DOI：10.7652/xjtuxb201704012

高效支持负载聚合的程序资源敏感度获取及分析方法

王琳[1]，钱德沛[1,2]，王锐[1]，栾钟治[1]，魏光[1]
（1.北京航空航天大学计算机学院，100191，北京；2.中山大学数据科学与计算机学院，510275，广州）

摘要：针对集群或数据中心中负载聚合导致的程序和系统性能不稳定的问题，提出一种程序资源敏感度获取及分析方法。该方法利用Linux中的Cgroups机制设置可调整资源大小的控制组，并让程序在控制组内执行，获得程序在资源受限时的性能；将程序在多个资源限制下的敏感度表示成敏感度超平面，从而预测程序协同执行时的性能；将程序所需的资源按照重要程度排序以帮助集群调度。实验结果表明，该方法预测出的敏感度与真实值的平均绝对误差为0.093。与最小负载的簇级调度策略相比，利用程序资源敏感度信息的调度策略可以提升程序性能22.2%，还可以提升磁盘读写带宽利用率和网络带宽的利用率，程序资源敏感度能够高效地支持负载聚合。

关键词：程序资源敏感度；负载聚合；集群调度；程序协同执行
中图分类号：TP303 **文献标志码**：A **文章编号**：0253-987X(2017)04-0079-06

A Method for Acquiring and Analyzing Program Resource Sensitivity for Workload Consolidation

WANG Lin[1], QIAN Depei[1,2], WANG Rui[1], LUAN Zhongzhi[1], WEI Guang[1]
(1. School of Computer Science and Engineering, Beihang University, Beijing 100191, China;
2. School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510275, China)

Abstract: A method for acquiring and analyzing program resource sensitivity is proposed to deal with the problem that the performances of programs and systems are unstable due to workload consolidation in clusters or data centers. A control group with tunable resources is set up by using Cgroups in Linux, and programs are executed in the control group under various resource conditions. The performance of programs within limited resources is obtained and used for calculating the resource sensitivity. The sensitivity values of programs under restriction of multiple resources form a sensitivity hyperplane to predict the performance of co-running programs. The resources required by a program are reordered according to their importance for cluster scheduling. Experimental results show that the average absolute error between the real sensitivity value and the sensitivity value predicted by the method is 0.093. A comparison with the common least load scheduling strategy shows that the scheduling strategy using resource sensitivity aware improves program performance by 22.2% and enhances both disk I/O and network I/O utilization. It also shows the contribution of program resource sensitivity to workload consolidation.

收稿日期：2016-11-06。 作者简介：王琳(1982—)，女，博士生；王锐(通信作者)，男，讲师。 基金项目：国家重点基础研究发展计划资助项目(2016YFB1000503)；国家自然科学基金资助项目(61133004，61361126011，61502019)。
网络出版时间：2017-03-03 网络出版地址：http://www.cnki.net/kcms/detail/61.1069.T.20170303.0925.014.html

Keywords: program resource sensitivity; workload consolidation; cluster scheduling; co-running programs

集群或数据中心利用负载聚合将多个异构程序同时放置到单个服务器节点上执行[1],该方法在降低程序性能和提高系统资源利用率之间实现权衡。近期调查显示,随着计算需求的增加,超过 60%的数据中心提供商会采用负载聚合[2],但是负载聚合使程序的执行环境变得复杂,为程序性能及系统性能带来挑战。协同执行的程序对共享资源产生竞争从而导致程序性能下降[3-5];这种干扰引起的程序性能下降随着协同执行程序特征的不同而变化,难以预测程序性能[6-7];由于某种资源被抢占导致程序推进缓慢,降低系统吞吐量[8]。

