基于随机森林的流处理检查点性能预测

褚 征 于 炯*

(新疆大学信息科学与工程学院 乌鲁木齐 830046)

摘 要：物联网(IoT)的发展引起流数据在数据量和数据类型两方面不断增长。由于实时处理场景的不断增加和基于经验知识的配置策略存在缺陷，流处理检查点配置策略面临着巨大的挑战，如费事费力，易导致系统异常等。为解决这些挑战，该文提出基于回归算法的检查点性能预测方法。该方法首先分析了影响检查点性能的6种特征，然后将训练集的特征向量输入到随机森林回归算法中进行训练，最后，使用训练好的算法对测试数据集进行预测。实验结果表明，与其它机器学习算法相比，随机森林回归算法在CPU密集型基准测试，内存密集型基准测试和网络密集型基准测试上针对检查点性能的预测具有误差低，准确率高和运行高效的优点。

关键词：流处理；预测方法；检查点性能；随机森林；回归算法

中图分类号：TN919; TP311 **文献标识码**：A **文章编号**：1009-5896(2020)06-1452-08

DOI: 10.11999/JEIT190552

Performance Prediction Based on Random Forest for the Stream Processing Checkpoint

CHU Zheng YU Jiong

(*School of Information Science and Engineering*, *Xinjiang University*, *Urumqi* 830046, *China*)

Abstract: Since real-time processing scenarios for ever-increasing amount and type of streaming data caused by the development of the Internet of Things (IoT) keep increasing, and strategies based on empirical knowledge for checkpoint configuration are deficiencies, the strategy faces huge challenges, such as time-consuming, labor-intensive, causing system anomalies, etc. To address these challenges, regression algorithm-based prediction is proposed for checkpoint performance. Firstly, six kinds of features, which have a huge influence on the performance, are analyzed, and then feature vectors of the training set are input into the regression algorithms for training, finally, test sets are used for the checkpoint performance prediction. Compared with other machine learning algorithms, the experimental results illustrat that the Random Forest (RF) has lower errors, higher accuracy and faster execution on CPU intensive benchmark, memory intensive benchmark and network intensive benchmark.

Key words: Stream processing; Prediction method; Checkpoint Performance; Random Forest (RF); Regression algorithm

1 引言

随着智能设备、车载设备、无人驾驶汽车等物联网终端的持续增加[1,2]，流数据在数量和种类两方面呈现出爆炸式的增长。数据的不断增长趋势已使数据分析方式从静态数据处理转变为流数据实时处理。另一方面，用户和各大公司也将更多的注意力投入到实时分析场景中。来自多方的驱动力使得流处理得到大力的发展，如Apache Flink[3]等流处理系统的大量出现。在Apache Flink中，既要保证低延迟，又要实时地处理源源不断流入系统内部的数据流是一个巨大的挑战，特别是像分布式系统还要提供容错机制，这完全得益于流处理的检查点技术[4]为容错提供了简单、高效的保障。

目前，针对检查点的研究主要集中在检查点性能优化[5]和检查点应用[6–9]两方面。在性能优化方面，使用有效的局部快照算法进行低频率协调[5]以提高检查点的性能。在检查点应用方面，使用检查点优化虚拟机放置策略以节省成本[6]，使用检查点进行早期的硬件木马检测[7]，使用多级容器检查点性能

收稿日期：2019-07-23；改回日期：2020-02-17；网络出版：2020-03-10
*通信作者： 于炯 yujiong@xju.edu.cn
基金项目：国家自然科学基金(61862060, 61462079, 61562086, 61562078)，新疆大学博士生科技创新项目(XJUBSCX-201901)
Foundation Items: The National Natural Science Foundation of China (61862060, 61462079, 61562086, 61562078), The Doctoral Science, Technology Innovation Project in Xinjiang University (XJUBSCX-201901)

优化策略提高大规模云应用程序的性能[8]，使用检查点技术对无线传感器网络的簇头进行恢复[9]等。

然而，当前缺少针对流处理检查点性能预测的研究工作。流处理的检查点策略需要由用户进行配置，这一方面增加了用户的操作负担，另一方面也会增加检查点策略配置不合理而引发系统异常的概率。如检查点并发数量配置不合理等问题。这些问题都极易引起系统异常，甚至增加系统的异常恢复成本而导致系统不可用时间增加，本文第2节(动机)将详细描述当前配置检查点策略不合理情况的示例。

本文的研究目标是利用高效的机器学习算法对流处理检查点性能做出准确的预测，降低系统用户的负担和用户配置检查点策略不合理的概率，从而使用户摆脱“尝试-错误-修正”的循环模式。为了实现该目标，本文首先介绍背景知识，识别影响流处理检查点性能的重要特征。然后对比多个回归算法在性能预测中的表现来选择合适的算法，并将其应用于检查点性能预测任务中。最后通过实验对研究工作进行评估并对实验结果进行讨论。

