行。采用平行四边形分块的循环程序可通过波阵面方式实现并行，其执行分为起始、满载、排空三个阶段，程序只有在满载阶段才达到并行度的最大化，在起始和排空阶段不能完全并行。

为提高Jacobi循环分块程序的并行度，Grosser等人[12]在平行四边形分块的基础上提出了梯形分块（trapezoidal tiling）方法。梯形分块是一种分裂分块（split tiling）方法，将平行四边形分裂为一个正梯形和一个倒梯形，并改变分块间数据依赖的方向。分裂分块中同一层的正梯形间和倒梯形间可完全并行，正梯形与倒梯形交替并行执行。梯形分块也可通过波阵面实现并行，在并行执行时全程满载。梯形分裂分块在并行度上优于平行四边形分块，并且比平行四边形能够达到更高的分块内数据重用率。

Grosser等人在梯形分块基础上进行改进，将正梯形与倒梯形上下拼接在一起，实现了六边形分块（hexagonal tiling）[13][14]。与梯形分块一样，六边形

分块也可完全并行，但六边形分块的块内数据重用率高于梯形分块。在六边形分块的基础上，Uday等人[15]通过最大化分块内的时间维长度得到菱形分块（diamond tiling）。菱形分块是六边形分块的特殊形式，目前适合Jacobi计算的分块形状中，菱形分块可达到最高的分块内数据重用率。但菱形分块上下顶端短边边长较短在数据规模相同的情况下控制开销大于一般的六边形分块，且不利于分块内的向量计算，而梯形分块和六边形分块则能够达到更好的向量化效率。

分块大小对循环分块的效果有重要的影响，因此大部分针对循环分块的研究围绕分块大小选择（Tile Size Selection，TSS）展开。目前分块大小选择方法主要有静态方法和经验搜索方法。静态方法上，斯坦福大学的Monica Lam等人[16]最早提出了描述程序数据重用的局部性的静态数学模型，在此基础上首先实现了面向矩形分块的定量化分块大小选择算法。刘松等人[17]提出了一种基于高速缓存均匀映射的分块因子选择算法UMC-TSS，充分利用局部性分析的收益模型，选择高收益的循环代码实施分块优化。在经验搜索方面，ATLAS系统[18]可通过遍历不同问题规模的线性代数求解程序，从中选择性能较优的分块大小。经验搜索方法通常与机器学习结合[19][20]，可获得局部最优性能的分块大小，但遍历测试用例的时间开销较大，因此通常先结合静态分析模型裁剪搜索空间，以减少遍历次数。

1.3 局部性量化

局部性原理作为重要的的计算机科学理论，一直收到广泛的研究，许多研究工作专注于局部性分析与量化的理论研究[21][22]。狄鹏等人[23]针对Jacobi计算提出了一个能够较客观地反映局部性的指标，时间维数据重用率（Temporal Data Reuse Rate，TDRR）。块内的时间维数据重用率可定义为分块中所有数据的时间维重用次数除以分块的访存地址跨度，如公式(1)所示。

$$TDRR = \frac{reuse}{range} \tag{1}$$

公式(1)中，*reuse*表示分块的时间维数据重用次数，仅考虑时间开销较大的写重用，*range*表示分块的访存地址跨度，分块局部性与TDRR值正相关。对于Jacobi计算，分块的数据重用次数等于分块的迭代实例个数减去分块的访存地址跨度。而分块的访存地址跨度定义为分块内最大访存地址与最小访存地址之差，因为访存地址跨度考虑的是静态存储区分配内存空间，数据行在内存中是是连续的，其值也等于分块投影在空间维上超平面的面积。分块迭代实例个数与访存地址跨度均与具体的分块形状有关。该公式的定义仅面向分块方法，而分块方法是为并行服务的，为了保证各线程计算结果正确性，每个分块的块间依赖必须在执行前得到满足，因此该公式中未考虑块间依赖。

从定性分析的角度上看，分块访存地址跨度衡量了分块内的数据在内存中的分布范围。若分块数据的分布范围越小，则访存地址跨度越小，那么访问该分块内的数据时更容易命中缓存。而数据重用次数则表示程序访问相同数据的次数。结合起来看，TDRR表示在一定的缓存命中率下访问内存的次数，当缓存命中率较高时，访问相同数据的次数越多，表明分块局部性越好。

1.4 波阵面并行

波阵面并行（wave-front parallelism）是一种面向DOACROSS循环的并行控制方式[24][25]，适用于以Jacobi计算为代表的具有同步操作的模板计算程序。波阵面将循环迭代空间划分为多个波前线(wave-front line)，每个波前线内放置若干个分块。同一个波前线内的分块可以同时并行执行，程序按顺序执行每个波前线，在推进波前线时进行同步。若循环迭代空间存在空间维超平面内的数据依赖，则依赖目标迭代实例必须滞后于依赖源迭代实例执行，对应的波阵面也必须倾斜。倾斜波阵面的执行分为起始、满载和排空阶段，在起始和排空阶段不能完全并行，相对于普通波阵面并行度较低。倾斜波阵面通常应用于Sweep模板计算和平行四边形分块的并行。

在任务调度方面，波阵面既可采用静态调度方法，也可以采用动态调度方法。静态调度方法在程序执行前预先为每个计算核分配负载范围，可实现分块与计算核的绑定，有利于提高分块间的局部性。动态调度方法[26][27][28]在运行时根据程序产生的实际数值调节负载分配，有利于维持运行时负载均衡。静态调度适用于包括模板计算在内的访存地址序列可预测、随机性弱的任务类型，而动态调度适用于访存地址序列不可预测、随机性强的任务类型，例如实时任务调度。Jacobi计算作为一类访存地址序列可预测的计算任务，适合采用静态调度方法。

2. 六边形分块策略

本节首先分析分块平面对分块访存地址连续性的影响，探讨适合Jacobi六边形分块的分块平面。接着从负载均衡、向量化效率和分块局部性收益三个角度讨论分块大小对分块性能的影响。最后，推导面向六边形分块的分块大小选择算法Hexagon_TSS。

六边形可视作一个正梯形与一个倒梯形的组

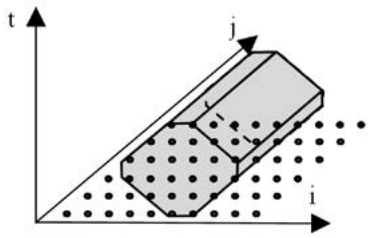
合，而梯形分块较为简单，因此下面每小节的分析都先从梯形分块入手，再将梯形分块的结论扩展到六边形分块中。

2.1 分块平面选择

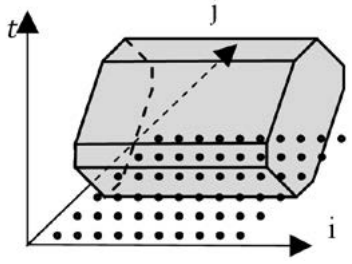
分块平面指循环分块在空间中投影出分块形状的平面，例如2维Jacobi计算的六边形分块在分块平面上投影出六边形，在其他平面上投影出矩形；2维Jacobi计算的平行四边形分块在分块平面上投影出平行四边形，在其他平面上投影出矩形。

在进行循环分块前首先为2维及2维以上的Jacobi计算选择合适的分块平面，多边形分块可以选择时间维和空间上任意一维构成分块平面。需要注意的是，越往内的空间维循环，迭代实例的访存连续性越强。特别地，最内维循环的迭代实例具有连续的访存地址。选择空间上的内维与时间维构成的平面作为分块平面将破坏该空间维的访存连续性，扩大分块的访存地址跨度，不利于局部性。

以图3中的Jacobi-2d六边形分块为例，t轴是时间维，i轴、j轴分别是空间的第1维和第2维。其中j轴是最内维，j轴上任意相邻两点的访存地址连续，图中圆点表示迭代实例。图3(a)中的分块以t轴和i轴构成的平面作为分块平面，那么分块内同一个i-j平面上的访存地址都是连续的。而图3(b)中的分块以t轴和j轴构成的平面作为分块平面，此时分块内只有i方向上迭代实例的访存地址才是连续的，而i方向上迭代实例的访存地址不连续。由此可见，将空间上的最外维与时间维构成的平面作为分块平面，在访存地址连续性上优于时间维与空间内维构成的分块平面，且前者的访存地址跨度更小。



(a) t-i分块平面



(b) t-j分块平面

图3 Jacobi-2d中两种六边形分块平面

为了最小化分块的访存地址跨度，本文为六边形分块选择空间维的最外维与时间维构成的平面作为分块平面。

2.2 六边形分块大小定义

记TS_i表示分块第i维的长度，N_i表示Jacobi计算第i维的迭代范围。由于选择时间维和空间第1维构成的平面作为分块平面，分块第1维的长度TS_1和第2维的长度TS_2是可调节的，TS_1和TS_2决定了分块大小;而分块第i维（i≥3）的长度固定等于全局循环迭代空间在第i维上的迭代范围。其中TS_1和TS_2分别是分块跨过的时间步范围和分块在空间第1维上投影的长度，它们与分块几何形状之间的对应关系如图4所示。

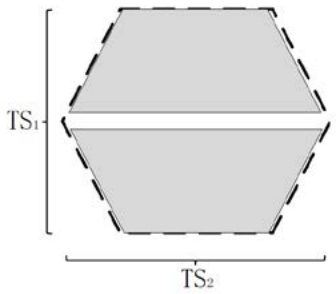


图4 六边形分块大小与几何形状之间的对应关系

由于分块第i维（i≥3）的长度已固定，在分块大小选择上主要考虑分块TS_1和TS_2的选择。下面将分别讨论分块大小TS_1和TS_2与分块形状、负载均衡、向量化效率以及局部性收益之间的关系。

2.3 分块大小选择

2.3.1 分块大小与分块形状

在梯形分块中，分块形状有上底边和下底边，其中较长的底边为长底边，较短的底边为短底边。长底边的长度等于tra_TS_2，短底边的长度与tra_TS_1、tra_TS_2和Jacobi的模板因子有关。在梯形分块中，tra_TS_1时间步后tra_TS_2缩短为$tra_TS'_2$，而在大多数Jacobi计算中，模板因子为1，因此每个时间步缩短的长度为2，故短底边长度$tra_TS'_2$可通过公式(2)计算。

$$tra\_TS'_2 = tra\_TS_2 - 2(tra\_TS_1 - 1) \quad (2)$$

为了维持梯形的形状，梯形分块的短底边长度必须大于0，同时梯形的时间维长度必然大于1，因此梯形分块第1维的长度tra_TS_1和第2维的长度tra_TS_2须满足公式(3)所示的约束条件。

$$\begin{cases} tra\_TS_2 \geq 2 * tra\_TS_1 - 1 \\ tra\_TS_1 \geq 2 \end{cases} \quad (3)$$

同样的原理运用到六边形分块上，注意到当时$TS_2=tra_TS_2$，六边形分块TS_1的是梯形分块tra_TS_1的两倍，所以六边形分块的TS_1必然是偶数。对于六边

形分块，为维持六边形的形状，分块第1维的长度TS_1和第2维的长度TS_2须满足公式(4)所示的约束条件。

$$\begin{cases} TS_2 \geq TS_1 - 1 \\ TS_1 \geq 4 \\ mod(TS_1, 2) = 0 \end{cases} \tag{4}$$

2.3.2 特别地，对于六边形分块，当TS_2=TS_1-1时，分块的形状为菱形。分块大小与负载均衡

在采用静态任务调度的波阵面并行方式下，设波阵面中每个分块都采用相同规格的分块大小，为平衡线程的计算负载，分块的分配应遵循平均原则。在就绪波前线中分块数目等于线程数的情况下，每个线程处理的分块数相同。对于问题规模较小的Jacobi计算，若分块大小过大，那么就绪波前线中分块数目的数量就可能小于线程的个数，将导致部分线程无负载，出现负载不均衡的情况。对于一般问题规模的Jacobi计算，若就绪波前线中分块数目不等于线程数的整数倍，那么在运行时将出现负载不均衡的现象，部分线程在每次推进波前线时总是比其他线程多执行一个分块，造成其他线程的等待。下面将分析分块大小TS_1和TS_2与负载均衡之间的定量关系。

在采用梯形分块的Jacobi计算中，任意时刻分块都是由一个正梯形和一个反梯形交错的形式排布的，且其执行过程也是交错执行的，如图5所示。因此就绪波前线中分块数目为单个时间分块内总分块数的一半，可通过公式(5)计算。

$$num = \left\lceil \frac{N_2}{2 * tra\_TS_1(tra\_TS_2 + tra\_TS_2')/2} \right\rceil = \left\lceil \frac{N_2}{2 * (tra\_TS_2 - tra\_TS_1 + 1)} \right\rceil \tag{5}$$

公式(5)中，N_2表示Jacobi计算第2维的迭代范围。可通过调节tra_TS_1和tra_TS_2使就绪波前线中分块数目等于线程数*threads*的整数倍，此时*num*除以*threads*的余数*remain*应等于0，*remain*通过公式(6)计算。

$$remain = mod(num, threads) \tag{6}$$

若无论怎样调节tra_TS_1和tra_TS_2都不能使*remain*的值等于0，此时应最小化等待线程的个数，即调节tra_TS_1和tra_TS_2使*remain*的值最大化。

对于采用六边形分块的Jacobi计算，在TS_2=tra_TS_2的情况下，六边形分块的TS_1是梯形分块tra_TS_1的两倍，那么任意时刻就绪波前线中六边形分块数目*num*通过公式(7)计算。

$$num = \left\lceil \frac{N_2}{2 * \left(TS_2 - \frac{TS_1}{2} + 1\right)} \right\rceil = \left\lceil \frac{N_2}{2(TS_2 + 1) - TS_1} \right\rceil \tag{7}$$

与梯形分块同理，就绪波前线中六边形分块数目除以线程数得到的余数*remain*也可通过公式(6)计算。

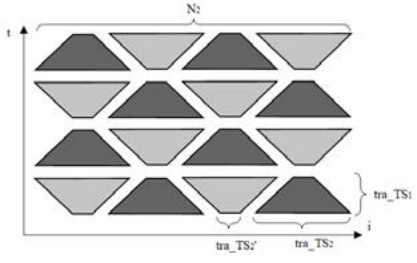


图5 梯形分块调度过程

2.3.3 分块大小与向量化效率

对于具有SIMD指令集扩展的CPU，可通过单条SIMD指令可以完成多组数据的运算，前提是参与运算的数据宽度与向量寄存器宽度W要对齐，即一个向量寄存器中可存放的浮点运算操作数个数对齐。若不对齐，则处理器只能通过标量运算指令完成不对齐部分的运算。对于循环分块，若分块边长与向量寄存器宽度对齐良好，那么大部分的运算可通过SIMD指令执行，完成块内数据的运算所需的指令个数更少，进而减少处理器运算该分块所需的时间。目前主流的编译器都支持自动向量化，在编译过程中将满足条件的标量计算指令替换成SIMD指令。因编译器具有良好的自动向量化功能，在编写高性能程序时，大多数情况下程序员不需要手动输入和编排SIMD指令，只需把主要的精力放在数据对齐上。

记分块中的平均每个迭代实例所需的指令条数（Instructions Per Iteration）为IPI，IPI与分块最内维的长度有关。IPI能够客观地反映分块的向量化效率，它的值越小，说明分块的向量化效率越好，程序所含的指令条数越少。对于n维（n≥3）梯形分块，它的IPI等于各个时间步对应的空间维超平面运算所需指令条数的叠加，再除以分块的迭代实例个数，其简化后的计算公式如公式(8)所示。

$$IPI = \frac{\sum_{t=0}^{TS_1-1}\left[\left\lfloor \frac{L_n}{W} \right\rfloor + mod(L_n, W)\right]}{S} \tag{8}$$

公式(8)W中是向量寄存器宽度，L_n是分块最内维的实际长度，随时间步变化，在n≥3的情况下固定等于最内层循环的迭代范围N_n，不可调节，S为分块的迭代实例个数，可通过计算分块对应的几何形状面积得到。特别地，当n=2时，即面向Jacobi-1d程序时，tra_TS_2作为分块最内维的边长，梯形分块IPI的计算方式比较特殊，通过公式(9)计算，其中K=mod（tra_TS_2-2t,W），$tra_TS'_2$可由公式(2)计算。

$$IPI = \frac{\sum_{t=0}^{TS_1-1}\left[\left\lfloor\frac{tra\_TS_2-2t}{W}\right\rfloor+K\right]}{tra\_TS_1(tra\_TS_2+tra\_TS_2')/2} = \frac{\sum_{t=0}^{TS_1-1}\left[\left\lfloor\frac{tra\_TS_2-2t}{W}\right\rfloor+K\right]}{tra\_TS_1(tra\_TS_2-1)+1} \quad (9)$$

六边形分块可看作两个梯形的组合，在循环维度n≥3时，六边形分块的IPI与具有相同TS_2的梯形分块的IPI相等。但在循环维度n=2时，六边形分块IPI的计算方式与梯形分块略有差别，此时六边形分块的IPI计算为公式(10)，其中$K=mod(TS_2-2t,W)$。

$$IPI = \frac{\sum_{t=0}^{TS_1-1}\left[\left\lfloor\frac{TS_2-2t}{W}\right\rfloor+K\right]}{\left[\frac{TS_1}{2}*(TS_2-1)+1\right]*2} \frac{2\sum_{t=0}^{\frac{1}{2}TS_1-1}\left[\left\lfloor\frac{TS_2-2t}{W}\right\rfloor+K\right]}{TS_1(TS_2-1)+2} \quad (10)$$

向量寄存器宽度W为取与处理器的SIMD指令集类型有关，表1给出了部分常用SIMD指令集对应的向量寄存器宽度。

表1 常用SIMD指令集的数据宽度

SIMD指令集类型	向量寄存器宽度		
	位宽	单精度操作数宽度	双精度操作数宽度
Intel SSE	128	4	2
Intel AVX	256	8	4
Intel AVX512	512	16	8
ARM NEON	128	4	2

由于只有在循环维度n=2时，可调节分块大小TS_1和TS_2才能影响分块的IPI，因此分块大小对分块向量化效率的影响仅存在于1维Jacobi计算中。在针对Jacobi-1d程序选择分块大小时，可将最小化IPI作为一个优化目标，通过调节TS_1和TS_2，改善数据对齐状况，减少分块所需的指令条数。

2.3.4 分块大小与局部性收益

由于处理器访问主存所需的时间远大于访存缓存的时间，因此完全命中缓存相比于部分不命中的情况，访存速度有显著的提升。在Jacobi循环分块程序中，通过调节空间维分块大小可改变分块的访存地址跨度。根据访存地址跨度的定义，Jacobi计算中梯形分块和六边形分块的访存地址跨度*range*与空间维分块大小TS_2之间的定量关系可由公式(11)描述。

$$range = 2*TS_2\prod_{i=3}^{n}TS_i \quad (11)$$

在选择分块大小时，应尽量使分块内的访存操作完全命中某一级缓存，分块的访存地址跨度应小于目标缓存的容量。据此，可为分块大小TS_2创造一个与目标缓存容量相关的约束条件，进一步裁剪分块大小选择的搜索空间，该约束条件如公式(12)所示。

$$\left\lfloor\frac{range}{ccline}\right\rfloor \times ccline \times typesize \le cpct \quad (12)$$

公式(12)中，*cpct*表示目标缓存容量，*typesize*表示目标缓存行大小，表示数据类型大小。由于缓存是以缓存行为单位进行数据置换的，所以在计算分块在缓存中的实际访存地址跨度时，须将分块的访存地址跨度换算成分块占用的缓存行个数，并乘上缓存行大小。在实际应用中，一个分块通常不能独占目标缓存，因此TS_2的选取应偏小一些。

前文提到，分块的时间维数据重用率TDRR能够客观地反映分块的局部性，它等于分块中所有数据的时间维重用次数*reuse*除以分块的访存地址跨度。按照时间维重用次数的定义，梯形分块的时间维重用次数等于分块的迭代实例个数减去分块在空间维超平面上投影的面积，其化简后的计算方法如公式(13)所示。

对于六边形分块，在TS_2相等的情况下TS_1是梯形分块的两倍，同时六边形分块的迭代实例个数等于两个梯形分块的迭代实例个数之和。在考虑分块的时间维重用次数时，还要再加上两个梯形之间的一层时间维长度为1的矩形分块的迭代实例个数。六边形分块的时间维重用次数计算格式与梯形分块相似，如公式(14)所示。

$$\begin{aligned} reuse &= tra\_TS_1(tra\_TS_2+tra\_TS_2')/2\prod_{i=3}^{n}tra\_TS_i-(tra\_TS_2-1)\prod_{i=3}^{n}tra\_TS_i \\ &= (tra\_TS_1-1)(tra\_TS_2-1)\prod_{i=3}^{n}tra\_TS_i \end{aligned} \quad (13)$$

$$reuse = [(TS_1-1)\left(\frac{TS_2}{2}-1\right)*2+TS_2]\prod_{i=3}^{n}tra\_TS_i = [TS_1(TS_2-1)+2]\prod_{i=3}^{n}TS_i \quad (14)$$

得到六边形分块的访存地址跨度和时间维数据重用次数后，联立公式(11)和公式(14)，六边形分块的TDRR通过公式(15)计算。

在规划分块大小时，应通过调节TS_1，TS_2以最大化分块的时间维数据重用率TDRR。

$$TDRR = \frac{reuse}{range} = \frac{[TS_1(TS_2-1)+2]\prod_{i=3}^{n}TS_i}{2TS_2\prod_{i=3}^{n}TS_i} \quad (15)$$

2.3.5 Hexagon_TSS算法

下面将面向六边形分块推导分块大小选择算法，将该算法记作Hexagon_TSS（Hexagon Tile Size Selection）算法。Hexagon_TSS算法的核心是多目标数学规划，利用约束条件构造一个搜索空间，在搜索空间里根据规划目标搜索最优解。其中约束条件包括分块大小满足维持六边形的形状、分块的访存地址跨度小于目标缓存容量。优化目标则包括最佳多线程运行时负载均衡、最小化分块内平均执行每个迭代实例的指令个数IPI、最大化分块的时间维数据重用率TDRR。而Hexagon_TSS算法只有两个可调节的自变量，分别为时间维分块长度TS_1和空间第1维分块长度TS_2。联立公式(4)、公式(6)、公式(10)、公式(11)、公式(12)和公式(15)，推导出关于六边形分块可调分块大小TS_1和TS_2的数学规划表达式的问题约束和优化目标。其中，问题约束如公式(16)所示。

$$\begin{cases} TS_1 \geq 4 \\ TS_1 \leq N_1 \\ mod(TS_1, 2) = 0 \\ TS_2 \geq TS_1 - 1 \\ remain = mod(num, threads) \\ TS_2 \leq min\left(N_2, \frac{cpct}{2typesize\prod_{i=3}^{n} TS_i}\right) \\ \min\left(\frac{2\sum_{t=0}^{\frac{1}{2}TS_1-1}\left(\left\lfloor\frac{TS_2-2t}{W}\right\rfloor+K\right)}{TS_1(TS_2-1)+2}\right) \\ \max\left(\frac{[TS_1(TS_2-1)+2]\prod_{i=3}^{n} TS_i}{2TS_2\prod_{i=3}^{n} TS_i}\right) \end{cases} \tag{16}$$

而Hexagon_TSS算法数学规划优化目标如表2所示。表2中，优化目标1和优化目标2存在互斥性，在进行数学规划时，若能够满足优化目标1，则不需要再考虑优化目标2；反之，若不能满足优化目标1，再考虑优化目标2。下面将讨论分别以4个优化目标为主要优化目标进行数学规划时，自变量的取值。

表2 Hexagon_TSS算法优化目标

优化目标编号	目标函数	最优解
1	公式(6)	0
2		最大值
3	公式(10)	最小值
4	公式(15)	最大值

首先是以最佳多线程运行时负载均衡为优化目标的数学规划。最理想的情况是就绪分块的数量刚好等于并行线程数的整数倍，即就绪分块的数量除以线程数的余数*remain*等于0。然而余数的计算涉及到与TS_1、TS_2相关的取模运算，而取模运算的结果具有较强的随机性，难以构造针对取模运算的精确优化模型。因此，在以最佳多线程运行时负载均衡为优化目前时，应通过遍历的方法记录使得*remain*等于0的TS_1和TS_2；若搜索空间内不存在TS_1和TS_2使得*remain*等于0，则取使*remain*达到最大值的TS_1和TS_2作为该优化目标下的分块大小。

其次是以分块内平均执行每个迭代实例的指令个数IPI最小化为优化目标的数学规划。观察公式(10)，由于该公式中存在与TS_1和TS_2相关的取模运算，IPI与TS_1和TS_2之间不存在明显的单调关系，也不能求IPI对TS_1和TS_2的偏导数。取模运算的结果具有较强随机性，难以推导IPI与TS_1和TS_2之间精确的换算关系。应该对搜索空间中的TS_1和TS_2进行遍历，记录使IPI最小的TS_1和TS_2作为该优化目标下的最佳分块大小。

最后是以分块的时间维数据重用率TDRR最大化为优化目标的数学规划。给定六边形分块P_h，观察公式(15)，TS_1和TS_2必然满足$TS_1 \geq 1$和$TS_2 \geq 1$，因此通过分别求TDRR对TS_1和TS_2的偏导数，可发现TDRR随TS_1或TS_2单调递增，提高分块的时间维重用次数的途径是增大TS_1或TS_2。但分块的访存地址跨度同样随着TS_2的增大而扩大，为了最小化分块的访存地址跨度，在最大化TDRR时应优先增大TS_1，再增大TS_2。

Hexagon_TSS算法的实现如算法1所示。首先根据循环的迭代范围设置第3维以及更内维的分块大小，根据公式(16)所示的约束条件构造一个搜索空间，并构造一个候选解的集合candidates，如算法1中第1~2行。然后在搜索空间中搜索满足优化目标1的解;若满足，则将解加入到candidates中，跳过优化目标2，如算法1中第3~9行;若优化目标1没有得到满足，则基于优化目标2遍历搜索空间，记录*remain*的最大值并将对应的解加入到candidates中，如算法1中第10~23行。接着若循环维度n=2，则基于优化目标3，遍历候选解集合candidates，记录IPI的最小值，并删除candidates中不满足IPI达到最小值的解；若循环维度n≥3，则跳过优化目标3，如算法1中第24~36行。最后基于优化目标4，在候选解集合candidates中，找出能够使TDRR达到最大值的解，并在其中找到TS_2最小的解作为最终解，如算法1中第37~48行。

