

# CounterMiner: Mining Big Performance Data From Hardware Counters

基于机器学习的硬件事件语义分析方法关键技术研究

选自 2018 51st MICRO

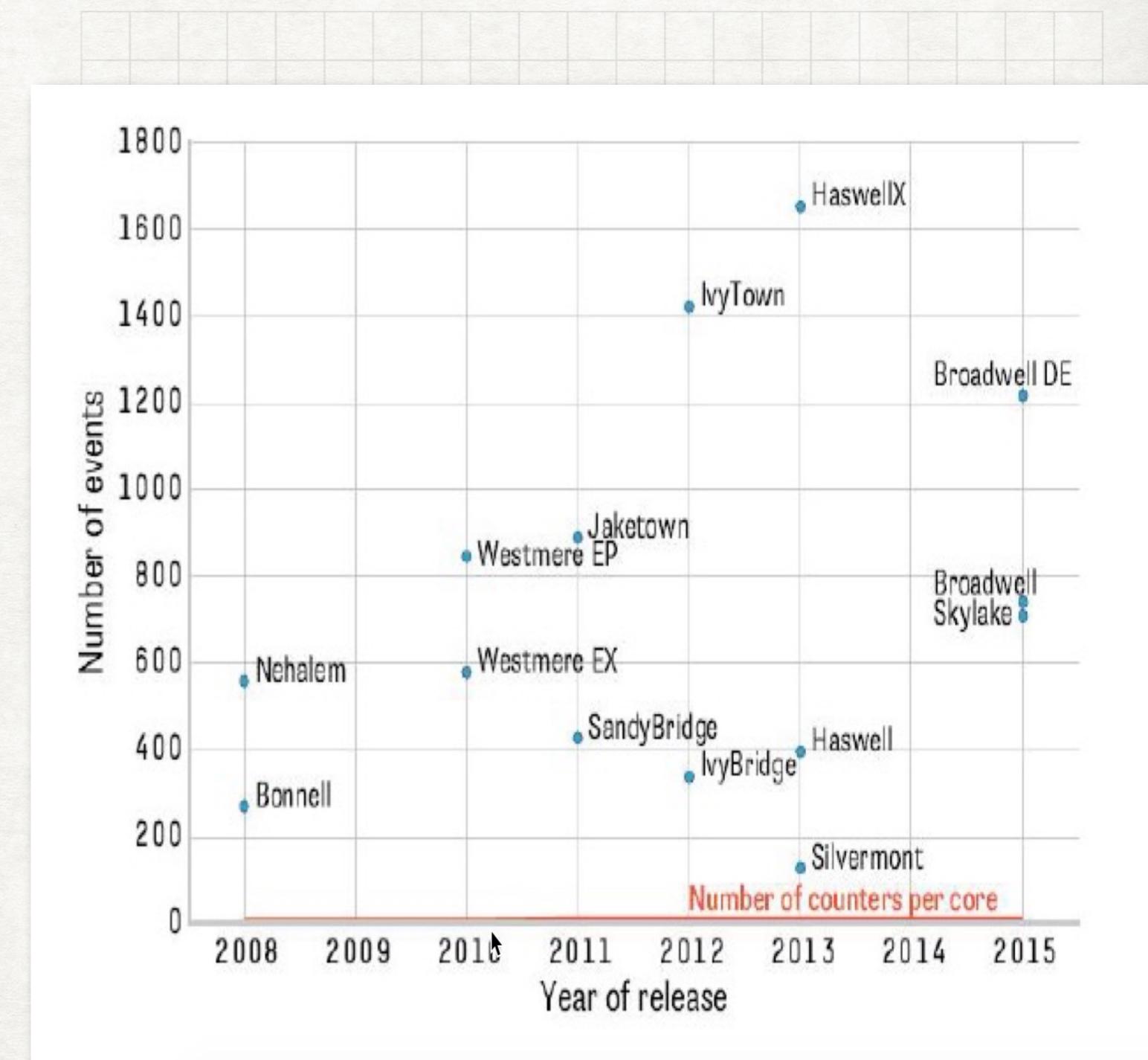
- 研究背景
- 性能计数单元技术简介
- CounterMiner系统设计
- 硬件事件空间化简
- 硬件事件相关性分析
- 总结展望

# 研究背景



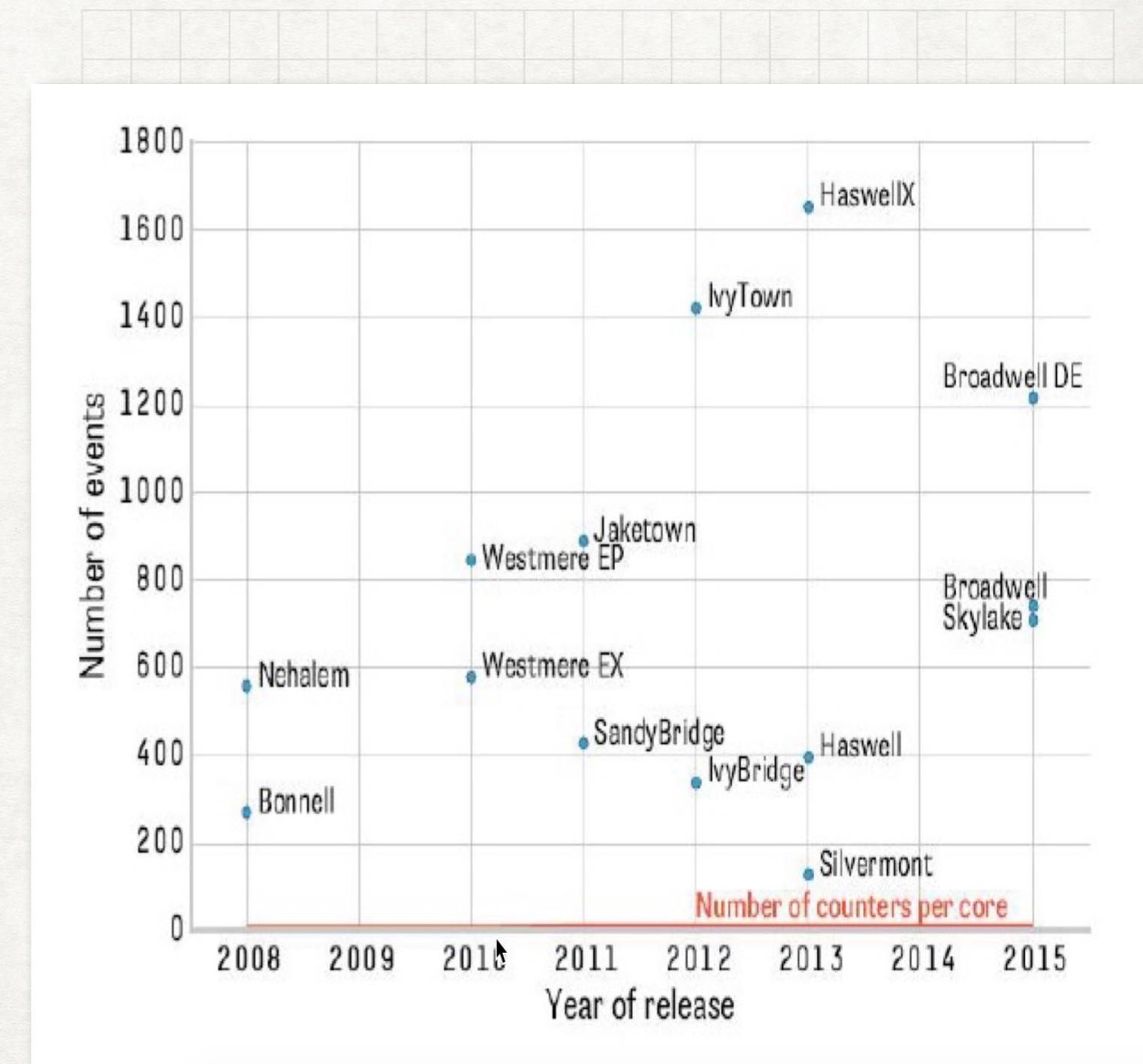
# 研究背景

- 英特尔微体系结构的性能事件与计数器数量变化图
  - $\Delta(\text{Counters}) \rightarrow 0$
  - $N(\text{Events}) \gg N(\text{Counters})$
  - 不同微体系结构所支持的性能事件并不完全相同