2　动机

流处理的检查点配置主要有3个参数，分别是检查点周期、超时时间和检查点并发数量。其中，检查点周期是每隔多久进行一次检查点操作，超时时间是每次执行检查点操作的最大容忍时间，检查点并发数量是在同一时刻可以执行的最大检查点操作数量。若执行一次检查点操作超过了超时时间则该次检查点操作失败。若在当前时刻执行检查点的操作数量已经达到了最大的检查点并发数量则无法继续执行新的检查点操作。以下对基于经验的检查点策略所存在的问题进行相应的介绍。在描述的过程中，Duration of Checkpoint代表检查点的执行时间，Free代表没有检查点操作的时间，Checkpoint Interval代表检查点周期，Timeout代表检查点超时时间，横箭头代表时间，灰色矩形代表检查点执行成功，黑色矩形代表检查点执行失败。图1展示了3种检查点策略配置不合理的示例。

检查点执行时间与检查点周期配置不合理：如图1(a)所示，当检查点操作执行完毕后，下一次的检查点操作需要等待较长的时间才能够进行。若在第2次的检查点操作未执行之前，流处理集群中的节点发生故障则有可能导致数据丢失或者集群恢复时间过长的问题，从而导致系统不可用时间较长。在另一种情况下，当第1个检查点操作执行完毕之后，由于已经超过了第2个检查点的触发时间，导致只能等待到第3个检查点进行触发的时刻才能进行检查点操作。这同样会带来与节点故障的恢复时间增加和系统的不可用时间过长问题。

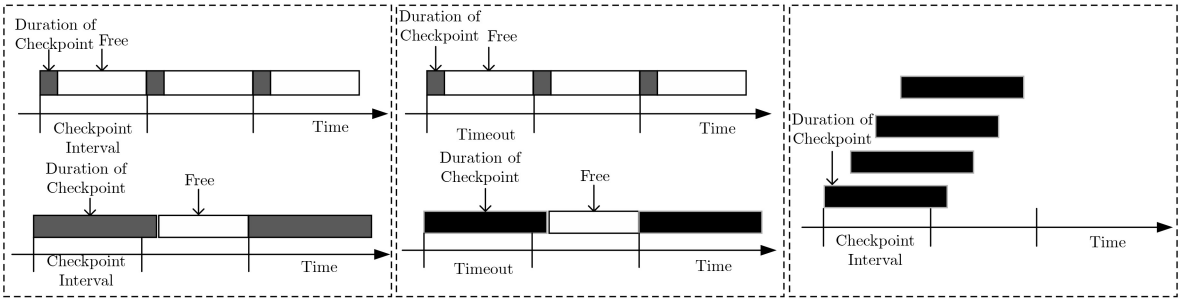
检查点执行时间与超时时间配置不合理：超时时间是为了防止检查点操作由于其它异常行为导致的长时间无法完成而影响集群的设置。图1(b)展示了超时时间小于执行时间的情况。由于执行时间已经超过了超时时间导致检查点操作失败。此种情况会导致流处理的容错功能失效，从而增加数据丢失的概率。另一种情况则是超时时间略小于执行时间，此种情况同样会增加数据丢失的概率。

检查点并发数量配置不合理：图1(c)中，由于检查点并发数量设置过大导致大量检查点操作失败。在同一时段内执行了大量的检查点操作而耗费了大量的可用资源导致大量的检查点操作失败甚至会引起系统的不稳定。这种情况加速了系统恶化的趋势，从而增加了数据丢失和系统不可用时间过长的概率。

在流处理检查点策略上存在检查点周期设置不合理，检查点超时时间设置不合理和检查点并发数量设置不合理的问题。归根结底，正是由于对检查点执行时间无法准确预测的原因导致的。

3　背景知识

流处理系统是一个对无界和有界数据流进行状态计算的流处理框架和分布式处理引擎,其设计目的是在所有常见的集群环境中运行，同时以内存速



(a) 检查点执行时间与检查点周期配置不合理　(b) 检查点执行时间与超时时间配置不合理　(c) 检查点并发数量配置不合理

图 1 检查点策略配置不合理示例

度和任意集群规模下执行并行计算的流处理系统。

流处理使用流重放和检查点技术的组合实现容错。检查点与每个输入流中的特定点以及每个操作符的对应状态相关。通过恢复算子的状态并从检查点重放事件，可以从检查点恢复数据流，同时保持一致性。通常流处理的容错技术是通过分布式快照[10]实现的，即在分布式集群中使用检查点技术。流应用程序的状态数据存储在可配置的流处理系统外部位置。检查点技术的目的是为了降低分布式集群发生错误后的恢复时间。通过系统用户定义的检查点触发时间，周期性的对运行在集群中的流处理任务进行快照操作，将状态信息等保存在流处理系统外部。这样就能够防止流处理系统崩溃而导致无法恢复的情况发生。

