容器中大数据业务特征分析及建模技术报告

本项目研究Spark/redis/kafka参数配置优化问题。鉴于周期指令数（IPC）与程序执行时间对于系统性能的充分体现，项目建立IPC预测模型与执行时间预测模型，通过计算不同参数配置输入下的IPC与程序执行时间，找出响应系统性能的重要参数。此外，检测不同环境（即Yarn环境和Kubernetes环境）下微体系结构事件和操作系统事件，分析事件差异，为不同环境下硬件设计策略提供理论依据。

# 一、 IPC预测模型与交互强度计算

项目构建了一种以“微体系结构事件+操作系统事件+容器事件”为特征的基于机器学习方法的IPC预测模型。此模型在充分调研与深入分析的基础上提出，综合考虑多层面事件特征，相对于现有方法而言，预测结果具备更高精度。

## 1.1 相关工作

目前为止，已调研IPC预测模型相关论文3篇。本项目所参考的主要文献如下所示：

（1）Counter Miner: Mining Big Performance Data from Hardware Counters（发表会议：Annual IEEE/ACM International Symposium on Microarchitecture）

**创新点1**：论文提出一种基于随机梯度提升树的IPC预测方法。该方法首先使用Perf Tools工具采集微体系结构层事件，然后，构建基于随机梯度提升树的机器学习模型，进行IPC预测和事件重要性计算。该方法采用迭代方式筛选重要事件，在每轮迭代过程中，计算IPC预测结果和实际值的平均绝对百分比误差(MAPE),并在下一轮迭代开始前，去除重要性最低的10个事件，将剩余事件作为下一轮迭代的输入。最终，以平均误差百分比最低点所对应的事件集合作为重要事件。此方法从229个微体系结构事件中挖掘出了59个重要事件。方法处理流程如图1.1所示。

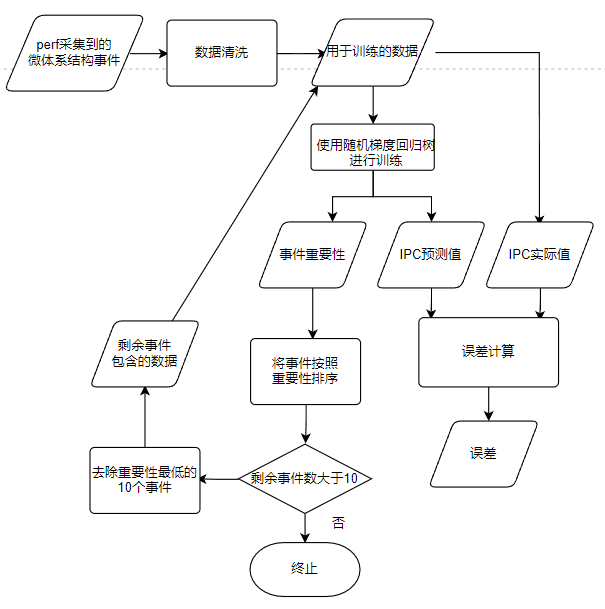


图1.1 迭代建模流程图

**创新点2**：论文提出了基于交互强度来衡量事件相关性的方法。该方法将事件两两组合构成事件对，并通过残差计算的方法得出事件对的交互强度。该文章通过计算硬件之间的交互强度，发现在160个交互对中，有83.4%的交互对包含branch相关的事件，进而得出branch事件很重要的结论。

交互强度的计算方法如下，步骤1.至步骤4.可参考流程图1.2.

1. 构造交互对：

任意两个事件两两组合构成交互对(记为事件A和事件B),M个事件可以构成 M\*(M-1)/2 个交互对

1. 构造数据：

将一次采集过程中事件A所遇到的最小值和最大值之间N等分，事件B也是在最小值和最大值之间N等分，共(N+1)\*(N+1)份数据。其他事件取各自的平均值，例如事件C在(N\*1)\*(N+1)份数据中全部取事件C的平均值。

1. 预测数据：

使用图1.1建模得出的机器学习预测模型预测第2.步构造出的数据，得到IPC预测值记为predictions

1. 线性拟合预测数据：

使用线性回归算法拟合第3.步预测数据，得到拟合残差记为residual。

5. 重复1-4直到计算完所有交互对的残差。

6. 交互强度:

将所有残差累加起来记为sum\_residual，并计算每个事件对的residual占sum\_residual的比例，得到交互强度，所有事件对的交互强度总和为1.0。

**不足**：仅以微体系结构层事件作为模型输入，无法全面反应系统实际运行状态。

**启发**：在微体系结构层事件基础上，融入其他层特征，以准确刻画系统实际性能。

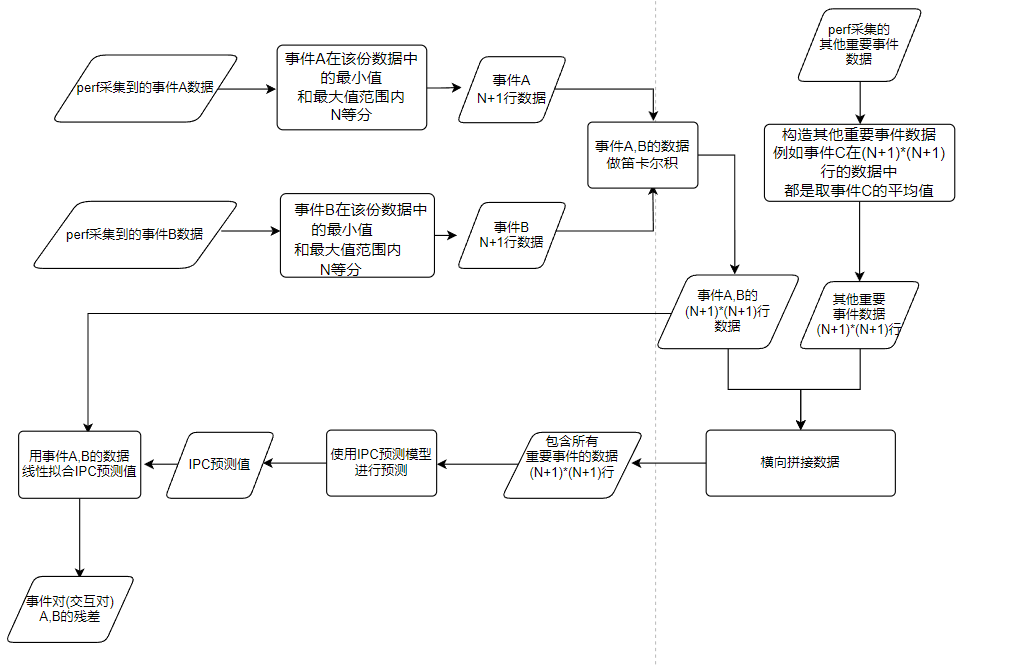


图1.2 交互强度计算流程图

（2）MIA: Metric Importance Analysis for Big Data Workload Characterization(发表会议: IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS)

**创新点1**:论文提出一种基于随机梯度提升树(SGBRT)的IPC预测方法，该方法使用随机梯度提升树(SGBRT)构建了一个以微体系结构事件和应用层事件为输入的IPC预测模型，然后通过和图1.1类似的方式计算出事件重要性。相对于(1)，该文章的改进之处在于融入了除微体系结构事件以外的事件进行IPC建模。

**创新点2**:论文提出一种基于事件重要性的用例相似性度量方法，该方法使用随机梯度提升树构建的机器学习模型输出微体系结构事件和应用层事件的重要性。进而通过事件的重要性构建出一个用于衡量用例相似性的指标称之为MKP(MIA-based Kiviat Plot)，该指标可以直观的反映出用例的运行特性。MKP示例见图1.3。用例相似的判定逻辑为，如果两个用例的MKP图相似，那么可以认为两个用例相似。MKP的重要意义在于，它通过重要事件相似性的方法去判定用例的相似性，对于一组相似的用例，只需对其中一个用例进行研究即可。

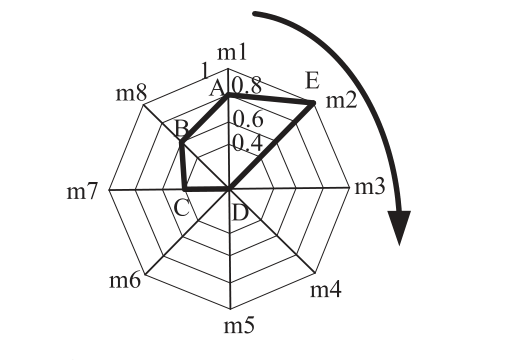


图1.3 MKP图例

**不足:** MKP图只刻画了微体系层和应用层事件的重要性，考虑事件不够全面。

**启发**：可以添加其他层（例如操作系统层）的事件，以准确反映用例特性。

## 1.2 创新方法

在上述工作的启发上，我们提出基于LightGBM的IPC预测方法。该方法的改进之处在于综合考虑了微体系结构层，操作系统层，容器层的事件，相当于包含硬件级，系统级，应用级三个方面。

该方法流程如下图1.4所示。

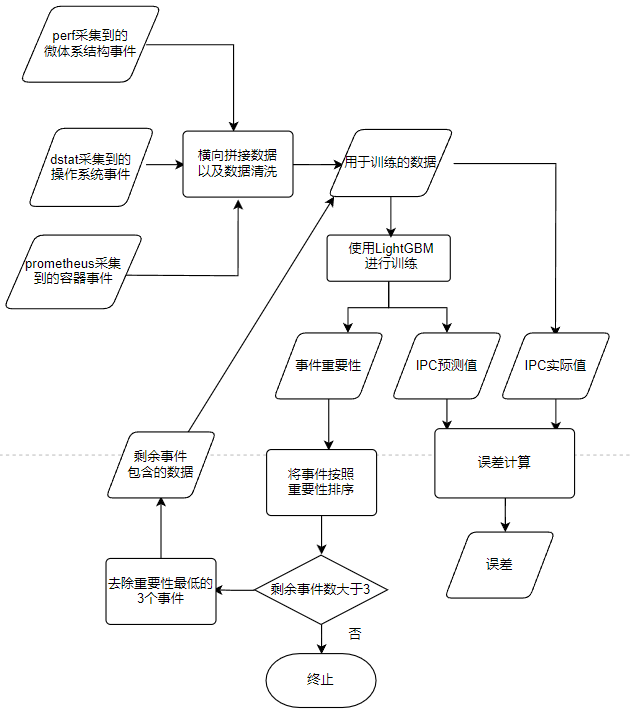


图1.4 基于LightGBM的IPC预测模型流程图

## 1.3 实验环境及测试用例

搭建以1个master节点、2个worker节点所构成的集群环境，所有节点统一使用Centos7 2009操作系统，各节点硬件配置如表1.1所示，集群组网图如图1.5所示，集群所使用软件版本如表1.2所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 节点名称 | CPU规格 | 内存 | 磁盘 |
| Master | Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz 20核 | 128G PC4 | 13T |
| Worker01 | Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz 20核 | 128G PC4 | 13T |
| Worker02 | Intel(R) Xeon(R) Gold 5218R CPU @ 2.10GHz 20核 | 128G PC4 | 13T |

表1.1集群节点硬件配置

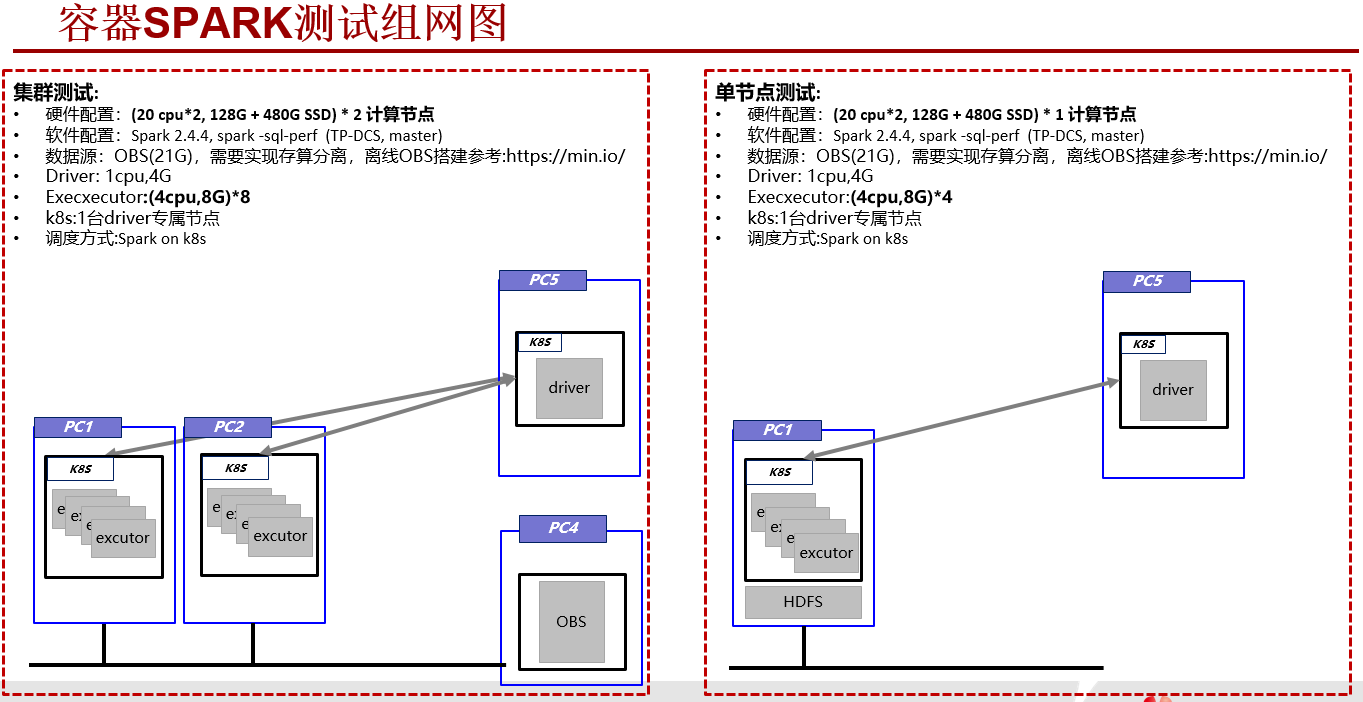


图1.5集群组网图

|  |  |
| --- | --- |
| 软件名称 | 版本号 |
| Spark | 2.4.4 |
| Docker | 20.10.5 |
| Kubernetes | 1.19.1 |
| Hibench | 6.0 |
| TPC-DS | 2.3.0 |

表1.2集群节点软件版本

测试用例如表1.3所示，其中，wordcount和terasort来自Hibench。

|  |  |
| --- | --- |
| 测试用例 | 数据量大小 |
| Wordcount | 50G,75G,100G |
| Terasort | 20G |
| TPC-DS中全部103个query | 21G |

表1.3测试用例

后续所有测试的实验环境都是以上配置。

## 1.4 测试结果

#### 1.4.1 IPC预测测试

**（1）用现有方法的测试**

**Wordcount测试**

数据源: 随机生成Spark的配置并运行WordCount50G, WordCount75G, WordCount100g。运行时收集微体系结构，操作系统和容器事件。

只用微体系结构事件作为模型输入，Instructions Per Cycle(IPC)作为预测目标，构建IPC预测模型，按图1.1的流程进行建模（机器学习算法由SGBRT变为LightGBM）。下图是模型训练过程中剩余事件数和误差(MAPE)的关系图。

