Journal of Computer Applications
计算机应用,2021,41(6):1597 - 1603
ISSN 1001-9081
CODEN JYIIDU
2021-06-10
http://www.joca.cn

文章编号:1001-9081(2021)06-1597-07
DOI:10.11772/j.issn.1001-9081.2020121929

面向并发程序中锁机制的智能化推荐方法

张　杨*,董士程
(河北科技大学 信息科学与工程学院,石家庄 050018)
(*通信作者电子邮箱zhangyang@hebust.edu.cn)

摘　要:软件开发人员在并发编程中面临着各种Java锁的选择。针对如何选择合适的锁机制进而提高程序性能的问题,提出了一种帮助并发程序开发人员选择锁机制的推荐方法LockRec。首先,采用程序静态分析技术,对并发程序中锁机制的使用进行分析并确定影响程序性能的程序特征属性;然后,使用改进的随机森林算法构建锁机制推荐模型,以帮助开发人员在同步锁、可重入锁、读写锁、邮戳锁这四种锁之中进行选择。分别选取了四种现有的机器学习数据集对LockRec进行了实验,所提方法的推荐准确率均值可达95.1%;此外使用真实并发程序对LockRec的推荐结果进行分析。实验结果表明LockRec可以有效提高并发程序的执行效率。

关键词:并发程序;锁机制;随机森林;推荐模型;程序性能
中图分类号:TP311.53　　**文献标志码**:A

Intelligent recommendation method for lock mechanism in concurrent program

ZHANG Yang*, DONG Shicheng
(*School of Information Science and Engineering*, *Hebei University of Science and Technology*, *Shijiazhuang Hebei* 050018, *China*)

Abstract: The choices of Java locks are faced by the developers during parallel programming. To solve the problem of how to choose the appropriate lock mechanism to improve the program performance, a recommendation method named LockRec for developers of concurrent program to choose lock mechanism was proposed. Firstly, the program static analysis technology was used to analyze the use of lock mechanism in concurrent programs and determine the program feature attributes that affect the program performance. Then, the improved random forest algorithm was used to build a recommendation model of lock mechanism, so as to help the developers to choose the lock among synchronization lock, reentrant lock, read-write lock, and stamped lock. Four existing machine learning datasets were selected to experiment with LockRec. The average accuracy of the proposed LockRec is 95.1%. In addition, the real-world concurrent programs were used to analyze the recommendation results of LockRec. The experimental results show that LockRec can effectively improve the execution efficiency of concurrent programs.

Key words: concurrent program; lock mechanism; random forest; recommendation model; program performance

0　引言

同步控制是并发程序中用于保证程序状态和数据正确的必备措施,目前学术界和工业界提出了多种同步控制方式,包括锁、原子块和软件事务型内存。锁是目前使用较为广泛且程序员熟悉度较高的一种同步控制方式;原子块使用乐观锁策略以原子方式更新数据,操作简单,但是原子操作的程序编写相对复杂;软件事务型内存使用事务机制实现并发操作,但在大量事务执行时对硬件要求较高,在数据竞争频繁的情况下开销较大。在多核时代,虽然使用锁会遇到锁竞争问题,但是锁仍然是目前作为同步控制的主要方式。

Java语言提供了同步锁、可重入锁、读写锁、邮戳锁等多种类型的锁,其中同步锁是一种互斥锁,程序开发者多以同步方法或同步块的形式使用同步锁;可重入锁是一种互斥锁,它和同步锁具有相同的基本行为和语义,但是在同步锁的基础上扩展了许多功能;读写锁除了提供了可重入锁的一些特性外,又把锁分为读锁和写锁,读写锁允许更大程度的并发;从JDK1.8版本开始引入的邮戳锁还提供了锁升级、降级、优化读等操作。每种机制都有优缺点与各自的适用场景,这要求程序开发者必须熟练掌握它们的特点,并在并发软件开发时作出正确的选择。

多种锁机制并存的局面以及在使用不同锁机制时程序性能的差异情况导致程序开发者在选择锁机制时常常感到困惑,以往的做法是程序员手动地把程序从一种同步机制改写为另一种同步机制,但这要求程序员对每种同步机制都非常的熟悉,手动的改写也容易给程序引入新的错误,破坏了程序的可维护性。一些研究使用自动重构工具帮助程序员在多种锁机制之间进行转换,比如针对重入锁以及读写锁提出了一种自动重构算法[1],或者面向可定制锁[2]和邮戳锁[3]的自动重构方法,以及在进行优化同步瓶颈的研究中提出的一种锁分解方式[4]。虽然上述方式可以帮助程序开发人员比较各种锁

收稿日期:2020-12-10;**修回日期**:2021-03-29;**录用日期**:2021-04-06。　**基金项目**:国家自然科学基金资助项目(61440012);河北省教育厅科学研究计划重点项目(ZD2019093);河北省基础研究计划重点基础专项(18960106D)。

作者简介:张杨(1980—),男,河北秦皇岛人,副教授,博士,CCF高级会员,主要研究方向:智能化软件、软件重构;　董士程(1995—),男,山东济南人,硕士研究生,主要研究方向:智能化软件。

的性能，但是有时不成熟的重构工具往往会给程序引入一些错误，特别是对于并发程序。从目前的研究来看，大部分工作局限于对于锁的使用及其相互转化，对于锁影响程序性能的原因缺乏全面的认识，缺少对锁机制的直接推荐使用方法，而且将机器学习算法应用于锁机制的研究还较少。

