数据中心保障应用服务质量面临的挑战与机遇

包云岗

（中国科学院计算技术研究所 北京 100190）

摘 要 在当今信息时代，随着移动设备、互联网应用以及云计算模式的快速发展，数据中心已成为社会基础设施。然而数据中心面临资源利用率与应用服务质量之间的矛盾，一方面通过多个应用同时在数据中心部署实现资源共享能有效提高资源利用率，另一方面多个应用共享资源又会出现相互干扰，严重影响应用的服务质量。因此，目前企业不得不采用预留额外资源以保障延迟敏感的关键应用服务质量，这导致数据中心的利用率很低。并且，随着多核技术的发展，单个服务器内的资源越来越多，其上混合部署的应用数目也在不断增加，更加剧这种矛盾。如何解决资源利用率与应用服务质量之间的矛盾，是数据中心面临的核心挑战之一，同时也为计算机系统结构研究带来很多机遇。文章主要介绍了数据中心所面临的上述矛盾以及一些研究进展，最后介绍了资源可编程体系结构 PARD（Programmable Architecture of Resourcing on-Demand）思想，从硬件上支持资源容量隔离与性能隔离，从而保障多应用混合环境下关键应用的服务质量，允许更大程度混合部署应用以提高数据中心资源利用率。

关键词 数据中心；资源利用率；服务质量；资源可编程体系结构

Challenges and Opportunities of Building Datacenters for Application's Quality-of-Service

BAO Yungang

(*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190, *China*)

Abstract Datacenters have become the fundamental infrastructure of modern internet services such as searching, social networking and online shopping. The resource utilization of typical datacenters, however, is quite low. The primary reason of such low utilization is that datacenter operators have to upload resources to online services that are latency sensitive. Co-locating multiple applications in a shared datacenter is an effective approach to improve resources utilization but can result in degradation of application's quality of service (QoS) because of interference between online services and other background workloads. Therefore, QoS guarantee method in a shared datacenter becomes the key technology for improving resourcing utilization of datacenters. In this paper, a survey of research efforts on this topic was firstly presented and then a programmable architecture for resourcing on-demand (PARD) was reviewed. The goal of PARD is to make hardware to support resources management so that the QoS of key applications can be guaranteed while multiple applications execute together in shared datacenters.

Keywords datacenter; resource utilization; Quality-of-Service (QoS); programmable architecture for resourcing on-demand (PARD)

作者简介：包云岗，副研究员，研究方向为计算机体系结构，E-mail：baoyg@ict.ac.cn。

1 引 言

1.1 数据中心成为基础设施

互联网应用如电子邮件、搜索、网络购物、社交网络、在线视频、网络地图等已经成为人们生活的一部分。这些应用往往要为上亿用户服务，意味着互联网应用已变成如电力一样的社会公共服务，而支撑拥有海量用户互联网应用的数据中心也成为如同发电厂一样的社会核心基础设施。

典型的数据中心一般由 5 万～10 万台中低端服务器组成，这些服务器通过内部网络互连，一起协同运行，为海量用户服务。因为这类数据中心规模很大，往往部署在大型仓库级别的机房，从应用角度来看就如同一台计算机，因此也被称为“仓库级计算机(Warehouse-scale Computer)”[1]。国内外著名的互联网公司往往拥有多个数据中心，服务器数量达到数十万甚至上百万台。例如，谷歌(Google)公司的数据中心服务器数量已经超过百万台，为全球用户提供搜索、邮件、地图等服务[2]；亚马逊(Amazon)公司仅 EC2 就部署了约 50 万台服务器，用以提供云计算服务[3]；据可靠消息，国内腾讯公司也拥有约 30 万台服务器，为用户提供各种互联网服务。

尽管目前互联网企业的数据中心已经颇具规模，但一个趋势是未来数据中心还将持续发展。一方面，互联网用户数量仍在不断增长，目前全球已有 24 亿网络用户，很多机构预测未来全球还将新增 30 亿网民融入互联网[4]，这对数据中心的数量和规模的需求都将加大。另一方面，快速发展的移动终端已超越个人计算机(PC)成为终端计算设备的主流。由于移动设备性能相对较低、存储容量较小，将计算与存储转移到数据中心的需求也变得越来越强烈。在这些需求的推动下，不仅企业不断扩大其数据中心规模，各地政府也开始建立各自的云计算中心(如山东云计算中心)，将一些政务、税收等数据逐步迁移到数据中心。因此数据中心作为基础设施的重要性也会日益凸显，其地位甚至可能上升到国家战略层面。

1.2 数据中心设计与运营维护的核心理念

低成本是数据中心设计的最核心理念，这是由互联网时代的商业模式所决定的——很多互联网公司的服务是免费的，通过免费服务吸引用户以获取广告收入。虽然设计原则是低成本，但因为数据中心规模巨大，典型的 5 万～10 万台服务器的数据中心成本高达到 10 亿人民币(1.5 亿美元)[5]。对于拥有数十万台服务器的互联网公司，其数据中心成本会达成几十亿甚至上百亿人民币。比如京东最近宣布在内蒙古自治区巴彦淖尔市和江苏省宿迁市建立云计算中心，每个中心容纳 10 万～15 万台服务器，投资总额高达 40 亿元。

因而，如何降低成本、提高资源利用率是数据中心的设计、运行与维护(运维)的核心目标。一般而言，主要有两种技术路线来降低成本：

(1)综合考虑性能、功耗与成本，选择性价比最高的服务器。

以运行数据库基准测试集 TPC-C 为例，为了提供更大容量的内存、可靠性等，高端服务器处理事务的单价是低端服务器的 20 倍[1]。Google 等互联网公司都是采购低端服务器。为了进一步节省软件成本，企业甚至会定制或开发自己的系统软件栈，包括操作系统、存储系统、数据库等。如，Google 公司的 BigTable、MapReduce，国内淘宝公司的 TFS 文件系统，腾讯公司的存储系统等。

(2)采用资源共享的方式提高数据中心资源利用率。对于价值几十上百亿规模的数据中心，即使提高 1% 的资源利用率也能带来非常可观的效益[6]。资源共享是提高利用率最有效的方式。以 Google 公司的数据中心为例，其中同时运行了各种应用，如 Gmail 邮件服务、街景地图(Street View)、科学计算等。最近哥伦比亚大学的研究人员对 Google 公司的数据中心进行分析后，发现有超过 1100 个应用同时在一个

