预测应用程序在多个系统中的性能-成本权衡

Amir Nassereldine[1], Safaa Diab[1], Mohammed Baydoun[1],Kenneth Leach[2], Maxim Alt[2],Dejan Milojicic[2],Izzat El Hajj[1]

[1]贝鲁特美国大学，贝鲁特，黎巴嫩 [2]惠普企业，美国加利福尼亚州米尔皮塔斯

arXiv:2304.01676v1 [cs.DC] 4 Apr 2023

摘要——**在现代计算环境中，用户可能可以使用多个系统，例如本地集群、私有云或公有云。如此丰富的选择使用户难以为应用程序选择最适合其性能和成本目标的系统及配置。为帮助这类用户，我们提出了一种预测工具，可预测应用程序在多个系统中的完整性能-成本权衡空间。该工具会在部分系统的少量配置上运行并剖析用户提交的应用程序，然后利用这些信息预测该应用程序在所有系统的所有配置下的性能。预测模型通过离线运行大量应用程序、并使用多种配置收集的数据进行训练。该工具的突出特点包括：提供具有不同在线剖析要求的多种预测范围；自动选择用于在线剖析的少量配置和系统；通过部分运行进行在线剖析，从而无需将应用程序运行至完成即可作出预测；使用分类器区分扩展性良好与扩展性较差的应用程序；以及预测应用程序对其他用户干扰的敏感性。我们使用 69 个数据分析和科学计算基准测试评估该工具，这些基准测试运行于三种单节点 CPU 系统上，每种系统具有 8 至 9 种配置。结果表明，该工具只需适度的剖析开销，即可实现较低的预测误差。**

I. 引言

如今，应用程序开发人员可以使用多个系统来运行其应用程序。他们可以在所在公司或机构提供的不同本地集群或私有云之间进行选择，也可以选择广泛的公有云服务。然而，为应用程序选择最佳系统和配置颇具挑战，因为这取决于应用程序的行为以及用户的性能和成本目标。在应用程序方面，不同应用程序具有不同的性能瓶颈，因此它们在不同系统上的表现最佳。此外，当为应用程序提供更多系统资源时，有些应用程序可能实现良好扩展，成本几乎没有增加；有些应用程序可能因成本增加而性能收益逐渐降低；还有些应用程序甚至可能变慢，导致执行时间和成本同时增加。在用户方面，一些用户可能希望将成本降至最低，另一些用户可能希望将执行时间降至最低，还有一些用户可能希望在两者之间找到最佳权衡。

我们的目标是让用户全面了解任意应用程序在多个系统和配置下的性能-成本权衡空间。这些信息可以帮助用户做出最佳决策，以满足其性能和成本目标。然而，要提供此类信息，就必须在所有系统和配置上运行应用程序，成本将高得难以承受。因此，我们希望拥有性能建模工具，能够预测任意应用程序的性能-成本权衡，而无需等到这些应用程序运行完成，也无需在大量系统和配置上运行它们。

许多以往的研究通过采用预测技术，旨在帮助用户为应用程序选择最佳系统和配置。其中一个主要方向是使用优化技术，寻找能够使某个目标函数达到最小值的最优或近似最优系统和配置 [1]–[12]。然而，仅仅找到近似最优配置，并不能让用户足够灵活地实现其多样化目标和偏好。例如，用户可能愿意牺牲少量性能，以换取显著的成本节省。当用户能够全面了解性能-成本权衡空间时，才能更好地做出此类决策。

许多已有研究旨在预测应用程序在所有可用系统和配置上的性能 [13]–[22]。一种方法是使用解析模型 [13]–[15]，，并利用收集到的应用程序数据，为每个系统训练这些解析模型的系数。然而，这些模型往往针对特定的应用程序和系统，无法预测多个系统之间的相对性能。此外，通常需要在某个系统上多次运行应用程序直至完成，才能对该系统作出预测。另一种方法是利用离线收集的其他应用程序的信息来训练通用模型，从而只需少量在线数据收集，就能预测新应用程序的性能 [16]–[20]。我们采用的正是后一种方法。

我们提出了一种工具，用于预测任意应用程序在不同系统以及每个系统的不同资源配置下的性能—成本权衡。我们的工具会在少量称为 **指纹配置** 的配置上对提交的应用程序进行性能剖析，以生成应用指纹。随后，它将指纹传递给分类器，用于区分扩展性良好和扩展性较差的应用程序。对于扩展性良好的应用程序，它会将应用指纹传递给回归模型，用于预测该应用程序在所有系统以及每个系统的所有资源配置下的性能。对于扩展性较差的应用程序，它会使用一个

不同的回归模型，该模型仅预测应用程序在每个系统最小资源配置下的性能。我们的工具还会预测应用程序对不同类型干扰的敏感性。为了训练预测模型，我们在线下针对不同配置运行大量基准测试，并收集性能和性能分析数据。利用这些数据，我们会自动确定用于生成指纹的最佳指纹配置集和性能分析指标，并据此训练分类器和回归模型。

我们基于性能剖析信息生成应用程序指纹的总体方法受到 PARIS [16] 中工作的启发。不过，我们还做出了若干额外贡献。我们提供多个预测范围（全局、单个系统和局部），在准确度和在线性能分析开销方面各有不同。我们自动选择指纹配置，而不是手动挑选。我们根据部分运行中收集的相对指标进行预测，无须测量完整运行的执行时间，因此也不要求用户为指纹识别过程提供具有代表性的短时运行任务。我们使用分类器区分扩展性良好和较差的应用程序，使回归模型能够专注于特定类别的应用程序。我们还预测应用程序对不同类型干扰的敏感性，并量化离线收集训练数据时未进行穷尽式数据收集的影响。

我们使用 69 个数据分析和科学计算基准测试评估了该工具。这些基准测试运行在三种不同的单节点 CPU 系统上，每种系统有 8 至 9 种配置（共 26 种配置）。评估结果表明，只需在三种配置上对应用程序进行 30 秒的性能剖析，我们的工具就能预测应用程序在这三种系统（26 种配置）上的性能-成本权衡，平均误差为 22.5%。此外，如果将预测范围缩小到单个系统，我们的工具可以将预测的平均误差控制在 11.4% 至 15.6%。如果进一步缩小预测范围，我们的工具可以在单种配置上对应用程序进行性能剖析，并预测相近配置下的表现；大多数配置的错误率低于 10%。我们通过预测性能-成本权衡，展示了该工具在现实场景中的有效性——GROMACS [23], a（一种常用的分子动力学模拟应用程序），在三个系统上的平均误差为 17.3%，而对其执行时间仅 5% 的部分进行分析。我们表明，所提出的分类器使平均预测误差降低了 6.67%。我们还量化了基于部分运行而非完整运行对应用程序进行预测，以及使用训练数据覆盖率不完整而非完整的训练数据训练模型所产生的影响。未来，我们将扩大工具和评估的范围，涵盖更多类型的系统，包括具有网络通信的多节点系统，以及配备 GPU 等加速器的系统。

本文其余部分安排如下。第二节阐述了预测应用程序完整性能—成本权衡的必要性。第三节介绍了我们的预测工具在对提交的应用程序进行性能预测时所采用的工作流。第四节介绍了我们的工具如何部署在特定系统中，包括如何收集训练数据以及如何选择指纹配置。第五节概述了我们的方法，第六节全面评估了我们的工具及所作的各种设计决策。最后，第七节回顾了相关工作，第九节总结全文。

II. 动机

为应用程序选择最佳资源配置是一项颇具挑战的任务，既取决于应用程序的行为，也取决于用户的性能和成本目标。图1展示了三种不同应用程序在执行时间与成本之间的权衡（方法见第五节）。我们观察到，每个应用程序都有独特的行为，因此不同用户会采取不同的措施。

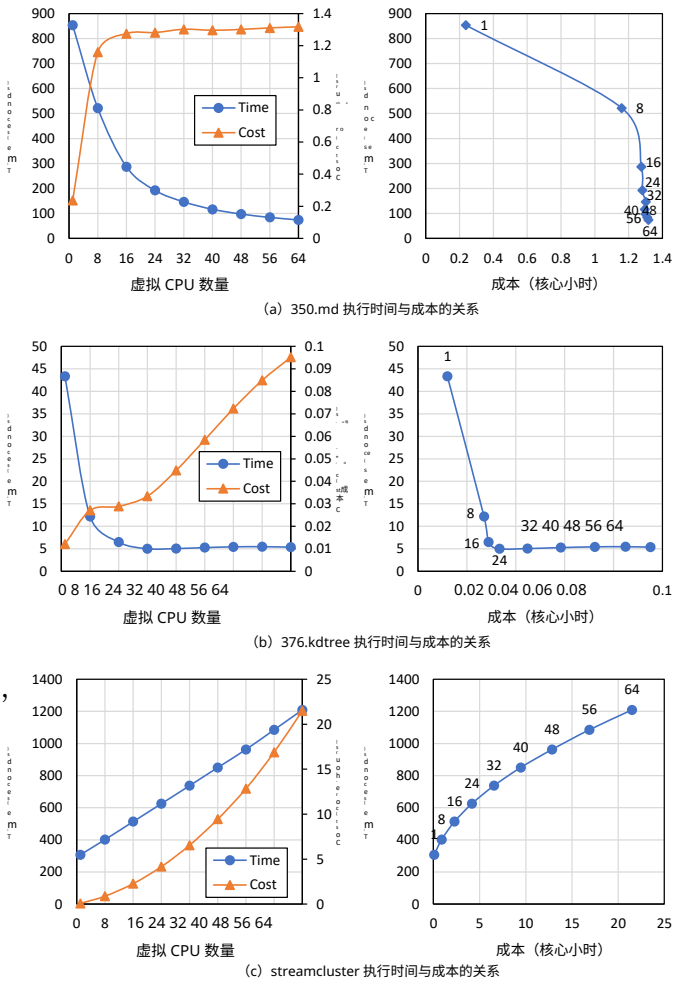


图1. 三种应用程序在执行时间与成本之间的权衡

图1(a)中的第一个应用程序 350.md 显示，从 1 个虚拟 CPU 增至 8 个虚拟 CPU 时，执行时间仅略有缩短

但成本却大幅增加。不过，超过这一点后，该应用程序开始表现出良好的扩展性，性能显著提升，而成本仅略有增加。对于这类应用程序，注重性能且愿意承担 8 个虚拟 CPU 预算的用户，不妨直接申请全部 64 个虚拟 CPU，因为只需增加少量成本，就能大幅提升性能。

