软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(10):2705−2719 [doi: 10.3724/SP.J.1001.2012.04197]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于负载模式识别的 Web 应用在线异常检测方法*

王 焘[1,2,3+], 魏 峻[1,2], 张文博[1], 钟 华[1]

[1](中国科学院 软件研究所 软件工程技术研究开发中心,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[3](中国科学院 研究生院,北京 100049)

Online Anomaly Detection Approach for Web Applications with Workload Pattern Recognition

WANG Tao[1,2,3+], WEI Jun[1,2], ZHANG Wen-Bo[1], ZHONG Hua[1]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[3](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: E-mail: wangtao08@otcaix.iscas.ac.cn

Wang T, Wei J, Zhang WB, Zhong H. Online anomaly detection approach for Web applications with workload pattern recognition. *Journal of Software*, 2012,23(10):2705−2719 (in Chinese). http://www.jos.org.cn/1000-9825/4197.htm

Abstract: The dynamic fluctuation of workload influences system metrics, affects the precision of anomaly detection. This paper proposes an online anomaly detection approach for Web applications, which handles workload fluctuation in both request pattern and volume. The study proposes an incremental clustering algorithm to recognize online workload patterns automatically. For a specific workload pattern, the study adopts local outlier factor to detect anomaly and qualify the anomaly degree, and then locate the abnormal metrics with a student's *t*-test method. The experimental results show that the clustering algorithm can accurately capture workload fluctuations in a typical Web application, and demonstrate that the approach is capable of not only detecting the typical faults in Web applications, but also locating the abnormal metrics.

Key words: Web application; anomaly detection; dynamic workload; incremental clustering; LOF (local outlier factor)

摘 要: 负载模式的动态变化会影响系统度量,使得异常难以准确检测.针对此问题,提出一种基于负载模式识别、在线检测Web应用异常的方法.该方法基于在线增量式聚类算法,运行时识别动态变化的负载模式,根据特定负载模式对应的度量空间,利用局部异常因数检测异常状态,并量化异常程度,并通过学生 *t* 测试方法计算度量异常值,以定位异常原因.实验结果表明,所提方法能够准确识别负载模式变化,有效检测出 Web 应用典型错误所引起的异常状态,并定位异常原因.

关键词: Web 应用;异常检测;动态负载;增量式聚类;局部异常因数

* 基金项目: 国家自然科学基金(61173004); 国家重点基础研究发展计划(973)(2009CB320704); "核高基"国家科技重大专项(2011ZX03002-002-01)

收稿时间: 2011-07-03; 修改时间: 2011-11-02; 定稿时间: 2012-02-15

中图法分类号: TP311 文献标识码: A

随着 Internet 技术的快速发展,电子商务、网上银行等网络服务不可或缺.特别是以 JEE 应用为代表的事务型 Web 应用,已经成为网络服务的主要提供方式.然而,面对复杂、难控的 Internet 环境以及 Web 应用本身的复杂性,特定环境下出现的上下文异常[1]难以避免,例如某个请求引发未经测试的内存泄露,或是在某种访问序列下并发线程竞争共享资源造成的死锁等.这些异常往往会在一段时间后引发系统失效,从而可能引起巨大的经济损失[2].据 Tellme Networks 报告指出,失效恢复时间的 75%用于检测失效,18%用于诊断失效[3].另有研究表明,早期检测到失效能够缓解或阻止 65%的失效发生[4].因此,针对事务型 Web 应用的异常检测,是消除或缓解服务失效的有效手段,对于提高网络应用的可靠性非常关键.

然而,已有的 Web 应用在线异常检测方法难以适应开放的 Internet 环境.基于信号的方法[5−7]事先定义错误发生时出现的信号,在运行过程中将观测到的现象与异常信号进行匹配.这种方法描述异常表现较为困难,特别是不能识别此前未定义的异常现象.基于执行路径的方法通过监测框架跟踪请求处理路径[8],路径偏离正常路径则为异常[9].但用户访问模式不断变化,会导致组件交互行为的改变[5].基于度量的方法建立度量之间的稳定关联关系[7,10],检测关联是否被打破.这种方法无法检测独立度量;同时,度量间关联关系也会随时间和负载发生变化[11].基于规则的方法通过设置各度量的阈值,当监测到的度量超出阈值时,根据规则执行预定动作.这种方法在工业产品中被广泛采用,如 IBM Tivoli,HP OpenView,EMC Smarts 等,但系统度量数量繁多,人工设置困难.