算法1：六边形分块大小选择算法Hexagon_TSS

```
输入:循环维度、循环迭代范围、并行线程数、向量寄存器宽度、目标缓存容量、数据类型大小
输出:六边形分块大小
// remain(a，b)表示计算以a，b为分块大小时的就绪波前线中分块数目，并对取余
// 表示将和的配对装入优先级队列，并对队列中的元素根据 按升序排序，若相等则根据按降序排序
// 表示编程语言支持的最大整数
1初始化，构造候选解集合，优先级队列;
2根据公式(16)初始化;
3  FOR  BY 2 DO
4     FOR  BY 1 DO
5        IF() THEN
6           ;
7        END IF
```

```
8     END FOR
9  END FOR
10 IF() THEN
11    令;
12    FOR  BY 2 DO
13       FOR  BY 1 DO
14          IF() THEN
15             ;
16             ;
17
18          ELSE IF() THEN
19             ;
20          END IF
21       END FOR
22    END FOR
23 END IF
24 IF() THEN
25    令;
26    FOR  IN  DO
27       IF() THEN
28          ;
29       END IF
30    END FOR
31    FOR  IN  DO
32       IF() THEN
33          ;
34       END IF
35    END FOR
36 END IF
37令;
38 FOR  IN  DO
39    IF() THEN
40       ;
41       ;
42       ;
43    ELSE IF() THEN
44       ;
45    END IF
46 END FOR
47 ;
48 RETURN ();
```

Hexagon_TSS算法中包含两层嵌套循环，第1层循环的迭代范围不超过N_1，第2层循环的迭代范围不超过N_2，循环内部运算的时间复杂度为常数级，那么Hexagon_TSS算法的时间复杂度为$O(N_1N_2)$；Hexagon_TSS算法中最多有$0.5N_1N_2$个候选解，则候选解集合与优先级队列的最大空间开销为$0.5N_1N_2$，算法的空间复杂度为$O(N_1N_2)$。此外，在选择目标缓存时，应优先选择距离计算核最近，访问速度最快的L1数据缓存。若L1数据缓存的容量不足以容纳最小的分块，则选择L2缓存。若L2缓存的容量仍不能容纳最小的分块，那么不考虑访存地址跨度小于缓存容量的约束条件，在Hexagon_TSS算法得到的候选解中选择最小的分块大小作为最终解。

3. 实验分析

本文的实验将围绕两个目标展开：1.验证Hexagon_TSS分块大小选择算法的有效性；2.对比结合了Hexagon_TSS分块大小选择算法的六边形分块与其他优化方法的性能表现。在开展实验前，首先说明实验的设置和实验环境。

3.1 实验设置与实验环境

所有实验都在同一台服务器上进行，实验涉及的具体软硬件信息如表3所示。

表3 实验环境信息

指标	实验环境
CPU	2×Intel Xeon Gold 6248 2.5GHz
微架构	Cascade Lake
核心数	2×20
SIMD指令集	Intel AVX512
缓存容量	32KB | 1024KB | 28160KB
缓存行大小	64字节
主存	6×32GB DDR4
操作系统	CentOS 7
编译器	GCC/G++ 10.2.0
编译选项	-O3 -mavx512f -fopenmp -ftree-vectorize -Wall -std=c++11
PLuTo版本	0.11.4
PLuTo flags	./polycc [source code] –tile --parallel
perf版本	3.10.0

本文选择jacobi-1d、heat-2d、heat-3d和seidel-2d作为测试程序[29]，它们分别属于常见的1维、2维、3维和2维的Jacobi计算。测试程序的计算访存比对循环分块的优化效果具有较大的影响，表4给出了四种测试程序的计算访存比，计算访存比为单次迭代的计算次数与读访存次数的比值，可以反映不同模板计算的差异。

表4 测试程序信息

程序名称	Jacobi模板类型	计算访存比
jacobi-1d	1d3pt	0.33
heat-2d	2d5pt	1.20
heat-3d	3d7pt	1.29
seidel-2d	2d9pt	1.00

实验对每种测试程序都实现了四种优化版本，分别为：baseline、pgram、diamond和hexagon，其中hexagon版本用于验证本文的实验目标，另外三个版本作为对照组与hexagon版本进行对比。baseline版本不进行任何循环分块，但基于OpenMP实现了多线程并行。pgram是基于pluto算法[30]实现的平行四边形循环分块，而diamond是基于pluto+算法实现的菱形分块，两种版本的代码都使用PLuTo编译器生成。pgram和diamond分块代码与本文hexagon分块代码一样使用空间循环最外维和时间维构成分块平面。

四种测试程序具有不同的Jacobi模板和循环维度，因此具有不同的表达问题规模的格式。为便

于问题规模的表达，将问题规模按从小到大的顺序划分为四个等级。表5列出了四种问题规模在不同Jacobi模板中对应的数值大小。

表5 问题规模分级

问题规模	问题规模数值		
	1d	2d	3d
sizeA	40k	200×200	40×40×40
sizeB	400k	600×600	80×80×80
sizeC	4,000k	2k×2k	160×160×160
sizeD	40,000k	6k×6k	400×400×400

不同的实验可能在部分实验参数上的数值相同，例如相同的问题规模、相同的并行线程数等。为避免赘述实验参数，称在某一实验参数在大部分测试中相同的值为该实验参数的默认值，表6给出了各项实验参数的默认值。在下文中，若无特殊说明，则每个实验参数的取值都为默认值。此外在默认情况下，hexagon版本测试程序通过Hexagon_TSS算法计算得到，pgram和diamond版本测试程序通过PLuTo得到。

表6 默认实验参数值

实验参数	默认值
数据精度	double
并行线程数	20
问题规模	sizeC
时间维长度	300

实验使用perf采集程序访问缓存的次数以及缓存失效次数等缓存相关信息。perf是一款基于Linux平台的开源profiling工具，它能够通过CPU事件记录各级缓存的读次数和读失效次数，将读失效次数除以读次数可得到读失效率.虽然perf无法采集程序的缓存写失效次数，但Jacobi计算的访存操作以读为主，因此测试程序的读失效率接近于实际读写失效率。

3.2 Hexagon_TSS算法有效性验证

本节将进行两方面的实验:1.记录和对比baseline、pgram、diamond、hexagon版本下四种测试程序的各级缓存失效率，验证六边形分块结合Hexagon_TSS算法对Jacobi计算缓存失效率的改善作用；2.针对hexagon版本的测试程序，对比通过Hexagon_TSS算法得到的分块大小与实际最佳分块大小对应的程序性能，计算前者与后者的比值。

3.2.1 缓存失效率验证

表7列出了四种测试程序的缓存失效率。注意由于L3共享缓存平均到每个计算核上的容量小于计算核私有的L2缓存，并且存在争用现象，所以L3缓存失效率普遍大于L2缓存失效率。

表7 不同优化版本测试程序缓存失效率对比

测试程序	优化版本	缓存读失效率		
		L1 cache	L2 cache	L3 cache
jacobi-1d	baseline	62.51%	13.74%	27.01%
	pgram	13.39%	31.75%	49.34%
	diamond	1.75%	25.13%	73.93%
	hexagon	3.41%	14.02%	69.84%
heat-2d	baseline	48.55%	1.46%	44.09%
	pgram	47.56%	1.52%	28.71%
	diamond	42.67%	0.71%	44.71%
	hexagon	41.40%	0.53%	37.74%
heat-3d	baseline	39.30%	9.00%	34.73%
	pgram	33.42%	13.25%	34.84%
	diamond	30.82%	11.63%	47.51%
	hexagon	31.18%	9.71%	32.53%
seidel-2d	baseline	38.84%	1.21%	43.73%
	pgram	38.18%	1.21%	29.14%
	diamond	34.35%	0.67%	39.24%
	hexagon	33.59%	0.46%	41.60%

表7中，pgram、diamond和hexagon版本的L1数据缓存失效率都低于baseline版本，在jacobi-1d测试程序中尤为明显，而hexagon版本在jacobi-1d中L1数据缓存的失效率仅为baseline的5.46%。这是因为1维Jacobi可以实现访存地址跨度较小的分块，使得针对1维Jacobi计算的循环分块优化能够显著地降低缓存失效率。整体上diamond版本的程序达到了最低的L1缓存失效率，这是因为菱形分块是最具局部性收益的分块形状，能够最大化分块内的时间维重用次数。hexagon版本在L1缓存失效率上相比于diamond版本略有不足，但在L2缓存失效率上表现相对较好。实验表明，六边形分块结合Hexagon_TSS分块大小选择算法可显著降低缓存失效率，其效率可与目前较先进的diamond的分块方法相当，甚至略优于diamond的分块方法。

3.2.2 Hexagon_TSS效率验证

Hexagon_TSS算法的效率指以使用Hexagon_TSS算法得到的分块大小对应的计算速率，除以遍历搜索空间得到最佳分块大小Best_Hexagon对应的计算速率。其中，Best_Hexagon是使程序计算速率最优的分块大小，其值通过线性规划确定搜索空间，然后遍历空间内的所有解进行测试获得。由于遍历并获得最佳分块大小的过程需要消耗大量的时间，本实验仅以计算访存比适中的seidel-2d作为测试程序。表8列出了使用Hexagon_TSS为hexagon版本的seidel-2d测试程序计算分块大小时，在四种问题规模下的效率。

表8 Hexagon_TSS算法效率

问题规模	Hexagon_TSS分块大小	Hexagon_TSS计算速率(GFLOPS)	Best_Hexagon计算速率(GFLOPS)	Hexagon_TSS efficiency
sizeA	10*9	13.42	14.83	90.47%
sizeB	30*29	85.27	129.45	65.87%
sizeC	16*32	216.70	222.75	97.29%
sizeD	10*10	116.62	117.55	99.21%
平均				88.21%

表8中，Hexagon_TSS算法效率随问题规模的增大而上升，在大多数问题规模下都能达到90%以上。但Hexagon_TSS算法在问题规模为sizeB时的效率明显低于其他问题规模，这是因为对于2维Jacobi计算，在sizeB的问题规模下Hexagon_TSS算法计算得到的分块大小使得六边形分块的访存地址跨度超过了L1缓存的容量，使得程序跨过了L1缓存性能悬崖（Cache Performance Cliffs）；但分块的访存地址跨度又远远没有达到L2缓存的容量，分块大小相对于L2缓存容量而言偏小，未能最大化分块的时间维重用次数。综合来看，Hexagon_TSS算法在绝大多数问题规模下都具有良好的效率，但在个别问题规模上仍然存在改进空间。

3.3 程序性能测试

本节将进行两次控制变量实验，在默认实验参数下，改变某一项实验参数，记录各个优化版本对应的测试程序的浮点计算速率：1.改变并行线程数；2.改变问题规模。

3.3.1 不同并行线程数下性能对比

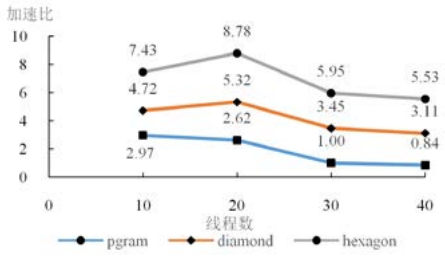
设置4种并行线程数:10、20、30、40，记录baseline、pgram、diamond、hexagon版本的测试程序在不同并行线程数下的浮点计算速率，其结果在表9中展示。

表9中，最高计算速率为287.79GFLOPS，对应测试程序为seidel-2d，优化方法为diamond，并行线程数为30。用pgram、diamond、hexagon版本的计算速率分别除以baseline版本的计算速率，得到pgram、diamond、hexagon版本相对于baseline版本的加速比，结果如图6所示。其中横轴表示并行线程数，纵轴表示相对于baseline版本的加速比。由图6可知，最大加速比为8.78，对应的测试程序为Jacobi-1d，优化方法为hexagon，并行线程数为20。在四种测试程序中，hexagon版本在40线程下的加速比与单核下的加速比的比值都达到了50%，表明hexagon方法具有良好的并行可扩展性。

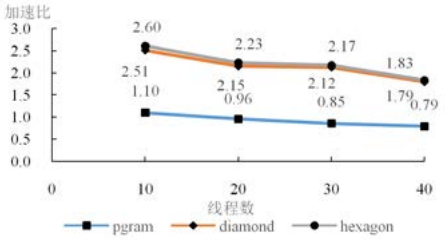
表9 不同优化版本测试程序在不同并行线程数下的计算速率

测试程序	优化版本	计算速率 (GFLOPS)			
		10线程	20线程	30线程	40线程
jacobi-1d	baseline	6.20	9.40	21.34	22.55
	pgram	18.38	24.58	21.23	18.94
	diamond	29.64	54.08	74.91	94.41
	hexagon	46.06	82.51	127.03	124.80
heat-2d	baseline	39.71	74.24	99.45	108.06
	pgram	43.64	71.79	87.02	98.11
	diamond	100.52	161.68	218.69	212.33
	hexagon	103.56	168.23	217.02	200.21
heat-3d	baseline	41.63	44.12	44.23	35.93
	pgram	12.82	20.26	29.87	31.01
	diamond	78.92	70.14	51.01	39.92
	hexagon	104.53	78.24	55.92	34.67
seidel-2d	baseline	51.75	87.87	130.05	150.78
	pgram	53.40	89.90	107.11	126.28
	diamond	122.79	209.58	287.79	271.15
	hexagon	133.22	217.18	276.85	277.78

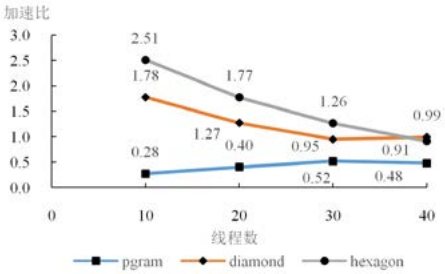
循环分块的加速效果总体上随着线程数的增加趋于下降，且hexagon优势变小，主要原因是随着线程数增多，程序执行的并行度饱和，主要性能瓶颈转变为访存，增加线程数获得的性能收益减小，同时线程同步开销逐渐增大，因此整体成下降趋势;而从表7结果显示hexagon方法在访存方面的效率仅略优于其他分块方法，当主要性能瓶颈逐渐转变为访存后，hexagon相比与其他方法的优势就减少了。特别地，Jacobi-1d测试程序中diamond和hexagon版本的加速效果则表现为先上升后下降，这是因为循环分块对1维Jacobi计算具有较好的访存优化效果，经过优化的Jacobi-1d程序在线程数较少时主要性能瓶颈是并行度，此时增加线程数有利于提升加速效果;当Jacobi-1d的并行度饱和后，主要性能瓶颈转变为访存，增加线程数获得的性能收益减小，同时线程同步开销逐渐增大。



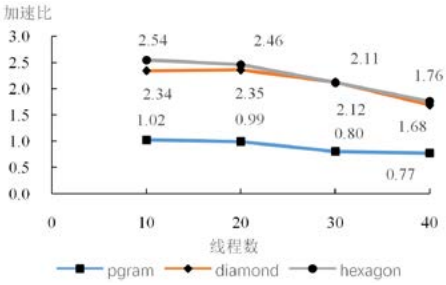
(a)jacobi-1d



(b) heat-2d



(c) heat-3d



(d) seidel-2d

图6 不同并行线程数下循环分块加速效果

此外，从图6的四张子图中可发现，diamond和hexagon版本的加速效果明显优于pgram版本，表明菱形分块和六边形分块相比于平行四边形分块在Jacobi计算优化上具有更好的性能表现。而hexagon版本在Jacobi-1d测试程序中的加速效果明显优于diamond版本，在其他测试程序中的加速效果也略微优于diamond版本或与其基本一致，这主要是因为Hexagon_TSS算法针对1维Jacobi进行了数据对齐优化，使得六边形分块在1维Jacobi计算上相对于菱形分块具有更高的向量化效率和更低的IPI。

注意到循环分块优化在四种测试程序中具有不同级别的加速效果，在Jacobi-1d上最佳、其次为seidel-2d、然后是heat-2d，加速效果最低的是heat-3d。循环分块优化的加速效果与程序本身的计算访存比有关，计算访存比越小，那么程序的访存作为性能瓶颈越突出，循环分块的优化效果越好。

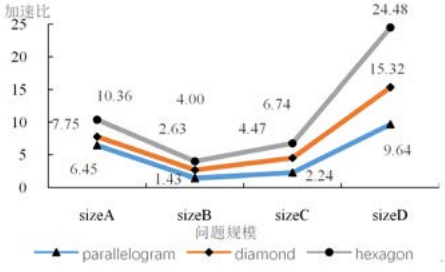
3.3.2 不同问题规模下性能对比

设置四种问题规模：sizeA、sizeB、sizeC、sizeD，记录baseline、pgram、diamond、hexagon版本的测试程序在不同问题规模下的浮点计算速率，其结果在表10中展示。表10中，最高计算速率为211.40 GFLOPS，对应测试程序为seidel-2d，优化方法为hexagon，问题规模为sizeC。

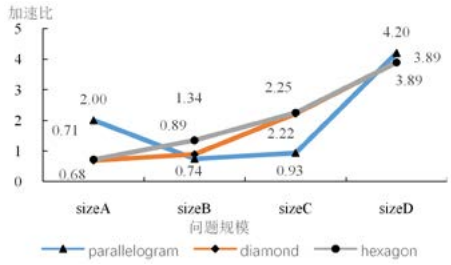
表10 不同优化版本测试程序在不同问题规模下的计算速率

测试程序	优化版本	计算速率 (GFLOPS)			
		sizeA	sizeB	sizeC	sizeD
jacobi-1d	baseline	2.22	12.32	11.82	3.38
	pgram	14.30	17.60	26.46	32.58
	diamond	17.17	32.34	52.78	51.81
	hexagon	22.96	49.23	79.62	82.76
heat-2d	baseline	12.89	48.23	73.41	20.01
	pgram	25.75	35.61	68.40	84.05
	diamond	8.78	42.78	162.61	77.85
	hexagon	9.21	64.75	164.96	77.93
heat-3d	baseline	12.61	38.36	44.74	21.55
	pgram	6.90	11.61	19.05	14.89
	diamond	8.04	28.65	76.06	40.09
	hexagon	8.27	31.59	90.04	39.95
seidel-2d	baseline	17.78	55.32	96.31	29.47
	pgram	31.14	39.26	86.14	102.92
	diamond	13.14	62.48	204.77	120.71
	hexagon	14.30	82.24	211.40	116.47

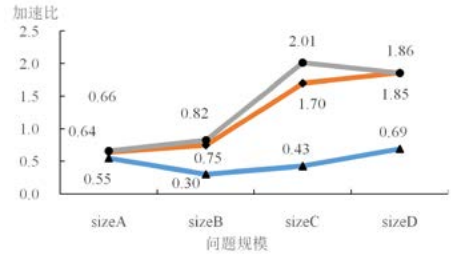
根据表10给出的数据计算pgram、diamond、hexagon版本相对于baseline版本的加速比，做加速比随问题规模变化的折线趋势图，如图7所示。其中横轴表示问题规模，纵轴表示相对于baseline版本的加速比。由图7可知，hexagon版本优化的最大加速比为24.48，对应的测试程序为Jacobi-1d，问题规模为sizeD.hexagon版本优化在Jacobi-1d程序中具有最佳的加速效果，在其他测试程序中加速效果与diamond版本基本一致。pgram版本优化在2维Jacobi计算的小问题规模上加速效果优于diamond和hexagon版本。



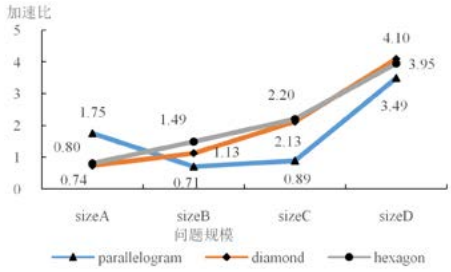
(a) jacobi-1d



(b) heat - 2d



(c) heat - 3d



(d) seidel - 2d

图7 不同问题规模下循环分块加速效果

整体上，循环分块在Jacobi - 1d测试程序中具有最佳的加速效果，在heat - 3d测试程序中的加速效果最低，pgram版本在heat - 3d程序中所有问题规模都表现出负优化的效果。循环分块的加速效果随问题规模的扩大而上升，但加速效果的上升趋势在问题规模达到sizeD时放缓，主要是由于问题规模的扩大导致程序的主要性能瓶颈从计算转换到访存。特别地，在2维Jacobi计算中循环分块的加速效果在问题规模达到sizeD后出现明显下降，这是因为在2维及以上的Jacobi计算中，分块大小随着问题规模而增大；当问题规模超过sizeD后，分块基本失去了局部性收益。理论上3维Jacobi计算在问题规模达到sizeD后分块局部性收益应出现明显下降，但由于循环分块在3维Jacobi程序上的加速效果本就不佳，所以实验中没有表现出加速效果的明显下降。

4. 总结

Jacobi计算的性能热点是嵌套循环，且Jacobi计算具有可观的数据重用，采用循环分块的优化方法能够显著地提升Jacobi计算的性能。分块大小对循环分块的效果具有重要的影响，现有关于分块大小的研究大多以最大化分块局部性、最优化并行性为优化目标，但随着处理器技术的进步，分块大小对向量化效率的影响逐渐变得不可忽视。针对上述问题，本文选择六边形分块作为优化Jacobi计算的主要措施，提升Jacobi计算的局部性，充分利用多核处理器的计算能力，提高了Jacobi计算的浮点运算速率。对Jacobi计算的优化为大量的科学计算应用提供了参考优化方案，具有重要的研究价值。

本文所做的工作还存在不少可优化的空间。例如，本文设计和实现的六边形循环分块优化方法并非在各种测试环境下都能达到最优的性能表现，现有的其他优化方法在部分环境下的效果优于六边形循环分块。因此，设计一种优化策略分支选择模型，根据运行环境选择最佳的优化方案是未来可以展开的工作。此外，在实际应用中，影响分块性能效果的因素非常复杂，难以建立精确的数学模型来求解最佳分块大小。本文提出的Hexagon_TSS算法是一种静态分块大小选择算法，不一定能够选择到最佳的分块大小，但可以获得接近最佳分块大小的性能。采用机器学习的方案，训练深度神经网络来求解最佳分块参数是当前的一个热门思路，但这需要大量复杂的特征工程方面的工作，并且需要海量的训练数据集。因此将机器学习方法与静态方法相结合的思路将是分块大小选择算法下一步的工作方向。

参考文献：

[1] Stoltzfus L， Hagedorn B， Steuwer M， Gorlatch S， Dubach C. Tiling optimizations for stencil computations using rewrite rules in Lift. ACM Trans. Archit. Code Optim. 2019， 16(4):1 - 25.

[2] Tu J， Yeoh G H， Liu C. CFD Techniques: The basics - sciencedirect. Computational Fluid Dynamics (Third Edition), 2018. 155 - 210.

[3] Pouchet L N， Bondhugula U， Bastoul C， et al. Loop transformations. In: Proc. of Acm Sigplan - sigact Symposium. ACM, 2011， 46:549.

[4] 刘松，伍卫国，赵博等. 面向局部性和并行优化的循环分块技术.计算机研究与发展。2015，52(5):1160 - 1176。

[5] Pop S， Cohen A， Bastoul C， Girbal S， Silber GA， Vasilache N. GRAPHITE: Polyhedral analyses and optimizations for

GCC. In: Proc. of the 4th GCC Developper's Summit. 2006: 179 - 198.
[6] Sun X H， Liu Y H. Utilizing Concurrency: A New Theory for Memory Wall. Springer， Cham， 2016.
[7] Xue J. Loop tiling for parallelism. 2nd Edition. New York: Springer Science & Business Media， 2012.
[8] Neumaier A. introduction to numerical analysis. Cambridge University Press， 2001.
[9] 陈国良.并行计算:结构·算法·编程.高等教育出版社，2011。
[10] 赵捷，李颖颖，赵荣彩.基于多面体模型的编译“黑魔法”。软件学报，2018，29(8):2371−2396. http://www.jos.org.cn/1000 - 9825/5563.htm。
[11] Bondhugula U，Hartono A，Ramanujam J， et al. A practical automatic polyhedral parallelizer and locality optimizer. In: Proc. of the 2008 ACM SIGPLAN conference on Programming language design and implementation.ACM， 2008.
[12] Grosser T，Cohen A，Kelly P，et al. Split tiling for GPUs: Automatic parallelization using trapezoidal tiles. In: Proc. of the 6th Workshop on General Purpose Processor Using Graphics Processing Units， 2013: 24 - 31.
[13] Grosser T，Cohen A，Holewinski J， et al. Hybrid texagonal/classical tiling for GPUs. In: Proc. of Annual IEEE/ACM International Symposium on Code Generation and Optimization.2014.
[14] Grosser T，Verdoolaege S，Cohen A，et al. The relation between diamond tiling and hexagonal tiling. In: Proc. of 1st International Workshop on High - Performance Stencil Computations. 2014.
[15] Bondhugula U，Bandishti V，Pananilath I. Diamond tiling: Tiling techniques to maximize parallelism for stencil computations. IEEE Transactions on Parallel and Distributed Systems， 2017， 28(5): 1285 - 1298.
[16] Wolf M E，Lam M S. A data locality optimizing algorithm. ACM SIGPLAN Notices， 1991， 26(6):30 - 44.
[17] 刘松，赵博，蒋庆等.一种面向循环优化和非规则代码段的粗粒度半自动并行化方法.计算机学报，2017，40(9):2127 - 2147。
[18] Whaley R C，Petitet A，Dongarra J J. Automated empirical optimizations of software and the ATLAS project. Parallel Computing， 2001， 27(1 - 2):3 - 35.
[19] Wang Z， O ’ Boyle M. Machine learning in compiler optimization. Proceedings of the IEEE， 2018.
[20] 刘慧，徐金龙，赵荣彩等. 学习模型指导的编译器优化顺序选择方法。计算机研究与发展， 2019， 56(9)。
[21] Xiang X， Chen D， Hao L， et al. HOTL: A higher order theory of locality. ACM SIGPLAN Notices， 2013， 48(1):343.
[22] Sabarimuthu J M， Venkatesh T G. Analytical miss rate calculation of L2 cache from the RD profile of L1 cache. IEEE Trans. Computers，2018，67(1): 9 - 15.
[23] 狄鹏，胡长军，李建江.GPU上高效Jacobi迭代算法的研究与实现。小型微型计算机系统，2012，33(9):1962 - 1967。
[24] Liu S，Cui YZ，Zou NJ，Zhu WH，Zhang D，Wu WG. Revisiting the parallel strategy for DOACROSS loops. Journal of Computer Science and Technology，2019，34(2): 456 - 475.
[25] Li Y， Schwiebert L. Memory - optimized wavefront parallelism on GPUs. International Journal of Parallel Programming，2020， 48(8):1 - 24.
[26] Assis I， Fernandes J B，Barros T，et al. Auto - tuning of dynamic scheduling applied to 3D reverse time migration on multicore systems. IEEE Access，2020，PP (99):1 - 1.
[27] Huang PH，Chen YS，Liao JH. QT - adaptation engine: Adaptive QoS - aware scheduling and governing in thermally constrained mobile devices. IEEE Transactions on Computer - Aided Design of Integrated Circuits and Systems，2019，39(3):1 - 1.
[28] Guo Y，Zhao J，Cave V，Sarkar V. SLAW: A scalable locality - aware adaptive work - stealing scheduler for multi - core systems. In: Proc. of IEEE International Symposium on Parallel Distributed Processing， 2010， 45 (5):341 - 342.
[29] Karimov J，Rabl T，Markl V. PolyBench: The first benchmark for polystores. Springer， Cham， 2018.
[30] Bondhugula U，Hartono A，Ramanujam J，Sadayappan P. A practical automatic polyhedral parallelizer and locality optimizer. In: Proc. of the 29th ACM SIGPLAN Conf. on Programming Language Design and Implementation. 2008. 101−113.