针对目前研究存在的问题，本文以并发程序中的锁机制作为主要研究对象，研究其制约性能发挥的因素。为了解决上述锁机制选择困难的问题，本文提出了一种帮助并发程序开发人员选择锁机制的推荐方法LockRec，它基于已有的并发程序信息，使用改进的随机森林算法构建锁机制推荐模型，可以帮助开发人员在同步锁、可重入锁、读写锁、邮戳锁四种锁之中进行选择，使用UCI的公开四种数据集对提出的模型进行验证，推荐准确率平均值可达95.1%。此外，本文还对两个真实程序给出了锁机制的推荐结果以及使用推荐前后的锁机制性能对比，结果显示推荐后的程序性能平均加速比分别为1.35、1.29，证明LockRec可以有效地帮助开发者选择合适的锁机制。

1 相关工作

为了提高并发编程的效率，学术界提出了很多辅助并发编程的技术和工具，国内外许多大学和研究机构也对同步机制相关的问题进行了研究。Schäfer等[5]对如何正确地重构并发程序进行了研究，并保证程序正确地进行同步控制，设计了一个软件重构工具Relocker，该工具可以实现源对源的锁重构。Rose等[6]通过对相关程序进行检测，收集在执行时有关锁关系的信息，从而对多态锁类型进行动态推理，以便检测出在多线程Java程序中的数据竞争。此外，Baum等[7]对面向软件事务性内存的重构技术进行了研究；Ernst等[8-9]提出了一种形式化语义的锁规范，使用抽象解释和类型理论两种不同的分析方式来表达形式语义，可以正确推断和检查Java锁规则的工具。AutoLocker[10]是一个锁的推测工具，可以实现对象访问前加锁和解锁操作，但是没有研究如何在不同的锁之间进行推荐。虽然上述方式可以帮助程序开发人员比较各种锁的性能，但是有时不成熟的重构工具往往会给程序引入一些错误，特别是对于并发程序。这种改写的不足之处表现在：第一，对程序员的要求较高，即要求程序员对每种同步机制都非常地熟悉；第二，浪费了宝贵的时间，程序员需要处理那些与业务逻辑不相关的内容，不能集中精力处理核心业务逻辑；第三，手动的改写容易给程序引入新的错误，破坏了程序的可维护性；第四，不同锁机制的性能受多种因素影响，难于抉择。此外，有些程序员通过借助已有经验的方式来选择锁，但是这种选择方式缺乏灵活性，一旦环境改变，选择的锁机制则不一定适用。因此，迫切需要对锁机制的推荐方法进行研究，并提供相应的算法支持自动重构。

近年来，利用大规模代码资源中蕴涵的众多知识进行智能化软件开发具有非常重要的理论与应用价值。Alon等[11]提出了一种将代码片段表示为连续分布式向量的神经网络模型，其主要思想是将代码片段表示为一个固定长度的代码向量，它可以用来预测代码片段的语义属性。传统的基于信息检索的方法往往将程序视为自然语言文本，这可能会错过源代码的重要语义信息，所以Zhang等[12]提出了一种新的基于抽象语法树的神经网络用于源代码表示，基于语句向量的序列，采用双向循环神经网络模型，利用语句的自然性，最终产生代码片段的向量表示。Raychev等[13]提出了一种新的方法来预测从大代码库中得到的程序属性，其从“大代码”中学习一个概率模型，并使用这个模型来预测新的、看不见的程序的属性。Madeiral等[14]提出了一个收集和存储bug到一个可扩展的bug基准中的项目，从GitHub上托管的开源项目中找到潜在的缺陷和修补程序版本对，用于Java中的自动修复研究。Saha等[15]提供了一个jar，用于研究Java程序自动调试、补丁和测试的大型数据集，其由1 158个错误和补丁组成，这些错误和补丁来自8个大型流行的开源Java项目，涵盖8个不同的著名应用程序类别。当前对于锁机制使用的代码推荐的相关研究尚有欠缺，目前大部分工作也是关注于使用相似代码来给予推荐，对于锁机制的使用无法在程序开发初期就能给出相应的推荐。

2 不同锁机制性能对比

本章给出了同步锁、可重入锁、读写锁、邮戳锁的性能分析结果，这里使用ArrayList、HashMap、LinkedList、TreeSet四种线程不安全的数据结构，使用多个线程对其进行读写操作，利用不同的锁机制对其读写操作进行保护，将其变为线程安全的操作。这里选择使用的数据结构、执行线程数、读写线程比例以及执行次数等四种变量进行执行时间的分析，其中每个线程将执行500次的读操作或写操作，所有实验数据是在执行5次的基础上取平均值后得到的。图1给出了四种数据结构使用四种锁机制（其中邮戳锁分为优化读锁形式与未优化的形式）的性能测试结果，在实验中分别选择10和100线程来执行读写操作，同时选取线程总数10%的作为读线程或写线程的数量（R表示读线程数，W表示写线程数）。