值得关注的是,Web 应用的异常很大程度上受负载变化的影响,而负载具有较强的动态性和周期性,呈现出 time-of-day 或 month-of-year 效应[12−14].如文献[15]对一个大型 Web 应用的负载进行分析,其日志表明,负载在每天不同时段和每周不同日期有不同的表现.而在不同负载状况下,请求处理路径和系统度量均会发生变化[11,16].

因此,本文提出一种基于负载模式识别的 Web 应用在线异常检测方法.该方法面向事务型 Web 应用,通过增量式聚类在线建立负载模式及其对应的度量空间;对当前负载进行模式识别,基于其对应的度量空间,利用局部异常因数检测系统异常状态和量化异常程度;利用学生 t 测试方法计算各度量的异常值以定位问题原因.实验结果表明,本文所提方法能够有效识别负载模式,检测 Web 应用典型错误并定位异常度量.

本文的主要贡献在于:

(1) 提出一种刻画 Web 应用负载以及基于增量式聚类在线识别负载模式的方法;
(2) 提出针对负载模式进行异常检测的方法,有效解决了负载动态变化引起的难以准确检测异常的问题;
(3) 提出利用局部异常因数检测 Web 应用异常状态的方法,其能够考虑度量间关联关系,并且量化异常严重程度.

本文第 1 节给出 Web 应用运行时状态的刻画方法.第 2 节提出基于增量式聚类的在线识别 Web 应用负载模式的方法.第 3 节给出基于局部异常因数的异常状态检测以及基于学生 t 测试的异常度量定位方法.第 4 节通过实验验证本方法的有效性.第 5 节对相关工作进行比较.最后,第 6 节总结全文并提出未来工作.

1 Web 应用运行时状态的刻画

事务型 Web 应用通常由 Web 应用服务器和数据库构成,其中,Web 应用服务器处理用户请求,通过容器提供应用组件的运行环境,并访问数据库.客户通过浏览器发出一系列 HTTP 请求以提取 Web 页面,每个事务就是完成一次请求的过程.通常,事务请求对象是一个 Web 页面,由 Web 应用组件(如 Java servlets,JavaServer Page,EJB 等)动态产生的 HTML 文件和嵌入式对象组成,服务器端通过调用 Web 应用组件来处理请求.

首先,我们需要刻画 Web 应用的工作负载.在 Web 应用中,工作负载通常由会话组成.即,在访问站点的过程中,由一个用户发起的不同类型请求的序列,根据会话特征可以分为不同的访问模式.我们借鉴客户行为模型图(custom behavior model graph,简称 CBMG)的思想描述工作负载的访问模式[17],其中,每个请求作为状态,用节点来表示,通过图中各状态间的转移概率表示访问模式.通过周期性统计组件间调用频率,以刻画 Web 应用的负载状况.这不仅反映了访问模式,而且表现了请求密度,定义 1 引入了负载向量以刻画运行时的工作负载.

定义 1(*t* 时刻的负载向量). $LV_t=\{i_1,i_2,\ldots,i_m,\ldots,i_{k\times k}\}$,$LV_t$ 由请求矩阵 RM_t 转换得到.其中,$i_m=j_{ab}$,$m=(a-1)\times k+b$,j_{ab} 为会话中用户调用组件 a 后再调用组件 b 的次数,k 为系统中组件的总数.

$$RM_t=\begin{bmatrix} j_{11} & j_{12} & \cdots & j_{1k} \\ j_{21} & j_{22} & \cdots & j_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ j_{k1} & j_{k2} & \cdots & j_{kk} \end{bmatrix}.$$

为了追踪请求序列,用户请求由会话标识和请求类型⟨*SessionID*,*type*⟩构成,其中,会话标识(*SessionID*)用以确定请求所属用户,请求类型(*type*)为调用的应用组件.下面给出工作负载刻画算法.如图 1 所示,对于客户发起的请求,首先确定是否是新会话,如果是,则为其创建会话(第 3 行~第 6 行);如果是已有会话,则确定是否超时:如果超时,则返回超时信息,并且将会话关闭(第 13 行~第 16 行);若未超时,则查找此会话上次请求类型,更新负载矩阵和会话信息(第 9 行~第 12 行).当经历一个周期后进行统计,根据负载矩阵生成负载向量(第 18 行~第 23 行).

