基于梯度提升回归树的处理器性能数据挖掘研究

吕依蓉[1,2] 孙 斌[3] 喻之斌[1]

[1](中国科学院深圳先进技术研究院 深圳 518055)
[2](中国科学院大学 北京 100049)
[3](首都师范大学信息工程学院 北京 100048)

摘 要 现代处理器一般只内置了 4～8 个性能计数器，但可以监测多达上千个时钟周期级别的性能事件。这些事件可以轻易地产生大量数据，称为处理器性能大数据。然而，如何从这些性能大数据中提取有价值的信息面临着许多挑战。该文提出一种处理器性能数据分析方法，通过迭代使用梯度提升回归树算法构建性能模型，为云计算负载的性能事件进行重要性排序，从而指导云计算平台的性能调优。

关键词 性能计数器；梯度提升回归树；云计算
中图分类号 TP 302.7 **文献标志码** A

Mining CPU Performance Data Based on Gradient Boosting Regression Tree

Lü Yirong[1,2] SUN Bin[3] YU Zhibin[1]

[1](*Shenzhen Institutes of Advanced Technology*, *Chinese Academy of Sciences*, *Shenzhen* 518055, *China*)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049, *China*)
[3](*Capital Normal University Information Engineering College*, *Beijing* 100048, *China*)

Abstract Modern processors typically have only 4-8 performance counters which can be programmed to measure up to thousands of cycle-level performance events. These events can easily generate large amount of data, which is called central processing unit (CPU) big performance data. However, how to extract value from the big performance data faces many challenges. This paper presents a performance data analysis approach, which builds a performance model by iteratively using the gradient boosting regression tree algorithm and quantifies the importance of the performance events of workloads in cloud to guide their performance optimization.

Keywords performance counter; gradient boosting regression tree; cloud computing

收稿日期：2018-03-19 **修回日期**：2018-04-15
基金项目：国家重点专项项目(2016YFB1000204)
作者简介：吕依蓉，硕士研究生，研究方向为计算机体系结构和数据挖掘；孙斌，硕士研究生，研究方向为计算机体系结构和数据挖掘；喻之斌(通讯作者)，博士，研究员，研究方向为计算机系统结构、体系结构支持的云计算和大数据处理系统，E-mail：zb.yu@siat.ac.cn。

1 引 言

随着云计算的发展，数据中心作为计算和存储的核心在运维管理方面遇到了诸多问题和挑战。为了分析、优化程序以及有效地管理系统资源，程序员和系统运维人员都需要精确获取系统运行时的底层信息。性能计数器是用于硬件事件计数的一类寄存器，它们可以在时钟周期级别、低代价地监测并记录系统运行中产生的各类硬件事件。这些性能事件随着时间推移可以轻易地产生大量数据。 在云计算的环境下，数以千计的服务器和数以亿计的负载应用使硬件事件数据量更大，如谷歌的云平台每天产生几个 GB 甚至几个 TB 的中央处理器(Central Processing Unit，CPU)性能数据，我们称之为处理器性能大数据。这些性能事件实际上隐含着影响云计算负载性能的关键因素。因此，性能计数器和硬件性能事件被广泛地应用在任务调度[1,2]、负载特征刻画[3-10]、编译器优化[11-13]和体系结构优化[14-17]等方面。与此同时，一系列基于性能计数器的软件工具也应运而生，如 perf_event[18]、PAPI[19]、VTune 和 Oprofile[20]等。

在云计算时代，性能计数器变得更加重要。许多云服务提供商都渴望通过性能计数器来理解云服务的性能瓶颈。这是因为即使少量的性能提升(如 1%)也能带来巨大的经济效益(如数百万美元)[3]。

然而，在云计算时代使用 CPU 性能计数器产生的数据面临着一个新的挑战——海量的、无规律的 CPU 性能数据难以理解。谷歌的工程师曾报告[3]他们每天通过 GWP 工具从谷歌的数据中心收集几个 GB 的 CPU 性能数据。如果将一个硬件事件视为一个维度，这些数据将是非常高维的数据。例如，因特尔的 Haswell 架构规定了 229 个性能事件，则这款处理器的性能数据有 229 维[21]；因特尔的 IvyTown 架构规定了 1 423 个性能事件，则 IvyTown 的性能数据有 1 423 维[22]。如果同等对待每个事件产生的数据，则需要付出巨大的努力来处理、分析和利用 CPU 性能大数据。为了方便对性能数据进行分析，需要通过以下 3 个方面：(1)量化性能事件对性能的重要性；(2)发现事件与性能的联系；(3)理解事件之间的相互作用。然而，这 3 个方面还未曾在计算机系统结构和云计算系统中进行研究。

因此，本文提出一种云平台上的 CPU 性能大数据挖掘方法，借助性能计数器性能监测工具，对大量硬件事件进行性能重要性排序，寻找事件模式，从而有效地度量和理解云平台上的性能大数据。本文的主要贡献如下：

(1)单计数器单事件(One Counter One Event，OCOE)方式下性能监测数据的拼接与整合。由于性能监测数据无法一次性测量，因此数据片段首先需要进行融合。而多组测量数据之间可能出现事件集重叠的现象，造成了数据不对齐的问题。本文通过设计并实现了 CPU 性能数据融合的方法。