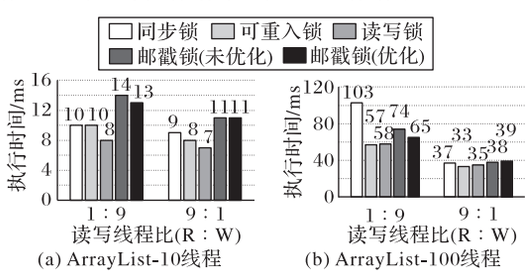


图1 不同ArrayList线程的性能对比

Fig. 1 Performance comparison of different ArrayList threads

如图1(a)所示，ArrayList使用四种锁时，在总线程为10且读写线程比为1∶9时，使用读写锁的时间最小，同步锁与可重入锁时间一样，两种邮戳锁的时间最多，其中优化读锁的要小于未优化读锁的时间；读写线程比为9∶1时，执行时间均有所减少，使用读写锁的时间仍是最小的，使用两种邮戳锁的时间仍较多。如图1(b)所示，在总线程为100且读写线程比为1∶9时，使用可重入锁与读写锁的时间最少，同步锁时间最多，使用两种邮戳锁的时间多于可重入锁，但少于同步锁的，其中优化读锁的要小于未优化读锁的时间；读写线程比为9∶1时，执行时间均有所减少，不同锁的执行时间相近，使用可重入锁与读写锁的时间仍要少于其他锁。

HashMap在总线程为10且读写线程比为1∶9时（如图2(a)所示），不同锁的执行时间相近；读写线程比为9∶1时，使用同步锁与可重入锁的执行时间均有所减少，其他锁执行时间没有明显变化，使用同步锁的时间最少。在总线程为100且读写线程比为1∶9时（如图2(b)所示），使用同步锁时间

最多,其他锁的时间则相近,但使用两种邮戳锁的时间要多于可重入锁跟读写锁,其中优化读锁的要大于未优化读锁的时间;读写线程比为9:1时,使用同步锁的时间仍是最多的,可重入锁与邮戳锁的执行时间都有所减少,使用读写锁的时间变化不大。

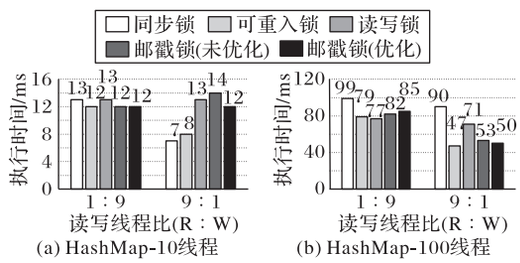


图2　不同HashMap线程的性能比较

Fig. 2　Performance comparison of different HashMap threads

当考虑LinkedList数据结构在使用四种锁时,当总线程为10且读写线程比为1:9时(如图3(a)所示),不同锁的执行时间比较接近;读写线程比为9:1时,执行时间均有所增加,使用同步锁与可重入锁的时间是最大的,使用两种邮戳锁的时间是最小的,其中优化读锁的要小于未优化读锁的时间。在总线程为100且读写线程比为1:9时(如图3(b)所示),不同锁的执行时间相近,使用同步锁的要小于其他锁的时间;读写线程比为9:1时,执行时间均有所增加,使用读写锁的时间是最小的可重入锁与两种邮戳锁的时间相近,都大于使用同步锁的时间。

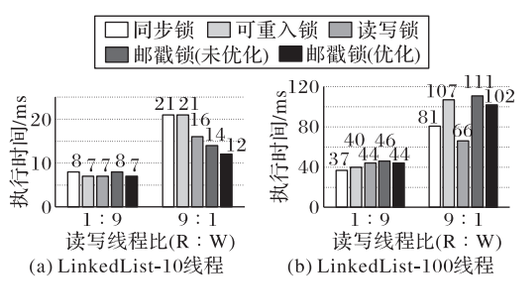


图3　不同LinkedList线程的性能比较

Fig. 3　Performance comparison of different LinkedList threads

对于TreeSet,在总线程为10且读写线程比为1:9时(如图4(a)所示),使用邮戳锁的时间最少,其中优化读锁的时间要小于未优化读锁的时间,读写锁的时间最少,可重入锁的时间较多;读写线程比为9:1时,执行时间均有所减少,使用两种邮戳的时间仍是最小的,其他锁的时间相近。在总线程为100且读写线程比为1:9时(如图4(b)所示),使用读写锁的时间最少,使用同步锁与可重入锁的时间最多,使用两种邮戳锁的时间多于读写锁,但少于另外两种锁;读写线程比为9:1时,执行时间均有所减少,使用读写锁的时间仍要少于其他锁,使用同步锁与可重入锁的时间仍较多。

综合上述分析可以得出以下结论:

1)不同线程总数和不同读写线程的比例对使用锁的应用程序的性能有较大影响;

2)对于使用同一个数据结构的应用程序,使用不同锁的程序性能往往不同;

3)即使每次运行的线程数量相同,不同数据结构的锁应用程序性能也不同;

4)一个锁机制并不是绝对优于另一个锁机制,手动的选择比较困难。

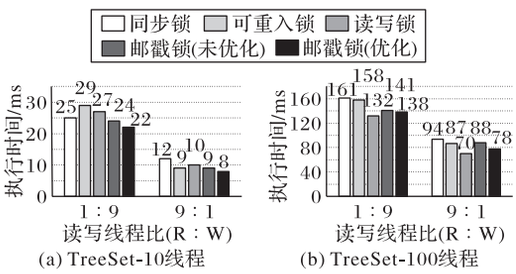


图4　不同TreeSet线程的性能比较

Fig. 4　Performance comparison of different TreeSet threads

通过对性能结果的分析,发现对锁结构进行决策是一项困难的任务,并且其影响因素有很多,单纯依靠经验判断很难抉择,因此迫切需要提供一种方法实现不同锁机制的推荐,帮助程序员了解程序使用哪一种锁性能较好。