利用程序资源敏感度信息可以高效地支持负载聚合。本文定义程序资源敏感度为程序在资源受限的情况下与资源不受限时相比的性能下降程度。程序资源敏感度描述程序与其他程序协同执行时对程序性能的影响。在集群环境下,假设一个程序同其他程序协同执行,如果程序对某类资源不敏感,则当该类资源被其他程序抢占时,该程序性能不会显著下降;反之,如果程序对该类资源敏感,当该类资源被其他程序抢占时,该程序性能会显著下降。程序资源敏感度的高低直接决定程序协同执行时的性能,所以程序对资源的敏感度也是衡量程序运行时特征的重要指标。了解程序资源敏感度,可以帮助预测程序与其他程序协同执行时的性能下降程度,支持调度器将程序调度到较优的执行环境中执行,提高程序性能和系统资源利用率。

当前利用程序资源敏感度支持负载聚合的方法并不完善。Mars 团队利用压力测试函数获得程序对资源的敏感度,来保证延迟敏感应用与其他应用协同执行时的服务质量[9-10],但是他们所涉及的资源大多是片上核外的共享资源,而忽略了对处理器、磁盘带宽和网络带宽的考虑,所以他们的负载聚合方法对于磁盘密集和网络密集的应用并不适用。Kozyrakis 团队利用协同过滤技术获得应用对各种服务器配置的敏感度,实现高效的负载聚合[11-12],但是受协同过滤技术的限制,他们只能得到程序在有限个服务器配置下的性能,并不能完备地分析程序的敏感度特征。综上所述,如果能够完整、系统地获取程序敏感度特征,则利用程序资源敏感度支持负载聚合将达到更好的效果。本文实现程序资源敏感度获取及分析方法,帮助获取程序在多个资源维度上的敏感度特征。

1 相关工作

Tang 指出利用 LLC miss rate 来预测程序的敏感度特征是不够准确的,随后建立了以 L2 lines in rate 和 LLC lines in rate 为输入的敏感度预测模型[13]。Bubble-up 利用压力测试程序动态调整对存储子系统的压力获得程序的敏感度曲线,保证程序协同执行时的服务质量[9]。Han 为分布式应用建立干扰感知的性能模型,通过改变干扰节点的个数获得应用性能变化曲线[14]。以上工作关注的共享资源只有存储子系统,而忽略了处理器核、磁盘带宽和网络带宽的竞争。

Paragon 和 Quasar 利用协同过滤技术为应用的易受干扰程度打分[11-12],其中包含的共享资源有处理器核、存储子系统、磁盘和网络带宽,但是他们获得的程序资源敏感度结果跟本文不同。文献[11]得到的敏感度结果是程序在不同服务器配置下的性能,而且受协同过滤技术的限制,只能得到程序在有限服务器配置下的性能,而本文可以得到程序在占用不同资源情况下的敏感度超平面。

Heracles 描述干扰特征并用软硬件隔离机制支持批量任务与延迟敏感任务协同执行[15],但是该文描述的是单个资源改变对程序性能的影响,也就是只分析了单个资源对应用的敏感度曲线,并没有分析多种资源同时改变对程序性能的影响。

2 程序资源敏感度获取方法

2.1 敏感度的定义

定义程序在服务器上单独执行时的性能为程序在资源不受限条件下执行时的性能;定义程序执行时服务器上的部分资源由于被控制或被其他程序占用不能全部提供给程序,该程序所表现出来的性能为程序在资源受限条件下执行时的性能;定义程序的资源敏感度为程序在资源受限的情况下执行与资源不受限情况下执行的性能比值,计算方法如下

$$S = P_{\mathrm{li}} / P_{\mathrm{un-li}} \tag{1}$$

式中:$P_{\mathrm{un-li}}$为程序在不受资源限制的情况下执行时的性能;P_{li}为程序在处理器核、磁盘读带宽、磁盘写带宽、内存和网络带宽为 $C[i]$、$D_{\mathrm{r}}[j]$、$D_{\mathrm{w}}[r]$、$M[s]$和 $N[t]$的限制下执行时的性能;S 为介于[0,1]之