(2)性能事件对性能的重要性量化。用来采集硬件事件信息的性能计数器数量是有限的，而且用户往往并不需要利用所有的硬件事件，即只需要少数重要的硬件事件便可以分析负载特征并进行性能优化。本文首先提出基于机器学习的硬件事件重要性量化方法，然后再用迭代排序不断约简事件空间，最后提炼出一组重要的事件。

本文采用 8 个 Apache Spark 程序进行实验，所建模型在测试集上的精度达到了 95%。另外，我们还用此方法精确刻画了不同的 Spark 程序之间的个性与共性特征。

2 研究背景

现代处理器一般集成了 4～8 个性能计数器[21,23]。其中，性能计数器是一类用在时钟周期级别测量处理器性能事件的特殊寄存器。性能事

件是指可由性能计数器测量的、处理器内的硬件事件，如时钟周期数、执行的指令数以及各级高速缓存的命中次数等[24]。此类硬件事件种类丰富、功能齐全，不仅可以进行实时监测，还可以精确到时钟周期级别。有的事件还被细分成更细粒度的子事件，如高速缓存缺失事件可以被分为数据读取阶段的缺失、数据写入阶段的缺失、取指令阶段的缺失及指令预取阶段的缺失等[25]。系统所支持的事件因处理器而异，不同的供应商会提供不同的硬件事件集；甚至同一处理供应商为不同架构的处理器提供不同数量的硬件事件。一般来说，处理器能支持数百个硬件事件。例如，因特尔为 Haswell 架构的处理器提供了 229 个硬件事件[21]，而为 IvyTown 架构的处理器提供了 1 423 个硬件事件[22]。

通过实时监测硬件事件，可以分析程序运行时的行为，从而获得负载的性能、功耗和能效等特征。而这些特征可以作为资源调整、任务调度策略的依据，最终达到优化能效的目的。

然而，性能计数器在使用过程中面临着一个重大挑战：可用的计数器数目相对保持固定，但需要监测的性能事件的数目却愈发增多，远远大于可用的计数器数目。

少量的性能计数器和大量的待监测性能事件使监测效率和监测结果准确度之间出现了矛盾。为了提高效率，很多研究采用多路复用(Multiplexing，MLPX)的监测方式[26-28]。采用 MLPX 方式时，只需要完整地运行负载程序一次，期间每个性能计数器通过轮询的方式监测多个硬件事件。最后，每个硬件事件根据被记录到的片段推算出在负载整个运行期间的表现。由于每个硬件事件只在负载程序运行过程中的一部分时间内被监测到，导致其性能表现在最终的结果数据中并未被完全体现出来，许多关键信息也许会在轮询间隙丢失。因此，这种方式虽然效率高但精确度低。

图 1 展示了 MLPX 方式监测事件带来的误差(如蓝线所示)。虽然误差并非严格地随事件数目的增加而递增，但从红线可以看出，整体上误差随着事件数目的增多而增加。一个性能计数器同时监测的事件由 10 个增加到 36 个时，误差上升了 46%。如果一个性能计数器同时承担着过多硬件事件数据采集的任务，那么将会导致采集到的数据不精确。实际上，采集全量的硬件事件并非是必须的，许多事件在程序运行的全生命周期中并未发生，时间序列上表现为全部 0 值。

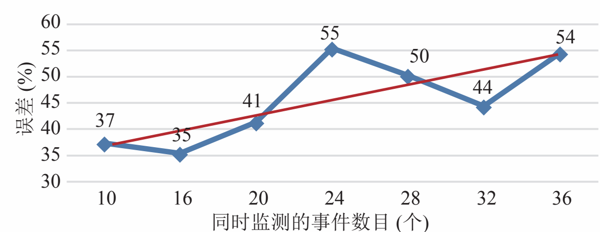


图 1　多路复用方式监测事件带来的误差

Fig. 1　The error variation with the number of events by multiplexing method

为了确保数据精确可靠，本文采用另一种监测方式——OCOE 方式进行 CPU 性能数据的监测。OCOE 方式规定每次测量时，一个性能计数器只承担一个硬件事件的监测任务，从负载程序开始运行到结束。测量更多事件时，需要重新运行负载，同时修改每个性能计数器对应的硬件事件。因此，OCOE 方式监测性能耗时较长，但结果精确度较高，能完整地保留每个事件的时间序列变化情况。

3　CPU 性能监测数据方法

3.1　数据融合和预处理

云环境下存在多个角色不同的服务器节点，如主节点和从节点，它们承担着不同的任务。本研究使用基准测试程序来运行分布式大数据分析负载，同时使用性能计数器来收集每个节点上不同硬件事件在工作负载运行期间的 CPU 性能数

据。最后将这些原始 CPU 性能数据整合到统一的数据集中，以便进行下一步的数据处理。

数据收集从工作负载开始运行开始，以 1 s 的采样频率记录数据，直到负载运行结束。本实验共记录 229 个硬件事件(由于本研究采用的处理器是 Haswell 架构)和每时钟周期机器指令数(Instruction Per Cycle，IPC)。其中，每个硬件事件可视作一维特征(共 229 维)，而 IPC 则作为响应变量，在后续步骤中建立机器学习模型。