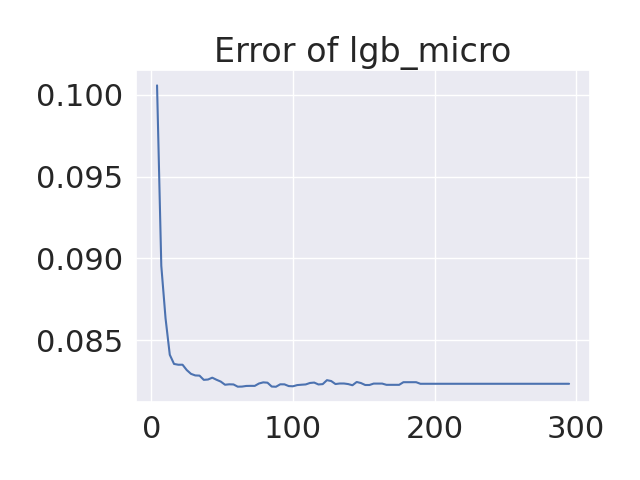


图1.6 微体系结构事件预测IPC\_误差图

在剩余83个事件的时候，达到误差最低点0.082。

**（2）用我们的方法测试**

将微体系结构，操作系统，容器的事件组合作为模型输入，IPC作为模型输出,基于LightGBM构建模型。

1. **WordCount测试**

数据源和(1)同。

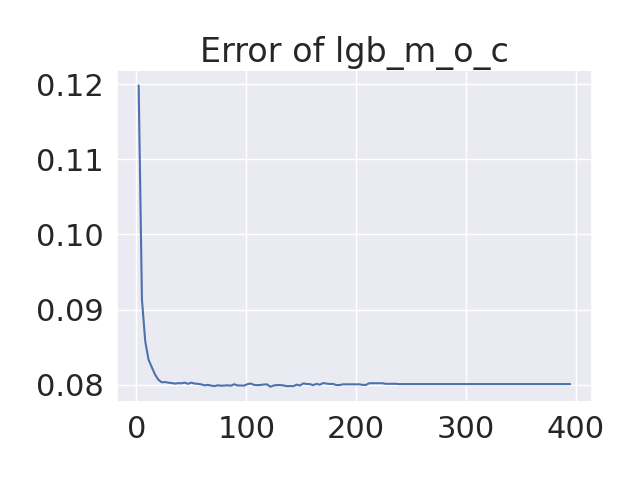


图 1.7 WordCount\_三层事件预测IPC\_误差图

在使用123个事件时达到误差最低点0.079。

1. **TeraSort 测试**

数据源: 随机生成Spark的配置并运行TeraSort20G,运行时收集微体系结构，操作系统和容器事件。

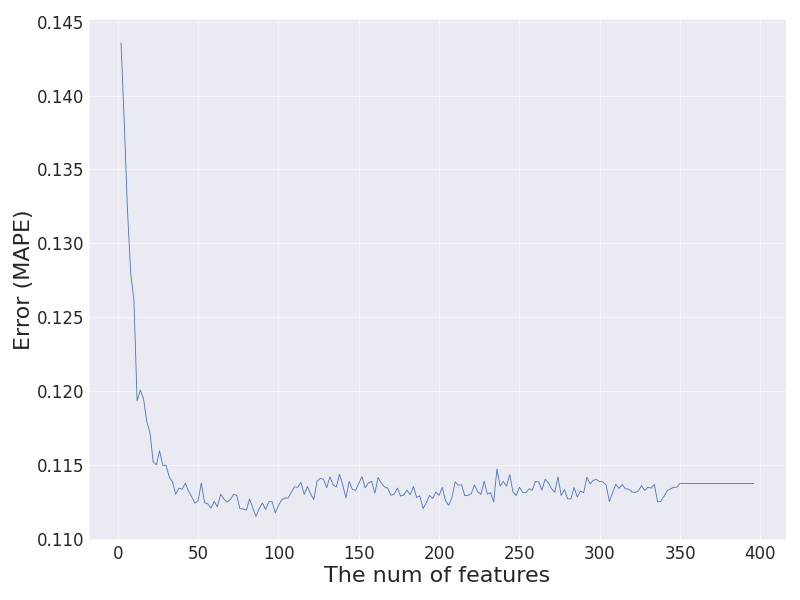


图1.8 TeraSort\_三层事件预测IPC误差图

在87个事件时达到误差最低点0.112。

**（3）测试结果比较**

**a. WordCount**

将(1)现有方法WordCount测试结果和(2)我们的方法WordCount测试结果进行比较，可以发现将三层事件作为IPC预测模型的输入，可以提高IPC预测模型的精度(由 0.082变为0.079)。然而，这里的精度提高是非常有限的(只有0.003)。我们猜测可能的原因在于数据清洗的过程存在的问题。因为采集工具的原因，perf采集10条微体系结构事件的时间内，dstat采集了11条操作系统层事件，为了让数据量相同，去除了操作系统层的一些数据，这就导致了训练数据存在一定程度的失真。

**b. TeraSort**

由于在WordCount的测试中，我们发现将三层事件作为IPC预测模型的输入 可以达到更高的精度。所以对于TeraSort，我们只构建了三层事件作为输入的IPC预测模型。

#### 1.4.2 交互强度测试

交互强度的计算首先要通过IPC预测模型挖掘出重要事件，所以在此部分我们将IPC建模的部分也进行简要展示。

**（1）WordCount**

数据源: WordCount100G的运行数据。

IPC预测模型训练结果如下图1.9所示。

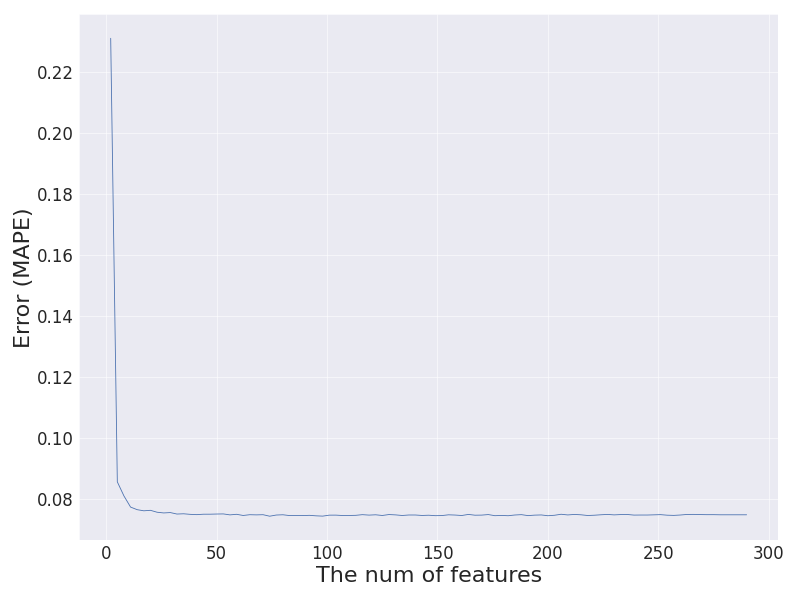


图 1.9 WordCount100G\_ IPC预测误差图

如果选出的的事件太多，那么会大大增加交互对的计算量，例如选20个事件，那么就有20\*19/2=190个交互对。所以这里我们对误差的要求**不再是取最低点，而是只要误差小于阈值0.1**(通过论文调研工作，我们认为0.1-0.15是一个合理的阈值选择)，那么就可以继续减少事件。在上图1.9中最终在留下6个事件的时候，误差达到0.08,这时如果再继续减少事件就会造成误差大于等于阈值，因此我们最终选6个事件来计算交互强度。

用于计算交互对的6个事件及其重要性如下图，以最重要的事件重要性为1.0，其余事件的重要性取其占最重要事件重要性的比例。

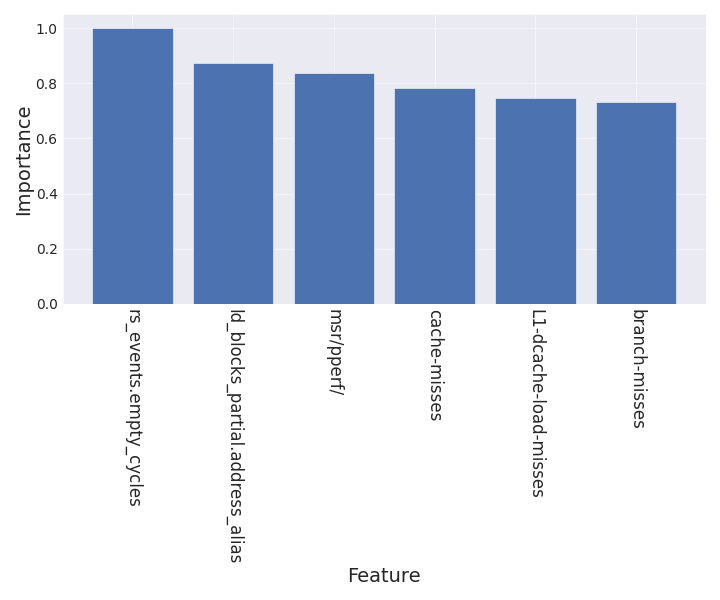


图 1.10 WordCount用于计算交互强度的微体系结构事件

|  |  |  |
| --- | --- | --- |
| 名称 | 描述 | 类别 |
| rs\_events.empty\_cycles | Cycles when Reservation  Station (RS) is empty  for the thread | Pipeline |
| ld\_blocks\_partial.address\_alias | False dependencies in MOB  due to partial compare  on address | pipeline |
| msr/pperf/ |  | Kernel PMU event |
| cache-misses | A cache miss is an event in  which asystem or application  makes a request  to retrieve datafrom a cache,  but that specific data is not  currently in cache memory. | Hardware event |
| L1-dcache-load-misses |  | Hardware cache event |
| branch-misses |  | Hardware event |

表 1.4 WordCount用于计算交互强度的事件列表

表1.4的6个事件可以组成15个事件对(6\*5/2=15)。这些事件对的交互强度计算结果如下图，取前10个重要的交互对进行举例说明。

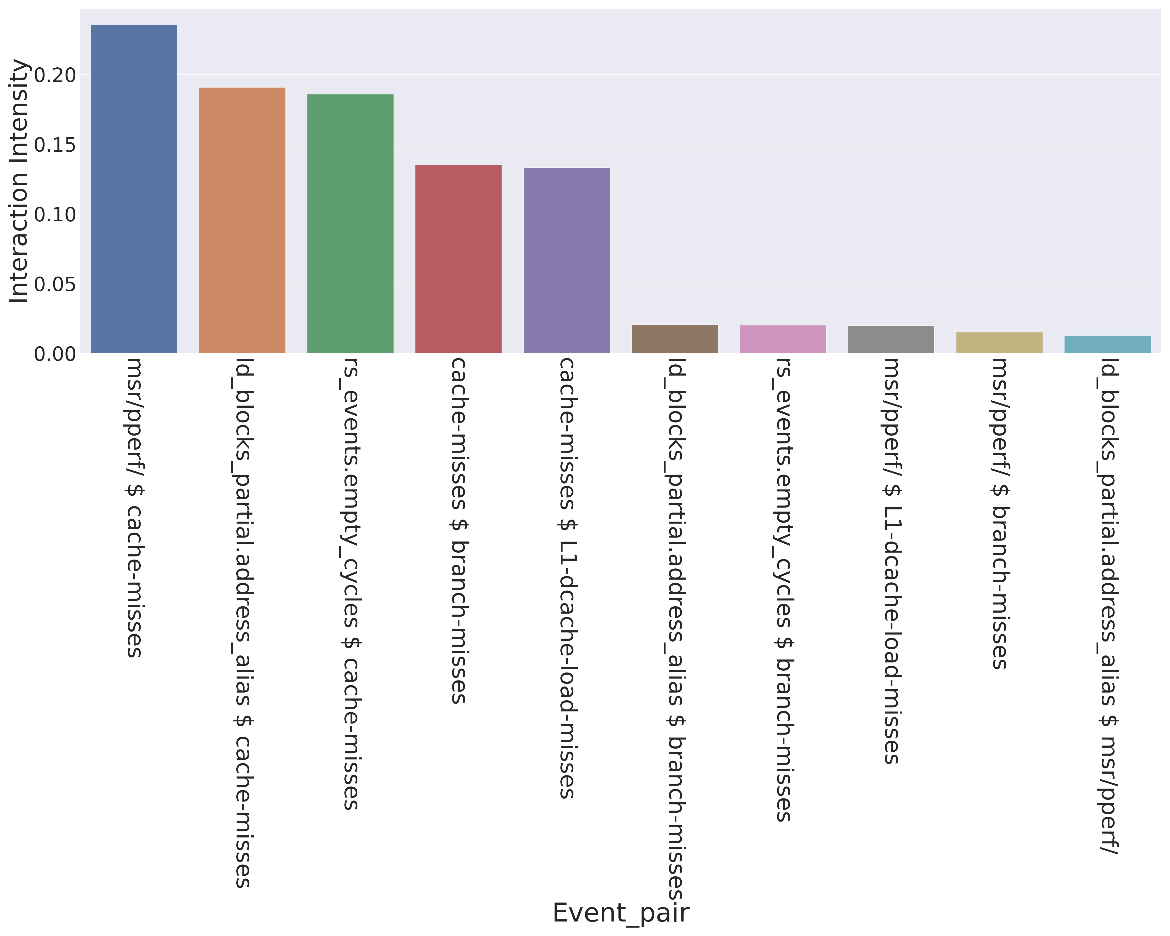


图1.11 WordCount重要事件交互强度

|  |  |
| --- | --- |
| 事件对 | 交互强度 |
| msr/pperf/ cache-misses | 0.236 |
| ld\_blocks\_partial.address\_alias cache-misses | 0.191 |
| rs\_events.empty\_cycles cache-misses | 0.186 |
| cache-misses branch-misses | 0.135 |
| cache-misses L1-dcache-load-misses | 0.133 |
| ld\_blocks\_partial.address\_alias branch-misses | 0.021 |
| rs\_events.empty\_cycles branch-misses | 0.020 |
| msr/pperf/ L1-dcache-load-misses | 0.020 |
| msr/pperf/ branch-misses | 0.015 |
| ld\_blocks\_partial.address\_alias msr/pperf/ | 0.013 |
| rs\_events.empty\_cycles msr/pperf/ | 0.009 |
| rs\_events.empty\_cycles ld\_blocks\_partial.address\_alias | 0.009 |
| rs\_events.empty\_cycles L1-dcache-load-misses | 0.006 |
| L1-dcache-load-misses branch-misses | 0.003 |
| ld\_blocks\_partial.address\_alias L1-dcache-load-misses | 0.002 |

表1.5 WordCount重要事件交互强度

图1.11事件之间使用 $ 分割，从图1.11和表1.5中的结果说明前五组交互对的交互强度很大，而且都和cache-misses有关。这表示与cache-misses关联的事件很多，也从另一方面说明cache-misses具有很高的重要性。

**（2）TeraSort**

数据源: TearSort20G。

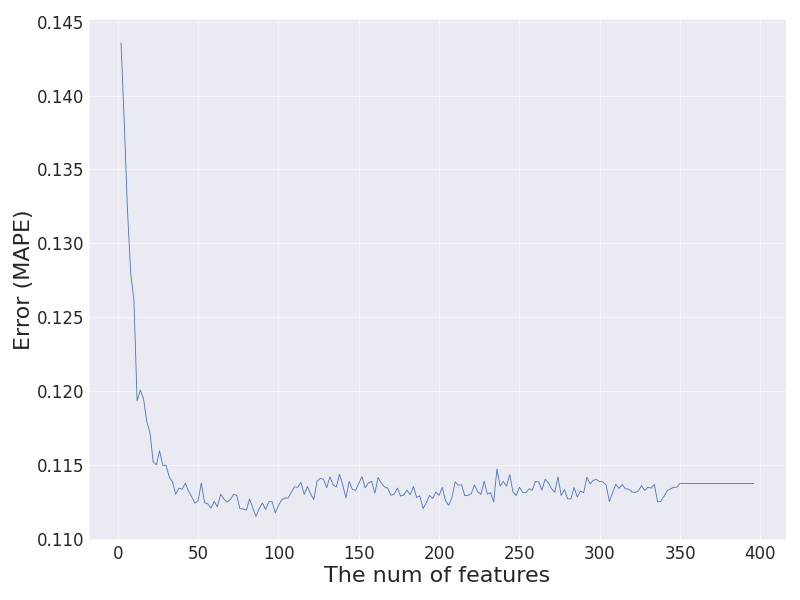


图1.12 TeraSortG\_ IPC预测误差图

由图1.8的结果可知，TeraSort的IPC预测模型很难将误差降到0.10以下(增大数据量也无法降到0.1)。因此这里使用0.13作为阈值，只要模型误差小于0.13，就可以继续减少事件。