4 识别重要特征

4.1 CPU类特征

流处理系统持续不断地对来自系统外部的数据流进行计算，这些计算行为涵盖了各种各样的算法，如分词、流表查询、异常检测等。必不可少地需要利用CPU资源来执行相关的处理任务。因此，CPU负载等特征需要考虑在内。

当前的CPU负载，如式(1)所示

$$\mathrm{CPU_{load}}=\frac{\mathrm{CPU_{used}}}{\mathrm{CPU_{total}}} \tag{1}$$

式(1)中，$\mathrm{CPU_{used}}$是已使用CPU能力，$\mathrm{CPU_{total}}$是CPU总能力，通过相除获得CPU负载$\mathrm{CPU_{load}}$。

当前的CPU使用时间，CPU时钟周期乘以电子时钟脉冲周期，如式(2)所示

$$\mathrm{CPU_{time}}=\mathrm{CPU_{clock-cycle}}\times\mathrm{Clock_{period}} \tag{2}$$

式(2)中，$\mathrm{CPU_{clock-cycle}}$是CPU时钟周期，$\mathrm{Clock_{period}}$是电子脉冲时钟周期，两个数值的乘积获得CPU时间$\mathrm{CPU_{time}}$。

4.2 内存类特征

流处理利用内存进行实时计算来达到秒级、毫秒级甚至纳秒级的延迟目标，同时也影响着检查点的执行时间，因此，内存指标必然是影响检查点性能的重要特征

$$\mathrm{Heap_{usage}}=\frac{\mathrm{Heap_{used}}}{\mathrm{Heap_{max}}} \tag{3}$$

式(3)中，利用已使用的JVM堆内存$\mathrm{Heap_{used}}$除以JVM的最大堆内存$\mathrm{Heap_{max}}$获得JVM的堆内内存使用率$\mathrm{Heap_{usage}}$。

在流处理中，内存管理模块不仅会对JVM堆内内存进行申请还会对JVM的堆外内存进行申请和管理，因此，堆外内存也是重要的特征。

$$\mathrm{NonHeap_{usage}}=\frac{\mathrm{NonHeap_{used}}}{\mathrm{NonHeap_{max}}} \tag{4}$$

式(4)中的JVM堆外内存利用率与堆内内存利用率的计算方式一致。此外，缓冲池利用率和映射缓冲池利用率也是重要的内存特征。其计算公式基本类似，不再具体描述，如式(5)和式(6)所示

$$\mathrm{BufferPool_{usage}}=\frac{\mathrm{BufferPool_{used}}}{\mathrm{BufferPool_{total}}} \tag{5}$$

$$\mathrm{MappdeBufferPool_{usage}}=\frac{\mathrm{MappdeBufferPool_{used}}}{\mathrm{MappdeBufferPool_{total}}} \tag{6}$$

流处理任务是运行在Java虚拟机上的程序，因此Java虚拟机的垃圾回收行为必然会对程序产生一定的影响，垃圾回收时间如式(7)所示

$$\mathrm{GC_{time}}=\alpha+\beta+\gamma \tag{7}$$

式(7)中的垃圾回收时间$\mathrm{GC_{time}}$由年轻代垃圾回收时间α、老年代垃圾回收时间β和持久代垃圾回收时间γ相加而得。

运行在Java虚拟机上的程序另外一个特点是会受JVM类加载操作的影响，类加载率如式(8)所示

$$\mathrm{CL_{usage}}=\frac{\mathrm{CL_{loaded}}}{\mathrm{CL_{loaded}}+\mathrm{CL_{unloaded}}} \tag{8}$$

式(8)中，$\mathrm{CL_{loaded}}$代表已加载的类数量，$\mathrm{CL_{unloaded}}$代表未加载的类数量，$\mathrm{CL_{usage}}$代表类加载率。

4.3 网络类特征

在流处理中，大量的节点之间需要进行数据交换，这些网络特征需要考虑在内，算子输入池平均利用率和算子输出池平均利用率如式(9)和式(10)所示

$$\mathrm{InPool_{usage}}=\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathrm{InPool_{\it i}^{used}}}{\mathrm{InPool_{\it i}^{used}}+\mathrm{InPool_{\it i}^{unused}}}\right) \tag{9}$$

$$\mathrm{OutPool_{usage}}=\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mathrm{OutPool_{\it i}^{used}}}{\mathrm{OutPool_{\it i}^{used}}+\mathrm{OutPool_{\it i}^{unused}}}\right) \tag{10}$$