一种基于SMT的指令级功能验证方法

谭坚 罗巧玲 王丽一 胡夏晖
江南计算技术研究所 无锡 214083 tanjian131@163.com

摘要：

处理器研制过程中需要对指令正确性进行充分验证。现有模拟验证方法在指令结果操作数约束、操作数之间关系以及指令内部关系等方面的建模过程中存在不足。本文提出一种基于可满足模型理论的指令约束求解方法，利用SMT求解器的强大表达能力，将指令功能验证任务转化成约束满足问题，生成可用于指令功能验证的测试数据。文章在三个方面分别给出示例，阐述了利用SMT求解器进行约束建模的关键过程。针对模型存在的效率较低问题，利用时间阈值实现超时终止求解的策略；利用线程加速技术，实现了指令功能约束并行求解框架，充分利用通用服务器的计算资源，在保证求解质量的同时大大加速了求解速度。该技术已成功应用于指令功能测试验证中，有效提升测试覆盖与质量，取得了很好的效益。

关键词：指令功能、约束求解、SMT求解器、验证数据

1. 概述

随着计算机体系结构理论与技术发展、集成电路设计日趋复杂、制造工艺水平不断提升，行业内科研人员设计实现的处理器逻辑功能变得越来越复杂，对处理器测试验证质量的要求变得越来越高。为保证处理器功能正确性，硬件设计、验证人员、系统软件测试人员甚至用户都或多或少会承担着处理器功能与性能的测试验证任务。指令级功能测试是处理器正确性验证的重要组成部分，贯穿处理器研制过程中的各个阶段，在软件模拟指令验证、FPGA实物验证、硬件模拟仿真验证以及SoC硅后验证等阶段都需要进行充分验证；指令级测试方法具有通用性的特点，与具体的处理器类型无关，在专用处理器与通用处理器研制过程中都需要用到，其实现方法依赖于指令语义、内部规范以及指令上下文等。

在开展指令级功能测试的过程中，我们发现对于测试方案所要求覆盖的部分特殊场景，采用现有模拟验证方法比较难以达到目的。例如，当需要对单指令的结果进行各种数据类型的遍历验证时、在约定浮点操作数的中间结果为某特定数值时、在约定两个操作数之间满足特定关系时尤其是约定输入操作数与输出操作数之间满足特定的约束关系时，采用现有主流的模拟验证方法[1]通常显得比较低效。此外，很多的指令功能是隐含一些特定的约束关系，如加减法、乘法指令时存在进、借位情况的发生。这些内部关系对于测试人员来说不直接，部分约束甚至需要处理器设计协助制定。采用现有模拟方法，在制定测试验证计划或者生成测试程序的时候很难全面覆盖，即使能够实现代价也比较大。以上提及的场景都是比较常见的验证功能点，要求测试人员必须覆盖。这些场景虽然属于不同的类型，但都是指令功能内部满足约束的功能点。

面对结果操作数与指令内部关系进行约束的验证场景，如果采用功能模拟测试方法，通常需要测试人员根据指令的语义模拟出该指令的相反语义，将输出转换成模拟程序的输入，反推出指令的输入操作数，形成验证元组数据，再根据验证数据形成相应指令的测试激励验证指令功能正确性。部分指令其实是没有严格意义对应的求反过程的，例如浮点指令，假若存在浮点操作数A×B=C，但是由于舍入模式等原因，导致数据元组<A,B,C>在很多时候并不满足C/B=A的关系。

因此，对于系统测试人员而言，很多的待验证场景采用现有的功能模拟等手段是比较难以实现的；即使采用算法模拟成功实现，得到验证数据的运行时间通常较长，生成的数据量也比较有限。

国内外处理器设计研发机构如Intel、IBM等公司，为满足处理器特殊约束场景的验证需求，组织了大量的科学家专门设计并实现了满足各自处理器

约束建模需求的求解器[2][3]。根据IBM公司官网显示，以色列海法实验室研究开发出了GEC、Stocs、Mage等专用求解器[4]，解决处理器验证过程中面临的各种功能验证需求。由于这些求解器的实现与处理器内部设计细节紧密相关，因此这些专用求解器仅限在公司内部以及合作单位之间进行验证，一般不用于商用和开源。

由于与设计紧密关联并且需要参考模型等底层环境和技术的支撑配合，目前现有以功能模拟为基础的系统级测试基本不大可能直接全面覆盖带约束的待测场景。很多带约束的待测场景对系统级测试人员来说是可见的，如此一来将会使系统测试在功能覆盖方面将存在明显缺陷。如果不加以覆盖，将会导致经过系统测试过的目标处理器仍然存在设计错误的隐患，这是处理器研制而言是不能容忍的。随着形式化验证技术的快速发展，我们可以在系统级测试过程中引入形式化方法，对指令约束关系描述并建模求解。在验证场景与处理器功能测试覆盖任务之间建立可靠的技术桥梁，解决工程实际需要，提升处理器验证测试质量。

本文首先对一种形式化技术方法——可满足模理论（Satisfiability Modulo Theory，以下简称SMT）进行简要介绍；然后提出了一种基于SMT求解器的关系约束求解技术，分别从指令结果操作数约束、操作数相互关系约束和指令内部关系约束三个方面，阐述SMT技术在解决指令功能关系约束中的具体实现方法，给出了具体约束建模求解算法；文章指出了求解过程中存在的常见问题并有针对性地提出了优化策略，提高了模型整体求解效率；最后，我们对SMT技术在指令功能验证中的应用进行了总结，对后续进一步工作进行了展望。

2. 可满足模理论概况

可满足模理论（Satisfiability Modulo Theory，简称SMT）[5]是基于布尔命题可满足问题（Boolean Satisfiability Problem，简称SAT）发展起来的一种形式化技术。其中SAT能够判定给定的合取范式（CNF）是否存在可以满足的模型，即判别是否存在满足所有给定公式都为真的变量赋值。值得一提的是，SAT求解技术在微处理器RTL级约束验证中被广泛应用。研究人员开发了很多成熟的SAT求解器；这些求解器被当作一种支撑的问题判定技术，已经广泛应用于人工智能领域和其他问题求解项目中。SMT技术是在SAT命题逻辑的基础上扩充了量词和项，融合了多种背景知识理论，扩充了问题描述与应用求解范围。

SMT求解器支持用户定义变量、声明变量之间约束关系；SMT求解器将用户自定义的约束描述自动转换成命题逻辑甚至谓词逻辑公式，然后通过一定的搜索策略、调用底层SAT求解器进行问题判定求解，问题可满足时将给出一个满足所有约束关系的随机模型，对用户定义的变量进行随机赋值；否则求解器将反馈问题是不可满足的。

SMT有很多种具体的、被广泛使用的求解器，各个求解器之间的背景理论存在差别，因此不同求解器有各自擅长的问题求解领域。其中比较通用的是纽约大学开发的CVC3/CVC4、微软公司开发的Z3等，其表达能力很强，支持比特位、整型、实数、浮点、集合类型的变量定义与约束关系描述。支持数组、线性算术运算、差分逻辑的关系表达，甚至可以将不同领域的知识进行组合，实现复杂问题的求解。SMT求解器强大的表达能力在处理器验证领域有着广泛的技术前景[6][7]。

本文正是利用SMT求解器强大的问题描述能力，对指令功能验证中存在比较复杂的内部约束关系进行建模。借助SMT求解器自动判断与可满足情况下随机模型生成的特点，生成测试验证数据，达到覆盖处理器系统级测试任务中复杂场景的目的。

需要说明的是，论文接下来的部分都是采用Z3求解器进行建模描述并获取求解结果的，实验求解器不仅限于Z3，采用其他通用SMT求解器如CVC4同样可以满足需求。

3. 基于SMT求解器的指令操作数约束求解技术

系统测试中基于指令的操作数约束是通过对指令操作数施加约束，验证是否满足给定的约定操作数，利用生成验证元组数据形成对应的测试激励，验证目标机器是否满足该约束对应的验证数据。本节针对采用模拟验证的方式在三种情形不便于进行覆盖的场景，说明采用SMT求解器实现指令功能约束求解技术的优越性。

3.1 结果操作数约束

模拟验证方式对于很多指令的结果操作数不能直接、高效地进行约束求解。例如无符号乘法指令MUL Ra,Rb,Rc实现是Ra×Rb=>Rc的功能，这里三个操作数均为无符号整型类型。如果用户需要对Rc进行数据类型遍历，同时限定Rb时大于1的整数。那么需要求解Ra，则需要测试人员模拟出形如Ra=RbOPRc，然后制定相应算法实现求解。然而很多时候存在实现代价太大甚至是无法求解的情形。但这时候采用SMT求解器则很容易表达与求解，因为其背景理论支持各种算术表达式的约束求解。对于

约定Rc=111，同时要求Rb为大于1的整型数据，那么要求解满足条件的Ra，用户可以采用如下伪代码约束表达方式进行描述：

args[0]=mk_gt(Rb,one); ……//描述Rb大于1
args[1]=mk_mul(Ra,Rb,Rc); ……//描述Rc=Ra × Rb
args[2]=mk_eq(Rc,111); ……//描述Rc=Ra × Rb
c=mk_and(args,3) ……//同时满足上述三个约束条件

其中mk_gt、mk_mul、mk_eq、mk_and都是SMT求解器算子，Ra，Rb，Rc则是用户使用SMT求解器API声明的变量；mk_gt(Rb,one)描述的是Rb大于1的约束关系；mk_mul(Ra,Rb,Rc)描述的是Rc等于Ra乘以Rb的约束关系；mk_eq(Rc,111)描述的是Rc的值等于111的约束关系。然后将以上三个约束关系分别赋值给args[0]、args[1]、args[1]子公式中，c=mk_and(args,3)描述的是将上述三个公式进行合取并赋值给CNF公式c。

上述结果操作数约束伪代码采用Z3求解器建模描述得到CNF范式以及随机求解模型结果如表1所示：

表1 结果操作数约束Z3建模CNF公式以及求解模型

Z3求解器CNF范式	and((bvugt (Rb #x0000000000000001)) (= Rc (bvmul Ra Rb)) (= Rc #x000000000000006f))
Z3求解模型	Ra -> #x03b0a38c0203d963 Rb -> #x51ff99244304b085 Rc -> #x000000000000006f

最后，形成的公式c表示需要满足所有三个约束关系；利用SMT求解器进行求解，求解器在运算过程将始终保持上述三个约束条件，如果有解，则系统将返回满足所有上述约束条件的一组Ra，Rb，Rc的随机赋值。

对于其他复杂语义的指令，可以通过增加约束变量以及借助SMT支持的命题逻辑运算符进行描述，实现给定约束的建模描述。

3.2 操作数之间关系约束

模拟验证的方式在对操作数之间的约束关系描述是也是不方便的。例如对于电气特性的测试需求，验证时通常会对操作数之间进行约束。仍然以无符号乘法MUL Ra, Rb, Rc为例。如对测试数据进行约束，要求遍历Rc的数值中1的个数从0到64的所有情况，同时要求Ra或者Rb中1的个数不能多于Rc的。这时采用模拟验证的方式就很难高效的实现。

相对而言，采用SMT求解器虽然也是比较复杂但仍然能够比较轻易地进行表述并求解。

用户可以增加变量N_1、N_2、……、N_{64}，令分别对应Rc中最低n比特中1的个数；则可以得到如下约束公式：

args[0]=mk_ite(extract(Rc,0,0),(N_1=1),(N_1=0)) ……//抽取Rc的第0位进行判断：如果Rc[0]为1则，N_1赋值为1，否则赋值为0；

args[j]=mk_ite(extract(Rc,i,i),(N_{i+1}=N_i+1)),(N_{i+1}= N_i))) ……//抽取Rc中的第i比特位判断：如果Rc[i]为1则，N_{i+1}赋值为N_i+1，否则N_{i+1}赋值为N_i；

以此类推，建立Rc所有的比特位计数约束关系之后，再分别建立Ra或者Rb的比特位计数约束关系NRa$_i$。

最后args[k]=mk_le(NRa64, N64)。

其中mk_ite、extract、mk_le是SMT求解器算子：mk_ite描述的是一种if-then-else的约束关系；extract实现的是抽取SMT变量的某几个比特位的功能；mk_le描述的是一种小于等于的约束关系。

mk_ite(extract(Rc,i,i),(N_{i+1}=N_i+1)),(N_{i+1}=N_i)))描述的是抽取Rc中的第i个比特位并进行判断：如果Rc[i]为1则，N_{i+1}赋值为N_i+1，否则N_{i+1}赋值为N_i。

通过上述关系的描述，系统将合取所有公式集合，形成满足所有上述约束条件的公式c，以此作为求解器的输入进行约束求解。在约束可满足时，系统将随机给出满足所有子公式约束的模型，测试人员可以根据模型形成验证数据，并由此生成覆盖特殊场景的测试激励。

上述操作数之间关系约束伪代码采用Z3求解器建模描述得到CNF范式以及随机求解模型结果如表2所示，这里仅给出满足上述条件且在Rc的操作数比特位表示法中1的个数为8的情况：

表2 操作数之间关系约束Z3建模CNF公式以及求解模型

Z3求解器CNF范式	and((ite (= ((_ extract 0 0) Rc) #b1) (=|N[0]| #0000000000000001) (=|N[0]| #0000000000000000)) (ite (= ((_ extract 0 0) Ra) #b1) (=|NRa[0]| #0000000000000001) (=|NRa[0]| #0000000000000000)) (ite (= ((_ extract 1 1) Rc) #b1) (=|N[1]| (bvadd |N[0]| #00000000000000001)) (=|N[1]| |N[0]|)) (ite (= ((_ extract 1 1) Ra) #b1) (=|NRa[1]| (bvadd |NRa[0]| #00000000000000001)) (=|NRa[1]| |NRa[0]|)) ……//与N[1],NRa[1]类似，重复2-63 (= Rc (bvmul Ra Rb)) (= |N[63]| #x0000000000000008) (bvule |NRa[63]| |N[63]|))
Z3求解模型	Ra -> #x3300000000000000 Rb -> #x0000000000000005 Rc -> #xff00000000000000

3.3 指令内部关系约束

如果测试验证人员关心计算结果中是否出现进借位的场景，采用SMT求解也是能够实现该场景约束的表达并求解。本节以64比特位的无符号加法ADDL Ra Rb Rc为例进行说明。

当约定ADDL操作为无符号长字加法以及所有操作数的值是无法确定结果是否存在进位的，因为结果约束最多就约束了64比特位。但是计算时可能发生Ra+Rb得到的有效位实际上时超过64比特位的情形。为达到加法进位场景覆盖验证的目的，我们继续给出这种情况SMT求解器的约束建模过程：

args[0]=mk_eq(Rc_64, extract(Rc_65,63,0))……//约束Rc_64与Rc_65的最低64比特位相同；

args[1]=mk_eq(Ra_64, extract(Ra_65,63,0))……//约束扩展源操作数最低64比特位与原始源操作数相同

args[2]=mk_eq(Rb_64, extract(Rb_65,63,0))…….//同上

args[3]=mk_eq(Rc_64,mk_add(Ra_64,Rb_64)).....//分别建立结果操作数Rc与未扩展之前的Ra和Rb之间的关系

args[4]=mk_eq(Rc_65,mk_add(Ra_65,Rb_65))……//建立扩展之后的结果操作数与扩展至后的源操作数之间仍然需要保持原约束关系

args[5]=mk_eq(one, extract(Rc_65,64,64))……//获取Rc_65的最高比特位并约定其为1，描述发生进位约束的情况。

args[6]=mk_not(mk_eq(one, extract(Ra_65,64, 64)))……//获取Ra_65的最高比特位并约定其不等于1。

args[7]=mk_not(mk_eq(one, extract(Rb_65,64, 64))) ……//获取Rb_65的最高比特位并约定其不等于1。

其中mk_add是SMT求解器算子，描述的是一种加法运算；mk_not描述的是公式求反运算，mk_eq，extract与前文描述功能相同。

上述操作数之间关系约束伪代码采用Z3求解器建模描述得到CNF范式以及随机求解模型结果如表3所示：

表3 指令内部关系约束Z3建模CNF公式以及求解模型

Z3求解器CNF范式	and((= Rc_64 ((_ extract 63 0) Rc_65)) (= Ra_64 ((_ extract 63 0) Ra_65)) (= Rb_64 ((_ extract 63 0) Rb_65)) (= Rc_64 (bvadd Ra_64 Rb_64)) (= Rc_65 (bvadd Ra_65 Rb_65)) (= #b1 ((_ extract 64 64) Rc_65)) (not (= #b1 ((_ extract 64 64) Ra_65))) (not (= #b1 ((_ extract 64 64) Rb_65))))
Z3求解模型	Ra -> #x8000000000000000 Rb -> #x8000000000000000 Rc -> #x0000000000000000

通过以上三种典型情况的展示，测试人员利用该技术解决绝大多数指令约束测试覆盖的验证求解需求。在解决约束表达与求解的问题之后，测试人员则可以将更多的时间精力放到测试场景的构建中，确保覆盖更多、更特殊情况的测试场景，提高测试针对性和测试质量，有效提升测试覆盖率。

4. 求解优化技术

采用SMT对测试场景进行约束建模与求解的过程中，往往会遇到约束遍历求解耗时较长的问题，尤其是建立较多SMT约束变量、约束关系较为复杂的情况下，求解时间可能会非常长。

在指令约束建模编码的实践过程中，通过SMT技术建立的求解程序通常是由若干个子问题的求解过程组成。而单次求解是一个单线程串行计算过程。当某个求解耗时很长甚至时无解时，求解器可能长时间在挂住某一次求解的情况，导致后续求解必须依赖于前面的求解结束之后才能继续的问题。经过作者时间发现，很多求解器都提供了超时终止求解的功能。针对该问题，SMT用户在构建求解框架时，通过设定时间阈值，系统将在求解开始之前进行计时，当达到给定时间仍然不能判定出问题是否可满足时，系统将终止本次求解过程；转而继续后续的求解判定。这样就可以避免整个求解模型挂死在单个无效的问题求解上。通过实现参数化框架，用户可以根据模型求解的特征，对不同的约束模型设定不同的时间阈值，保证约束求解不会因为时间阈值的限制导致求解质量明显下降。

作者还发现在构建遍历所有约束条件的模型求解程序中，各个子约束求解过程相互之间是松耦合的。通过约束模型求解的合理设计与组织，不同求解过程是可以拆解成相互独立的计算过程。这为采用并行化技术加速整个框架求解提供了可能。作者将耦合度不高的模块，分布到不同的线程，由各个线程并行独立求解，所有线程求解结束之后将结果进行整合，形成统一的约束求解数据。

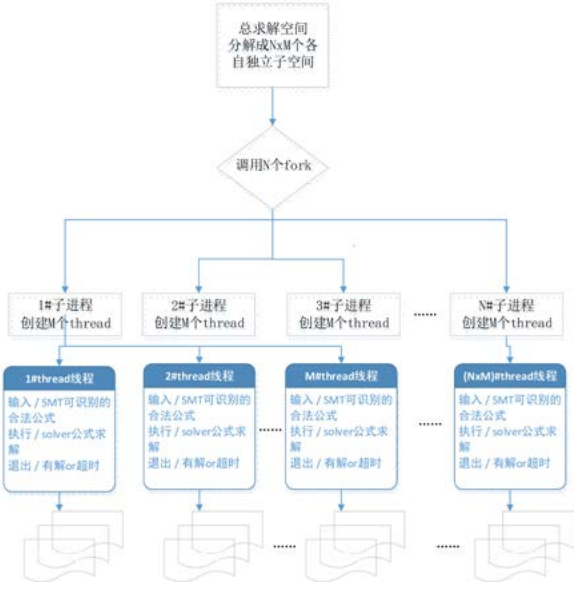


图1 基于fork+thread的并行加速求解模型示意图

本文作者在实践过程中成功实现了一种多进程与多线程相结合的并行求解框架。通过分析整个求解框架的结构特点，顶层使用fork创建多个进程，并在进程中创建多个线程的方式，实现将原本顺序、串行的遍历约束求解过程分散到$N\times M$个线程中，充分利用当前服务器的计算资源进行问题求解，运用空间换取时间，结合前述超时终止求解的技术，采用参数控制的形式，实现有限时间内对指令约束模型的求解框架。

作者使用4.8.0版本的Z3求解器，在一台32核的intel服务器上进行试验，分别设置N=4、M=5。

这里N对应指令约束模型中操作数的数目，因为作者面对的处理器指令最多只可能为4操作数，因此将N选定为4。M对应的是作者模型中操作数的类型。由于采用32核处理器，在保证服务器不用满的情况下选定M为5，该值可以根据实验环境进行调整。

同时我们选择了ADDL和逻辑运算指令LOG2XX两个比较有代表性的指令，分别对这两条指令的约束求解模型的加速前后的效果进行比较分析。其中ADDL模型相对简单，求解建模实现时仅有6个约束变量、针对5个约束关系形成了5个公式，运行过程中单次求解时间相对较短。LOG2XX指令较为复杂，根据指令语义每种数据类型遍历的情况下存在16个不同的操作码，分别定义了8个约束变量，对约束变量建立的约束关系超过128个，求解时间相对较长。

此外，实验过程中，作者将单次求解的时间阈值设置为10秒，即10秒内如果约束公式不能判定是够有解则终止，转而对下一约束关系模型进行求解。每条指令均选择了4组数据，分别对应了单种约束情况下求解得到1、10、100和1000组随机模型的情况。这里的1、10、100和1000分别对指令的每一个操作数、每一种操作数数据类型进行完全遍历时，在每种情况下运行的求解次数。结合前述说明，我们得知系统在设置的每种情况求解1次随机模型时，ADDL指令对应的模型将累计产生20次求解，LOG2XX指令将累计产生20×16=320次求解。两条指令的在串、并行模式下运行求解时的加速比分别如图2、图3所示。

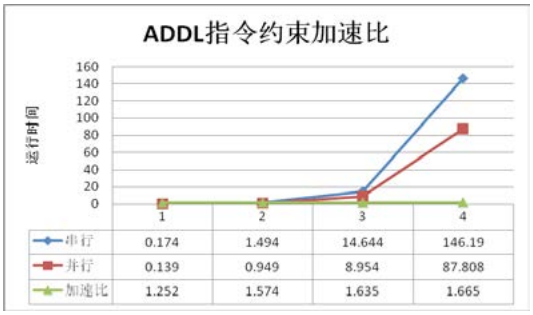


图2 ADDL指令约束求解模型并行加速示意图

通过分析，我们看到在累计使用20个线程并行加速的情况下，ADDL在四种情况下实际得到的加速效果最多可达1.66倍，LOG2XX最大可达2.81倍。数据显示，在相同约束求解模型内，随着求解次数的增加，加速效果越好。尽管远没有达到$N\times M$的加速比，但是相比串行顺序求解的方式，并行求解模型的加速效果依然非常明显。

图3 LOG2XX指令约束求解模型并行加速示意图

综上所述，通过比较两条指令的求解时间，我们看到复杂模型的求解时间较长，在约束复杂度相同的情况下，求解时间基本与求解数目成线性关系；不同模型之间，复杂度较大的模型加速效果较为明显；初步结论显示，随着约束模型的复杂度增加和求解次数增多，并行求解模型的加速效果较为明显。

5. 结束语

本文首先阐述了现有基于功能模拟验证的指令功能测试方法在结果操作数约束、操作数之间的约束、指令内部约束等场景存在不足的场景。我们提出了一种基于SMT求解器的指令操作数约束求解技术。针对三种典型待验证场景，分别给出了详细的建模描述过程与求解结果。针对模型求解过程中存在求解时间较长的问题，采用时间阈值机制、超时则终止本次求解的策略；同时利用进程与线程管理技术，牺牲CPU资源换取时间的策略，对指令约束模型求解实现加速。实验结果表明，时间阈值策略与进程加线程的并行框架提升了模型求解效率。

实验实践表明，采用SMT技术基本能够实现指令功能验证任务中各种场景的建模描述与求解的需求，为测试实施人员提供可靠的技术支撑。

在今后的工作中，我们将继续探索SMT技术在复杂约束生成、指令序列约束等验证场景中的应用前景；进一步探索优化约束求解技术，充分利用现有商用处理器资源，寻找加速模型求解的有效途径；尽可能在短时间内生成更多的验证测试数据，更好、更高效地满足处理器研制过程中的功能正确性覆盖需求，全面提升处理器系统测试的质量。