# 研究背景

- PMU缺点
  - 典型事件过多，使用门槛高
  - 计数器稀缺
  - 性能事件特定于处理器，不同架构处理器对应性能事件并不完全通用
- 需求
  - 降低待监测事件数量
  - 从一定范围内的事件中提取有价值信息
  - 降低门槛，用户无需考虑PMU细节即可使用PMU进行事件监测与性能分析



# 研究现状

- 2-8个性能计数器监测上百个关键的性能事件
- 一系列可编程的性能检测工具
- 任务调度、编译器优化、体系结构优化、负载特征刻画、性能建模、能耗问题、应用程序延迟等方面
- 两种监测方式：
  - 单计数单事件（OCOE）
    - 低效高准确性
  - 多路复用（MLPX）多次事件累加运算+倒推
    - 高效低准确性

# 研究内容与目标

- 研究目标
  - 对大量硬件事件的表现进行语义分析
  - 寻找事件模式
  - 刻画负载特征
- CounterMiner
  - 数据采集与整合
  - 硬件事件提纯
  - 硬件事件的内在联系

# 性能计数单元技术简介

- 性能事件：
  - 可由PMU测量的硬件事件，如时钟周期数、执行指令数、各级缓存命中次数等
  - 种类丰富、功能齐全、实时监测
  - 与系统层性能监控相比更细粒度、更直接反映机器实际运行状况
- 性能计数器
  - 用于测量单个性能事件的特殊用途的寄存器
  - 使用专用资源，不与应用程序竞争。
  - 精确度很高，甚至可以达到时钟级别的精度
  - 分为专用寄存器(fixed counter)、通用寄存器(general-purpose counter)

# PMU关键技术问题

- 可移植性
  - `perf_event/PAPI`尝试通过引入通用事件集来解决
    - 通用事件十分有限
    - 并非所有通用事件都适用于全平台
    - PMU相同或相似名称的事件并不意味着同一个事件

# PMU关键技术问题

- 单计数器单事件
  - 一个性能计数器只承担一个硬件事件的监测任务
  - 总耗时较长，监测效率低
- 多路复用
  - 只需完整地运行负载程序一次，期间每个性能计数器通过轮训的方式监测多个硬件事件
  - 效率高，精确度低

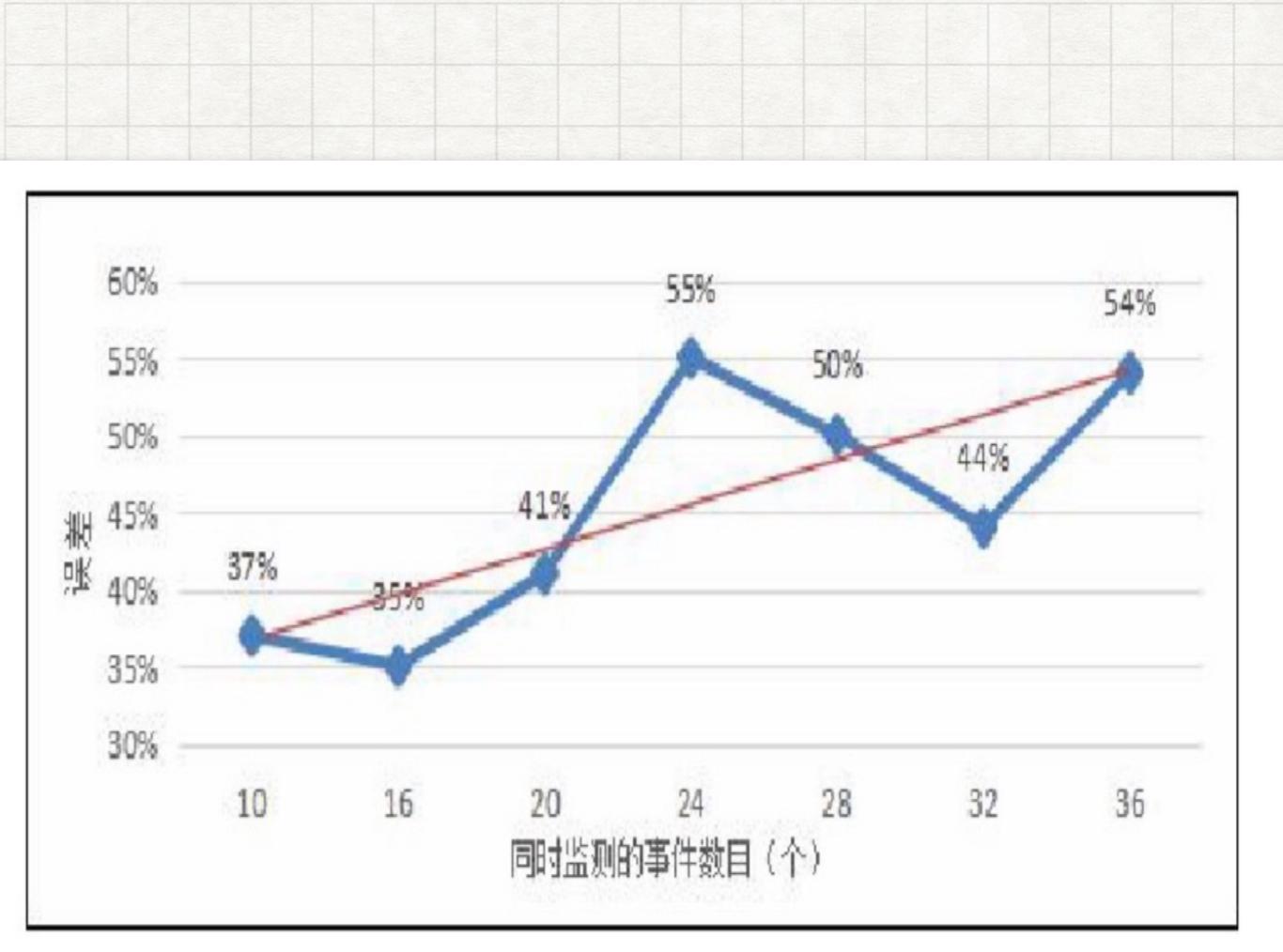
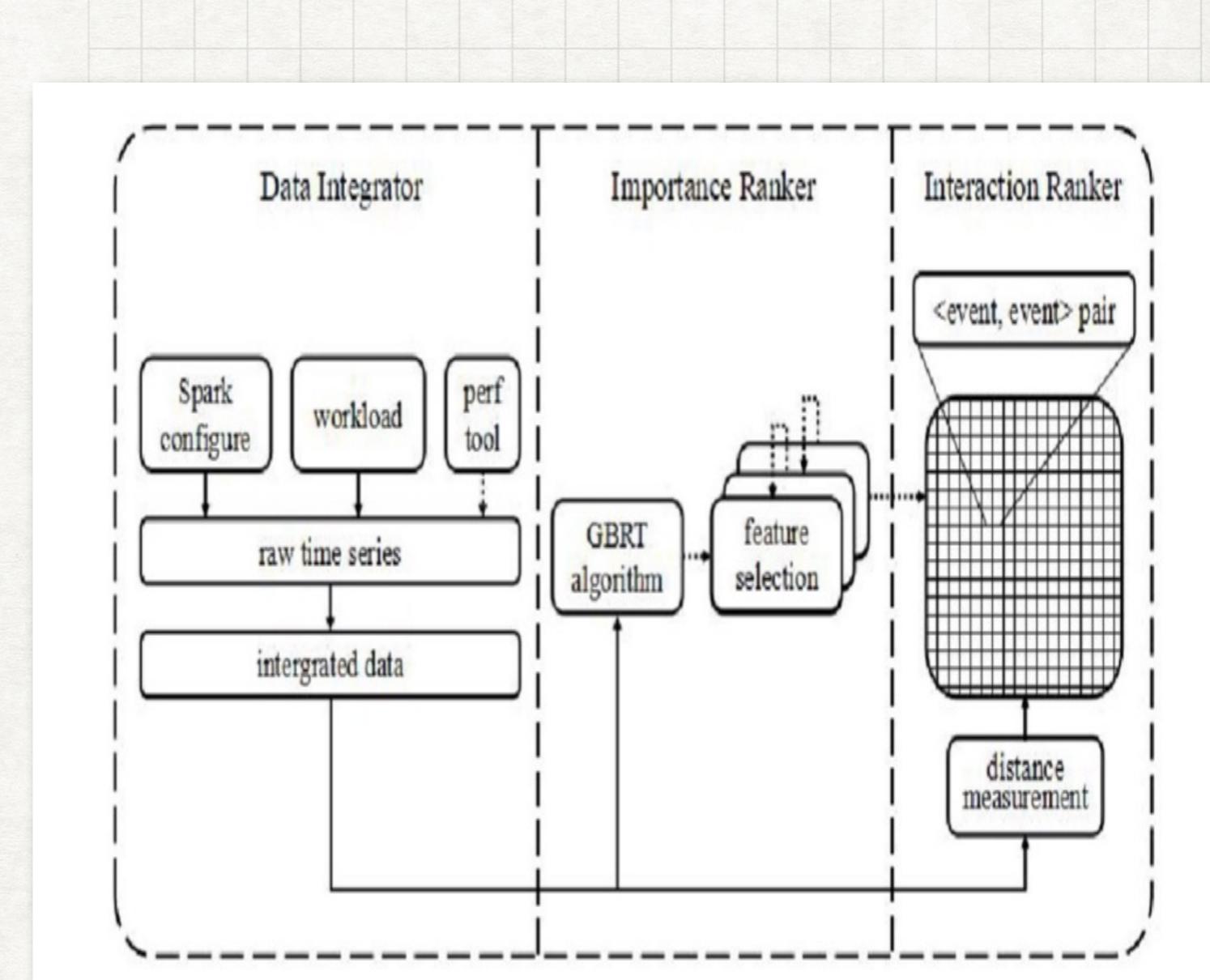