表 1 服务响应时间对用户行为、企业收入的影响[8]

Server delay (ms)	Increased time to next click(ms)	Queries/user	Any clicks/user	User satisfaction	Revenue/user
50	--	--	--	--	--
200	500	--	−0.3%	−0.4%	--
500	1200	--	−1.0%	−0.9%	−1.2%
1000	1900	−0.7%	−1.9%	−1.6%	−2.8%
2000	3100	−1.8%	−4.4%	−3.8%	−4.3%

数据中心内部运行[6]。除了企业内部应用共享数据中心，将资源以某种形式出租给用户也是一种共享模式，即云计算模式。这种云计算模式不仅能节省成本，还开辟了新的商业模式，能为企业带来巨大的收益。如 Amazon EC2 数据中心约有 50 万台服务器，2013 年度预计营业额已达到 38 亿美元[7]，远高于 50 台服务器的成本。

总结而言，为了追求低成本目标，数据中心在设计阶段采用性价比高的硬件设备来降低采购成本与管理成本，在运营与维护时则采用资源共享方式提高资源利用率。

1.3 应用服务质量(QoS)成为制约数据中心资源利用率的关键因素

尽管通过资源共享方式能显著提高数据中心利用率，但共享同时也会带来应用之间相互干扰问题，这种干扰会严重影响许多延迟敏感型应用的服务质量（Quality of Service，QoS），如搜索、在线购物等。

1.3.1 服务质量是互联网应用的关键指标

活跃用户数与用户访问量是影响互联网公司营收的主要因素。互联网公司一般都采用免费服务吸引用户，快速的服务响应时间是让用户满意、留住用户的关键，也是衡量服务质量的关键。有研究表明，如果服务响应时间增加，公司收入就会减少。2009 年微软在 Bing 搜索引擎上开展测试服务响应时间对收入影响的实验[8]。如表 1 所示，当服务响应时间为 200 ms 时，用户点击下一个链接的时间也变长，用户的点击数与满意度都在下降。当服务响应时间增加到 2000 ms 时，用户满意度下降了 3.8%，而每个用户带给企业的收益下降了 4.3%。由于这个实验结果对公司产生了负面影响，最终不得不永久终止实验[8]。

由此可见，应用服务质量对互联网企业以及云计算中心都至关重要。事实上，应用服务质量也正是很多互联网企业对数据中心运维工程师业绩考核的核心指标之一。为了让用户有更好的体验，许多公司采用各种技术减少服务响应时间，提高服务质量，例如 Google 公司甚至提出千兆网入户方案。

1.3.2 应用服务质量与资源利用率之间的矛盾

如前所述，资源共享方式能提高数据中心利用率，但我们的一些实验数据初步揭示了“资源共享会影响应用服务质量”的现象。如图 1 所示，在一台部署了 Web 服务的机器上，当同时访问 events.php 网页的用户数从 600 增加到 1200 时，CPU 利用率从 10% 增加到 20%。对比相应的请求响应时间分布曲线（红色与绿色），虽然请求的平均响应时间并没有明显增加，但在 1200 个用户时出现了明显的长尾延迟现象（Long Tail Latency），即曲线右侧比重增加。

在数据中心中，各种不同类型的应用会混合部署在一起。为了模拟数据中心环境，我们在 1200 个用户的基础上又运行一个 MapReduce 数据分析应用。此时 CPU 利用率提高到 100%，但是 web 服务的请求响应时间的长尾延迟更加显著（蓝色曲线）。图 1 右边显示了对 addEventResult.php 网页的访问的实验数据，因为涉及数据库与磁盘 I/O 操作，干扰也变得更加严重。从上述实验可见：资源利用率与应用服务质量之间存在矛盾——资源利用率提高却导致应用服务质量下降，响应时间出现长尾延迟现象。另一个角度而言，为了保障服务质量，不得不限制资源利用率过高，亦即“应用服务质量会制约资源利用率的提高”。

1.3.3 数据中心会放大长尾延迟现象、降低服务质量

更严重的是，上述长尾延迟现象在数据中心环境下会被进一步放大。这是因为互联网应用已经被分解

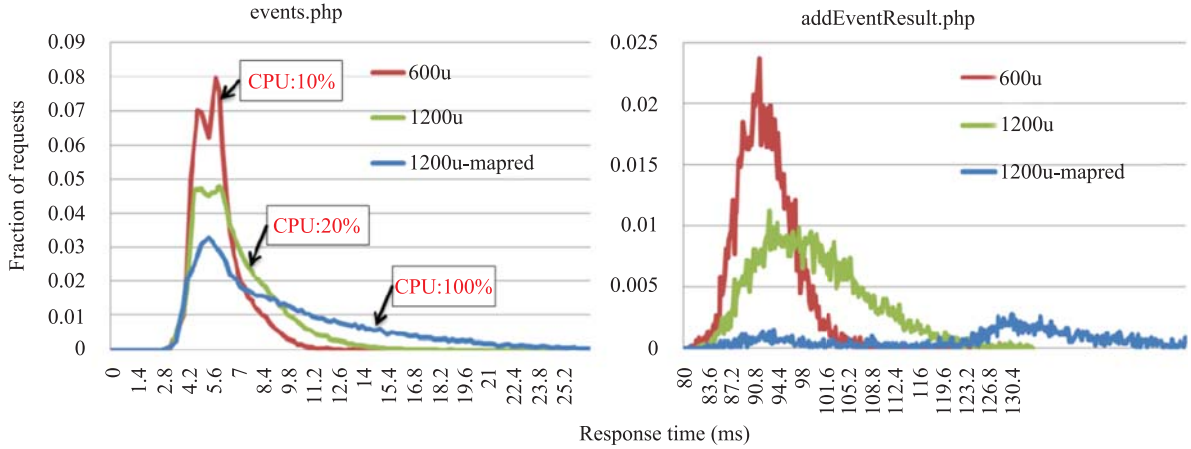

图 1　Web Server 请求响应时间在共享环境下的变化

为许多服务，这些服务部署到数据中心的大规模服务器上。一般而言，服务之间的耦合方式有两种[9]，但两种模式都会放大长尾延迟现象：

(1)划分/聚合模式(Partition/Aggregate)。有些互联网应用如 Google、百度搜索，每个用户请求涉及 TB 级别甚至更大规模的数据访问，这些数据分布存储在成千上万台服务器。因此完成一个请求需要访问上千台机器，具有很大的扇出(fan-out)。这种场景下，请求的响应时间取决于最慢的那台服务器。Google 公司的 Jeff Dean 在 2012 年 Berkeley 的报告[10]中就指出了长尾现象的严重性。假设一台机器处理请求的平均响应时间为 1 ms，有 1% 的请求处理时间会大于 1 s(99th-Percentile)。如果一个请求需要由 100 个这样的节点一起处理，那么就会出现 63% 的请求响应时间大于 1 s。