式(9)中，n代表算子数量，$\mathrm{InPool_{\it i}^{used}}$是一个算子的输入池已使用量，$\mathrm{InPool_{\it i}^{unused}}$是一个算子的输入池未使用量。通过计算每个算子的输入池利用率累加求和除以算子数量获得算子输入池平均利用率。算子的输出池平均利用率类似，不再具体描述，如式(10)所示。

4.4 状态数据类特征

状态数据是算子或者集群节点内部存储的数据，这些数据包含系统状态数据和用户定义的状态数据。如流处理任务的窗口缓冲状态数据是典型的系统状态数据，由算子执行相关算法所创建的数据

则是用户定义的状态数据，如分词过程需要统计键值数据等。这类状态数据需要及时的存储到系统外部，所以这类数据的大小直接影响检查点执行的性能，状态数据大小特征如式(11)所示

$$\mathrm{Size}_{\mathrm{statedata}}=\sum_{i-1}^{n}\left(\mathrm{Size}_{i}^{\mathrm{system-state}}+\mathrm{Size}_{i}^{\mathrm{user-state}}\right) \tag{11}$$

式(11)中，n代表算子数量，$\mathrm{Size}_{i}^{\mathrm{system-state}}$是一个算子的系统状态数据大小，$\mathrm{Size}_{i}^{\mathrm{user-state}}$是一个算子的用户状态数据大小，累加求和计算出状态数据总大小。

4.5 飞行数据类特征

飞行数据是从系统外部已经进入系统内部但还没有在系统内部处理完的数据。这类数据在执行检查点操作时需要进行记录，所以会对检查点性能产生影响，飞行数据类特征如式(12)所示

$$\mathrm{Quantity}_{\text{on-the-air}}=\sum_{i=1}^{n}q_i \tag{12}$$

式(12)中，n代表算子数量，q_i代表一个算子的飞行数据量。通过对每个算子飞行数据的累加获得总的飞行数据量特征。

4.6 动态特征

动态特征是一类随流处理任务变化而变化的特征。表1总结了动态特征信息。

5 性能预测

5.1 集成学习

随机森林[11]属于集成学习[12]算法中的一种，它的基学习器是决策树。其利用多个基学习器构建Bagging集成，同时又在决策树的训练过程中引入随机属性选择的特性，这使基学习器间的差异增加，从而使最终模型的泛化能力得到进一步的提升。随机森林回归算法使用Bootstrap抽样方法从原始的数据集合执行有放回地抽样操作，将抽样出来的原始数据子集构建一个决策树，最终将多个决策树构建成一个随机森林。这是一种基于统计学的理论，最终将构建成的决策树均值作为随机森林回归预测的结果。

表 1 动态特征总结

特征名称	描述
本地进入记录数	算子每秒接收的本地记录数。
远程进入记录数	算子每秒接收的远程记录数。
本地缓存记录数	算子每秒缓存的本地记录数。
远程缓存记录数	算子每秒缓存的远程记录数。

5.2 算法流程

图2展示了随机森林算法模型。森林中的每个决策树包含一个树状的决策节点序列，基于该序列，树分裂成各种分支，直到到达树的末端(叶子)。每个决策树的预测结果通过叶节点输出。最后，将多个决策树的输出结合在一起进行预测。随机森林算法具有训练速度快，避免过拟合的优点。森林中每棵树的生成过程分为以下4个步骤：

步骤 1　选择最佳分割值j和分割点s，求解(j,s)，其中y_i是样本的标签

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right] \tag{13}$$

步骤 2　使用选定的(j,s)将输入空间分成两部分R_1和R_2并计算相应的$\hat{c}_m$，即R_m中所有相应y_i的平均值

$$R_1(j,s)=\left\{x|\chi^{(j)}\le s\right\},\quad R_2(j,s)=\left\{x|\chi^{(j)}>s\right\} \tag{14}$$

$$\hat{c}_m=\frac{1}{N_m}\sum_{x_i\in R_m(j,s)}y_i,\quad x\in R_m,\quad m=1,2 \tag{15}$$