图 2.3 MLPX 方式采集硬件事件的误差变化

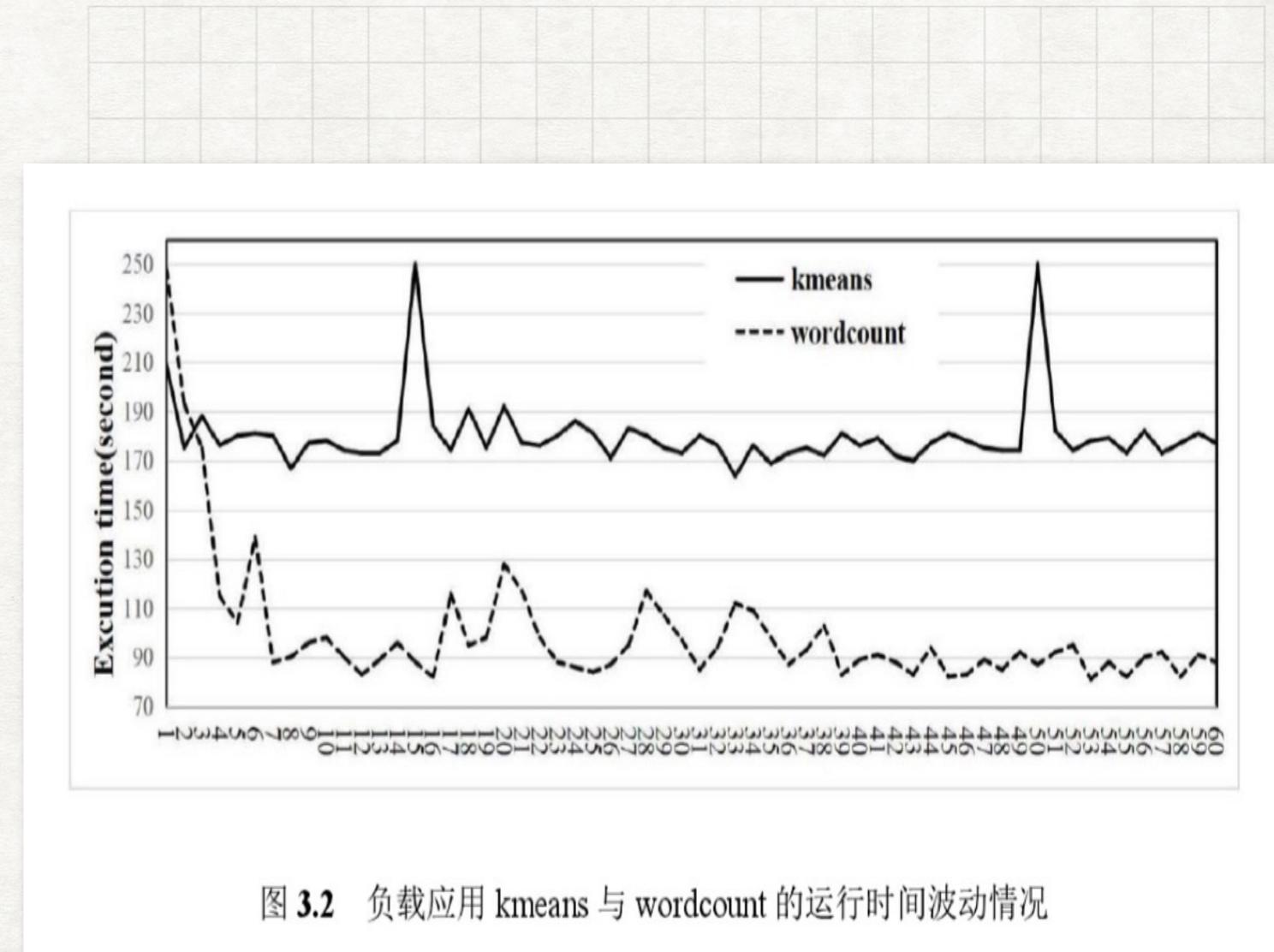
# Counter Miner

- 数据整合模块 Data Integrator
- 重要性排序模块 Importance Ranker
- 相关性排序模块 Interaction Ranker



