中国科学: 信息科学 2017 年 第 47 卷 第 9 期: 1149–1163
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
云计算与大数据国家重点研发专项专题 • 论文

低熵云计算系统

徐志伟[1,2*], 李春典[1,2]

1. 中国科学院计算技术研究所计算机系统结构国家重点实验室, 北京 100190
2. 中国科学院大学, 北京 100049
* 通信作者. E-mail: zxu@ict.ac.cn

收稿日期: 2017–04–08; 接受日期: 2017–07–03; 网络出版日期: 2017–09–06
国家重点研发计划 (批准号: 2016YFB1000200) 和国家自然科学基金重点项目 (批准号: 61532016) 资助

摘要 当前的云计算系统, 不论是虚拟化云还是分区云, 难以同时满足用户体验和系统效率需求, 产业界和学术界都开始研究下一代云计算系统以应对这个难题. 本文指出, 这个难题的一个重要原因是计算系统熵 (无序、干扰和不确定性) 居高不下, 并归纳了云计算系统中存在的 4 类无序现象. 本文提出了低熵云计算系统的学术概念, 刻画了其主要特点, 比较了低熵云计算系统与虚拟化云和分区云在用户体验、开发效率、运行效率、资源适配方面的区别, 并讨论了低熵云的新概念和新技术: (1) 不同于图灵可计算性和算法可计算性的实用可计算性概念, 形式化地刻画了云计算行业的 “用户体验差的功能是不存在的功能” 的实践经验; (2) 刻画云计算系统能够实现实用可计算性的充分必要条件, 即 DIP 猜想; (3) 支持 DIP 猜想, 即能够区分、隔离、优先化计算任务相空间, 从而降低干扰, 有潜力同时满足用户体验和系统效率需求的标签化 von Neumann 体系结构; (4) 适配深度学习负载与神经网络处理器的云计算协同设计技术.

关键词 云计算, 用户体验, 系统效率, 计算系统熵, 分布式系统, 计算机体系结构

1 引言

过去十余年来, 云计算 [1,2] 的研究和使用发展迅速, 已经成长为主流的新兴计算模型, 为科学研究、企业计算、互联网服务、移动互联网等众多领域提供了各种多租户弹性服务, 包括基础设施服务 (IaaS)、平台服务 (PaaS) 和应用软件服务 (SaaS) 等. 同时, 云计算存在的种种问题也暴露出来, 尤其 “同时满足用户体验与系统效率要求” 的难题变得日益突出, 成为下一代云计算系统研究的一个焦点 [3~8].

当代的云计算平台主要有两大类, 它们都难以同时满足用户体验与系统效率要求.

(1) 分区云. 将云端数据中心计算机划分成为互不影响的分区 (即机群), 每个应用独占一个分区. 这有利于保障用户体验, 但削弱了资源共享程度, 降低了系统利用率. 分区 (partition) 可以理解为应

引用格式: 徐志伟, 李春典. 低熵云计算系统. 中国科学: 信息科学, 2017, 47: 1149–1163, doi: 10.1360/N112017-00069
Xu Z W, Li C D. Low-entropy cloud computing systems (in Chinese). Sci Sin Inform, 2017, 47: 1149–1163, doi: 10.1360/N112017-00069

用独占的机群. 采用分区云的一个典型云服务商是 Facebook. 在云计算实践中, 资源利用率小于 10% 的分区云并不罕见.

(2) 虚拟化云. 多个虚拟计算机共享一个云端数据中心计算机或一个机群. 虚拟化云可采用虚拟机 [9~11]、容器 [12]、无服务器函数[1)]等技术实现. 虚拟化有助于共享资源, 提升系统利用率. 但为了保障用户体验, 云服务商往往超占资源, 这又降低了资源利用率. 技术实力强的云服务公司, 也只能达到 20% ~ 40% 的资源利用率 [13].

下一代云计算系统需要综合分区云和虚拟化云的优点, 它既像分区云那样保障用户体验, 又像虚拟化云那样支持资源共享. 本文提出低熵云计算系统, 其主要特点是将系统的无序 (disorder) 与不确定性 (uncertainty) 控制在一个可接受的小范围, 从而避免无序共享, 在保障用户体验的同时, 提升系统效率.

低熵云计算系统需要有效地应对当前存在的负载无序、负载干扰、系统噪声、阻抗失配等 4 类无序现象. 低熵云计算系统需要新的计算机体系结构机制, 支持对每个资源访问可区分、可隔离、可优先响应. 低熵云计算系统是一种负载之间干扰少、系统噪声和开销小, 从而负载的执行时间较为确定, 高优先级服务不被低优先级操作妨碍的, 有序共享的云计算系统.

2 低熵云与虚拟化云和分区云的比较

图 1 比较了虚拟化云、分区云、低熵云. 双线表示分割机群, 单线表示隔离. 当代云计算的一个主要特征是虚拟化 (virtualization), 其主要目标是通过多个虚拟计算机共享一个云端数据中心计算机 (datacenter computer) [5], 提高系统效率, 体现云计算的弹性资源优点. 每个虚拟计算机各有自身的虚拟机软件、操作系统和应用软件. 虚拟化的早期典型包括 2003 年左右 VMWare 与 XEN 虚拟机为代表的虚拟化技术 [10, 14], 以及 2006 年 Amazon 推出弹性计算云服务 (EC2)[2)].

图 1(a) 展示了一个数据中心计算机的 3 个机群, 每个机群内部的多个虚拟计算机共享一套硬件资源. 第 1 个机群展示了传统的虚拟机技术, 在一套硬件资源上虚拟出 3 个虚拟计算机, 运行各自的虚拟机软件、操作系统和应用软件. 这些应用包括前端万维网服务 (Web)、Redis 缓存服务 (Redis), 以及后端数据计算服务 (Hadoop) 等.

近年来虚拟化技术有两个重要发展. 一是开源软件 Docker 的流行引发的容器 (container) 技术的普及 [12, 15][3)]. 二是 Amazon 等公司推出了 “无服务器计算” (serverless computing) 技术. 图 1(a) 第 2 个机群展示了当前主流的容器技术, 即 Linux container 技术 (LXC), 其主要特点是在一套 Linux 操作系统中隔离出多个容器, 每个容器运行各自的应用软件. 与虚拟机技术相比, 容器技术的虚拟化开销小, 且有利于部署应用, 近年来流行很快. 图 1(a) 第 3 个机群展示了 Amazon 公司推出的 AWS Lambda 无服务器计算技术, 它是一种更细粒度的虚拟化技术. 它不需要用户租用虚拟机或容器, 而是提供待计算的函数 (称为 Lambda), 云计算厂商按照函数计算使用的资源收费. 多个租户的多个函数 (如 sort, wordcount) 共享云计算硬件、操作系统 (如 Linux) 与运行时软件 (如 Hadoop).

这些虚拟计算机 (容器、Lambda 函数) 应该保证相互隔离 (isolation), 包括 (1) 语义隔离: 一个虚拟计算机的操作不影响其他虚拟计算机的应用语义; (2) 故障隔离: 一个虚拟计算机出错不影响其他

1) Amazon AWS Lambda. https://aws.amazon.com/lambda.