最终在剩余9个事件时，误差为0.128，剩余的9个事件如下。

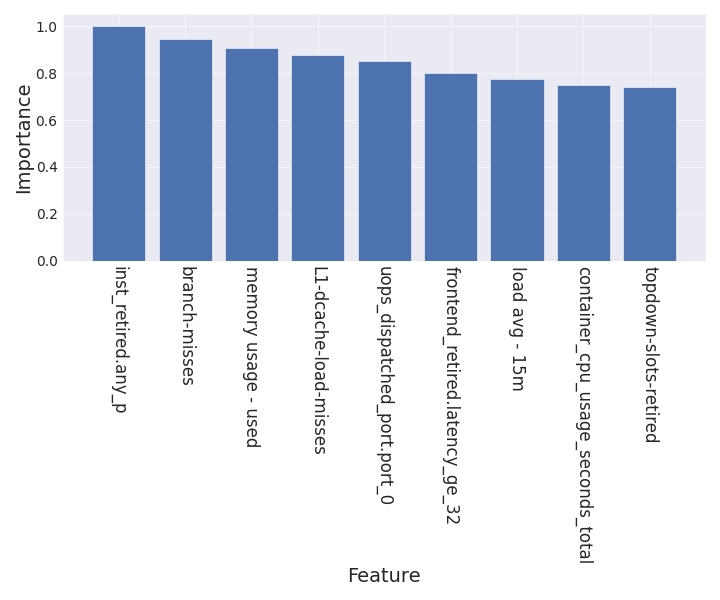


图 1.13 TeraSort用于计算交互强度的事件

|  |  |  |
| --- | --- | --- |
| 事件名 | 描述 | 类别 |
| inst\_retired.any\_p | Number of instructions retired. General Counter - architectural event Spec update: SKL091, SKL044 | 微体系结构  pipeline |
| branch-misses |  | 微体系结构  Hardware event |
| memory usage - used | 内存使用量 | 操作系统  内存使用相关 |
| L1-dcache-load-misses |  | 微体系结构  Hardware cache event |
| uops\_dispatched\_port.port\_0 | Cycles per thread when uops are executed in port 0 | 微体系结构  pipeline |
| frontend\_retired.latency\_ge\_32 | Retired instructions that are fetched after an interval where the front-end delivered no uops for a period of 32 cycles which was not interrupted by a back-end stall (Precise event) | 微体系结构  fronted |
| load avg - 15m |  | 操作系统  负载均衡统计 |
| container\_cpu\_usage\_seconds\_total |  | 容器 |
| topdown-slots-retired |  | 微体系结构  Kernel PMU event |

表 1.6 TeraSort用于计算交互强度的事件

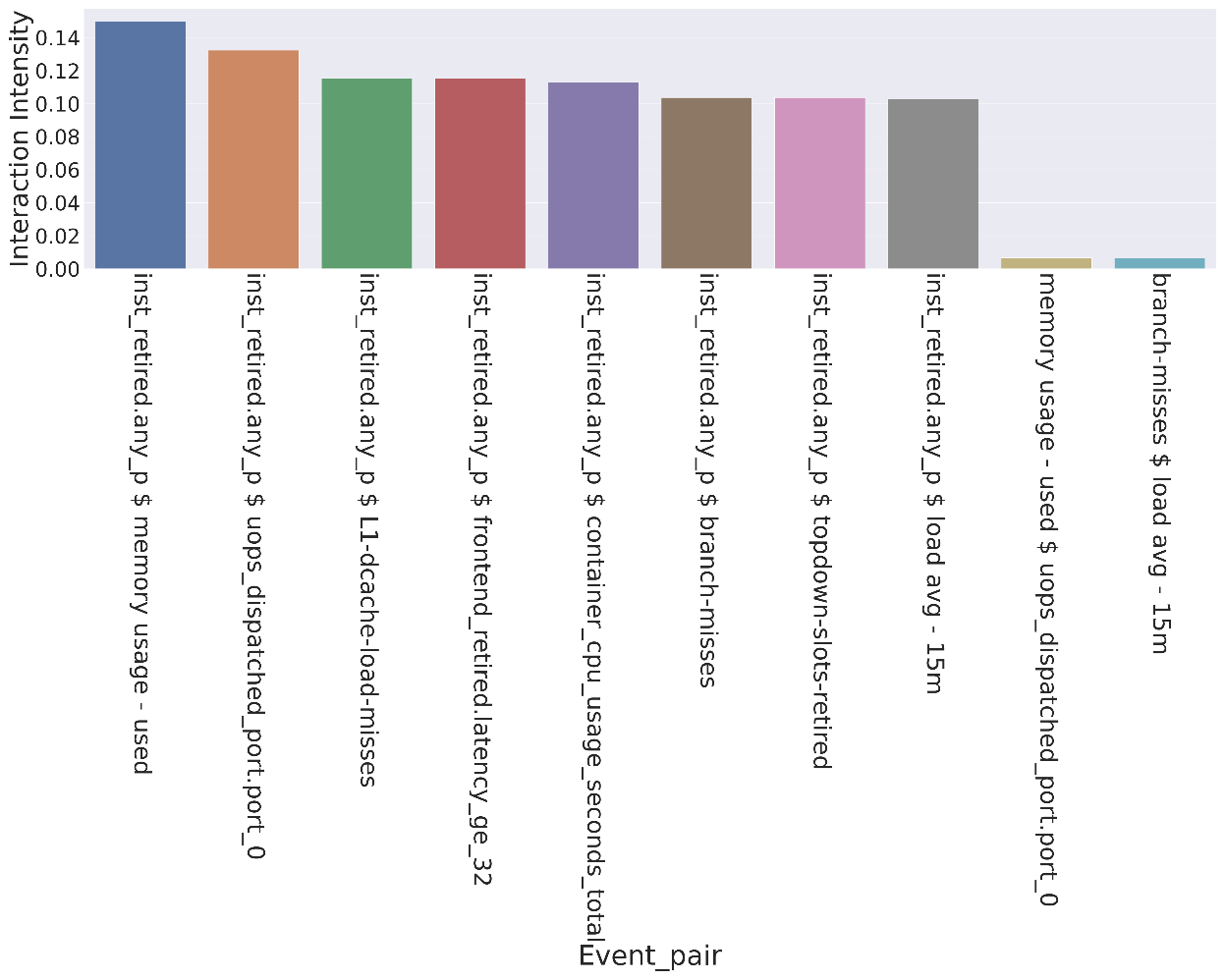


图 1.14 TearSort前10个交互对交互强度

|  |  |
| --- | --- |
| 事件对 | 交互强度 |
| inst\_retired.any\_p memory usage - used | 0.150 |
| inst\_retired.any\_p uops\_dispatched\_port.port\_0 | 0.132 |
| inst\_retired.any\_p L1-dcache-load-misses | 0.115 |
| inst\_retired.any\_p frontend\_retired.latency\_ge\_32 | 0.115 |
| inst\_retired.any\_p container\_cpu\_usage\_seconds\_total | 0.113 |
| inst\_retired.any\_p branch-misses | 0.104 |
| inst\_retired.any\_p topdown-slots-retired | 0.104 |
| inst\_retired.any\_p load avg - 15m | 0.103 |
| memory usage - used uops\_dispatched\_port.port\_0 | 0.007 |
| branch-misses load avg - 15m | 0.006 |

表1.7 TeraSort前10个交互对交互强度

在计算交互强度的事件中和误差最低点的事件中，全都包含操作系统事件memory usage - used,并且memory usage - used和inst\_retired.any\_p 组成的交互对具有最高的交互强度，进一步说明内存对于TeraSort的执行具有非常重要的影响。

# 二、执行时间预测模型

项目构建了一种以大数据框架配置参数为模型输入的基于机器学习方法的执行时间预测模型。此模型在充分调研与深入分析的基础上提出，它从应用的层面挖掘出影响用例运行的配置参数。

## 2.1 相关工作

目前为止，已调研执行时间预测模型相关论文3篇。本项目所参考的主要文献如下所示：

（1）Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing（发表会议：ASPLOS）

**创新点1**： 论文提出一种数据感知的自动调优方法。它可以将输入数据集的大小和41个配置参数作为给定spark程序的性能模型的参数，并结合遗传算法搜索最优配置。构建模型的关键思想是通过多个简单模型的合作来预测性能，而不是单个复杂模型。如公式2.1和图2.1所示，第一步使用回归树来构建子模型M1，作为输入数据集大小和配置参数的函数。第二步，增加训练样本构建一个不同的回归树模型(M2)，以反映M1没有捕获到的Spark作业执行时间的变化。此时，将前两个子模型α1M1和α2M2结合，生成一个初始组合模型CM1，其中M1和M2为两个子模型预测的执行时间，α1和α2分别为学习率。如果此时的模型精度达到90%，则停止训练，否则构建新的回归树模型与原有模型结合，构建新的组合模型。递归构建模型，直到精度满足90%的要求。HM本质上是一个层次化的顺序过程，在该过程中随着HM的进行，模型变得更加准确。



公式2.1 性能模型输入的性能向量

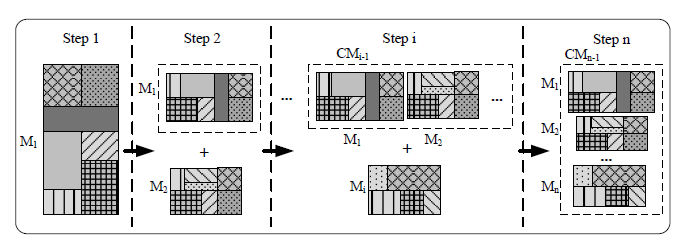


图2.1 层次建模流程图

**不足**：随机抽样构建训练样本可能会导致产生性能极差样本，特别是对于TPCDS等运行时间较长的负载，性能极差样本可能运行一次程序需要3h，极大增加了调优成本。

**启发**：采用其他抽样方式或设置阈值，当执行时间超过阈值时关掉程序并记录该配置不再对该配置附近进行抽样。

### 2.2 前期改进工作

配置参数预测执行时间的性能模型和IPC性能模型类似，只是将**模型输入由采集的各层事件变为配置参数，预测目标由IPC变为程序执行时间，迭代筛选事件变为迭代筛选配置参数。**

为了获取训练样本，我们在一定范围内(该范围会随着具体的程序发生变化)随机生成配置参数，然后执行程序获取执行时间。

和IPC预测的不同之处在于，**执行时间预测模型的构建面临着获取数据样本非常耗时的问题**。因为在IPC性能预测模型中，每运行一次用例都可以采集到上百条甚至上千条数据(每秒采集一次数据),所以通常运行不到60次就可以获得一个精度较高的IPC预测模型(误差一般低于15%)。而在执行时间预测模型的构建中，每运行一次用例只能采集到一条样本（配置参数，执行时间）。这就使得为了获取足够的样本，需要花费很长的时间。

为了解决样本获取花费时间过长的问题，我们从三方面进行了改进。

#### 2.2.1 初步筛选参数

预测模型的输入维度越高，模型训练需要的数据量也就会越多。为此我们考虑将一些对程序性能影响很小的配置参数直接排除，以达到降低数据维度的目的。

为了选出这些需要被排除的的配置参数，我们**一方面是根据自己构建的模型训练得出的结果。比如spark.rpc.message.maxSize在模型训练过程计算出的重要性一直排在末尾，那么这种参数对性能影响很小，我们在以后的样本采集中就不再配置该参数**。另一方面是根据以往论文的研究成果进行筛选。在进行筛选后，目前考虑的配置参数个数降到了27个，详见下表2.1。

表2.1的配置参数是针对WordCount和TeraSort而言。

tpcds的配置参数见表2.2

|  |  |
| --- | --- |
| spark.memory.offHeap.enabled | spark.memory.fraction |
| spark.memory.offHeap.size | spark.memory.storageFraction |
| spark.executor.memory | spark.rdd.compress |
| spark.executor.cores | spark.reducer.maxBlocksInFlightPerAddress |
| spark.executor.memoryOverhead | spark.reducer.maxReqsInFlight |
| spark.executor.instances | spark.reducer.maxSizeInFlight |
| spark.broadcast.blockSize | spark.scheduler.mode |
| spark.broadcast.checksum | spark.scheduler.revive.interval |
| spark.broadcast.compress | spark.shuffle.compress |
| spark.default.parallelism | spark.shuffle.file.buffer |
| spark.kryoserializer.buffer | spark.shuffle.io.numConnectionsPerPeer |
| spark.kryoserializer.buffer.max | spark.shuffle.sort.bypassMergeThreshold |
| spark.locality.wait | spark.storage.memoryMapThreshold |
| spark.maxRemoteBlockSizeFetchToMem |  |

表 2.1 初步筛选出的配置参数

|  |  |  |
| --- | --- | --- |
| |  | | --- | | spark.memory.offHeap.enabled | | spark.memory.storageFraction |
| spark.memory.offHeap.size | spark.rdd.compress |
| spark.executor.memory | spark.reducer.maxBlocksInFlightPerAddress |
| spark.executor.cores | spark.reducer.maxReqsInFlight |
| spark.executor.memoryOverhead | spark.reducer.maxSizeInFlight |
| spark.executor.instances | spark.scheduler.revive.interval |
| spark.broadcast.blockSize | spark.shuffle.file.buffer |
| spark.broadcast.checksum | spark.shuffle.io.numConnectionsPerPeer |
| spark.broadcast.compress | spark.shuffle.sort.bypassMergeThreshold |
| spark.default.parallelism | spark.storage.memoryMapThreshold |
| spark.kryoserializer.buffer | spark.sql.shuffle.partitions |
| spark.kryoserializer.buffer.max | spark.sql.autoBroadcastJoinThreshold |
| spark.locality.wait | spark.sql.inMemoryColumnarStorage.batchSize |
| spark.maxRemoteBlockSizeFetchToMem | spark.sql.inMemoryColumnarStorage.compressed |
| spark.memory.fraction | spark.sql.crossJoin.enabled |

表 2.2 tpcds的配置参数

#### 2.2.2 模型选择

通过多种算法的使用以及论文调研工作，我们的机器学习算法使用的都是集成树算法**，因为集成树具有很好的抗过拟合性，适用于目前的小数据训练集**。我们发现同一个机器学习算法在不同的程序中的表现是不一样的。在Spark中运行WordCount100g，使用220条样本训练时，达到的误差(MAPE)最小值如下表2.2所示。

|  |  |
| --- | --- |
| 使用的算法 | 误差 |
| LightGBM | 0.183 |
| AdaBoost | 0.264 |
| GBDT | 0.166 |
| XGboost | 0.145 |
| Random Forest | 0.144 |

表2.3 WordCount100g各模型误差

|  |  |
| --- | --- |
| 使用的算法 | 误差 |
| LightGBM | 0.249 (615条数据) |
| AdaBoost | 0.276 (200条数据) |
| GBDT | 0.211 (325条数据） |
| XGboost | 0.203 (200条数据) |
| Random Forest | 0.213 (325条数据) |

