容器虚拟化支撑平台监测系统①

顾泽宇[1,2], 王 焘[1], 宋云奎[1], 杨红涛[3], 张文博[1]

[1](中国科学院 软件研究所, 北京 100190)
[2](中国科学院大学, 北京 100049)
[3](太原罗克佳华工业有限公司, 太原 030032)

摘 要: 在容器虚拟化支撑平台中, 多种应用竞争共享资源造成监测系统难以通过设定固定阈值的方式进行异常报警, 同时固定的监测周期需要在监测及时性与监测开销之间权衡. 针对这些问题, 本文提出了一种基于 PCA 的在线异常检测方法, 并根据异常程度动态调整监测周期以节省监测开销. 首先通过非侵入方式收集每个容器的监测数据构成数据矩阵, 然后利用主成分分析的方法计算矩阵的主方向, 最后计算当前与上次主方向的余弦相似度, 将其作为当前系统状态的异常程度. 对于异常程度超过设定的用户容忍度的容器, 发送告警信息, 同时根据异常程度增大或者减小监测周期. 实验结果表明, 对于在分布式存储系统中注入的典型错误, 本方法的监测准确性达到 80%以上, 告警延迟控制在 5 秒以内, 监测开销低于固定监测周期的方法.

关键词: Docker 容器; 虚拟化; 运行时监测; 主成分分析; 监测周期

Monitoring System for Container-Based Virtualization Platforms

GU Ze-Yu[1,2], WANG Tao[1], SONG Yun-Kui[1], YANG Hong-Tao[3], ZHANG Wen-Bo[1]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100049, China)
[3](Taiyuan RocKontrol Industrial Corporation Limited, Taiyuan 030032, China)

Abstract: In container-based virtualization platforms, detecting anomalies with fixed thresholds is not practical, because many kinds of applications completely shared physical resources. Furthermore, monitoring systems need to consider the tradeoff between the timeliness and overhead by adjusting the monitoring period. To address these issues, this paper proposes an online anomaly detection method based on Principle Component Analysis (PCA), and then adjusts the monitoring period according to the anomaly significance. First, it non-intrusively collects the monitoring data of containers to get a matrix for each container. Then, it uses PCA to present the main direction of each matrix, and calculates the abnomal significance by calculating the cosine similarity between the current direction and the last one. If the significance is out of the defined tolerability threshold, the monitoring system sends an alert message to administrators, and automatically adjusts the monitoring period according to the significance. The experimental results demonstrate that our method can detect typical faults injected in HDFS with the accuracy of 80%, the delay of alerts iswithin 5 seconds, and the monitoring overhead ismuch lower than that with a fixed monitoring period.

Key words: docker container; virtualization; runtime monitoring; principal component analysis; monitoring period

1 引言

容器本质是一种复用 Kernel, 为应用进程提供隔离机制的虚拟化技术. 容器虚拟化不同于 Hypervisor 的虚拟化, 具有资源开销小, 隔离性强, 部署时效性好的特点, 因此互联网企业倾向于使用大规模容器集群应对突发的峰值访问. 例如京东运行着世界上最大规模的 Docker 集群(6 万个), 所有在线应用在新机房都将通过 Docker 技术进行发布, 支持同时在线的 Docker

① 基金项目:国家科技支撑计划(2013BAH45F01);国家自然科学基金(61402450);北京市自然科学基金(4154088)
收稿时间:2016-01-27;收到修改稿时间:2016-03-14 [doi:10.15888/j.cnki.csa.005372]

实例达到 10 万规模, 可以在 10 秒内快速扩展上千个容器, 以应对商品秒杀、商品详情页、图片展示产生的突发流量; 容器成功地在羊年春晚为 1.02 亿用户同时抢微信红包提供服务; 蘑菇街采用容器技术实现了持续集成, 灰度升级, 弹性伸缩; 腾讯的 Gaia 平台作为腾讯大数据平台的底层资源管理和调度系统, 服务于腾讯所有事业群, 在 2014 年 10 月份正式上线 Docker, 解决各个业务的环境依赖; 阿里云 ECS 支持部署 Docker 容器应用; Baidu App Engine(BAE)以 Docker 作为其 PaaS 云基础环境.