许多研究将负载运行一次作为一条实验样本，把程序运行时间作为响应变量[25,29]。本文把程序一次完整的运行视为一个时间序列，而不是简单地将运行过程中各种硬件事件作粗粒度的统计，这样可以最大限度地保留硬件事件的变化细节。另外，硬件事件在采样过程中时间粒度细(采样频率为 1 s)，在后续的模型训练中有助于提高精确度。

定义硬件事件空间 U。用户、运维者或程序开发人员运行 m 次实验，每次工作负载从开始运行到结束的时间段为 t s。则硬件事件信息的时间序列长度为 t(每秒记录一次)。在每次实验中，用户根据自己的经验，选择一部分他们希望监测的事件 $E_i(E_i \subseteq U)$。同时，本文定义 E_i 包含的硬件事件数量为：

$$q_i = card(E_i) \tag{1}$$

那么，一共可以监测到事件子空间为：

$$E = E_1 \cup E_2 \cup \cdots \cup E_i \cup \cdots \cup E_m \tag{2}$$

这也是后续模型训练的特征空间。

事实上，每次实验中，负载程序的运行时长并不确定，但在某个值上、下波动。图 2 为 K-means 和 Wordcount 这两个负载在相同实验环境下多次运行的执行时间变化。其中，K-means 的平均运行时长为 180.08 s，方差为 210.68 s；Wordcount 的平均运行时长为 99.92 s，方差为 769.94 s。这也解释了每次实验中 IPC 序列的变化并不完全一致，趋势一致但值不一致的原因。同时又考虑到每次用户选择监测的事件可能有重叠。这些现象造成了数据不对齐的问题。所以，m 次实验所得的全部数据需要进行整合与拼接。

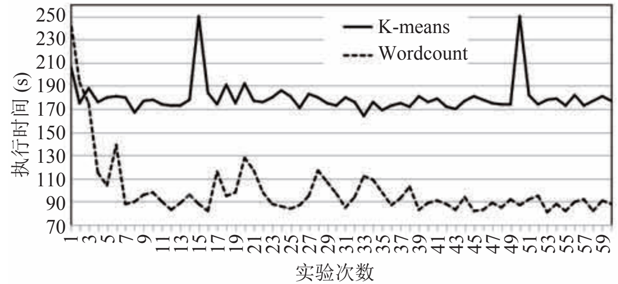


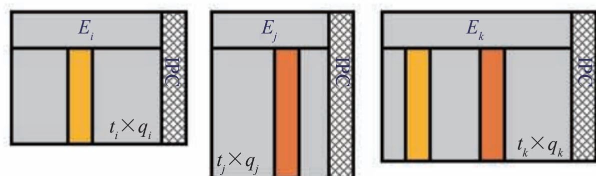
图 2 负载应用 K-means 与 Wordcount 的运行时间波动情况

Fig. 2 Execution time fluctuation on K-means and Wordcount

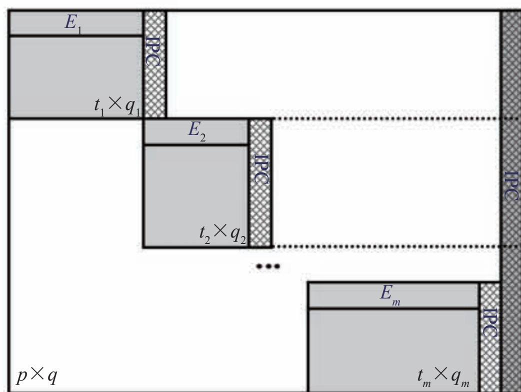
图 3 展示了如何将每轮运行负载所产生的 CPU 性能数据整合在一起。其中，图 3(a)是 3 次独立实验产生的数据，垂直方向表示时间序列的长度，水平方向表示事件集的维度(多少个监测事件)。它们展示了一种典型的事件重叠的情况。其中，黄色竖条代表事件 e_1 产生的时间序列，它分别在第 i 次和第 k 次实验中被监测，即 $e_1 \in E_i$ 且 $e_1 \in E_k$；橙色竖条代表事件 e_2 产生的时间序列，它分别在第 j 次和第 k 次实验中被监测，即 $e_2 \in E_j$ 且 $e_2 \in E_k$。

随后，将每次实验产生的数据(除 IPC 外)按对角线方向依次拼接在一起。而 IPC 序列头尾拼接，作为最终数据集的响应变量，如图 3(b)所示。接着，重复出现的事件(如 e_1 和 e_2)序列依次平移至同一列上。最后，将其他位置填 0，如图 3(c)所示。经过 m 次实验，数据合并成一个 $p \times q$ 的矩阵 $\boldsymbol{S}$，其中，$p = \sum_{i=1}^{m} t_i$；$q = card(E)$。如此形成的矩阵是一个分块的、稀疏的大矩阵。其中，矩阵中每一列代表一个性能事件，每一行代表一次采样，每一个分块代表负载完整运行一次。