表2.4 TeraSort20g各模型误差

Random Forest在 WordCount100G的用例中，表现最好。而在TeraSort20g的测试中表现中等。

由以上案例可知**，如果仅仅依赖一种算法去训练不同的用例，得到的结果并不能达到最优**。所以我们在进行执行时间预测模型的训练时，会同时训练5种模型，**只要其中一个模型的误差达到要求**（因样例不同会发生变化）就可以停止训练样本的收集，该方法能有效降低样本采集时间。

#### 2.2.3 固定或去除某些对模型干扰很大的参数

在做tpcds的优化时(只包含敏感query)，我们发现配置参数 spark.sql.adaptive.enabled对tpcds的运行时间影响很大。如果某次运行存在query执行过长(超过1小时)，那么该次运行必定将spark.sql.adaptive.enabled设置为True，也就是下图的1.0。

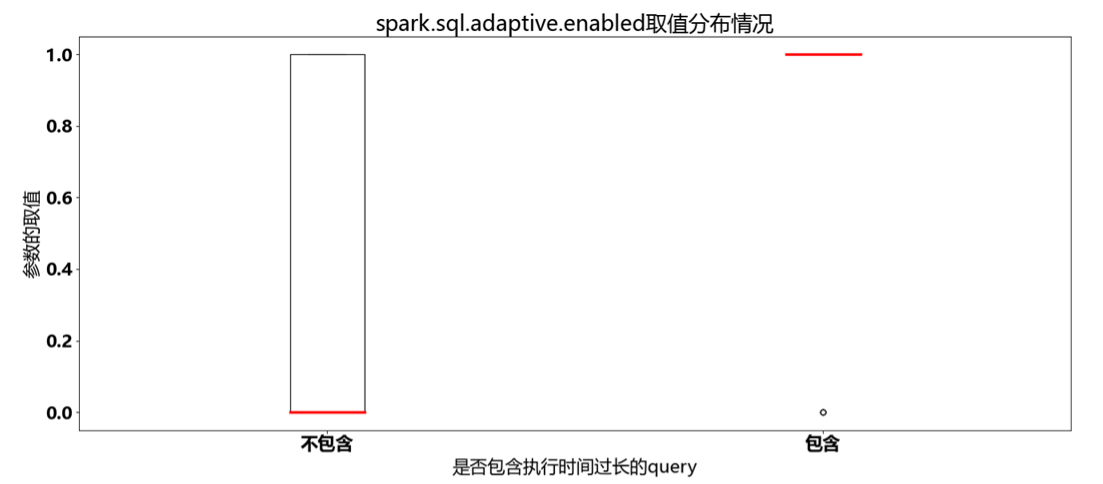


图2.2 spark.sql.adaptive.enabled与运行是否过长的关系

如图2.2所示，如果某次执行存在query执行过长，那么此次执行必定将spark.sql.adaptive.enabled设置为True(对应上图中的1.0)。

而将spark.sql.adaptive.enabled设置为False(对应上图中的0.0)的配置中，全都不存在query执行过长的情况。

以上的例子说明了spark.sql.adaptive.enabled对程序运行存在着巨大的负面影响，极大的降低了程序的运行速度。所以在后续的优化过程中，我们将该配置参数固定成false。

下图为不固定spark.sql.adaptive.enabled时的最优模型训练结果

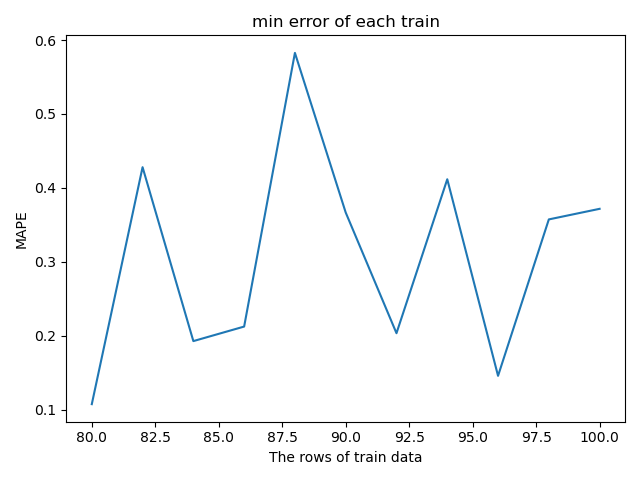


图 2.3 gbdt模型在不同数据量下的训练误差图

图2.3的误差波动很大，在80条数据时达到最低0.1，88条数据时却能达到0.6.

下图为固定spark.sql.adaptive.enabled时的最优模型训练结果

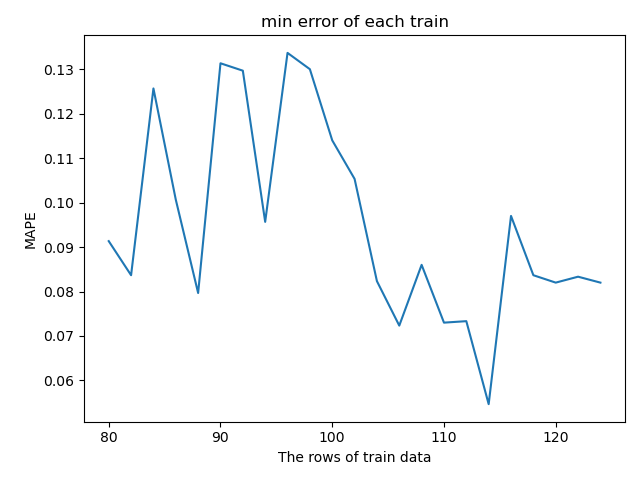


图 2.4gbdt在不同数据量下模型训练误差图\_1

图2.4的误差范围在0.055到0.134之间，与图2.3相比，无论是模型的精度还是训练的稳定性都有了非常大的提升，这说明我们将spark.sql.adaptive.enabled为false的做法是行之有效的。

### 2.3 测试结果

实验环境同1.3。

由于执行时间建模使用的机器学习算法有5种，所以这里每个用例只用最好的训练结果进行展示。

**(1) WordCount**

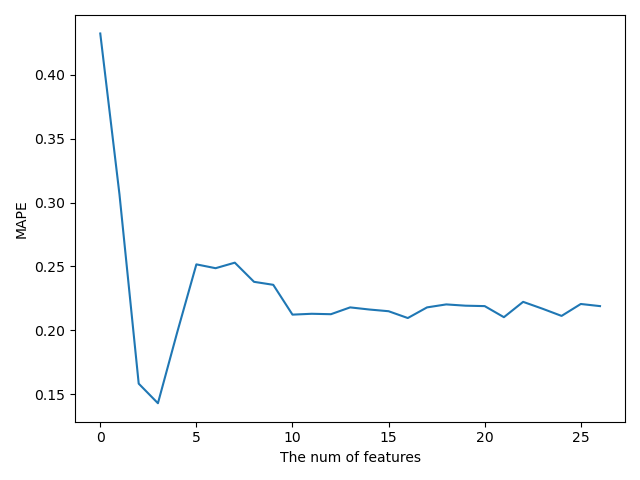


图2.5 RandomForest使用220条训练样本的误差图

RandomForest在使用220条数据时获得最低误差0.144，误差最低点使用的4个配置参数如下。

|  |  |
| --- | --- |
| spark.executor.instances | spark.default.parallelism |
| spark.executor.cores | spark.memory.offHeap.size |

表2.5 WordCount 重要配置参数

**(2) TeraSort**

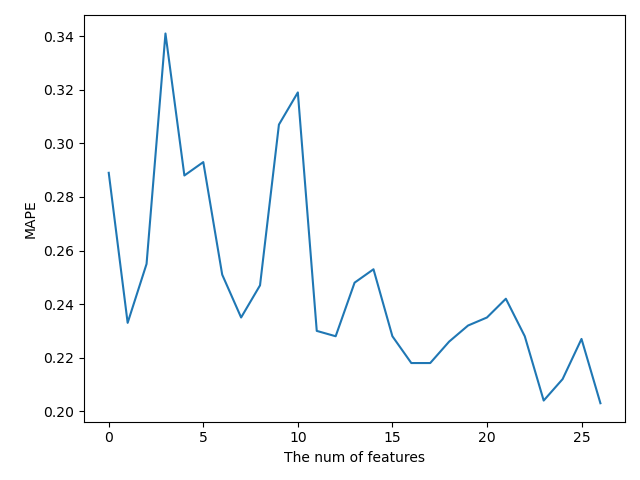


图2.6 XgBoost使用200条训练样本的误差图

在我们使用的5个机器学习算法中，XgBoost在使用了全部配置参数做模型输入时得到的训练误差达到最低(0.202)，GBDT在误差最低点(0.211)时选了19个配置参数，Random Forest在误差最低点（0.213）选了23个配置参数。

在TeraSort的重要配置参数选取中，各机器学习算法在误差最低点选取的配置参数个数偏多(19个及以上)。

而在WordCount的重要配置参数选取中，各机器学习算法选取的配置参数个数全部小于等于10个。很可能是因为TeraSort比wordCount更加复杂，关联的配置参数也更多。

对于TeraSort而言, 执行时间预测模型在训练过程中将spark.executor.memory的重要性排在前3位，这进一步验证了我们在TeraSort的IPC建模部分（1.4.2）得出的结论，即memory对TeraSort的运行影响很大。

**(3) Tpcds**

由2.2.2的分析可知，将spark.sql.adaptive.enabled固定为false可以改善训练效果，所以这里我们的训练是在spark.sql.adaptive.enabled固定为false的基础上进行的。

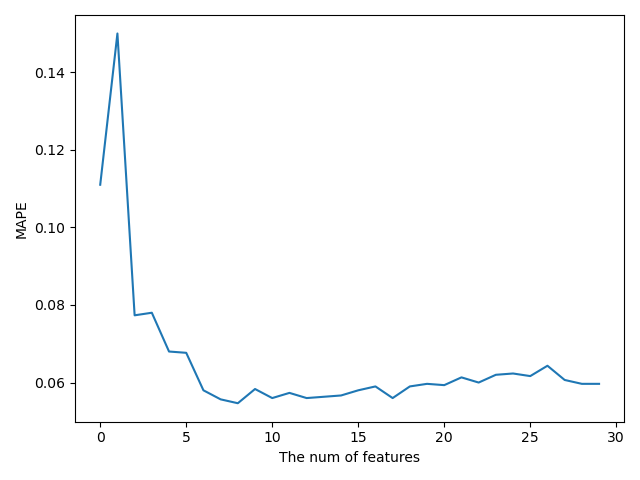


图 2.7 GBDT使用114条训练样本的误差图

在误差最低点( 0.055 )选出的9个配置参数如下表

|  |  |
| --- | --- |
| spark.sql.shuffle.partitions | spark.executor.instances |
| spark.executor.memory | spark.memory.storageFraction |
| spark.shuffle.sort.bypassMergeThreshold | spark.shuffle.file.buffer |
| spark.executor.cores | spark.kryoserializer.buffer |
| spark.reducer.maxSizeInFlight |  |

表2.5 Tpcds 重要配置参数

# 三、微体系结构层、操作系统层事件对比（Yarn VS Kubernetes）

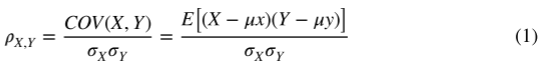
项目基于多种数据分析方法对不同环境下微体系结构层和操作系统层事件进行分析，进行环境差异探究，事件相关性计算等方面的工作。因为Yarn没有使用容器，所以以下只从微体系结构层、操作系统层进行分析。

### 3.1相关工作

#### 3.1.1 相关性分析

（1）Daily Activity Feature Selection in Smart Homes Based on Pearson Correlation Coefficient（发表会议：International Journal of Electrical Power & Energy Systems）

**研究思路**：提出使用皮尔逊相关系数(PCC)进行特征选择来减少日常活动特征的数量，与使用所有特征训练模型相比提高机器学习分类器的精度。皮尔森相关系数用于评估两个变量之间的线性相似度。如公式（1）所示，X和Y的函数COV(X, Y)是X和Y的协方差和是X和Y的标准差。Y的取值范围为[-1,+1]。值为+1表示X与Y完全线性相关,值为0表示X与Y完全不线性相关。值为−1表示X与Y完全负相关。在大多数情况下，当P(X,Y)大于0.8时，X和Y之间表现出极强的相关性。当P(X,Y)大于0.6时，X和Y可以说是强相关的。



**不足**：样本中存在的极端值对Pearson相关系数的影响极大。

**启发**：对特征进行数据清洗后再使用Pearson相关系数分析特征之间的相似性。

#### 3.1.2 相似性分析

目前为止，已调研相似性分析相关论文2篇。本项目所参考的主要文献如下所示：

（1）To Tune or Not to Tune? In Search of Optimal Configurations for Data Analytics（发表会议：ACM SIGKDD）

**研究思路**：论文提出使用贝叶斯优化算法调优spark程序，当调优负载到达一定数量时，使用负载相似分析技术分析新负载与已调优负载的相似性，简化新负载调优过程。在运行时根据指标之间的曼哈顿距离找到与当前工作负载最相似的工作负载。然后将在调整源时获得的知识转移到目标。这包括：1)重要的配置参数；2)源执行样本及其高斯模型。再在原有知识基础上进行贝叶斯优化调优过程。度量相似性的表征指标包括大量运行时的指标，并将它们表示为可比较的比率（GC时间/总运行时间、shuffle数据/总输入数据等），在这些指标的基础上使用曼哈顿距离找到最接近的已调优负载。

**不足**：曼哈顿距离只能用于比较长度相等的数据，如该文章把运行时的指标转为可比较的比率。

**启发**：如果需要比较时间序列数据，找到其中的相似性和差异，曼哈顿距离和欧式距离都不适用。可采用DTW（Dynamic Time Warping）。

（2）Counter Miner: Mining Big Performance Data from Hardware Counters（发表会议：Annual IEEE/ACM International Symposium on Microarchitecture）

**研究思路**：使用PMU tools收集微体系结构层事件，预测IPC从而找出重要事件。收集事件的方式有两种：①OCOE：一次只测量一个程序，尽管能确保准确，但耗时过长，共需要采集200多个事件。 ②MLPX：一次采集多个事件值这样会带来数据波动，难以确保事件值的准确性。为了衡量不同采集方式下事件的误差，从而确定使用哪一种采集方式较适合该项目，作者使用DTW(Dynamic Time Warping，动态时间归整)分析两种方法带来的结果差异。比如取出ICACHE.MISSES事件在OCOE和MLPX下的时间序列，使用DTW算法计算两者之间的距离，观察得出平均误差28.3%。进而分析原因，得出是由于MLPX下的缺失值和异常值的影响。如果考虑这些异常值能够显著改善结果，之后作者选择对MLPX使用数据清理技术来改善。

**不足**： DTW算法的时间复杂度比欧氏距离和曼哈顿距离更高。目前的DTW对象主要集中在静态时间序列, 利用DTW处理实时数据的研究还较少。DTW需要先匹配好两条时间序列的数据点, 进而计算出两者的距离, 然而对于动态变化、不稳定的实时数据, 目前常用的DTW相关方法无法直接使用。

**启发**：结合数据挖掘和数据清洗技术提升DTW的性能。

#### 3.1.3 基于TopDown分析方法的性能瓶颈探究

目前为止，已调研TopDown分析方法相关论文2篇。本项目所参考的主要文献如下所示：

（1）A Top-Down Method for Performance Analysis and Counters Architecture分析方法(发表会议/期刊：IEEE International Symposium on Performance Analysis of Systems and Software）

（2）Extending the roofline model: Bottleneck analysis with microarchitectural constraints（发表会议/期刊：2014 IEEE International Symposium on Workload Characterization）