# Data Integrator

- 数据整合与预处理
  - 使用基准测试程序来运行大数据负载
  - 使用性能计数器来收集每个硬件事件在工作负载运行期间的性能数据（raw time series）
  - 记录229个硬件事件与IPC（Instruction Per Cycle）
  - 每个硬件事件视作一维特征，IPC作为响应变量



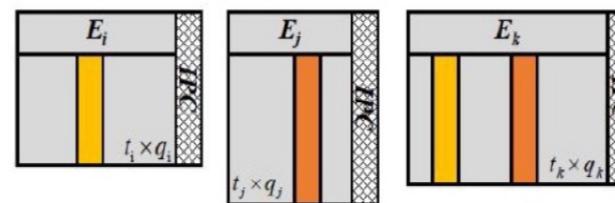
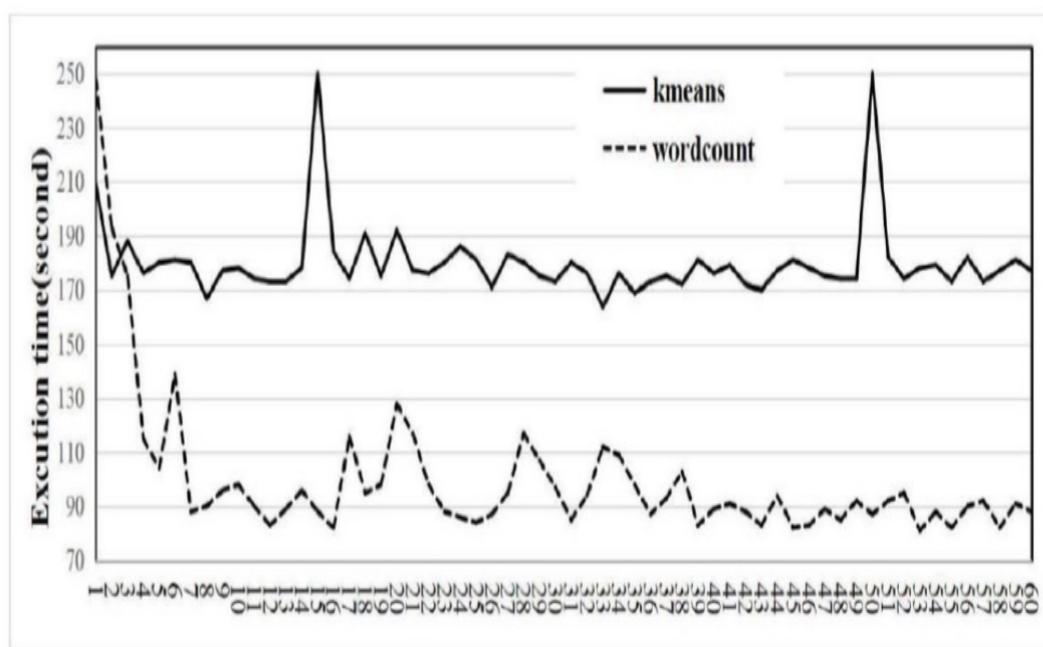
# Data Integrator

- 数据整合与预处理
  - 使用基准测试程序来运行大数据负载
  - 使用性能计数器来收集每个硬件事件在工作负载运行期间的性能数据（**raw time series**）
  - 记录**229**个硬件事件与IPC（**Instruction Per Cycle**）
  - 每个硬件事件视作一维特征，IPC作为响应变量

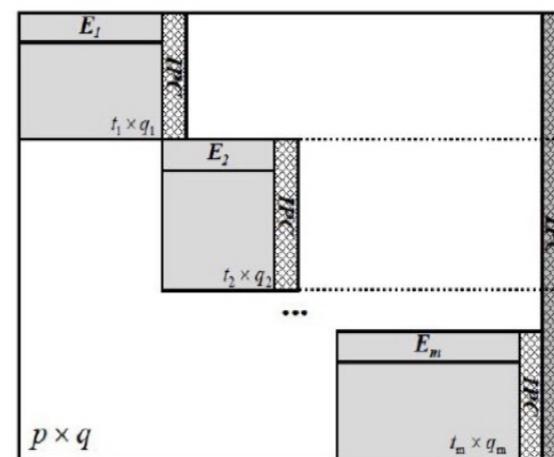
# Data Integrator

- 不同负载程序运行时长不一样
- 需要对数据进行整合与拼接
- 事件  $E_i \subseteq U$
- 硬件事件数量  $q_i = \text{card}(E_i)$
- 对硬件事件特征进行归一化

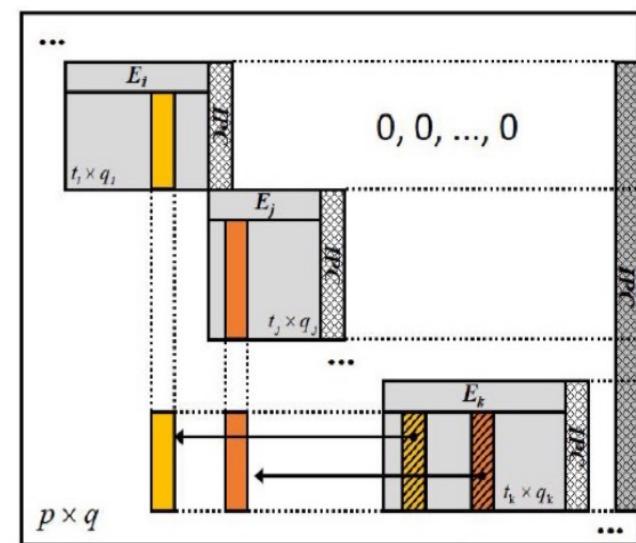
$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$



(a) 第 i 次、第 j 次和第 k 次实验监测数据



(b) 按对角线拼接，取 IPC 为响应变量



(c) 重叠事件对齐

图 3.3 数据整合

# 硬件事件空间化简与ImR模块

- GBRT算法 (Gradient Boosting Regression Tree)
  - 优点:
    - 能处理混合类型的数据
    - 预测能力强
    - 通过强大的损失函数对异常值的鲁棒性强
  - 缺点:
    - 需要按序提升，不能并行处理