图1(b)中的第二个应用程序 376.kdtree 从 1 个虚拟 CPU 增加到 16 个虚拟 CPU 时，执行时间显著缩短，而成本仅小幅增加。然而，超过这一点后，执行时间几乎不再缩短，成本却大幅增加。对于此类应用程序，即使是注重性能的用户也不应请求超过 16 个虚拟 CPU，因为这只会增加额外成本，却无法带来性能收益。

图1(c)中的第三个应用程序 streamcluster 在分配更多虚拟 CPU 后，执行时间反而增加。对于此类应用程序，其扩展性较差，无论是注重预算的用户还是注重性能的用户，都应仅使用 1 个虚拟 CPU 来执行该应用程序。

在上述每种情况下，用户都需要能够查看执行时间与性能之间的完整权衡空间，才能做出符合其目标的正确决策。仅了解成本最低或性能最佳的配置，并不足以做出细致的决策。当用户可以使用多个系统时，这一问题会进一步复杂化，因为不同的帕累托最优选择可能来自不同的系统。基于此，我们提出了一种性能预测工具，可预测任意应用程序在多个系统以及每个系统上的不同配置下的性能。

III. 预测工具工作流

本节介绍预测工具部署后的工作流。换言之，我们将说明工具如何对提交的应用程序进行性能预测。我们在第 III-A 节概述全局权衡预测器的工作流，然后在第 III-B 、III-C 和 III-D 节讨论该工作流中的各个步骤。我们在第 III-E 节讨论如何预测应用程序对干扰的敏感性。最后，我们在第 III-F 节介绍如何通过权衡预测范围来提高预测准确率并降低在线性能分析开销。

A. 全局权衡预测器

全局权衡预测器可预测应用程序在所有可用系统的所有配置下的性能-成本权衡。图 2 展示了全局权衡预测器的整体工作流。第一步，提交给工具的应用程序将在不同系统中选定的一组配置（称为指纹配置）上运行，以生成该应用程序的指纹。应用程序的指纹 是一组性能分析指标值，这些值是在应用程序运行于每个指纹配置时收集的。指纹将作为输入特征向量，提供给工具的预测模型。

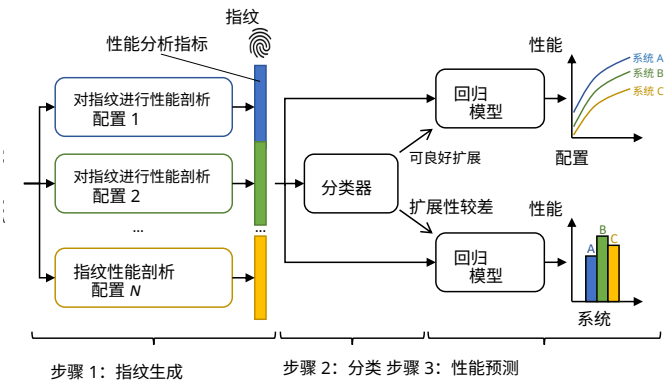


图 2. 全局权衡预测器工作流

第二步，将指纹传递给分类器，由其将应用程序分为两类之一：可良好扩展或扩展性较差。根据分类结果，第三步将对该应用程序采取不同的处理方式。

在第三步中，根据应用程序是可良好扩展还是扩展性较差，将其指纹传递给两个回归模型之一。对于可良好扩展的应用程序，回归模型会预测该应用程序在所有系统和配置下的某项性能指标。本文主要评估中使用的性能指标，是相对于基准系统和配置的加速比。随后，可利用此相对加速比推导相对成本，并可视化完整的性能-成本权衡空间。如果应用程序在某种配置下运行至完成，还可利用此加速比推导绝对执行时间和绝对成本。对于扩展性较差的应用程序，无须预测每个系统上的所有配置，因为使用资源最少的配置来执行应用程序始终是最佳选择。因此，系统会将指纹传递给另一个回归模型，该模型仅预测应用程序在资源最少的配置下于所有系统上的相对性能。

B. 指纹生成

在指纹生成步骤中，会在一组指纹配置上运行并进行性能剖析已提交的应用程序，以生成该应用程序的指纹。执行此步骤时的重要方面包括：选择要运行的指纹配置、选择要收集的性能分析指标，以及数据收集的时长。

1）指纹配置的选择：用于分析应用程序的指纹配置数量及其选择，取决于目标系统集合，并在训练预测模型的部署时确定。使用更多指纹配置有助于提高预测准确率，但随着数量增加，收益会逐渐减少。另一方面，这也会增加生成所提交应用程序指纹所需的执行资源

。我们将在第四节-B中介绍如何在部署时选择指纹配置，并在第 VI-B 节评估选择指纹配置所带来的影响。

2）性能分析指标的选择：要收集的性能分析指标取决于执行性能分析的系统。不同系统可能提供不同的性能分析工具和指标，因此，我们的工具支持在指纹中使用来自不同系统和配置的不同指标。在所有情况下，我们都使用相对指标（即每秒事件数），因为它们独立于应用程序的绝对执行时间。使用独立于绝对执行时间的指标具有两项优势。第一项优势是，无需运行应用程序直至完成即可收集这些指标。第二项优势是，可以使用执行时间范围广泛的各种应用程序来训练模型。在训练预测模型并执行特征选择的过程中，会在部署时决定从每个系统和配置中使用的具体指标集合。

3）数据收集范围：在指纹配置上运行提交的应用程序时，可以仅运行应用程序的一部分，也可以运行至完成。对于执行频繁且每次工作负载规模相同的短时应用程序，运行至完成可能是可以接受的，因为运行至完成所产生的开销可以通过摊销降低。在这种情况下，可以测量应用程序在指纹配置上的绝对执行时间和相对执行时间，并将其纳入指纹，从而使模型更容易对其他配置进行预测。然而，对于执行时间较长且不经常执行的应用程序，在所有指纹配置上运行至完成将需要大量执行资源，所产生的成本也无法得到摊销。因此，最好能够仅基于相对性能分析指标进行预测，而无需测量应用程序的绝对执行时间和相对执行时间。本文的主要评估使用从部分运行中收集的相对性能分析指标，因为我们优先考虑无需运行提交的应用程序直至完成即可对其进行预测。不过，在第六节-F中，我们评估了将应用程序运行至完成，并将其在指纹配置上的相对执行时间纳入指纹所产生的影响。

C. 按可扩展性分类

在分类步骤中，我们将提交的应用程序分为扩展性良好和扩展性较差两类。如果某个应用程序从资源最少的配置扩展到资源最多的配置时，在大多数系统上运行速度变慢，我们就将其定义为扩展性较差（实际上，这种行为往往在所有系统上都表现一致）。如果应用程序扩展性较差，那么为该应用程序提供更多资源只会使其产生显著更高的成本，却无法获得任何明显的性能收益。图1(c)展示了这类应用程序的一个示例。在这种情况下，始终应在资源最少的配置上执行该应用程序，因此没有必要对其他配置做出准确预测。我们只需预测该应用程序在不同系统之间的相对性能。通过将扩展性良好的应用程序与扩展性较差的应用程序分开，并为每一类使用不同的回归模型，我们可以让每个回归模型专注于提高其所属类别的准确度。

我们使用随机森林分类器按可扩展性对应用程序进行分类。我们在第 VI-A 节评估该分类器的准确度。本文的主要评估使用这一分类器；不过，第 VI-E 节评估了不使用该分类器、而对所有应用程序使用同一回归模型的影响。

D. 性能预测

在性能预测阶段，我们将已提交应用程序的指纹输入相应的回归模型，由该模型预测应用程序在不同系统和配置下的性能。我们将相对于基准系统和配置的加速比作为预测的性能指标，但也可以训练模型来预测其他指标。基准系统和配置在指纹配置选择过程中确定。我们使用 XGBoost [24] 作为回归模型。我们在第 VI-B 节评估这些回归模型的准确度。