2) Amazon AWS EC2. https://aws.amazon.com/ec2.

3) An update on container support on Google Cloud Platform. https://cloudplatform.googleblog.com/ 2014/06/an-update-on-container-support-on-google-cloud-platform.html.

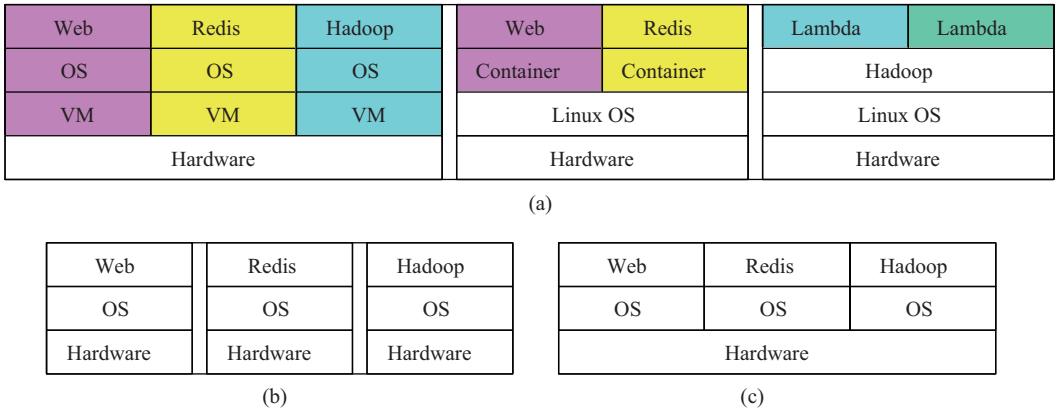


图 1 (网络版彩图) 3 类云计算技术

Figure 1 (Color online) Three types of cloud computing techniques. (a) Virtualization cloud; (b) partitioned cloud; (c) low-entropy cloud

虚拟计算机运行; (3) 性能隔离: 一个虚拟计算机的资源需求不影响其他虚拟计算机的计算、存储、带宽性能, 也就是说, 一个虚拟计算机运行不应挤占其他虚拟计算机的资源. 经过十几年的发展, 云计算技术已经基本做到了语义隔离, 故障隔离次之, 性能隔离还远未做到. 同样的一个计算任务, 独占本地单机运行 10000 次, 性能波动往往小于 1% [16], 即计算时间为 $T \pm 0.01 \times T$ 秒. 但在共享资源的云计算平台上, 性能波动要高得多. Google 的数据显示, 其云计算平台的性能波动高达 600% [9].

虚拟化云能够很好地支持小数据负载 (例如网站托管), 但针对大数据负载 (例如前端海量查询请求响应, 以及后端大数据挖掘与分析) 却暴露出来很多问题. 一个典型的场景是用户将本地的应用移植到云计算平台后, 应用性能变差, 而且应用性能不确定, 性能波动很大, 用户体验得不到保障.

应对的办法是分区云, 即为每个应用租用独占的机群. 图 1(b) 显示了一个数据中心计算机被划分成为 3 个分区, 分别运行各自的操作系统以及 Web, Redis, Hadoop 3 个应用. 大数据计算技术包括 Google 提出的 Map-Reduce [17] (对应的开源应用为 Hadoop[4])、BigTable [18] (对应的开源应用为 HBase[5])、Pregel [19] (对应的开源应用为 GraphX [20]). 为了保证用户体验, 这些大数据系统往往采用分区方式, 即一套应用运行在一套独占的机群上面.

既然虚拟化可以提升效率, 而分区云有利于保障用户体验, 能不能综合它们的优点, 提出一种新的云计算模式呢? 近年来, 学术界和产业界正在努力发展下一代云计算技术, 以便综合分区云和虚拟化云的优点, 既允许多个应用负载运行在数据中心计算机上以提高系统利用率, 又要保证用户体验. 图 1(c) 展示了中国科学院计算技术研究所提出的低熵云计算系统. 它与分区云以及虚拟化云相比的最大区别是, 即使不使用虚拟机与容器, 多个应用也能够弹性共享一套数据中心计算机的硬件资源, 以同时保障用户体验与系统效率.

3 低熵云计算系统的难点与目标

当前云计算系统难以在保障用户体验的同时得到较高的系统效率, 一个重要原因是存在 4 类无序现象 (disorder).

4) Apache Hadoop. http://hadoop.apache.org.
5) Apache HBase. http://hbase.apache.org.

• 负载无序 (intra-workload disorder). 第 1 类无序是一个应用负载内部的无序, 包括各种依赖、干扰和不确定性 [21]. 这类无序的主要特征是, 即使将应用负载部署在独占的机群上面也不能缓解. 一个典型的例子是采用了随机数发生器的非确定性算法在云计算平台上执行, 它的执行时间是不确定的.

• 负载干扰 (workloads interference). 第 2 类无序是两个或多个应用负载之间相互干扰造成的无序. 这类无序的主要特征是, 隔离应用负载, 例如将应用负载部署在相互隔离的独占的机群上面运行, 往往会显著缓解负载干扰带来的无序. 一个典型的负载干扰例子是, 后端批处理大数据计算作业挤占了交互式云服务的资源, 造成交互式云服务的用户体验急剧变差.

• 系统噪声 (system jitter). 第 3 类无序是硬件、虚拟机、操作系统、运行时等系统部件干扰应用负载执行带来的无序, 这些干扰也称为系统噪声. 一个典型的例子是运行时垃圾回收干扰应用负载运行. 这类无序的主要特征是, 将应用负载部署在独占的机群上面运行, 并关闭可能的系统噪声源 (例如关闭垃圾回收), 往往会显著缓解系统噪声带来的无序.

• 阻抗失配 (impedance mismatch). 第 4 类无序是应用负载与硬件部件和底层软件不匹配, 造成显著的拷贝、转换、重映射开销. 一个典型的例子是应用需要列存储, 而硬件和底层软件提供的却是行存储. 另一类实例是云计算和大数据系统中缺乏 MPI 这类高性能通信库支持, 使得需要频繁的拷贝和变换. 这类无序可由应用开发者解决, 但更好的办法是系统提供协同设计 (co-design) 支撑, 使得应用开发者有更好的工具和界面 (例如支持大数据计算的 DataMPI [22]). 近年来, 异构计算在云计算中日益普及, 下一代云计算系统需要考虑如何利用好 GPU, FPGA 等加速部件.

负载无序带来的问题主要应该由应用开发者解决. 为了应对负载干扰、系统噪声、阻抗失配这 3 类无序现象, 低熵云计算系统需要解决用户体验、开发效率、运行效率、资源适配 4 个难点问题. 它们反映了过去十余年来云计算领域积累的重要实践经验, 是低熵云研究的主要目标.

3.1 用户体验保障媲美分区云

低熵云的用户体验保障应当媲美分区云. 用户体验直接关系到云计算服务是否满足用户需求、是否能够获得并保住用户, 因此用户体验是第一优先, 必须得到保障. 云计算行业内将这个实践总结成了一句名言: “在云计算环境中, 用户体验差的功能是不存在的功能.” 事实上, 这条名言是很多用户在业务迁移到云计算系统的 “云化” 过程中, 看不到云计算服务的优势和益处的主要原因.