参考文献：
[1] 沈海，卫文丽，陈云霁. 覆盖率驱动的随机测试生成技术综述[J].计算机辅助设计与图形学学报,2009, Vol. 4: 419 - 431.
[2] Roope Kaivola, Rajnish Ghughal, Naren Narasimhan, Amber Telfer, Jesse Whittemore, SudhindraPandav, Anna Slobodov´a,Christopher Taylor, Vladimir Frolov,Erik Reeber, and ArmaghanNaik. Replacing Testing with Formal Verification in Intel CoreTM i7 Processor Execution Engine Validation. CAV, volume 5643 of Lecture Notes in Computer Science. Springer, 2009：page 414 - 429
[3] Yehuda Naveh , Michal Rimon , Itai Jaeger. Constraint - Based Random Stimuli Generation for Hardware Verification. National Conference on Artificial Intelligence & the Eighteenth Innovative Applications of Artificial Intelligence Conference,2007
[4] https://www.research.ibm.com/haifa/dept/vst/csp_solver.shtml
[5] https://en.wikipedia.org/wiki/Satisfiability_modulo_theories
[6] 金继伟，马菲菲，张健. SMT求解技术简述[J]. 计算机科学与探索, 2015, 9(7): 769 - 780.
[7] 王翀,吕荫润,陈力,王秀利,王永吉. SMT求解技术的发展及最新应用研究综述[J].计算机研究与发展,2017, Vol. 54 Issue (7): 1405 - 1425

要闻集锦

调研机构：2025年AI软件市场将达1260亿美元

据数据调研机构BanklessTimes预测，由于对更高效和有效的解决方案的需求不断增加，全球AI软件市场预计将在未来两年内达到1260亿美元。

BanklessTimes的首席执行官Jonathan Merry表示，AI软件市场正以前所未有的速度增长，我们每天都在看到AI的新应用和创新。这是一个强大的工具，可以彻底改变企业的运营方式，允许更精确决策，提高效率和改善客户体验。

AI软件正被用于工业和消费市场，以改善企业运营。人工智能解决方案也被作为客户服务战略的一部分，聊天机器人帮助客户提供产品信息和客户支持咨询。此外，AI驱动的语音助手可以帮助用户在移动设备上快速、方便地获取信息或完成任务。

医疗保健、零售、银行和制造业等多个行业的公司都成功地实现了AI软件，使日常任务自动化，同时从以前无法获得的数据中获得见解。此外，AI软件还被用于改善客户服务，为客户提供个性化的建议，创造更互动的体验。

银行业最积极地投资于AI软件，以简化操作，降低客户服务成本，并加强欺诈检测。

AI软件不仅用于商业，也用于我们的日常生活;亚马逊的Alexa或苹果的Siri等语音助手已经成为人们日常生活中不可或缺的一部分。像谷歌Now和Cortana这样的个人助理应用程序被用来为用户提供提醒、通知和更新。

AI软件市场分为自然语言处理、机器人过程自动化、机器学习等多个领域。

自然语言处理(NLP)是一种AI，它通过算法从文本中推断出意义，帮助计算机理解人类语言和自然对话。这有助于计算机更好地理解对话的上下文，并做出相应的响应。

机器人过程自动化(RPA)是AI软件的另一个例子，它可以自动化日常任务，如数据输入，这可以帮助组织提高效率和准确性。同时，机器学习使用算法识别大型数据集中的模式，并基于这些模式进行预测。这有助于组织更快地做出更好的决策，从而提高业务绩效。

随着对AI软件的需求和其革命产业的潜力的增长，全球AI软件市场预计将在未来几年稳步增长。AI软件在各个垂直行业的日益普及将有助于推动这一增长。微软、IBM、百度和谷歌等行业领导者正在大举投资AI研发，以获得竞争优势。此外，云计算服务的日益可用性将进一步推动AI软件的采用。

（陈继军）

KubeGPU: 面向容器云的GPU资源共享和隔离策略的研究

沈文枫　刘政森
上海大学计算机工程与科学学院 上海 200444

摘要：

随着越来越多的容器应用开始依赖GPU资源，在容器云中支持GPU就显得尤为重要。虽然面向虚拟机的GPU共享技术已经得到广泛研究，但是面向容器的GPU共享技术仍未得到广泛研究。现有的工作只使用一种特定的GPU虚拟化技术，如GPU直通或API转发技术，进行容器部署，并且缺乏远程GPU虚拟化的优化。由于容器资源需求和GPU虚拟化技术特性的多样性，以及通信开销和资源争夺的问题导致系统吞吐量和容器性能下降。因此，我们设计并实现了KubeGPU,其利用自适应资源共享策略扩展了Kubernetes的GPU资源管理和调度能力。自适应资源共享策略使得KubeGPU能够根据系统可用GPU资源情况以及容器请求参数，如GPU资源请求,动态选择GPU虚拟化部署容器以满足的容器运行时性能需求，提高GPU利用率和整体系统吞吐量。此外，我们提出了网络感知和远程GPU资源细粒度分配机制用于优先远程GPU虚拟化。最后，我们在一个异构GPU集群上验证了KubeGPU，实验结果表明了KubeGPU相较于其他算法的优势以及KubeGPU能够有效降低通信开销和远程GPU虚拟化的资源竞争。

1. 引言

在过去几年间，容器技术被广泛应用于工业，研究和教育等领域。由于容器水平扩展的特性，现代应用可以由成千上万的容器组成[1]。大量的容器实例所占用的计算、存储和网络资源如何有效的调度，才能更好地支持复杂应用稳定可靠地运行并且提高系统整体的资源利用率，成了一个无法回避的问题。正是在这样的需求背景下，容器集群调度管理系统应运而生。目前市面上已经存在许多容器集群管理调度系统，如Apache Mesos[2]，OpenShift[3]，Docker Swarm[4]， Kubernetes[5]。其中，Kubernetes是目前使用范围最广、最具代表性的容器集群调度管理系统，其目标是构建一个可扩展的、可配置的，同时提供应用部署、维护、扩展等功能的容器集群调度管理工具。然而，Kubernetes原生仅支持对容器实例所占用的CPU，内存和磁盘存储资源进行有效调度、管理和分配，并不支持GPU，FPGA等资源的管理[6, 7]。随着GPU的不断发展，GPU的并行运算的能力和浮点运算能力已明显优于CPU，GPU正在逐渐取代CPU，被广泛应用于人工智能，高性能计算等领域[8]。然而，Kubernetes并不能根据GPU的特性进行高效的资源调度，导致GPU无法被充分利用。因此，扩充Kubernetes的资源调度管理能力，使得Kubernetes能够调度管理GPU资源成了亟待要解决的问题。

近年来，研究人员提出了一些GPU虚拟化管理框架用于协助Kubernetes管理和调度集群上的GPU资源，这些框架可以大体分为两类：（1）第一类框架会为请求GPU资源的容器分配独占GPU资源。这种方法能够提供原生GPU性能以及良好的兼容性[9]。（2）第二类框架实现了在容器间共享同一物理GPU资源。为了实现容器间共享GPU资源，一些工作直接将一整个物理GPU资源分配给多个容器使用，一些工作则将物理GPU拆分成若干个虚拟GPU资源，并将一个或多个虚拟GPU资源分配给容器。此外，研究人员[10]实现了面向Kubernetes的远程GPU虚拟化技术，该技术可以让运行在非GPU节点的容器能够共享GPU节点的GPU资源，进一步提高了GPU资源利用率，减少了能源消耗以及集群所需安装GPU数量[11]。虽然已有一些工作致力于拓展Kubernetes管理和调度集群上的GPU资源的能力，但仍然存在下述一些限制：

（1）现有的GPU框架难以同时满足系统吞吐量和容器性能两个需求指标。第一类框架将整个GPU资

源分配给某个容器独占使用，使得其他容器无法与该容器共享使用GPU资源，这可能导致GPU利用率和系统吞吐量下降[12]。第二类框架通过在容器间共享GPU资源，提升了GPU利用率和系统吞吐量。然而，大多数GPU产品并没提供开源驱动，硬件详细规格或者通信协议以实现对GPU功能的修改[9]。大多数研究通过使用API转发技术，或者修改GPU库接口来实现GPU虚拟化技术。但是API转发技术通常带有额外的性能开销并且存在兼容性问题[13]，难以满足对计算性能有较高需求的用户。

（2）目前，面向Kubernetes的远程GPU虚拟化技术主要关注点在于利用GPU进行任务加速[10]。网络开销，共享资源干扰等问题并没有得到解决。然而，节点间的网络开销是造成远程GPU虚拟化性能下降的重要原因之一[14]。此外，共享资源干扰可能导致容器实例之间对某一资源进行争抢。比如，当GPU资源充足且不存在一种合理的资源隔离策略时，一个容器可能会占用全部GPU资源，导致其他容器的性能受到影响[15]。

为了解决上述问题，我们提出了KubeGPU，一个构建在Kubernetes之上的GPU管理框架，该框架以较低的性能开销提高总体系统吞吐量。KubeGPU还采用了网络感知调度方法和远程GPU资源细粒度分配机制优化远程GPU虚拟化。本文的贡献总结如下：

（1）GPU虚拟化技术动态选取。我们提出了一种新的GPU资源调度策略—自适应资源共享策略。自适应资源共享策略可以自动根据容器的资源需求和系统可用GPU资源量，选取合适的虚拟化技术进行容器部署。通过自适应资源共享策略，KubeGPU在不过多影响容器性能的同时，提高了GPU利用率和系统吞吐量，更好地满足了云计算中不同用户的不同需求。

（2）减少Kubernetes的远程GPU虚拟化通信开销。我们提出了一种网络感知的调度方法，通过分析底层网络并将可用的且性能更好的网络模式和网络设备®分配给容器以减少远程GPU虚拟化的网络开销。

（3）缓解远程GPU资源争抢。我们提出了细粒度资源分配机制允许用户声明它们对远程GPU资源的需求。KubeGPU根据用户声明的计算资源和显存资源需求，实现容器间远程GPU资源的隔离。

论文内容安排如下：第二节为读者介绍必要的背景知识；第三节对相关工作进行详细介绍；第四节详细介绍了KubeGPU的实现细节；第五节对KubeGPU进行实验验证；第六节对论文内容进行总结。

2. 背景

本节简要介绍了GPU容器资源管理所涉及的背景知识包括：GPU虚拟化技术远程GPU虚拟化的通信开销以及Kubernetes网络实现。

2.1 GPU虚拟化

GPU虚拟化技术可以使得容器高效地访问并使用GPU资源。容器云中使用的GPU虚拟化技术大体可以分为三类：（1）API转发技术，通过修改GPU库来拦截GPU调用[16]。（2）辅助虚拟化技术，在GPU驱动层面上实现虚拟化[17]。（3）GPU直通技术，直接将GPU分配给容器[18]。表1对上述三个GPU虚拟化技术的相关特性进行了汇总。

表1 容器云中GPU虚拟化实现方案

虚拟化方案	描述	优点	缺点
API转发技术	API转发技术通过构建一个自定义库去劫持并转发容器的GPU调用以实现GPU虚拟化	开发人员可以在不掌握GPU驱动实现细节的情况下实现GPU虚拟化; 无需修改用户程序代码.	性能开销较大; 存在不兼容问题
辅助虚拟化技术	辅助虚拟化技术通过构建自定义GPU驱动，在驱动层拦截并转发容器的GPU调用以实现GPU虚拟化	无需对原始GPU库和用户程序进行修改	大多数GPU厂商并未开源GPU驱动代码，需要使用逆向工程对原始GPU驱动进行分析以实现辅助虚拟化技术
GPU直通技术	直接将一个GPU资源分配给一个或多个容器	可以提供近乎原生的性能表现	不支持细粒度的GPU资源分配

我们基于API forwarding和GPU pass-through设计实现了KubeGPU。GPU pass-through为需要原生性能的容器分配独占GPU资源以满足容器的性能需求。API forwarding在容器间分配共享GPU资源以提高GPU利用率和系统吞吐量。此外，我们提出了计算资源动态分配机制以尽可能减少API forwarding带来的额外性能开销，计算资源动态分配机制的实现细节将在4.2小节进行详细讨论。更进一步地，我们设计实

现了自适应共享策略，该策略会根据当前系统可用GPU数量以及容器请求参数，从API forwarding和GPU pass-through选取一个合适的GPU虚拟化技术用于容器部署，以平衡容器性能，GPU利用率和系统吞吐量三者关系。自适应共享策略的实现细节将在4.2小节进行详细讨论。

2.2 远程GPU虚拟化的通信开销

远程GPU虚拟化是实现容器间共享GPU的一种特殊方法。虽然远程GPU虚拟化技术提高了系统资源利用率，减少了集群能源开销以及GPU安装数量[11]，但是远程GPU虚拟化的性能受限于节点间的通信开销[19, 20]。研究人员通过分析远程GPU虚拟化在三代GPU上的性能表现[21]，发现远程GPU虚拟化在新一代GPU上的性能损耗是旧一代的8到14倍。这是因为GPU每经历一次升级，其计算效率就变得更快，核函数的执行时间也因此变短。相应地，对于数据传输的要求就越高，这对远程GPU虚拟化的网络交互带来了压力。因此，我们提出了网络自适应感知技术以减少网络间开销以优化远程GPU虚拟化技术，以减少节点间通信对远程GPU虚拟化性能的影响。

2.3 Kubernetes网络

Kubernetes为容器提供了两种网络模式：主机网络模式和虚拟网络模式。主机网络模式通过提供原生的网络IO以减少通信开销。主机网络模式的一个缺点是会占用宿主机的网络资源。相较于主机网络模式，虚拟网络不占用宿主机网络资源，因为Kubernetes会为使用虚拟网络模式的容器分配一个独立的网络空间。Kubernetes自身并不负责网络空间的创建，其通过调用第三方插件完成网络空间的管理。这些第三方插件性能各异[22]，因此集群的网络性能取决于用户在Kubernetes集群中安装的插件。一些插件以代理的形式进行数据包转发，这会带来较大的网络开销[23]。为了透明地②减小网络开销，我们提出了一种网络感知的调度方法，通过分析底层网络并将可用的且性能更好的网络模式分配给容器以减少网络交互成本及其延迟。

此外，Kubernetes允许开发者开发并部署自定义网络设备插件以支持IB等自定义网络设备，提供更高的带宽以及更低的延迟。因此，网络感知调度策略还会为容器分配性能更好的网络设备，减少网络设备对远程GPU虚拟化性能的影响。

3. 相关工作

本节首先详细介绍了容器云中GPU共享技术的相关工作，接着对这些相关工作进行了详细对比分析并将对比结果汇总至表2，最后介绍了相关工作存在的问题以及本论文的改进措施。

表2 容器云中GPU共享相关工作详细对比

文献编号	GPU共享	细粒度资源分配	远程GPU虚拟化	自适应资源共享策略	网络感知调度方法	兼容性问题
k8s-device-plugin[18]	不支持	不支持	不支持	不支持	不支持	不存在
Deepomatic[24]	支持	不支持	不支持	不支持	不支持	不存在
GPU Sharing Scheduler[25]	支持	不支持	不支持	不支持	不支持	不存在
ConVGPU[26]	支持	有限支持	不支持	不支持	不支持	存在
Gaia GPU[16]	支持	支持	不支持	不支持	不支持	存在
KubeShare[27]	支持	支持	不支持	不支持	不支持	存在
DynamoML[28]	支持	支持	不支持	不支持	不支持	存在
Satzke[29]	支持	支持	不支持	不支持	不支持	存在
Oscar[10]	支持	支持	支持	不支持	不支持	存在
KubeGPU(Our Work)	支持	支持	支持	支持	支持	不存在

一些工作利用GPU直通技术为容器分配GPU资源。k8s-device-plugin[18]利用GPU直通技术为容器分配独占GPU资源，Deepomatic[24]则利用GPU直通技术将一个GPU资源分配给多个容器共享使用。GPU Sharing Scheduler[25]允许用户设置不同容器的资源配额，并在调度层上保证GPU资源不会被超额占用。

API转发技术是实现容器云中GPU共享的另一个重要途径。ConVGPU[26]将一个物理GPU资源拆分成若干个虚拟GPU资源以实现容器间GPU资源共享，并通过API转发技术拦截容器的显存调用以实现容器

间显存隔离。然而，ConVGPU只对容器的显存使用量进行限制，缺少对容器计算资源的管理。因此，一些工作[16, 27-29]通过API转发技术限制容器加载核函数的频率以实现容器间的计算资源隔离。此外，OSCAR[10]使用API转发技术提供了无服务函数访问远程GPU资源的能力。

先前的工作只采取某种特定的虚拟化方案进行容器部署，因此他们难以同时保证容器性能以及系统吞吐量。k8s-device-plugin基于GPU直通技术为容器分配独占GPU资源，这会导致GPU利用率不足以及较低的系统吞吐量[12]。Deepomatic和GPU Sharing Scheduler基于GPU直通技术为容器分配共享GPU资源，这很容易引发共享资源干扰问题，因为GPU直通技术无法提供细粒度的资源分配机制。其他的解决方案基于API转发技术实现容器间GPU共享，然而API转发技术会引来额外的性能开销和兼容性问题[13]。为了保证容器性能以及系统吞吐量，我们提出了自适应共享策略，该策略会根据当前系统可用GPU数量以及容器请求参数动态选择GPU虚拟化进行容器部署。比如，适应共享策略为那些非计算密集型应用分配共享GPU资源，以保证GPU利用率以及系统吞吐量；并为有高性能需求的容器分配独占GPU资源，以确保容器性能。

此外，只有OSCAR实现了Kubernetes上的远程GPU虚拟化。虽然OSCAR为Kubernetes集群提供了访问远程GPU资源的能力，但是该方法并没有解决通信开销问题，并且不支持细粒度资源分配。研究人员通过分析远程GPU虚拟化在三代GPU上的性能表现[21]，发现远程GPU虚拟化在新一代GPU上的性能损耗是旧一代的8到14倍。这是因为GPU每经历一次升级，其计算效率就变得更快，核函数的执行时间也因此变短。这个实验结果表明，必须对远程GPU虚拟化的通信层进行恰当的设计，才能有效减少远程GPU虚拟化的性能开销。因此，我们提出了网络感知的调度方法以减少通信开销。此外，为了减少共享资源竞争对容器性能的影响，论文实现了支持细粒度资源隔离的GPU远程虚拟化技术。

4. 设计和实现

本论文提出了一种新的扩展Kubernetes的GPU管理能力的框架—KubeGPU。KubeGPU由5个核心组件构成，这些组件分别是KubeGPU Scheduler，Device Manager，Resource Pool，Device Agent和Device Resource。图1展示了KubeGPU的整体架构

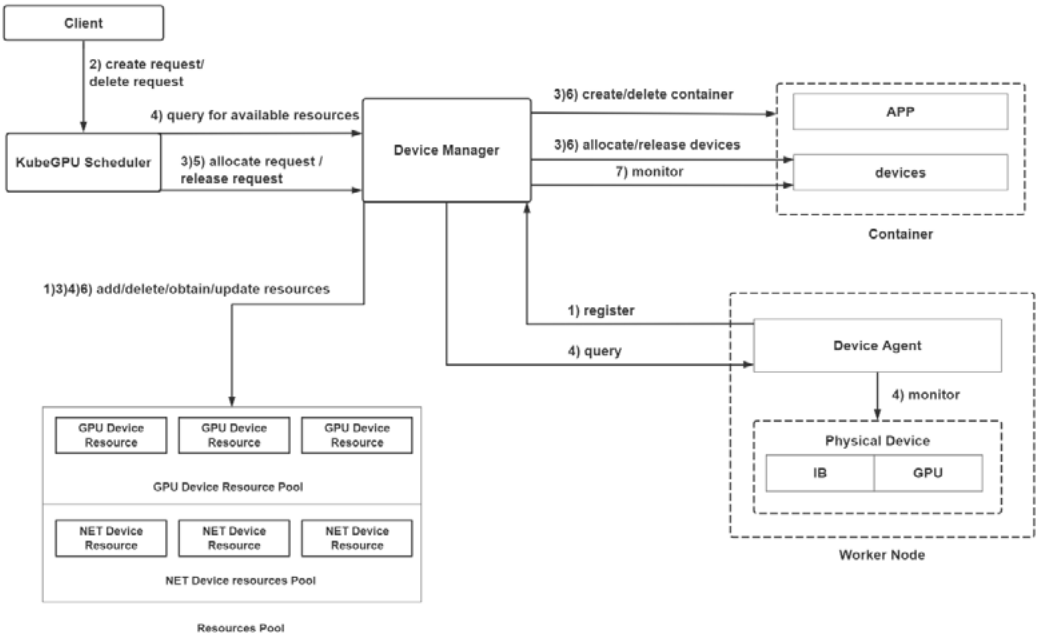


图1 KubeGPU系统架构图

客户端与KubeGPU Scheduler之间通过KubeGPU specification传递信息。列表1提供了一份KubeGPU specification的示例。

如列表1所示，KubeGPU specification包含了5个容器配置参数：gpuCore，gpuMem，virtualAffinity，waitTime和timeoutHandling。KubeGPU Scheduler会根据这些容器配置参数和系统可用资源情况动态选取GPU虚拟化方案进行容器部署。第4.1小节对KubeGPU

specification进行了详细介绍。

```
apiVersion : kubegpu/v1
kind : gpushare
metadata :
   name : gpushare_example
configuration:
   gpuCore: 0.5
   gpuMem: “ 256Mi ”
   virtualAffinity: “ local-shared ”
   waitTime: “ 60s ”
   timeoutHandling: “ default ”
spec :
   containers :
   - name: example_container
     image: nvidia/cuda
     command : [“ nvidia - smi “ , “ - L “ ]
```

列表1 KubeGPU specification示例

下面对KubeGPU的核心组件进行详细介绍：

（1）KubeGPU Scheduler。 我们基于责任链模式[30]设计并实现KubeGPU Scheduler。责任链包含了一系列的处理器，每一个处理器负责处理特定类型的请求，当某个请求无法被处理器处理时，处理器会将该请求传递给下一个处理器进行处理。KubeGPU Scheduler包含了两个处理器：GPU资源处理器和网络资源处理器。KubeGPU Scheduler为客户端提供了一组Restful API。客户端可以通过调用这些Restful API向KubeGPU Scheduler传递KubeGPU specification以实现容器的删除和创建。如果收到客户端发送的删除容器请求，KubeGPU Scheduler会直接将转换成release request请求，并将release request请求发送给Device Manager。如果收到客户端发送的创建容器请求，KubeGPU Scheduler会先向Device Manager查询可用的Device Resource③。然后将可用的GPU Device Resource和容器的配置参数传递给GPU资源处理器。GPU资源处理器根据容器的配置参数以及可用的GPU Device Resource使用自适应资源共享策略动态选取用于容器部署的GPU虚拟化方案。GPU资源处理器完成处理后，会将容器的配置参数和可用的网络资源会传递给网络资源处理器，网络资源处理器通过网络感知调度方法为远程GPU虚拟化选取合适的网络模式和网络设备以减少网络开销。最后，KubeGPU Scheduler通过向Device Manager发送allocate request请求，要求Device Manager创建容器并分配相应资源。

（2）Device Manager。Device Manager根据KubeGPU Scheduler发送的allocate request请求从Resource Pool中获取相应的Device Resource。然后，Device Manager开始创建容器并将Device Resource分配给容器使用，最后Device Manager会更新Resource Pool中的相应Device Resource信息。Device Manager还会时刻监控容器的资源使用情况，以确保容器不会超额使用设备资源。如果Device Manager收到了KubeGPU Scheduler发送的release request请求，它会删除相应容器并且释放容器占用的设备资源，最后更新Resource Pool中的相应Device Resource信息。Device Manager还支持弹性计算资源伸缩机制，该机制在有空闲GPU计算资源的情况下，允许容器超额使用空闲的计算资源。

（3）Device Agent。Device Agent作用在于向Device Manager通告所有可用的Device Resource。Device Agent运行在每一个节点上，收集节点上所有可用物理设备信息并将这些信息封装成为Device Resource。Device Agent会在自己启动后向Device Manager注册自己。而Device Manager会向Device Agent查询可用的Device Resource。如果某一个Device Resource变的不可用了，Device Manager会从Resource Pool删除相应的Device Resource。当出现一个新的Device Resource，Device Manager也会将其添加进Resource Pool中。

（4）Resource Pool。 Resource Pool存储了集群中所有的Device Resource。目前集群中的Device Resource分为两类即GPU Device Resource和Network Device Resource。GPU Device Resource是物理GPU设备的一个抽象，而Network Device Resource是网络设备的一个抽象。Resource Pool为Device Manager提供了四个API以管理集群中的Device Resource：（1）add。用于向Resource Pool添加一个新的Device Resource。（2）delete。用于从Resource Pool中删除一个Device Resource。（3）obtain。获取一个指定的Device Resource。（4）update。更新Resource Pool中某个Device Resource的属性，比如剩余容量。

（5）Device Resource。Device Resource是物理设备的一个逻辑抽象。Device Resource包含了物理设备的一些信息，比如剩余容量，物理位置以及设备类型等。

综上，KubeGPU的工作流程汇总如下：

（1）Device Agent向Device Manager注册自己，并通告所有可用的Device Resource。Device Manager将新的可用Device Resource添加到Resource Pool中。

（2）KubeGPU Scheduler接收来自客户端的请求.

（3）如果客户端请求删除容器，KubeGPU Scheduler会向Device Manager发送release request请求。随后，Device Manager删除相应的容器，释放相应的设备资源并更新相应Device Resource的剩余容量.