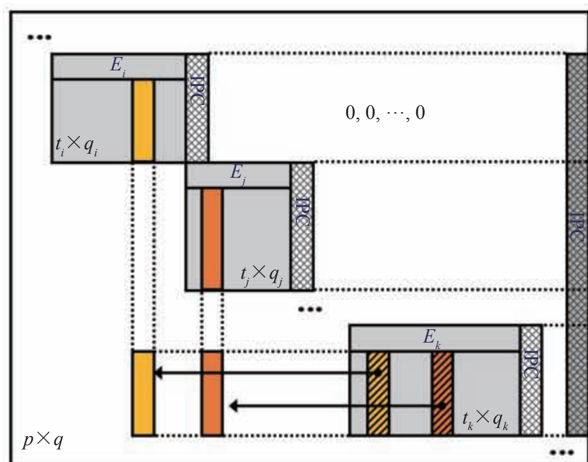
接下来，本文继续对矩阵 $\boldsymbol{S}$ 中的数据做预处理。不同硬件事件的取值范围不同，且差异很大。例如，MEM_UOPS_RETIRED.STLB_MISS_LOADS 的范围是 0～4；而 UOPS_EXECUTED_



(a) 第 i 次、第 j 次和第 k 次实验监测数据



(b) 数据按对角线拼接，同时提取 IPC 作为响应变量



(c) 重叠事件对齐

图 3　数据融合

Fig. 3　Data integration

PORT.PORT_1 的范围是 96～305。如果一个事件的取值方差远大于另一个事件，前者会对最后生成的估计模型占主导作用，这个模型将很难从后者中准确地评估到期待的价值。所以本文采用公式(3)的方法对其做归一化处理，使取值范围全部落在[0,1]内。

$$X'=\frac{X-X_{\min}}{X_{\max}-X_{\min}} \tag{3}$$

其中，X' 为归一化后的值；X 为源数据；$X_{\max}$ 为源数据序列中的最大值；$X_{\min}$ 为源数据序列中的最小值。这个方法实现了对源数据的等比例缩放。

3.2　基于迭代梯度提升树模型的特征选择方法

在机器学习领域，“特征工程”的思想为我们约简硬件事件空间的工作带来了灵感。对高维硬件事件进行数据过滤，使得可在低维空间内对其进行分析。本方法自动选择出一个对 IPC 影响最大的事件子集，并对其进行排序。相比特征工程问题来说，本工作就是将每个硬件事件视为一个特征，将 IPC 视为目标，构建机器学习模型拟合目标，并对每个特征进行重要性量化，最后对其进行排序。

本文选用梯度提升回归树算法[30]作为事件选择的基本模型，在模型的构造过程中逐渐计算出每个事件的重要性。同时经过硬件事件选择，每次减少 10 个最不重要的事件，多次迭代建模，直到模型误差最小。此时的事件重要性排序最准确。

分析硬件事件的重要性，量化研究对性能影响较大的硬件事件(单个或多个)，从而帮助理解复杂情况下的程序性能行为。为了量化硬件事件的重要性，本文构造了一个以 IPC 为目标的机器学习模型。该模型的输入为硬件事件在负载运行期间的时间序列值，模型输出为 IPC 的时间序列，可用公式(4)表示。

$$IPC=perf(e_1,e_2,\cdots,e_i,\cdots,e_n) \tag{4}$$

其中，IPC 为负载程序运行期间以 1 s 为采样频率采集到的每时钟周期机器指令数；e_i 为第 i 个硬件事件；n 为硬件事件的数目。对于现代处理器来说，硬件事件的数量成百上千，故带有如此大量的参数来构建精确的性能模型极具挑战，而简单的统计模型尚不能满足以上要求。为了解决这个问题，机器学习算法成为有可能解决高维参数建模问题的方法。但选择哪一种机器学习算法

是一个值得研究的问题。本文使用梯度提升回归树模型，因为它在许多计算机体系结构方向的建模任务中表现均较好[31,32]。

梯度提升回归树是一个典型的针对任意可微的损失函数的提升模型，把弱的模型组合成一个强的模型，可用于回归和分类问题。提升树模型可以表示为以决策树为基函数的加法模型，是决策树的线性组合，称为集成模型。梯度提升回归树支持多种不同的回归损失函数，本文默认回归损失函数为最小二乘损失函数，由公式(5)表示。

$$L[y,f(x)]=[y-f(x)]^2 \tag{5}$$

其中，y 为真实值；$f(x)$ 为模型预测值。

梯度提升回归树算法由 Friedman[30]提出并实现，同时提出了如何估计预测变量的相对影响。对于这个集成模型中一个单独的回归树 T，本文可以使用 $I_j^2(T)$ 作为每个参数 e_j 对目标变量(IPC)的影响程度的度量。$I_j^2(T)$ 解释为 e_j 被选中作为决策树的节点进行分裂的次数，然后根据对分裂结果的影响程度的平方来加权[31]，具体计算见公式(6)。

$$I_j^2(T)=nt\cdot\sum_{i=1}^{nt}p^2(k) \tag{6}$$

其中，nt 为 e_j 被选中作为树的节点进行分裂的次数；$p^2(k)$ 为第 k 次分裂后对树模型的性能提升的平方。一般地，$p(k)$ 被定义为 IPC 的相对误差，即 $(IPC_k-IPC_{k-1})/IPC_{k-1}$。那么对于全部的回归树来说，$e_j$ 的重要性可以用公式(7)表示。

$$I_j^2=\frac{1}{R}\sum_{m=1}^{R}I_j^2(T_m) \tag{7}$$

其中，R 为组合模型中树的个数；$I_j^2(T_m)$ 为第 m 棵树中 e_j 对 IPC 的影响程度。

为了让最后的结果更好理解，本文将每个变量对目标的贡献值进行归一化处理，即保证这些值的总和为 100%。其中，数值越高表示该变量对目标的响应更强，也意味着这个硬件事件对性能的影响更为重要。在获得每个硬件事件对性能的重要性之后，首先对它们进行降序排序，然后将排在最末(最不重要)的 10 个事件剔除，最后用剩余的事件重新训练模型，具体操作流程如图 4 所示。如此迭代，直到硬件事件的数量减少到使模型精确度最高。本文将这个最终的模型称为最精确性能模型。这么做的原因是硬件事件的信息存在冗余，有些硬件事件对模型的构建会产生负面的影响。