(2)依赖/串行模式(Dependent/Sequential)。这些互联网应用代表是 Facebook 社交网络，Amazon 和淘宝的在线购物等，这些应用处理用户请求需要依次访问一系列有依赖的服务，才能最终得到结果返回给用户。比如，淘宝的在线购物应用接收到一个用户请求，会先获取用户的个人信息，然后得到购物历史记录，再根据历史记录调用推荐系统等。在这种模式下，服务都在关键路径上，因此每个服务的处理延迟会累加。正是由于这种累加效应，Facebook 限制每个网页的处理不能访问超过 150 个服务[9]。

这两种模式都会导致处理一个请求需要访问多台服务器，因而只要有一台服务器的处理速度受到干扰，就会导致整个请求的处理时间增加。由于一个请求处理的全生命周期里会涉及成百上千台服务器，因此如何保障请求全生命周期的服务质量成为数据中心环境下的新挑战。

1.4 实例分析：Google 数据中心以及其他互联网企业的现状

Google 公司的数据中心技术一直处于领先地位，我们以 Google 为例分析其数据中心资源利用率与应用服务质量的现状。图 2 显示了 2006 年 Google 数据中心 5000 台服务器 6 个月的 CPU 利用率统计[1]，其资源利用率并不高，整个数据中心的平均 CPU 利用率仅为 30% 左右。此时较低的资源利用率主要由于单个服务器资源有限，单个应用就会占有大部分资源，导致无法满足其他应用需求，即所谓的“装包问题(Bin-Packing)”[1]：因为包太小而只能装一样物品。在这种场景下，实际上每台服务器处于独占状态，鲜有应用间干扰现象。

随着处理器芯片内的核数目不断增加，单服务器节点内的核数目也在增加。如，2012 年 Google 的单服务器节点已有 12 个核、24 线程(启用 Intel 超线程技术)[6]。单节点已有足够资源，可有效地缓解资源分配时遇到的装包问题，在不考虑服务质量的情况下，可以将多个应用部署到同一个节点以提高资源利用率。如图 3 所示，2012 年 Google 的 1000 台服务器平均已能达到 50% 以上的 CPU 利用率，比 2006 年有显著提高，但却带来了严峻的应用服务质量问题。而图 4(b)显示，2013 年 1～3 月 Google 数据中心 2 万台离线作业服务器的 CPU 利用率高达 75%。

但是，长尾延迟导致的服务质量问题依然没有解决。Google 的 Jeff Dean 与 Luiz Barroso 在 2013 年 2 月的《Communication of the ACM》上就专门撰文

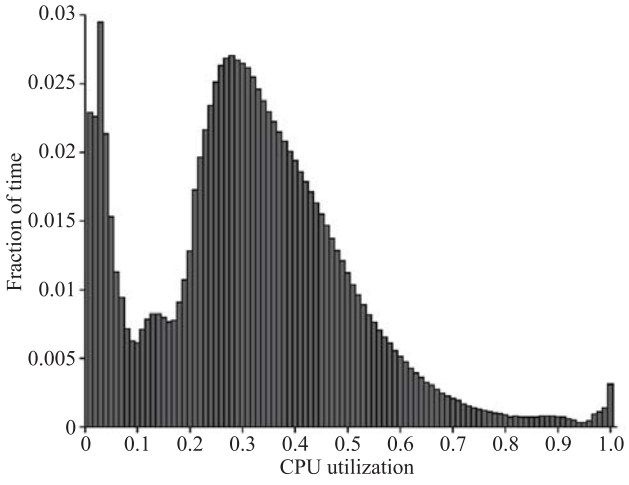

图 2 2006 年 Google 数据中心 5000 台服务器 6 个月的 CPU 利用率分布[1]

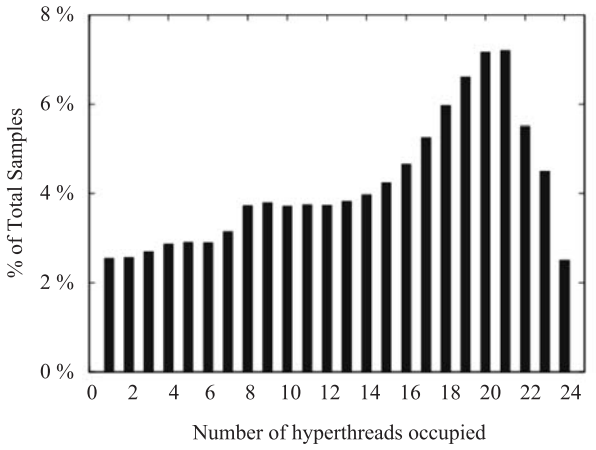

图 3 2012 年 Google 数据中心 1000 台 24 线程服务器的 CPU 利用率[6]

“The Tail at Scale”[11]，阐述了在数据中心的长尾延迟问题。他们分析了导致长尾延迟的首要原因是资源共享，包括体系结构层次的 CPU 核、Cache、访存带宽、网络带宽等，而干扰不仅来自应用，还来自系统软件层次的后台守护作业、监控作业、共享文件系统等。

Dean 与 Barroso 在参考文献[11]中介绍了 Google 采用的缓解长尾延迟的技术，包括操作系统容器隔离技术[12]、应用优先级管理[13]、备份请求[10]、同步后台管理进程[10]等，取得了一定的效果，但仍无法避免资源共享对应用服务质量的负面影响。如图 5 所示，一些延迟敏感型应用(如街景地图 Streetview)的性能仍有较大幅度的波动[6]。因此，在实际部署中，对于那些关键的在线应用，CPU 利用率仍然只有 30% 左右(见图 4(a))。