3　面向不同锁机制的推荐方法

本章首先给出面向不同锁机制的推荐框架,然后对推荐系统的各个实现部分进行详细描述。

3.1　推荐框架

在推荐框架中,首先对使用程序静态分析工具对锁的并发程序信息进行收集,抽取如数据结构、线程数等程序相关的特征属性值;其次将得到的样本数据经过抽样[16]算法处理后,输入到随机森林模型中,进行模型训练;最后分析验证模型的适用性以及合理性,从而得到锁机制推荐模型。当有一个待推荐的并发程序时,收集其相关特征属性并输入到锁机制推荐模型中,输出推荐使用的锁机制。锁机制的推荐框架如图5所示。

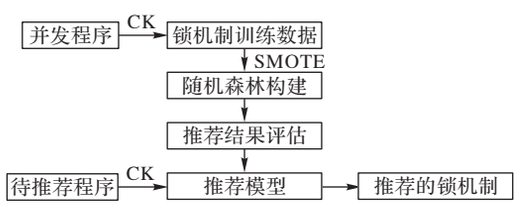


图5　推荐框架

Fig. 5　Recommendation framework

3.2　特征选择与数据处理

特征选择是从原始特征中根据一定的评估准则剔除一些不相关特征而保留一些最有效特征的过程,且在特征选择后分类正确率比选择前更高或近似。在锁机制推荐的模型中,本文首先使用CK(https://github.com/mauricioaniche/ck)工具收集58个程序相关的特征向量,其中包括方法级的参数(如:使用的数据结构、线程数、方法权重等)以及类参数(如:同步字段数量、同步方法数量等)。在多数情况下,由于选取的初始特征向量较多,部分特征的数值是缺失的,存在数据不平衡的问题,所以会使少数类的分类器的性能下降。由于初始选取的特征向量较多,会导致多数特征值存在缺失,就会造成训练样本成为少数类样本,影响模型的泛化能力,产生过拟合现象,所以这里采用合成少数过采样技术(Synthetic Minority Oversampling Technique, SMOTE)工具,使用过采样方法,利用少类完全样本生成新样本来提升少数类完全样本

数量，从而获得有效的分类模型。锁机制部分特征属性说明 如表1所示。

表1 特征属性说明

Tab. 1 Feature attribute description

字段	说明
Structure	加锁方法数据结构，仅对四种数据结构进行标注，其他情况标记为0
NumOfThread	线程数，获取实际程序中显式标注，其他情况标记为0
methodWmc	方法权重，McCabe环路复杂度
classNumberOfSynchronizedFields	使用Synchronized关键字的代码块
classNumberOfSynchronized	使用Synchronized关键字的方法

为了更直观地观察不同特征属性对于锁机制选择的重要性区分，通过选取五个相关性最高的属性进行热力图绘制，如图6所示。两个属性间的相关性使用皮尔逊相关系数进行计算，该系数反映了两个变量线性相关程度的统计量[17]。可以看到对于锁机制(lockid)的选择，使用的数据结构(Structure)、线程数(NumOfThread)、操作数(NumOfOperate)、方法权重(methodWmc)等属性与其之间的相关性数值分别是0.641、0.612、0.499、0.69，根据这些相关性将使用改进的随机森林分类算法对数据进行分析。

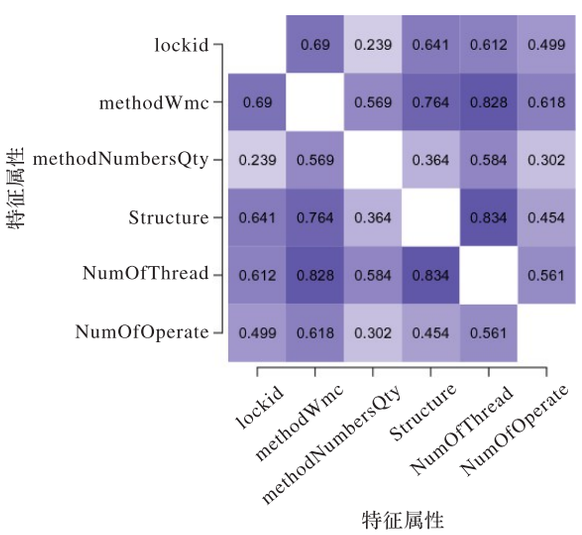


图6 特征属性相关性热力图

Fig. 6 Heat map of feature attribute correlation

3.3 随机森林推荐算法

考虑到对于不同锁机制的选择时影响因素较多，所以需要选取一个合适的推荐模型进行推荐。随机森林算法能够处理很高维度的数据，并且在训练数据较多的情况时，训练速度较快。尽管随机森林具有很多优点，但仍然有不足之处，传统随机森林模型中的决策树拥有相同的投票权重，而且随着随机森林构建的扩大，决策树生成过于庞大的子叶，使得实验预测结果过拟合，这影响了其整体分类能力的可靠性，由于每棵决策树在投票时权重都相等，并且存在生成过于庞大的子叶现象，会影响随机森林的整体分类能力[18]。

为了解决上述问题，采用基于错误的剪枝法和Kappa系数加权随机森林分类方法对锁机制的选择进行了推荐分析，其中使用剪枝操作主要考虑到锁机制数据数量较大，进而造成的决策树过于庞大的问题；使用加权操作主要考虑影响锁机制选择的特征属性较多，相同权重的决策树在分类效果上较低，不能保证最终结果的准确度。