Algorithm 1. Workload characterization.
Input: Request; monitoring period: *mp*;
Output: Load vector at time *t*: lv_t.
1. Get session ID, *sID* from request;
2. Get request type, *rType* from request;
3. **IF** (*sID*==*NULL*)
4. Create new session *sID*;
5. Insert ⟨*sID*,*rType*⟩ in session set, *ss*;
6. **ENDIF**
7. **ELSE**
8. Find *sID* from session set *ss*;
9. **IF**(*sID* is not timeout)
10. Get last request type *lrType* from *ss* with *sID*;
11. Update matrix element *RM*[*lrType*,*rType*]++;
12. **ENDIF**
13. **Else**
14. Clear session *sID*;
15. Update ⟨*sID*,*rType*⟩ in session set, *ss*;
16. **ENDELSE**
17. **ENDELSE**
18. Get last time *lt*;
19. Get current time *ct*;
20. **IF**(*ct*−*lt*>*mp*)
21. $LV_t=\{l_1,l_2,\ldots,l_m,\ldots,l_{k\times k}\}$, $m=(i-1)\times k+j$ in matrix *RM*[*i*,*j*];
22. Reset matrix *RM*;
23. **ENDIF**

Fig.1 Algorithm for workload characterization

图 1 负载刻画算法

Web 应用组件的调用序列和请求密度反映了工作负载,而 Web 应用组件执行造成物理资源的消耗表现为度量的变化,如 CPU、内存、网络利用率和磁盘读写频率等.因此,工作负载与系统度量之间存在着对应关系,Web 应用的运行时状态由工作负载和系统度量共同刻画,由定义 2 给出.

定义 2(*t* 时刻的运行时状态). $RS_t=\langle LV_t,MV_t\rangle$,其中,$LV_t$ 为负载向量;MV_t 为度量向量,$MV_t=\{m_1,m_2,\ldots,m_i,\ldots,m_k\}$,$m_i$ 为监测的第 i 种度量值,如 CPU、内存、网络利用率、磁盘读写频率等,k 为度量数量.

2 基于增量式聚类的在线负载模式识别

在实时监测得到 Web 应用运行时状态序列之后,具有相似资源需求的负载划分为一种负载模式,符合该负载模式的请求具有相似的请求密度和用户访问模式,可以通过负载向量间的相似度来衡量.聚类是一种非监督学习方法,适合在线分类,且能够表示高维向量,因而我们利用其识别负载模式.网络环境中,负载模式具有动态性,难以预知,需要以增量式聚类在线学习.同时,负载模式在一段时间内应具有稳定性,负载波动引起生成新的聚类应当作为负载异常引起注意.如图 2 所示,我们提出了基于增量式聚类的在线负载模式识别算法.初始化聚

类,由最初得到监测点的运行时状态创建聚类(第 1 行~第 5 行).当得到新的监测点时,计算当前负载向量与已知聚类中心点的最小距离:如果大于门限值,则由此监测点的运行时状态创建新聚类(第 6 行~第 10 行);否则,基于此聚类的度量空间进行异常检测,将该运行状态点加入聚类,更新中心点(第 12 行~第 18 行).遍历聚类,当滑动窗口已满时:若有α(0%<α<100%)以上的点落入新创建聚类,则认为其代表一种新的负载模式;否则,认为该聚类是由于负载波动所引起的负载异常,并删除该聚类(第 20 行~第 26 行).