**研究思路**: 论文提出了一种自顶向下的分析——一种可行的、快速的方法，用于识别无序CPU中的关键瓶颈。论文以分层的方式进行结构化的深入研究，引导用户进入正确的研究区域。将权重分配给树中的节点，以指导用户将其分析工作集中在确实重要的问题上，而忽略无关紧要的问题。例如，假设给定的应用程序受到指令获取问题的严重影响；该方法在树的最上层将其分类为前端绑定。用户/工具应（仅）在层次结构的前端子树由上向下找出瓶颈。向下递归找出瓶颈，直到到达树叶。叶子可以指向工作负载的特定停顿，或者它可以表示具有共同微架构的问题的子集。这些问题子集可能会限制应用程序运行。

**不足**:在某些特定的硬件条件下可能无法正确分类瓶颈。例如在硬件超线程中较难确定问题修复是否会转化为加速，工作负载通常会转移到下一个关键瓶颈。

**启发**:Top-Down分析方法使用多核中指定的PMU事件，采用分层分类，能够直接找出性能瓶颈的问题，有效分析无序CPU中的关键性能瓶颈。

#### 3.1.4 基于主成分分析的降低事件维度方法探究

目前为止，已调研基于主成分分析的降低数据复杂度相关论文3篇。本项目所参考的主要文献如下所示：

（1）Application-Specific Feature Selection and Clustering Approach with HPC System Profiling Data（发表刊物：The Journal of Supercomputing, 2021, 1-15）

（2）A Cluster-Principal-Component-Analysis-Based Indoor Positioning Algorithm（发表刊物：IEEE Internet Things, 2021, 187-196）

（3）A Presentation and Retrieval Hash Scheme of Images Based On Principal Component Analysis(发表刊物：Vis. Comput., 2021, 2113-2126)

**研究思路**：针对多核环境下处理器发生的性能事件有大量的指标，很难分析它们之间的相关性的特点，论文提出了一种基于高性能计算系统分析数据的应用特定特征选择和聚类方法。所提出的方法执行基于主成分分析的特征提取，以获得高效的性能分析方法，主成分分析是一种多变量数据处理方法，它计算协方差矩阵和特征向量，将数据降维到低维，同时保持由多维特征向量组成的数据的信息方向。论文中使用主成分分析进行特征提取，以构建以数据为中心的剖析数据。通过该算法，原始数据的276个事件减少到74个。通过这种方式，解决了数据收集过程中的成本降低、计算过程中的复杂性和数据稀疏问题

**不足**：论文有效地进行了高性能计算中特定应用的特征选择和性能分析，但受限于单个节点的实验存在局限性。

**启发**：在容器中大数据业务特征分析及建模技术项目中，同样存在多变量数据分析场景，在这种场景下，可考虑使用主成分分析方法来降低数据复杂度，提高数据分析效率。

#### 3.1.5 聚类分析

（1）Discriminative subspace matrix factorization for multiview data clustering

(发表期刊：Pattern Recognition 2021)

**研究思路**：本论文提出了一种基于非监督多视图NMF的框架，称为判别多视图子空间矩阵分解（DMSMF），用于聚类。简而言之，原始数据集中不同视图被视为多个矩阵。把每个视图看作一个连续的矩阵和几个特定于视图的矩阵。对于具体的矩阵，我们将建议的统一框架扩展到结合非负矩阵分解和线性判别分析的多视图版本。对于视图特定的矩阵，通过流形学习设计了两个正则化项。这些约束要求他们从内部结构中学习并避免异常值。最后，将两个处理结果合并。

**不足**：对于不同数据集的敏感程度，需要手动初始化参数，对聚类结果会产生影响

**启发**：使用多视点线性判别分析与多视点非负矩阵分解相结合的判别子空间聚类的方法，可以用于将数量众多的事件进行聚类，从而在有针对性的按照类别进行研究。

#### 3.1.6 基于平均值比较的差异事件快速定位方法

该方法用于简单快速的找出那些在Yarn环境和Kubernetes环境存在差异的事件，DTW计算能反应同一事件在不同环境的差异，但并不能反应出大小关系。例如，对于操作系统事件net/total – send，我们通过DTW距离发现该事件在Yarn环境和Kubernetes环境下存在差异，但并不能知道该事件在哪种环境下取值更大。因此提出此方法来解决上述问题。

该方法的思路如下：

1.使用多种配置(随机生成运行程序的配置参数)，多种数据规模，分别在Yarn和Kubernetes环境下运行各种用例，并采集微体系结构和操作系统事件。将Yarn和Kubernetes运行环境下，使用同一配置和同一数据规模的的同一事件(记为A)进行比较。

例如WordCount50G使用第一组配置分别在Yarn和Kubernetes环境下运行，得到两组事件A的数据，将这两组数据进行比较。

2.比较方式为取Yarn环境下事件A的平均值(记为mean\_yarn),取Kubernetes环境下事件A的平均值(记为mean\_k8s)。如果在大多数（微体系结构事件要求90%，操作系统事件要求100%）的运行结果中，都有mean\_k8s大于mean\_yarn，或者mean\_k8s小于mean\_yarn，那么可以认定该事件在Yarn和Kubernetes环境下表现出了不同的特性。

为了尽可能减小偶然因素的影响，该方法需要较多的运行结果来进行比较。

### 3.2 测试结果

#### 3.2.1 相关性分析测试

1.4.2中交互强度的方法可以反映事件之间的相关性但无法反映是该相关性是正相关还是负相关，此处的相关性分析可以当做交互强度的一个补充。

数据源: Kubernetes环境下WordCount和TeraSort使用各自最优配置运行，并采集事件

计算方法: 使用1.4.2中计算交互强度所用的重要事件进行Pearson相关系数 的计算。

**（1）WordCount**

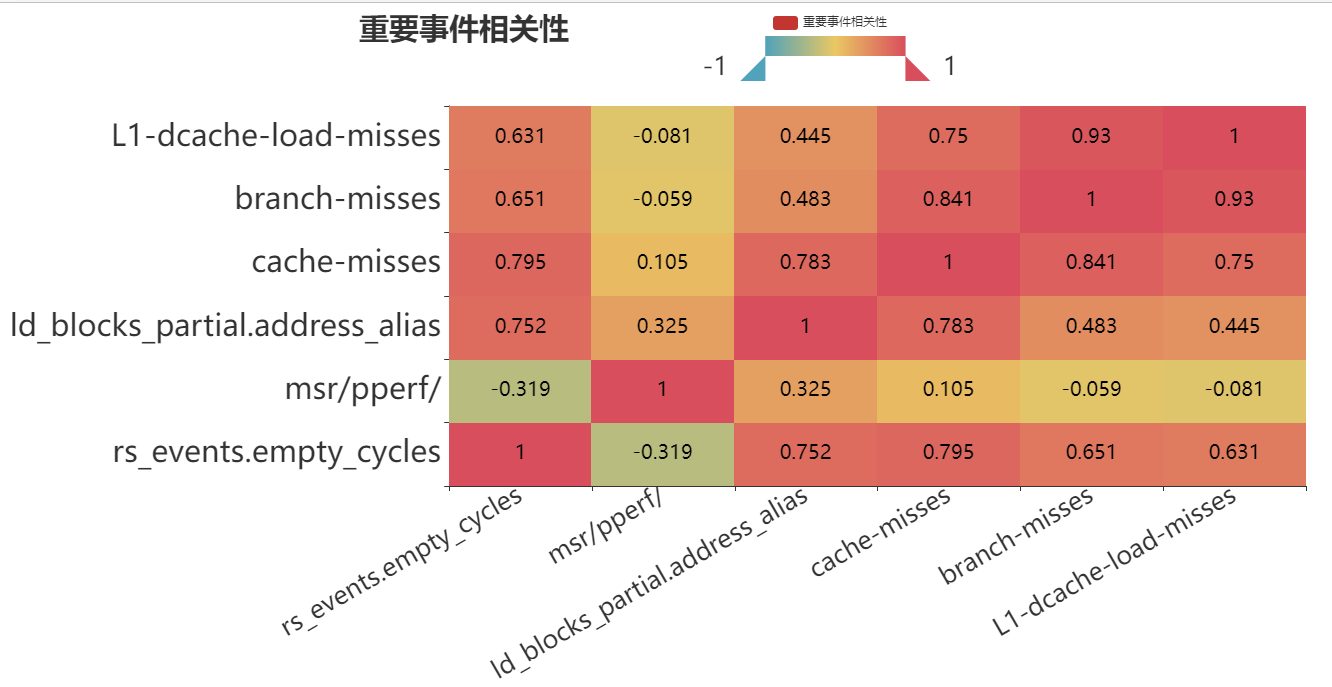


图3.1 WordCount重要事件Pearson相关系数

除了msr/pperf以外，cache-misses和其他事件的相关系数都在0.75及以上，而且在1.4.2WordCount交互强度测试中最高的五个交互对都和cache-misses有关，进一步说明了cache-misses关联的事件很多。

**（2）Terasort**

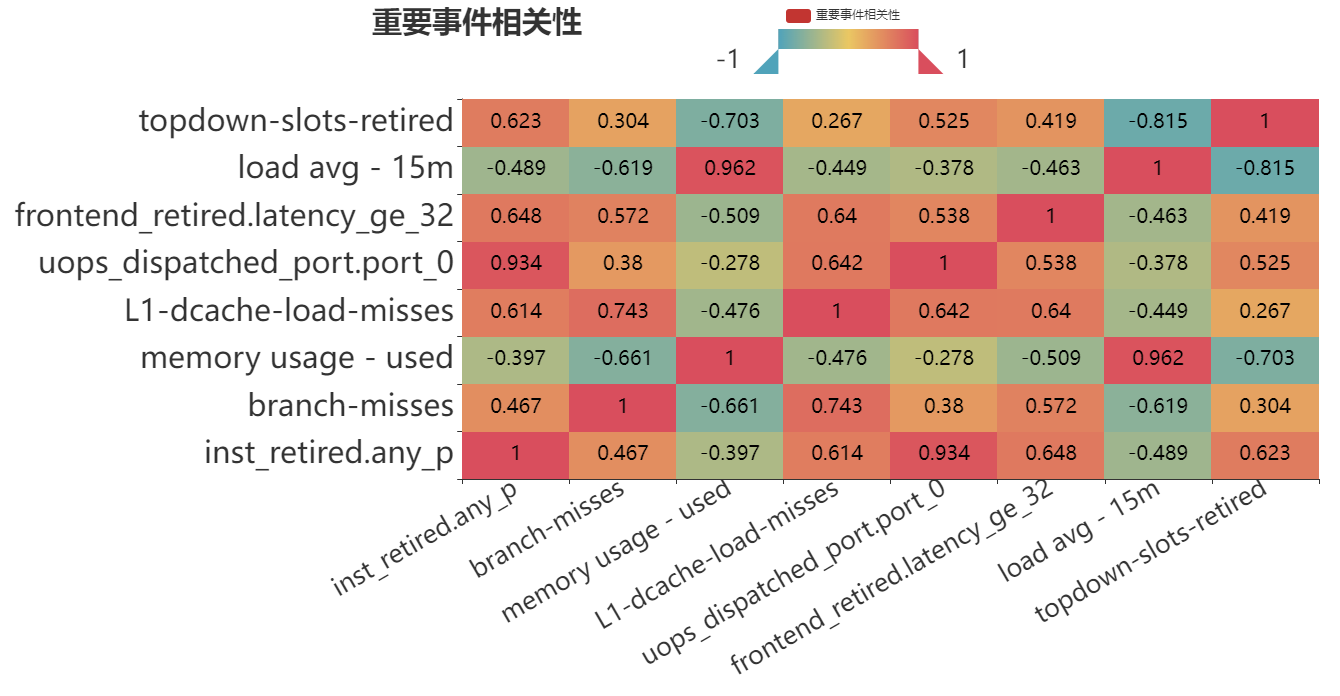


图3.2 TeraSort重要事件Pearson相关系数

两个操作系统事件load avg - 15m和memory usage – used之间表现出很强的相关性，相关系数达到了0.962。而这两个操作系统事件和其他微体系结构事件之间的相关性全部都是负数。

**（3）不同用例比较**

同样的事件在WordCount和TeraSort中表现出的相关性是有少量差异的，例如L1-dcache-load-misses和branch-misses在WordCount用例中表现出的相关性0.93要大于在TeraSort中表现出的0.743。

#### 3.2.2 相似性分析测试

基于Dtw(Dynamic Time Warping)的相似性分析，计算同一事件在Yarn和Kubernetes环境下的dtw距离。

**（1）WordCount**

数据源: 随机生成50组配置，并在Yarn和Kubernetes环境下分别运行WordCount100g，采集微体系结构和操作系统事件。

计算方法:

1. 在计算事件A的 dtw距离时,首先找出同一组配置分别在Yarn和Kubernetes环境运行时采集到的事件A的时间序列

2. 计算1.中两个时间序列的dtw距离

3. 将1.和2.重复50次，并将这50份距离累加起来作为事件A的dtw距离

4. 对其他事件执行以上操作，得出所有事件的dtw距离。

**a.微体系结构事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| uncore\_iio\_free\_running\_0/bw\_in\_port1/ |  | Kernel PMU event |
| power/energy-pkg | Counts the number of times a class of instructions that may cause a transactional abort was executed. Since this is the count of execution, it may not always cause a transactional abort | Kernel PMU event |
| tx\_exec.misc1 |  | memory |
| major-faults', |  | Hardware event |
| dtlb\_store\_misses.walk\_completed\_1g | Page walk completed due to a demand data store to a 1G page |  |

表3.1 WordCount测试中dtw距离最小的5个微体系结构事件

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| topdown-total-slots |  | Kernel PMU event |
| cpu\_clk\_unhalted.thread\_any | Core cycles when at least one thread on the physical core is not in halt state | pipeline |
| cpu\_clk\_unhalted.thread\_p\_any | Core cycles when at least one thread on the physical core is not in halt state | pipeline |
| uops\_retired.total\_cycles |  |  |
| inst\_retired.total\_cycles\_ps |  |  |

表3.2 WordCount测试中dtw距离最大的5个微体系结构事件

**b.操作系统事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| dbus - ses |  | dbus连接的数量(需要python-dbus) |
| sysv ipc - shm | shm共享内存使用 | 展示IPC（进程间通信）状态 |
| udp - lis | lis监听的数量 | 开启udp统计 (listen, active) |
| snooze - snooze |  | 显示每秒运算次数 |
| file locks - rea |  | 开启文件所统计，包括 (posix, flock, read, write) |

表3.3 WordCount测试中dtw距离最小的5个操作系统事件

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| memory usage - cach | cache | 内存使用相关 |
| memory usage - used', | used | 内存使用相关 |
| memory usage - free' | free | 内存使用相关 |
| net/total - send | send：网络发包带宽 | 展示总体网络收发状态 |
| dsk/total - read | read：磁盘读带宽 | 展示磁盘设备读写总计 |

表3.4 WordCount测试中dtw距离最大的5个操作系统事件

**（2）Terasort**

数据源:随机生成50组配置，并在Yarn和Kubernetes环境下分别运行TeraSort20g，采集微体系结构和操作系统事件。

**a.微体系结构事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| uncore\_iio\_free\_running\_0/bw\_in\_port1 |  | Kernel PMU event |
| tx\_exec.misc1 |  | memory |
| uncore\_iio\_free\_running\_3/bw\_in\_port0/ |  | Kernel PMU event |
| uncore\_iio\_free\_running\_3/bw\_out\_port0/ |  | Kernel PMU event |
| major-faults |  | Hardware event |