虽然大规模的容器集群为互联网应用提供了快速扩容的能力, 但是在突发的峰值访问到来的时候, 容器的稳定性依然需要得到足够的关注. 例如 2015 年双十一购物狂欢节, 京东, 支付宝等网站访问异常, 造成数据加载失败, 订单无法完成, 无法完成支付, 严重影响了用户体验. 根据亚伯丁集团的报告, “1 秒的延迟, 会带来 11% 用户点击数的下降, 16% 客户满意程度的下降和 7% 的经济损失”. 所以通过监测技术及时发现异常并通知系统管理员, 以保证系统的服务质量变得非常重要.

对于容器虚拟化支撑平台的监测, 由于物理机上运行的容器个数及类型随业务需求不断发生变化, 导致待监测对象具有动态性. 我们用下面的例子来解释被监测对象的动态变化对监测造成的挑战. 监测初期, 我们只需要监测装着 Tomcat 的容器, 在传统的监测方式中, 我们为度量设定固定的阈值, 例如内存使用上限是 1GB. 当监测进行到某一时间点, 系统中新增了一个装着 MySQL 的容器, 由于容器环境中存在资源抢占问题, 所以 Tomcat 原有的内存阈值就要根据 MySQL 内存占用率进行调整, 这种调整在被监测实例个数很多的时候变得不切实际, 因此容器虚拟化环境下为度量设定静态阈值难以准确检测异常. 另一方面, 监测系统多久能够发现异常, 很大程度上依赖于监测周期. 当突发的峰值访问到来的时候, 容器更容易异常, 因此应该使用更小的监测周期进行密集监测. 但是如果将监测周期设置的过小开销较大. 而且已有的研究表明容器集群的监测会占用较多的系统资源. 例如普通配置的物理机可以启动数百个容器, 由于集群节点可能位于地理位置不同的数据中心, 如果 1000 个应用部署在节点规模为 500 的集群上, 监测这些应用需要 230M/S 的带宽. 这就导致容器虚拟化环境下很难通过固定的监测周期在保证监测及时性的前提下降低监测开销.

2 相关工作

当前的研究工作提出了很多异常检测方法, 这些方法大致可以分为如下几类: 基于规则的方法通过为每个度量设定阈值来检测异常, 例如 Zabbix, Nagios, Hyperic HQ 等监测系统, 但是这种方法不适合度量繁多且动态变化的场景. 基于统计的方法假设数据服从一些标准的分布模型, 找出偏离分布的离群点[1], 但是由于大部分模型基于单变量的假设, 对于多维数据的鲁棒性不够好, 与此同时, 这些模型从原始数据空间计算得到, 会受到噪音数据的干扰[2]. 基于距离的方法[3]不需要对于数据分布的先验知识, 通过计算样本与其第 k 个最邻近实例的距离来判断是否异常, 但是这种方法不适用于数据分布属于多重簇结构的场景[4]. 为了解决上述发现的问题, 学术界提出了基于密度的方法, 这种类型的方法中比较有代表性的是 LOF(Local Outlier Factor)[5], 即基于密度的本地异常因子来度量每个数据实例的异常程度. LOF 最重要的属性是能够通过对本地数据的密度估计来识别离群点. 但是本地数据的密度估计在样本数据规模很大的时候会造成很大的计算开销[6]. 基于角度的离群点检测算法(Angle-Based Outlier Detection, ABOD)[7]计算每一个目标实例与其他数据之间的角度变化, 这种方法基于一个观察: 离群点会比正常点造成更大的角度偏差. 由于需要计算大量的样本对, 所以在样本数据规模很大的时候 ABOD 方法的计算开销过大.

现有监测系统普遍采用固定的监测周期来轮询系统的异常情况, 这种方式可以通过离线测试确定适用于当前环境的监测周期, 在被监测度量比较固定、单一的场景下取得了很好的效果, 但是该方法需要特定的领域知识, 而且容器虚拟化环境下被监测对象多且动态变化的特点导致难以通过离线测试确定合适的监测周期.

3 整体思路

当前互联网应用种类繁多, 需要监测的度量达到上万规模, 因此难以通过设定静态阈值进行异常检测, 因此我们尝试将监测度量看成一个整体进行异常评估. 主成分分析(Principal Component Analysis, PCA)可以

获得当前数据集的主方向，主方向的偏离程度即可衡量当前数据的异常程度[8]. PCA 算法只需要保存上一次的计算结果即可得到算法需要的协方差矩阵，具有较低的时间复杂度，因此适合在度量动态变化的容器虚拟化环境中进行在线异常诊断的应用场景.

为了保证监测任务不给应用系统增加额外的开销，本文根据 PCA 算法得到的异常程度动态调整监测周期，在异常程度较小时，使用较大的监测周期，在异常程度较大时，使用较小的监测周期. 通过这种方式可以在不降低监测准确性与及时性的前提下减小监测开销.