（4）如果客户端请求创建容器，KubeGPU Scheduler会向Device Manager中查询可用的Device Resource，并且根据可用的Device Resource以及容器配置参数指定调度策略。注意，Device Manager会

进一步从Device Agent查询可用的Device Resource，而Device Agent根据收集到的设备监控数据向Device Manager实时地返回可用的Device Resource。并且Device Manager会根据返回的Device Resource更新Resource Pool。

（5）KubeGPU Scheduler通过向Device Manager发送allocate request请求，让Device Manager创建容器

（6）Device Manager收到allocate request请求后，开始进行容器创建，为容器分配设备资源，并更新Resource Pool中的相应Device Resource信息。

（7）Device Manager时刻监控容器的资源使用情况，以确保容器不会超额使用设备资源。

4.1 KubeGPU Specification

用户在Kubernetes上创建资源时，必须为Kubernetes提供描述该资源的基本信息，即object specification。因此，KubeGPU为用户提供了KubeGPU Specification用于创建容器。如列表1所示，KubeGPU Specification的Configuration字段包含了五个容器配置参数，用户可以通过设置这些参数指定容器的资源需求，调度约束以及超时处理策略。

容器的资源需求包括对GPU的计算资源需求和显存资源需求。GPU计算资源采用分时共享策略进行资源分配，而GPU显存资源采用显存隔离策略进行资源分配。比如gpuCore=0.5意味着容器一秒内至少可以占用0.5秒的GPU时间。gpuMem=“256Mi”意味着容器可以申请分配256Mib的独占显存空间。GPU计算资源对容器性能有较大影响，并且容器完成任务时会迅速释放计算资源。然而GPU显存资源对容器性能影响较小，并且许多容器不会在完成任务时释放显存资源，只有当容器销毁后显存资源才会释放[16]。因此我们只针对计算资源进行弹性伸缩，而对显存资源采取硬性分配措施。

容器的调度约束包括virtualAffinity和waitTime。virtualAffinity强制KubeGPU Scheduler使用特定的GPU虚拟化进行容器部署。waitTime用于指定客户端等待调度成功的最长等待时间。如果KubeGPU无法在waitTime期间内完成容器调度，则会执行timeoutHandling指定的超时处理动作。

4.2 弹性计算资源分配

计算资源不仅会影响容器的性能，甚至还会影响容器的运行状态。然而，用户难以估计容器实际运行过程中需要使用多少的计算资源。因此，我们采取弹性计算资源分配机制保证使用共享GPU资源的容器性能。弹性计算资源分配机制运行容器在存在空闲的GPU计算资源的情况下，超额使用GPU计算资源。

当KubeGPU Scheduler采取API Forwarding技术部署容器时，KubeGPU Scheduler会让Device Manager将一个自定义的CUDA库注入到容器中，以拦截容器的GPU调用。当容器通过该自定义CUDA库发射一个核函数时，自定义CUDA库会检查该容器是否含有一个有效token。如果容器没有有效的token，自定义CUDA库会向Device Manager发送一个请求以申请token。另一方面，Device Manager会将所有申请token的请求放入一个队列中。

只有拥有有效token的容器能够发射核函数，因此弹性计算资源分配机制可以通过控制token发放来管理容器的计算资源使用。

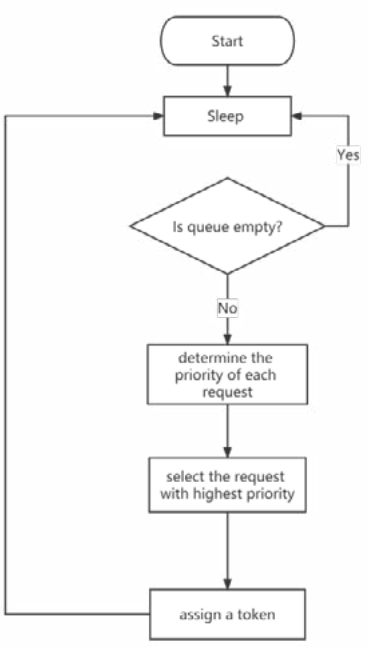


图2 弹性计算资源分配机制的整体流程

图2介绍了弹性计算资源分配机制的整体流程：

（1）为了能够保证每隔一个固定时间分发一个token，弹性计算资源分配机制会在循环起始处休眠一个固定长度的时间。这个时间长度可以通过参数设置，默认值为100毫秒。

（2）弹性计算资源分配机制检查队列中是否有待处理的请求

（3）弹性计算资源分配机制根据两个指标决定队列中请求的优先级：（1）GU_BIAS。GU_BIAS=容器申请的计算资源量-实际计算资源使用量。（2）GU。GU=最大计算资源使用量（100%）-容器实际的计算资源量。GU值越大，表明容器实际使用的资源资源越小。

（4）弹性计算资源分配机制根据请求的优先级进行token分配而不是采取先入先出的分配机制，因此能够更为公平地分配计算资源。为了确定两个请

求中哪个请求的优先级更高，弹性计算资源分配机制先比较这两个请求的GU_BLAS值，如果两请求的GU_BLAS值不同，GU_BLAS值高者优先级更高。如果GU_BLAS值相同，则比较两个请求的GU值，GU值高者优先级更高。

（5）最后，弹性计算资源分配机制为优先级最高的请求分配一个token。

弹性计算资源分配机制在休眠和分配token阶段的时间复杂度为O(1)。在决定请求优先级阶段的时间复杂度为O(N)。因此整体的时间复杂度为O(N)。此外，弹性计算资源分配机制需要一个长度为N的队列，因此空间复杂度为O(N)。

4.3 自适应资源共享策略

论文提出了自适应资源共享策略(下称算法1)，算法1使得KubeGPU能够根据系统可用资源情况以及容器配置参数动态选择GPU虚拟化进行容器部署。

Algorithm 1 Adaptive resource sharing strategy

Input:
C: container configuration
R_G: a list GPU Device Resources available in resource pool
Output:
G: a lists of GPU allocation solutions
/* branch 1: select GPU virtualization based on virtualAffinity constrain*/
1: **if** $C.virtualAffinity \neq null$ **then**
2: **do**
3: **if** $C.virtualAffinity ==$ "$exclusive$" **then**
4: /* get exclusive GPU allocation solution*/
5: $G = get\_exclusive\_GPU\_solution(R_G, C.waitTime, C.gpuMem)$
6: **if** !$G.isEmpty$ **or** $C.timeoutHandling ==$ "$drop$" **then**
7: **return** G
8: **end if**
9: **else**
10: /* get shared GPU allocation solution*/
11: $G = get\_shared\_GPU\_solutions(R_G, C.waitTime, C.gpuMem)$
12: **if** !$G.isEmpty$ **or** $C.timeoutHandling ==$ "$drop$" **then**
13: **return** G
14: **end if**
15: **end if**
16: **while** $C.timeoutHandling ==$ "$retry$"
17: **end if**
18:
/* branch 2: select GPU virtualization based on adaptive resource sharing strategy*/
19: **do**
20: **if** $C.gpuCore \geq EXCLUSIVE\_THRESHOLD$ **then**
21: /* get exclusive GPU allocation solution*/
22: $G = get\_exclusive\_GPU\_solution(R_G, C.waitTime, C.gpuMem)$
23: **else**
24: /* get shared GPU allocation solution*/
25: $G = get\_shared\_GPU\_solutions(R_G, C.waitTime, C.gpuMem)$
26: **end if**
27: **while** $G.isEmpty$ **and** $C.timeoutHandling ==$ "$retry$"
28: **return** G

算法1的输入是C和RG。C是容器配置参数，RG是Resource Pool中可用的GPU Device Resource。算法1的输出是一个含有若干个GPU分配方案的数组。每个GPU分配方案包含GPU Device Resource，容器资源需求，部署容器的GPU虚拟化方案和Network Device Resource信息。

图3展示了自适应资源共享策略的整体流程图。

算法1包含两个分支。在第一个分支中(算法第1~17行)，算法1根据容器配置参数C.virtualAffinity选择参数指定的GPU虚拟化方案部署容器。在第二个分支(算法第18~28行)，算法1根据容器的计算资源需求（对应容器配置参数C.gpuCore）动态选择GPU虚拟化方案部署容器。首先，算法1检查用户是否设置参数C.virtualAffinity(算法第1行)。如果用户设置了参数C.virtualAffinity，算法1将进入第一个分支，并执行如下步骤：

（1）如果参数C.virtualAffinity的值为exclusive，算法1将尝试采取GPU直通技术进行容器部署，并为容器分配独占GPU资源（算法第3行）

（2）算法1通过方法get_exclusive_GPU_solution生成独占GPU分配方案（算法第5行）。生成的独占GPU分配方案将被封装到allocate request请求中。Device Manger会根据该独占GPU分配方案为容器分配独占GPU资源。方法get_exclusive_GPU_solution含有三个参数RG，C.waitTime和C.gpuMem。get_exclusive_GPU_solution在生成独占GPU分配方案时首先检查RG中是否存在空闲的GPU Device Resource[4]。如果存在，则利用最佳适配算法从RG中选取一个能够满足容器的显存需求（C.gpuMem）的最小显存的空闲的GPU Device Resource，并将这个GPU Device Resource放入结果G中。如果get_exclusive_GPU_solution无法在C.waitTime时间内找到一个满足条件的GPU Device Resource，则返回一个不含任何结果的G。

（3）算法1检查G是否是空的或者C.timeoutHandling的值是否是drop（算法第6行）。如果G不是空的意味着算法1成功生成了一个独占GPU分配方案。算法1将这个方案返回给KubeGPU Scheduler。如果G是空的，意味着没能成功生成独占GPU分配方案。这种情况下，如果C.timeoutHandling值为drop，那么算法1会返回一个空的G给KubeGPU Scheduler(算法第7行)。这种情况下，KubeGPU Scheduler会丢弃此容器创建请求，不再处理。

（4）如果参数C.virtualAffinity的值不为exclusive，算法1会尝试使用API转发技进行容器部署，并为容器分配共享GPU资源（算法第9行）。

（5）算法1调用方法get_shared_GPU_solutions去生成若干个共享GPU分配方案（算法第11行）。方法get_shared_GPU_solutions含有三个参数RG，C.waitTime和C.gpuMem。方法get_shared_GPU_solutions在生成共享GPU分配方案过程中首先检查RG中是否存在满足容器显存需求的GPU Device Resource。如果存在，则将这些GPU Device Resource添加到结果G中。如果get_shared_GPU_solution无法在C.waitTime时间内找到满足条件的GPU Device Resource，则返回一个不含任何结果的G。

（6）算法1检查G是否是空的或者C.timeoutHandling的值是否是drop（算法第12~14行）。如果G为空且C.timeoutHandling的值是

drop，KubeGPU Scheduler会丢弃此容器创建请求，不再处理。

（7）如果C.timeoutHandling的值为retry（算法第16行），算法1跳转回步骤1，重新尝试获取一个GPU分配方案。

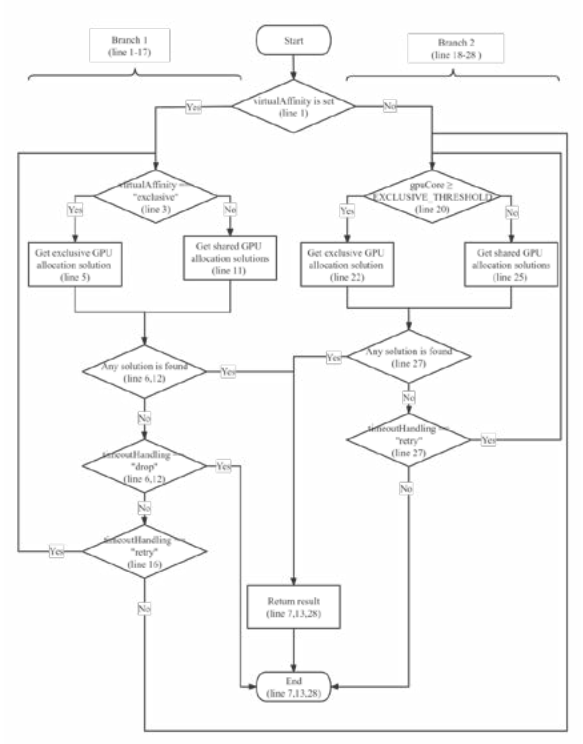


图3 自适应共享策略流程图

如果用户没有设置C.virtualAffinity（算法第1行），算法1将进入第二个分支。除此之外，如果算法1在第一个分支中无法找到用户指定的GPU分配方案，并且C.timeoutHandling不是drop和retry，算法1也会进入第二个分支去查找GPU分配方案。第二个分支包含如下步骤：

（1）如果C.gpuCore不小于EXCLUSIVE_THRESHOLD，算法1将尝试采取GPU直通技术部署容器，并为容器分配独占GPU资源。EXCLUSIVE_THRESHOLD是一个可设置的参数，默认值为0.8（算法第20行）

（2）本步骤（算法第22行）和分支1的第2步一样（算法第5行）

（3）如果C.gpuCore小于EXCLUSIVE_THRESHOLD，算法1会尝试使用API转发技术部署容器，并为容器分配共享GPU资源（算法第23行）

（4）本步骤（算法第25行）和分支1的第5步一样（算法第11行）

（5）算法1检查G是否为空或者C.timeoutHandling的值是否为retry（算法第27行）。如果G是空的，意味着没有找到合适的GPU分配方案。这种情况下，如果C.timeoutHandling是retry，算法1跳转回分支2的步骤1，重新尝试获取一个GPU分配方案。

（6）算法1返回G给KubeGPU Scheduler（算法第28行）。注意，返回的G可能为空，如果G是空，KubeGPU Scheduler会丢弃此请求，不再处理。

算法1中方法get_exclusive_GPU_solution和get_shared_GPU_solutions的时间复杂度均为O(N)。此外，算法1还需要一个列表存放生成的GPU分配方案，故而算法1的空间复杂度为O(N)。

4.4 网络感知调度策略

KubeGPU Scheduler利用网络资源处理器为远程GPU虚拟化提供合适的网络设备和网络模式。为此，论文提出了网络感知调度策略（下称算法2）用于减少远程GPU虚拟化的网络开销。

Algorithm 2 Network-aware scheduling apporach

Input:
C: container configuration
G: a lists of GPU allocation solutions
NR: a list Net Device Resources available in resource pool
Output:
solution: a GPU allocation solution

```
1: /*network_mode = 0 means virtual network mode, and network_mode =
   1 means host network mode*/
2: network_mode = 0
3: G = ascending_order(G)
4: if disable_remote_GPU(C, SHARED_THRESHOLD) then
5:     solution = G[0]
6:     return solution
7: end if
8: for g ∈ G do
9:     if IB_available(g.node, NR) then
10:        ib = NR.get_IB_by_node(g.node)
11:        peer = IB.find_peer(ib.node)
12:        network_mode = 1;
13:        solution = g
14:        /* information of network resource is encapsulated in solution by
       set_network_resource*/
15:        set_network_resource(ib, peer, network_mode, solution)
16:        break
17:    end if
18:    if hostnetwork_available(g.node, NR) then
19:        if network_mode == 0 then
20:            host_device = get_host_device(g.node, NR)
21:            solution = g;
22:            network_mode = 1;
23:            set_network_resource(host_device, network_mode, solution)
24:        end if
25:    else
26:        if solution == null then
27:            solution = g
28:            network_mode = 0;
29:            set_network_resource(network_mode, solution)
30:        end if
31:    end if
32: end for
33: return solution
```

算法2的输入是C,G和NR，其中C是容器配置参数，G是一组GPU分配方案，NR可用的Network Device Resource。网络感知调度策略的输入是一个GPU分配方案，该方案会被发送给Device Manager用于创建容器和分配相应资源。注意，网络感知调度策略的输入G来自算法1的输出。图4展示了网络感知调度策略的整体流程图。

网络感知调度策略的基本思想是通过分析底层网络，尽可能地为使用远程GPU虚拟化的容器分配性能更好的网络资源。网络感知调度策略流程如下：

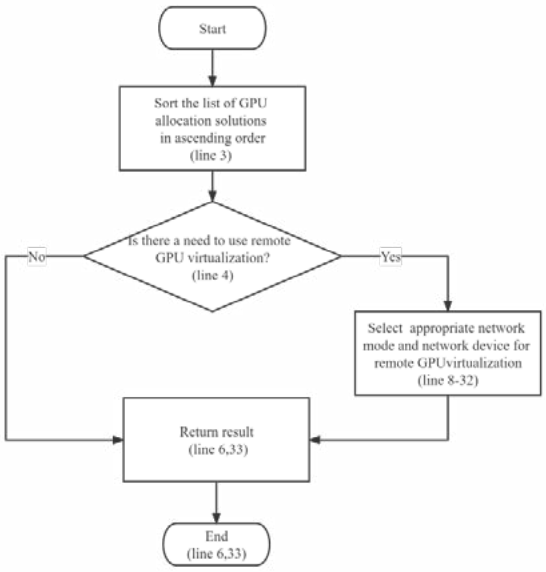


图3 网络感知调度策略流程图

（1）按照每个GPU分配方案的可用显存和可用计算资源对G进行排序（算法第3行）。

（2）算法2调用方法disable_remote_GPU检查是否需要使用远程GPU虚拟化部署容器（算法第4行）。disable_remote_GPU有两个参数C和SHARED_THRESHOLD。SHARED_THRESHOLD是一个可设置的参数，默认值为0.2。用户可以通过将C.virtualAffinity设置为exclusive或local-share阻止网络感知调度策略使用远程GPU虚拟化。此外，如果C.gpuCore大于SHARED_THRESHOLD并且C.virtualAffinity的值不为remote-share，网络感知调度策略也不会采用远程GPU虚拟化。

（3）如果不使用远程GPU虚拟化部署容器，算法2会将G中第一个元素返回给KubeGPU Scheduler（算法第5-6行）。此GPU分配方案含有一个满足容器显存需求、且显存容量最小的GPU Device Resource。之后，KubeGPU Scheduler将该GPU分配方案封装到allocate request请求中，并发送给Device Manger进行容器创建。

（4）算法2通过选取合适的网络模式和网络资源减少网络开销，优化远程GPU虚拟化技术（算法第8~32行）。算法2按照如下的优先级进行网络模式和网络设备的选取，（1）IB网络模式（算法第9~17行）（2）宿主机网络模式（算法第18~24行）（3）虚拟机网络模式（算法第26~29行）。当GPU节点含有IB资源，且存在一个非GPU节点通过IB网络与该GPU节点相连时（算法第9行），算法2选取IB网络模式（算法第10~15行）。如果GPU节点含有可用的宿主网络资源（算法第18行），算法2选择宿主网络模式（算法第19~24行）。当没有充足的IB资源和宿主网络资源时，算法2才会选择虚拟网络资源（算法第26~30行）。

（5）算法2将包含有Network Device Resource的GPU分配方案返回给KubeGPU Scheduler（算法第30行）。KubeGPU Schedule根据该GPU分配方案可以为容器分配网络设备并设置相应的网络模式。

（6）算法2需要对G进行排序，排序的时间复杂度为O(NlogN)。方法disable_remote_GPU的时间复杂度为O(1)，选取合适网络模式和网络设备的时间复杂度为O(N)。因此，算法2的时间复杂度为O(NlogN)。算法2需要若干个变量保存选取的网络模式和网络设备，因此算法2的空间复杂度为O(1)。

4.5 远程GPU虚拟化的细粒度资源分配

共享资源干扰通常会影响远程GPU虚拟化的性能。然而目前面向Kubernetes的远程GPU虚拟化的研究仍处于初级阶段，资源隔离问题并未得到解决。为了解决该问题，我们必须保证容器不能使用超过自己申请的资源配额。

大多数远程GPU虚拟化技术采用客户端-服务器的架构，客户端通过修改后的GPU API提交任务，该GPU API会将任务请求转发给服务器。服务器接受请求后，代替客户端完成实际的GPU操作。为了在远程GPU虚拟化技术上实现细粒度资源分配，我们设计了一个前端模块和一个后端模块。前端模块安装在每一个需要使用GPU资源的客户容器中，后端模块安装在每一个GPU服务器上。

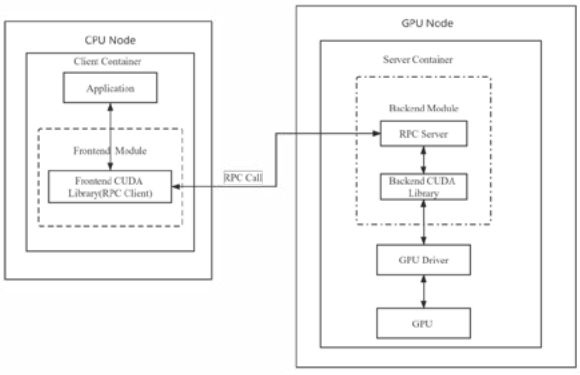


图5 用于实现远程GPU虚拟化细粒度资源分配的架构图

图5展示了用于实现远程GPU虚拟化细粒度资源分配的架构。如图5所示，前端模块包含了一个Frontend CUDA library。Device Manager会将这个前端模块挂载到使用远程GPU资源的容器（客户容器）中。后端模块由RPC Server和Backend CUDA library组成。Device Manager会为每个使用远程GPU资源的客户容器在GPU服务器上创建对应的服务容器，并将这个后端模块挂载到服务容器中。Frontend CUDA library和Backend CUDA library都是自定义CUDA库，它们复写了原始CUDA库的相关API[32]。

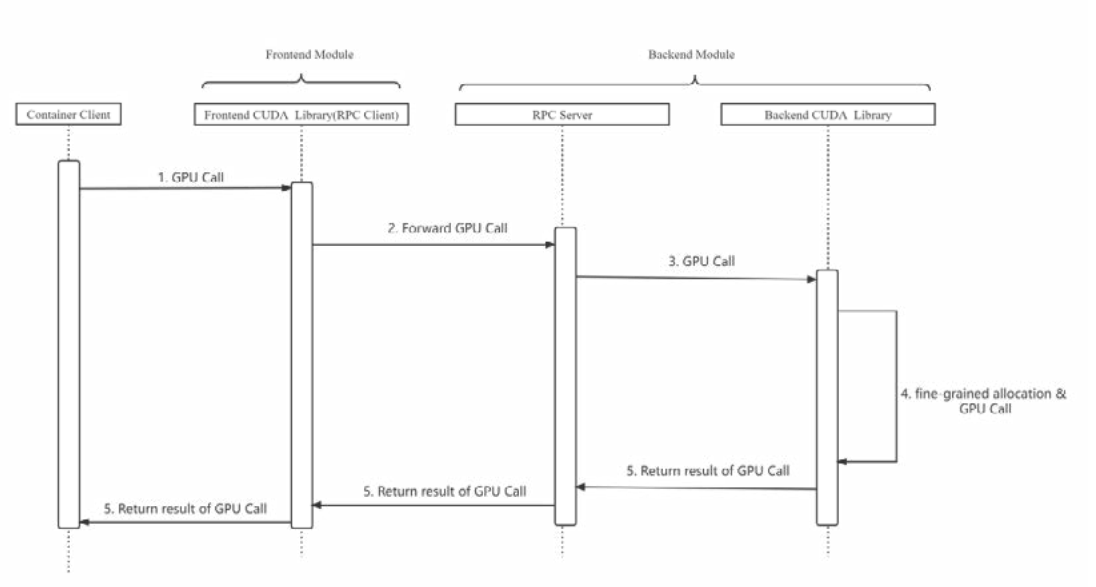


图6 远程GPU虚拟化细粒度资源分配的工作流程

图6展示了细粒度资源分配的整个工作流程：

当一个客户容器进行GPU调用时，Frontend CUDA Library通过LD_LIBRARY_PATH 机制[33]拦截客户容器的GPU调用。

（1）Frontend CUDA Library通过RPC机制将GPU调用转发给RPC Server。

（2）RPC Server代表客户容器发起GPU调用。RPC Server发起的GPU调用又会被Backend CUDA library利用LD_LIBRARY_PATH机制拦截。

（3）Backend CUDA Library发起GPU调用。Backend CUDA Library采用弹性计算资源分配机制分配GPU调用过程中使用到的计算资源。Backend CUDA Library对GPU调用过程中使用到的显存资源采取硬限制。这意味着该GPU调用不能超额使用显存资源。

（4）Backend CUDA Library将结果返回给客户容器。

5. 实验与分析

为了验证KubeGPU的正确性和有效性，我们组件了一个由异构节点组成的Kubernetes集群：

Node1是一个KVM虚拟机，其包含16核CPU，16GB内存，一个Mellanox Technologies MT27700 IB网卡以及一个Tesla M60 GPU。

Node2是一个物理节点，拥有32个核心CPU，187GB内存以及一个Mellanox Technologies MT27700 IB网卡。

上述节点的详细信息请参见表3。此外，在本次实验中，参数EXCLUSIVE_THRESHOLD和SHARED_THRESHOLD均使用默认值0.8和0.2。

表3 Kubernetes集群配置信息

节点名	节点类型	内存大小（GB）	CPU	GPU	Infniband	操作系统	CUDA版本	Kubernetes版本
Node1	KVM 虚拟机	16	16个虚拟核心的Intel Xen 6130	NVIDIA Tesla M60 8GB	Mellanox Technologies MT27700 Family	CentOS 7.9	11.2	1.18.2
Node2	物理节点	192	2个Intel Xeon Gold 6130	无	Mellanox Technologies MT27700 Family	Ubuntu 18.04.5	无	1.18.2

5.1 性能开销

为了确保KubeGPU没有引入过多的额外性能开销，我们评估了GPU容器运行在不同GPU管理框架上的性能表现。不同类型的GPU容器对于GPU资源有不同需求，这导致GPU资源调度策略会对不同类型的GPU容器分配产生不同的性能影响。因此，本实验选取了两个具有代表性的常用的GPU容器，一个是GROMACS，一个是Pytorch MNIST。GROMACS是一个利用GPU进行分子动力学分析的容器应用，Pytorch MNIST是一个进行手写识别的深度学习应用。在

本次实验中，GROMACS请求的计算资源配额为0.9（gpuCore=0.9），Pytorch MNIS请求的计算资源配额为0.5（gpuCore=0.5）。本次实验结果如表4所示。

表4中的Native_time表示容器运行在裸金属上的执行时间。Container_time表示GPU容器运行任务的时间。我们根据公式（1），计算容器性能开销：

如表4所示，GaiaGPU和KubeShare引入了极大的性能开销，这是因为它们使用API转发技术部署容器，API转发技术受限于自身的额外开销和兼容性问题[9]®。虽然KubeGPU和k8s-device-plugin都没有引入过多的性能开销，但是k8s-device-plugin只采取GPU直通技术部署容器，并为容器分配独占GPU资源。不同于k8s-device-plugin，因为GROMACS申请的计算资源配额大于EXCLUSIVE_THRESHOLD，所以KubeGPU采取GPU直通技术为GROMACS容器分配独占GPU资源。另一方面，KubeGPU采取API转发技术为Pytorch MNIST容器分配共享GPU资源，并使用弹性计算资源分配机制保证容器运行性能。