步骤 3　重复步骤1和步骤2，直到满足停止条件(通常为回归树的数量和树的深度)；

步骤 4　将输入空间分为$R_1, R_2, \cdots, R_m$并生成决策树

$$f(x)=\sum_{m=1}^{M}\hat{c}_m I\left(x\in R_m\right) \tag{16}$$

式(16)中，I是指标函数。然后，重新采样样本和特征以获取B个不同的样本并为每个样本生成一个回归树。最后，计算B个回归树的平均值作为最终的预测值

$$\hat{f}=\frac{1}{B}\sum_{b=1}^{B}f_b(x) \tag{17}$$

6 实验

6.1 实验方法

实验设置3组流处理任务基准测试，分别为

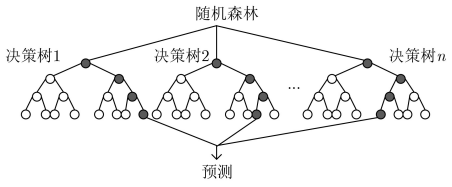


图 2 随机森林算法模型

CPU密集型(ChecKpoint CPU, CKCPU)、内存密集型(ChecKpoint MEMory, CKMEM)和网络密集型(ChecKpoint NETwork, CKNET)。将3组基准测试分别运行在一个本地的流处理系统Apache Flink集群中，通过一段时间的运行，使用程序主动采集本文所述与检查点性能相关的特征数据和检查点执行时间数据。将预处理后的样本数据输入随机森林回归算法(Random Forest, RF)和其它对比算中进行预测，将预测结果进行对比分析。对比算法包括支持向量回归(Support Vector Regression, SVR)算法[13]、K最近邻(K-Nearest Neighbor, KNN)回归算法[14]和5层反馈神经网络(Back Propagation Neural Network, BPNN)[15]，其中SVR回归算法分别使用poly和linear内核。3个基准测试生成的数据集描述如表2所示。

本地搭建的Apache Flink集群由4个节点构成，其中1个节点是Master，另外3个节点是Slave。除此之外还有1个外部服务器用来实时采集基准测试的特征数据和检查点执行时间。5个服务器使用的硬件配置一致：8核CPU “Intel (R) Core (TM) i7-4790 3.60 GHz”，内存8 G，硬盘500 G，操作系统为Centos-6.5。

6.2 基准测试

为全面的评估本文所提的流处理检查点性能预测效果，实验设计的3个基准测试如图3所示。

图3中，R代表Source, F代表Flat Map, K代表Key Agg, ST代表Stream Table, TT代表To Tuple, M代表Map, S代表Sink。图3展示的3个基准测试描述如下。

CPU密集型基准测试(CKCPU)：该基准测试并行度设置为8，即n为8。Source算子读取双城记(英文版)小说并将内容发送给下游算子Flat Map。该算子对接收到的内容分割成单词然后发送给下游算子Key Agg。Key Agg算子对接收到的单词进行统计然后将统计结果发送给下游算子Sink。该基准测试除了Flat Map算子到Key Agg算子存在跨节点的网络传输外其它的逻辑都不存在网络传输。Flat Map算子需要频繁地对语句进行拆分，Key Agg算子需要持续不断地对单词进行统计，从而大量地消耗CPU资源。

内存密集型基准测试(CKMEM)：该测试中Source算子读取同样的小说内容，然后将内容转换成两个流表，即Stream Table。To Tuple算子将两个流表内容组合并转换为元组，最终将元组数据发送给Sink算子，通过频繁地对流数据转换操作而大量地消耗内存资源。该基准测试并行度同样设置为8，即n为8。

网络密集型基准测试(CKNET)：该基准测试中，Source算子读取同样的数据并分别发送给下游的每一个算子，下游算子Map接收到数据后不进行任何处理再将数据发送给下游的每一个算子Sink。通过最小化系统对CPU和内存资源的消耗，而将流数据不断地发送给集群中的每一个节点来消耗集群的网络资源。

表 2 数据集描述

基准测试	样本数量	特征数量	训练样本数量	预测样本数量
CKCPU	47100	332	37680	9420
CKMEM	10290	172	8232	2058
CKNET	18900	524	15120	3780

6.3 评价指标

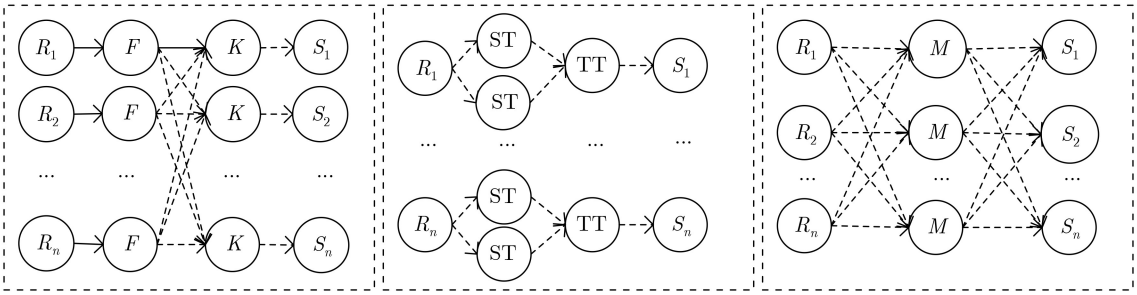
本文使用平均绝对误差、均方根误差、中值误差和准确率作为评价指标，如式(18)—式(21)所示