本文采用了非侵入的监测方式，通过容器自发现引擎自动发现当前物理机上处于运行状态的容器，并自动将容器注册成为监测平台中的监测项，然后采用并行收集的方式获取容器的性能数据，这些数据一部分持久化到数据库，另一部分直接发送给异常检测模块. 异常检测模块将每个容器的数据分开存储，对每个容器的性能数据运行 PCA 算法，得到当前数据分布的主方向，然后计算当前主方向与上一次主方向的余弦相似度作为当前样本数据的异常程度，同时对判定为异常的容器及时报警. 监测周期调谐器根据异常检测模块得到的异常程度动态调整监测周期.

4　监测方法

4.1　监测度量选择

PCA 算法处理线性结构有很好的效果，然而实际中的监测数据矩阵并非线性结构. 因此需要选取监测数据中满足线性关系的度量进行分析. 本文根据历史数据，利用回归分析筛选出对于异常检测具有显著影响的度量作为 PCA 算法的输入.

在一元线性回归分析中，通常可以首先通过散点图判断变量 x 与 y 之间是否存在线性关系. 如果散点图上的实验数据接近于某一条直线，便可直观地初步认为二者之间存在线性关系. 但在多元线性回归分析中情况略有不同. 首先无法用直观的方法帮助判断度量间是否具有线性关系，为此必须对回归方程进行显著性检验. 其次在多个自变量中，每个自变量在异常检测中的重要性程度是不同的，这表现在回归系数中就是有的绝对值很大，有的很小或接近于零，这就需要对回归系数进行显著性检验.

本文在线下利用回归分析的方法，根据历史数据得到具有线性关系且对异常诊断影响显著的度量，在线上只收集这些度量构建矩阵，利用 PCA 进行分析.

4.2　面向动态度量的异常检测

使用 PCA 进行异常程度评估的步骤包括：1)如果有 m 条 n 维数据，将原始数据按列组成 n 行 m 列矩阵 X；2)将 X 的每一行进行零均值化处理；3)求出协方差矩阵；4)求出协方差矩阵的特征值及对应的特征向量；5)将特征向量按特征值大小从上到下按行排列成矩阵，取前 k 行组成矩阵 P；6)$Y=PX$ 即为降维到 k 维后的数据.

PCA 算法第 4 步计算得到的特征向量就是数据集分布的主方向. 同时，计算特征向量使用的协方差矩阵和平均值都对离群点比较敏感，所以离群点会导致数据集主方向有较大的改变，如图 1 所示. 因此我们可以把异常数据定义成使数据集的主方向偏离到一定程度的监测数据.

每次收集到新的监测数据以后，计算新的监测数据集的主方向与原有监测数据集主方向之间的角度偏差，然后利用余弦相似度来衡量当前样本的异常程度.

方法的输入包括：

$X_i=(x_1, x_2,\ldots, x_p)$：收集到的监测数据，$p$ 是度量的个数；

n：样本的个数；

A：已有监测数据集，np 矩阵；

d：已有监测数据集的分布方向；

r：数据复制比例.

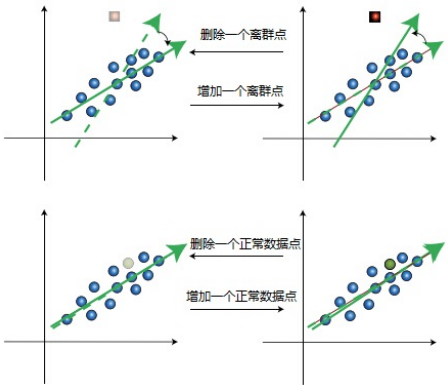

图 1　离群点对数据主方向的影响

具体步骤如下：

① 放大离群点影响：将当前监测数据复制 nr 份，加入 A，得到 An；

② 零均值化：计算 An 的平均值 avg=$(u_1,u_2,\ldots,u_p)$，

对 *An* 进行零均值处理;

③ 计算 *An* 的协方差矩阵:

$$\sum_{A_n} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (x_i - \mu)(x_j - \mu)^T ;$$

④ 获得数据集分布方向: 计算$\sum A$ 的特征向量, 作为当前数据集分布方向 d_t.