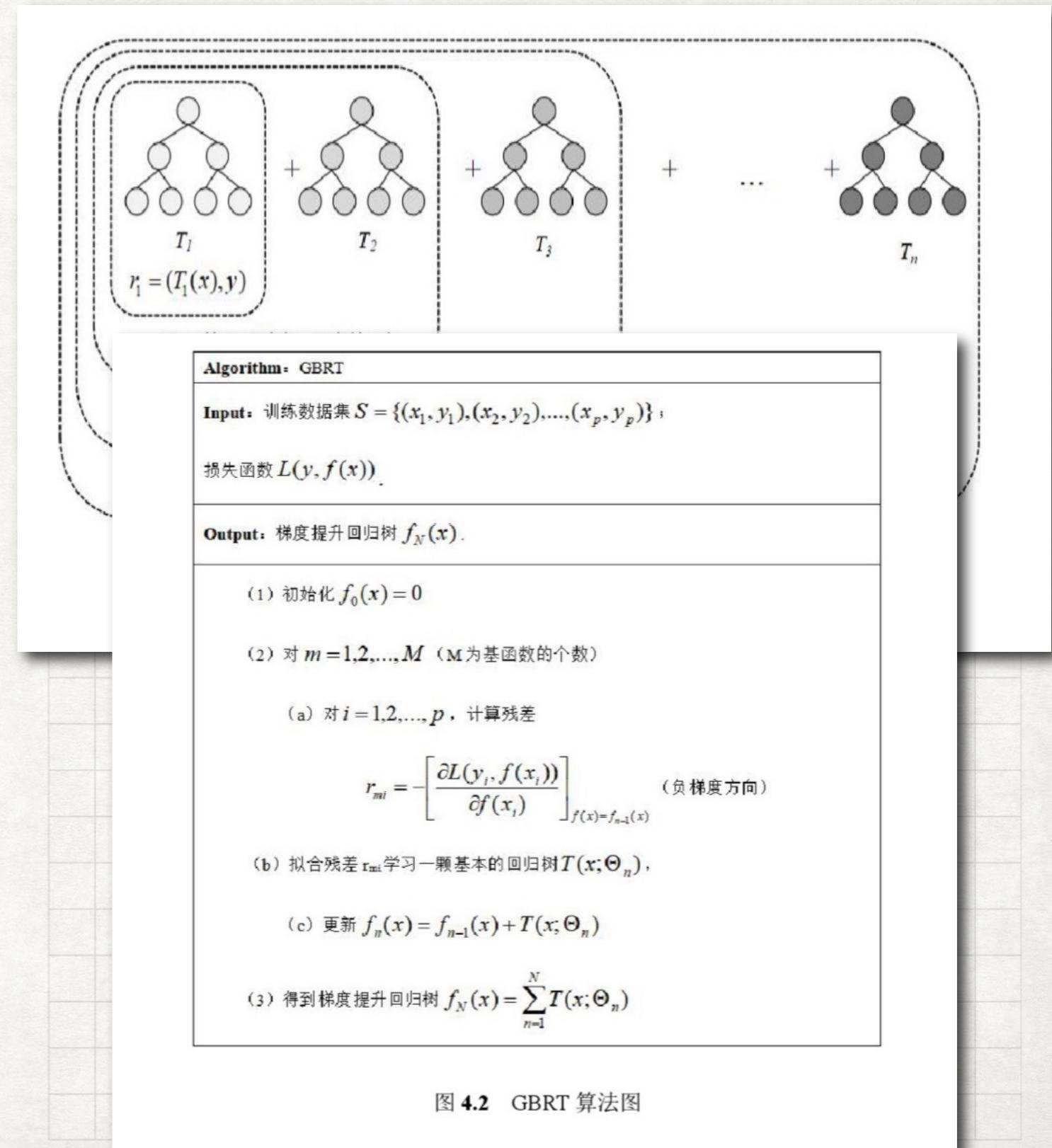


图 4.2 GBRT 算法图

# 硬件事件空间化简与ImR模块

- ImR模块的主要目的  $IPC = perf(e_1, e_2, \dots, e_n)$ 
  - 分析硬件事件的重要性
  - 量化研究受负载程序影响最大的主导因素
  - 从而帮助理解复杂情况下硬件事件结果
- 通过最小二乘损失函数获得回归树  $T$ 
$$L(y, f(x)) = (y - f(x))^2$$
- 度量每个参数变量  $e$  对目标变量的影响程度
$$I_j^2(T) = nt \cdot \sum_{i=1}^{nt} p^2(k)$$
- 解释全部回归树中  $e$  的重要程度
$$I_j^2 = \frac{1}{R} \sum_{m=1}^R I_j^2(T_m)$$