Algorithm 2. Online workload pattern recognition with incremental clustering.
Input: Monitoring points: $mp_s=\{mp_0,mp_1,\ldots,mp_t,\ldots\}$, where $mp_t=\{lv_t,mv_t\}$, lv_t is load vector and mv_t is metric vector slide window: *sw*; point distance: *pdist*; min cluster size: *cz*; anomaly score: *as*;
Output: Anomaly points: $aps=\{ap_i,ap_j,\ldots\}$, where $ap_i=\{lof_i,\{m_1,\ldots,m_k,\ldots,m_n\}\}$, m_k is the *k*th metric anomaly score.

```
1.  Get runtime mp_t from monitoring system at time t;
2.  IF clusters==NULL
3.      Initialize cluster with mp_t;
4.      Set clusterNum as 0, and Set createPointNum of cluster as 0;
5.  ENDIF
6.  Select the minimum distance d[j], and get the cluster ID, j;
7.  IF d[j]>pdist
8.      Initialize cluster with mp_t;
9.      Set clusterNum as clusterNum++, and Set createPointNum of cluster as currentPointNum;
10. ENDIF
11. ELSE
12.     Compute lof of mv_t with metric space of cluster j;
13.     IF (lof >as)
14.         Insert mp_t in aps;
15.         Compute ap_i={lof_1,{m_1,…,m_k,…,m_n}} and sort m_i in sequence;
16.         Break;
17.     ENDIF
18.     Insert mp_t and update mean in Cluster j;
19. ENDELSE
20. FOR (i=0; i<clusters.size(·); i++)   //traverse existing clusters
21.     Compute distance between lv_t and mean of cluster i;
22.     IF (currentPointNum−createPointNum>sw) && (cluster[i].size(·)<cz)
23.         Delete cluster i from clusters;
24.         Alert visit anomaly;
25.     ENDIF
26. ENDFOR
```

Fig.2 Algorithm for online workload pattern recognition with incremental clustering

图 2 基于增量式聚类的在线负载模式识别算法

在聚类过程中,通过计算负载向量与聚类中心间的距离,得到其与聚类的相似度.我们选取 Mahalanobis 距离[18]是考虑到其具有两点优势:首先,与常见的 *n* 维 Euclidean 距离[18]相比,Mahalanobis 距离考虑到了点分布的特点,减少了各维度波动幅度不同所造成的测量偏差;同时,考虑到了各变量间的关联关系.而在 Web 应用中,同一个负载模式中各数值间存在关联关系.例如在 CBMG 中,$l_{select,add}=0.3\times l_{bowse,select}=0.3\times 0.2\times l_{home,browse}$[19].由此可见,$l_{select,add}$, $l_{bowse,select}$ 和 $l_{home,browse}$ 之间存在着关联关系.负载向量与聚类间距离的计算方法由定义 3 给出.

定义 3(负载向量 LV_i 与聚类 j 间的距离). $PD_{ij}=\sqrt{(LV_i-LM_j)A^{-1}(LV_i-LM_j)^T}$, 其中,

(1) 聚类 *i* 的中心点 $LM_i=\frac{1}{L}\sum_{k=1}^{L}LV_k$, LV_k 为聚类 *i* 中第 *k* 个负载向量,*L* 为聚类大小;

(2) *A* 为聚类中负载向量的各维度变量 l_i 的协方差矩阵,以表示各维度变量间的关联关系,可由下式计算得到:

$$A=\begin{bmatrix} cov(l_1,l_1) & cov(l_1,l_2) & \cdots & cov(l_1,l_k) \\ cov(l_2,l_1) & cov(l_2,l_2) & \cdots & cov(l_2,l_k) \\ \vdots & \vdots & \ddots & \vdots \\ cov(l_k,l_1) & cov(l_k,l_2) & \cdots & cov(l_k,l_k) \end{bmatrix},$$

其中,$cov(l_i,l_j)=E[(l_i-E(l_i))(l_j-E(l_j))]$,*k* 为组件数.