方法的输出: 当前监测数据 $X_i=(x_1, x_2,\ldots, x_p)$的异常程度为:

$$s_t = 1 - \left| \frac{\langle \tilde{d}_t, d \rangle}{\|d_t\| \|d\|} \right| .$$

4.3 基于异常程度调整监测周期

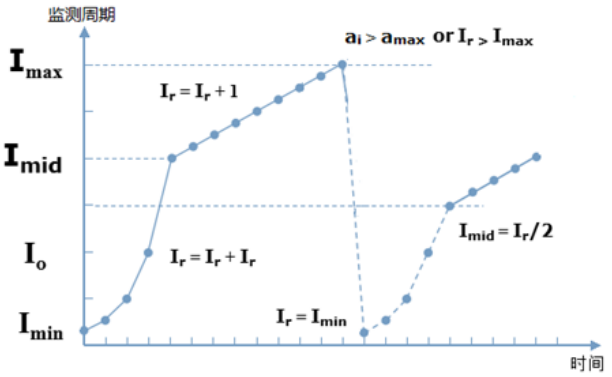

图 2 监测周期调整示意图

监测周期调整原理如图 2 所示, 用户首先在配置文件中设定监测周期的上限、下限, 异常避免的监测周期阈值以及异常容忍度. 如果新收集到的样本的异常程度小于用户设定的异常容忍度, 就将监测周期翻倍; 当监测周期达到异常避免的监测周期阈值, 就进入密集监测阶段, 每次将监测周期增加一个时间单位, 而不是原来的翻倍; 无论何时检测到异常程度超过异常容忍度或者当前监测周期达到监测周期上限, 立即将监测周期调整为用户设定的监测周期下限, 并且将异常避免的监测周期阈值调整为当前监测周期的一半.

调整算法描述如下:

算法 1 基于异常程度的监测周期调整算法

Input: current detection period I_0, maximum detection period I_{max}, minimium detection period I_{min}, maximum detection period anomaly avoidance I_{mid}, user-defined maxmium intensity of anomaly a_{max}, current intensity of anomaly a_i

Output: adjusted detection period I_r

Pseudocode:

set $I_r = I_0$

if $a_i < a_{max}$ **or** $I_r < I_{max}$ **then:**

  if $I_r < I_{max}$ **then:**

    set $I_r = I_r + I_r$

  else:

    set $I_r = I_r + 1$

else:

  set $I_r = I_{min}$

  set $I_{mid} = I_r/2$

end if

5 系统设计与实现

监测系统面向中科院软件所研发的具有自主知识产权的 OnceCloud 容器虚拟化支撑平台进行设计与实现, 监测系统架构如图 3 所示, 包含以下模块:

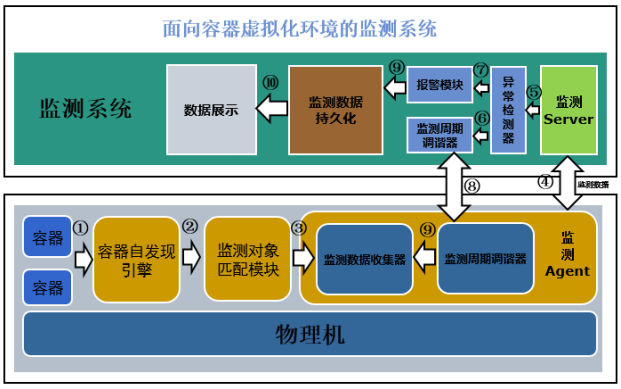

图 3 系统架构图

监测 Agent: 通过监测数据收集器, 监测 Agent 实现了监测数据的收集; 监测 Agent 会跟监测 Server 进行通信, 建立一个数据传输的通道, 收集到的监测数据会通过这个通道进行传输; 监测 Agent 中的监测周期调谐器会接收位于监测 Server 中的监测周期调谐器发送的监测周期调整指令, 根据指令调整监测周期.

容器自发现引擎: 通过脚本发现物理机上的所有容器的 UUID 信息, 这些信息发送给监测的 GUI 进行监测注册.

监测对象匹配模块: 为不同中间件匹配不同的监

测脚本, 使用对应的监测脚本获取监测数据.

监测 Server: 与监测 Agent 进行通信, 接收监测 Agent 发送的监测数据.

异常检测器: 封装了基于 PCA 的异常程度评估方法, 以监测 Server 中的监测数据作为输入, 异常程度作为输出.

报警模块: 如果异常程度大于用户设定的异常容忍度, 报警模块就会将异常信息显示在监测 GUI.