对于其他互联网企业，在资源利用率与服务质量之间不得不二选一。为了保障应用服务质量，很多企业放弃将多个应用混合部署的方案。据调研，国内主流互联网企业的在线应用都采用独占数据中心的方式，其数据中心资源利用率一般在 10%～30%，与 Google 的 50%～60% 利用率相比仍有不少的差距。相反，Amazon EC2 为了提高数据中心资源利用率，在其条款上直接声明不向用户承诺服务质量。因此有研究发现相同的程序在 Amazon EC2 上运行多次，性能波动幅度会比本地机群高一个数量级[15]，性能不可预测。

技术发展的趋势是处理器内部的核数目仍会不断增加，一些研究人员甚至已经开始关注片上数据中心(On-Chip Datacenters)[16]。因此单节点内资源将越来越多，意味着必然会有更多应用同时在一个节点内运行，干扰会越来越严重，应用性能波动幅度也会随之不断扩大，严重影响应用服务质量。而资源利用率与服务质量关系密切，以单机为例，虚拟化技术能有效地解决应用之间的资源隔离(但仍未解决性能隔离问题)，使多个虚拟机能部署到一台机器，实现物理资源共享，提高利用率。因此，如果数据中心环境下无法做到性能隔离以保障服务质量，最终很可能不得不采用限制数据中心资源利用率以避免服务质量的不可控。我们预计：保障应用服务质量将成为数据中心核心技术之一。而软件定义网络(Software Defined Networking)兴起正印证这个趋势。

2　国内外相关工作

为了保障数据中心全生命周期服务质量，需要从服务器节点内部、服务器之间通信以及分布式架构多个层次协同工作。近年来，学术界、工业界在这三

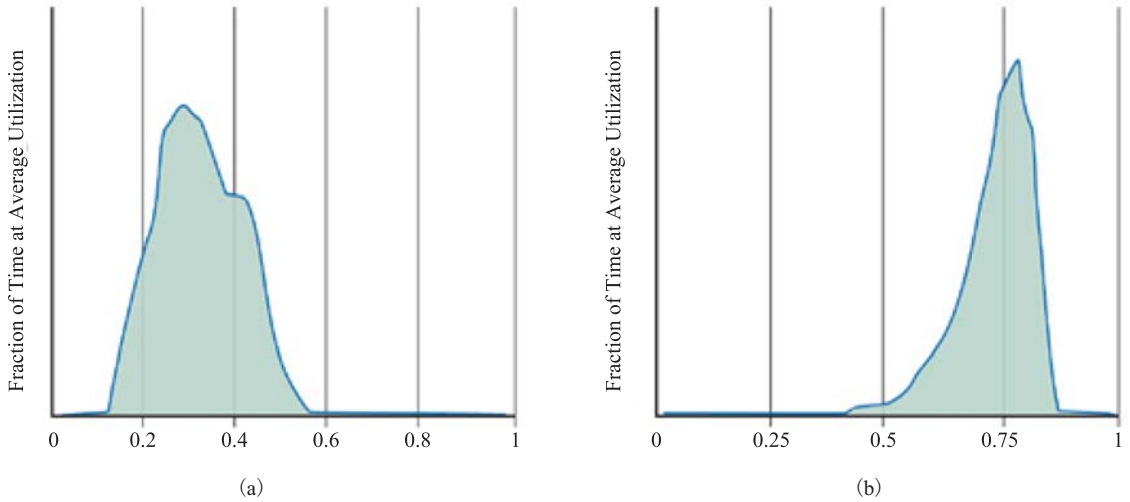

图 4　Google 数据中心 2013 年 1 至 3 月运行在线应用的 CPU 利用率平均为 30%(a)，离线作业数据中心的 CPU 利用率为 75%(b)[14]

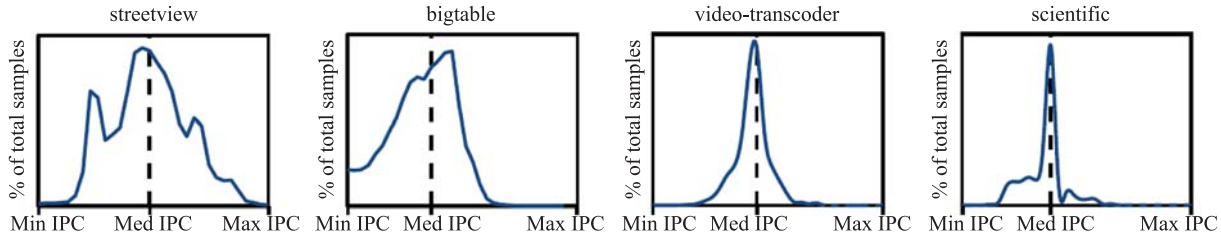

图 5　2012 年 Google 数据中心应用性能存在波动性[6]

个层次都不断努力。如近年流行的 Software Defined Networking(SDN)技术[17]，其目标就是为了解决数据中心网络通信的管理、共享与性能隔离问题。Google 在分布式架构上采用了超时发送备份请求同步后台管理进程等技术[18]对改善“划分/聚合”模式应用的长尾延迟有帮助，但并不能显著改善“依赖/串行”模式应用的响应时间。此外，单节点内服务质量保障技术已成为短板。现在主流的软件隔离技术能对资源容量隔离起到较好的效果，但无法保障性能隔离，无法保障服务质量。

总的来说，国内外在保障应用服务质量方面的研究包括单节点与分布式环境两个方面。以下分别介绍这两个方面的相关工作。

2.1 单机保障服务质量研究

单机保障服务质量研究的相关技术包括软件资源隔离技术、软硬件调度技术、硬件支持等。

软件资源隔离技术：针对数据中心里多个应用相互干扰的问题，一般采取虚拟化手段在资源共享的条件下保障资源隔离，如虚拟机技术 Xen[19]或 Linux Container[20]。

传统虚拟机技术通过将多台虚拟机 VM 部署到物理机上，每台虚拟机运行一个应用或应用的一个组件，每个应用在自己的操作系统环境中独立运行，减少相互干扰。但这些隔离主要是资源的隔离，而无法实现性能隔离，如，为不同应用分配不同的访存带宽。并且，使用虚拟机也会带来一定的性能损失[21]，增加延迟，带来性能波动。