间的敏感度。在改变受限资源大小的前提下，当程序对某类资源的敏感度变化范围较大时，说明这类资源大小的变动会对程序性能造成较大变化，程序对这类资源敏感；当程序对某类资源的敏感度变化很小且接近于 1 时，说明这类资源大小的变动对程序性能影响较小，程序对这类资源不敏感。在执行环境中某一种资源改变大小，而其他 4 种资源不受限时，获得的一系列敏感度点构成程序对该资源的敏感度曲线，当执行环境中所有资源参数值变化时，获得的敏感度点构成该程序对多种资源的敏感度超平面。

2.2　受控资源环境设置实现

本文利用 Cgroups[16] 和其他系统工具获取程序资源敏感度。Cgroups 是 Linux 内核提供的一种可以限制、记录、隔离进程组所使用的物理资源的机制，可以为系统中所运行的进程分配资源。Cgroups 分为几个子系统，每个子系统代表一种资源控制器，如处理器核、内存、网络等。本文利用 Cgroups 建立控制组，管理处理器核、磁盘读/写、内存和网络资源，并将程序拉入控制组中控制程序的资源占用。在网络方面，Cgroups 当前提供的功能只能为数据包贴标记，所以本文借助专业的网络带宽限制工具 TC[17] 来限制网络流量。首先利用 Cgroups 为网络数据包贴上标记，然后利用 TC 分类并设置网络带宽。

在控制组内执行程序时需要记录程序性能。对于不同类型的程序，性能的度量标准不同：对于延迟敏感的程序，平均时延是该程序的性能指标；对于访问数据库的程序，每秒操作数是该程序的性能指标；对于其他多数程序，执行时间是性能指标。

2.3　算法实现

本文动态调整程序可占用的资源，并记录程序在这种资源配置下的性能。调整资源规则如下：①将资源划分为 5 个维度，分别是处理器核、磁盘读带宽、磁盘写带宽、内存和网络带宽；②定义最大的实验点为服务器可提供的最大资源，最小实验点为最大实验点的 1/4；③对于每一种资源，按照由小到大的顺序设置多个资源限制点，例如，对于 8 GB 内存，设置资源限制点为 2、2.5、3、3.5、4、4.5、5、5.5、6、6.5、7、7.5、8 GB，所以内存一共设置 13 个实验结点，类似于内存，处理器核、磁盘读带宽、磁盘写带宽和网络带宽分别设置了 7、24、21 和 17 个实验结点；④设置多重循环，让程序在各种资源限制下迭代执行，并记录程序性能，实验需要执行各个资源限制点的全排列，嵌套循环的层数即为资源的个数，每一层循环的迭代次数为该资源上设置的资源限制点的个数。

最简单直观的方法获取程序在各种资源限制下敏感度超平面是让程序在所有资源限制下执行，获取其性能信息。本文用 5 个数组的笛卡尔积来描述所有可能的资源限制组合。如果数组内元素的个数分别为 7、24、21、13 和 17，则执行整个迭代需要执行 $7\times24\times21\times13\times17=779\ 688$ 轮程序。以程序 Data Serving 为例，在资源不受限情况下执行一轮 Data Serving 需要 550 s，所以如果想获得 Data Serving 的敏感度超平面，所需要的时间要长于 $(550\times779\ 688)$ s，这种时间开销显然是无法接受的。所以，我们设计了两级加速获取策略来减少实验执行轮数，缩短实验周期。第一级策略为资源天花板法，查找程序不受限执行时占用的最大资源，将该最大资源作为实验中资源设置的上边界点；第二级策略利用基于插值的预测模型根据已知的少量的敏感度点来获取其他未知的大量的敏感度点。