为了量化地刻画用户体验保障, Amazon 公司提出了三元组 (Amazon triple) [23]:

(并发请求数, 请求响应延迟阈值, 延迟小于阈值的请求百分比).

一个云计算服务的用户体验三元组为 (1 M, 100 ms, 99%), 代表着只有当该服务同时支持一百万个并发请求, 而且 99% 的请求的响应时间小于 100 ms, 才满足用户体验需求.

Google 公司发现了用户体验的尾延迟 (tail latency) 现象 [7]. 在很多重要的云计算服务 (例如搜索引擎) 中, 需要将 100 个乃至 1000 个并发请求整合起来得到最终计算结果. 哪怕只有 1% 的并发请求响应延迟超出 100 ms, 最终结果的延迟超过 100 ms 的概率将会超过 63%, 并不是人们往往期待的只有 1%. 如何控制尾延迟已经成为近年来云计算和大数据研究的重要方向. Google 团队认为 [5], “下一代云服务需要始终如一的及时响应 (consistently responsive) 大规模计算系统, 人们今天才开始思考这类系统”. 加州大学伯克利分校的 FireBox 项目 [3] 研究 2020 年的云计算系统, 一个主要目标是应对尾延迟.

3.2 应用开发效率媲美分区云

开发效率低是影响云计算实用的最大障碍. 由于 “用户体验差的功能是不存在的功能”, 开发一个云计算服务不仅需要体现其功能, 还必须保证该服务的响应延迟等用户体验, 这增加了开发难度. 在充满干扰的共享资源云平台上, 云服务的性能不确定, 性能优化工作的结果难以重现, 开发出性能稳定的云计算服务尤其困难.

技术实力雄厚的公司具有经验和人才, 可以在云计算服务的应用开发阶段, 通过应用层优化, 来解决用户体验难题. 但是, 这样的公司毕竟是少数. 我们需要这样一种共享资源云平台, 它的应用开发效率也媲美分区云. 开发云服务时, 开发者好像使用着干扰很小的独占的机群, 它更加接近用户熟悉的本地计算系统, 性能优化具备可重现性. 这种云平台将使得技术实力并不强的大量中小企业和个人用户也能够将他们的应用迁移到云计算平台上, 大大扩展云计算的应用面.

3.3 系统运行效率高

即使是实力雄厚的高科技公司, 尽管他们能够开发出满足用户体验的云计算服务, 但是在今天的共享资源云平台上也难以实现较高的系统运行效率. 例如, 美国著名互联网服务商 Twitter 在使用 Mesos [24] 共享资源云平台时, 租用并预留了接近 80% 的 CPU 和内存, 但绝大部分时间实际使用不到 20% 的 CPU 和 40% 的内存, 超过 50% 的 CPU 和 30% 的内存都浪费了 [13]. 这种实践被称为超容预留 (overprovisioning), 即在云计算系统中预留远超实际需求的资源. 低熵云应该显著减少这种超容预留浪费.

相对于应用开发效率, 系统运行效率用来刻画系统在运行时的有效性. 它关注利用率 (utilization)、执行效率 (efficiency) 和能效 (energy efficiency). CPU 利用率是 CPU 用于执行应用软件和系统软件的时间除以总时间. 执行效率是系统执行应用所得到的实际速度除以系统的峰值速度. 能效是系统执行应用所得到的实际速度除以系统的功耗.

图 2 显示了 1945 年到 2025 年的高性能计算机速度、功耗和能效的历史演变与发展趋势. 可以观察到一个历史性的现象. 在 2005 年以前的 60 年发展历史中, 性能 (每秒运算数) 和能效 (每度电运算数) 基本上保持了同步增长, 但从 2005 年开始, 能效增长显著慢于性能增长. 云计算当中的大数据计算尤其令人担心. 在 2015 年, 高性能计算的能效已经达到了每度电 6.85 千万亿次运算 (6.85×10^{15}) 的水平, 但使用大数据计算框架 Hadoop 做 1 PB 数据排序的能效仅达到每度电 576 亿次运算 (5.76×10^{10}), 即使是以速度快的内存计算著称的 Spark 应用框架 [25], 其做 1 PB 数据排序的能效仅有每度电 4320 亿次运算 (4.32×10^{11}) 的水平, 与高性能计算机相比还有 4 ∼ 5 个数量级的差距. 今天的云计算系统运行效率还有数量级的提升空间 [26, 27], 下一代云计算系统需要充分挖掘这个潜力.

3.4 硬件资源适配应用负载

低熵云计算系统需要支持硬件资源适配应用负载, 降低阻抗失配造成的无序开销. 例如, 在异构计算日益流行的今天, 这意味着云计算负载能够充分利用各种加速部件, 云计算系统带来的摩擦很小. 云计算的负载类型已经从科学计算、企业计算、互联网服务拓展到了大数据分析与机器学习, 负载类型变得多种多样. 为了提高运算速度和能效, 云计算系统越来越多地使用了 GPU、FPGA、神经网络处理器等加速器部件 [28∼30], 微软公司甚至号称在其 Azure Cloud 的每一个计算节点上都部署了 FPGA 加速部件 [31]. 低熵云计算系统需要显著降低使用加速部件的开销, 在应用层能够得到 70% 以上的裸硬件性能.

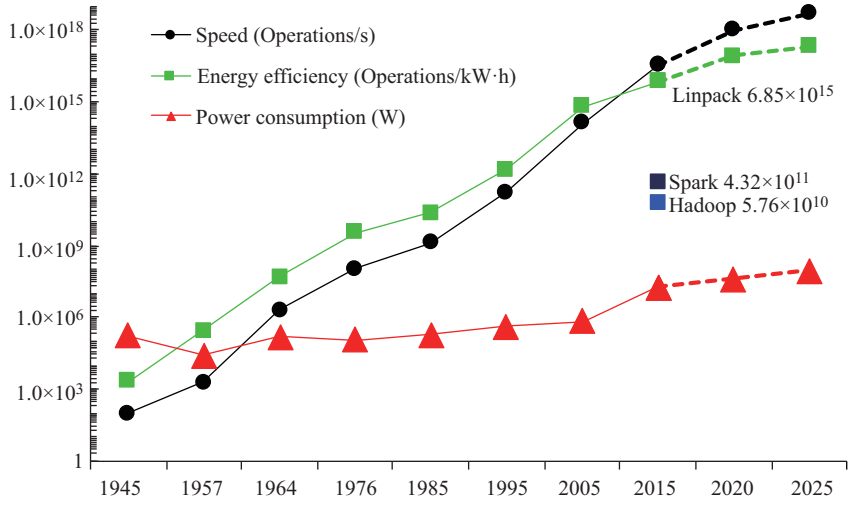


图 2　(网络版彩图) 高性能计算机的速度、能效、功耗的历史演变与发展趋势

Figure 2　(Color online) The trends of speed, energy efficiency, and power consumption of the world's fastest computers

4　低熵云计算系统技术要点

计算系统熵 (即无序) 居高不下, 是云计算系统难以同时满足用户体验和系统效率需求的重要原因. 而无序的主要原因是云计算系统中出现的各种干扰. 当多个租户多个应用共享云计算系统资源时, 降低干扰的一个基本思路是: 对每个计算任务, 自动地界定其使用资源的时间与空间 (合起来称为该任务的相空间), 并使用创新的低开销的硬件和系统软件技术, 动态地区分、隔离、优先化该相空间. 为此, 需要下列理论与技术:

• 实用可计算性 (production computability) 概念, 即判定计算任务在云计算系统中能否保证尾延迟小于用户体验阈值概念.