Linux Container[20]是一种轻量级虚拟化，通过在操作系统层面增添虚拟服务器功能，操作系统内核能够提供多个互相独立的用户态实例。每个用户态实例对于它自己的用户来说都像是一台独立的计算机，有自己独立的网络、文件系统、库函数和系统设置等。操作系统级虚拟化技术的优点是性能开销较小，不需要硬件的特别支持，而且能为用户态实例之间提供一定的隔离性，所以被广泛地应用在虚拟主机服务器环境中。然而，容器虚拟化技术的虚拟对象不是实际的物理资源(处理器、内存和外设)，而是从用户角度出发的抽象操作系统内部资源，如 CPU 时间、内存、I/O 带宽等。在性能隔离方面也有相关研究，Rice 大学的 Druschel 等人[22]设计了 Resource Container 系统，实现了单台物理机上多个应用间的性能隔离和 CPU 细粒度资源分配机制支持。然而局部隔离并不能保证全局隔离，Druschel 等人又设计了 Cluster Container 系统[23]，以解决应用在集群范围内的隔离问题。但十几年前单机核数目非常小，如今单节点已经有几十个核，同时运行的应用也增加了一个数量级。

页着色(Page Coloring)[24]是一种以软件方式控制内存物理页映射到处理器缓存上的技术，映射到同一缓存块中的物理页对应同一颜色。基于页着色技术可以实现对共享二级缓存的划分(Partition)[25]，能缓解应用在共享二级缓存上的干扰。Cho 等人[26]使用 Page Coloring 技术来管理共享缓存，Tam 等人[27]在 Linux 内核上实现了基于页面着色的缓存划分策略。而对于 DRAM 系统，也可以使用页着色技术对共享 DRAM 颗粒进行划分[28-30]，Liu 等人[31]在 Linux 内核上实现了基于页着色的 DRAM 颗粒划分。北京大学李晓明教授团队也在这方面做了很多工作[32-34]，研究利用页着色技术在虚拟环境下对共享 Cache 进行了划分。页着色技术能缓解一些粗粒度共享资源层次的干扰，但无法解决微体系结构的干扰，比如共享队列等，并且使用不灵活。总结而言，软件隔离技术能对资源容量隔离起到较好的效果，但无法保障性能隔离，无法保障服务质量。

软硬件调度技术：在多核微体系结构上，由于共享片上和片外资源的竞争会引起跨核应用之间的干扰。Mars 和 Vachharajani 等人提出了竞争感知的轻量级运行时环境 CAER[35]，能在提高利用率的同时减少由竞争引起的跨核干扰问题。他还和 Tang 等人在参考文献[36]中介绍了 CiPE 框架，可以直接用于测量和量化多核结构下应用的跨核干扰敏感度。Mars 等人还设计了 Bubble-Up[37]机制，通过使用气泡(Bubble)来代表内存子系统的可变压力情况，能准确预测在内存子系统中竞争共享资源而导致的性能下降。Tang 等人在参考文献[38]中提出了一种动静结合的编译方法 ReQos，在确保高优先级应用的服务质量的同时让低优先级应用也可以自适应地执行。ReQoS 包含一种由配置文件引导的编译技术，来识别低优先级应用中有争议的代码段。这些调度相关的工作在具有大规模真实应用的 Google“混布”数据中心里，使用该机制能在保证延迟敏感性应用的服务质量的同时，能显著提高 50%～90% 的资源利用率。但这些工作属于 Ad-hoc 类型，针对特定场景有效，并没有从根本上解决问题。

以 CMU 的 Onur Mutlu 为代表的一些学术界专家提出了一系列调度算法[39-41]以缓解内存控制器的不公平问题，从而提高系统吞吐量以及服务质量。但这些

算法是固化的，并不能针对某个应用进行调节，不具有灵活性。

体系结构支持：Iyer 在参考文献[42]中提出了一种保障 CMP 体系结构上缓存 Qos 的管理框架，设计了 CQos 优先级分类、优先级分配和优先级执行。CQos 优先级分类和优先级分配采用的是从用户到开发人员驱动的编译检测和基于流的方法。CQos 优先级执行则包括(1)选择高速缓存分配；(2)动静态结合设置分区；(3)异构缓存区域。实验结果表明，CQoS 在多线程或多核平台上能提高共享缓存的效率和系统性能。然而，Iyer 并没有详细描述保障 CMP 体系结构上缓存 Qos 的具体策略和软硬件支持。在参考文献[43]中，Iyer 等人又再实现了一种在 CMP 平台上保障 Qos 的内存体系结构，允运行时的动态资源再分配，能在减少低优先级应用性能下降的同时优化高优先级应用的性能。Herdrich 等人[44]证明了用于功耗管理的基于速率(rate-based)技术能适应于 CMP 结构上缓存/内存的 Qos 管理，其基本方法是当正在运行的低优先级任务由于资源争用而干扰了高优先级任务的性能时，就减缓核心的处理速率。通过评估时钟调制和频率缩放这两个速率限制机制，发现时钟调制更适用于缓存/内存 Qos 管理。

Iyer 在体系结构支持服务质量方面做了一些有价值的工作，但主要集中在内存方面，并没有从整个系统角度去考虑。我们认为这个方向在未来会越来越重要，值得深入研究。

2.2　分布式环境保障服务质量

在分布式环境下，影响应用服务质量的因素主要是节点故障与干扰引起的长尾延迟，下面将从这两个方面介绍相关优化技术。

软硬件故障：Dean 等人设计 MapReduce[45]的初衷是使用由成百上千机器组成的集群来处理超大规模数据，所以，要求 MapReduce 必须能很好地处理机器故障。MapReduce 采用了任务重新调度或重新执行任务(backup task)的方法来解决节点故障或短暂忙碌。如果一个机器的硬盘出了问题，读取数据的速度从 30 M/s 降低到 1 M/s，MapReduce 框架中将发送 backup task 机制来减少这一类长尾延迟。Backup task 机制通常只会占用比正常操作多几个百分点的计算资源，但能显著改善因为故障出现的长尾延迟。类似于 TCP 重传机制，backup task 的有效性会随着负载的提高而削弱。

竞争共享资源引起的干扰：为了缓解干扰引起的长尾延迟现象，Dean 等人[11]介绍了 Google 采用的缓解长尾延迟的技术，包括操作系统容器隔离技术[12]、应用优先级管理[13]、备份请求[10]、同步后台管理进程[10]等。Kapoor 等人在参考文献[46]中提出了 Chronos 架构，以降低数据中心应用的长尾延迟。Chronos 基于 NIC 上应用层数据包头字段的请求划分、应用实例负载均衡和 NIC 负载均衡模块的加载来消除关键通信路径上的共享资源，如内核和网络协议栈，以减少应用延迟以及相关干扰。