表4 不同GPU管理框架下的容器性能开销

容器应用	容器计算资源使用率	GPU管理框架	Native_time(秒)	Container_time(秒)	性能开销(%)
Gromacs	0.85	k8s-device-plugin	857	877	2.33
Gromacs	0.85	GaiaGPU	857	958	11.78
Gromacs	0.85	Kubeshare	857	1700	98.36
Gromacs	0.85	KubeGPU(our work)	857	864	0.81
Pytorch MNIST	0.5	k8s-device-plugin	144	145	0.69
Pytorch MNIST	0.5	GaiaGPU	144	244	69.44
Pytorch MNIST	0.5	Kubeshare	144	150	4.16
Pytorch MNIST	0.5	KubeGPU(our work)	144	147	2.08

5.2 系统吞吐量

本节，我们通过实验验证KubeGPU是否能够实现较高的系统吞吐量。本次实验包含三个子实验，这些子实验分别提交10个，20个和30个容器任务，并测量在这些实验条件下KubeGPU及其同类框架的系统吞吐量。此外，三个子实验提交容器的频率均为5秒一次，换句话说，我们每5秒提交2个容器（1个GPU-BLAST容器，1个HOOMD-blue容器）。在本次实验中，每一个GPU-BLAST容器会请求0.2个GPU计算资源（gpuCore=0.2），每一个HOOMD-blue容器会请求0.8个计算资源（gpuCore=0.8）。本次实验结果如图7所示。

如图7所示，k8s-device-plugin的系统吞吐量最差，这是因为它只为容器分配独占GPU资源，极大影响了系统吞吐量。KubeShare 和GaiaGPU的系统吞吐量略低于KubeGPU。Kubeshare使用API转发技术在容器间分配共享GPU资源，然而Kubeshare实现的API转发技术与GPU-BLAST容器存在兼容性问题，导致GPU-BLAST容器的GPU利用率存在较大波动，对容器性能造成影响，降低系统吞吐量。GaiaGPU不存在兼容性问题，但是GaiaGPU只采取API转发技术在容器间共享GPU资源。然而，与一个需要大量计算资源的GPU容器共享同一块GPU时，极容易导致容器间发生资源争抢，影响容器性能，降低系统吞吐量。与KubeGPU和GaiaGPU不同，KubeGPU会为HOOMD-blue容器分配独占GPU资源以避免因GPU资源争抢而导致容器性能损耗，进而提升系统吞吐量。此外，KubeGPU会在GPU-BLAST容器间分配共享GPU资源，增加GPU-BLAST容器的并行数量，提升系统吞吐量。最终,相较于其他GPU管理框架，KubeGPU实现了至少5%的系统吞吐量的提升。

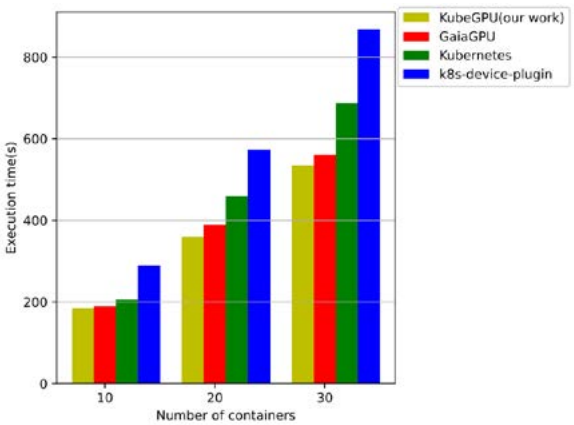


图7 不同GPU管理框架的系统吞吐量。图中x轴是所需执行容器的数量，y轴是执行相应数量的容器所需的时间

此外，根据5.1小节和5.2小节的实验结果，我们可以发现只有KubeGPU可以同时满足容器性能和系

统吞吐量。这是因为KubeGPU采取动态资源共享策略动态选择GPU虚拟化技术部署容器，而其他框架只能采取单一的GPU虚拟化技术部署容器。

5.3 调度时间

虽然KubeGPU拥有许多优势，我们还需要确保KubeGPU不会花费过多的时间进行容器的调度和创建。因此，我们进行了一个对比实验，比较了KubeGPU和其他框架的容器调度时间。在本次实验中，调度时间指的是接受到容器创建申请到容器成功创建所需花费的时间。图8展示了调度不同数量的容器所需的调度时间。

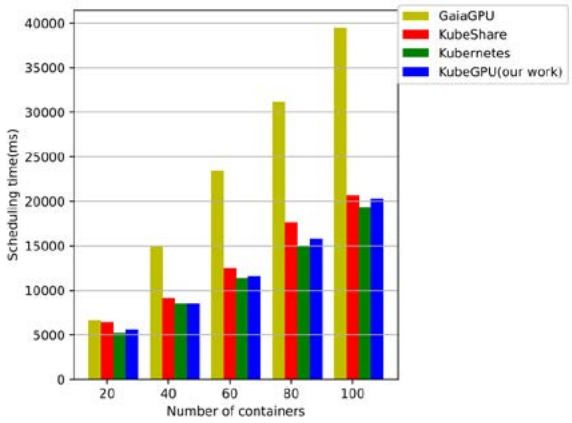


图8 不同GPU管理框架的容器调度时间。图中x轴是所需调度容器的数量，y轴是调度相应数量的容器所需的时间

原生的Kubernetes调度器拥有最短的调度时间，因为该调度器不需要考虑GPU资源调度问题。GaiaGPU拥有最长的调度时间，因为GaiaGPU基于一个树形拓扑进行GPU资源调度，维护树形拓扑是非常耗时的操作。KubeShare的调度时间略长于KubeGPU。这是因为Kubeshare引入了GPU标签调度策略，该策略会引入较大的时间开销[38]。根据第4.3小节和第4.4小节描述，KubeGPU的时间复杂度是O（NlogN），并且KubeGPU使用多线程技术对调度过程进行优化，因此KubeGPU能够快速完成调度决策。

5.4 弹性计算资源分配机制

为了验证弹性计算资源分配机制的有效性，本实验将两个Tensorflow容器（下称容器A和容器B）运行在同一个GPU上。在t=0秒时，容器A开始运行（计算资源请求配额为0.5）。在t=65秒时，容器B开始运行（计算资源请求配额为0.5）。实验结果如图9所示。图中，x轴表示容器运行时间，y轴表示时刻t时容器的GPU利用率。

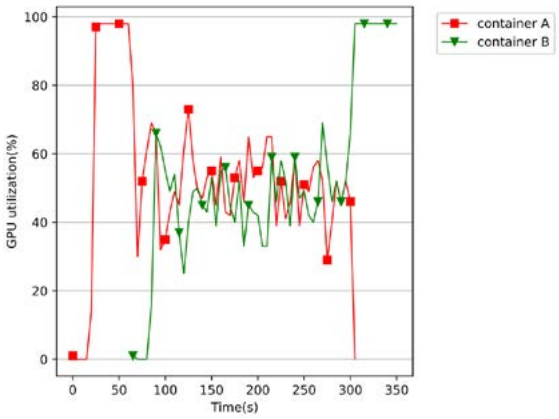


图9 两个Tensorflow容器GPU资源利用率变化情况

如图9所示，一开始时整个GPU上只有容器A在运行，因此它可以使用整个GPU资源。在t=65秒时，容器B开始在同一个GPU上运行，KubeGPU将GPU计算资源均分给容器A和容器B。因此从t=90秒到t=300秒，容器A和容器B分别占用0.5个GPU计算资源。t=300秒时，容器A被中断运行，容器B随之迅速占用整个GPU的计算资源。上述实验结果表明KubeGPU可以允许容器充分使用空闲的GPU资源，并在必要时回收过度分配的计算资源并分配给其他容器使用。

本节还进行了另一个实验，用于验证弹性计算资源分配策略对于容器性能的影响。实验结果如图10所示。

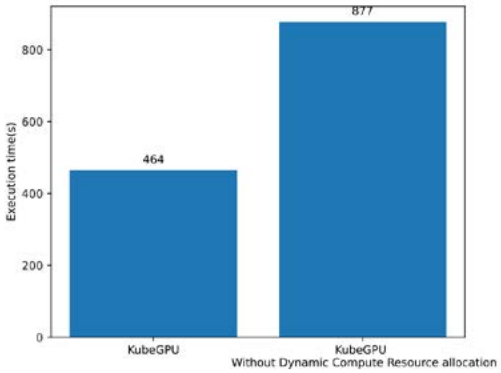


图10 弹性计算资源分配机制对容器性能的影响

实验比较了同一个GPU容器在使用弹性计算资源分配策略和不使用弹性计算资源分配策略分别所需的执行时间。从图10中可以看出，相较于不使用弹性计算资源分配策略，使用弹性计算资源分配机制的容器执行时间减少了47%。因此，弹性计算资源分配策略能够有效提升容器性能。

5.5 远程GPU虚拟化的显存隔离

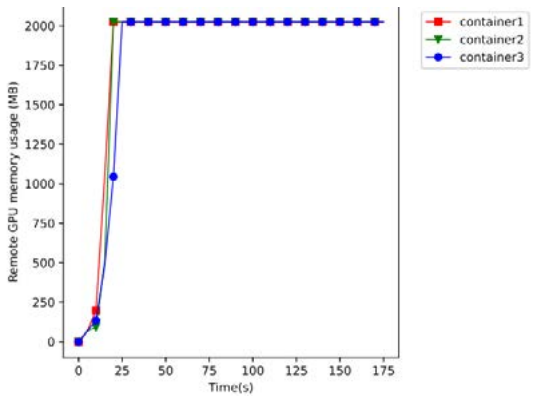
KubeGPU需确保使用远程GPU资源的容器不能占用超过其申请配额的显存资源。另一方面KubeGPU利用弹性计算资源分配机制管理远程GPU的计算资

源。论文在5.4小节已经验证了弹性计算资源分配机制的正确性和有效性，本小节主要目的是验证KubeGPU能够实现远程GPU显存资源隔离。

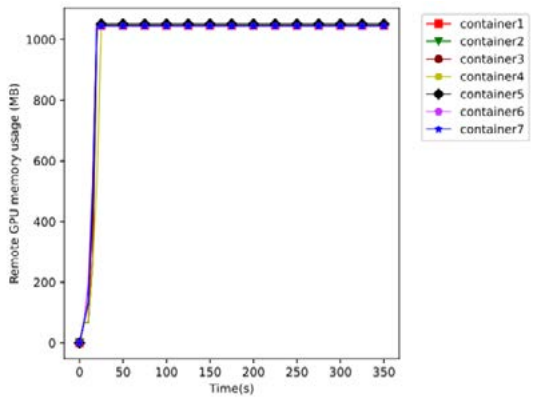
为了证明KubeGPU可以实现容器间的远程GPU显存隔离，我们进行了两次实验，分别将3个Tensorflow[39]容器和7个Tensorflow容器部署到同一个远程GPU资源上，并测量容器的GPU内存使用量。本实验的Tensorflow容器会在CIFAR100数据集上执行模型验证任务。表5展示了实验中容器的远程GPU资源配置情况，图11展示了测试结果。

表5 容器的远程GPU显存资源配置情况

容器数量	每个容器申请的远程GPU显存配额(MB)
3	2048
7	1024



(a)



(b)

图11 弹性计算资源分配机制对容器性能的影响

在图11中，x轴表示容器运行时间，y轴表示容器占用的GPU显存大小。如图11所示，远程GPU内存在短时间内就达到了所请求的值，因为tensorflow框架会在启动时一次性请求全部所需显存。图11(a)表明3个容器同时使用一个远程GPU资源时，这些容器的远程GPU内存使用量均为2024MB。如图11(b)所示，当一个远程GPU被7个容器同时共享时，这些容器的远程GPU资源使用量在执行过程中未存在明显波动，并且不同容器间远程GPU使用量的差距小于1%。本实验结果表明KubeGPU可以有效限制容器的远程GPU显存使用，避免容器超量使用远程GPU显存。

5.6 远程GPU虚拟化的通信开销

本小节通过实验验证网络感知调度策略能够减少远程GPU虚拟化的通信开销。

我们测试了Tensorflow容器运行在不同共享模式下所需的执行时间，实验结果如图12所示。在本次实验中我们选取了四种共享模式，包括Shared Local，Remote Shared(IB)，Remote Shared(host) and Remote Shared(virtual network)。Shared Local表示容器使用本地的共享GPU资源。Remote Shared表示容器使用远程GPU资源。此外，为了验证网络感知调度策略的有效性，我们测试了在使用不同网络资源（IB，host和virtual network）情况下，使用远程GPU资源的容器的执行时间。

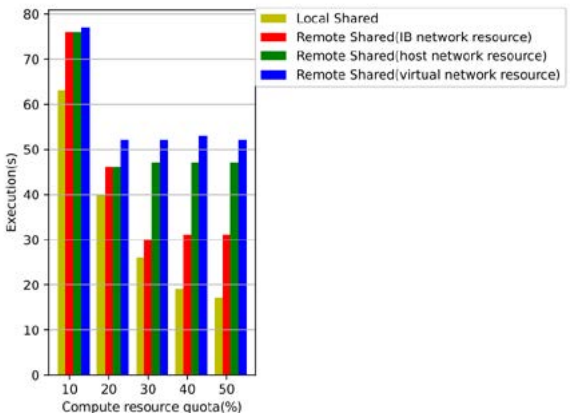


图12 Tensorflow容器在不同共享模式下的性能表现。图中x轴代表容器的计算资源配额，这里的计算资源配额是容器可以使用的最大计算资源数量，y轴表示容器执行任务所需的时间

如图12所示，随着容器计算资源配额的不断增大，Local Shared模式下的容器执行任务时间不断缩短。当GPU容器能够执行速度增快，核函数执行时间随之变短，对于数据IO的要求会不断增大，这会给数据交互层带来更大的压力，在这种情况下，数据交互层通常会引入更多的通信开销，并成为容器性能的瓶颈。当计算资源配额到达20%时，使用虚拟网络和宿主网络的容器遭遇性能瓶颈并达到最大性能，当计算配额到达30%时，使用IB的容器遭遇性能瓶颈并达到了最大性能。与Remote Shared不同的是，Local Shared模式下的容器能够使用CPU-GPU间带宽，CPU-GPU间带宽远远大于网络带宽[14]，缓解了数据IO造成的性能瓶颈。

我们从实验结果中可以发现，相较于virtual network，host和IB分别提升了1.24倍和1.77倍的最大容器性能。这是因为它们能够提供更充足的IO资

源，减小通信带来的开销，缓解了IO造成的性能瓶颈。因此，选取合适网络资源可以减小通信带来的开销并缓解性能瓶颈，提高容器性能。网络感知调度策略相较于先前只使用虚拟网络的方案[10]更能够更有效提高使用远程GPU资源的容器性能。

值得注意的是，虽然远程GPU共享相较于本地GPU共享存在一定性能差距，但是远程GPU虚拟化可以使得非GPU节点使用GPU节点上的GPU资源进而减少集群GPU安装数量以及能源消耗[11]。之前一些工作[11, 40]也使用远程GPU虚拟化为轻量任务和低功耗系统提供GPU加速服务。另一方面，实验结果指出不适合使用远程GPU虚拟化部署计算资源要求较高的容器，这会导致计算资源浪费，因此在默认情况下自适应资源共享策略只会为资源请求小于SHARED_THRESHOLD的容器分配远程GPU资源。SHARED_THRESHOLD可以通过参数设置，默认值是0.2。

6. 结论

本文设计实现了一个扩展Kubernetes的GPU管理能力的框架——KubeGPU。为了满足容器性能需求以及提高系统吞吐量，我们提出了自适应资源共享策略。自适应资源共享策略根据容器配置以及可用GPU资源选取合适的GPU虚拟化技术部署容器。此外，目前Kubernetes在远程GPU虚拟化方面的研究还处于初级阶段，诸如资源隔离和通信开销等问题还未引起过多关注。为了减少通信开销，我们提出了基于网络感知的调度策略，通过分析底层网络为远程GPU虚拟化选取合适的网络资源和网络模式。为了在远程GPU虚拟化中实现资源隔离，我们设计并实现了前端模块和后端模块，前端模块是一个安装在客户容器上经过修改的GPU运行库，用于拦截CUDA相关的API调用，并将API调用转发给服务容器。后端模块是一个安装在服务容器上经过修改的GPU运行库，用于管理并分配客户容器使用的GPU资源，避免容器超额独占整个GPU资源。实验结果表明自适应资源共享策略提升了1.05倍的系统吞吐量，网络感知策略相较于Kubernetes默认的网络策略至少提升了1.24倍性能，并且KubeGPU可以有效避免容器超量使用共享远程GPU资源。

文中注释：

① 论文将在第2.3小节详细介绍网络模式和网络设备

② 透明意味着KubeGPU能够与k8s其他已有组件共同运行并减小额外开销。因此KubeGPU不能够替换集群中已安装的插件，而是通过为容器分配可用的，性能更好的网络模式减小额外开销。

③ 为了方便表示并管理集群的设备资源，我们提出了Device Resource的概念，Device Resource是集群设备资源的一种抽象。

④ 如果一个GPU设备上没有运行任何任务，则其相对应的GPU Device Resource称为空闲的GPU Device Resource

⑥ Kubeshare管理框架与GROMACS容器存在兼容性问题，GaiaGPU管理框架与Pytorch MNIST容器存在兼容性问题。因此，他们在管理相应容器时，引入了较大的资源开销。这里的不兼容性是指GPU管理框架的资源分配策略无法满足容器对GPU资源的需求。

参考文献：

[1] Al Jawarneh IM, Bellavista P, Bosi F, Foschini L, Martuscelli G, Montanari R, Palopoli A (2019) Container orchestration engines: a thorough functional and performance comparison. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC), pp 1–6. IEEE

[2] Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, Shenker S, Stoica I (2011) Mesos: a platform for fne - grained resource sharing in the data center. In: NSDI, 11:22–22

[3] Red Hat OpenShift makes container orchestration easier (2021) https://www.redhat.com/en/technologies/cloud - computing/openshift. Accessed 29 Mar

[4] Swarm mode overview (2021) https://www.redhat.com/en/technologies/cloud - computing/openshift. Accessed 29 Mar

[5] Kubernetes (2021) https://github.com/kubernetes/kubernetes. Accessed 18 Nov

[6] Altintas I, Marcus K, Nealey I, Sellars SL, Graham J, Mishin D, Polizzi J, Crawl D, DeFanti T, Smarr L (2019) Workfow - driven distributed machine learning in CHASE - CI: a cognitive hardware and software ecosystem community infrastructure. In: 2019 IEEE international parallel and distributed processing symposium workshops (IPDPSW), pp 865–873. IEEE

[7] Managing Resources for Containers (2021) https://kubernetes.io/docs/concepts/confguration/manage - resources - containers. Accessed 22 Oct
[8] Yoon DH, Han Y (2020) Parallel power fow computation trends and applications: a review focusing on gpu. Energies 13(9):2147
[9] Hong CH, Spence I, Nikolopoulos DS (2017) GPU virtualization and scheduling methods: a comprehensive survey. ACM Comput Surv (CSUR) 50(3):1–37
[10] Naranjo DM, Risco S, de Alfonso C, P é rez A, Blanquer I, Molt ó G (2020) Accelerated serverless computing based on GPU virtualization. J Parallel Distrib Comput 139:32–42
[11] Silla F, Prades J, Iserte S, Reano C (2016) Remote GPU virtualization: Is it useful?.In: 2016 2nd IEEE international workshop on high - performance interconnection networks in the exascale and big - data era (HiPINEB), pp 41–48. IEEE
[12] Thinakaran P, Gunasekaran JR, Sharma B, Kandemir MT, Das CR (2019) Kube - knots: resource harvesting through dynamic container orchestration in gpu - based datacenters. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, pp 1–13
[13] Lu Q, Yao J, Guan H, Gao P (2019) gQoS: a QoS - oriented GPU virtualization with adaptive capacity sharing. IEEE Trans Parallel Distrib Syst 31(4):843–855
[14] Gonzalez NM, Elengikal T (2021) Transparent i/o - aware gpu virtualization for efcient resource consolidation. In: 2021 IEEE international parallel and distributed processi - ng symposium (IPDPS), pp 131–140. IEEE
[15] Tang D, Li L, Ma J, Liu X, Qi Z, Guan H (2021) gremote: cloud rendering on gp - u resource pool based on api - forwarding. J Syst Archit 116:102055
[16] Song S, Deng L, Gong J, Luo H (2018) Gaia scheduler: a kubernetes - based scheduler framework. In: 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable
[17] cGPU overview (2021) https://www.alibabacloud.com/help/en/container - service - for - kubernetes/latest/cgpu - overview. Accessed 22 Sep
[18] k8s - device - plugin (2021) https://kubernetes.io/docs/concepts/confguration/manage - resources - containers. Accessed 13 Nov
[19] Reaño C, Silla F, Shainer G, Schultz S (2015) Local and Remote gpus Perform Si - milar with edr 100g Infniband. In: Proceedings of the Industrial Track of the 16th International Middleware Conference, pp 1–7
[20] Reaño C, Silla F (2016) Reducing the performance gap of remote gpu virtualizationwith infniband connect - ib. In: 2016 IEEE symposium on computers and communication (ISCC), pp 920–925. IEEE
[21] Reaño C, Silla F (2017) A Comparative Performance Analysis of Remote gpu Virtualization Over Three Generations of gpus. In: 2017 46th International Conference on Parallel Processing Workshops (ICPPW), pp. 121–128. IEEE
[22] Qi S, Kulkarni SG, Ramakrishnan K (2020) Understanding container network interfa - ce plugins: design considerations and performance. In: 2020 IEEE international symposium on local and metropolitan area networks (LANMAN, pp 1–6. IEEE
[23] Xu C, Rajamani K, Felter W (2018) Nbwguard: Realizing Network qos for Kuberne - tes. In: Proceedings of the 19th International Middleware Conference Industry, pp 32–38
[24] Deepomatic (2021) https://github.com/Deepomatic/shared - gpu - nvidia - k8s - device - plugin. Accessed 12 Oct
[25] gpushare - scheduler - extender (2021) https://github.com/AliyunContainerService/gpushare - scheduler - extender Accessed 4 Nov
[26] Kang D, Jun TJ, Kim D, Kim J, Kim D (2017) Convgpu: Gpu Management Middleware in Container Based Virtualized Environment. In: 2017 IEEE International Co - nference on Cluster Computing (CLUSTER). IEEE, pp 301–309
[27] Yeh TA, Chen HH, Chou J (2020) KubeShare: a framework to manage GPUs as fr - st - class and shared resources in container cloud. In: Proceedings of the 29th internati - onal symposium on highperformance parallel and distributed computing, pp 173–184
[28] Chiang MC, Chou J (2021) DynamoML: dynamic resource management operators for machine learning Workloads. In: CLOSER, pp 122–132
[29] Satzke K, Akkus IE, Chen R, Rimac I, Stein M, Beck A, Aditya P, Vanga M, Hilt V (2020) Efcient GPU sharing for serverless workfows. In: Proceedings of the 1st workshop on high performance serverless computing, pp 17–24
[30] Vinoski S (2002) Chain of responsibility. IEEE Internet Comput 6(6):80–83

[31] Single Root I/O Virtualization and Sharing Specifcation Revision 1.1. https://members.pcisig.com/wg/PCI-SIG/document/download/8238. Accessed 20 Jan, 2010
[32] Kang J, Lim J, Yu H (2020) Partial migration technique for GPGPU tasks to Prevent GPU Memory Starvation in RPC-based GPU Virtualization. Softw: Pract Exp 50(6):948–972
[33] Xiao S, Balaji P, Zhu Q, Thakur R, Coghlan S, Lin H, Wen G, Hong J, Feng Wc (2012) Vocl: an optimized environment for transparent virtualization of graphics processing units. In: 2012 innovative parallel computing (InPar), pp 1–12. IEEE
[34] Abraham MJ, Murtola T, Schulz R, P á ll S, Smith JC, Hess B, Lindahl E (2015) GROMACS: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX 1:19–25
[35] Paszke A, Gross S, Massa F et al (2019) PyTorch: an imperative style, high-performance deep learning library. In: Advances in Neural Information Processing Systems 32 (NeurIPS 2019), pp 8024–8035
[36] Vouzis PD, Sahinidis NV (2011) GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics 27(2):182–188
[37] Anderson JA, Glaser J, Glotzer SC (2020) Hoomd-blue: a python package for high- performance molecular dynamics and hard particle monte carlo simulations. Comput Mater Sci 173:109363
[38] Liu Z, Chen C, Li J, Cheng Y, Kou Y, Zhang D (2022) Kubfbs: a fne-grained and balance-aware scheduling system for deep learning tasks based on kubernetes. Concurr Comput: Pract Exp 34(11):6836
[39] Abadi M, Barham P, Chen J, Chen Z, Davis A, Dean J, Devin M, Ghemawat S, Irving G, Isard M et al (2016) Tensorfow: a system for large-scale machine learning. In: 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16), pp 265–283
[40] Reaño C, Prades J, Silla F (2018) Exploring the use of remote gpu virtualization in low-power systems for bioinformatics applications. In: Proceedings of the 47th International Conference on Parallel Processing Companion, pp 1–8

要闻集锦

英伟达与英特尔联合打造AI超级计算机：效能提升25倍

英伟达称目前正和英特尔合作，打造AI超级计算机，搭载英伟达的H100计算卡以及英特尔的第四代至强处理器，在各种参数上均比上代提升许多，比如说AI计算中，内存算力有着突飞猛进的变化，基于全新的传输技术可以让数据实现每秒400Gbps的传输，从而让CPU等硬件与存储和服务器之间的传输更加便捷。

此外根据英伟达的计算，得益于英伟达的AI神经网络以及英特尔的至强处理器，新一代AI超级计算机相比较传统的X86计算机在AI训练以及高性能计算上能够提升20至40倍的能效提升，在特定的计算项目中性能提升9倍，例如传统计算机进行某个庞大的语言模型训练，需要大约40天的时间，而采用AI超级计算机，这个训练时间就将大幅减少，最低可以减少至1-2天。

人工智能正在推动整个企业和工业计算的下一波自动化浪潮，使企业在应对经济不确定性时能够事半功倍。新的AI超级计算机将首先应用于微软的Azure云计算平台，为其在虚拟机实例中带来其独特的全球可扩展性，英伟达表示这将有助于加快各种人工智能工具的训练和部署。英伟达表示，像Megatron Turing NLG 530B这样的基础模型将在该计划下得到快速发展，其目标是开发用于构建代码、文本、数字图像、音频和视频的“无监管”自我学习算法，为每个企业提供最先进的AI能力。