# 硬件事件空间化简 ImR模块

- ImR模块的主要目的
  - 分析硬件事件的重要性
  - 量化研究受负载程序影响最大的主导因素
  - 从而帮助理解复杂情况下硬件事件结果

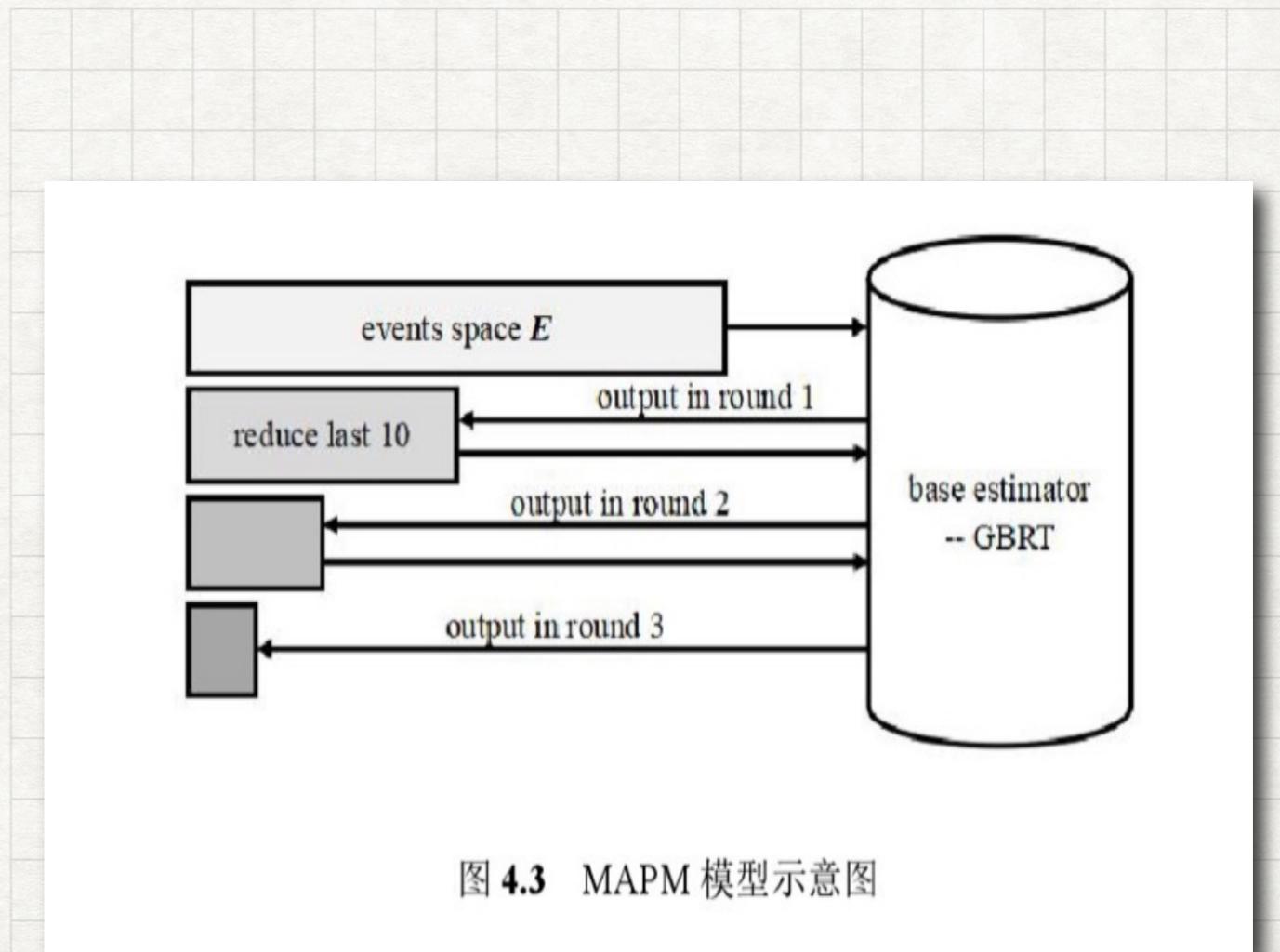
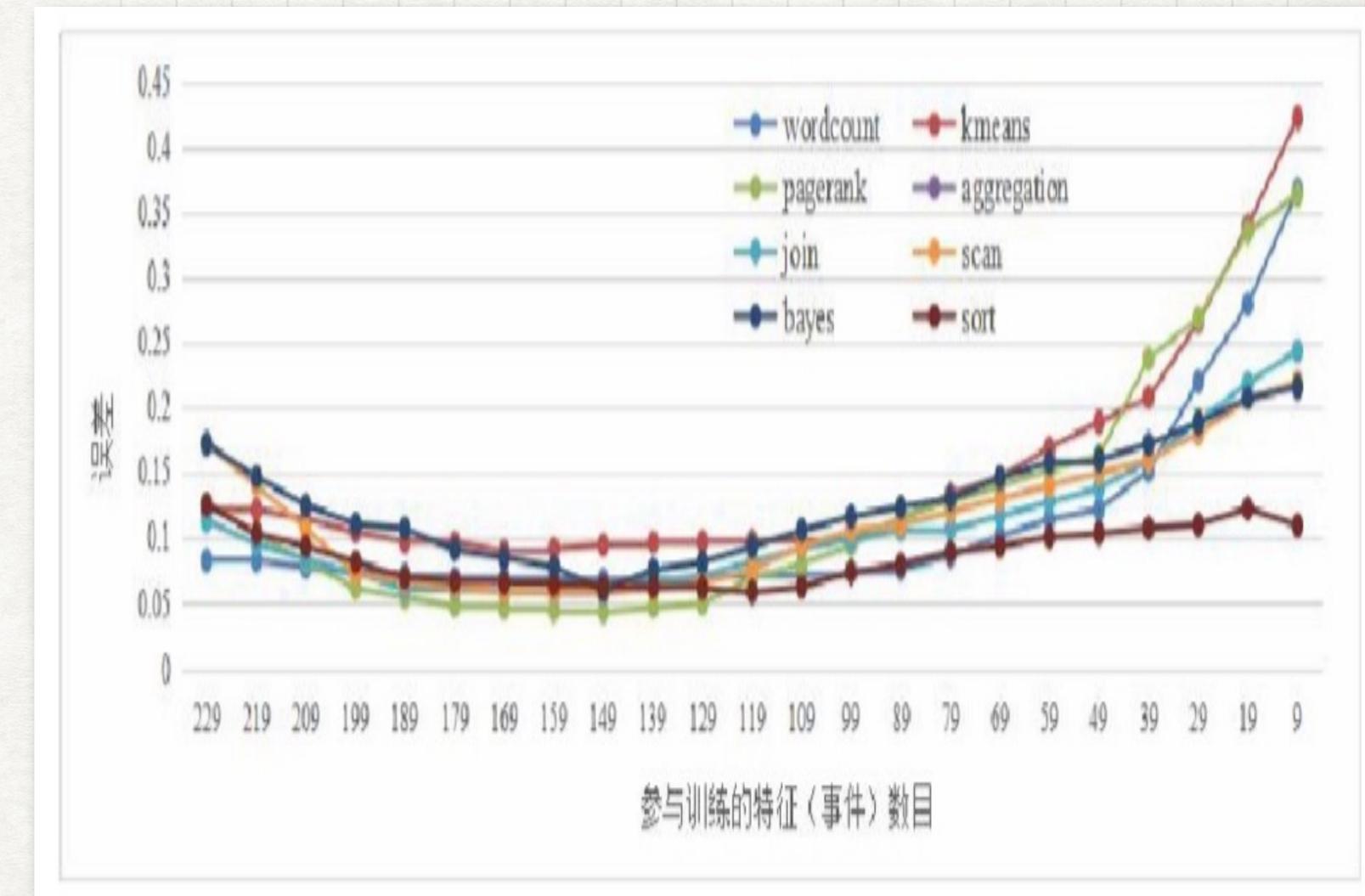


图 4.3 MAPM 模型示意图

# 硬件事件空间化简 ImR模块

- 模型精度评估

$$err = \frac{|IPC_{means} - IPC_{pred}|}{IPC_{means}} * 100\%$$



# 硬件事件空间化简 lmR模块

- 硬件事件重要性计算

$$err = \frac{|IPC_{means} - IPC_{pred}|}{IPC_{means}} * 100\%$$

**Table 4.1** 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_RETIRIED. 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_RETIRIED.ALL_BRANCHES	执行完的分支指令数。
BMRC	BR_MISP_RETIRIED.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_RETIRIED. 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_CLEAR_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_RETIRIED.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。
LRAP	L2_RQSTS.ALL_PF	所有二级缓存硬件预取指请求次数。
MRML	MEM_UOPS_RETIRIED.STLB_MISS_L OADS	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_RETIRIED.STLB_MISS_S TORES	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。
TFSA	TLB_FLUSH.STLB_ANY	从二级地址翻译缓存写出的次数。

# 案例分析

- Spark 配参问题
  - 传统的Spark配参方法
  - 基于IMR知识的Spark配参方法（IBTM）
    - 将MAPM模型得出的重要硬件性能事件与Spark参数进行关联