这些研究工作从分布式架构上一定程度上缓解了“划分/聚合”模式应用的长尾延迟，但对“依赖/串行”模式应用并不显著。另一方面，随着单个服务器节点的核数目不断增加，甚至未来“片上数据中心”[16]出现，单节点内同时运行的应用也会增加，那么干扰将会越来越严重，仅仅依赖分布式架构以及软件方法将无法保障性能隔离。如何从硬件上支持服务质量保障技术将是一个值得关注与研究的方向。

2.3　小　结

从现有技术来看，单节点内服务质量保障技术的不足，导致节点内应用相互干扰严重，这某种程度上已成为目前数据中心整体服务质量保障的短板，是成为长尾延迟现象的主要因素之一。同时这也是一个非常具有挑战的问题，需要跨层次协同设计。美国计算共同委员会(Computing Community Consortium)于 2012 年 5 月发布的计算机体系结构共同体白皮书《21 世纪计算机体系结构》中也将单节点内保障服务质量作为未来研究方向之一，其中认为[19]：“管理应用之间的相互作用也带来了挑战。例如，这些应用如何表达服务质量(QoS)目标并且让底层的硬件、操作系统以及虚拟层共同工作来保障它们。”

另一方面，数据中心应用涉及成百上千个节点，保障节点内应用服务质量仅仅是基础，而如何在数据中心环境下保障应用全局服务质量也非常具有挑战，需要在数据中心分布式环境下大规模节点协同保障才能实现，从分布式理论到系统设计多个层次均需考虑。

3　PARD：资源可编程体系结构

针对上述数据中心面临的挑战，中国科学院计算技术研究所(中科院计算所)计算机体系结构国家重点实验室已经开展了如何在数据中心环境下保障应用服务质量的相关研究。

首先，我们认同计算机体系结构共同体白皮书《21 世纪计算机体系结构》中的观点——要实现性能隔离以保障应用服务质量，需要软硬件协同工作。数据中心互连网络技术已经在资源管理、性能隔离、服务质量等方面开展了一系列研究，软件定义网络 SDN（Software Defined Networking）。能有效地解决不同网络流之间的性能隔离，保障网络服务质量。SDN 的主要思想是：（1）网络数据包附上流标识（flowid）；（2）将网络路由器的数据面（Data Plane）与控制面（Control Plane）分离；（3）数据面只负责转发请求，控制面可由软件编程，这样实现灵活高效地网络管理。

受 SDN 的启发，如图 6 所示，我们可以将单节点看作是一个小网络，内存控制器、I/O 控制器等看作是网络交换机，负责转发处理器核、内存和 I/O 设备之间的请求与数据包；运行在单节点上的多应用同时产生的请求，可视为不同的网络流。然而，在计算机节点内部，许多资源仍然处于没有管理的共享状态，比如片内共享缓存、共享内存控制器、共享访存带宽等。

我们借鉴 SDN，提出了资源可编程体系结构——PARD（Programmable Architecture of Resourcing on-Demand）。从某种角度来看，PARD 结构属于一种软件定义体系结构 SDA（Software Defined Architecture）的范畴，是 SDA 的一个具体实现方案，其核心思想是：

（1）计算机内的请求包与数据包附上标签；

（2）将主要控制器（如内存控制器）的数据面与控制面分离；

（3）数据面负责转发请求与数据，控制面可由软件编程，根据标签实现资源分配、性能隔离。

PARD 技术希望通过硬件支持资源管理与软件可编程相结合的方式，实现更高效灵活的资源共享与性能隔离。如果将计算机内部的资源类比于城市交通系统。无 PARD 技术支持的计算机系统就如城市交通不设多个车道、不设红路灯、不设交通规则，这会造成大量的冲突和混乱，也会导致关键车辆（如救护车、消防车等）通行无法得到保障。而 PARD 技术的目标，正如交通管理系统能在保障救护车等关键车辆顺利通过的前提下提高道路利用率，针对多种应用混合的数据中心，能在保障关键应用的服务质量前提下提高资源利用率。

如图 7 所示，以内存控制器为例，我们将内存控制器分为控制面与数据面。其中数据面负责传统的包转发，和内存芯片交互；控制面则负责资源管理与控制，包括 QoS 优先级、内存容量、带宽限制等。这些资源的管理与控制可由控制面的编程接口来访问。所