由于决策树采用自上而下、分而治之的学习策略，随着迭代深度增加，节点所使用的训练数据不断减少，其对总体数据的代表性也会相对降低，导致无法对新数据进行合理的分析预测，即“过度拟合”现象[19]。决策树剪枝是解决上述问题的一个有效途径，其利用某种评价标准对决策树进行简化，减小规模，提高决策树的泛化能力。本文选取后剪枝操作[20]，后剪枝是指构造出完整的决策树之后再去查看哪些子树可以剪掉，从而能够对未知数据有更准确的预测。

由于每棵树训练集不同，又互相独立训练，所以决策树的分类精度会不同，加之受到样本类别不平衡的影响，会进一步影响决策树的分类效果，使得一些效果不好的树投出错误的票数，从而影响随机森林的分类能力。在随机森林元分类器训练的过程中，会随机地抽取总训练样本中的部分数据对单个决策树进行训练，而未被抽取的数据称为袋外(Out-Of-Bag, OOB)数据[21]。为了把较大的权重值分配给效果更好的分类器，本文采用OOB估计计算分类器中的元分类器的投票权重，利用引入Kappa系数[22]对权重值进行进一步的计算，加权随机森林算法如算法1所示。利用袋外数据对决策树进行的预测能力的评估就成为OOB估计，在随机森林构建的同时，每一个决策树都会计算一个对应的OOB数值。通过CK评价每棵决策树整体分类效果，并根据CK给每棵决策树分配投票权重，减少分类效果不好的决策树对结果的影响，使算法的输出结果更加合理，提高对数据集的整体分类性能。Kappa系数的计算方法如下所示：

$$CK = (p_0 - p_l)/(1 - p_l)$$

其中：p_0表示的是总分类准确度；p_l表示某一类锁机制的准确度，$p_l = \dfrac{a_1 \times b_1 + a_2 \times b_2 + \cdots + a_n \times b_n}{n \times n}$($a_i$表示第$i$类真实样本个数，$b_i$表示第$i$类预测出来的样本个数)。

算法1 结合Kappa系数的加权随机森林算法。

输入 训练数据D，全部特征T，决策树个数n，随机森林模型RF；

输出 加权的随机森林模型。

1)遍历RF中所有决策树。

2)将数据集分为袋内数据和袋外数据。

3)计算整体$OOB(D)$；选取特征变量i，随机排序，计算$OOB(D_i)$。

4)针对所有特征变量，重复步骤2)。

5)计算权重值$W(i)$为每一个特征变量OOB的值减去整体OOB值的差乘以每棵决策树的CK值。

6)将带有权重的决策树更新到RF输出。

在算法1中，$W(i)$越高，说明变量i越重要；如果$W(i) = 0$，那么说明变量i重要性不大；如果$W(i)<0$，那么说明变量i有很明显的噪声，对模型产生了负面影响。利用特征重要性度量值作为特征排序的重要依据，依次从特征中去掉一个重要性最低的特征，并计算每次剔除后的分类正确率，选取

最高正确率对应的特征变量作为最优特征选择结果[23]。

在实现LockRec时,设置决策树数量、决策树最大深度、分类数量以及树的大小等数值分别为50、50、4、50,在剪枝算法中,设置的置信水平值越高则剪去的分支越少;值越低,剪去的分支越多,对于置信水平,通常将其值设置为0.25[24]。通过计算待剪枝决策树T的所有叶节点的分类错误率上限,当前决策树某叶节点对n个训练样本进行分类训练,并且计算其对某个样本分类错误的概率,从而求得该节点的分类错误率估计值,依据算法描述中的剪枝方法进行剪枝操作。在加权算法中,通过计算待加权的决策树对袋外数据的分类效果,并且计算当前决策树的CK系数值,将两者的乘积作为当前决策树的权值。针对同步锁、可重入锁、读写锁、邮戳锁等四种锁,使用上述模型构建一个锁机制的推荐模型,当有一个新的待推荐并发程序时,首先获取其对应上述两种模型的特征属性值,将待推荐数据进行10次模型分类,统计这10次中出现的推荐锁机制,将次数最多的锁机制作为最终锁机制推荐的结果。

4　实验与结果分析

4.1　实验平台及实验数据说明

本文所有实验在惠普Z820工作站上完成的,硬件上,CPU为Intel Xeon E5-2650 2.60 GHz两个处理器,每个CPU有8个处理核且均支持超线程,可以支持32线程同时运行,内存为128 GB。软件上,使用了Deepin Linux操作系统和软件工具Eclipse 4.11,JDK版本为1.8.0_211。

为了验证剪枝以及加权随机森林的有效性,选取锁机制数据集(lock)和另外4个不同的数据集(来自UCI的公开数据集,http://archive.ics.uci.edu/ml/datasets),进行了模型准确率、召回率等指标的验证。数据集有关信息如表2所示。

表2　实验中使用的数据集

Tab. 2　UCI datasets used in experiments

数据集	类数	实例数	维数	训练测试集比
glass	7	213	9	4:1
wine	3	178	13	4:1
iris	3	150	4	4:1
breast	2	570	30	4:1
lock	4	96 000	58	4:1

对于LockRec中所使用的决策树参数以及模型相关的评价指标如下:

1)决策树有关参数。分别统计传统随机森林与改进的两种随机森林模型,决策树个数在5~100内变化时模型的准确率,这里对应的最大深度与决策树大小与当前决策树的个数相同。