监测数据持久化模块: 模块中可以配置监测数据持久化的时间间隔, 该模块还会存储被监测节点的 IP、监测端口号, 报警信息, 被监测度量的 key, value 等信息.

监测 GUI: 用来展示监测系统的概要信息, 被监测的节点数, 被监测度量, 监测数据, 报警信息.

6 实验评价

6.1 实验环境

实验环境如图 4 所示, 包含使用 Docker 容器搭建的 HDFS 分布式存储服务, HDFS 建立在三台 24G 内存, 500GB 硬盘的物理机搭建的 Docker 容器集群上, 其中包括 90 个 DataNode 节点和 1 个 NameNode 节点. 还有一台 8G 内存的 PC 机上运行着错误注入组件, 负载发生器和监测平台.

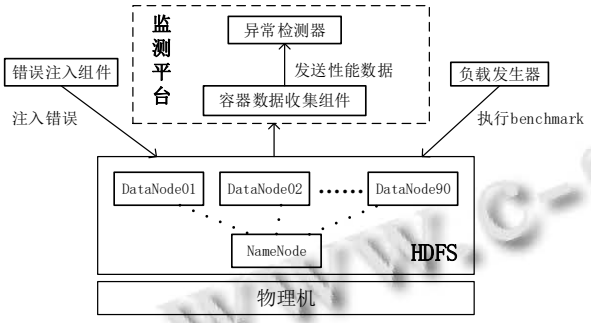

图 4　实验架构图

6.2 错误注入

本文注入了 6 种类型的 HDFS 的典型错误, 包括: 进程类型的错误, 即杀死某个 DataNode 上的 HDFS 进程; 网络类型的错误, 即断开与某个 DataNode 的网络连接; 存储类型的错误, 即删除某个 DataNode 上的数据文件; Hadoop bug list 中的错误(Hadoop-issue-6502), 对较长目录进行 ls 操作延迟较大; Hadoop bug list 中的错误(Hadoop-issue-5588), 在 0.20 分支版本上执行 Hadoop 命令特别慢; Hadoop bug list 中的错误(Hadoop-issue-9150), 对逻辑 URI 进行了过多不必要的 DNS 解析.

6.3 负载类型

使用了 Hadoop 中提供的 TestDFSIO: Distributed I/O Benchmark. 实验用到的负载包括两种类型, 读文件请求和写文件请求, 负载发生器中的脚本各自产生一种请求类型的负载, 通过循环将请求持续地发给 HDFS.

6.4 实验步骤

一共进行了 6 次实验, 每次注入一种错误, 每次实验步骤如图 5 所示.

① 启动监测平台及 HDFS 服务;

② 启动负载发生器;

③ 错误注入器随机选择 10 个时间点注入错误.

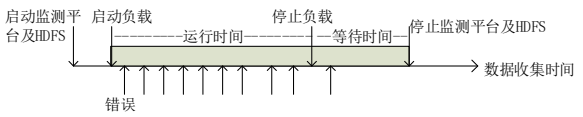

图 5　实验步骤

6.5 实验结果

1) 监测准确性:

实验结果如表 4 所示, 前三个错误类型都是对 DataNode 进行错误注入, 方法并不是总能检测出所有错误. 因为 HDFS 使用心跳协议等机制自动地检测和避免错误, 如果某些错误注入以后遇到心跳检测, 系统度量会很快恢复正常, 难以通过监测平台发现异常. 后面三个错误会造成系统度量较长时间处于异常水平, 因此监测平台判定的错误次数大于实际的注入错误数.

表 4　准确性

错误类型	注入错误数	检出错误数
杀死某个 DataNode 上的 HDFS 进程	10	8
端口与某个 DataNode 的网络连接	10	8
删除某个 DataNode 上的数据文件	10	10
ls 一个大小为 1300 的文件夹耗时长	10	12
在 0.20 分支上执行 Hadoop 命令很慢	10	9
对逻辑 URI 进行不必要的 DNS 解析	10	10

2) 监测及时性:

错误注入时间与异常发现时间的差值定义成 delay={ $x_1, x_2, x_3, \ldots, x_{10}$}; 纵坐标是 delay 的平均值; 横坐标是注入的 6 种错误类型的标号. 监测结果如图 6

所示，由于动态调整监测周期，报警延迟明显小于采用固定监测周期的方法.