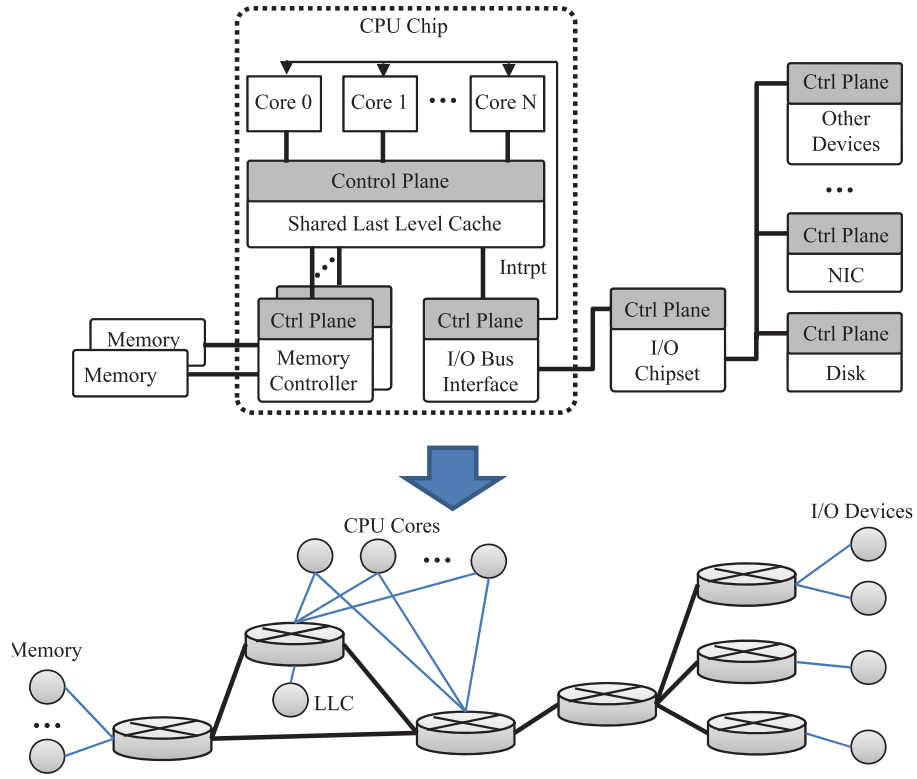

图 6 从网络互连角度看计算机结构

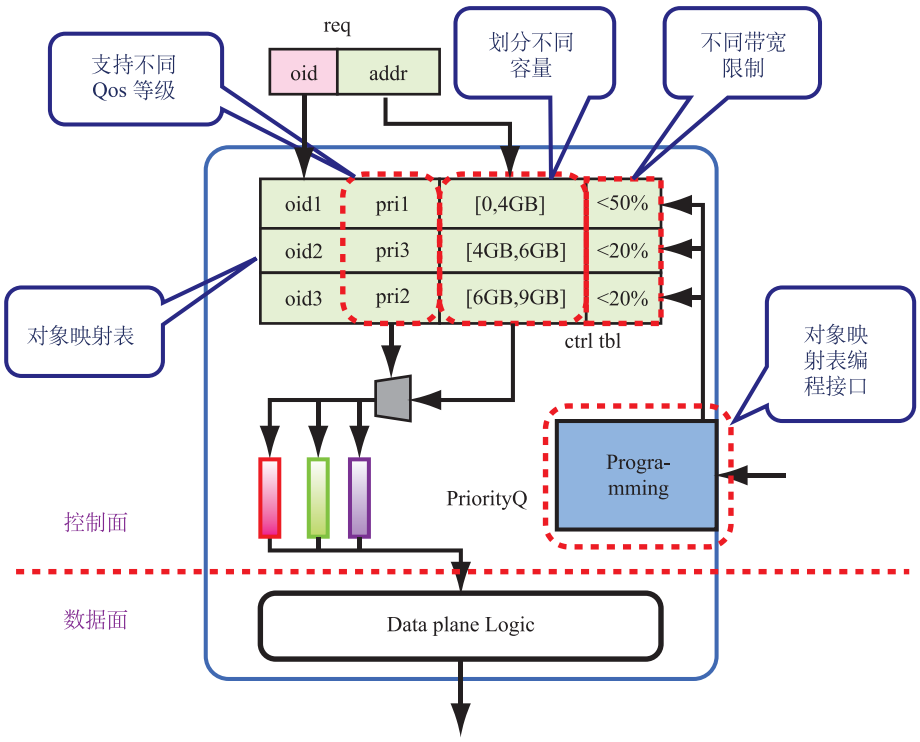

图 7　内存控制器图

有访存请求将赋予对象标签 oid(Object id)，控制面中有一张控制表(Control Table)存储基于 oid 索引的控制信息。因此，针对每一个访存请求，控制面中逻辑可以根据该请求的 oid 获取对应的控制信息，进而做权限审查、带宽控制等。软件方面，控制面提供编程接口给软件，操作系统可以通过编程接口对控制表进行修改。

目前本课题组已经初步完成了 Cache 控制器、内存控制器的设计与实现，并在模拟器平台成功加载操作系统、启动应用程序，使操作系统能根据应用需求来控制执行优先级以获取相应的资源。同时课题组也在基于 OpenSPARC 的 FPGA 仿真平台上进一步验证 PARD 技术。

4　总　结

数据中心相关产业在全球已经达到 1500 多亿美元，并且在未来依然有很大的增长空间。Google 公司代表了世界领先的数据中心技术水平，但是依然面临着资源利用率与应用服务质量之间的矛盾。数据显示，2006 年在线应用数据中心 CPU 利用率仅为 30%，但到 2013 年依然没有大的提高。对于动辄价值上十亿美元的数据中心，即使提供 1% 的效率也有很客观的收益。要解决资源利用率与应用服务质量之间的矛盾，需要计算机软硬件协同设计，这也是计算机系统结构研究的新机遇。

本课题组针对数据中心面临的上述关键挑战，提出的资源可编程体系结构 PARD 技术。PARD 技术提供一种新的体系结构平台。传统计算机软件与硬件交互的界面是指令集，一方面软件基于指令集对低层结构进行控制，另一方面指令集为软件开发人员屏蔽了低层结构细节。但随着底层结构越来越复杂，仅仅通过指令集已无法充分发挥低层结构的效率，亟需一种新的软硬件交互模式。而 PARD 提供了一个新的交互界面，当然，这其中带来了很多新的问题，如交互效率、安全问题、分布式环境下扩展性等。目前本课题组正对基于模拟器与 FPGA 仿真平台开展概念验证工作，已取得初步进展，希望能与国内工业界、学术界的同行们进行更多的交流合作。

参 考 文 献

[1] Hoelzle U, Barroso LA. The Datacenter as a Computer: an Introduction to the Design of Warehouse-scale Machines [M]. Morgan and Claypool Publishers, 2009.

[2] Google throws open doors to its top-secret data center [EB/OL]. http://www.wired.com/wiredenterprise/2012/10/ff-inside-google-data-center/all/.

[3] Steven J, Nichols V. Amazon EC2 cloud is made up of almost half-a-million Linux servers [EB/OL]. [2012-03-16]. http://

www.zdnet.com/blog/open-source/amazon-ec2-cloud-is-made-up-of-almost-half-a-million-linux-servers/10620.

[4] Diamandis P, Kotler S. Abundance: The Future is Better Than You Think [M]. Free Press, 2012.

[5] Patterson D, Hennessy J. Computer Architecture: A Quantitative Approach [M]. Morgan Kaufmann, 2011.

[6] Kambadur M, Moseley T, Hank T, et al. Measuring interference between live datacenter applications [C] // Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, 2011.

[7] Dignan L. Amazon's AWS: $3.8 billion revenue in 2013 [EB/OL]. [2013-01-07]. http://www.zdnet.com/amazons-aws-3-8-billion-revenue-in-2013-says-analyst-7000009461/

[8] Schurman E, Brutlag J. The user and business impact of server delays [C] // Proceedings of Velocity: Web Performance and Operations Conference, 2009.

[9] Kapoor R, Porter G, Tewari M, et al. Chronos: predictable low latency for data centerapplications [C] // Proceedings of the 3rd ACM Symposium on Cloud Computing, 2012.