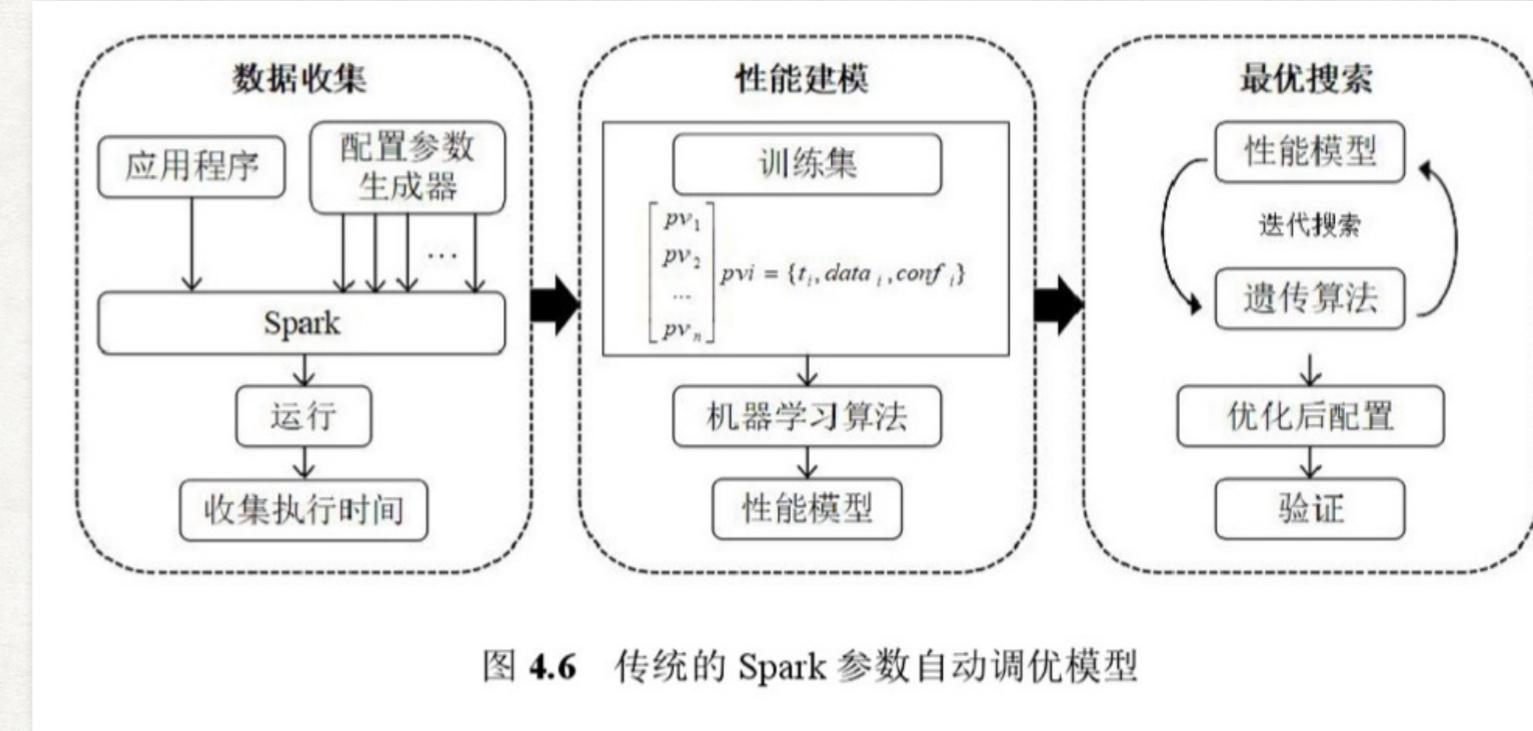


图 4.6 传统的 Spark 参数自动调优模型

$$\bar{v} = (e_1, e_2, \dots, e_n, p_1, p_2, \dots, p_n)$$

# 实验结果对比

表 4.2 传统方法与 IBTM 方法性能对比

**Table 4.2** Comparison of traditional methods with IBTM method

传统方法	IBTM 方法	
	spark 参数模型	MAPM 模型
程序运行次数	6000	60
误差	6%	12%

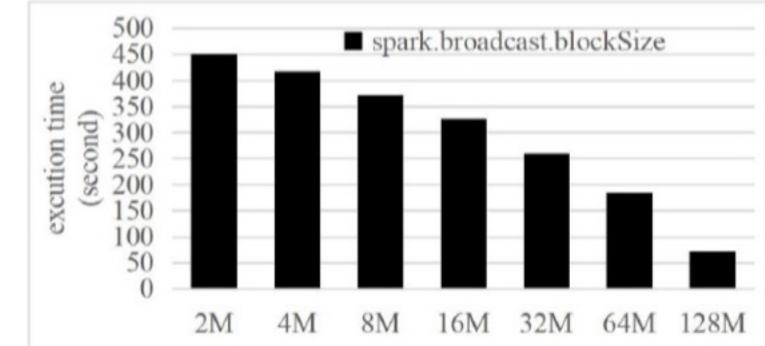
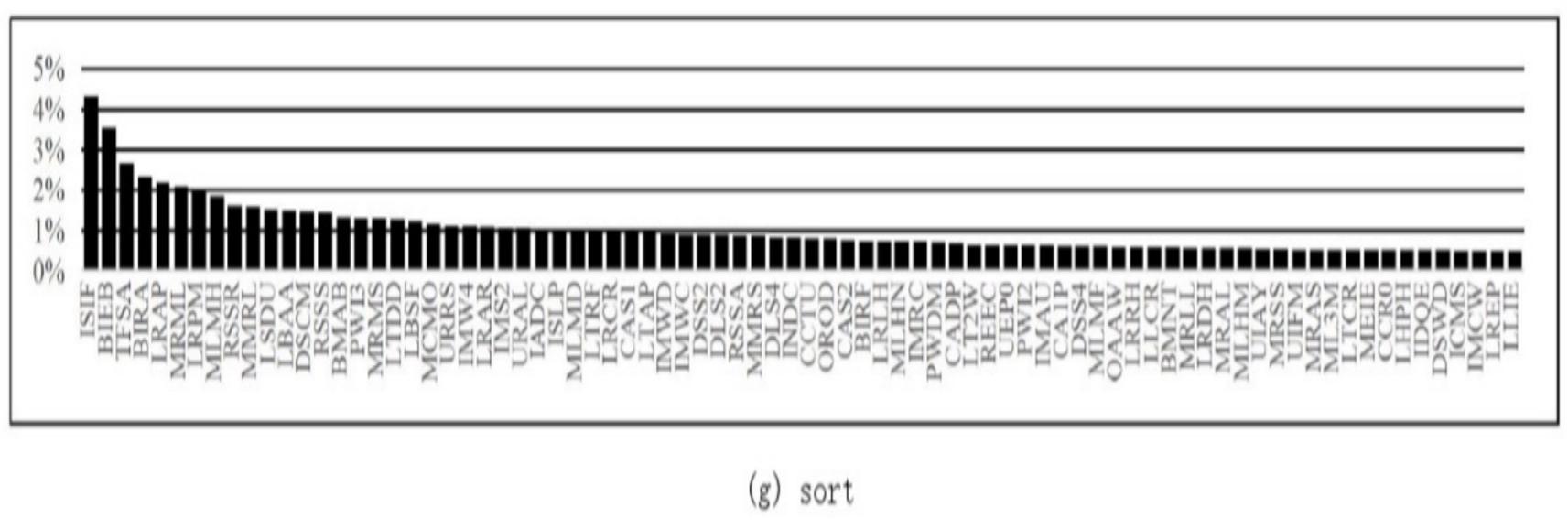


图 4.7 程序执行时间随 Spark 参数取值的变化

Figure 4.7 The execution time variance with the different Spark parameter values



# 硬件事件相关性分析

- CounterMiner中InR模块的工作
  - 两个性能事件之间的交互情况
  - 简单：L1 cache命中率与L2 cache命中率
  - 复杂：一些统计手段探测其潜在的相关性

# 基于距离的时间序列相关性度量方法

- 欧式距离

$$Distance(X, Y) = \sqrt{\sum_{t=1}^T (x_t - y_t)^2}$$

只能用于长度相同的序列比较

- 相关系数

$$r(X, Y) = \frac{Cov(X, Y)}{\sqrt{var[X] var[Y]}}$$

- DTW距离 (动态时间规整算法 Dynamic Time Wrapping)