2)运行次数。对于同一个数据集,在同一决策树个数的情况下,重复构建5次模型,并得到当前情况下的准确率的平均值。

3)评价指标。准确率(Precision,P)、召回率(Recall,R)、平均绝对误差(Mean Absolute Error, MAE)和均方根误差(Root Mean Squared Error, RMSE)的计算式如下:

$$P = TP/(TP + FP)$$

$$R = TP/(TP + FN)$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\hat{y}_i - y_i\right|$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}_i - y_i\right)^2}$$

其中:真阳性(True Positive, TP)表示样本的真实类别是正例,并且模型预测的结果也是正例;真阴性(True Negative, TN)表示样本的真实类别是负例,并且模型将其预测成为负例;假阳性(False Positive, FP)表示样本的真实类别是负例,但是模型将其预测成为正例;假阴性(False Negative, FN)表示样本的真实类别是正例,但是模型将其预测成为负例;$\hat{y}_i - y_i$为预测值减去真实值。MAE值越小表示推荐系统的推荐精度越高,同样RMSE值越小也表示推荐系统的推荐精度越高。

4.2　算法适用性验证

为了验证所提出的模型适用性,本文使用了iris、breast、wine和glass四个数据集分别对不进行剪枝与加权操作的传统模型和LockRec模型进行准确率验证。

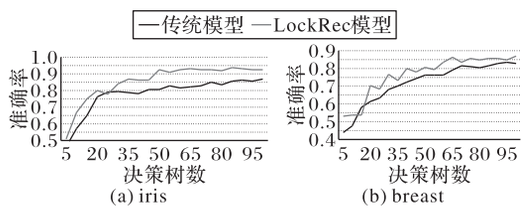


图7　iris和breast数据集上不同决策树个数的模型准确率对比

Fig. 7　Accuracy comparison of models with different numbers of decision trees on iris and breast datasets

从图7的对比可以看出,对于iris和breast数据集,所提出的LockRec模型在决策树数量较少时波动大,预测准确率有时相较传统模型要较小,随着决策树个数的增加,准确率稳定提高,预测准确率均优于传统模型,并且对iris数据集LockRec的准确率最后较为稳定,在92%左右,对breast数据集则为86%左右。

在图8中,对于wine数据集,所提出的LockRec模型在决策树数量较少时与传统模型预测准确率几近一致,随着决策树个数的增加,两个模型准确率均稳步提高,对wine数据集LockRec的准确率最后较为稳定在98%左右;对于glass数据集,LockRec的准确率一直高于传统模型,并且在决策树数量为30时,就达到了稳定,准确率为99%。

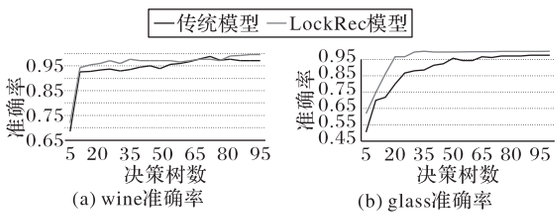


图8　wine和glass数据集上不同决策树个数的模型准确率对比

Fig. 8　Accuray comparison of models with different numbers of decision trees on wine and glass datasets

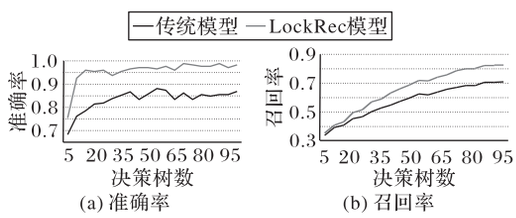
4.3　算法合理性验证

为了验证提出的模型对于锁机制数据的推荐合理性,对于锁机制的数据集,实验还选取推荐系统中常用的如召回率、平均绝对误差(MAE)、均方根误差(RMSE)等评价指标对LockRec进行了进一步的验证。

图9是针对锁机制数据集的两种模型的对比,可以看到,

传统模型在决策树个数较少时其准确率不高，随着个数的增加，其准确率稳定在85%左右。对比准确率，LockRec模型即使在决策树个数较小时也呈现出了比较好的效果，并且随着决策树个数的增加，LockRec模型的准确率很快稳定95%以上。对比召回率，在决策树个数较少时，两者召回率相近，均随着决策树个数的增加两种模型均趋于直线上涨，两个模型召回率最后分别稳定在82%和70%，说明LockRec模型要更优于传统模型。

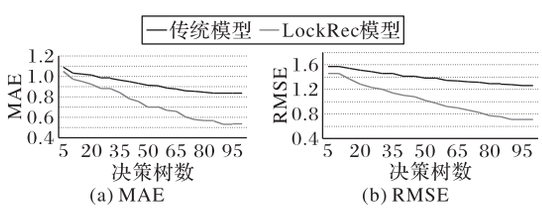
由图10可以看出，无论MAE值还是RMSE值，随着决策树数量的增加，传统模型与LockRec模型都呈现直线下降的趋势，并且LockRec模型的下降幅度要大于传统模型。两个模型MAE最后分别稳定在0.53和0.83，RMSE分别稳定在0.7和1.2。LockRec模型的MAE值和RMSE值都更小，表明LockRec模型的预测精度要更高。



(a) 准确率 (b) 召回率

图9 不同决策树个数的针对锁机制数据集的两种模型的准确率及召回率

Fig. 9 Accuracy and recall rate of two models with different numbers of decision trees for lock mechanism dataset



(a) MAE (b) RMSE

图10 不同决策树个数的针对锁机制数据集的两种模型的MAE值及RMSE值

Fig. 10 MAE and RMSE of two models with different numbers of decision trees for lock mechanism dataset