E. 预测干扰敏感性

在用户可能访问的系统中，有些系统可能由其他用户共享，他们可以在同一节点上同时运行各自的应用程序，例如云环境。因此，了解预测性能在受到其他用户干扰时可能波动的程度，对用户很有帮助。为了预测应用程序对干扰的敏感性，我们训练另一个干扰感知回归模型，预测已提交应用程序在所有系统和配置下的性能，以及在每种系统和配置下处于不同干扰极端情况时的性能。我们针对四种干扰模式进行预测：无干扰、计算密集型干扰、缓存密集型干扰和内存密集型干扰。因此，对于每种系统配置组合，该模型会为每种干扰模式分别生成一个输出，共四个输出。

该干扰感知回归模型所做的预测之所以有用，是因为它们能够为用户提供一个性能波动范围，说明当用户将其应用程序与其他应用程序同时运行时，可能会观察到多大的性能波动。这些预测还可以帮助调度器决定应将哪些应用程序共置在一起。最后，即使在非共享环境中，同样的方法也可用于预测应用程序对不同服务质量约束的敏感性。例如，了解应用程序的性能在调整不同服务质量约束时可能产生的波动，有助于服务提供商决定系统的哪些部分

应进行限流，以满足不断变化的功率预算。我们在第六节-D中评估了干扰感知回归模型的准确度。

F. 预测范围

迄今为止介绍的全局权衡预测器会在一组特定的系统和配置上对应用程序进行性能剖析，从而预测其在所有系统和配置上的表现。不过，在某些情况下，用户可能只关心某个特定系统，甚至只关心某个系统中的特定配置范围。在这种情况下，缩小预测范围有望提高准确度并降低在线性能分析开销。因此，除了全局权衡预测器，我们还提供单系统预测器和局部权衡预测器。

单系统预测器与全局权衡预测器类似，不同之处在于它会针对每个系统分别训练模型。通常，它还只需在来自目标系统的较少指纹配置上运行应用程序。

另一方面，局部权衡预测器会在单个系统的单个配置上对应用程序进行性能剖析，并预测应用程序在邻近配置上的相对性能。图3展示了局部权衡预测器的工作流。第一步，在用户指定的系统和配置上运行并剖析所提交的应用程序，以生成由收集到的性能分析指标值组成的指纹。第二步，将该指纹传递给针对该系统和配置的回归模型，由该模型预测应用程序在核心数略多或略少的配置上的相对性能。同样，我们使用 XGBoost [24] 作为回归模型。

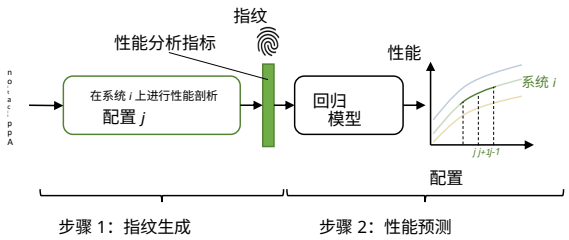


图 3. 局部权衡预测器工作流

提供这一替代预测器的原因是，用户可能已经对适用的系统和配置范围有了初步想法，只需对这一决策进行微调。在这种情况下，局部权衡预测器能够提供更准确的预测，且开销更低，因为只需对应用程序进行一次性能剖析。指纹也可以从应用程序的历史运行记录中获取。在这种情况下，如果用户选择在未来扩展或缩减规模，我们的局部预测器便可用于回顾性地提出建议，预测预期的性能-成本权衡。

IV. 预测工具部署

本节介绍如何在目标系统集合上部署我们的预测工具。我们首先在第四节-A中介绍如何收集训练数据，然后在第四节-B中介绍如何训练预测模型。

A. 训练数据收集

部署的第一步是收集训练模型所需的数据。为此，我们在不同系统和配置上运行大量基准测试，并在运行期间对其进行性能剖析。基准测试会运行至完成，并测量其执行时间，以便计算其在每个系统和配置上的相对加速比。在本文的主要评估中，收集训练数据时，我们会在所有系统上运行每项基准测试，并在每个系统的所有配置上运行。这种方法的优点是能为预测模型提供大量信息，从而提高其准确度。不过，这也意味着收集训练数据需要大量运行。另一种做法是，在系统和配置的随机子集上对每项基准测试进行性能剖析，并在训练数据覆盖范围不完整的情况下训练预测模型。我们在第六节-G中评估使用非穷尽训练数据的影响。

B. 全局和单系统权衡模型的训练

对于局部权衡预测器，我们针对每个系统和配置分别训练一个预测模型。该预测模型的输入是从该系统和配置中收集的性能分析指标，输出则是同一系统上邻近配置的相对性能。

对于全局和单系统权衡预测器，一个重要环节是选择用于生成这些模型输入的指纹配置的数量和具体配置。尝试所有可能的指纹配置组合以找到最优组合，代价会高得难以承受。因此，我们改用贪心方法，具体如下：首先假设只使用一个指纹配置。我们尝试将所有系统上的所有配置分别作为该指纹配置，并选择使可良好扩展的应用程序的回归模型错误最低的配置，将其确定为第一个训练配置。接着假设使用两个指纹配置。我们尝试将所有剩余配置分别作为第二个指纹配置，再次选择错误最低的配置，并将其确定为第二个指纹配置。我们不断重复这一过程，逐步增加指纹配置，直到错误的改善幅度低于某个阈值，表明增加指纹配置带来的边际收益很小。基线配置的选择方式类似。

通过尝试一组指纹配置，我们指的是使用这些配置生成指纹，并利用这些指纹训练预测模型。在此过程中，我们将所有收集到的性能分析指标都用作指纹的一部分，因为同时选择指纹配置和性能分析指标的代价过高。

表 I 从各系统收集并由特征选择选出的指标（括号中的数字表示全局预测器在哪些指纹配置中选用了该指标）

系统	指标	已选
系统 1	对齐故障	No
	所有数据缓存访问	No
	所有 TLB 已刷新	No
	分支预测 L1 TLB 未命中 L2 命中	是 (1)
	BP L1 TLB 未命中、L2 TLB 未命中	是（1）
	分支指令	是（1）
	所有分支中的分支预测失败	No
	所有缓存引用中的缓存未命中	是 (1)
	缓存引用	是（1）
	上下文切换	No
	CPU 时钟 已使用的 CPU	是（1）
	CPU周期 GHz	No
	CPU 迁移	No
	解码、分派与调度令牌停顿（ALU 令牌停顿）	是（1）
	解码、分派与调度令牌停顿（退休令牌停顿）	是（1）
	de_dis_分派令牌停顿1_加载队列 令牌停顿	No
	de_分派令牌停顿1_存储队列 令牌停顿	是（1）
	数据 TLB 加载未命中占所有数据 TLB 缓存访问的比例	是（1）
	数据 TLB 加载	No
	已退休指令	是 (1)
	已退休的 MMX 浮点指令_	No
	已退休的 SSE 浮点指令_	No
	已退休的浮点、SSE 和 AVX 操作	No
	指令缓存提取停顿：任意指令缓存停顿	是（1）
	指令缓存取指停顿：指令缓存停顿反压	是（1）
	指令缓存取指停顿：指令缓存停顿分派队列为空	是（1）
	每周期指令数	No
	所有指令 TLB 缓存访问中的指令 TLB 加载未命中	No
	指令 TLB 加载	是（1）
	一级数据 TLB 未命中	是（1）
	所有一级数据缓存访问中的一级数据缓存加载未命中	是（1）
	L1 数据缓存加载次数	No
	L1 数据缓存预取	是（1）
	所有 L1 指令缓存访问中的 L1 指令缓存加载未命中	是 (1)
	L1 指令缓存加载	No
	由数据缓存未命中导致的 L2 缓存访问	No
	由指令缓存未命中导致的 L2 缓存访问	是（1）
	由数据缓存未命中导致的 L2 缓存命中	No
	由指令缓存未命中导致的 L2 缓存命中	No
	来自二级缓存硬件预取的二级缓存命中	No
	二级缓存未命中来自数据缓存未命中	是（1）
	来自指令缓存未命中的二级缓存未命中	No

系统	指标	已选择
系统 1（续）	二级缓存请求统计_ls_读取_blk_c_	是 (1)
	二级缓存请求统计_ls_读取_blk_cs_	No
	L2_DTLB_未命中	是（1）
	L2 指令 TLB 未命中	是 (1)
	L2 延迟，L2 等待填充的周期数	No
	L2_请求，G1 可缓存指令缓存读取	是（1）
	L2_请求，G1_LS 读取_blk_C_S_	是 (1)
	L2 请求 g1_rd blk l_	是 (1)
	L2 请求 g1_rd blk x_	No
	L2 请求 g2_bus 锁发起者	No
	ls_一级数据 TLB 未命中、TLB 重载、1G 二级缓存命中	No
	ls_TLB刷新	No
	重大缺页	是（1）
	次要缺页故障	是（1）
	缺页	No
	SSE_AVX_停顿	No
	每条指令的停顿周期数	是（1）
	后端停滞周期 后端空闲周期	No
	前端停滞周期 前端空闲周期	是 (1)
	任务时钟 已使用的 CPU	是（1）
	已分派的微操作	是（1）
	已退休微操作	No
系统 2 和 3	分支指令	No
	分支未命中	是（2）
	总线周期	是（2）
	缓存未命中	是（2、3）
	缓存引用	是（3）
	上下文切换	No
	CPU 迁移	是（2、3）
	已使用的 CPU	是（2、3）
	周期活动周期数 一级数据缓存未命中	是（3）
	周期活动周期数 二级缓存未命中	是（2、3）
	周期活动周期 L3 未命中	是（2）
	周期活动周期任意内存_	是 (2)
	周期活动停顿 L1 数据缓存未命中	No
	周期活动停顿 L2 缓存未命中	是（2）
	周期活动停顿 L3 缓存未命中	是 (3)
	周期活动内存任意停顿_	No
	数据 TLB 加载未命中	是（3）
	数据 TLB 加载	是（3）
	数据 TLB 存储未命中	是（2、3）
	数据 TLB 存储操作	No