表3.5 Terasort测试中dtw距离最小的5个微体系结构事件

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| offcore\_requests\_outstanding  .demand\_data\_rd | Offcore outstanding Demand Data Read transactions in uncore queue | Kernel PMU event |
| offcore\_requests\_outstanding  .all\_data\_rd | Offcore outstanding cacheable Core Data Read transactions in SuperQueue (SQ), queue to uncore | memory |
| topdown-total-slots |  | Kernel PMU event |
| l1d\_pend\_miss.pending | L1D miss outstandings duration in cycles | cache |
| l1d\_pend\_miss.pending\_cycles\_any | Cycles with L1D load Misses outstanding from any thread on physical core | cache |

表3.6 Terasort测试中dtw距离最大的5个微体系结构事件

**b.操作系统事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| dbus - ses |  | dbus连接的数量(需要python-dbus) |
| udp - lis | lis监听的数量 | 开启udp统计 (listen, active) |
| snooze - snooze |  | 显示每秒运算次数 |
| sockets - udp | udp套接字数量 | 展示套接字状态 |
| file locks - lck |  |  |

表3.7 Terasort测试中dtw距离最小的5个操作系统事件

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| memory usage - cach | cache | 内存使用相关 |
| memory usage - used | used | 内存使用相关 |
| memory usage - free | free | 内存使用相关 |
| net/total - send | send：网络发包带宽 | 展示总体网络收发状态 |
| net/total - recv | recv：网络收包带宽 | 展示总体网络收发状态 |

表3.8 Terasort测试中dtw距离最大的5个操作系统事件

**（3）不同用例的比较**

将(1)和(2)中同类事件取交集

**a.微体系结构事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| uncore\_iio\_free\_running\_0/bw\_in\_port1/ |  | Kernel PMU event |
| tx\_exec.misc1 |  | memory |
| major-faults |  | Hardware event |

表3.9 WordCount和Terasort测试中dtw距离最小的5个微体系结构事件的交集

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| topdown-total-slots |  | Kernel PMU event |

表3.10 WordCount和Terasort测试中dtw距离最大的5个微体系结构事件的交集

**b.操作系统事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| dbus - ses |  | dbus连接的数量(需要python-dbus) |
| udp - lis | lis监听的数量 | 开启udp统计 (listen, active) |
| snooze - snooze |  | 显示每秒运算次数 |

表3.11 WordCount和Terasort测试中dtw距离最小的5个操作系统事件的交集

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| memory usage - cach | cache | 内存使用相关 |
| memory usage - used | used | 内存使用相关 |
| memory usage - free | free | 内存使用相关 |
| net/total - send | send：网络发包带宽 | 展示总体网络收发状态 |

表3.12 WordCount和Terasort测试中dtw距离最大的5个操作系统事件的交集

从dtw距离最大的事件中可以看出，微体系结构事件topdown-total-slots以及操作系统事件中和内存，网络相关的事件在Kubernets和Yarn环境下表现出一定的差异。

#### 3.2.3 TopDown分析测试

在后续工作中进行。

#### 3.2.4 主成分分析测试

在后续工作中进行。

#### 3.2.5 聚类分析

在后续工作中进行

#### 3.2.6 平均值比较方法

##### 3.2.6.1 mean\_k8s大于mean\_yarn事件举例

**（1）WordCount**

数据源: WordCount50g, WordCount75g ,WordCount100g各自随机生成50组配置，分别在Yarn和Kubernetes环境下运行，运行时采集微体系结构事件和操作系统事件。共3\*50\*2=300份数据, 3\*50=150份比较结果。

**a.微体系结构事件**

90%及以上的比较结果都有mean\_k8s大于mean\_yarn的事件包括

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| ild\_stall.lcp | Stalls caused by changing prefix length of the instruction | pipeline |
| uops\_dispatched\_port.port\_7 | Cycles per thread when uops are executed in port 7 | pipeline |
| uops\_dispatched\_port.port\_4 | port 4 | pipeline |
| uops\_dispatched\_port.port\_5 | port 5 | pipeline |
| idq.all\_dsb\_cycles\_4\_uops | Cycles Decode Stream Buffer (DSB) is delivering 4 Uops | frontend |
| uops\_dispatched\_port.port\_1 | port 1 | pipeline |
| uops\_dispatched\_port.port\_2 | port 2 | pipeline |
| uops\_dispatched\_port.port\_3 | port 3 | pipeline |
| idq.dsb\_cycles | Cycles when uops are being delivered to Instruction Decode Queue (IDQ) from Decode Stream Buffer (DSB) path | frontend |
| idq.dsb\_uops | Uops delivered to Instruction Decode Queue (IDQ) from the Decode Stream Buffer (DSB) path | frontend |
| exe\_activity.3\_ports\_util | Cycles total of 3 uops are executed on all ports and Reservation Station was not empty | pipeline |
| mem\_inst\_retired.split\_stores | Retired store instructions that split across a cacheline boundary Supports address when precise (Precise event) | cache |
| dsb2mite\_switches.penalty\_cycles | Decode Stream Buffer (DSB)-to-MITE switch true penalty cycles | fronted |
| uops\_issued.vector\_width\_mismatch | Uops inserted at issue-stage in order to preserve upper bits of vector registers | pipeline |
| core\_snoop\_response.rsp\_ifwdfe | null | other |
| uops\_dispatched\_port.port\_0 | Cycles per thread when uops are executed in port 0 | pipeline |
| idq.all\_dsb\_cycles\_any\_uops | Cycles Decode Stream Buffer (DSB) is delivering any Uop | fronted |

表3.13 WordCount测试中90%的比较结果是mean\_k8s大于mean\_yarn的微体系结构事件

在以上的17个事件中，有10个属于pipeline，5个属于fronted，Cache和other各一个。

**b.操作系统事件**

在150份比较结果中全部比较结果都有mean\_k8s大于mean\_yarn的事件包括。

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| procs - total | 展示进程数量，侧面反映系统负载 | 进程数量 |
| net/total - send | 网络发包带宽 | 总体网络收发状态 |
| unix sockets - str | 流 | 开启unix统计(datagram, stream, listen, active)UNIX Domain SOCKET 是在Socket架构上发展起来的用于同一台主机的进程间通讯（IPC） |
| unix sockets - lis | lis监听的数量 | unix统计 |
| unix sockets - act | act(established)活动的确立的数量 | unix统计 |

表3.14 WordCount测试中全部比较结果都有mean\_k8s大于mean\_yarn的操作系统事件

**（2）TeraSort**

数据源: TeraSort20G随机生成80组配置，分别在Yarn和Kubernetes环境下运行，运行时采集微体系结构事件和操作系统事件。

共80\*2=160份数据, 80份比较结果。

**a.微体系结构事件**

在90%及以上的比较结果都有mean\_k8s大于mean\_yarn的事件包括

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| topdown-recovery-bubbles |  | Kernel PMU event |
| tlb\_flush.dtlb\_thread | DTLB flush attempts of the thread-specific entries | virtual memory |
| ild\_stall.lcp | Stalls caused by changing prefix length of the instruction | pipeline |
| dtlb\_load\_misses.walk\_completed\_1g | Page walk completed due to a demand data load to a 1G page | virtual memory |
| topdown-fetch-bubbles |  | Kernel PMU event |
| tx\_exec.misc1 | Counts the number of times a class of instructions that may cause a transactional abort was executed. Since this is the count of execution, it may not always cause a transactional abort | memory |
| topdown-slots-retired |  | Kernel PMU event |
| offcore\_requests.demand\_code\_rd | Cacheable and noncachaeble code read requests | cache |
| memory\_disambiguation.history\_reset |  | other |
| frontend\_retired.latency\_ge\_32 | Retired instructions that are fetched after an interval where the front-end delivered no uops for a period of 32 cycles which was not interrupted by a back-end stall (Precise event) | frontend |
| core\_snoop\_response.rsp\_shitfse |  | other |
| instructions |  | Hardware event |
| frontend\_retired.latency\_ge\_8 | Retired instructions that are fetched after an interval where the front-end delivered no uops for a period of 8 cycles which was not interrupted by a back-end stall (Precise event) | frontend |
| frontend\_retired.latency\_ge\_16 | Retired instructions that are fetched after an interval where the front-end delivered no uops for a period of 16 cycles which was not interrupted by a back-end stall (Precise event) | frontend |
| frontend\_retired.latency\_ge\_4 | Retired instructions that are fetched after an interval where the front-end delivered no uops for a period of 4 cycles which was not interrupted by a back-end stall (Precise event) | frontend |
| baclears.any | Counts the total number when the front end is resteered, mainly when the BPU cannot provide a correct prediction and this is corrected by other branch handling mechanisms at the front end | pipeline |
| itlb\_misses.miss\_causes\_a\_walk | Misses at all ITLB levels that cause page walks | virtual memory |
| dtlb\_store\_misses.walk\_completed\_2m\_4m | Page walk completed due to a demand data store to a 2M/4M page | virtual memory |
| frontend\_retired.latency\_ge\_64 | Retired instructions that are fetched after an interval where the front-end delivered no uops for a period of 64 cycles which was not interrupted by a back-end stall (Precise event) | frontend |
| frontend\_retired.latency\_ge\_128 | Retired instructions that are fetched after an interval where the front-end delivered no uops for a period of 128 cycles which was not interrupted by a back-end stall (Precise event) | frontend |
| itlb\_misses.walk\_completed | Code miss in all TLB levels causes a page walk that completes. (All page sizes) | virtual memory |
| icache\_64b.iftag\_miss | Instruction fetch tag lookups that miss in the instruction cache (L1I). Counts at 64-byte cache-line granularity | frontend |
| frontend\_retired.l2\_miss |  | frontend |
| frontend\_retired.l1i\_miss |  | frontend |

表3.15 TeraSort测试中90%的比较结果是mean\_k8s大于mean\_yarn的微体系结构事件

**b.操作系统事件**

在80份比较结果，全部比较结果都有mean\_k8s大于mean\_yarn的事件包括

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| memory usage - used: | used | 内存使用相关 |
| net/total - recv | recv：网络收包带宽 | 展示总体网络收发状态 |
| net/total - send | send：网络发包带宽 | 展示总体网络收发状态 |
| procs - total | 进程总数 | 展示进程数量，侧面反映系统负载 |
| sdd - util |  | 显示某一时间磁盘的忙碌状况 |
| unix sockets - act | act(established)活动的确立的数量 | 开启unix统计 |
| unix sockets - lis | lis监听的数量 | 开启unix统计 |
| unix sockets - str | 流 | 开启unix统计 |

表3.16 TeraSort测试中全部比较结果都有mean\_k8s大于mean\_yarn的操作系统事件

**（3）不同用例比较**

将(1)和(2)中WordCount和TeraSort列出的微体系结构事件和操作系统事件取交集。

**a.微体系结构事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| ild\_stall.lcp | Stalls caused by changing prefix length of the instruction | pipeline |

表3.17 90%的比较结果中存在mean\_k8s大于mean\_yarn的微体系结构事件

下图是ild\_stall.lcp 在Wordcount100g下的50组比较结果。

横坐标表示当前是第几组配置的比较结果。比较方式为 (mean\_k8s-mean\_yarn) / mean\_yarn。纵坐标为1.4表示mean\_k8s是mean\_yarn的2.4倍(1+1.4)。

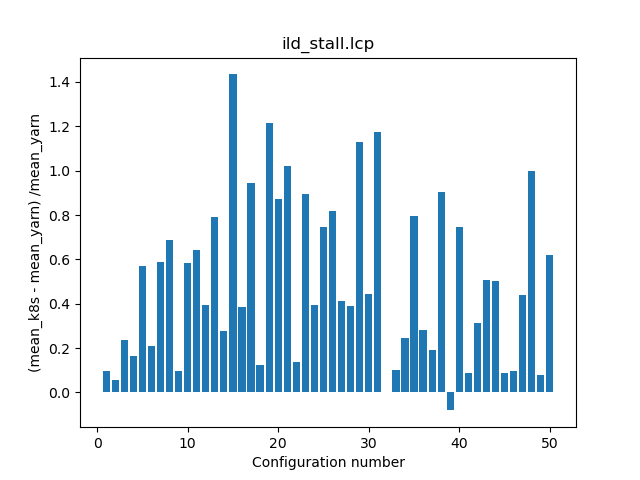
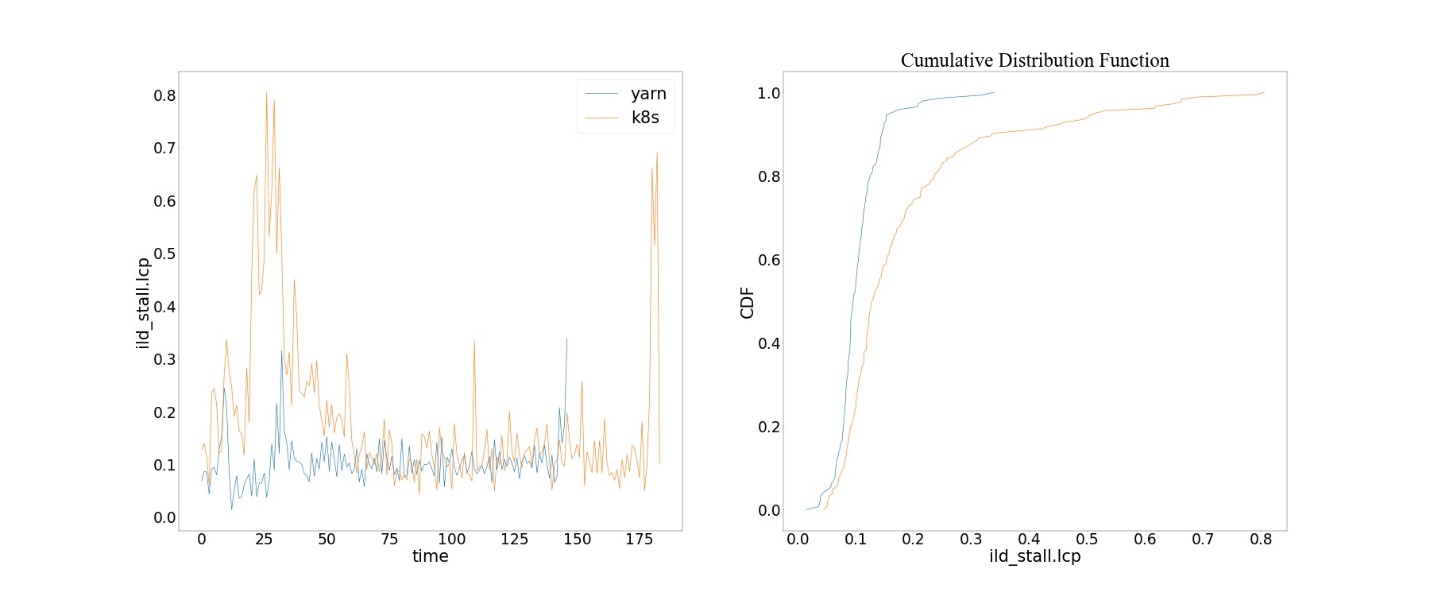


图3.3 WordCount测试下ild\_stall.lcp平均值在不同环境的比较结果

使用第50组配置运行结果举例

图3.4 WordCount测试第50组配置ild\_stall.lcp数据

下图是TeraSort20G下的80组比较结果.