（卢永捷）

Karmada：云原生多云编排系统简介

隋新元 上海超级计算中心 上海 201203 xysui@ssc.net.cn

1. 背景

在数字化浪潮的大背景下，随着云计算相关产业的不断完善，我们迎来了一个万物上云的时代，Kubernetes作为云计算的操作系统，为容器化的应用提供了部署运行、资源调度、服务发现、动态伸缩等方案，提高了大规模容器集群管理的便捷性。但是随着业务模型的复杂度不断上升，集群规模也随之快速增长。为了构建多集群系统，多云架构越来越成为云计算使用者的一个新诉求，异构多云逐渐成为企业未来常态化的基础设施面貌，并且逐渐呈现出从孤岛模式到水城模式，再到最终的大航海模式的发展趋势。同时多云也逐步成为数据中心建设的基础架构，多区域容灾与多活、大规模多集群管理、跨云弹性与迁移等场景也进一步推动了云原生多云相关技术的快速发展。

目前，从业界的产品和用户使用来看，多云应用还处于从一群孤岛到威尼斯水城的过渡阶段，大部分开源软件和厂商的产品还是致力于进行统一的集群生命周期管理与集群信息收集，从而实现快速的选择切换集群以及集群的外部流量打通，从而实现全局流量的分配。但跨集群自动分配以及应用的跨集群能力还存在明显的不足。具体来看，在实际的生产落地过程中云原生的多云仍面临如下挑战：

- **集群繁多的重复劳动**：缺少标准化的集群管理接口，运维工程师不得不在集群的生命周期管理上维护繁琐重复的配置，以及碎片化的API访问入口.
- **业务过度分散的维护难题**：应用在各集群的差异化配置繁琐；业务跨云访问以及集群间的应用同步难以管理。
- **集群的边界限制**：应用的可用性受限于集群；资源调度、弹性伸缩受限于集群。
- **厂商绑定**：业务部署的黏性问题，缺少自动化故障迁移；缺少中立的开源多云容器编排项目。

此外，当前云原生体系下的多云多集群，和云计算体系下的概念认知存在相当大的理念沟壑，这也导致了在云原生领域多云相关技术演进的方向，实际上是一个复杂的系统工程。在云原生体系下，既有的多云多集群，都是围绕应用为中心的管理视角，这超越了云计算下的仅仅以资源分配为中心的管理视角。不能让应用无感知的进行多云多集群部署，并不是原生的多云多集群。

2. 多云编排的设计思想

原生的多云多集群，是实现了云自由下的，多云与多集群一体的设计理念。并全面兼容既有的应用编排与调度体系，实现对应用的跨云跨集群无感迁移。从设计角度来看，包含：「集群基础设施管理」，「集群生命周期管理」，「集群配置策略管理」，「集群应用编排管理」和「集群应用运维管理」，五大模块。

集群基础设施管理，由CNCF基金会定义的Cluster API规范所定义，其规定了所有云计算服务或产品提供商需要满足的基础设施驱动接口。该API直接代表了云计算公司对云原生领域的承诺与开放度，是多云兼容的底层基础。

集群生命周期管理，由CNCF基金会的Kubeadm项目定义，约束了集群所有控制单元的模块所应该遵循的样本模块和部署架构。并需要提供模块和架构的 Kubernetes LTS 版本支持策略。

集群配置策略管理由 Kubernetes LTS 版本最佳实践不断演进迭代，从而进行集群最佳参数配置与策略设定。并能够根据实际生产需要，施加不同等级的数据保护策略。

集群应用编排管理规范了跨集群管理应用的编排与调度。主流方案包括由谷歌主导、基于垂直扩展的方式，该方式将多个Kubernetes集群组成联邦集

群；以及基于水平扩展的方式，具体包含阿里Fuxi调度系统，字节跳动Godel，以及华为Karmada统一调度器。

集群应用运维管理，该领域涉及到完整的可观测性能力提升，跨云跨集群的网络管理和存储管理，及多云多集群下的容灾方案。

3. Karmada简介

Karmada是由华为开源的多云编排框架，是CNCF的云原生沙箱项目，结合了华为云容器平台MCP以及 Kubernetes Federation核心实践，并融入了众多新技术：包括Kubernetes原生API支持、多层级高可用部署、多集群自动故障迁移、多集群应用自动伸缩、多集群服务发现等，并且提供原生Kubernetes平滑演进路径，让基于 Karmada的多云方案无缝融入云原生技术生态，为企业提供从单集群到多云架构的平滑演进方案。Karmada的优势主要有以下几点：

- **Kubernetes 原生API兼容**

既有应用配置及基础设施无需改造，可以实现由单集群架构平滑升级到多集群（多云）架构，无缝集成Kubernetes现有工具链生态。

- **开箱即用**

支持面向多场景的内置策略集，包括两地三中心、同城双活、异地容灾等。支持应用的跨集群上的自动伸缩、故障迁移和负载均衡。

- **避免供应商锁定**

与主流云提供商集成。自动分配，跨集群迁移。不受专有供应商编排的约束。

- **集中管理**

可以实现与位置无关的群集管理。支持公有云、私有云或边缘集群。

- **富有成效的多集群调度策略**

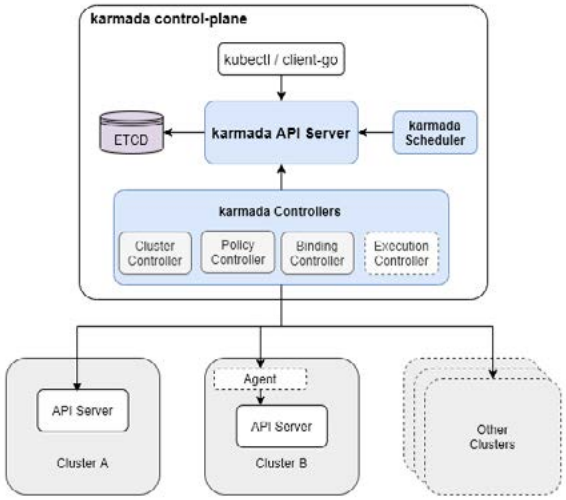
支持多集群亲和性调度、应用跨集群拆分、资源重新平衡。可以实现多维度多层次的高可用部署：区域/可用区/集群/供应商等。

- **开放和中立**

项目由互联网、金融、制造、电信、云提供商等共同发起。同时使用CNCF实现开放式治理的目标。

4. Karmada架构

4.1 控制平面与负载平面



从宏观上来看，Karmada分为控制平面（Control Plane）与负载平面（Workload Plane）两部分组成。Karmada本身需要运行在Kubernetes集群中，称作为Host Cluster（宿主集群），用来运行Karmada控制平面的组件，其中包含Karmada的etcd，karmada-api server，karmada-controller manager，Kubernetes controller manage，karmada-scheduler，karmada-webhook，karmada-scheduler-estimator等控制面的组件。负载平面负责真正运行工作负载的集群，称之为Workload Cluster。在Workload Cluster集群中，会真正运行业务的容器、一些Kubernetes的资源对象、存储、网络、DNS等，同时对于pull模式的部署方式，还会运行Karmada的Agent组件，用于和控制面组件通信，完成工作负载的下发能力，从而实现私有云的编排能力。

4.2 基本概念

- **Resource Template**

Kubernetes原生API定义的资源对象，无需对资源对象进行修改即可创建多集群应用。

- **Cluster**

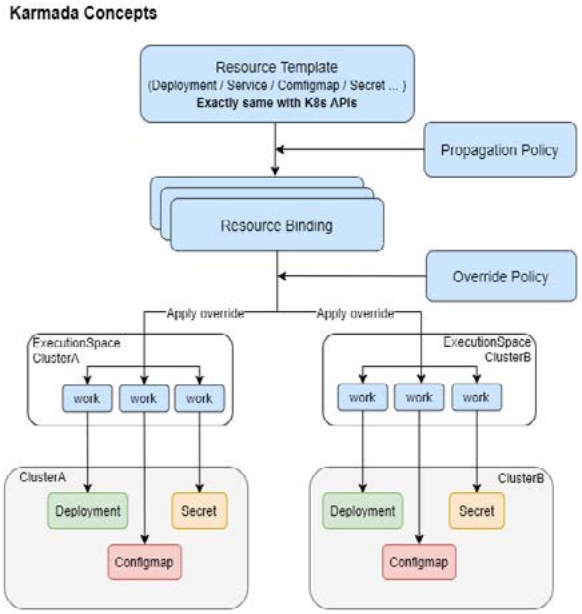
代表一个完整的，单独的一套Kubernetes集群，是可用于运行工作负载的集群的抽象和连接配置。

- **Propagation Policy**

多云调度策略，可以定义集群亲和性，集群容忍。以及实现按集群标签、故障域进行负载分发。

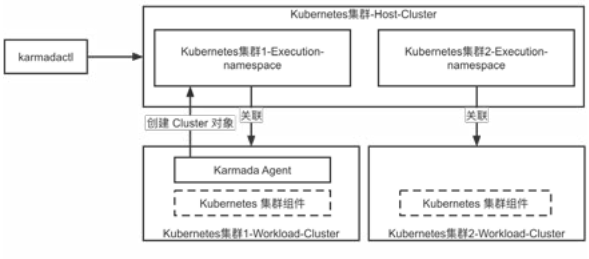
- **OverridePolicy**

定义在下发到不同集群中的配置，实现不同集群的差异化配置能力。如针对不同的集群配置不同镜像仓库的地址，实现异构调度。



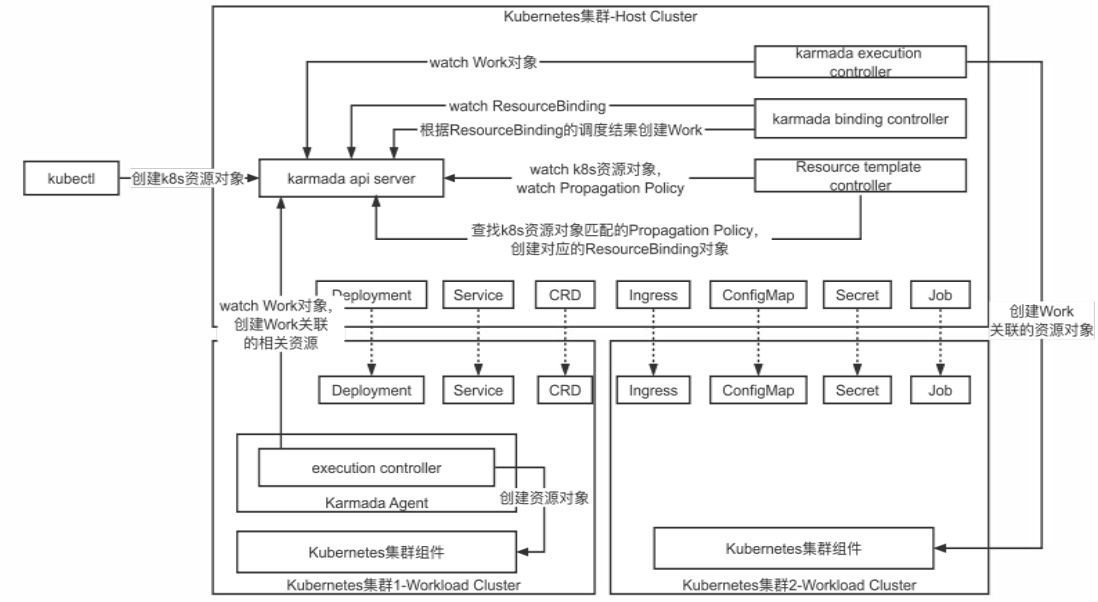
4.3 主要模块

Controller(控制器)是Karmada的核心，同时在Kubernetes中是逻辑能力的主要体现所在，根据资源对象的状态来完成调和工作，让资源对象逐步接近期待的状态，也就是Kubernetes的申明式特性。在Karmada中，同样需要对Karmada自己的资源对象，实现对应的申明式特性，这就需要实现对应的Controller。



- **Cluster Controller**

负责处理Cluster资源对象，建立Host-Cluster与Workload-Cluster之间的关联关系，以及在控制平面中对其进行逻辑表示。



- **Resourse Template controller**

负责对resource template资源对象的调度和处理，匹配PropagationPolicy，派生出资源对象对应ResourceBinding对象。

- **Binding controller**

处理ResourceBinding资源对象的增删改逻辑，ResourceBinding的调和能力是派生出work对象，work对象是和特定集群关联的。一个work只能属于一个集群，代表一个集群的资源对象的模型封装。

- **Execution Controller**

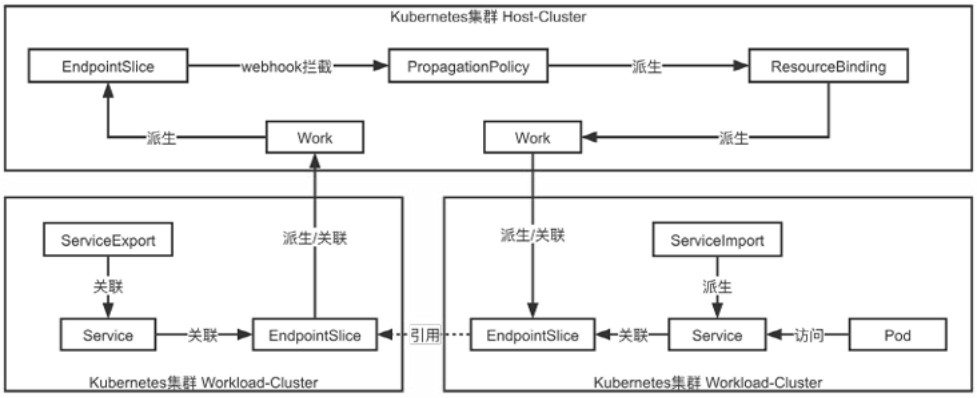
主要就是处理Work资源对象的增删改逻辑，用于处理Work，将Work负责的Kubernetes资源对象在对应的集群上创建出来。

- **ServiceExport Controller**

主要就是处理ServiceExport资源对象的状态逻辑，将需要被其它集群发现的服务暴露出来。

- **EndpointSlice Controller**

根据ServiceExport资源对象对应到处的Service，Service对应的EndpointSlice上报到Karmada的控制面。



- **ServiceImport Controller**

主要负责根据ServiceExport暴露出来的Service，在自己负责的集群中创建对应的Service，Service的名称不是完全一样的，同时在自己负责的集群中也创建对应的EndpointSlice，这个EndpointSlice的数据就是来源于EndpointSlice Controller中上报到Karmada控制平面的EndpointSlice对象，具体是通过在karmada-webhook中给ServiceImport的PropagationPolicy中增加了EndpointSlice的下发能力。

5. 小结

多云异地部署越来越成为了云计算的一个发展趋势，Karmada提供了提供丰富的多集群调度策略以及开箱即用的内置策略集，可以极大的简化两地三中心、异地容灾和同城双活架构下的系统复杂性。其具备的原生API支持、多层级高可用部署、多集群自动故障迁移、自动伸缩等功能为多云异构的管理、运维提供了一个新的思路，推动了多云编排领域的发展。为多云和混合云架构保驾护航。

要闻集锦

Deepmind推出AI系统AlphaTensor，可发现矩阵相乘的精确算法

提高基础计算算法的效率是当今一项至关重要的任务，因为它会影响大量计算的总体速度，而这些计算可能会产生重大影响。其中一个简单的任务就是矩阵乘法，它可以在神经网络和科学计算程序等系统中找到。机器学习有潜力超越人类直觉，打败目前可用的最典型的人类设计算法。

然而，由于可能的算法数量众多，自动发现算法的过程是复杂的。DeepMind最近通过开发AplhaTensor取得了突破性的发现，这是首个可为矩阵乘法等基本任务发现新颖、高效且正确算法的AI系统。解决了一个已经开放了50多年的数学难题：如何尽可能快地将两个矩阵相乘。

矩阵乘法被用于许多事情，包括处理智能手机上的图像，识别口头命令，为视频游戏创建图形等等。开发有效地矩阵相乘的计算硬件会消耗大量资源;因此，即使矩阵乘法效率的微小提高也会产生重大影响。该研究调查了如何通过使用当代人工智能方法来推进新矩阵乘法算法的自动开发。

AlphaTensor挖掘了大量最先进的复杂度算法，每一种矩阵乘法的复杂度算法多达数千种，证明矩阵乘法算法的空间比以前认为的更丰富。这些方法证明了AlphaTensor在优化任意目标时的适应性，通过乘大矩阵比在相同硬件上的传统算法快10-20%。

由于矩阵乘法是许多计算的基本操作，由AlphaTensor开发的技术可以显著提高各个领域的计算效率。该系统的适应性可以考虑任何形式的目的，这可能会激发开发优化算法的新应用，如能耗和数值稳定性等指标。尽管这项研究集中在矩阵乘法的具体问题上，但DeepMind希望鼓励其他人将人工智能应用于指导其他重要计算工作的算法开发。

（陈继军）

城市治理数字化转型：动因、内涵与路径

陈水生 复旦大学 国际关系与公共事务学院 上海 200433

摘要：

数字时代呼唤与之相适应的城市治理新形态。技术变革的驱动力、治理生态的不确定性、治理问题的复杂性以及治理需求的多样性共同驱动城市治理数字化转型。城市治理数字化转型要及时回应数字时代需求，坚守以人民为中心的核心价值，重视制度变革的基石作用，充分发挥技术创新的驱动力，实现城市治理的科学化、精准化、智慧化与高效能。城市治理数字化转型的实现，需构建全功能集成、全网络融合、全周期管理、全要素连接的数智治理体系，促进城市治理现代化。

1. 问题提出

数字时代与智能技术推动着城市数字化全面转型。数字化转型是一个旨在通过信息、计算、沟通、连接技术的组合等方式促进治理方式发生实质性改变的过程。也有学者将其界定为在信息技术应用不断创新和数据资源持续增长的双重叠加作用下经济、社会和政府的变革和重塑过程。近年来，我国出台了一系列促进数字化发展和数字化转型的政策。党的十九大报告提出新时代建设网络强国、数字中国、智慧社会的发展战略，要求各级政府快速适应数字化环境，推动政府职能转变。《中华人民共和国国民经济和社会发展第十四个五年规划和2035年远景目标纲要》提出加快数字化发展，建设数字中国，加快建设数字经济、数字社会、数字政府，以数字化转型整体驱动生产方式、生活方式和治理方式变革。

全国多个省级政府也出台了促进数字化转型的政策文件。2018年11月，浙江省通过了《浙江省保障“最多跑一次”改革规定》，提出推动政府数字化转型。2020年12月出台的《浙江省数字经济促进条例》强调数字基础设施建设与数据资源支撑，聚焦数字产业化、产业数字化以及治理数字化等方面。2021年5月，广东省政府发布了《关于加快数字化发展的意见》，聚焦数字经济、数字社会和数字政府三大领域，提出全方位赋能经济社会转型升级，把广东建设成为全球领先的数字化发展高地。2021年1月，上海市委、市政府公布《关于全面推进上海城市数字化转型的意见》，提出充分运用数字化方式探索超大城市社会治理新路子，回应人民对美好生活的新期待，明确城市数字化转型的总体要求；强调城市数字化转型是一项系统工程，要整体性促进城市经济、城市生活和城市治理数字化转型。2021年10月，《上海市全面推进城市数字化转型“十四五”规划》出炉，明确了未来五年上海城市数字化转型的发展目标和重点任务。

为适应数字时代的发展要求，从中央到地方，都把数字化转型作为未来经济、社会和治理的发展方向，城市治理数字化转型也成为国家治理现代化的重要内容。那么，城市治理数字化转型的动因与内涵是什么？转型方向与构建路径如何？这是本文试图回答的问题。

2. 城市治理数字化转型的动因分析

数字时代的城市治理面临的治理任务、难题和挑战日趋复杂多元，技术进步给城市治理创新提供了技术支持，多重因素共同推动城市治理的数字化转型。具体而言，技术变革的驱动力、治理生态的不确定性、治理问题的复杂性以及治理需求的多样性共同驱动城市治理的数字化转型。

2.1 技术变革的驱动力

技术革命引发国家和社会结构的改变，要求城市治理积极变革。全球范围内政府数字化转型的步伐正在加快，数字化转型成为各国的普遍发展趋势。技术变革的驱动力主要体现在两个层面：一方面，数字化技术助推城市治理进入智慧时代。信

本文已刊发于《理论与改革》2022年第1期

息社会中数据和信息不但成为社会关系和社会生活的基础，也成为公共治理的基础。作为重要治理工具的数字化技术能提升信息传递效率、降低层级沟通成本，让城市治理变得更聪明更智慧。通过技术赋能，政府治理能力不断增强，“最多跑一次”改革、一网通办、城市大脑等技术变革提升了办事效率与管理效果，城市治理从数据、信息整合走向智能、智慧的新阶段。另一方面，数字技术所带来的技术滥用、隐私保护、故障风险也增加了城市治理的技术复杂性，同时，数据缺失、数据标准化程度较低，以及数据调取难、处理效率低等问题制约着城市治理创新。可见，技术是一把“双刃剑”，既赋予城市治理更大的便捷性和更高的效率，但技术引发的高度复杂性和不确定性也导致传统的组织决策模式因过于僵化而难以应对新需求、新问题和新挑战。技术治理效果依赖于治理者的管理，如何管理技术使用是城市治理数字化转型的直接动因；技术进步及其带来的消极影响亟需通过治理变革和转型加以解决。概言之，城市治理数字化转型是适应技术发展需求、化解技术带来的问题、促进技术治理健康发展的客观要求。

2.2 治理生态的不确定性

在大变革时代，城市治理生态愈发复杂和多变，不确定性给城市治理带来系列风险。为了降低治理生态的不确定性，就需要借助各种新技术和新工具，城市治理数字化转型是应对不确定性的最佳选择。首先，数字时代与信息社会充满了不可预测性与不确定性。城市治理数字化转型有利于实时、动态、灵活地调整城市治理工具，提升风险应对能力。其次，城市非传统风险带来的不确定性。城市中各种非传统风险的出现具有突发性与不确定性，容易引发社会失序、经济失调和政治失治的复合型危机。在新兴经济迅猛发展的社会中，居民生活中面临的各类风险和不确定性内化于城市的方方面面，风险问题呈指数级增长，不可预期事件更加频繁，比如极端气候灾害、恐怖袭击、传染性疾病传播等。在危机处置和应急管理过程中，政府数字化平台、大数据和智能技术的大规模使用，有利于提升政府应对治理生态不确定性的反应速度、能力和效能。最后，国际竞争、城市竞争和数字经济博弈带来的不确定性。数字全球化成为当下主流发展趋势，数据、技术对国家安全和发展至关重要，数字经济的迅速发展加剧了大国之间的博弈，数字博弈成为地缘博弈新焦点。中国只有通过不断促进政府治理转型融入全球数字化与智能化浪潮，才能不断缩小与发达国家的差距，维护我国国际地位与经济安全。总之，不论是在经济领域还是社会生活中，不确定性无处不在，风险不可避免且难以预测，城市治理数字化转型能够及时预测、处置和化解风险，维护经济稳定、社会秩序和城市发展。

2.3 治理问题的复杂性

中国城市发展进入从速度扩张到内涵式发展的新阶段，因此如何摆脱粗放式发展模式，迈向高质量发展和高效能治理是城市现代化的重要议题。城市治理问题的复杂性首先表现为超大规模城市的治理压力。中心城市和城市群已成为我国承载发展要素的主要空间形式。以城区常住人口为统计口径，我国已有超大城市7座、特大城市14座、大城市98座。大城市的集聚效应也意味着城市治理的难度不断增加：由于城市的超大规模，人口和企业高度集中，引发城市空间的压力；人口结构的多元化和文化的多样性，加大了超大城市的多样性和复杂性；超大城市的公共安全和社会秩序风险加大，超大城市及其治理变成了一个复杂的巨系统，这给城市治理带来很大压力。其次，“城市病”叠加复杂化。城市治理面临一系列越来越难以解决的“城市病”，如城市无序扩张、人口总量剧增、资源环境承载能力超过极限、交通堵塞、居住拥挤、环境恶化、空气污染、疾病流行、房价高企等。这些问题的解决往往与其他“棘手难题”交织在一起，问题链条变动不居，治理难度也随之增加。为了破解“城市病”，就需要积极引入大数据、人工智能、数字技术等，通过数字技术赋能实现“城市病”的高效能治理。最后，城市新业态的治理难题。随着数字经济、人工智能等新技术应用到经济领域，产生了一系列新兴产业和新业态，这些新业态的发展、监管和治理呼唤与之相适应的数字化治理技术和治理政策，以适应经济新业态健康可持续发展的需要。

2.4 治理需求的多样性

治理需求的多样性首先表现为国家治理、政府治理和城市治理的多层次治理需求的叠加。国家治理体系与治理能力现代化需要城市治理作为排头兵，城市治理现代化要为国家治理现代化做出积极贡献，这就需要城市治理适应数字时代的治理需求，加快转型步伐。其次，治理需求的多样性要求城市治理满足城市利益相关者的多元需求，比如为企业提供良好的营商环境，为社会组织提供适宜的生长土壤，为民众提供宜居乐居的生活环境等等，而这些需求往往存在冲突。再次，治理需求的多样性还表现为城市居民对高品质美好生活的追求。十九大报告指出，我国社会主要矛盾已经转化为人民

日益增长的美好生活需要和不平衡不充分的发展之间的矛盾。人民选择到城市工作和生活，就是为了享受更加美好的城市生活。随着生活水平的不断提升，民众的需求也越来越多元化，期待享受高品质、便捷化和个性化的服务，还期望更积极地参与城市治理过程。数字治理不仅实现了对政府组织的内部赋能，也实现了对外部的公众赋权，使公众可以借助信息技术所开辟的通道，参与政府决策过程，促进政府管理走向以公民为中心的治理转型之路。数字技术为更好地洞察和回应社会需求提供了基础，数据的刻画更加精准地回应民众的需求，并且能够提供高效的组织内和组织间协调。可见，城市治理的数字化转型契合了人民对美好生活的需要与城市高效能治理的需求。为满足和适应治理需求多样性的要求，要积极运用数字技术对发展需求、治理需求和服务需求加以整体性解决，全方位满足城市利益相关者的多元化需求。