为了验证模型对其他非Java语言的锁机制的适用性，选取了Python语言中的锁机制进行了程序信息收集，并使用LockRec模型对其进行训练预测，得到结果如图11所示。从图11的对比可以看出，对于Python的锁机制数据集，所提出的LockRec模型在决策树数量较少时与传统模型准确率相近，在决策树数量为25~35时预测准确率相比传统模型要较小，但随着决策树个数的增加，在决策树数量35个之后，其准确率稳定提高，预测准确率均优于传统模型。

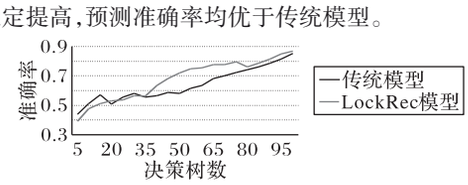


图11 对于Python的锁机制数据集不同决策树个数的模型准确率对比

Fig. 11 Accuracy comparison of models with different numbers of decision trees for lock mechanism dataset of Python

综上所述，本文所提出的LockRec模型在对锁机制数据集进行分析预测时有着比其他数据集更好的效果，并且所得到数值结果表明LockRec模型不论是在准确率还是其他指标上，都要优于传统模型。

4.4 锁机制推荐模型验证

使用LockRec对Xalan(http://xalan.apache.org/xalan-j/)、Fop(https://xmlgraphics.apache.org/fop/)进行锁机制使用推荐，其中Xalan和Fop分别有51个和25个使用锁的方法。使用LockRec进行锁机制推荐结果如图12所示。

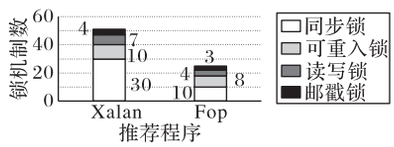
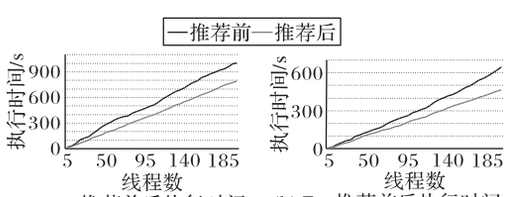


图12 LockRec推荐结果

Fig. 12 Recommendation results of LockRec

可以看到，对于Xalan使用LockRec之后，推荐仍使用同步锁的有30个，推荐使用可重入锁的有10个、使用读写锁的7个、使用邮戳锁的有4个；对于Fop中原始使用同步锁的25个方法，使用LockRec之后，推荐仍使用同步锁的有10个，推荐使用可重入锁的有8个、使用读写锁的4个、使用邮戳锁的有3个。根据上述推荐结果，本文对Xalan、Fop两个程序进行了手动重构，并分析使用原锁机制与推荐锁机制的前后程序执行时间。这里使用不同个数的线程去执行重构前后的程序，每个线程数下均执行五次，最后取五次执行时间的平均值作为当前线程数的执行时间。

如图13(a)所示，对于Xalan来说，使用LockRec推荐使用的锁机制后，在不同执行线程数时，程序整体执行时间均有所缩短。程序整体平均缩短时间为127.65 s，当执行线程数为195时，缩短时间最多为222 s，平均加速比为1.35；如图13(b)所示，对于Fop来说，使用LockRec推荐使用的锁机制后，在不同执行线程数时，程序整体执行时间均有所缩短。程序整体平均缩短时间为70.75 s，当执行线程数为200时，缩短时间最多为180 s，平均加速比为1.29。



(a) Xalan推荐前后执行时间 (b) Fop推荐前后执行时间

图13 LockRec推荐前后性能对比

Fig. 13 Comparison of performance before and after LockRec recommendation

4.5 有效性威胁

实验中有几个可能威胁有效性的因素：1)由于并发程序执行的不确定性，不排除性能测试实验结果可能存在细微偏差。为了避免误差，所有实验结果都是在5次运行的基础上取平均值得到的。2)随机森林模型的构建依赖测试数据的好坏，为了验证模型的适用性，本文选取多个著名数据集对所提出的改进随机森林模型进行了验证。3)本实验只选取了2个真实应用程序对LockRec进行了评估，它们并不能代表所有程序，不能完全说明LockRec在所有的应用程序中可以进行效果良好的推荐，未来的工作中也将使用更多应用程序对LockRec进行测试。

5　结语

当程序员开发并发程序时,锁是最常使用的一种同步机制,而使用哪个锁作为最佳选择是一个大问题,因为使用不同锁构造的性能会有很大的不同。本文实现了一个锁机制使用推荐的方法 LockRec,可以帮助程序开发者选出当前程序代码的最合适的锁机制进行使用。在设计和实现 LockRec 的过程中,使用两种改进的随机森林模型,并且结合实际测试以及真实程序进一步确保最终推荐结果的可信程度。实验结果表明,LockRec 的推荐效率较为高效,模型推荐准确率平均值可达 95.1%,使用推荐后的锁机制,两个真实程序性能均有所提高,平均加速比分别为 1.35、1.29。在未来的工作中,将针对推荐方法进行进一步的研究,使用可以体现程序间关系的代码推荐方法来提高锁机制推荐使用的适用性。

参考文献 (References)

[1] SCHÄFER M, DOLBY J, SRIDHARAN M, et al. Correct refactoring of concurrent java code [C]// Proceedings of the 2010 24th European Conference of Object-Oriented Programming, LNCS 6183. Berlin: Springer, 2010: 225-249.