3　程序资源敏感度分析方法

3.1　获取程序资源敏感度超平面

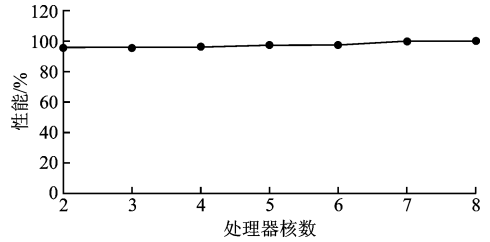
本文方法可以获得程序在各种资源配置下的敏感度，这些敏感度点构成一个超平面。该超平面由五维组成，每个维度分别为处理器核数、磁盘读带宽、磁盘写带宽、内存大小、网络带宽。由于程序的运行时特征不同，不同程序的敏感度超平面是不同的。程序资源敏感度超平面可以预测程序协同执行时的性能。

3.2　根据资源重要程度排序

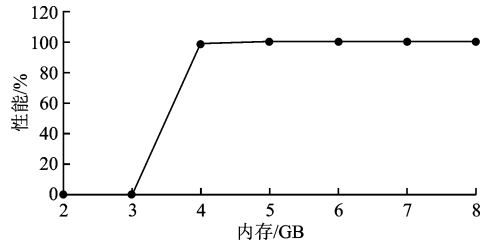
当只限制某种资源而不限制其他资源时，可以得到程序对该资源的敏感度曲线。程序在该资源下边界点上的敏感度可以作为评价资源重要程度的标准，不同的资源对程序的重要程度是不同的。如图 1 所示，当把分配给程序的处理器资源从最小点调到最大点时，Data Caching 程序的性能提升了 5%，当把分配给它的内存资源从最小点调到最大点时，Data Caching 的性能提升了 100%，当把分配给它的网络带宽从最小点调到最大点时，Data Caching 的性能提升了 78%，所以得出结论，程序 Data Caching 最需要的资源排序为内存、网络、处理器。

获得资源重要程度排序的好处是，当为程序选择最优可执行环境时，可以先考虑程序最迫切需要的资源，同时忽略资源敏感度接近于 1 的资源。与只考虑处理器或内存的映射策略相比，根据资源重

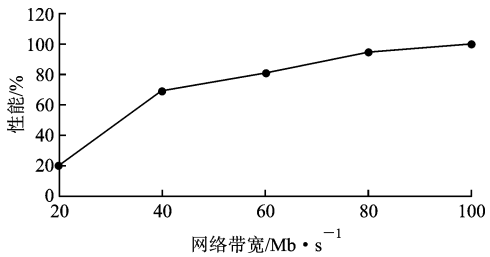
要程度进行映射可以实现个性化调度，从而提升程序性能。



(a)处理器核数变化



(b)内存大小变化



(c)网络带宽变化

图 1 程序 Data Caching 对处理器、内存、网络带宽的敏感度

3.3 利用敏感度信息为程序贴标签

通过本文方法获取程序的敏感度信息可以为程序贴上标签。程序的标签由一个二元组组成，包括程序在 5 个资源维度上的资源使用和资源敏感度。该程序标签可以为调度器提供先验信息，帮助实现高效的负载聚合。

4 实验验证

4.1 实验设置

本文使用 4 节点的集群，每个服务器由 2 个 Intel Xeon E5506(Westmere)处理器组成，服务器的内存为 8 GB，磁盘读带宽为 135 MB/s，磁盘写带宽为 120 MB/s，网络带宽为 100 Mb/s，服务器运行的操作系统是 CentOS 6.5，所使用的内核版本为 Linux kernel version 2.6.32。

集群或数据中心中包含多种多样的程序，为了保证实验的真实性和多样性，从多个基准测试集中选择程序，见表 1。对于不同的程序性能测量标准是不同的，程序 Data Caching 的性能测量标准是平均延迟；程序 Data Serving 的性能测量标准是每秒的操作数；NPB[18] 和 SPEC CPU 2006[19] 中程序的性能测量标准是每秒执行的指令数；程序 fileiord 和 fileiowr 的性能测量标准是每秒读写带宽。