• 计算时空与相空间, 即任一计算任务使用资源的时间与空间, 合起来称为在计算时空中该任务的相空间, 即 phase space in cyberspacetime.

• 云计算系统的实用可计算性判定条件, 最好是充分必要条件, 用以连接实用可计算性与计算时空.

• 标签化 von Neumann 体系结构, 通过在计算时空中加标签, 能够动态地实现计算任务相空间的区分、隔离、优先化.

• 可动态适配计算任务的加速部件.

4.1　实用可计算性

由于 "用户体验差的功能是不存在的功能", 一个云服务的可计算性不能只用图灵可计算性或多项式复杂度算法可计算性来刻画, 而需要一种新的考虑用户体验的可计算性, 称为实用可计算性 (production computability): 一个云服务是实用可计算的, 当它的尾延迟小于用户体验阈值. 表 1 比较了这 3 种可计算性.

传统计算机科学精确定义了理论可计算性, 即图灵可计算性 (Turing computability), 并通过 Turing-Church 论题给出了图灵机的通用性, 以及停机问题等图灵不可计算问题实例. 图灵可计算性主要关注待解问题. 它与具体算法无关, 也与实现该算法的具体计算系统无关.

表 1 3 种可计算性之比较

Table 1 Comparison among three types of computability

Computability	Deciding reason	Example	Concerned elements
Theoretic computability	Turing computability	Halting problem is not decidable	Problem
Algorithmic computability	Polynomial time complexity	Gaussian elimination to solve equation	Problem & algorithm
Production computability	User experience (tail latency)	Search engine service in the cloud platform	Problem & algorithm & system

算法可计算性 (algorithmic computability, 也称为 tractability) 一般关注算法是否具备多项式时间复杂度. 例如, Linpack 软件使用 Gauss 消元法来求解 n 阶线性方程组, 其算法复杂度是 $O(n^3)$, 是多项式复杂度, 因此是算法可计算的. 算法可计算性既关注待解问题, 又关注解决问题的算法.

实用可计算性是考虑用户体验的可计算性, 它不仅关注待解问题与算法, 还关注实现算法的软硬件系统. 我们需要针对云计算场景下 "用户体验差的功能是不存在的功能" 的实践, 给出实用可计算性的精确理论定义, 并研究造成 "用户体验差" 的关键因素, 从而指导设计新型云计算模型, 使更多的应用得到足够好的用户体验.

给定云计算平台, 一个计算任务 $y = f(x)$ 的执行时间并不是一个常量. 即使是给定输入 x, 同一个计算任务 $y = f(x)$, 在同一个云计算平台上执行 1000 次, 每次执行时间也不是一个常量, 而是具有不确定性 (uncertainty). 将某个计算任务的执行时间称为其延迟, 并将大于某个百分位 (通常设为 99 百分位) 的延迟称为尾延迟 (tail latency) [7].

实用可计算性定义. 给定用户体验阈值 T, 如果存在共享资源云计算系统 S, 计算任务 A 在 S 中执行时, S 能够限制负载干扰、系统噪声、阻抗失配 3 类无序的影响范围, 使得 A 的 99 百分位尾延迟小于用户体验阈值 T, 那么 A 是实用可计算的.

实用可计算性不同于图灵可计算性与算法可计算性. 它是更加注重在云计算系统上实际满足用户体验需求的可计算性. 实用可计算性问题更加接近 Gray 的 TeraSort 问题 [32]. 也就是说, 实用可计算性关注云计算系统的基础理论, 而不是传统的理论可计算性. 下面用一个例子将实用可计算性放在图灵可计算性问题、算法可计算性问题、TeraSort 问题的上下文中比较, 进一步阐明实用可计算性的意义.

大数据排序问题. 这个问题是图灵奖得主 Gray 在 1994 年左右提出的. 他还制定了一个基准程序, 并在他 2007 年失踪之前一直维护这个基准程序网站[6]. 针对大数据排序, 可以提出如下 5 个层次的科学问题, 越来越接近实用云计算系统.

(a) 数据排序问题是否图灵可计算? 答案是 "可计算". 数据排序问题的一般表示是: 给定可比较大小的 n 个同类数据值, 输出这 n 个数据值从小到大的排序. 业界早已认识到数据排序问题的基础性和重要性 [32], 存在丰富的研究文献.

(b) 数据排序问题是否算法可计算 (tractable)? 答案是肯定的, 因为存在快速排序 (quicksort) 算法, 对任意给定 n 个数据值, 平均排序时间为 $O(n \log n)$, 最坏情况下排序时间为 $O(n^2)$, 都是多项式复杂度.

6) http://sortbenchmark.org/.

(c) TeraSort 问题: 能否在 1 min 内排序 1 TB 数据? 这 1 TB 数据由 100 亿条记录构成, 每条记录包含 100 字节. 这个问题是 Gray 在 1994 年提出的, 当时最好的世界纪录是每分钟排序 1.1 GB 数据, 采用 quicksort 算法在并行机上实现 [32]. TeraSort 问题在 2008 年基本得到了解决, Yahoo 公司在 3.48 min 完成了 1 TB 数据排序[7]. 随后, 社区将基准程序的数据规模扩展到了 100 TB. 今天的世界纪录保持者是腾讯公司, 其 Tencent Sort 系统在 2016 年取得了在 1 min 内排序 55 TB 的成绩, 排序 100 TB 则花了 1.65 min.

(d) TeraSort 的实用可计算性问题: 能否在具有干扰等无序的共享资源云计算系统上, 排序 100 TB 数据, 使得 99 百分位尾延迟小于 1 min? 这个问题目前尚无确定的答案. 这个问题与 (a), (b), (c) 有本质区别. 第一, 这个问题不只是关注理论可计算性和时间复杂度, 还关注具体的用户体验阈值, 即必须在 1 min 内完成排序. 第二, 这个问题不关心最坏的情况, 而是关注 99 百分位尾延迟. 也就是说, 哪怕还有 1% 的情况, 排序时间超过 1 min, 也是可以接受的. 实用可计算性的定义反映了实践中用户对云计算的容忍度. 用户知道云计算具有不确定性, 并可以容忍一定的不确定性. 我们使用微信通信时, 并不指望它像电报一样能够百分之百地保证消息按时到达接收方. 第三, 这个问题不只关心计算问题 (排序) 和算法 (quicksort), 还关心系统, 并特别指明是在具有无序的共享资源云计算系统中完成数据排序计算任务. TeraSort 问题则可以使用较少无序和干扰的本地计算机系统, 而不一定是云计算系统.