系统	指标	已选择
系统 2 和 3（续）	已退役浮点算术指令.128b 双精度打包	是 (2、3)
	已退役浮点算术指令.128.位单精度打包	是（2）
	已退役浮点算术指令。256.位双精度打包	是（3）
	已退役浮点算术指令。256.位单精度打包	是 (2)
	浮点算术指令退役，512.位打包双精度	是（3）
	浮点算术指令退役，512 位打包单精度	是（3）
	浮点算术指令 已退役的标量双精度指令	No
	已退役浮点算术指令.已退役的标量单精度指令	是（2、3）
	浮点辅助.any	是（3）
	GHz	是（2、3）
	每周期指令数	是（3）
	已退役指令.any	No
	指令 TLB 加载未命中	No
	指令 TLB 加载	是（2、3）
	一级数据缓存加载未命中	是（3）
	L1 数据缓存加载次数	是（2、3）
	L1 数据缓存存储次数	No
	二级缓存请求总数（预取）	是（2、3）
	二级缓存请求总数（读所有权）	是 (3)
	二级缓存请求未命中	是（3）
	二级缓存请求引用	是（3）
	二级缓存读请求所有权命中	是（2、3）
	二级缓存读请求所有权未命中	是（3）
	末级缓存加载未命中	是（2、3）
	末级缓存加载	是（2、3）
	末级缓存存储未命中	No
	末级缓存存储操作	是（3）
	最长延迟缓存未命中	是（2、3）
	最长延迟缓存引用	是（2）
	已退役内存指令总加载数	是（3）
	内存指令退休总存储数	是（2）
	内存加载退休一级缓存命中	是（2、3）
	内存加载退役 L1 未命中	是 (2)
	内存加载退役 L2 命中	是（2）
	内存加载退役 L2 缓存未命中	是（2）
	内存加载退役 L3 缓存命中	是（2）
	已退役的内存加载 L3 缓存未命中	是 (2、3)
	内存加载	是（3）
	内存存储操作	是（2、3）
	缺页	是（2、3）
	resource_stalls.any	是（3）

确定指纹配置集合后，我们会应用标准特征选择技术，以减少每种配置所需的性能分析指标数量。每种配置选出的指标数量和集合可能各不相同。特征选择能够提高预测准确率，还能减少在线收集的指标数量，从而降低在线性能分析开销。

V. 方法

我们使用以下规格的三台单节点 CPU 系统对工具进行评估：

- 系统 1：一台双路 AMD EPYC7532 CPU 系统，具有 64 个核心（64 个线程）和 512 吉字节主存
- 系统 2：一台双路 Intel Xeon Gold 6242 CPU 系统，具有 32 个核心（64 个线程）和 192 吉字节主存
- 系统 3：一台双路 Intel Xeon Gold 5120 CPU 系统，配备 28 个核心（56 个线程）和 64 吉字节主存

在每个系统上，所考虑的配置包括 1 个虚拟 CPU，以及虚拟 CPU 数量为 8 的所有倍数的配置，主存平均分配给各个虚拟 CPU。例如，系统 1 有以下配置：1 个虚拟 CPU + 8 GB、8 个虚拟 CPU + 64 GB、16 个虚拟 CPU + 128 GB、24 个虚拟 CPU + 192 GB、32 个虚拟 CPU + 256 GB、40 个虚拟 CPU + 320 GB、48 个虚拟 CPU + 384 GB、56 个虚拟 CPU + 448 GB，以及 64 个虚拟 CPU + 512 GB。总的来说，三个系统共有 26 种配置。

在本文的评估中，我们使用 Docker [25] 配置分配给每个应用程序的虚拟 CPU 数量和物理内存容量。我们还使用 perf [26] 对基准测试进行性能剖析，并收集大量性能分析指标。不过，我们的工作流和预测模型并不依赖于 Docker 和 perf。也可以采用其他方式配置资源和收集性能分析指标，而我们的训练与预测工作流仍可按相同方式运行。

在获取训练数据时，我们从每个系统收集约 60 项性能分析指标。由于不同 CPU 提供的性能计数器各不相同，因此各系统收集的指标也有所不同。每个系统收集的指标如表 I 所示。该表还列出了在全局权衡预测器使用的各项指纹配置中，经特征选择后保留的指标。我们使用 perf 测量了收集 60 项性能分析指标所产生的性能开销，结果显示其平均值低于 2%。

为预测应用程序对干扰的敏感性，我们使用 stress-ng [27] 在系统中未使用的核心上模拟计算密集型、缓存密集型或内存密集型干扰。不过，我们的方法并不局限于 stress-ng，也可以使用其他模拟干扰（或服务质量约束）的工具，而不会影响工作流。

为了训练和测试我们的预测模型，我们使用了来自七个基准测试套件和库的 69 个数据分析和科学计算基准测试。这些基准测试列于表 II 中。

我们的训练和推理工作流使用

评估中使用的基准测试 表 II

套件	基准测试
NAS 并行基准测试 [28]	bt、cg、ep、ft、is、lu、mg、sp
PARSEC3.0 [29]	blackscholes, bodytrack, canneal, freqmine, streamcluster, 期权互换, splash2.barnes, splash2.cholesky, splash2.radiosity, splash2.volrend
SPEC ACCEL [30]	550.pmd, 552.pep, 554.pcg, 555.pseismic、556.psp、559.pmniGhost、560.pilbdc, 563.pswim, 570.pbt
SPEC OMP 2012 [31]	350.md, 351.bwaves, 352.nab, 357.bt, 358.botsalgn, 359.botsspar, 360.ilbdc, 363.swim, 367.imagick, 370.mgrid, 371.applu, 372.smithwa, 376.kdtree
Parboil [32]	bfs, cutcp, 直方图, lbm, 磁共振成像网格化, 单精度通用矩阵乘法, 稀疏矩阵向量乘法、模板计算、两点角相关函数
Rodinia [33]	bfs, 心脏壁, 热点, K均值, 白细胞, LUD OpenMP, Needle, Pathfinder, SRAD v2
MLlib [34]	相关性, 决策树分类器, 因子分解机分类器, 梯度提升树分类器, gmm, K均值, 逻辑回归, 线性支持向量分类器, mlp, pca, 随机森林分类器 , 摘要器

表 III 可扩展性分类器的混淆矩阵

True 可扩展性	预测的可扩展性：可良好扩展	预测的可扩展性：扩展性较差
可良好扩展	58	2
扩展性较差	1	8

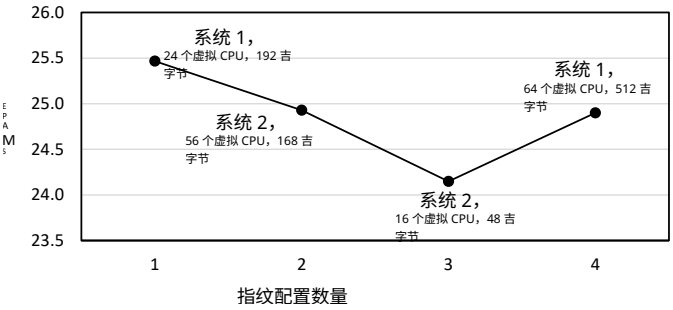


图4. 随着指纹配置数量增加，全局权衡预测器的回归误差变化（标签表示每次迭代中选定的配置）

Python 实现。我们使用 NumPy [35]和 pandas [36]执行数据准备和处理任务。我们使用 Scikit- Learn [37]和 XGBoost [24] 库来实现分类、回归和特征选择算法。

在整个评估部分，我们使用十折交叉验证来评估模型的预测误差。对于每一折，我们使用训练集基准测试中完整运行和部分运行的指纹，以及测试集基准测试中部分运行的指纹。对于部分运行，我们在实验中对基准测试进行 30 秒的指纹采集。不过，用户可以调整指纹采集时长，以权衡准确度和指纹采集开销。

我们使用对称平均绝对百分比误差（SMAPE） [38] 作为指导优化的主要错误指标。也可以使用其他错误指标，例如平均绝对百分比误差（MAPE），但我们选择 SMAPE，因为其取值范围限定在 0% 到 200%，而且无论预测值高于还是低于真实值，惩罚都是对称的（因为我们预测的是比率）。