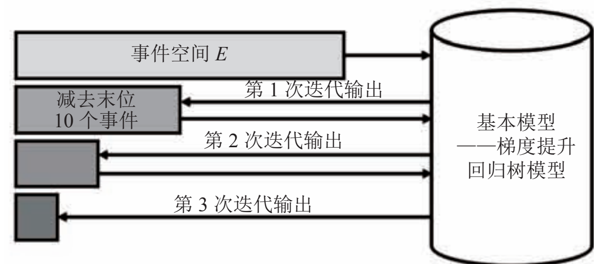


图 4 迭代的梯度提升回归树模型

Fig. 4 Iterative gradient boosting regression treemodel

4 实验结果与分析

4.1 实验环境

本文的实验环境为 4 台戴尔服务器，其中 1 台作为主节点，其余为从节点。每台服务器配备有 16 核 Intel(R) Xeon(R) CPU E5-2630 v3 @2.4GHz 处理器，处理器微体系结构为 Haswell-E，内存大小为 64 GB，32K L1d cache，32K L1i cache，256K L2 cache，20480K L3 cache，硬盘容量 2 TB。服务器的操作系统为 Ubuntu 14.04.5 LTS (GNU/Linux 3.16.0-77-generic x86_64)，集群管理系统为 Mesos 1.0，应用计算框架为 Spark 2.2。

本文的基准测试工具选用 HiBench[33]，并从中挑选了 8 个典型的负载作为实验程序，包括图分析(Pagerank)、分布式数据库应用(Scan、Join 和 Aggregation)、机器学习应用(Bayes 和 K-means)及微基准程序(Sort 和 Wordcount)。

系统开发语言使用 Python 2.7。其中，Python 作为一种易读、易维护的高级编程语言已被大量用户广泛使用。同时 Python 具有丰富和强大的库，本文使用 scikit-learn 0.19.0 机器学习库[34]来实现文中用到的梯度提升回归树算法。

4.2 模型精度评估

在最精确性能模型中，本文随机选择 80% 的样本作为训练集，剩余的作为测试集。图 5 为 8 个基准测试程序在迭代训练过程中模型测试集上的误差变化情况。模型误差的定义如公式(8)所示。

$$err=\frac{|IPC_{\text{means}}-IPC_{\text{pred}}|}{IPC_{\text{means}}}\times100\% \tag{8}$$

其中，IPC_{means} 为实际度量值；IPC_{pred} 为模型预测值。

当 229 个事件全部参与训练时，模型在 8 个基准测试程序下的平均误差为 14%。而当事件数目减少到 150 个时，平均误差下降到了 6.3%。这说明全量的硬件事件确实存在冗余。

4.3 性能事件重要性

图 6 为 8 个基准测试程序的硬件事件重要性排序，以及将这 8 个程序产生的硬件事件时间序列数据合并之后，为整体建模得到的硬件事件重要性排序(Mix)。其中，y 轴表示事件重要性(%)；x 轴为各个事件名称的缩写。事件名称的全称和事件含义说明见表 1。由于篇幅有