(e) 扩展的 TeraSort 的实用可计算性问题. 例子包括, 考虑更大的问题规模 (例如在 1 min 内排序 1 PB, 10 PB, 甚至 1 EB), 考虑能耗的 JouleSort (排序 1 TB 需要多少焦耳能量), 考虑成本的 CloudSort (在云计算平台上排序 1 TB 需要花费多少美元), 等等. 业界已经在独占机群或分区云平台上实现了 1 PB 排序和 10 PB 排序, 不过延迟都远高于 1 min. JouleSort 的最好记录是 1 TB 排序消耗了 168242 J 的能量. CloudSort 的最好记录是 100 TB 排序消耗了 144 美元, 延迟为 50 min. 与 Tencent Sort 系统排序 100 TB 仅需 1.65 min 相比, 使用云计算牺牲了用户体验.

实用可计算性是针对云计算系统的一种基本度量. 它综合考虑了算法可计算性、Gray 的 TeraSort 基准, 以及云计算实践中的用户体验与尾延迟挑战. 对未来云计算系统的一个基础性挑战是: 在特定的计算时间、能耗、成本约束下, 以 99 百分位的保证, 完成大数据排序计算任务. 例如, 能否在共享资源的云计算平台上, 在 99% 的情况下, 1 min 内完成 1 PB 的数据排序, 能耗低于 10 度电, 成本小于 100 美元?

4.2 计算时空与 DIP 猜想

当多个租户的多个应用在一个云计算系统上执行时, 称每个应用的执行实例 (instance) 是一个计算任务. 计算任务需要使用云计算系统的 4 种资源: 时间资源、计算资源 (如处理器)、存储资源 (如内存、闪存、硬盘)、通信资源 (如总线、网络等). 后三者称为空间资源. 这 4 种资源合起来组成一个四维空间, 称为计算时空 (cyberspacetime).

一个计算任务从开始到结束, 其使用的资源构成了计算时空中的一个子集, 称为该计算任务的相空间. 多个计算任务的执行, 对应着计算时空中的多个相空间.

DIP 猜想. 假设计算任务 A 是实用可计算的. 在给定用户体验阈值的前提下, 系统 S 能够实现 A 的实用可计算, 当且仅当 S 具备 DIP 能力: 区分 D (distinguishing)、隔离 I (isolation)、优先化 P (prioritizing). 也就是说, S 能够在运行时动态地区分、隔离、优先化 A 的相空间, 即任务 A 在 S 上执行时的计算时空中的相空间.

7) Sort benchmark home page. http://sortbenchmark.org.

区分是指系统能够自动区别出是哪个计算任务或哪个系统软件在访问资源. 隔离是指一个计算任务的相空间与别的相空间不相交. 优先化是指可以对相空间, 甚至对相空间的每一点, 赋予优先级, 以便系统按照优先级从高往低处理.

DIP 猜想关注如下科学问题: 假设计算任务 A 是实用可计算的. 那么, 什么样的云计算系统才能实现任务 A 的实用可计算性呢? "计算任务 A 是实用可计算的" 只是说明存在这样一个云计算系统, 并没有说明该系统需要具备什么样的能力 (即系统功能). DIP 猜想说, 要实现任务 A 的实用可计算性, 云计算系统需要具备 DIP 能力, 即区分 D (distinguishing)、隔离 I (isolation)、优先化 P (prioritizing).

DIP 猜想的最本质的内容是: DIP 能力是在云计算系统中限制无序影响的基础能力. 在云计算系统中, 无序行为会一直存在, 无法彻底消除. 但通过系统创新限制住无序的影响范围, 从而使计算任务的尾延迟满足用户体验阈值, 还是有可能的.

DIP 猜想受到了已有研究成功经验的启发, 包括虚拟化定理、CAP 定理、并发任务的拓扑分类、Rice 定理. 前两个定理是偏系统的结果, 后两个是纯理论的结果.

作为云计算的先驱性结果, 1974 年发表的虚拟化定理给出了计算机体系结构支持虚拟化的充分条件, 并指导了 IBM 370 大型机支持虚拟化 [33]. 这个结果在 2005 年被 Smith 与 Nair 用当代语言重新陈述 [10], 继续指导着当今的虚拟化工作. 2000 年提出 [34], 2002 年被证明的 CAP 定理 [35], 说明任何分布式系统只能同时支持一致性 (consistency)、可用性 (availability)、分区容错性 (partitioned fault tolerance) 这 3 个系统功能性质的任意两个, 但不能同时支持三者. CAP 定理广泛影响了当代互联网服务系统、云计算系统、大数据系统的设计. Herlihy 是当代拓扑分布式计算理论的创立者 [36, 37], 2004 年因此获哥德尔奖. Herlihy 与 Rajsbaum [38] 在 2003 年使用拓扑方法, 给出了 loop agreement 分布式计算任务的完全分类, 这里的计算任务事实上是考虑了问题、算法和分布式系统. 2009 年, 本文作者的团队将 Herlihy 与 Rajsbaum 的结果重新定义成二维拓扑空间的 rendezvous tasks 分类问题结果, 并将其结果推广到任意维度的 nice rendezvous tasks [39]. 这些工作的重要意义是: 可以精确地对分布式计算任务分类, 即将它们严格地区分开来.

这些结果表明, 研究分布式计算的系统能力, 即其系统功能, 不仅对现实系统的设计和创新具有指导作用, 对可计算性理论研究也可以提供新问题、新思路、新方法, 并得到正面的结果, 而不只是负面的不可能结果. DIP 猜想继承了这些工作的创新思想.

Rice 定理则从反面给出了限制. Rice 定理的一个简要版本是: 有关图灵机语言的非平凡性质是不可判定的. 即, 对任意非平凡性质 P, 语言 $L = \{\langle M\rangle : P(L(M)) = \text{True}\}$ 是不可判定的 [40]. Rice 定理要求在定义云计算系统的能力 (系统功能) 时必须小心, 以免掉进不可判定的陷阱. 有 4 个因素显示 DIP 能力可能不会掉进不可判定陷阱. 第一, DIP 猜想并没有要求必须显式地精确计算出每个任务的相空间, 再判断 D, I, P. 可能存在系统机制和功能, 使得云计算系统可以在不计算相空间的前提下, 能够实现 (enforce) DIP. 第二, 虚拟化定理与 CAP 定理已经成功地给出了分布式系统的能力刻画, 并没有掉进不可判定陷阱. 第三, Rice 定理中不可判定的性质往往是 What 类性质, 而不是 How 类性质, 即系统如何具体动作的性质, 而 "Exactly how a Turing machine M works may be decidable" [40]. D, I, P 能力都是规范云计算系统如何动作的系统功能, 因而可能是可判定的. 第四, 前期实验工作表明 (见本文 4.3 小节), 原型系统对某些基准程序展示了 DIP 能力, 进而同时保障用户体验与系统利用率.

从计算时空角度, 可以更深刻地理解虚拟化云、分区云、低熵云.

虚拟化云的特点是多个虚拟计算机共享一个云端数据中心计算机. 在一个处理器核上, 虚拟化的主要手段是时间共享 (time sharing), 又称为时间切片 (time slicing). 云端数据中心计算机也可以被划分成互不影响的机群, 多个虚拟计算机共享一个机群.