VI. 评估

A. 分类准确率

表 III 展示了随机森林分类器的混淆矩阵，该分类器在全局权衡预测器中根据基准测试的可扩展性对其进行分类。结果表明，分类器的准确度很高，正确识别了 60 个扩展性良好的应用程序中的 58 个，以及 9 个应用程序中的 8 个扩展性较差的应用程序。因此，我们的工具能够有效筛除扩展性较差的应用程序，使主回归模型能够专注于扩展性良好的应用程序。

B. 回归模型准确率

图4展示了随着指纹配置数量增加，适用于可良好扩展的应用程序的全局权衡预测器的回归错误如何变化。图中的各点标注了每次迭代找到的最佳指纹配置。

根据图4中的结果，我们得出两个重要观察。第一个观察是，随着指纹配置从一个增加到三个，回归模型误差持续下降，平均误差达到 24.2%。然而，指纹配置数量超过三个后，准确度会降低，因为模型会因信息过载而受到影响。因此，我们将指纹配置的数量固定为三个（总共 26 个）。请注意，应用特征选择后，平均误差进一步降低至 22.5%。

第二个观察是，选出的三个指纹配置涵盖了两个不同的系统和三种不同的虚拟 CPU 数量。这一结果表明，指纹配置选择过程会自动使选出的指纹配置集合更加多样化。不过请注意，三个系统中有一个未出现在任何指纹配置中。这一结果表明，我们的工具无需在某个系统上运行应用程序，也能对该系统做出预测。

上述结果表明，只需在三种配置上对应用程序进行分析，我们的工具就能在多个系统上实现合理的预测准确率。不过，在某些情况下，用户可能已经确定了某个特定系统，并不关心其他系统上的相对性能。对于这类用户，我们的工具还为每个系统提供单系统回归模型，仅针对该系统进行性能-成本权衡预测。表 IV 展示了随着指纹配置数量的增加，这些单系统模型的预测误差如何变化。我们发现，将问题限定在单个系统后，即使仅在较少数量的配置上分析应用程序，我们的工具也能降低预测误差。请注意

应用特征选择后，使用三种指纹配置时，系统 1、2 和 3 的平均误差分别进一步降至 11.4%、12.5% 和 15.6%。

表 IV 指纹配置数量增加时单系统模型回归误差的变化

配置数量		1	2	3	4
系统 1	错误	13.5	13.2	12.3	12.3
	配置	24 个虚拟 CPU，192吉字节	32 个虚拟 CPU，256吉字节	8 个虚拟 CPU，64GB	56 个虚拟 CPU，448吉字节
系统 2	错误	15.5	13.6	13.4	13.4
	配置	16 个虚拟 CPU，48 吉字节	56 虚拟 CPU，168吉字节	40 个虚拟 CPU，120吉字节	48 个虚拟 CPU，144吉字节
系统 3	错误	19.0	17.4	17.0	17.1
	配置	24 个虚拟 CPU，24GB	32 个虚拟 CPU，32GB	40 个虚拟 CPU，40GB	8 个虚拟 CPU，8GB

全局预测器和单系统预测器的预测误差在不同基准测试之间有所差异。图 5 展示了各预测器在不同基准测试中的预测误差分布。显然，我们的模型能够以很低的误差预测许多基准测试。此外，中位数误差始终低于平均误差，这表明预测误差低于平均值的基准测试更多。不过，某些基准测试的预测误差仍然较高，尤其是离群值。如何通过更先进的分类方法筛除这类基准测试，是未来的研究方向。

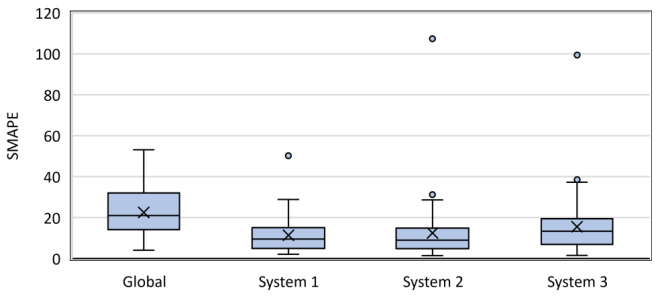


图 5. 全局和单系统预测器在各基准测试中的误差分布

C. 应用程序案例研究：*GROMACS*

为了在现实场景中测试我们工具的有效性，我们将其应用于一种常用的分子动力学模拟应用程序 GROMACS [23]。我们使用一个全局权衡预测模型，该模型基于表 II 中的基准测试训练而成，且此前从未接触过 GROMACS。我们仅在执行时间最快的系统和配置上，使用该模型的指纹配置执行并剖析 GROMACS，耗时仅占其总执行时间的 5%。随后，我们将获得的指纹传递给模型，由模型预测 GROMACS 在所有系统和配置上的相对性能。图 6 展示了实际和预测的相对性能值。显然，我们的工具能够很好地预测 GROMACS 在这三个系统上的表现。总体而言，我们的工具对 GROMACS 的平均预测误差为 17.3%。

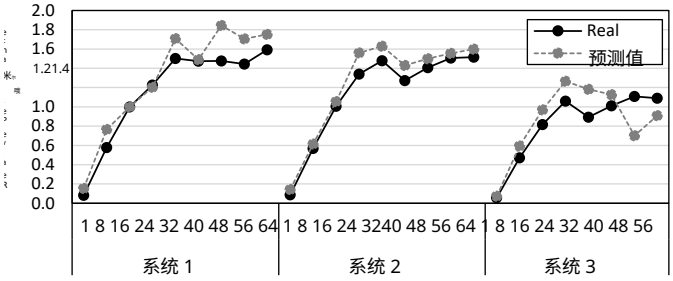


图 6. GROMACS [23]的实际相对性能与预测相对性能

D. 预测干扰敏感性

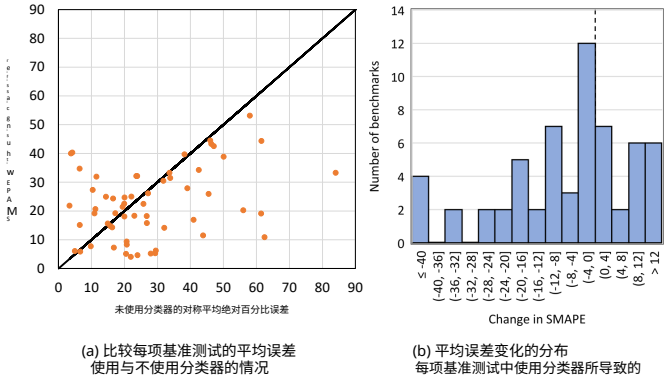
回顾第二节 III-E，我们的工具还提供干扰感知回归模型，用于预测应用程序在每个系统和配置上、面对不同干扰极端情况时的性能，让用户了解应用程序对不同类型干扰的敏感性。表 V 展示了全局预测器和单系统预测器在预测所考虑的三种干扰极端情况下的性能时所达到的预测误差。显然，与假设不存在干扰时相比，预测误差相近，但略高。值得注意的是，对于大多数系统，缓存密集型干扰似乎是最难预测的干扰类型，其误差略高于其他类型。

表 V 在不同干扰极端情况下预测性能时的全局与单系统回归误差

预测器	干扰类型		
	计算密集型	内存密集型	缓存密集型
全局	24.7	24.6	26.1
仅系统 1	14.2	13.0	16.1
仅系统 2	15.5	14.1	17.9
仅系统 3	22.9	19.4	14.8

E. 设置分类阶段的影响

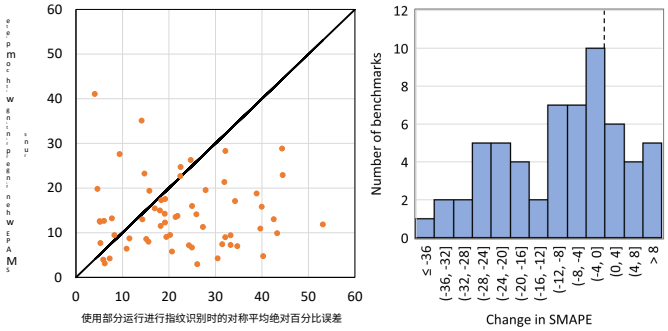
请回想一下，全局权衡预测器工作流中的一个步骤是使用分类器区分扩展性良好和扩展性较差的应用程序，使主回归模型能够专注于扩展性良好的应用程序。在本小节中，我们评估所提出分类器的作用。图 7(a) 比较了使用和不使用分类器时回归模型的平均预测误差。图中的每个点代表一个基准测试。显然，大多数基准测试在使用分类器后平均误差有所降低。图 7(b) 展示了使用分类器后各基准测试平均误差变化的分布。显然，大多数基准测试的平均误差再次有所降低。所有基准测试的平均误差变化均值和中位数分别为 -6.67% 和 -2.25%。如果不使用分类器，整体平均误差将为 29.2%。这些结果验证了所提出的分类阶段能够提升回归模型质量。