[10] Dean J. Achieving Rapid Response Times in Large Online Services [Z]. Berkeley, 2012.

[11] Dean J, Barroso L. The tail at scale [J]. Communication of the ACM, 2013, 56(2): 74-80.

[12] Cgroups [EB/OL]. http://en.wikipedia.org/wiki/Cgroups.

[13] Google cluster workload traces [EB/OL]. http://code.google.com/p/googleclusterdata/.

[14] Barrosa L, Clidaras J, Holzle U. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines [M]. Morgan and Claypool Publishers, 2013.

[15] Schad J, Dittrich J, Quiané-Ruiz JA. Runtime measurements in the cloud: observing, analyzing, and reducing variance [J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 460-471.

[16] Kas M. Towards on-chip datacenters: a perspective on general trends and on-chip particulars [J]. The Journal of Supercomputing, 2011, 62(1): 214-226.

[17] ONF. Software-Defined Networking: the New Norm for Networks [Z]. In Open Networking Foundation White Paper. 2012.

[18] Deshane T, Dimatos D, Hamilton G, et al. Performance Isolation of a Misbehaving Virtual Machine with Xen, VMware and Solaris Containers [Z]. Technical Report, 2007.

[19] Barham P, Dragovic B, Fraser K, et al. Xen and the art of virtualization [C] // Proceedings of the 19th ACM Symposium on Operating Systems Principles, 2003: 164-177.

[20] Linux Container(LXC) [EB/OL]. http://lxc.sourceforge.net/.

[21] Menon A, Santos JR, Turner Y, et al. Diagnosing performance overheads in the Xen virtual machine environment [C] // Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments, 2005: 13-23.

[22] Banga G, Druschel P, Mogul JC. Resource containers: a new facility for resource management in server systems [C] // Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, 1999: 45-58.

[23] Aron M, Druschel P, Zwaenepoel W. Cluster reserves: a mechanism for resource management in cluster-based network servers [C] // Proceedings of the Conference on Measurement and Modeling of Computer Systems, 2000: 90-101.

[24] Lyneh W, Bray B, Flynn M. The effect of page allocation on caches [C] // Proceedings of the 25th Annual International Symposium, 1992: 222-225.

[25] Lin J, Lu Q, Ding X, et al. Gaining insights into multicore cache partitioning: bridging the gap between simulation and real systems [C] // IEEE 14th International Symposium on High Performance Computer Architecture, 2008: 16-20.

[26] Cho S, Jin L. Managing distributed, shared L2 caches through OS-level page allocation [C] // Proceedings of the IEEE/ACM International Symposium on Microarchitecture, 2006: 455-465.

[27] Tam D, Azimi R, Soares L, et al. Managing shared L2 caches on multicore systems in software [C] // Proceedings in Workshop on the Interaction between Operating Systems and Computer, 2007.

[28] Jeong MK, Yoon DH, Sunwoo D, et al. Balancing DRAM locality and parallelism in shared memory CMP systems [C] // IEEE 14th International Symposium on High Performance Computer Architecture, 2012.

[29] Muralidhara SP, Subramanian L, Mutlu OM, et al. Reducing memory interference in multicore systems via application-aware memory channel partitioning [C] // Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, 2011: 374-385.

[30] Mi W, Feng X, Xue J, et al. Software-hardware cooperative DRAM bank partitioning for chip multiprocessors [J]. Network and Parallel Computing, 2010, 6289: 329-343.

[31] Liu L, Cui Z, Xing M, et al. A software memory partition approach for eliminating bank-level interference in multicore systems [C] // Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques, 2012: 367-376.

[32] Wang XL, Wen X, Li YC, et al. Dynamic cache partitioning based on hot page migration [J]. Frontiers of Computer Science, 2012, 6(4): 363-372.

[33] Chen H, Wang X, Wang Z, et al. DMM: a dynamic memory mapping model for virtual machines [J]. Science China Information Sciences, 2010, 53(5): 1097-1108.

[34] Jin X, Chen H, Wang X, et al. A simple cache partitioning approach in a virtualized environment [C] // Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications, 2009.

[35] Mars J, Vachharajani N, Hundt R, et al. Contention aware execution: online contention detection and response [C] //

Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization, 2010: 257-265.

[36] Mars J, Tang LJ, Soffa ML. Directly characterizing cross core interference through contention synthesis [C] // Proceedings of the 6th International Conference on High Performance and Embedded Architectures and Compilers, 2011: 167-176.

[37] Mars J, Tang LJ, Hundt R, et al. Bubble-up: increasing utilization in modern warehouse scale computers via sensible co-locations [C] // Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, 2011: 248-259.

[38] Tang LJ, Mars J. ReQoS: Reactive static/dynamic compilation for QoS in warehouse scale computers [C] // Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems, 2013.

[39] Mutlu O, Moscibroda T. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared DRAM systems [C] // Proceedings of 35th International Symposium on Computer Architecture, 2008.

[40] Kim Y, Han D, Mutlu O, et al. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers [C] // Proceedings of 16th IEEE International Symposium on High-Performance Computer Architecture, 2010: 1-12.

[41] Kim Y, Papamichael M, Mutlu O, et al. Thread cluster memory scheduling: Exploiting differences in memory access behavior [C] // Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture, 2010: 65-76.

[42] Iyer R. CQoS: a framework for enabling qos in shared caches of cmp platforms [C] // Proceedings of 18th Annual International Conference on Supercomputing, 2004.

[43] Iyer R, Zhao L, Guo F, et al. Reinhardt: QoS policies and architecture for cache/memory in CMP platforms [C] // Proceedings of the 2007 ACM International Conference on Measurement and Modeling of Computer Systems, 2007: 25-36.

[44] Herdrich A, Illikkal R, Iyer RR, et al. Rate-based QoS techniques for cache/memory in CMP platforms [C] // Proceedings of the 23rd International Conference on Supercomputing, 2009: 479-488.

[45] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters [J]. Communications of the ACM, 2008, 51(1): 107-113.

[46] Kapoor R, Porter G, Tewari M, et al. Chronos: predictable low latency for data centerapplications [C] // Proceedings of the Third ACM Symposium on Cloud Computing, 2012.

[47] Computing Community Consortium. 21st Century Computer Architecture [Z]. White Paper, 2012.