表 1 程序选择

基准测试集	程序
CloudSuite[20]	Data Caching, Data Serving, Media Streaming
NAS Parallel Benchmark[18]	mg. C. 4, ft. C. 4
SPEC CPU2006[19]	bwaves, milc, leslie3d, soplex, libquntum, lbm, wrf
SysBench[21]	fileiord, fileiowr

4.2 敏感度值的验证

为了验证获取的敏感度是否准确，本节实验将被测程序与资源竞争程序协同执行，获取被测程序的资源占用信息。利用资源占用信息预测被测程序的性能，与被测程序的真实性能比较，验证本文敏感度获取方法的准确率。其中，被测程序为 Data Caching、mg. C. 4、ft. C. 4、bwaves、wrf、fileiord 和 fileiowr。资源竞争程序需要对多种资源产生压力，所以该实验设置资源竞争程序为 Media Streaming、mg. C. 4、fileiord 这 3 个程序的组合，其中 Media Streaming 对网络带宽产生压力，mg. C. 4 对处理器和内存产生压力，fileiord 对磁盘带宽产生压力。

图 2 描述了用本文方法预测出的敏感度与真实值的比较。其中定义程序单独执行时的性能为 1，绝对误差为真实值与预测值差值的绝对值。该实验的平均绝对误差为 0.093。可以看出本文方法对程序 bwaves 和 wrf 预测的绝对误差较大，这是因为这两个程序只占用一个核和极少的内存，从系统级监测到的资源占用情况与程序单独执行时是相同的，所以用本文的预测方法预测出来的性能为 1。但是，在实际执行中，由于干扰程序竞争片上共享资源，用本文的方法无法获取微体系结构级的资源竞争，所以导致绝对误差较大。对于微体系结构级的资源竞争，作者另文讨论。本文方法对程序 Data Caching、ft. C. 4、fileiord 和 fileiowr 预测的绝对误差较小，是因为这 4 个程序执行时需要较多资源，所以竞争程序能够对它们产生压力，导致它们协同执行时所占用的资源低于单独执行时占用的资源。由此可得出结论，在程序协同执行时获得的资源低于单独执行时资源占用的情况下，本文的预测方法将发挥较大的优势。

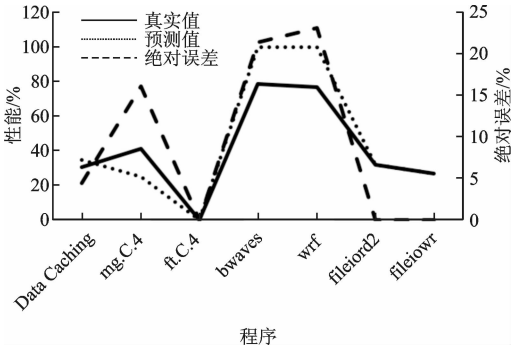


图 2　敏感度的预测值与真实值的比较

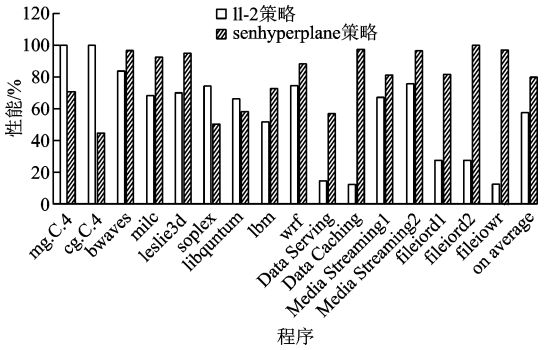


图 3　两种调度策略下程序的性能

4.3　敏感度对负载聚合的贡献

本节实验将 16 个程序映射到 4 台服务器上，并采用以下两个调度策略来验证程序资源敏感度对负载聚合的贡献。第一个是当前最通用的最小负载调度策略[11]，该调度策略在做调度决策时根据程序对处理器和内存的占用情况将程序映射到负载最小的服务器节点上，简称该调度策略为 ll-2 策略。第 2 个是依赖于敏感度超平面的调度策略，该调度策略在为程序查找目标服务器时，将服务器节点的 5 个空闲资源作为参数代入敏感度超平面，从而预测出程序在该节点执行时的性能，如果性能损失较小，则调度程序到该服务器，否则继续查找目标服务器节点，该调度策略简称为 senhyperplane。