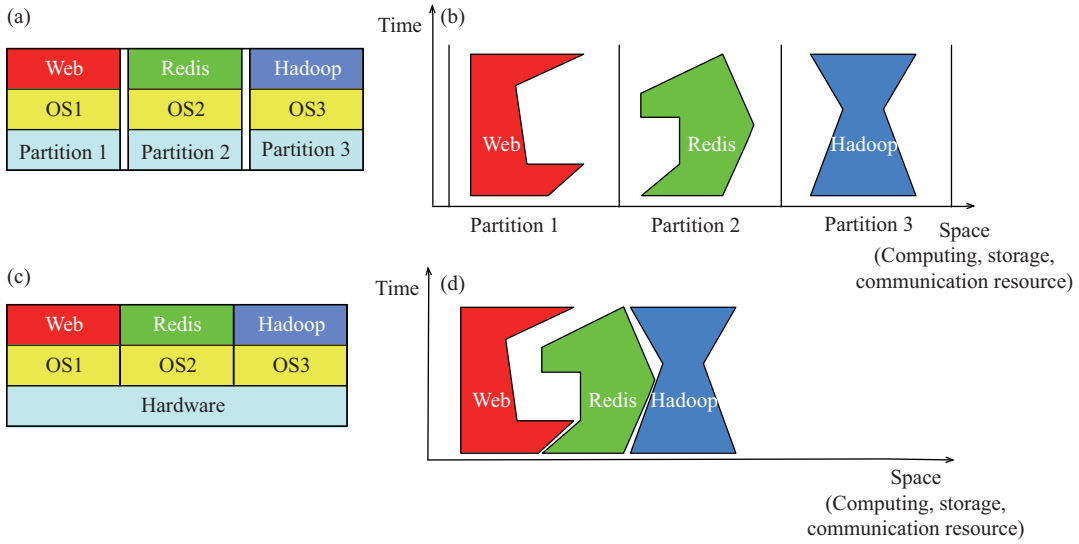


图 3 (网络版彩图) 分区云与低熵云的应用相空间比较

Figure 3 (Color online) Comparison of phase spaces in (a) (b) partitioned cloud and (c) (d) low-entropy cloud. (a) Partitioned cloud; (b) phase spaces in partitioned cloud; (c) low-entropy cloud; (d) phase spaces in low-entropy cloud

分区云的特点是一个应用独占一个机群, 主要手段是空间共享 (space sharing), 又称为空间切片 (space slicing), 即多个应用共享一个云端数据中心计算机 (空间), 但将云端数据中心计算机划分成为互不影响的机群, 每个应用独占一个机群. 应用独占的机群也称为分区.

低熵云的特点是多个应用共享一个云端数据中心计算机, 主要手段是时空共享 (spacetime sharing), 又称为时空切片 (spacetime slicing). 时空共享是指在总线周期 (bus cycle) 粒度共享资源, 有利于同时保障用户体验和资源利用率.

图 3 比较了 3 个应用在分区云和低熵云上的执行情况, 以及对应的相空间. 为了突出应用相空间, 此图忽略了 3 个操作系统的相空间. 低熵云的特点是通过由创新的硬件机制支持的细粒度弹性共享, 更加高效地共享硬件资源. 这里细粒度是指总线周期粒度.

4.3 标签化 von Neumann 体系结构

如果 DIP 猜想成立, 那么云计算系统需要具备能够在足够细粒度的情况下, 针对所有计算任务以及系统软件, 区分相空间、隔离相空间、优先化相空间. 当前的云计算系统做不到这点. 考虑最普通的情况: 两个计算任务在一个云计算节点上执行. 任务 A 与 B 共享该节点的内存系统, 都向内存发出读写请求. 但是, 系统无法区分一个内存请求是来自任务 A, 任务 B, 还是来自系统软件. 内存性能隔离就更难以做到. 优先化 (例如, 让任务 A 的访问优于任务 B 的访问) 也难以实现.

具备这些能力的一个前提, 是对计算任务 (更准确地说, 是访问资源的主体), 在计算机系统中标记它们的每一次的资源访问. 这就是标签化 von Neumann 体系结构 (labeled von Neumann architecture) 的主要特点 [4]. 图 4 显示了低熵云如何运用标签, 在总线周期粒度的级别, 区分 6 个主体对内存资源的访问. 这 6 个实体是: Web, Redis, Hadoop 3 个应用, 以及 OS1, OS2, OS3 3 个操作系统. 每一个内存请求, 在使用内存总线时, 都打上了相应主体的标签, 有利于系统区分、隔离、优先化这些内存请求. 例如, 相对于后端离线 Hadoop 应用的内存请求, 在线 Web 的内存请求可被赋予更高的优先级.

中国科学院计算技术研究所已经对这种标签化请求资源的思路做了初步的原型验证. 实验表明,

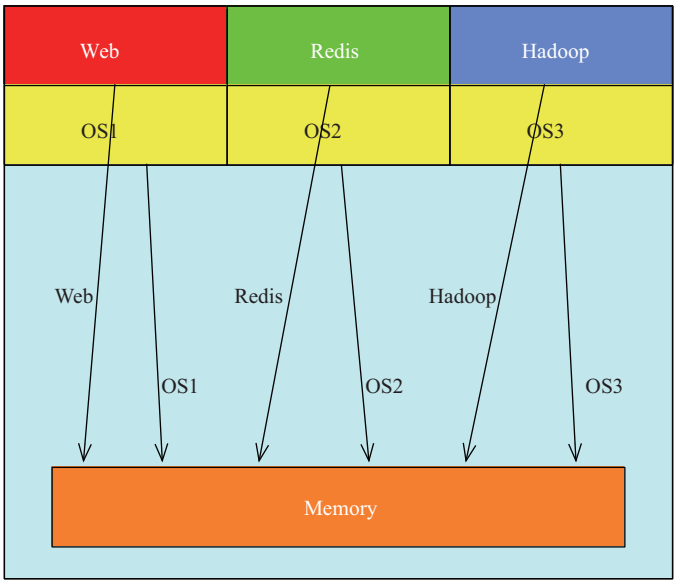


图 4 (网络版彩图) 标签化 von Neumann 体系结构之内存访问示意

Figure 4 (Color online) Accessing memory in the labeled von Neumann architecture

在保证与独占分区云相当的用户体验前提下, 采用具备 DIP 能力的标签化 von Neumann 体系结构, 低熵云可将 CPU 利用率提升至分区云的 4 倍 [8]. 另一项最新研究显示 [41], 当两个计算任务并发访问内存时, 平均分配内存块容量的简单方法可以到达最优方法性能的 74%, 是随意分配的最坏方法性能的 380%. 也就是说, 只需要可区分 (D)、可隔离 (I), 即使没有优先化 (P), 也可以明显提升性能. 在产业界, Intel 公司也推出了在处理器芯片内对高速缓存打标签的技术 Cache Allocation Technology (CAT), 其目标是实现按照处理器核, 区分对高速缓存的访问 [42,43].