(a) 比较每项基准测试的平均误差
使用与不使用分类器的情况

(b) 平均误差变化的分布
每项基准测试中使用分类器所导致的

图 7. 在回归模型之前使用分类器对回归模型误差的影响



(a) 比较每项基准测试的平均误差 时
使用部分运行与完整运行进行指纹识别

(b) 每个基准测试的平均误差变化分布
因使用以下方式进行指纹识别而导致：
使用完整运行而非部分运行

图8. 使用部分运行与完整运行进行指纹识别的影响

F. 使用部分运行与完整运行进行指纹识别的影响

在主要评估中，我们仅使用从应用程序在指纹配置上部分运行期间收集的相对性能分析指标。不过，如果应用程序在指纹配置上完整运行，就可以推导出其在各指纹配置之间的相对性能，并将其纳入指纹中，从而为回归模型提供更多信息，帮助其做出准确预测。在本小节中，我们评估使用完整运行而非部分运行对回归模型准确率的影响。

图 8(a) 比较了使用部分运行与完整运行进行指纹识别时，回归模型的平均预测误差。图中的每个点代表一个基准测试。显然，使用完整运行时，大多数基准测试的平均误差都会降低。图 8(b) 展示了进行指纹识别时，以完整运行代替部分运行所导致的各基准测试平均误差变化分布。再次可以看出，大多数基准测试的平均误差都有所降低。所有基准测试的误差平均变化和中位变化分别为 -8.44% 和 -7.27%。如果在指纹配置上测量性能，并将测量结果纳入指纹，则总体平均误差为 14.1%。这些结果表明，如果部署者愿意在进行指纹识别时让提交的应用程序运行至完成，就能显著降低我们工具的预测误差。不过，使用部分运行仍可提供相当准确的预测。

G. 部分数据覆盖训练的影响

在主要评估中，我们通过在每个系统和配置上运行每项基准测试来收集数据，以训练回归模型。如此全面地覆盖训练数据需要大量执行资源。在本小节中，我们通过对每项基准测试使用所有系统和配置的不同随机子集进行运行，评估部分训练数据覆盖率的影响。

图9展示了全局回归模型和单系统回归模型的预测准确度如何随训练数据覆盖率变化。正如预期，预测误差会随着训练数据覆盖率的降低而增大，但相比全局模型，单系统模型的这一变化更加平缓。事实上，在单系统模型中，即使数据覆盖率低至 25%，错误仍保持在 20% 以下。这些结果表明，工具的部署者可以根据所需的预测质量，在训练数据收集期间使用的资源量与准确度之间进行权衡。

在本实验中，使用部分数据覆盖训练时，每个基准测试所采用的配置均为随机选择。通过采用更复杂的技术来选择数据收集期间每个基准测试所使用的配置，可以进一步提高使用部分数据覆盖训练时的准确度。这些技术将作为我们未来的工作内容。

H. 局部权衡预测器准确率

回顾第二节VI-B，我们的全局权衡预测器仅通过分析应用程序在四种指纹配置下的表现，就能在预测三个系统上的权衡空间时实现 22.5% 的预测误差。此外，通过缩小预测范围，我们的单系统模型仅分析应用程序在两种指纹配置下的表现，便将预测误差控制在 11.4% 至 15.6% 之间。本小节评估第三节III-F中描述的局部权衡预测器。该预测器进一步缩小预测范围，尝试仅分析应用程序在一种配置下的表现，从而实现更低的预测误差。

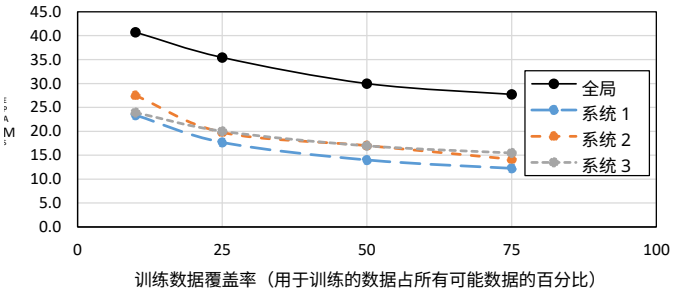


图 9. 使用部分数据覆盖训练对全局和单系统回归模型准确度的影响

图 10 展示了局部权衡预测器中各系统和配置的回归模型的平均预测误差。这里，每个系统和配置的回归模型所报告的误差，是指预测同一系统上邻近配置中应用程序相对性能时的误差。我们观察到，在大多数情况下，这些模型的平均预测误差低于 10%。这一结果表明，局部权衡预测器通过缩小预测范围有效降低了预测误差。

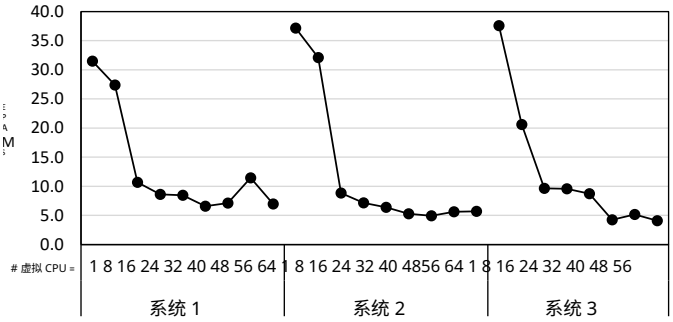


图 10. 各系统上每种配置的本地权衡预测器预测误差

图 10 的一个有趣发现是，使用 1 个虚拟 CPU 和 8 个虚拟 CPU 的配置，其预测误差始终较高。尤其是，预测器难以预测使用 1 个虚拟 CPU 和 8 个虚拟 CPU 的配置之间的相对性能。在考察基准测试后，我们发现，1 个虚拟 CPU 与 8 个虚拟 CPU 之间的切换会对性能产生格外难以预测的影响，原因有二。一方面，应用程序使用 1 个虚拟 CPU 运行时，不会产生任何并行化开销，因此与其他配置相比，这种配置具有性能优势。另一方面，应用程序使用 1 个虚拟 CPU 运行时，内存预算更紧张，因此这种配置在性能方面处于劣势。

VII. 相关工作

大量文献探讨了如何利用预测工具，帮助用户选择最适合运行其应用程序的系统和/或配置。其中一个主要方向是运用优化技术，寻找能够最小化某个目标函数 [1]–[12] 的最优或近似最优系统与配置。例如，CherryPick [1]利用贝叶斯优化，为公有云中的大数据分析应用程序寻找近似最优配置。Arrow [2]通过将底层性能信息融入贝叶斯优化，降低了其脆弱性。Accordia [3] 利用上置信界技术并支持针对前景不佳配置的提前中止机制，提升了基于高斯过程的方法的性能。Selecta [4] 同时为计算和存储寻找近似最优配置。Micky [5], SARA [6], 和 Vanir [7] 则同时为一组工作负载寻找近似最优配置。Lynceus [8] 同时优化选择配置以及应用程序级参数（例如机器学习算法的超参数）。Bilal 等人 [9]评估了多种黑盒优化算法，并表明采用梯度提升回归树的贝叶斯优化表现良好。谷歌的 Autopilot [39] 会自动调整工作负载的资源使用量，在降低资源过度分配的同时，尽可能降低因资源不足而导致应用程序受到限制或被终止的风险。

所有这些先前的研究都着眼于寻找能够优化特定目标的近乎最优配置。然而，用户可能有多样化的目标和偏好，其中一些目标可能较为微妙。例如，寻求性能最佳的系统和配置的用户，可能愿意牺牲少量性能，以换取大幅降低成本。反过来，优化成本的用户也可能愿意承担略高的成本，以获得显著更好的性能。为了做出这类决策（或使用自动化工具来辅助决策），用户需要了解性能与成本（或其他关注指标）之间的完整权衡，从而在不同的帕累托最优选项之间进行选择。

许多先前的研究旨在预测应用程序在所有可用系统和配置上的性能，从而为用户提供对权衡空间的完整视图 [13]–[22]。其中一些研究使用解析模型 [13]–[15]，通过手工构造的方程对性能进行建模，并针对每个应用程序和系统训练该方程的系数。例如，Ernest [13] 提出了一个包含多个项的方程，用于捕捉固定成本、计算和不同的通信模式。它使用非负最小二乘法训练该方程的系数，并为应用程序与系统类型的每种组合推导出不同的系数。Baughman 等人 [14] 提出了一个包含不同项的方程，这些项表示配置的组成部分，例如虚拟 CPU 数量或内存，并使用非线性最小二乘法为每个应用程序寻找系数。这些解析方法的缺点在于，它们往往针对相关的特定应用程序和/或系统，这意味着在能够预测该应用程序在该系统上的性能之前，必须先对目标应用程序在目标系统上进行大量在线剖析。要将解析模型推广到多个应用程序和系统，就需要考虑更多能够区分不同应用程序和系统的特征。然而，手工构造能够接收如此大量特征的方程是一项困难的任务。相比之下，更复杂的机器学习模型能够更好地处理如此庞大的特征集。

许多研究提出了通用模型，这些模型利用离线收集的信息进行训练，涵盖各种应用程序和各种配置 [16]–[22]。Fu 等人 [40]研究了机器学习在性能预测方面的通用适用性。PARIS [16] 训练随机森林模型，并通过对用户提供的、能够代表该应用程序且在选定配置下运行的一项短时

任务进行性能剖析，生成应用指纹并将其作为该模型的输入。ARVMEC [17]训练 XGBoost [24]模型，并采用与 PARIS 类似的方法生成指纹（或基因）。Mariani 等人 [18]训练随机森林模型，但该模型的输入是应用程序的硬件无关性能剖析，可在任何系统上生成，但需要重新编译应用程序。Paragon [19] 和 Quasar [20]使用协同过滤来预测应用程序在多种系统和配置下的性能，并利用预测信息进行调度和资源分配。尽管这些预测以这种方式使用时效果良好，但先前研究 [16] 发现，协同过滤提供的预测结果未经处理时准确度不足，我们的经验也与此一致。