$$\mathrm{MAE}=\frac{1}{c}\sum_{i=1}^{c}|f_i-y_i|=\frac{1}{c}\sum_{i=1}^{c}|e_i| \tag{18}$$

$$\mathrm{RMSE}=\sqrt{\frac{1}{c}\sum_{i=1}^{c}(f_i-y_i)^2} \tag{19}$$

$$\mathrm{MediaAE}=\mathrm{media}\left(|y_i-\hat{y}_i|,\cdots,|y_i-\hat{y}_c|\right) \tag{20}$$

$$\mathrm{Accurate}_{\mathrm{average}}=\frac{1}{c}\sum_{i=1}^{c}\left(1-\left|\frac{f_i-y_i}{y_i}\right|\right) \tag{21}$$



(1) CPU密集型基准测试 (2) 内存密集型基准测试 (3) 网络密集型基准测试

图 3 基准测试

在所有的评价指标中，c为样本数量，f_i为检查点执行时间的预测值，y_i均为检查点执行时间的真实值。

6.4 预测误差对比

本文使用多个对比算法对流处理检查点性能进行预测，其误差结果如表3所示。

表3中，在CKCPU上，RF在MAE, RMSE和MediaAE指标分别为0.040178, 0.068811和0.125560，此数值在所有回归算法的误差中最小。在CKMEM上，RF在MAE和MediaAE指标上的误差分别是0.096046和0.206272，该数值在所有回归算法的误差中最小。在RMSE指标上，RF误差为0.196619，该数值在所有回归算法中排名第2。在CKNET上，RF在MAE, RMSE和MediaAE指标上的数值都是最小，分别为0.019501, 0.089315和0.089082。综合分析可知，随机森林回归算法在3个基准测试上的误差结果均保持在较低的水平。

6.5 预测准确率对比

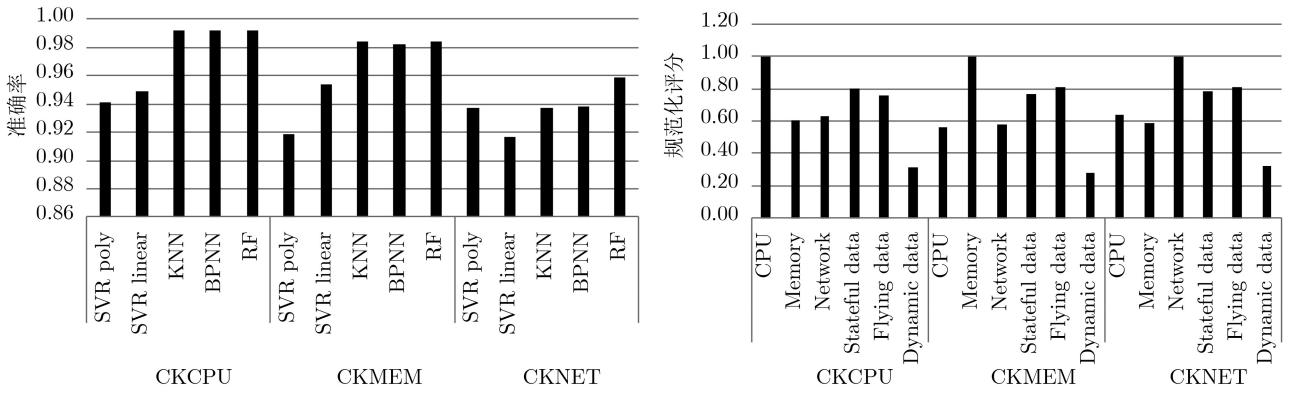
本节使用式(21)对不同算法的准确率做相应的对比，同时展示不同特征的评分结果，如图4所示。

图4(a)中，在CKCPU上，RF, BPNN和KNN的准确率最高。在CKMEM上，只有两个SVR算法的准确率相对较低，其它回归算法(包括RF)的准确率都较高且基本一致。在CKNET上，RF算法的准确率明显高于其它算法。总之，在不同的基准测试上，RF均表现出较高的准确率，其平均准确率为0.97。