[2] ZHANG Y, SHAO S, LIU H, et al. Refactoring Java programs for customizable locks based on bytecode transformation [J]. IEEE Access, 2019, 7: 66292-66303.

[3] ZHANG Y, DONG S, ZHANG X, et al. Automated refactoring for StampedLock [J]. IEEE Access, 2019, 7: 104900-104911.

[4] YU T, PRADEL M. SyncProf: detecting, localizing, and optimizing synchronization bottlenecks [C]// Proceedings of the 2016 25th International Symposium on Software Testing and Analysis. New York: ACM, 2016: 389-400.

[5] SCHÄFER M, SRIDHARAN M, DOLBY J, et al. Refactoring Java programs for flexible locking [C]// Proceedings of the 2011 33rd International Conference on Software Engineering. New York: ACM, 2011: 71-80.

[6] ROSE J, SWAMY N, HICKS M. Dynamic inference of polymorphic lock types [J]. Science of Computer Programming, 2005, 58(3): 366-383.

[7] BAUM M V. Refactoring for software transactional memory [D]. Austin: University of Texas at Austin, 2011: 12-14.

[8] ERNST M D, LOVATO A, MACEDONIO D, et al. Locking discipline inference and checking [C]// Proceedings of the 2016 IEEE/ACM 38th International Conference on Software Engineering. Piscataway: IEEE, 2016: 1133-1144.

[9] ERNST M D, MACEDONIO D, MERRO M, et al. Semantics for locking specifications [C]// Proceedings of the 2016 8th NASA Formal Methods Symposium, LNCS 9690. Cham: Springer, 2016: 355-372.

[10] MCCLOSKEY B, ZHOU F, GAY D, et al. Autolocker: synchronization inference for atomic sections [C]// Proceedings of the 2006 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York: ACM, 2006: 346-358.

[11] ALON U, ZILBERSTEIN M, LEVY O, et al. Code2vec: learning distributed representations of code [J]. Proceedings of the ACM on Programming Languages, 2019, 3(POPL): Article No. 40.

[12] ZHANG J, WANG X, ZHANG H, et al. A novel neural source code representation based on abstract syntax tree [C]// Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering. Piscataway: IEEE, 2019: 783-794.

[13] RAYCHEV V, VECHEV M, KRAUSE A. Predicting program properties from "Big Code" [J]. Communications of the ACM, 2019, 62(3): 99-107.

[14] MADEIRAL F, URLI S, MAIA M, et al. BEARS: an extensible Java bug benchmark for automatic program repair studies [C]// Proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering. Piscataway: IEEE, 2019: 468-478.

[15] SAHA R K, LYU Y, LAM W, et al. Bugs. jar: a large-scale, diverse dataset of real-world Java bugs [C]// Proceedings of the 2018 ACM/IEEE 15th International Conference on Mining Software Repositories. New York: ACM, 2018: 10-13.

[16] DOUZAS G, BACAO F. Geometric SMOTE a geometrically enhanced drop-in replacement for SMOTE [J]. Information Sciences, 2019, 501: 118-135.

[17] MU Y, LIU X, WANG L. A Pearson's correlation coefficient based decision tree and its parallel implementation [J]. Information Sciences, 2018, 435: 40-58.

[18] LI H B, WANG W, DING H W, et al. Trees weighting random forest method for classifying high-dimensional noisy data [C]// Proceedings of the 2010 IEEE 7th International Conference on E-Business Engineering. Piscataway: IEEE, 2010: 160-163.

[19] 张家伟,郭林明,杨晓梅. 针对不平衡数据的过采样和随机森林改进算法[J]. 计算机工程与应用, 2020, 56(11): 39-45. (ZHANG J W, GUO L M, YANG X M. Improved oversampling and random forest algorithm for imbalanced data [J]. Computer Engineering and Applications, 2020, 56(11): 39-45.)

[20] 郑伟,马楠. 一种改进的决策树后剪枝算法[J]. 计算机与数字工程, 2015, 43(6): 960-966, 971. (ZHENG W, MA N. An Improved post-pruning algorithm for decision tree [J]. Computer and Digital Engineering, 2015, 43(6): 960-966, 971.)

[21] ABDOH A F, ABO RIZKA M, MAGHRABY F A. Cervical cancer diagnosis using random forest classifier with SMOTE and feature reduction techniques [J]. IEEE Access, 2018, 6: 59475-59485.

[22] TANG W, HU J, ZHANG H, et al. Kappa coefficient: a popular measure of rater agreement [J]. Shanghai Archives of Psychiatry, 2015, 27(1): 62-67.

[23] 徐少成,李东喜. 基于随机森林的加权特征选择算法[J]. 统计与决策, 2018, 34(18): 25-28. (XU S C, LI D X. Weighted feature selection algorithm based on random forest [J]. Statistics and Decision, 2018, 34(18): 25-28.)

[24] 蒋艳凰,赵强利. 机器学习方法[M]. 北京:电子工业出版社, 2009: 56. (JIANG Y H, ZHAO Q L. Machine Learning Techniques [M]. Beijing: Publishing House of Electronics Industry, 2009:56.)

This work is partially supported by the National Natural Science Foundation of China (61440012), the Key Project of Hebei Education Department Scientific Research Planning (ZD2019093), the Key Basic Project of Hebei Fundamental Research Planning (18960106D).

ZHANG Yang, born in 1980, Ph. D., associate professor. His research interests include intelligent software, software refactoring.

DONG Shicheng, born in 1995, M. S. candidate. His research interests include intelligent software.