- 优点：处理非等长时间序列相关性度量；对时间序列中的异常点不敏感
- 缺点：时间复杂度高，为 $O(mn)$ ，前两者为线性时间；DTW不满足三角不等式，欧式距离满足三角不等式

$$DTW(X, Y) = \min_p \sum_{k=1}^K d(z_k)$$

# DTW实验结果

- 6个基准测试程序下top12的硬件事件之间相关性热力图
- 白色代表两个事件相关性强

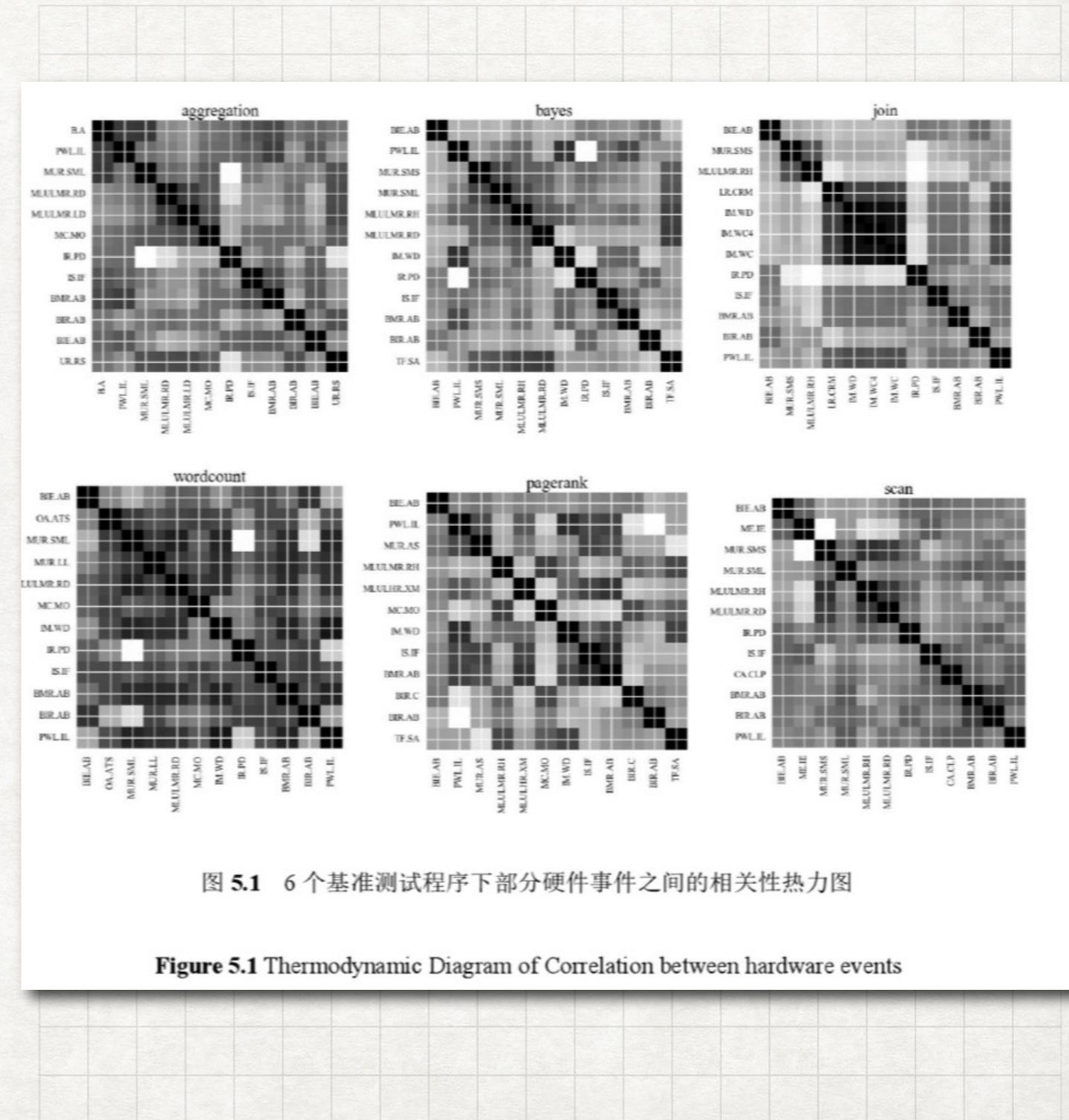


图 5.1 6 个基准测试程序下部分硬件事件之间的相关性热力图

Figure 5.1 Thermodynamic Diagram of Correlation between hardware events

# 基于线性回归模型的时间序列相关性度量算法

- 事件对 (events pair)
- 计算每一对事件的模型残差
- 对残差进行排序和归一化
- 对硬件事件的相关性进行计算

$$\nu = \sum_{i=1}^n (p_i - \bar{p})^2$$

$$I_i = \left( \frac{\nu_i}{\sum_{j=1}^n \nu_j} \right) * 100\%$$

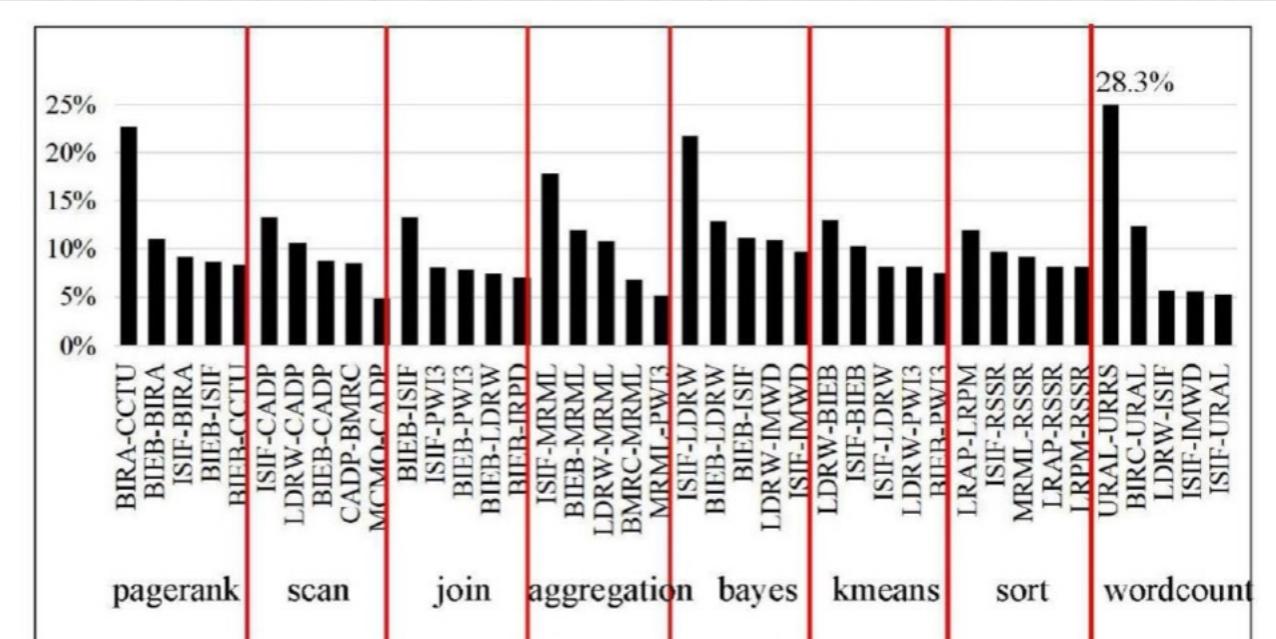


图 5.2 8 个基准测试程序单独建模后的性能事件对相关性排名（前 5）

Figure 5.2 The interaction rank of events for 8 workload models

# 总结与展望

- 总结
  - PMU检测底层硬件事件
  - DI模块对性能信息的记录进行整合与预处理
  - ImR模块提出MAPM模型，通过事件受IPC影响的重要程度排序来降低时间空间，帮助用户寻找重要事件，便于快速分析系统性能和性能调优
  - InR模块提出一种基于线性回归模型的相关性计算方法，利用模型残差衡量事件之间的相关性

# 总结与展望

- 展望
  - OCOE方式检测性能数据效率低，可以通过改进MLPX提高效率
  - 性能事件的表现会收数据量大小影响，可以从数据集角度分析性能数据
  - 快速调参方法很多，还可以使用迁移学习方法。
  - 事件时间序列分析及其对资源利用的预测