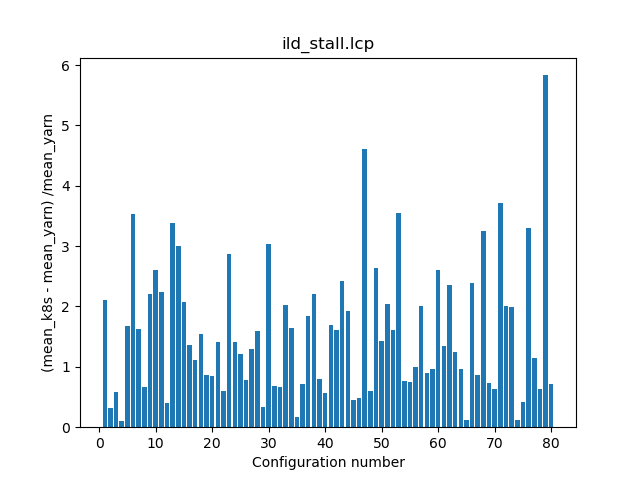


图3.5 TeraSort测试下ild\_stall.lcp平均值在不同环境的比较结果

使用第80组配置运行结果举例

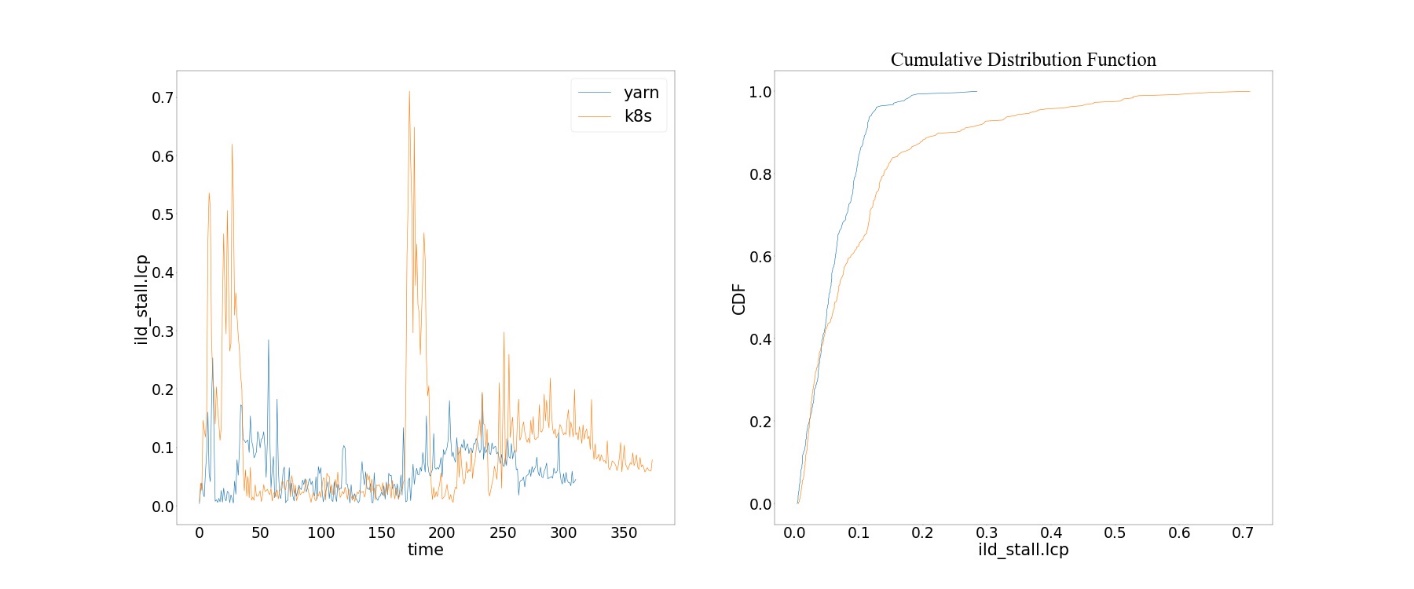


图3.6 TeraSort测试第80组配置ild\_stall.lcp数据

对于ild\_stall.lcp，在图3.3和图3.5的比较结果中，除了图3.3的第39组配置外，其余情况都表现出mean\_k8s大于mean\_yarn。而且在tearSort测试中，上述差异更加明显。

**b.操作系统事件**

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| procs - total | 展示进程数量，侧面反映系统负载 | 进程数量 |
| net/total - send | 网络发包带宽 | 总体网络收发状态 |
| unix sockets - str | 流 | 开启unix统计(datagram, stream, listen, active)UNIX Domain SOCKET 是在Socket架构上发展起来的用于同一台主机的进程间通讯（IPC） |
| unix sockets - lis | lis监听的数量 | unix统计 |
| unix sockets - act | act(established)活动的确立的数量 | unix统计 |

表3.18 全部比较结果都有mean\_k8s大于mean\_yarn的操作系统事件

以 net/total – send 作为示例。下图是Wordcount100g的50组比较结果。

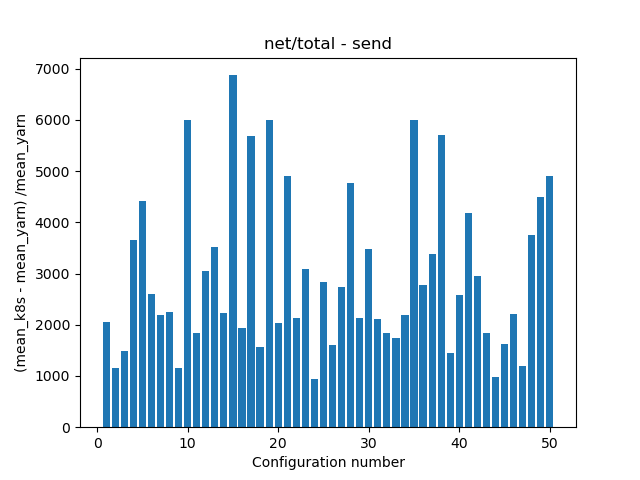


图3.7 WordCount测试下net/total – send平均值在不同环境的比较结果

使用第50组配置运行结果举例

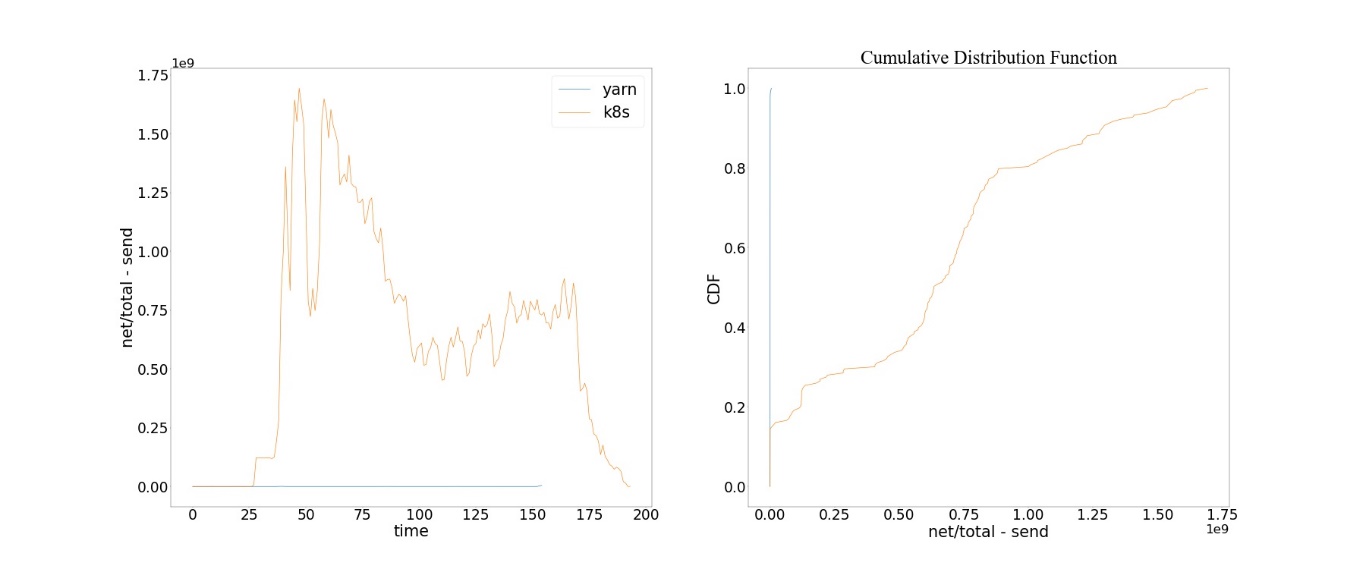


图3.8 WordCount测试第50组配置net/total – send数据

下图是TeraSort20G的80组比较结果。

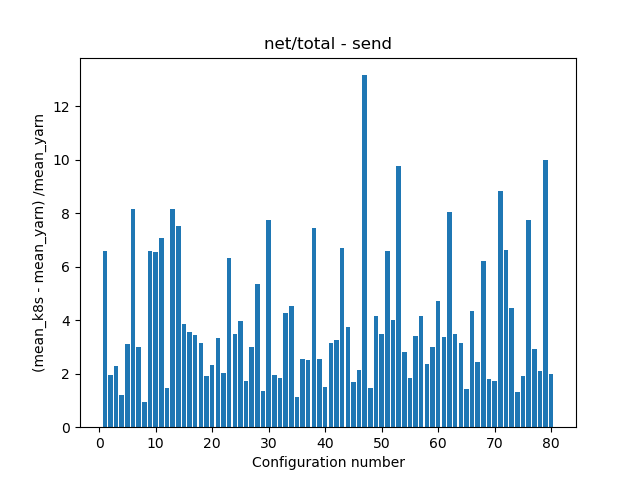


图3.9 TeraSort测试下net/total – send平均值在不同环境的比较结果

使用第80组配置运行结果举例

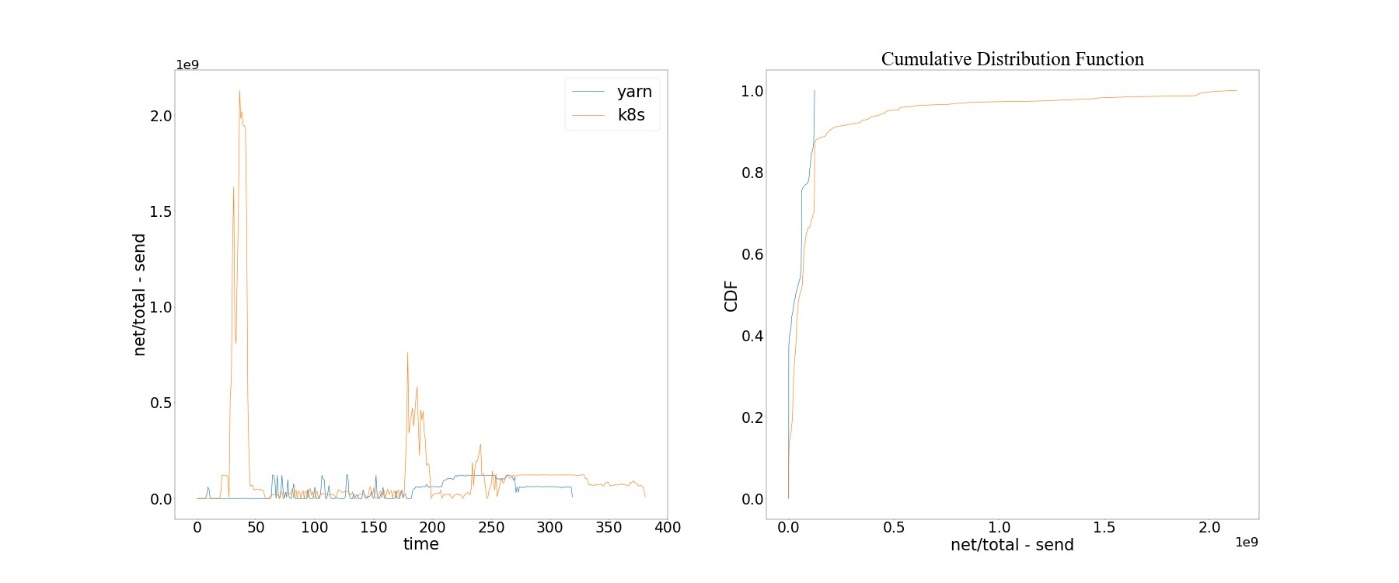


图3.10 TeraSort测试第80组配置net/total – send数据

基于上图3.7和3.9的结果，对于net/total – send事件，WordCount测试和TeraSort测试都有mean\_k8s大于mean\_yarn的现象出现。

在wordCount测试中，mean\_k8s的值通常达到mean\_yarn的**千倍以上，这说明环境不同给net/total – send带来的差异尤为明显**。而在TeraSort测试中，**这一差异通常不到10倍。说明net/total – send事件不仅受运行环境的影响大，而且受测试用例的影响也大**。

并且在3.2.2相似性分析测试中我们通过**dtw距离也发现net/total – send**在Yarn和Kubernetes环境下存在差异。

##### 3.2.6.2 mean\_k8s小于mean\_yarn事件举例

上述3.2.6.1的内容展示了那些在90%及以上的对比结果中都有mean\_k8s大于mean\_yarn的事件。

而对于在90%及以上的对比结果中都有mean\_k8s小于mean\_yarn的事件，限于篇幅原因，我们在此直接给出这些事件在WordCount测试和TeraSort测试中的交集，并做简单示例。

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| machine\_clears.memory\_ordering | Counts the number of machine clears due to memory order conflicts Spec update: SKL089 | memory |

表3.19 90%的比较结果中存在mean\_k8s小于mean\_yarn的微体系结构事件

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| memory usage - cach | 展示进程数量，侧面反映系统负载 | 进程数量 |
| tcp sockets - lis | 网络发包带宽 | 总体网络收发状态 |

表3.20 全部比较结果都有mean\_k8s小于mean\_yarn的操作系统事件

下图是machine\_clears.memory\_ordering在Wordcount100g下的50组比较结果。比较方式为 (mean\_k8s-mean\_yarn) / mean\_yarn。纵坐标为-0.8表示mean\_k8s是mean\_yarn的0.2倍( 1-0.8 )。