4.4 适配深度学习负载的云计算协同设计技术

为了应对阻抗失配造成的无序, 中国科学院计算技术研究所提出了寒武纪神经网络处理器, 与 CPU 和 GPU 相比, 能够数量级提升速度与能效 [44,45]. 一个重要原因是处理器可以在运行时动态地重塑成为专用硬件, 适配应用层神经网络负载的需求. Google 公司最近发表了在云计算数据中心中使用其 “张量处理器” (tensor processing unit, TPU) 的性能与能效分析论文 [29].

一个挑战是让云计算用户能够方便地使用此类神经网络处理器, 并且还能尽量保持裸硬件的性能. 例如, 用户可以继续使用熟悉的 Caffe 或 TensorFlow 等上层应用框架编写 DNN, CNN, MLP, LSTM 等各种深度学习应用, 并且执行速度能够达到 80% 的裸硬件性能.

图 5 显示了在低熵云中使用寒武纪神经网络处理器的协同设计 (co-deisgn) 技术. 针对深度学习应用, 中国科学院计算技术研究所提出了 Cambricon 神经网络指令集 [46], 用户可以用这个指令集手工编程, 达到寒武纪处理器的硬件性能, 但只有汇编语言层次的可编程性. 低熵云则部署了支持 Caffe 和 TensorFlow 的深度学习处理器函数库 DLPlib [47], 它带来了两个优点: (1) 用户可以继续使用熟悉的 Caffe 或 TensorFlow 应用框架编写各种深度学习应用; (2) 应用的运行速度到达了手工编程程序速度的 79%, 也就是裸硬件性能的 79%. 这种协同设计 (co-deisgn) 技术特点是: 部署在低熵云中的 DLPlib 自动完成从 Caffe 或 TensorFlow 应用到手工程序的变换; 在 TensorFlow 对 CPU 通用处理器提供的

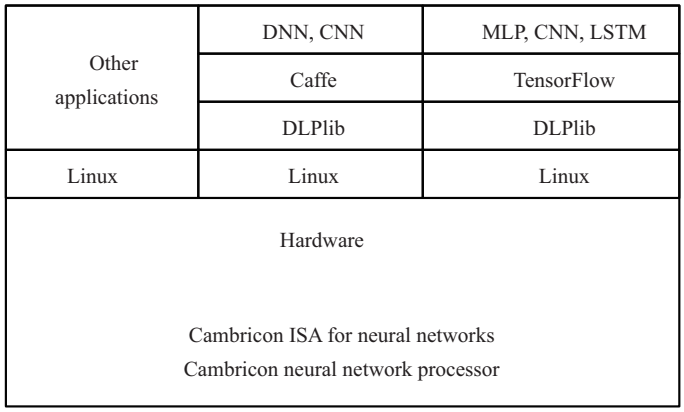


图 5　在低熵云中使用寒武纪神经网络处理器示意

Figure 5　Using the Cambricon neural network processor in low-entropy cloud

张量抽象的基础上, DLPlib 针对寒武纪神经网络处理器, 提供了 “张量 + 滤波器” (tensor+filter) 抽象, 保证了云计算用户能够得到 79% 的裸硬件性能.

5　结论

云计算系统中存在 4 类无序现象, 即负载无序、负载干扰、系统噪声和阻抗失配. 它们是当前的云计算系统 (不论是虚拟化云还是分区云) 无法同时满足用户体验和系统效率需求的重要原因. 我们将这些无序、干扰和不确定性统称为计算系统熵, 进而提出了低熵云计算系统的学术概念, 即通过降低云计算系统中的无序影响, 在满足用户体验的前提下, 提升系统效率. 低熵云吸取了虚拟化云和分区云的优点, 有潜力成为在用户体验、开发效率、运行效率、资源适配方面具有综合优势的下一代云计算系统.

图灵可计算性以及算法可计算性不足以刻画云计算服务的用户体验. 本文提出实用可计算性概念, 刻画了云计算行业的 “用户体验差的功能是不存在的功能” 的实践经验. 提出 DIP 猜想, 以刻画云计算系统能够实现实用可计算性的充分必要条件. 本文讨论了标签化 von Neumann 体系结构, 作为支持 DIP 能力的云计算系统实例.

本文讨论了学术界和产业界的前期研究以及原型系统性能评价结果, 为低熵云计算系统提供了初步论据. 本文讨论的研究正在进行中, 很多创新工作需要深入展开以及同行合作, 例如:

- 计算系统熵的精确定义, 最好是接近 Shannon 信息熵那样精确的、普适的数学定义.
- 计算时空与计算任务相空间的简洁但又足够细粒度的刻画.
- 云计算混合负载的计算任务的相空间分析. 目前已有 5 大类云计算负载: 科学计算、企业计算、互联网服务、大数据分析与机器学习.
- DIP 猜想的证明, 或者是反证与修正.
- 标签化 von Neumann 体系结构的量化定义与优化.

致谢　感谢匿名评审人对本文提出问题和建议, 他们辛勤的工作对改善本文的内容与表达有实质性的启发. 感谢中国科学院计算技术研究所软件定义云计算课题组与寒武纪团队对本文的贡献和支持.

参考文献

1 Fox A. Cloud computing — what's in it for me as a scientist? Science, 2011, 331: 406–407
2 Panitkin S. Look to the clouds and beyond. Nat Phys, 2015, 11: 373–374
3 Asanovic K, Patterson D. FireBox: a hardware building block for 2020 warehouse-scale computers. In: Proceedings of the 12th USENIX Conference on File and Storage Technologies, Santa Clara, 2014
4 Bao Y G, Wang S. Labeled von Neumann architecture for software-defined cloud. J Comput Sci Technol, 2017, 32: 219–223
5 Barroso L A, Clidaras J, Hölzle U. The Datacenter as A Computer: An Introduction to the Design of Warehouse-Scale Machines. San Rafael: Morgan & Claypool Publishers, 2013
6 Cai B L, Zhang R Q, Zhou X B, et al. Experience availability: tail-latency oriented availability in software-defined cloud computing. J Comput Sci Technol, 2017, 32: 250–257
7 Dean J, Barroso L A. The tail at scale. Commun ACM, 2013, 56: 74–80
8 Ma J Y, Sui X F, Sun N H, et al. Supporting differentiated services in computers via programmable architecture for resourcing-on-demand (PARD). ACM SIGPLAN Notice, 2015, 50: 131–143
9 Krushevskaja D, Sandler M. Understanding latency variations of black box services. In: Proceedings of the 22nd International Conference on World Wide Web, Rio de Janeiro, 2013. 703–714
10 Smith J, Nair R. Virtual Machines: Versatile Platforms for Systems and Processes. San Francisco: Morgan Kaufmann Publishers, 2005
11 Kivity A, Kamay Y, Laor D, et al. kvm: the Linux virtual machine monitor. Proc Linux symp, 2007, 1: 225–230
12 Merkel D. Docker: lightweight Linux containers for consistent development and deployment. Linux J, 2014, 239: 2
13 Delimitrou C, Kozyrakis C. Quasar: resource-efficient and QoS-aware cluster management. ACM SIGPLAN Notice, 2014, 49: 127–144
14 Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization. ACM SIGOPS Operating Syst Rev, 2003, 37: 164–177
15 Chung H, Nah Y. Performance comparison of distributed processing of large volume of data on top of xen and docker-based virtual clusters. In: Proceedings of International Conference on Database Systems for Advanced Applications. Berlin: Springer, 2017. 103–113
16 Chen T S, Guo Q, Temam O, et al. Statistical performance comparisons of computers. IEEE Trans Comput, 2015, 64: 1442–1455
17 Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM, 2008, 51: 107–113
18 Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst, 2008, 26: 1–26
19 Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing. In: Proceedings of the ACM SIGMOD International Conference on Management of data, Indianapolis, 2010. 135–146
20 Gonzalez J E, Xin R S, Dave A, et al. GraphX: graph processing in a distributed dataflow framework. In: Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation, Broomfield, 2014. 599–613
21 Huang J, Mozafari B, Wenisch T F. Statistical analysis of latency through semantic profiling. In: Proceedings of the 12th European Conference on Computer Systems, Belgrade, 2017. 64–79
22 Lu X Y, Liang F, Wang B, et al. DataMPI: extending MPI to hadoop-like big data computing. In: Proceedings of International Parallel and Distributed Processing Symposium, Phoenix, 2014. 829–838
23 DeCandia G, Hastorun D, Jampani M, et al. Dynamo: amazon's highly available key-value store. ACM SIGOPS Operating Syst Rev, 2007, 41: 205–220
24 Hindman B, Konwinski A, Zaharia M, et al. Mesos: a platform for fine-grained resource sharing in the data center. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, Boston, 2011. 295-308
25 Zaharia M, Chowdhury M, Tathagata D, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation, San Jose, 2012
26 Xu Z W, Chi X B, Xiao N. High-performance computing environment: a review of twenty years of experiments in China. National Sci Rev, 2016, 3: 36–48