我们提出的方法是：在选定的配置上对应用程序进行剖析，生成应用程序指纹，再将该指纹输入通用模型。这一方法受到 PARIS [16] 相关工作的启发。不过，我们的工作与 PARIS 有几个方面的不同。首先，PARIS 在多个系统和配置范围内全局预测性能，而我们提供多种预测范围（全局、单系统和局部），并针对不同用例在准确度与在线性能分析开销之间进行权衡。其次，在 PARIS 中，指纹配置是预先确定的，或必须由部署者选择；而我们提供了一种自动化方法，用于识别最佳的指纹配置集合。第三，PARIS 要求用户提交一个具有代表性的短时运行任务来生成指纹，而我们使用原始应用程序生成指纹，无需将其运行至完成。第四，我们使用分类器区分扩展性良好和扩展性较差的应用程序，并表明所提出的分类器有助于提升性能。第五，我们预测应用程序对不同类型干扰的敏感性。

已有不少研究针对特定应用程序预测其在多个系统和/或配置下的性能，例如分布式矩阵乘法 [41], [42],深度学习训练 [43], Hadoop MapReduce 作业 [44], VASP 作业 [45], 以及电子科学工作流 [46]。我们的工作面向通用应用程序，不针对特定框架或应用领域。

许多研究旨在建模干扰对应用程序执行的影响 [19], [20], [47]–[49]。例如，Paragon [19] 和 Quasar [20]使用协同过滤，预测新应用程序在不同干扰和异构条件下的表现，以便有效地协同调度应用程序。uPredict [47] 使用微基准测试训练应用专用性能模型，从而学习应用程序对不同资源争用的敏感性。在我们的工作中，我们通过预测应用程序在不同极端干扰条件下的性能，界定其对干扰的敏感性范围。

PseudoApp [50] 通过构建一个能够模仿原始应用程序行为的伪应用程序，免去了将应用程序迁移到云端以评估其性能的需要。这种方法与我们的工作互为补充：如果用户希望在迁移应用程序之前了解性能-成本权衡，也可以在指纹识别过程中使用伪应用程序。

VIII. 局限性与未来工作

我们当前工作的一个局限是，仅在单节点 CPU 系统上运行的数据分析和科学计算基准测试中评估了我们的工具。未来，我们计划将评估扩展到其他类别的应用程序，以及其他类型的系统，例如通过网络通信的多节点系统和配备 GPU 等加速器的系统。另一个局限是，尽管我们的工具对许多基准测试的预测误差较低，但对某些基准测试的预测误差仍然较高。未来，我们计划扩展分类阶段，以筛除这类难以预测的基准测试。还有一个局限是，我们当前的工作流会对每个提交的应用程序进行指纹识别，以便做出预测。不过，一种可能的优化方法是维护一个已见过的应用程序指纹数据库，从而避免重复对同一应用程序进行指纹识别。这种优化能否奏效，取决于应用程序的性能剖析和相对性能在多大程度上受输入数据集影响。研究这种优化方法也是我们未来工作的内容。

IX. 结论

我们提出了一种工具，用于预测应用程序在不同系统以及每个系统的不同资源配置下的性能-成本权衡。我们的工具会在少量系统和配置上对提交的应用程序进行性能剖析，无需运行至完成即可测量其执行时间。随后，工具利用收集到的性能剖析信息，根据应用程序的可扩展性对其进行分类，并预测该应用程序在所有系统和配置下的性能。该工具提供多种预测范围（全局、单系统和局部），其准确度和在线性能分析开销各不相同；同时，它还会预测应用程序对不同类型干扰的敏感性。我们通过在多种配置下对大量应用程序进行性能剖析来训练预测模型，并在进行预测时自动确定一组适合对提交的应用程序进行性能剖析的配置。我们在三个单节点 CPU 系统上进行的评估表明，该工具能够以较低的错误有效预测应用程序在多个系统和配置下的性能。

致谢

本研究由 Hewlett Packard Enterprise 资助。感谢匿名审稿人提出的意见，这些意见有助于改进本文。我们也感谢 Aditya Dhakal、Alok Mishra、Barbara Chapman、Diman Zad Tootaghaj、Eitan Frachtenberg、Gourav Rattihalli、Lianjie Cao、Mohammad Sonji、Ninad San- jay Hogade、Norbert Bianchin、Paulo Roberto Pereira de Souza Filho、Pedro Bruel 和 Rolando Pablo Hong Enriquez 提供的反馈与帮助。