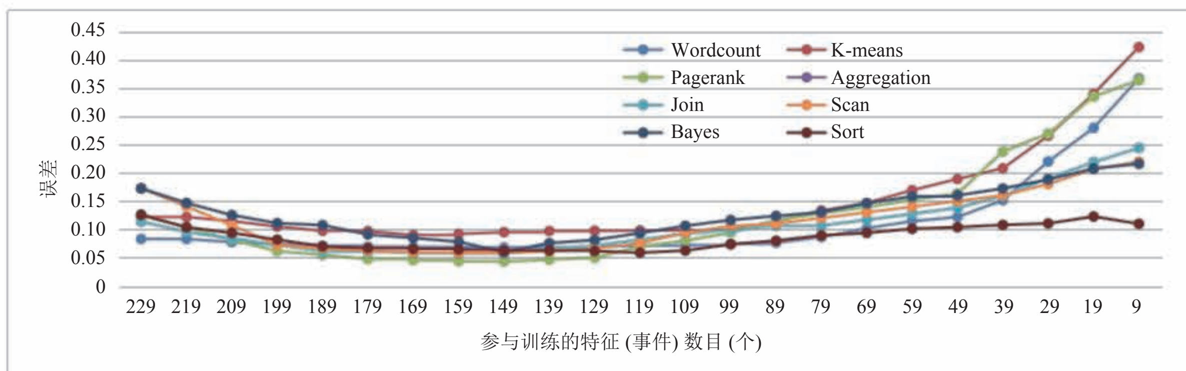


图 5 性能事件约简过程中模型误差的变化

Fig. 5 Model error varies with the number of events

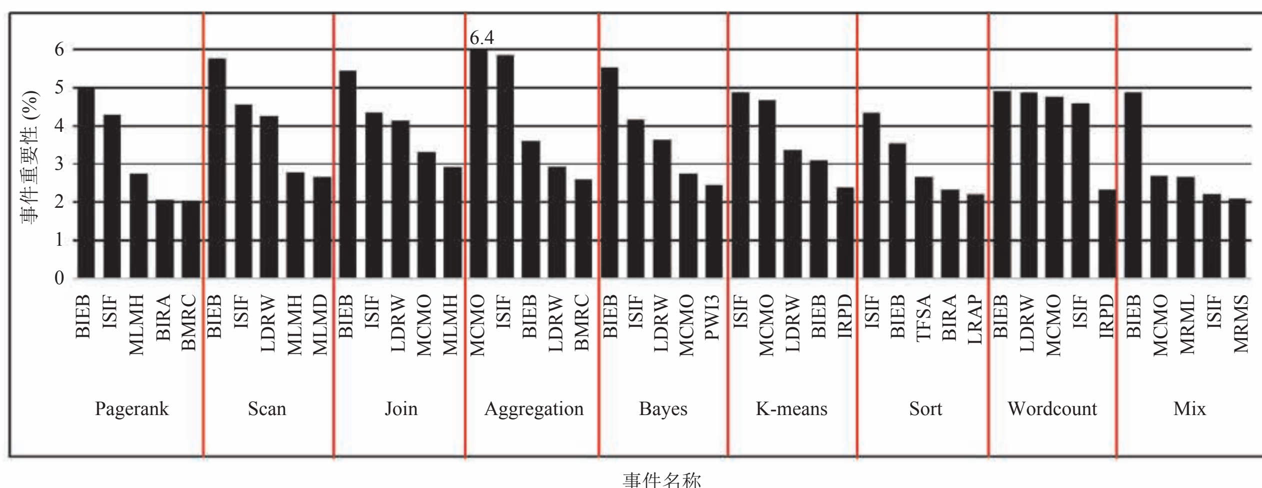


图 6 8 个负载程序单独建模及其混合建模后的性能事件重要性排名(前 5)

Fig. 6 The importance rank of events for 8 workload models and a hybrid model

表 1 事件描述

Table 1 Event name and description

缩写	事件名称	描述
BIEB	BR_INST_EXEC.ALL_BRANCHES	对所有附件执行的分支计数（并不一定执行完整）
ISIF	ILD_STALL.IQ_FULL	由于指令队列已满而造成 CPU 停止运转的周期数
MLMH	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_HITM	三级缓存缺失且从远端缓存得到数据载入的微操作计数
BIRA	BR_INST_RETIRED.ALL_BRANCHES	执行完的分支指令数
BMRC	BR_MISP_RETIRED.CONDITIONAL	执行完但预测错误的分支指令数
LDRW	L2_DEMAND_RQSTS.WB_HIT	命中二级缓存且不拒绝写回的操作数目
MLMD	MEM_LOAD_UOPS_L3_MISS_RETIRED.REMOTE_DRAM	三级缓存缺失且从远端内存得到数据载入的微操作计数
MCMO	MACHINE_CLEARS.MEMORY_ORDERING	计算由于内存顺序冲突而进行清理的次数
PWI3	PAGE_WALKER_LOADS.ITLB_L3	三级缓存命中时的指令地址翻译缓存页面加载次数
IRPD	INST_RETIRED.PREC_DIST	以减少 PEBS 阴影在 IP 分配中的影响的硬件指令精准完成事件
TFSA	TLB_FLUSH.STLB_ANY	从二级地址翻译缓存写出的次数
LRAP	L2_RQSTS.ALL_PF	所有二级缓存硬件预取指请求次数
MRML	MEM_UOPS_RETIRED.STLB_MISS_LOADS	由于二级地址翻译缓存缺失而从内存数据载入的微操作计数
MRMS	MEM_UOPS_RETIRED.STLB_MISS_STORES	由于二级地址翻译缓存缺失而对内存数据存储的微操作计数

限，图中只展示了每个负载排名前 5 的事件。

下面阐述本研究的重要发现。首先，同一个测试程序中总有 1 个或 2 个事件的重要性远远高于其他事件。例如，在 Aggregation 程序中，MCMO 和 ISIF 是最重要的两个事件，它们的重要性都大于 5.8%，而其余事件的重要性都低于 3.6%。其次，对于不同的基准测试程序，其最重要的事件是不尽相同的。例如，受 K-means 程序影响最大的硬件事件是 ISIF；而受 Bayes 程序影响最大的硬件事件是 BIEB。这意味着不同的测试程序在微体系结构层面的特征不一样。最后，纵观 8 个 Spark 测试程序，它们较重要的事件存在交集，说明 Spark 程序之间也存在共同的重要硬件事件。其中，ISIF 和 BIEB 在所有的基准测试程序中都很重要。同时，这个特点也体现在 Mix 程序中，一定程度上说明这些结论得到了佐证。另外， PWI3、MCMO 和 LDRW 在数据库操作相关的负载中重要性较高。以上发现都可以有效地用来指导以 Spark 为例的 in-memory 程序的性能优化。