图 3 给出在不同调度策略下各测试程序的性能，定义当程序在服务器上单独执行时性能为 1。与调度策略 ll-2 相比，在采用调度策略 senhyperplane 时，程序 mg. C. 4、cg. C. 4、soplex 和 libquntum 的性能有明显下降，其他程序的性能有明显提升，而且程序的平均性能从 57.7%提升到了 79.9%。这是由 senhyperplane 调度策略的理念产生的结果，senhyperplane 考虑了程序在处理器、磁盘读、磁盘写、内存和网络这五维资源的占用情况下的性能，能够实现更为精准的调度。概括来说，这种调度策略会牺牲处理器密集型程序的一小部分性能来挽救磁盘或网络密集型程序的较大部分的性能。

图 4 给出两种调度策略下的资源利用率情况，两种调度策略的处理器及网络带宽的资源利用率基本相同，但是 senhyperplane 策略的磁盘读带宽利用率比 ll-2 策略的磁盘读带宽利用率高出 20.8%，senhyperplane 策略的磁盘写带宽利用率比 ll-2 策略的磁盘写带宽利用率高出 20%，senhyperplane 策略的内存利用率比 ll-2 策略的内存利用率高出 71.7%。这是因为 senhyperplane 策略实现了更为合理的程序映射策略，可保证对磁盘读写、网络带宽有压力的程序映射更为合理。总的来说，与 ll-2 策略相比，senhyperplane 策略通过合理的程序映射，保证了在处理器利用率不变或小幅度降低的前提下，大幅度提高磁盘读写和网络带宽的资源利用率，从而达到了总体上资源利用率的提升。

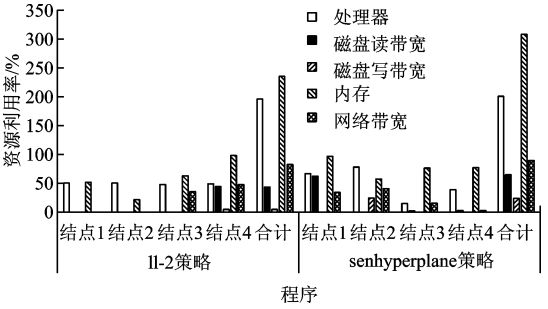


图 4　两种调度策略下的资源利用率

5　结　论

本文介绍程序资源敏感度的获取及分析方法。从数据中心运营商及承租商的角度出发，在系统层次分析程序的运行时特征，为各种程序性能调优及系统性能调优提供支持。通过本文方法可以得到的信息有：程序的资源敏感度超平面可以在已知程序资源占用的情况下预测程序性能；资源的重要程度排序可以帮助人们了解每个程序最必不可少的资源；程序的运行时特征标签可以较完整地描述程序的运行时特征，帮助合理调度。利用本文方法不仅可以分析程序特征、支持个性化精准调度，还有助于保证程序服务质量，实现合理的资源提供，这些将在下一步工作中实现。

参考文献：

[1]　张伟，宋莹，阮利，等. 面向 Internet 数据中心的资源管理 [J]. 软件学报，2012，23(2)：179-199.
ZHANG Wei，SONG Ying，RUAN Li，et al. Resource management in Internet-oriented data centers

[J]. Journal of Software, 2012, 23(2): 179-199.

[2] STANSBERRY M, KUDRITZKI J. Uptime institute 2012 data center industry survey [EB/OL]. (2012-07-11)[2016-07-01]. http://www.etherworks.com.au/index.php?option=com_k2&Itemid=230&id=91_2276d1f6c67a40bd451926a7ab5ccc49&lang=en&task=download&view=item.