图4(b)中，不同类型的基准测试对特征的任务偏重不同。CKCPU中，CPU类特征的重要性评分最高。在CKMEM中，内存类特征的评分最高。在

表 3　不同回归算法预测误差结果

基准测试	回归算法	MAE	RMSE	MediaAE
CKCPU	SVR poly	0.107006	1.900023	37.921288
	SVR linear	0.095006	27.06338	37.529361
	KNN	0.108006	0.323870	0.286494
	BPNN	0.042380	0.070043	0.129856
	RF	**0.040178**	**0.068811**	**0.125560**
CKMEM	SVR poly	0.115007	0.037560	10.924428
	SVR linear	0.178010	2.524596	4.085918
	KNN	0.148008	0.370660	0.373577
	BPNN	0.097356	0.199461	0.214980
	RF	**0.096046**	**0.196619**	**0.206272**
CKMEM	SVR poly	0.091005	0.645619	0.634070
	SVR linear	0.301017	0.545833	0.523365
	KNN	0.102006	0.742873	0.742375
	BPNN	0.020343	0.103857	0.147659
	RF	**0.019501**	**0.089315**	**0.089082**



(a) 不同算法预测准确率　　(b) 特征重要性评分

图 4 不同回归算法的预测准确率和不同特征重要性评分

CKNET中，网络类特征的重要性评分最高。另外，状态数据类特征和飞行数据类特征在所有基准测试上的重要性评分较高。该结果表明这两类特征在3类不同基准测试上表现稳定，不会因基准测试的类别不同而表现出较大的差异。

6.6 算法效率对比

本文将随机森林回归算法和其它对比算法的效率进行对比，实验结果如图5所示。

图5中，在CKCPU上，RF和KNN回归算法在执行时耗费的时间最少，同时它们两个算法的执行时间基本一致。在CKMEM上，RF的执行时间最短，KNN的执行时间略长一点。在CKNET上，RF执行时间依然是最少的，KNN的执行时间略长一点。通过图5可以看出，RF在所有对比算法中的效率都是最高的。对应的平均执行时间为0.0024 s。特别地，BPNN的执行效率未在图中展示。由于该算法的执行耗费时间比随机森林算法多8.5倍左右，若展示在图中将影响其它算法的展示效果。

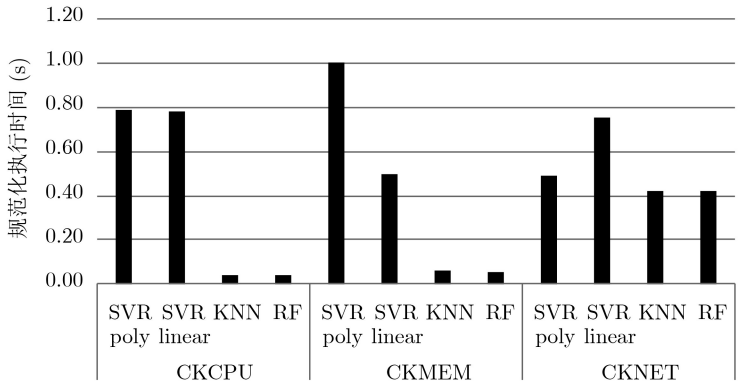


图 5 不同回归算法的执行效率

7 讨论

预测准确率对比实验结果表明随机森林算法具有较高的准确率。首先，随机森林中的每棵决策树在训练时使用的是随机样本，即总体样本中的随机子集。该特点在样本层面增加了多样性。其次，每棵决策树在构建过程中都选择当前的最佳特征进行分裂，这使对预测准确率贡献大的特征得到优先考虑。最后，随机森林的预测结果是由多棵决策树共同预测的结果，这使得算法具有较强的泛化能力。预测效率对比实验结果表明随机森林算法具有较高的效率，这是由于该算法的运行方式是并行的。

不同类别特征重要性评分结果表明状态数据类特征和飞行数据类特征对算法的预测贡献较大。在检查点执行过程中，状态数据是需要转存至外部的数据，这类数据的多少将直接影响检查点的性能。飞行数据是在流处理系统中还未处理完的数据，检查点执行时需要等待该类数据被处理完，所以检查点性能受该类特征影响较大。特别地，这两类特征在不同基准测试上的重要性评分稳定，这也证明所识别出的特征具有一定的普遍性。

8 结束语

本文对流处理检查点性能预测进行了相关的研究。由于当前的检查点策略是基于经验进行设置的，必然使系统用户陷入“尝试-错误-修正”的循环模式中。为降低系统用户的使用负担，本文首先对检查点设置策略所存在的问题进行了分析和总结，然后识别出对其性能产生影响的重要特征。利用随机森林算法对3个基准测试进行性能预测，通过分析随机森林与对比算法在误差、准确率和执行效率上的结果，表明随机森林算法在性能预测上拥有较低的误差、较高的准确率和较高的执行效率。未来针对流处理检查点的研究将基于本文的研究成果实现自动化策略，从而彻底使系统用户从检查点策略设置任务中解放出来。

参 考 文 献

[1] 彭建华, 张帅, 许晓明, 等. 物联网中一种抗大规模天线阵列窃听者的噪声注入方案[J]. 电子与信息学报, 2019, 41(1): 67–73. doi: 10.11999/JEIT180342.