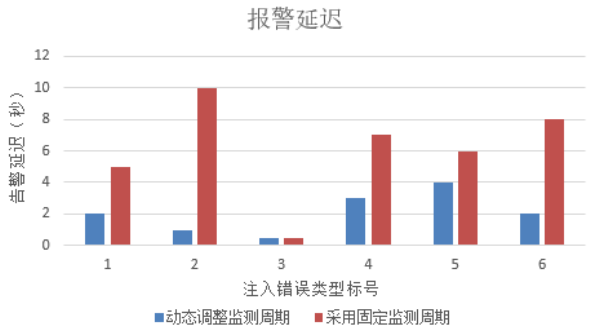

图 6 报警延迟

3) 监测方法计算开销:

实验结果如图 7 所示，当监测度量个数增长的时候，方法计算开销增长的快；监测数据个数增长的时候，方法计算开销增长的慢. 方法对于监测度量个数更敏感，根本原因是方法包含计算矩阵特征值和特征向量的过程. 从图中可以看出方法更适合计算度量个数在 100 个以内的场景.

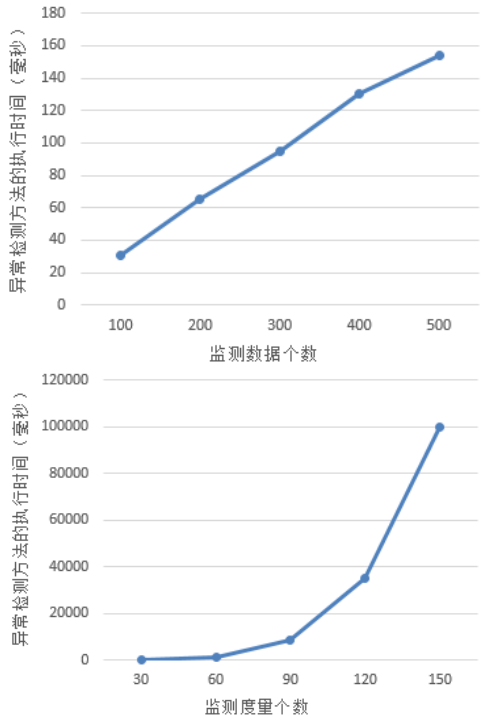

图 7 计算开销

7 结论

本文针对容器虚拟化支撑平台中被监测度量动态变化的特点和监测的低开销要求，提出了一种基于 PCA 的异常程度评估方法来保证监测的准确性和及时性，然后根据异常程度动态调整监测周期以降低开销. 实验结果表明，对于在分布式存储系统中注入的典型错误，本方法的监测准确性达到 80%以上，告警延迟控制在 5 秒以内，监测开销低于固定监测周期的方法. 同时，我们观察到，文中提出的异常评估方法更适合计算度量个数在 100 个以内的场景. 这是由于在计算数据分布主方向时，需要保存所有样本数据以计算协方差矩阵及特征向量，这一计算过程对样本数据的维度更敏感，在处理高维数据时，时间复杂度较高. 进一步工作，将探索高维数据的处理方式，对低维样本数据和高维样本数据使用不同的算法分开处理.

参考文献

1 Chandola V, Banerjee A, Kumar V. Anomaly detection: A survey. New York: ACM Comput Surv, 2009, 41(3): 1–58.

2 Khoa NLD, Chawla S. Robust outlier detection using commute time and eigenspace embedding. Proc. Pacific-Asia Conf. Knowledge Discovery and Data Mining. Springer Berlin Heidelberg. 2010. 422–434.

3 Angiulli F, Basta S, Pizzuti C. Distance-based detection and prediction of outliers. IEEE Trans. on Knowledge and Data Engineering, 2006: 145–160.

4 Knox EM, Ng RT. Algorithms for mining distance-based outliers in large data sets. Proc. Int'l Conf. Very Large Data Bases. San Francisco: Morgan Kaufmann Publishers Inc. 1998: 392–403.

5 Breunig M, Kriegel HP, Ng RT, Sander J. LOF: Identifying density-based local outliers. Proc. ACM SIGMOD Int'l Conf. Management of Data. New York. 2000. 93–104.

6 Pokrajac D, Lazarevic A, Latecki L. Incremental local outlier detection for data streams. Proc. IEEE Symp. Computational Intelligence and Data Mining. Honolulu. 2007. 504–515.

7 Kriegel HP, Schubert M, Zimek A. Angle-based outlier detection in high-dimensional data. Proc. 14th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining. New York. 2008. 444–452.

8 Sadik S, Gruenwald L. Online outlier detection for data streams. Proc. of the 15th Symposium on International Database Engineering & Applications. New York. 2011. 88–96.