参考文献

[1] O. Alipourfard、H. H. Liu、J. Chen、S. Venkataraman、M. Yu 和 M. Zhang，“CherryPick：为大数据分析自适应地发掘最佳云配置”，载于第 *14* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 17*），2017 年，第 469–482 页。[2] C.-J. Hsu、V. Nair、V. W. Freeh 和 T. Menzies，“Arrow：用于寻找最佳云虚拟机的低层增强贝叶斯优化”，载于*2018* 年 *IEEE* 第 *38* 届国际分布式计算系统会议（*ICDCS*）。IEEE，2018 年，第 660–670 页。[3] Y. Liu、H. Xu 和 W. C. Lau，“Accordia：针对重复数据密集型应用的自适应云配置优化”，载于*ACM* 云计算研讨会论文集，2019 年，第 479–479 页。[4] A. Klimovic、H. Litz 和 C. Kozyrakis，“Selecta：面向数据分析的异构云存储配置”，载于*2018* 年 *USENIX* 年度技术会议（*USENIX ATC 18*），2018 年，第 759–773 页。[5] C.-J. Hsu、V. Nair、T. Menzies 和 V. Freeh，“Micky：一种更经济的云实例选择方案”，载于 *2018* 年 *IEEE* 第 *11* 届国际云计算会议（*CLOUD*）。IEEE，2018 年，第 409–416 页。[6] A. Xiao、Z. Lu、J. Li、J. Wu 和 P. C. Hung，“SARA：快速稳定地为异构大数据工作负载寻找最优云配置”，《应用软计算》，第 85 卷，文章编号 105759，2019 年。[7] M. Bilal、M. Canini 和 R. Rodrigues，“为分析集群寻找合适的云配置”，载于第 *11* 届 *ACM* 云计算研讨会论文集，2020 年，第 208–222 页。[8] M. Casimiro、D. Didona、P. Romano、L. Rodrigues、W. Zwaenepoel 和 D. Garlan，“Lynceus：数据分析作业的经济高效调优与资源配置”，载于*2020* 年 *IEEE* 第 *40* 届国际分布式计算系统会议（*ICDCS*）。IEEE，2020 年，第 56–66 页。[9] M. Bilal、M. Serafini、M. Canini 和 R. Rodrigues，“最佳云配置会从树上长出来吗？用于优化云工作负载的黑盒算法实验评估”，《*VLDB* 会刊》，第 13 卷，第 12 期，第 2563–2575 页，2020 年。[10] C.-J. Hsu、V. Nair、T. Menzies 和 V. W. Freeh，“Scout：寻找最佳云配置的经验丰富的向导”，*arXiv* 预印本 *arXiv:1803.01296*，2018 年。[11] Y. Lin、J. Briggs 和 A. Barker，“FIFE：一种基于基础设施即代码的框架，用于评估来自多个云平台的虚拟机实例”，载于*2020* 年 *IEEE/ACM* 第 *13* 届效用计算与云计算国际会议（*UCC*）。IEEE，2020 年，第 91–100 页。[12] Y.-W. Wu、Y.-J. Xu、H. Wu、L.-G. Su、W.-B. Zhang 和 H. Zhong，“Apollo：使用数据驱动方法快速为大数据分析选取最优云配置”，《计算机科学与技术学报》，第 36 卷，第 5 期，第 1184–1199 页，2021 年。[13] S. Venkataraman、Z. Yang、M. Franklin、B. Recht 和 I. Stoica，“Ernest：高效预测 {大规模} 高级分析的性能”，载于第 *13* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 16*），2016 年，第 363–378 页。[14] M. Baughman、R. Chard、L. Ward、J. Pitt、K. Chard 和 I. Foster，“云上应用性能的分析与预测”，载于第 *11* 届 *IEEE/ACM* 效用计算与云计算国际会议（*UCC*），2018 年。[15] O. Aaziz、J. Cook 和 M. Tanash，“用于表征和评估作业性能的预期应用运行时间建模”，载于*2018* 年 *IEEE* 集群计算国际会议（*CLUSTER*）。IEEE，2018 年，第 543–551 页。[16] N. J. Yadwadkar、B. Hariharan、J. E. Gonzalez、B. Smith 和 R. H. Katz，“在多个公有云中选择最佳虚拟机：一种数据驱动的性能建模方法”，载于*2017* 年云计算研讨会论文集，2017 年，第 452–465 页。[17] Y. Xu、J. Li、Z. Lu、J. Wu、P. C. Hung 和 A. Alelaiwi，“ARVMEC：面向边缘云环境中物联网的虚拟机自适应推荐”，《并行与分布式计算学报》，第 141 卷，第 23–34 页，2020 年。[18] G. Mariani、A. Anghel、R. Jongerius 和 G. Dittmann，“面向高性能计算应用的云性能预测：一种以用户为中心的方法”，载于*2017* 年 *IEEE/ACM* 第 *17* 届集群、云与网格计算国际研讨会（*CCGRID*）。IEEE，2017 年，第 524–533 页。[19] C. Delimitrou 和 C. Kozyrakis，“Paragon：面向异构数据中心的服务质量感知调度”，载于第十八届编程语言与操作系统架构支持国际会议论文集，丛书 ASPLOS ’13。美国纽约州纽约市：美国计算机协会，2013 年，第 77–88 页。[20] ——，“Quasar：资源高效且具备 QoS 感知能力的集群管理”，载于《编程语言与操作系统体系结构支持国际会议》第19届会议论文集，ASP-LOS ’14系列。美国纽约州纽约市：美国计算机协会，2014年，第127–144页。[21] Y. Lin、A. Barker和J. Thomson，“使用半监督非负矩阵分解对虚拟机潜在特征进行建模并预测应用性能”，载于2020年第13届IEEE云计算国际会议（CLOUD）。IEEE，2020年，第470–474页。[22] Y. Chen、L. Lin、B. Li、Q. Wang和Q. Zhang，“Silhouette：面向大规模分析的高效云配置探索”，《IEEE并行与分布式系统汇刊》，第32卷，2021年，第2061、2049页。[23] D. Van Der Spoel、E. Lindahl、B. Hess、G. Groenhof、A. E. Mark和H. J. Berendsen，“GROMACS：快速、灵活且免费的工具”，《计算化学杂志》，第26卷第16期，第1701–1718页，2005年。[24] T. Chen和C. Guestrin，“XGBoost：可扩展的树提升系统”，载于第22届ACM SIGKDD知识发现与数据挖掘国际会议论文集，2016年，第785–794页。[25] S. Hykes，“Docker”，2022年。 [在线]。可访问：https://www.docker.com [26] A. C. De Melo，“新的Linux‘perf’工具”，载于Linux Kongress演示文稿，第18卷，2010年，第1–42页。[27] C. I. King，“stress-ng”，2022年。 [在线]。可访问：https://wiki.ubuntu.com/Kernel/Reference/stress-ng [28] D. H. Bailey、E. Barszcz、J. T. Barton、D. S. Browning、R. L. Carter、L. Dagum、R. A. Fatoohi、P. O. Frederickson、T. A. Lasinski、R. S. Schreiber等，“NAS并行基准测试总结及初步结果”，载于Supercomputing’91：1991年ACM/IEEE超级计算会议论文集。IEEE，1991年，第158–165页。[29] X. Zhan、Y. Bao、C. Bienia和K. Li，“PARSEC3.0：带有网络协议栈和SPLASH-2X的多核基准测试套件”，《ACM SIGARCH计算机体系结构新闻》，第44卷第5期，第1–16页，2017年。[30] G. Juckeland、W. Brantley、S. Chandrasekaran、B. Chapman、S. Che、M. Colgrove、H. Feng、A. Grund、R. Henschel、W.-M. W. Hwu等，“SPEC ACCEL：用于测量硬件加速器性能的标准应用程序套件”，载于《高性能计算机系统性能建模、基准测试与仿真国际研讨会》。Springer，2014年，第46–67页。[31] M. S. M¨uller、J. Baron、W. C. Brantley、H. Feng、D. Hackenberg、R. Henschel、G. Jost、D. Molka、C. Parrott、J. Robichaux等，“SPEC OMP2012——使用OpenMP的并行系统应用程序基准测试套件”，载于OpenMP国际研讨会。Springer，2012年，第223–236页。[32] J. A. Stratton、C. Rodrigues、I.-J. Sung、N. Obeid、L.-W. Chang、N. Anssari、G. D. Liu和W.-m. W. Hwu，“Parboil：面向科学与商业吞吐量计算的修订版基准测试套件”，可靠与高性能计算中心，第127卷，第27页，2012年。[33] S. Che、M. Boyer、J. Meng、D. Tarjan、J. W. Sheaffer、S.-H. Lee和K. Skadron，“Rodinia：异构计算基准测试套件”，载于2009年IEEE工作负载特征分析国际研讨会（IISWC）。IEEE，2009年，第44–54页。[34] X. Meng、J. Bradley、B. Yavuz、E. Sparks、S. Venkataraman、D. Liu、J. Freeman、D. Tsai、M. Amde、S. Owen等，“MLlib：Apache Spark中的机器学习”，《机器学习研究杂志》，第17卷第1期，第1235–1241页，2016年。[35] C. R. Harris、K. J. Millman、S. J. van der Walt、R. Gommers、P. Virtanen、D. Cournapeau、E. Wieser、J. Taylor、S. Berg、N. J. Smith、R. Kern、M. Picus、S. Hoyer、M. H. van Kerkwijk、M. Brett、A. Haldane、J. F. del R’ıo、M. Wiebe、P. Peterson、P. G’erard-Marchant、K. Sheppard、T. Reddy、W. Weckesser、H. Abbasi、C. Gohlke和T. E. Oliphant，“使用NumPy进行数组编程”，《自然》，第585卷第7825期，第357–362页，2020年9月。 [在线]。可访问：https://doi.org/10 1038 41586 020 2649 2 。/s---T. pandas开发团队，“pandas-dev/pandas：Pandas”，2020年2月。 [36] [ ] 10 5281 3509134 在线。可访问：https://doi.org/。/zenodo。F. Pedregosa、G. Varoquaux、A. Gramfort、V. Michel、B. Thirion、O. Grisel、M. Blondel、P. Prettenhofer、R. Weiss、V. Dubourg等，“Scikit-learn：Python中的机器学习”，《机器学习研究杂志》，第12卷，第2825–2830页，2011年。[37] [38] Wikipedia，“对称平均绝对百分比误差”，2022年。[在线]。可访问：https://en.wikipedia.org/wiki/Symmetric mean absolute percentage error

– –

– –

[39] K. Rzadca、P. Findeisen、J. Swiderski、P. Zych、P. Broniek、J. Kusmierek、P. Nowak、B. Strack、P. Witusowski、S. Hand 等，“Autopilot：谷歌的工作负载自动扩展”，载于第十五届欧洲计算机系统会议论文集，2020年，第1–16页。[40] S. Fu、S. Gupta、R. Mittal和S. Ratnasamy，“关于使用机器学习进行黑盒系统性能预测”，载于第*18*届*USENIX*网络系统设计与实现研讨会（*NSDI 21*），2021年，第763–784页。[41] M. Son和K. Lee，“云计算中机器学习作业的分布式矩阵乘法性能估计器”，载于*2018*年第*11*届*IEEE*云计算国际会议（*CLOUD*）。IEEE，2018年，第638–645页。[42] J. Kim、M. Son和K. Lee，“MPEC：面向数据挖掘作业的扩展式云环境中的分布式矩阵乘法性能建模”，*IEEE*云计算汇刊，2019年。[43] P. Dube、T. Suk和C. Wang，“AI Gauge：云中深度学习的运行时估计”，载于*2019*年第*31*届计算机体系结构与高性能计算国际研讨会（*SBAC-PAD*）。IEEE，2019年，第160–167页。[44] H. Herodotou、F. Dong和S. Babu，“没有一种（集群）规模适用于所有场景：面向数据密集型分析的自动集群规模调整”，载于第*2*届*ACM*云计算研讨会论文集，2011年，第1–14页。[45] Q. Wang、J. Li、S. Wang和G. Wu，“一种新颖的两步作业运行时间基于系统输入参数的估算方法，”载于*2019 IEEE* 第四届国际云计算与大数据分析会议（*ICCCBDA*）。IEEE，2019 年，第 311–316 页。[46] F. Nadeem、D. Alghazzawi、A. Mashat、K. Faqeeh 和 A. Almalaise，“使用机器学习集成方法预测异构分布式系统中电子科学工作流的执行时间”，*IEEE Access*，第 7 卷，第 25 138–25149 页，2019 年。[47] H. Moradi、W. Wang、A. Fernandez 和 D. Zhu，“uPredict：一种基于用户级性能剖析器的多租户云预测框架”，载于 *2020 IEEE* 国际云工程会议（*IC2E*）。IEEE，2020 年，第 73–82 页。[48] S. Govindan、J. Liu、A. Kansal 和 A. Sivasubramaniam，“Cuanta：量化共享片上资源干扰对整合型虚拟机的影响”，载于第二届 *ACM* 云计算研讨会论文集，2011 年，第 1–14 页。[49] Y. Koh、R. Knauerhase、P. Brett、M. Bowman、Z. Wen 和 C. Pu，“虚拟环境中的性能干扰效应分析”，载于*2007 IEEE* 国际系统与软件性能分析研讨会。IEEE，2007 年，第 200–209 页。[50] B. C. Tak、C. Tang、H. Huang 和 L. Wang，“PseudoApp：面向云迁移的应用程序性能预测”，载于*2013 IFIP/IEEE* 综合网络管理国际研讨会（*IM 2013*）。IEEE，2013 年，第 303–310 页。