27 Xu Z W. Cloud-sea computing system: towards thousand-fold improvement in performance per watt for the coming zettabyte era. J Comput Sci Technol, 2014, 29: 177–181
28 Byma S, Steffan J G, Bannazadeh H, et al. FPGAs in the cloud: booting virtualized hardware accelerators with OpenStack. In: Proceedings of the 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, Boston, 2014. 109–116
29 Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit. In: Proceedings of the 44th International Symposium on Computer Architecture, Toronto, 2017
30 Putnam A, Caulfield A M, Chung E S, et al. A reconfigurable fabric for accelerating large-scale datacenter services. ACM SIGARCH Comput Archit News, 2014, 42: 13–24
31 Caulfield A M, Chung E S, Putnam A, et al. A cloud-scale acceleration architecture. In: Proceedings of IEEE/ACM International Symposium on Microarchitecture, Taipei, 2016. 1–13
32 Nyberg C, Barclay T, Cvetanovic Z, et al. Alphasort: a cache-sensitive parallel external sort. Int J Very Large Data Bases, 1995, 4: 603–628
33 Popek G J, Goldberg R P. Formal requirements for virtualizable third generation architectures. Commun ACM, 1974, 17: 412–421
34 Brewer E A. Towards robust distributed systems. In: Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing, Portland, 2000
35 Gilbert S, Lynch N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. ACM SIGACT News, 2002, 33: 51–59
36 Herlihy M, Shavit N. The topological structure of asynchronous computability. J ACM, 1999, 46: 858–923
37 Herlihy M. Topology approach in distributed computing. In: Encyclopedia of Algorithms. Berlin: Springer, 2008. 2239–2242
38 Herlihy M, Rajsbaum S. A classification of wait-free loop agreement tasks. Theor Comput Sci, 2003, 291: 55–77
39 Liu X W, Xu Z W, Pan J Z. Classifying rendezvous tasks of arbitrary dimension. Theor Comput Sci, 2009, 410: 2162–2173
40 Rich E. Automata, Computability and Complexity: Theory and Applications. Upper Saddle River: Pearson Prentice Hall, 2008
41 Liu Y H, Sun X H. Optimizing Memory Concurrency at Each Memory Layer in a Multi-Tasking Environment. Illinois Institute of Technology Technical Report, 2015
42 Xu M, Phan L T, Choi H Y, et al. vCAT: dynamic cache management using CAT virtualization. In: Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium, Pittsburgh, 2017
43 Herdrich A, Verplanke E, Autee P, et al. Cache QoS: from concept to reality in the Intel®Xeon®processor E5-2600 v3 product family. In: Proceedings of IEEE International Symposium on High Performance Computer Architecture, Barcelona, 2016. 657–668
44 Chen Y J, Chen T S, Xu Z W, et al. DianNao family: energy-efficient hardware accelerators for machine learning. Commun ACM, 2016, 59: 105–112
45 Chen Y J, Luo T, Liu S L, et al. DaDianNao: a machine-learning supercomputer. In: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, 2014. 609–622
46 Liu S L, Du Z D, Tao J H, et al. Cambricon: an instruction set architecture for neural networks. In: Proceedings of the 43rd International Symposium on Computer Architecture, Seoul, 2016. 393–405
47 Lan H Y, Wu L Y, Zhang X, et al. DLPlib: a library for deep learning processor. J Comput Sci Technol, 2017, 32: 286–296

Low-entropy cloud computing systems

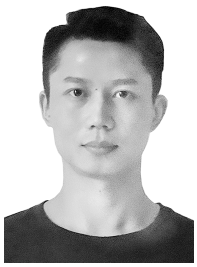
Zhiwei XU[1,2*] & Chundian LI[1,2]

1. *State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing* 100190*, China*;
2. *University of Chinese Academy of Sciences, Beijing* 100049*, China*
* Corresponding author. E-mail: zxu@ict.ac.cn

Abstract Current cloud computing systems, whether virtualization clouds or partitioned clouds, face the challenge of simultaneously satisfying user experience and system efficiency requirements. Both the industry and the academia are investigating next-generation cloud computing systems to address this problem. This paper points out a main cause of this problem: existing cloud systems have high computing system entropy (i.e., disorder and uncertainty), which manifest as four classes of disorders. We propose a new concept of "low-entropy cloud computing systems", and contrast them to virtualization clouds and partitioned clouds, in terms of user experience, application development efficiency, execution efficiency, and resource matching. We discuss four new features and techniques of low-entropy clouds: (1) a notion of production computability that, unlike Turing computability and algorithmic tractability, formalizes the user experience requirements of cloud computing practices; (2) a conjecture, named the DIP (differentiation, isolation, prioritization) conjecture, that tries to capture the necessary and sufficient conditions for a cloud computing system to realize production computability; (3) the labeled von Neumann architecture that has the potential to support the DIP capabilities and thus simultaneously satisfy user experience and system efficiency requirements; and (4) a co-design technique allowing a cloud computing system to adaptively match deep-learning workloads to neural network accelerator hardware.

Keywords cloud computing, user experience, computational efficiency, entropy, distributed computer system, computer architecture

Zhiwei XU received the Ph.D. degree from University of Southern California in 1987. He is a professor and CTO of the Institute of Computing Technology (ICT) of Chinese Academy of Sciences (CAS). His research interests include distributed systems and high-performance computer architecture.



Chundian LI is a Ph.D. candidate of the Institute of Computing Technology (ICT) of Chinese Academy of Sciences (CAS). He received his B.S. degree in software engineering from Wuhan University in 2013. His research interests include distributed system, cloud computing, programming and execution model.