由于目前在研究文献中尚没发现利用硬件事件指导大数据框架参数调优的研究。为验证本文提出的硬件事件重要性排序的效果，我们构建了一个传统的基于梯度提升回归树的 Spark 参数调优模型，与本文提出的在性能事件重要性指导下的 Spark 参数调优进行对比。

传统的 Spark 参数调优模型[35]将 Spark 配置参数视为模型的特征，对这些参数随机设定参数值后运行负载，记录程序完整的运行时间，并将这个运行时间视为模型的响应变量。这个过程需要构造充足的实验样本才能让模型精确，以确定影响程序性能最重要的参数。另一方面，通过将最精确性能模型得出的排名前 10 的硬件性能事件与 Spark 参数进行关联，直接关注与重要事件紧耦合的 Spark 参数，即可快速得出影响程序性能的重要参数。基于硬件性能事件重要性指导的 Spark 参数调优方法的数据收集阶段与传统的 Spark 参数配置方法相同。而在训练性能模型阶段，我们构造事件与 Spark 参数的混合向量，如公式(9)所示。

$$\vec{v}=(e_1, e_2, \cdots, e_n, p_1, p_2, \cdots, p_m) \tag{9}$$

其中，e 代表事件；p 代表 Spark 参数，一般 n 取

10 以内。然后将这样的样本向量投入机器学习模型中训练，以执行时间作为模型拟合的目标变量。

在模型训练过程中，可以找出任一事件和任一 Spark 参数之间的相关性。在最紧耦合的一对事件和参数中，我们即可确定这个参数为调优目标。

实验结果表明，以 Pagerank 为例，构建 Spark 参数模型至少需要构建 6 000 组训练样本(6 000 次程序运行)才能有效地找出最重要的参数。而训练最精确性能模型本身需要花费 3 277 条样本(60 次程序运行)，计算排名前 10 的硬件性能事件与 Spark 参数的耦合程度需要构造 1 520 条样本，如表 2 所示。而二者最终确定的重要 Spark 参数一致。

表 2　模型结果对比

Table 2　Comparison of model results

模型类型	样本量	误差（%）
Spark 参数模型	6 000	6
最精确性能模型	60	12
Spark 参数与事件耦合模型	1 520	9

由此可以发现，最精确性能模型方法的性能优于传统的基于机器学习方法的 Spark 参数调优性能。但是，受到负载程序运行时间有限的影响，最精确性能模型的精度略低于对 Spark 参数直接建模。所以在对精度要求高的任务中，二者可以互补，先由最精确性能模型快速选出大致的调参范围，再建立这些参数的机器学习模型，更精确地得出具体的调参对象。

5　总结与展望

由性能计数器记录的性能监测数据，能够很好地洞察云计算负载的性能表现。本文提出一种 CPU 性能大数据的挖掘方法，通过迭代使用梯度提升回归树算法构建性能模型，分析出云环境下负载的性能事件重要性排序，从而指导云平台的性能调优。

由于终端用户很少拥有关于系统底层和性能事件的领域知识，这让用户对使用相关的性能分析工具造成了障碍。本文提出的最精确性能模型，通过对事件受 IPC 影响的重要程度排序来降低事件空间。这种方法使得用户不需要完全了解几百个全部事件的含义，只需要熟悉掌握个别少数事件，就能快速分析系统性能，参与性能调优。文中还实现并分析了一个对 Spark 配置参数调优的案例，这个案例很好地将硬件层和应用层结合在一起。

尽管本文获得了一定的成果，但仍存在一些不足之处。首先，OCOE 方式收集性能数据记录始终效率较低，数据准备过程时间较长。在后续研究中，可以使用统计机器学习领域的一些方法对 MLPX 方式监测到的性能数据进行数据清洗，提高数据质量。由此，解决了 MLPX 方式监测数据带来的误差，使得性能监测过程既提高了效率，又保证了数据的可靠准确。其次，除了本文所述的关注事件受性能影响的重要程度，量化研究事件与事件之间的相互影响程度也十分必要。综合利用事件重要性排名和事件之间的相关性可用于快速优化大数据分析程序的性能。

参 考 文 献

[1] Chen Q, Guo M, Huang Z. CATS: cache aware task-stealing based on online profiling in multi-socket multi-core architectures [C] // ACM International Conference on Supercomputing, 2012: 163-172.

[2] Blagodurov S, Fedorova A. User-level scheduling on NUMA multicore systems under Linux [C] // Proceedings of Linux Symposium, 2011: 81-92.

[3] Ren G, Tune E, Moseley T, et al. Google-wide profiling: a continuous profiling infrastructure for data centers [J]. IEEE Micro, 2010, 30(4): 65-79.

[4] Jia Z, Wang L, Zhan JF, et al. Characterizing data analysis workloads in data centers [C] // IEEE International Symposium on Workload Characterization, 2014: 66-76.

[5] Ferdman M, Adileh A, Kocberber O, et al. Clearing the clouds: a study of emerging scale-out workloads on modern hardware [C] // ASPLOS XVII Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems, 2012: 37-48.

[6] Wang L, Zhan JF, Luo CJ, et al. BigDataBench: a big data benchmark suite from internet services [C] // 2014 IEEE 20th International Symposium on High Performance Computer Architecture, 2014: 488-499.

[7] Yasin A, Ben-Asher Y, Mendelson A. Deep-dive analysis of the data analytics workload in CloudSuite [C] // 2014 IEEE International Symposium on Workload Characterization, 2014: 202-211.