总之，城市治理面临着日益复杂和多样化的挑战，数字技术的发展为城市治理提供了技术支持，也带来了诸多挑战。城市治理的数字化转型是大势所趋，也是城市治理变革之道。城市治理数字化对破除城市治理难题、提升治理效能、增强人民的幸福感与获得感具有重要价值。城市治理数字化转型要积极回应上述挑战，深刻把握数字化转型的内涵体系。

3.“价值-制度-技术”三位一体：城市治理数字化转型的内涵体系

城市治理数字化转型要适应数字时代的发展与治理要求，充分运用大数据、人工智能、数字孪生等技术，推动城市治理的技术变革与制度创新的融合，实现城市高质量发展、高品质生活和高效能治理的有机统一。城市治理数字化转型的内涵体系可从治理价值、治理制度、治理技术三个层面理解，其中，以人民为中心是核心价值，制度变革是治理基石，技术创新是重要驱动力。

3.1“以人民为中心”是城市治理数字化转型的核心价值

“以人民为中心”的理念是城市治理数字化转型的核心价值。早在2013年，习近平就在中央城镇化工作会议上提出要以人为本，推进以人为核心的城镇化。2015年中央城市工作会议指出，做好城市工作，要顺应城市工作新形势、改革发展新要求、人民群众新期待，坚持以人民为中心的发展思想，坚持人民城市为人民，这是城市工作的出发点和落脚点。2019年，习近平在上海杨浦滨江考察时提出“人民城市人民建，人民城市为人民”的重要思想。在新时代，城市治理数字化转型要坚持以人民中心的价值追求，围绕人民需求，把城市建设成为老百姓宜业宜居的美好生活家园。因此，城市治理数字化转型要从人民立场出发，不断更新治理理念，通过敏捷治理、韧性治理和智慧治理变革以更好地践行以人民为中心的城市治理理念。

首先，树立敏捷治理理念。城市治理数字化要求城市政府快速适应新环境，将城市政府打造为具有快速响应力的敏捷组织。敏捷思想最初源于敏捷制造，其核心要义包括完整设计开发、快速测试修改、积极响应用户需求、加强团队沟通协作等。为了缓解数字时代快速变化的治理问题与滞后的政府反应之间的矛盾，敏捷思想被引入公共管理领域，进而重塑了政府形态与治理方式。敏捷政府强调政府以有效方式响应不断变化的公共需求。敏捷治理被视为一种自适应、以人为本、具有包容性、可持续性的决策过程，主要是指对现有治理结构和治理流程进行升级，从而以更快的速度应对更复杂、更多变的城市问题的治理模式。敏捷治理体现了对公共价值和人民期盼的快速响应，与城市治理数字化转型所强调的以人民为中心的理念不谋而合。

其次，树立韧性治理理念。韧性原本被用于描述物体受到外力后恢复原状的能力，面临外界冲击不仅能保持原有功能运转，还能在冲击后快速恢复原样。因此，韧性概念既包含面对压力冲击的承受和适应能力，又包括冲击后的恢复与再生能力。城市韧性是指城市系统通过合理准备，缓冲和应对不确定性，实现城市正常运行的能力。韧性治理理念强调城市在面临突发事件、外力冲击所导致的不确定性时，城市治理体系由被动处置变主动管控，城市功能常态运作、自我调整能力提升，这也是在快速变化和充满不确定性时代背景下，治理数字化转型所要达成的治理目标。在上海市推进城市数字化转型“十四五”规划中，强化城市运行新韧性是城市治理数字化转型的一大任务。

最后，树立智慧治理理念。数字技术的发展与公共管理变革共同催生了智慧治理的兴起。智慧治理可以理解为综合运用物联网、人工智能、数字孪生等现代信息技术，驱动治理制度变革，使城市治理各领域、各环节和各事务迈向精准化、智能化和高效化。“智慧”意味着治理体系可以迅速精准地感知、判断、预测和解决各种城市问题。一方面，数字技术在数据收集、智能感知、计算分析方面展现出前所未有的应用前景，新技术成为实现智慧治理不可或缺的工具。另一方面，智慧治理强调技术组合而产生的技术应用综合效应，更多地将技术应

用置于“人-技术-技术”和“技术-技术-技术”能级层面，进而提升治理有效性。同时，相比于电子政务通过技术治理来实现智能化的取向，智慧治理更加强调技术服从于理念、价值等因素的系统治理需要，将技术与理念深度融合，更快更好地满足民之所需，最终落实到为民众创造美好生活上来。

总之，治理数字化转型是认知与思维的系统转变。不论是敏捷治理、韧性治理还是智慧治理，归根结底都要以人民为中心，及时响应人民的真实需求，维护公共利益和公共价值，最终满足人民对美好生活的期盼。城市治理数字化转型要明确城市发展的本质在于服务人的发展，厘清城市治理数字化转型与“以人民为中心”城市工作的紧密关联，将数字化转型作为满足人民对美好生活向往的重要手段。

3.2 制度变革是城市治理数字化转型的治理基石

制度变革是由技术创新、流程再造和服务优化所引发的制度变迁与政策革新。在新兴技术快速迭代的背景下，数字化转型要求传统城市治理制度、规则体系和治理政策与时俱进，积极变革以适应技术创新和技术应用的发展要求，同时满足民众多元化的美好生活需求。城市治理制度变革要兼顾公共价值、秩序稳定、驱动创新和治理效能等多元目标，重点可从构建契合性、开放性、整合性制度体系出发，不断推动治理制度变革。

首先，构建契合性的治理制度。契合性的治理制度是指在数字化背景下，迅猛发展的技术要有及时变革的制度与之适配，制度供给要跟进数字化转型的进程，整体设计，因势驱动。上海在推进城市数字化转型中坚持“技术+制度”双轮驱动，及时研究制定一批标准、规则、政策、法规，为全面推进城市数字化转型提供更加精准有效的法律和制度保障，即技术为基、制度为要。不论是职能部门的协作与多层级的联动，还是需求与服务的有效对接，都需要制度变革发挥有效配合作用，如果制度变革跟不上技术的创新速度，再先进的治理技术也难以发挥其应有的治理效用。

其次，创设开放性的治理制度。开放性的治理制度是指通过制度的吸纳和整合机制，秉承共建共治共享原则，构建开放型吸纳和适应性改革的多元、复合、融合性的制度体系，实现治理制度的创新性变革。传统的城市治理制度更多指向政府内部闭环的制度规则，而城市治理数字化转型所需的制度体系更具开放性，既包含支持技术创新的容错机制，又具有包容审慎的监管机制，还指向制度本身的持续创新，从而实现技术赋能与制度赋权的双轮驱动。

最后，构建整合性的治理制度。城市治理制度体系由多层级、多领域和多样化的治理制度所构成，既包括中央政府、城市政府等层级性制度体系，也包括社会治安、教育制度、产权制度、监管制度和社会保障等领域性制度，还包括法律、政策、规则等多样化形态所构成的整体性制度网络，这就需要加强不同制度样态之间的整合、匹配和协同，发挥制度合力，减少制度摩擦和冲突。以往分裂式的城市治理制度带来治理议题分隔、治理权责分割、治理政策碎片化等弊端，导致治理制度效能难以有效发挥，不能解决涉及全局性的重大问题。因此，治理数字化转型要构建整合性与一体化的治理制度体系，实现从分域治理向整合治理的转型，提升制度效能。

概言之，制度变革是推进城市治理数字化转型的治理基石。推进城市治理数字化转型的关键在于制度的有效供给和优化，通过构建契合性、开放性和整合性的制度体系以适应数字技术的发展和数字化转型的需要。城市数字化治理的有效转型需要实现技术驱动与制度变革双向适配，双轮驱动。上海市提出要按照“数字化转型推进到哪里，技术制度同步驱动到哪里”的原则，以城市数字化转型为创新试验田，鼓励前沿技术和应用的创新实践，加快消除数字化转型的制度门槛，完善包容开放的发展环境，建立适应数字化转型的技术体系、规范体系和政策体系。

3.3 技术创新是城市治理数字化转型的重要驱动力

城市治理数字化转型需要充分发挥技术的驱动力。大数据、物联网、云计算、人工智能等数字技术的发展使得信息的存储、传输、计算与交互发生了极大变化，为城市治理创造了全新的空间，数字技术应用的整体性、系统性突破深度改变了社会结构和状态。公共服务供给由低效向高效、由粗放向精准、由机械向智能转型。可以说，技术创新为城市治理数字化转型提供了变革新动能。

弗洛里迪将技术分为三级，一级技术连接人与自然，二级技术连接人与技术，三级技术连接技术与技术，交互链条无限延长，当媒介技术连接起作为使用者的技术与作为敦促者的技术时，就会发生革命性飞跃，即技术会呈指数级发展,而现代信息与通信技术则是三级技术的最佳典范。新一代技术的协同发展，为城市治理数字化转型提供了丰富的技术支持，正是这种技术组合特质深刻推动了治理数字化转型，驱动城市治理从信息化到智慧化的发展演进。

那么，技术创新如何驱动治理数字化转型呢？首先，技术创新提供了治理新资源。新一代信息通信技术将城市部件和基础设施连接起来，能够将实体城市虚拟化与数字化，这使得全面感知整座城市得以可能，可以精确感知城市生命体征，精准把握城市脉动和治理需求。而且，城市海量数据得以累积与储存，拥有丰富数据是释放技术算法算力的基础，这为城市治理数字化转型提供了源源不断的资源基础。其次，技术创新促进治理体系优化。数字化转型将新技术作为战略工具支撑，跨越多元治理主体边界，整合部际信息共享与协同，将传统割裂式治理转变为整合连接式治理，打破“九龙治水”治理格局。新兴数字技术为不同治理主体之间广泛的互联互通、协作共享、业务协同提供了有力的技术支持，为问题处置从封闭低效向协同高效转变打下根基。数字技术的发展为跨部门、跨层级和跨区域问题的解决提供了技术支持，降低了治理成本。再次，技术创新驱动治理模式变革。技术创新通过加强治理主体、治理对象、治理目标、治理工具以及治理资源的互联互通，形成复合化、网络化治理形态。数字技术的信息处理与数据分析能力改变了传统的城市治理手段与治理工具，使城市治理告别了人海战术，聚焦于技术变革推动治理工具创新，治理流程优化，技术为治理提供了强大的算力底座和治理功能。最后，技术创新提升城市治理效能。技术创新有助于促进城市治理政策的科学制定与高效执行，提高城市服务的供需匹配程度，增强城市对潜在问题的智能预测预警能力，提前化解城市风险，提升城市危机治理水平。数字化技术创新强调以整合性、一体化的治理增进城市公共利益，改善民众服务体验和满意度，提升城市治理效能。

城市治理的技术驱动效应在实践中已广为体现。以上海市的技术助力防台防汛工作为例，上海市以大数据、人工智能、云计算为技术支撑建立“一网统管”平台，在台风来临之际，平台汇聚了3万多路视频、实时气候数据以及传感器数据，为及时预判各种潜在风险位置提供了技术保障。“一网统管”平台还打通了部门壁垒，向各业务部门提供包括算法和数据在内的、统一的数字底座支撑，为后续业务协同奠定基础，有利于及时指导各个部门精准部署处置力量。在“一网统管”平台提供的数据和算力支撑下，消防救援队精准掌握易积水点和高空坠物警情，快速进行灾情处置；水务局及时发布相应工地撤离信息，保证工人人身安全；绿化与市容管理局通过树干断层扫描将树的安全系数进行分级，更精准地对树木进行修剪或支撑，加固树木排除隐患。技术的力量使得部际之间实现了有效连接与协同，最终实现线上线下协同“高效处置一件事”。

4. 数智治理：城市治理数字化转型的构建路径

城市治理数字化转型积极发挥理念的价值引领、技术创新的驱动效能、制度变革的系统红利，迈向城市数智治理。数智治理将数字技术与智慧治理高度融合，在以人民为中心的价值理念引领下，充分发挥数字技术和智能技术的效用，通过治理技术创新、治理制度变革、治理过程优化和治理体系再造等方式，促进城市治理体系与治理能力双重变革，实现城市治理的科学化、精准化、便捷性、高效能与智慧化等目标。城市治理数字化转型要构建全功能集成、全网络融合、全周期管理、全要素连接的数智治理体系，实现城市治理现代化的发展目标。

4.1 城市治理数字化的全功能集成

城市治理数字化转型首先要构建一个集发展功能、服务功能、治理功能于一体的全功能集成体系。首先，城市的全面发展功能。城市全面发展要服务于城市自身的建设、发展与进步，促进数字经济、数字社会和数字服务的整体性发展。城市的全面发展既包括物质繁荣，又包括精神文明，还包括技术进步；既包括城市形象等外在表现，也包括城市文化等内在品质；既包括居民个体的发展，又包括城市整体的发展。就个体而言，城市治理数字化转型秉持以人民为中心的理念，其目的在于为人民提供更好的发展机会和公共服务，使个体能够发挥出自我价值，实现人的自由、充分和全面发展。上海市提出“五个人人”的发展理念，即人人都有人生出彩机会、人人都能有序参与治理、人人都能享有品质生活、人人都能切实感受温度、人人都能拥有归属认同，很好地阐释了“以人民为中心”促进个体发展的价值导向。就城市而言，数字化转型有利于激发城市活力，提高城市整体竞争力，构建智慧、敏捷、韧性的城市。通过推动经济数字化转型助力高质量发展，推动生活数字化转型创造高品质生活，推动治理数字化转型实现高效能治理，这是充分发挥城市治理数字化转型的发展功能的重要体现。

其次，城市的系统服务功能。公共服务是市民幸福感、满足感和获得感的重要来源，也是衡量一个城市是否宜居乐居的重要指标。与传统城市政府所提供的服务相比，数智治理能够提供更为精准与便捷的服务。就服务精准度而言，数字技术与智慧

治理能够更加灵敏地感知与识别民众需求，从而针对性地提供个性化服务，提高公共服务供给的精准程度。从服务的便捷性来看，数字化转型促进数据库的完善与共享，打通政府内部信息壁垒，系统梳理并重构业务流程，以“数据跑路”替代“群众跑腿”，将传统单纯线下的政务服务模式转变为线上线下融合的服务模式，民众通过联网设备便可随时随地办理业务，极大提高了服务可得性与便捷性。浙江省推行的“最多跑一次”改革、上海市的“一网通办”都是典型的实践。

最后，城市的整体治理功能。数智治理重视技术辅助决策、技术赋能监测预警、技术助推系统治理。在技术辅助决策方面，数字技术凭借其极强的信息收集和数据计算能力，能够将复杂的治理问题抽象为模型，通过算法优化求解，帮助决策实现从经验判断型向数据分析型的转变，提高决策科学性。此外，数据的可视化与可追踪性在一定程度上能够增加决策的透明度。在技术赋能监测预警方面，传统治理更多的是在发现问题后采取应对措施，数智治理的不同之处在于，它能通过数字化、网络化、智慧化的技术手段主动发现甚至预测问题，更好地预见并防范风险，充分发挥预警功能。在技术助推系统治理方面，通过数字技术构建平台治理体系，有利于从更高视野、更大范围、更深层次系统治理城市问题，集治理主体、治理资源、治理对象、治理工具等要素于一体，构建一体化、整体性、系统性的智能治理平台。

4.2 城市治理数字化的全网络融合

数智治理体系融合数据之流、技术之网与智慧大脑，通过要素网络的相互融合发挥数据、信息、技术的叠加效应，提升数据治理和智慧治理效能。数字技术使得万物互联成为可能，通过数据之流、技术之网和智慧大脑形成连接治理。连接治理通过技术、信息与数据等，实现万物互联、人人互联、人机互联、人-组织互联、技术-制度互联，运用数据构建连接的桥梁与基点，有利于打破碎片化和割裂化治理，实现全景、全程和全域治理。

首先，数智治理强调数据之流的流动共享。数字要素的累积和流动，数据与信息汇集是数智治理的关键要素，为数智治理提供数据之源。数据是基于各种各样的自然现象和社会现象的原始记录与原始素材。数据是城市治理中最基础、最底层的要素构成。数据治理是城市治理精准化的基础，如果没有数据支撑，精准化治理就不可能实现。数智治理要构建自由流动、开放共享的数据体系和数据平台，及时收集、归整、分析、交换和共享海量城市数据。城市数据集成与数据治理要依托神经元系统、数据底座和数据中台支持，用数据描绘城市实体，用数字刻画城市态势，将城市政府的公共数据，事业单位和公共企业的准公共数据，以及城市公民的需求数据进行深度整合，将城市医疗、就业、教育、社会保障等各个领域的数据进行系统集成，从而构建城市运行数字体征体系。同时，城市数据治理要强化全面感知，加强神经元网络布局，与市民和市场主体建立更多链接，从对物的感知向对人和组织的感知拓展。按照智能化、科学化、精准化的方向，推动形成共建共治共享的数据治理格局。

其次，数智治理强化技术之网的贯通融合。数智治理依赖于各种新兴技术手段、方式、工具及其应用，将各种数字技术引入城市治理领域和场景，实现城市治理技术的升级迭代。现代技术发展为数智治理提供了技术基础，包括以物联网为代表的感知技术，以移动通信网、互联网为代表的信息传输技术，以云计算、人工智能等为基础的支撑技术。技术之网的功效关键在于贯通融合，将整个城市纳入一张“网”，各种技术可以在这张网中有效对接、汇通与融合，形成技术治理合力。发挥技术之网之功效，一方面要加强城市信息基础设施的建设与完善，加强关键技术的研发和攻关，加快布局关键共性和前瞻引领的数字技术，发展人工智能技术、云计算技术、大数据技术、区块链技术、数据脱敏脱密技术等，为数智治理提供强有力的技术支撑。另一方面，要加强互联网、大数据、计算机这三大关键要素的融合，打造统一的数字治理平台，推动数字化平台、数字围网和数字智能物联网同步建设，发挥技术的系统合力。

最后，数智治理重视智慧大脑的指挥联动。智慧大脑是城市治理数字化的指挥中枢、决策核心和协调平台。智慧大脑强调技术赋能，它通过汇集、储存和运用城市不同领域数据资源，运用强大的运算能力和算法模型，以数据资源为基础，推动全面、全程、全域构建城市治理体系和治理能力现代化的数字系统，实现城市治理模式和服务模式创新。智慧大脑通过数据、信息、技术等要素网络的相互融合，发挥叠加效应，将其提升到系统集成的层面，以满足数智治理所追求的高效、整合、智慧治理的要求。《上海市全面推进城市数字化转型“十四五”规划》中提出要构建以底座、中枢、平台互联互通的城市数基，经济、生活、治理数字化“三位一体”的城市数体，政府、市场、社会“多元共治”的城市数治为主要内容的城市数字化总体架构。《杭州城市大脑赋能城市治理促进条例》进一

步推动城市智慧大脑赋能城市治理的实践创新，运用城市大脑实现全面实时感知、全程实时分析、全域实时处置，不断完善“一整两通三同直达”的中枢系统，优化“一脑治全城、两端同赋能”的运行模式，提升智慧大脑赋能城市治理水平。

4.3 城市治理数字化的全周期管理

数智治理要对城市公共事务实行全周期管理。伴随着城市公共事务的日益复杂化，城市治理迫切需要连贯化、系统化、统筹化的城市问题处置方法，全周期管理的逻辑正好与之契合。数智治理将城市视作生命体和有机体，建立反映城市作为生命体的体征指标体系，将影响城市生命体健康的风险隐患尽早预测、及时预警、全力预防、有效应对，构建“观、管、防、处”一体化、全过程、全周期的治理体系。

全周期管理将管理对象视作生命体，按阶段划分产品的生命周期，在每一阶段都介入跟踪来保证产品质量。全周期管理注重从系统要素、结构功能、运行机制、过程结果等层面进行全周期统筹和全过程整合，以确保整个管理体系从前期预警研判、中期应对执行再到后期复盘总结学习，各个环节均能运转高效、系统有序、协同配合。全周期管理的理念能使城市治理的不同领域与环节从分散走向聚合，从而由点到线、由线到面、由面到网。全周期管理的精要在于依托信息集成平台推进城市全要素统筹和全流程整合，连接城市治理主体和对象、结构与功能、系统与要素、过程与结果，最终实现系统完备的全平台治理体系。平台治理模式是实现城市公共事务全周期管理的实践创新的典范。

数智治理的全周期管理要加快构建平台治理模式。最先将平台理念引入公共管理领域的是Tim O'Reilly，他提出了“政府即平台”的理论政府结构的科层制特征萌发了分层级性质平台的出现，同时，部门的横向分工与分割化的治理领域导致出现多个不同的系统平台。这些平台彼此之间并不相通，而是相互分离与割裂的。随着数字技术的发展，通过技术赋能构建一体化平台成为可能。平台治理可视为一种借鉴平台经济和平台企业的理念、技术和方法，以平台思维创新政府治理理念和治理模式，构建整体性、一体化、智能化与高效能的数字化治理体系。平台治理具有数字化、感知性、互动性、无界性和智慧化等特征。在此背景下的平台治理成为一种高度技术化、网络化、智能化与民本化的一体化治理新模式。比如，广州以经济数字化带动生活、治理数字化，通过建设中新广州知识城国际数字枢纽、南沙（粤港澳）数据服务试验区等数字服务平台，建立和优化了数字交易共享机制，保障数据安全，防范出现“数字悬浮”风险。可见，数智治理充分运用数字技术，通过技术整合、制度变革与流程再造，构建全流程管控、全周期管理、全平台治理体系，降低城市数字化治理的风险。

4.4 城市治理数字化的全要素连接

数智治理同时亦是一种实现万物互联的连接性治理，实现城市物联、数联和智联。以物联网、大数据、人工智能为代表的数字技术具有连接性的特征，其连接特性将治理主体、治理对象、治理资源和治理工具等各个要素有机连接起来，实现物-物、人-人、人-机、人-组织、技术-制度全要素连接。具体而言，万物互联、技术互联、技术与制度贯通可以看作全要素连接的三个层次。

首先是由物联网技术开启的万物互联，这是人-机-物要素的相互连接，也是技术赋能数智治理的基础层次。物联网能够将特定空间环境中的所有物体连接起来，进行拟人化信息感知和协同交互。并且，物联网具备自我学习、处理、决策和控制的行为能力，为城市治理由信息化向智能化转变提供了契机。既可以在线提供矢量化数据，又可以通过对城市部件与设施的数据采集、分类与建库，满足各部门在线调用、共享交换及应用开发等需求，为城市治理从粗放转向精细、从机械转向智慧、从随意转向规范提供基础数据。

其次是技术互联。技术与技术的互联分为三级技术，作为使用者的技术与作为敦促者的技术被媒介技术连接起来后，技术互联就会发生革命性飞跃。技术互联要促进云网融合发展，云网融合是“云计算”与“通信网”的结合，其本质是技术与技术的连接与融合。从技术角度出发，云计算能够提供极强的数据存储和计算能力，而通信网则能将地理上分散用户终端互连起来，实现通信和信息的传输与交换。二者的结合将计算和连接相融合，进而实现治理数字化，促使城市减少治理盲点。

最后，全要素连接更深层次是要实现技术与制度的贯通。数字技术嵌入治理过程将影响和塑造治理制度。技术嵌入促进了信息共享、互联互通平台的出现与发展，相应的制度体系如公民参与机制、监督体系等在技术影响下也进行了调整或重构。制度设计则关系到技术发挥的程度，因此要与时俱进地调整完善制度安排，通过构建契合性、开放性、整合性的制度体系促进技术的迭代升级。数智治理的全要素连接最终要贯通技术与制度，实现“技术+制度”双轮驱动数字化转型。

5. 结语

无论是从全球还是国内来看，数字化都是大势所趋。数字化发展和转型本身构成了一个整体性体系，城市经济数字化转型是城市的发动机，城市治理数字化转型是城市的火车头，而城市生活数字化转型是城市的会客厅，三者缺一不可，共同推进城市数字化转型。在这三者中，城市治理数字化由于其具有强烈的规划引领和政策导向等功能，更由于城市治理效能往往决定城市经济数字化发展成效和城市生活数字化目标的实现，故要加强对城市治理数字化转型的研究，把握城市治理数字化转型的动因、内涵、方向与路径。城市治理数字化转型也不能单兵突进，需要城市经济数字化的强有力的支持，也需要满足城市生活数字化的最终目标，实现以人民为中心的数字化发展与数字化转型。

数智治理是城市治理数字化转型的新方向。数智治理重视新兴技术的创新、应用与驱动作用，没有技术的创新驱动，城市治理数字化转型就失去了技术之基，也失去了重要驱动力。为此，城市治理数字化转型首先要重视对数字产业的培育和激励，通过有效的产权制度、产业政策和营商环境引导数字经济和数字产业的发展壮大，为城市治理数字化提供源源不断的技术之源。这就需要政府在推动城市治理数字化转型中树立系统思维，整体推进数字经济、数字治理和数字生活的一体化发展。其次，政府在推进数字经济发展和城市治理政策创新时，要努力做到“在地化”，学会因地制宜、因城施策，不用“一刀切”的政策进行产业引导和城市治理，也不宜强行推行数字化转型政策。政策的生命力在于灵敏、调适和适宜。数字经济政策和数字治理政策还要“懂人心”，政策既要读懂人心，也要满足人心。对数字经济企业而言，人心背后是需求，需求背后是商机；对政府而言，民意民心是政策生命线，也是施政之根基，及时响应民众需求并有效满足是城市治理数字化转型的应有职责。

总之，城市治理数字化转型要坚持人民为核，制度为基，技术为要。城市数字化发展和治理数字化转型是一个囊括了价值共创、财富共享、多元共治的过程。城市治理数字化转型，政府不能单打独斗，必须秉承政府、市场、社会、公众共治的价值理念，坚持多元参与和共建共享的原则，这样才能在数字化时代真正实现协同治理。城市治理数字化转型要重视价值、制度和技术的共演发展，将企业、政府和民众有机链接，通过城市治理数字化转型促进城市发展、服务与治理的有机统一，实现科学化、精准化、智能化和便捷化等治理目标，最终为人民创造美好生活，促进城市治理现代化。

高性能计算
发展与应用 （内部刊物）

DEVELOPMENT & APPLICATION OF HIGH PERFORMANCE COMPUTING

主　办：上海超级计算中心

协　办：江南计算技术研究所
上海科学技术情报研究所

编辑部：上海张江高科技园区郭守敬路585号

邮　编：201203

电　话：021-61872222

传　真：021-61872288

投　稿：journal@ssc.net.cn

电子版：https://www.ssc.net.cn