PENG Jianhua, ZHANG Shuai, XU Xiaoming, *et al*. A noise injection scheme resistant to massive MIMO eavesdropper in IoT[J]. *Journal of Electronics & Information Technology*, 2019, 41(1): 67–73. doi: 10.11999/JEIT180342.

[2] 刘素艳, 刘元安, 吴帆, 等. 物联网中基于相似性计算的传感器搜索[J]. 电子与信息学报, 2018, 40(12): 3020–3027. doi: 10.11999/JEIT171085.

LIU Suyan, LIU Yuan'an, WU Fan, *et al*. Sensor search based on sensor similarity computing in the Internet of Things[J]. *Journal of Electronics & Information Technology*,

2018, 40(12): 3020–3027. doi: 10.11999/JEIT171085.

[3] CARBONE P, EWEN S, FÓRA G, *et al*. State management in Apache Flink®: Consistent stateful distributed stream processing[J]. *Proceedings of the VLDB Endowment*, 2017, 10(12): 1718–1729. doi: 10.14778/3137765.3137777.

[4] VENKIVOLU D R and NALE M N. Adaptive encryption in checkpoint recovery of file transfers[P]. US, 20190306221, 2019-10-03.

[5] KIM Y, NAKAMURA J, KATAYAMA Y, *et al*. A cooperative partial snapshot algorithm for checkpoint-rollback recovery of large-scale and dynamic distributed systems[C]. The 6th International Symposium on Computing and Networking Workshops (CANDARW), Takayama, Japan, 2018: 285–291. doi: 10.1109/CANDARW.2018.00060.

[6] TAO Yangyang and YU Shucheng. kFHCO: Optimal VM consolidation via k-Factor horizontal checkpoint oversubscription[C]. 2019 International Conference on Computing, Networking and Communications (ICNC), Honolulu, USA, 2019: 380–384. doi: 10.1109/ICCNC.2019.8685604.

[7] GOUNTIA D and ROY S. Checkpoints assignment on cyber-physical digital microfluidic biochips for early detection of hardware Trojans[C]. The 3rd International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2019: 16–21. doi: 10.1109/ICOEI.2019.8862598.

[8] ZHANG Hanlin, CHEN Ningjiang, TANG Yusi, *et al*. Multi-level container checkpoint performance optimization strategy in SDDC[C]. The 4th International Conference on Big Data and Computing, Guangzhou, China, 2019: 253–259. doi: 10.1145/3335484.3335487.

[9] TITOUNA C, MOUMEN H, and ARI A A A. Cluster head recovery algorithm for wireless sensor networks[C]. The 6th International Conference on Control, Decision and Information Technologies (CoDIT), Paris, France, 2019: 1905–1910. doi: 10.1109/CoDIT.2019.8820414.

[10] OVENS S and WOELFEL P. Strongly linearizable implementations of snapshots and other types[C]. 2019 ACM Symposium on Principles of Distributed Computing, Toronto, Canada, 2019: 197–206. doi: 10.1145/3293611.3331632.

[11] ATHEY S, TIBSHIRANI J, WAGER S, *et al*. Gemeralized random ferests[J]. *Annals of statistics*, 2019, 47(2): 1148–1178. doi: 10.1214/18-AOS1709.

[12] CHOI J, GU B, CHIN S, *et al*. Machine learning predictive model based on national data for fatal accidents of construction workers[J]. *Automation in Construction*, 2020, 110: 102974. doi: 10.1016/j.autcon.2019.102974.

[13] LYU J and MANOOCHEHRI S. Dimensional prediction for FDM machines using artificial neural network and support vector regression[C]. ASME 2019 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Anaheim, USA, 2019. doi: 10.1115/DETC2019-97963.

[14] DABERDAKU S, TAVAZZI E, and DI CAMILLO B. Interpolation and K-nearest neighbours combined imputation for longitudinal ICU laboratory data[C]. 2019 IEEE International Conference on Healthcare Informatics (ICHI), Xi'an, China, 2019: 1–3. doi: 10.1109/ICHI.2019.8904624.

[15] ASAAD R R and ALI R I. Back Propagation Neural Network (BPNN) and sigmoid activation function in multi-layer networks[J]. *Academic Journal of Nawroz University*, 2019, 8(4): 216–221. doi: 10.25007/ajnu.v8n4a464.

褚　征：男，1991年生，博士生，研究方向为分布式计算、内存计算和机器学习.

于　炯：男，1966年生，教授，研究方向为分布式计算、内存计算和绿色计算.