[8] Yasin A. A top-down method for performance analysis and counters architecture [C] // 2014 IEEE International Symposium on Performance Analysis of Systems and Software, 2014: 35-44.

[9] Xiong W, Yu ZB, Eeckhout L, et al. Shenzhen transportation system (SZTS): a novel big data benchmark suite [J]. Journal of Supercomputing, 2016, 72(11): 4337-4364.

[10] Kanev S, Darago JP, Hazelwood K, et al. Profiling a warehouse-scale computer [C] // ACM/IEEE International Symposium on Computer Architecture, 2015: 158-169.

[11] Chen DH, Vachharajani N, Hundt R, et al. Taming hardware event samples for FDO compilation [C] // Proceedings of the 8th annual IEEE/ACM International Symposium on Code Generation and Optimizations, 2010: 42-52.

[12] Moseley T, Grunwald D, Peri R. OptiScope: performance accountability for optimizing compilers [C] // Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization, 2009: 254-264.

[13] Wang Z, O’Boyle MFP. Mapping parallelism to multi-cores: a machine learning based approach [J]. ACM SIGPLAN Notices, 2009, 44(4): 75-84.

[14] Kozyrakis C, Kansal A, Sankar S, et al. Server engineering insights for large-scale online services [J]. IEEE Micro, 2010, 30(4): 8-19.

[15] Bare KA, Kavulya S, Narasimhan P. Hardware performance counter-based problem diagnosis for e-commerce systems [C] // 2010 IEEE Network Operations and Management Symposium, 2010: 551-558.

[16] Bhatia S, Kumar A, Fiuczynski ME, et al. Lightweight, high-resolution monitoring for troubleshooting production systems [C] // Proceedings of the 8th Usenix Conference on Operating Systems Design and Implementation, 2008: 103-116.

[17] Williams S, Waterman A, Patterson D. Roofline: an insightful visual performance model for multicore architectures [J]. Communications of the ACM, 2009, 52(4): 65-76.

[18] Bircher WL, John LK. Complete system power estimation using processor performance events [J]. IEEE Transactions on Computers, 2012, 61(4): 563-577.

[19] Browne S, Dongarra J, Garner N, et al. A portable programming interface for performance evaluation on modern processors [J]. International Journal of High Performance Computing Applications, 2000, 14(3): 189-204.

[20] Eranian S. Perfmon2: a flexible performance monitoring interface for Linux [C] // Proceedings of the 2006 Ottawa Linux Symposium, 2006: 269-288.

[21] Intel Corporation. Intel 64 and IA-32 architectures software developer’s manual [OL]. [2018-04-30]. https://software.intel.com/sites/default/files/managed/39/c5/325462-sdm-vol-1-2abcd-3abcd.pdf.

[22] Zellweger G, Lin D, Roscoe T. So many performance events, so little time [C] // Proceedings of the 7th ACM Sigops Asia-Pacific Workshop on Systems, 2016: 14.

[23] Weaver VM, Mckee SA. Can hardware performance counters be trusted? [C] // IEEE International Symposium on Workload Characterization, 2008: 141-150.

[24] Advanced Micro Devices. BIOS and Kernel developer's guide (BKDG) for AMD family 15h models 00h-oFh processors [OL]. 2013-01-23[2018-04-30]. https://support.amd.com/TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf.

[25] Lin D. Blackboxing performance monitoring units [D]. Zurich: ETH Zurich, 2016.

[26] Mytkowicz T, Sweeney PF, Hauswirth M, et al. Time interpolation: so many metrics, so few registers [C] // Proceedings of the 40th Annual IEEE/ACM International Symposium on MicroArchitecture, 2007: 286-300.

[27] Dimakopoulou M, Eranian S, Koziris N, et al. Reliable and efficient performance monitoring in Linux [C] // Proceedings of the International Conference for High Performance Computing, 2017: 34.

[28] Mytkowicz T, Diwan A, Hauswirth M, et al. Producing wrong data without doing anything obviously wrong! [J]. ACM Sigarch Computer Architecture News, 2009, 37(1): 265-276.

[29] Ofenbeck G, Steinmann R, Caparros V, et al. Applying the roofline model [C] // 2014 IEEE International Symposium on Performance Analysis of Systems and Software, 2012: 76-85.

[30] Friedman JH. Greedy function approximation: a gradient boosting machine [J]. The Annals of Statistics, 2001, 29(5): 1189-1232.

[31] Friedman JH, Meulman JJ. Multiple additive regression trees with application in epidemiology [J]. Statistics in Medicine, 2003, 22(9): 1365-1381.

[32] Yu ZB, Wang J, Eeckhout L, et al. QIG: quantifying the importance and interaction of GPGPU architecture parameters [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2017: 99.

[33] Huang S, Huang J, Dai J, et al. The HiBench benchmark suite: characterization of the MapReduce-based data analysis [C] // IEEE 26th International Conference on Data Engineering Workshops, 2010: 41-51.

[34] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: machine learning in Python [J]. Journal of Machine Learning Research, 2011, 12(10): 2825-2830.

[35] Bei ZD, Yu ZB, Zhang HL, et al. RFHOC: a Random-forest approach to auto-tuning Hadoop's configuration [J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 27(5): 1470-1483.