[3] ZHURAVLEV S, BLAGODUROV S, FEDOROVA A. Addressing shared resource contention in multicore processors via scheduling [J]. ACM SIGARCH Computer Architecture News, 2010, 45(3): 129-142.

[4] MORETO M, CAZORLA F J, RAMIREZ A, et al. FlexDCP: a QoS framework for CMP architectures [J]. ACM SIGOPS Operating Systems Review, 2009, 43(2): 86-96.

[5] NATHUJI R, KANSAL A, GHAFFARKHAH A. Q-clouds: managing performance interference effects for QoS-aware clouds [C]// Proceedings of the 5th European Conference on Computer Systems. New York, USA: ACM, 2010: 237-250.

[6] DWYER T, FEDOROVA A, BLAGODUROV S, et al. A practical method for estimating performance degradation on multicore processors, and its application to HPC workloads [C]// Proceedings of the 12th International Conference on High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ, USA: IEEE, 2012: 1-11.

[7] PACHECO-SANCHEZ S, CASALE G, SCOTNEY B, et al. Markovian workload characterization for QoS prediction in the cloud [C]// Proceedings of the 2011 IEEE International Conference on Cloud Computing. Piscataway, NJ, USA: IEEE, 2011: 147-154.

[8] 薛正华，刘伟哲，董小社，等. 基于思维进化的集群作业调度方法研究 [J]. 西安交通大学学报，2008，42(6)：651-655.
XUE Zhenghua, LIU Weizhe, DONG Xiaoshe, et al. Research on mind evolutionary computation based job scheduling for server clusters [J]. Journal of Xi'an Jiaotong University, 2008, 42(6): 651-655.

[9] MARS J, TANG L, 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. New York, USA: ACM, 2011: 248-259.

[10] YANG Hailong, BRESLOW A, MARS J, et al. Bubble-flux: precise online QoS management for increased utilization in warehouse scale computers [J]. ACM SIGARCH Computer Architecture News, 2013, 41(3): 607-618.

[11] DELIMITROU C, KOZYRAKIS C. Paragon: Qos-aware scheduling for heterogeneous datacenters [C]// Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM, 2013: 77-88.

[12] DELIMITROU C, KOZYRAKIS C. Quasar: resource-efficient and QoS-aware cluster management [C]// Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM, 2014: 127-144.

[13] TANG Lingjia, MARS J, SOFFA M. Contentiousness vs. sensitivity: improving contention aware runtime systems on multicore architectures [C]// Proceedings of the First International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era. New York, USA: ACM, 2011: 12-21.

[14] HAN J, CHOI Y, HUN J, et al. Interference management for distributed parallel applications in consolidated clusters [C]// Proceedings of the 21th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM, 2016: 443-456.

[15] LO D, CHENG Liqun, GOVINDARAJU R, et al. Heracles: improving resource efficiency at scale [J]. ACM SIGARCH Computer Architecture News, 2015, 43(3): 450-462.

[16] MENAGE P. CGROUPS [EB/OL]. (2016-08-08)[2016-09-08]. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.

[17] KUZNETSOV A N. TC(8) [EB/OL]. (2012-05-19)[2015-01-21]. http://lartc.org/manpages/tc.txt.

[18] NAS Organization. The NASA advanced supercomputing division [EB/OL]. (2016-03-21)[2016-03-25]. http://www.nas.nasa.gov/publications/npb.html.

[19] System Performance Evaluation Cooperative. SPEC CPU 2006 [EB/OL]. (2007-04-03)[2012-01-11]. http://www.spec.org/cpu2006/.

[20] CloudSuite. A benchmark suite for cloud services [EB/OL]. (2016-09-27)[2016-09-30]. http://parsa.epfl.ch/cloudsuite/cloudsuite.html.

[21] AKOPYTOV. SysBench [EB/OL]. (2016-02-01)[2016-02-21]. https://github.com/akopytov/sysbench.

(编辑 武红江)