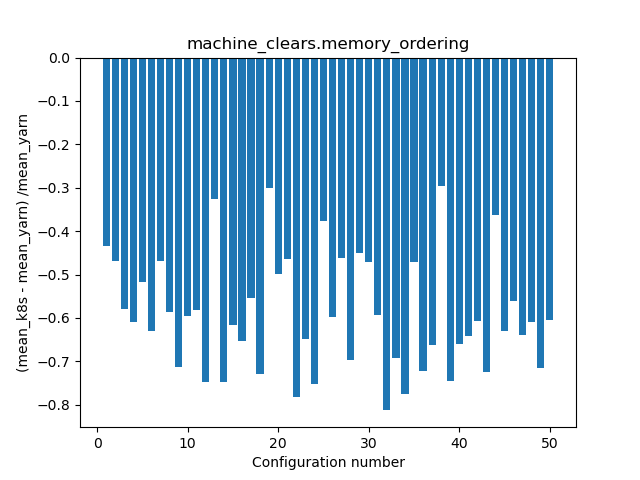


图3.11 WordCount测试下machine\_clears.memory\_ordering平均值在不同环境的比较结果

使用第50组配置运行结果举例

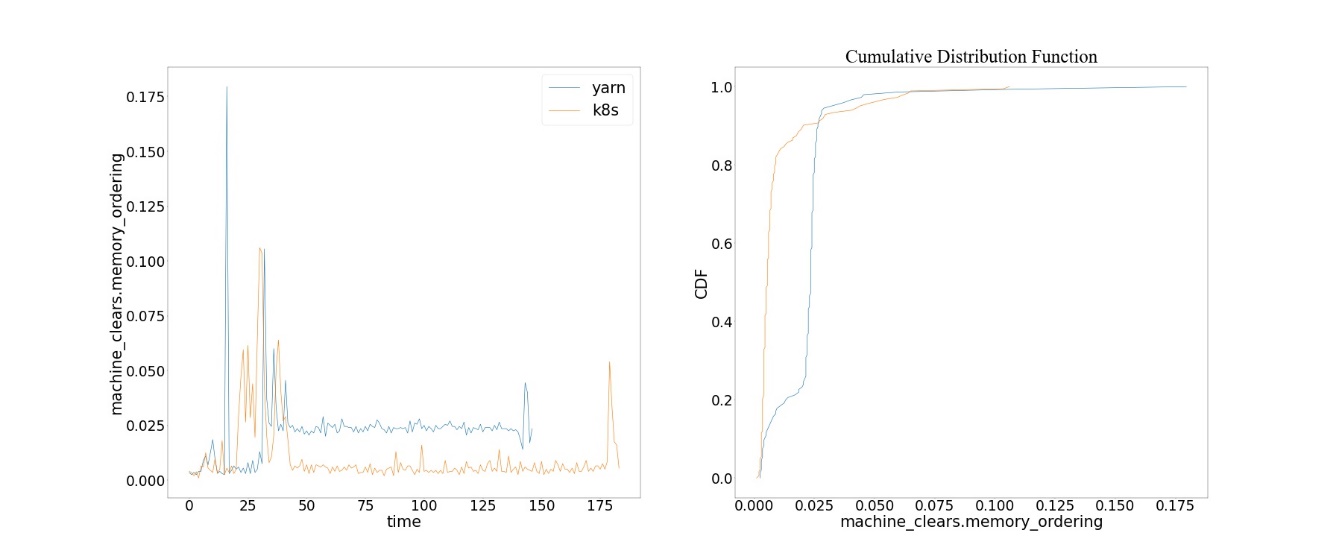


图3.12 WordCount测试第50组配置machine\_clears.memory\_ordering数据

下图是machine\_clears.memory\_ordering在teraSort20g下的80组比较结果。

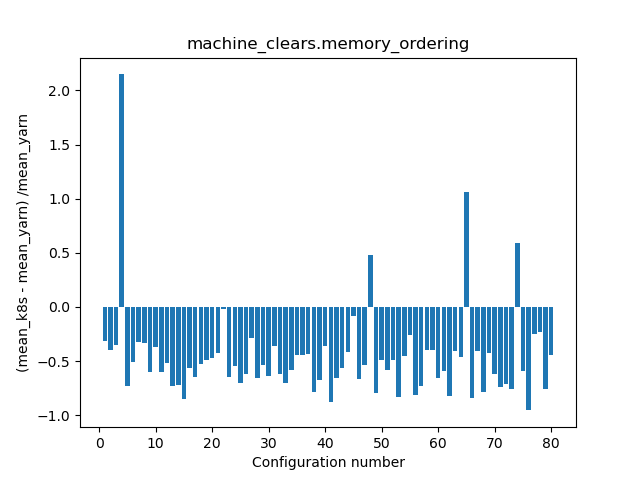


图3.13 TeraSort测试下machine\_clears.memory\_ordering平均值在不同环境的比较结果

使用第80组配置运行结果举例

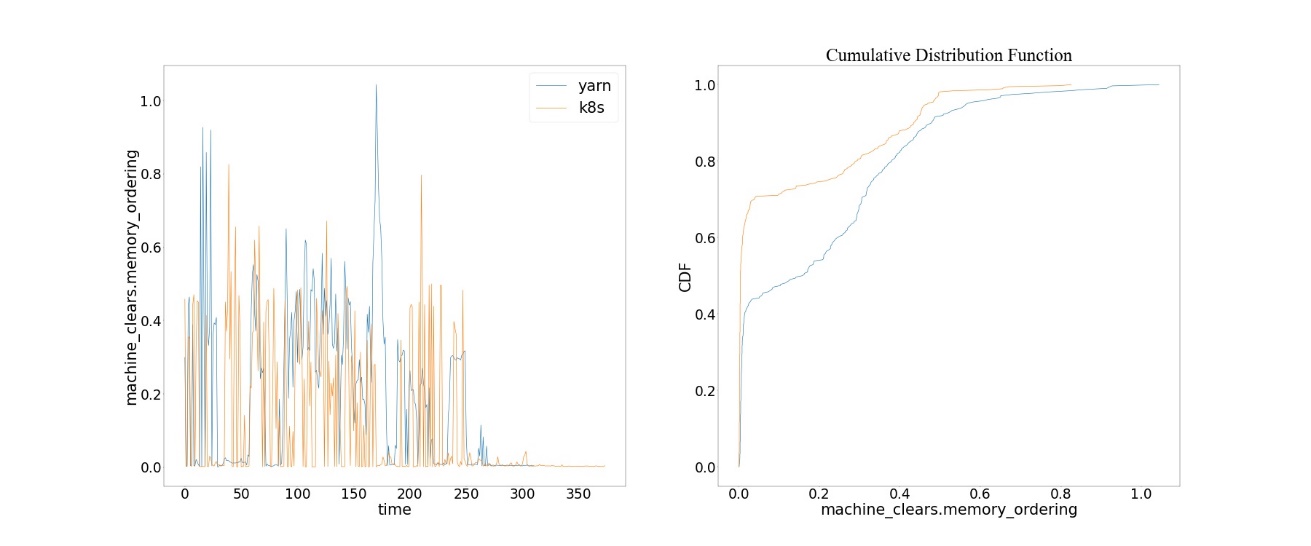


图3.14 TeraSort测试第80组配置machine\_clears.memory\_ordering数据

下图是memory usage - cach在Wordcount100g下的50组比较结果。

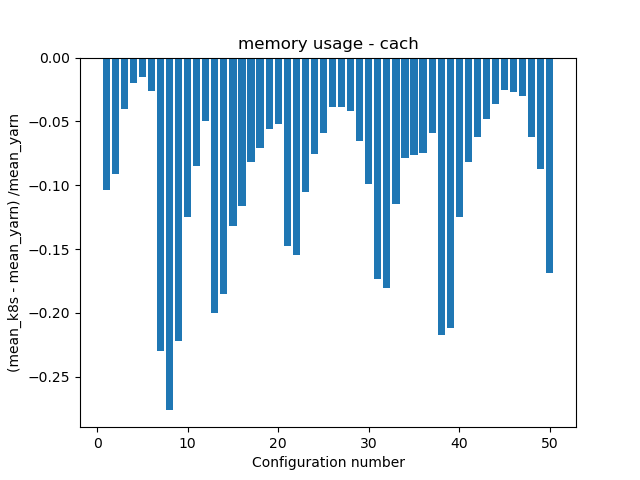


图3.15 Wordcount测试下memory usage - cach平均值的在不同环境的比较结果

使用第50组配置运行结果举例

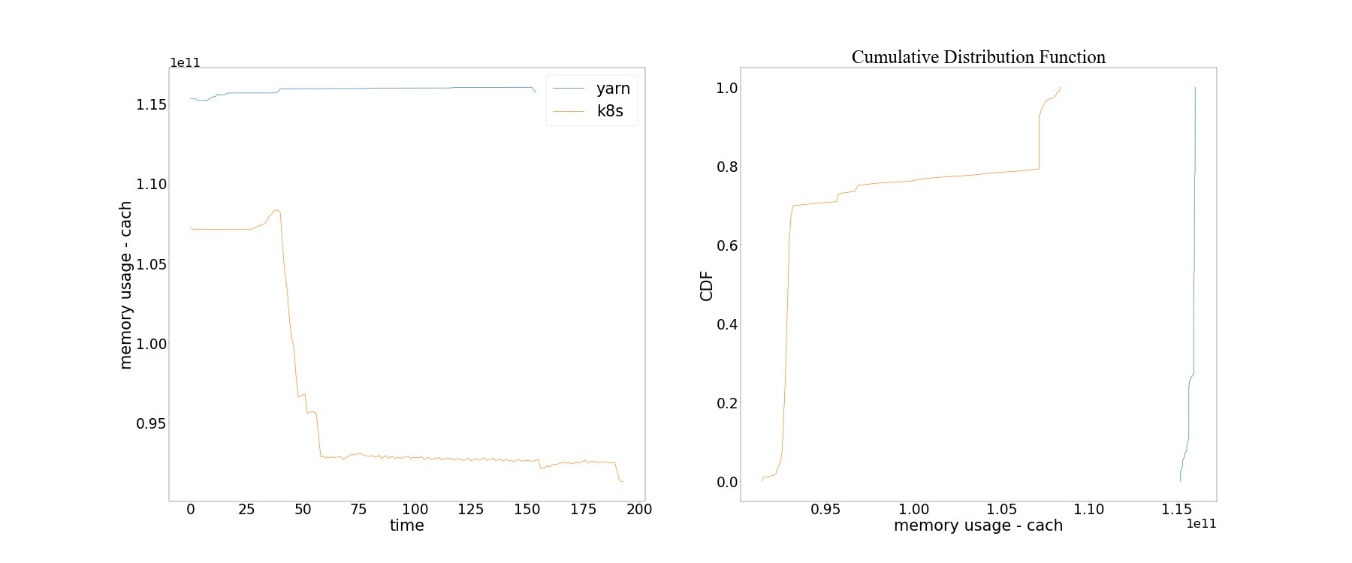


图3.16 WordCount测试第50组配置memory usage - cach数据

下图是memory usage - cach在teraSort20g下的80组比较结果。

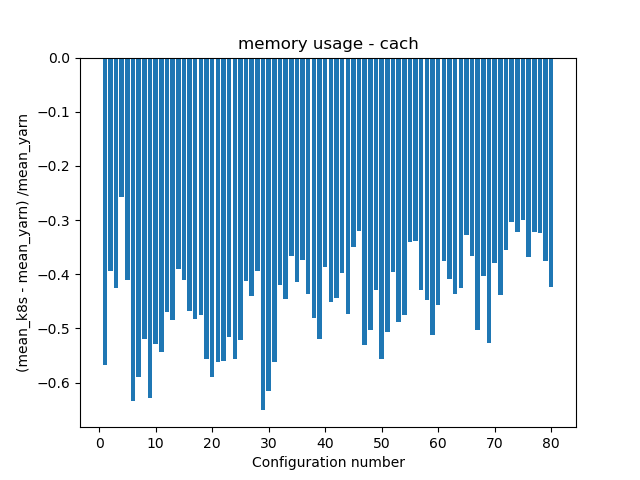


图3.17 TeraSort测试下memory usage - cach平均值的在不同环境的比较结果

使用第80组配置运行结果举例

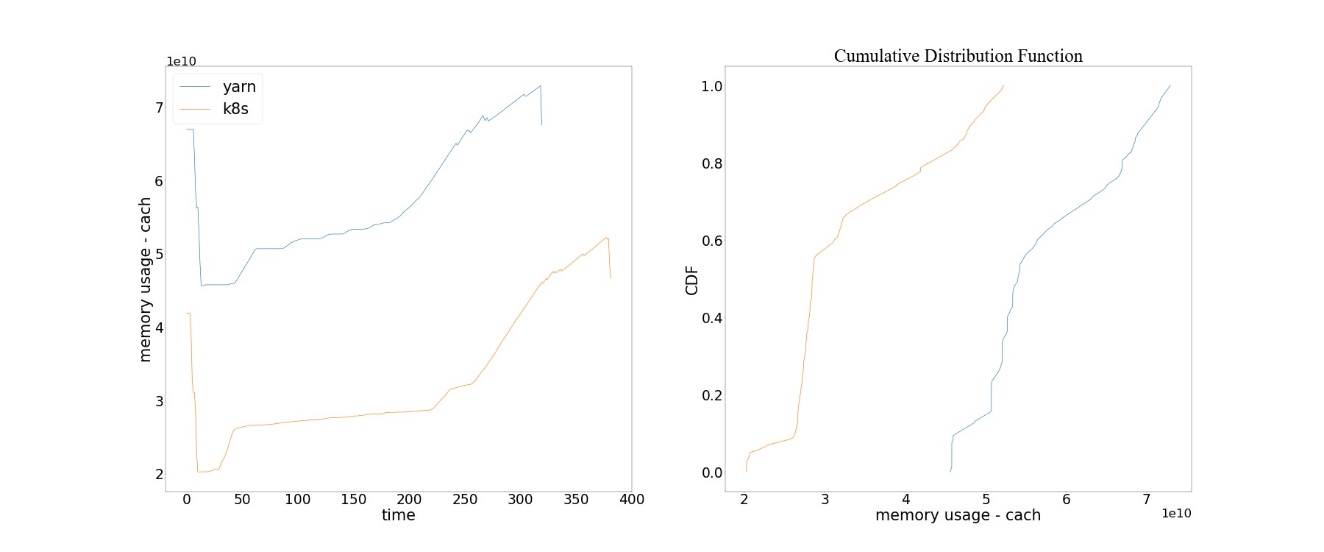


图3.18 TeraSort测试第80组配置memory usage - cach数据

基于上图3.15和3.17的结果，对于**memory usage - cach**事件，WordCount测试和TeraSort测试都有mean\_k8s大于mean\_yarn的现象出现。

在WordCount测试中，mean\_k8s的值为mean\_yarn的70%至98%。而在TeraSort测试中，mean\_k8s的值为mean\_yarn的35%至80%。

在3.2.2相似性分析测试中我们通过**dtw距离也发现memory usage - cach**在Yarn和Kubernetes环境下存在差异，而且该事件的dtw距离是所有事件中最大的。

##### 3.2.6.3 平均值比较方法总结

通过平均值比较的方法，我们找出以下在Yarn环境和Kubernetes环境下存在差异的事件。

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| ild\_stall.lcp | Stalls caused by changing prefix length of the instruction | pipeline |
| machine\_clears.memory\_ordering | Counts the number of machine clears due to memory order conflicts Spec update: SKL089 | memory |

表3.21 在Yarn环境和Kubernetes环境下存在差异的微体系结构事件

|  |  |  |
| --- | --- | --- |
| 事件名 | 含义 | 类别 |
| procs - total | 展示进程数量，侧面反映系统负载 | 进程数量 |
| net/total - send | 网络发包带宽 | 总体网络收发状态 |
| unix sockets - str | 流 | 开启unix统计(datagram, stream, listen, active)UNIX Domain SOCKET 是在Socket架构上发展起来的用于同一台主机的进程间通讯（IPC） |
| unix sockets - lis | lis监听的数量 | unix统计 |
| unix sockets - act | act(established)活动的确立的数量 | unix统计 |
| memory usage - cach | 展示进程数量，侧面反映系统负载 | 进程数量 |
| tcp sockets - lis | 网络发包带宽 | 总体网络收发状态 |

表3.22 在Yarn环境和Kubernetes环境下存在差异的微体系结构事件

### 3.3 总结

以上各种方法有其各自的特点，Dtw距离可以反映事件在波形和数值方面的差异，但不能反映出哪一种环境下该事件的值更大。

平均值比较法着重于数值上的差异，而且量化了数值差异。

Dtw距离的问题在于如果存在某一次的比较结果异常高，会对最终累加出的dtw距离产生巨大影响，而平均值比较法要求至少90%的情况下都满足mean\_k8s大于mean\_yarn或者mean\_k8s小于mean\_yarn，这就避免了极端情况带来的影响，但平均值比较法的问题在于仅仅通过一个平均值代表一次运行的所有数据，损失了大量信息。

以上的问题也可以在后续的工作中进行改进，例如在累加所有比较结果的Dtw距离时可以将值最高的5%和最低的5%去除掉，减少极端结果的影响。

另外一个值得注意的点是，同一事件在不同用例中的差异也会有巨大的差异。对于net/total – send,该事件在WordCount和TeraSort测试中都满足全部比较结果存在mean\_k8s大于mean\_yarn的情况。在WordCount测试中，mean\_k8s通常是mean\_yarn的1000倍以上，而在TeraSort测试中，mean\_k8s不到mean\_yarn的15倍。memory usage – cach也有类似情况出现。

在后续工作中，我们一方面会对已有方法进行一些改进，另一方面也会尝试更多的方法进行探究 。