点与聚类间距离的门限值是算法的关键参数,随着距离的增加,聚类的粒度加大,每个聚类对应于更大的度量空间,每个聚类可能会包括多个相似的负载模式,系统召回率也会随之下降;相反地,如果距离过小,更多的点会形成独立的聚类,而可能被认为负载异常.因此,该距离应该与负载波动情况相适应,使得同一负载模式被归于尽量少的聚类,同时,不同的负载模式归于不同的聚类.我们利用离线模拟负载模式的方式,基于统计的方法计算得到该值.如图 3 所示,设定某种负载模式并离线模拟产生负载,监测得到此条件下负载向量集合(第 1 行);计算向量各维度均值,得到集合中心点(第 2 行);计算各向量距离中心点的距离,由此得到样本标准差(第 3 行~第 6 行);*pdist* 为待定值进行 *t* 变换,使其等于 *t* 分布的α置信点(第 7 行、第 8 行).设置多种负载模式,重复执行该算法以求得距离均值.

Algorithm 3. Threshold distance calculation between point and cluster.
Input: α confidence interval (0%<α<100%);
Output: Point distance: *pdist*.

1. $LVS=\{lv_1,lv_2,\ldots,lv_i,\ldots,lv_L\}$; //Simulate a load pattern offline, and get the load vector set
2. $LV_0=\frac{1}{L}\sum_{k=1}^{L}LV_k$; //Calculate the set mean vector
3. **FOR** (i=1; i<=L; i++)
4. $d_i=\sqrt{\sum_{k=1}^{n}(m_{i,k}-m_{0,k})^2}$; //Calculate the distance between LV_i and mean where $m_{i,k}$ is the kth dimension in LV_i, and n is the size of vector dimension;
5. **ENDFOR**
6. $S=\sqrt{\frac{1}{L-1}\sum_{i=1}^{L}d_i^2}$; //Calculate the standard deviation of distances
7. $t(d)=\frac{d}{s/\sqrt{L}}\sim t(L-1)$; //$t$-transform d, and it meets t distribution with L−1 degrees of freedom
8. $t(pdist)=t_{\alpha/2}(L-1)$, and then $pdist=\frac{S\times t_{\alpha/2}(L-1)}{\sqrt{L}}$ //Get the (1−α) confidence interval

Fig.3 Algorithm for threshold distance calculation between point and cluster
图 3 点与聚类间距离门限值计算算法

3 异常检测方法

系统运行过程中,监测负载向量序列得到各种负载模式,其中每种负载模式对应于一个度量空间,如图 4 所示.负载模式与度量空间的对应关系由稳定运行状态来表示,由定义 4 给出.

定义 4(稳定运行状态 *i*). $SS_i=\{LM_i,A_i,MS_i\}$,其中,LM_i 为聚类 i 的中心点;A_i 为聚类 i 中负载向量各维度的协方差矩阵;MS_i 为负载模式 i 的度量空间,由度量向量集合$\{MV_1,MV_2,\ldots,MV_i,\ldots\}$和 Euclidean 距离构成.

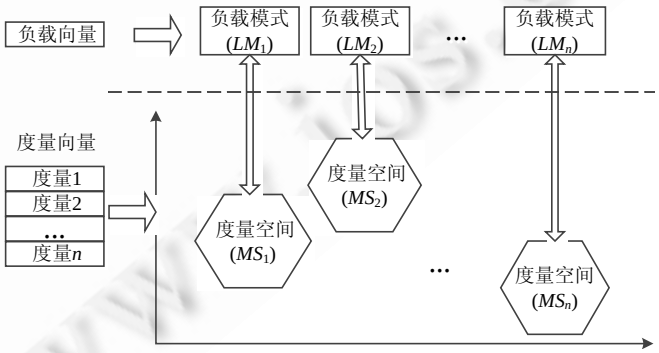


Fig.4 Workload pattern and metric space
图 4 负载模式与度量空间关联关系

异常检测方法如图 5 所示,在运行过程中,我们将运行时状态的负载向量与已知负载模式进行匹配,从而获得其对应的度量空间,利用局部异常因数(local outlier factor,简称 LOF)[20]根据度量空间计算当前度量向量的异常分数;当判定异常时,则利用学生 *t* 测试对各个度量分别考察,以定位异常度量.我们通过监测度量的变化以检测 Web 应用运行过程中出现的异常状态,同时,可以通过负载模式识别检测出负载剧烈波动而引起的负载异常.

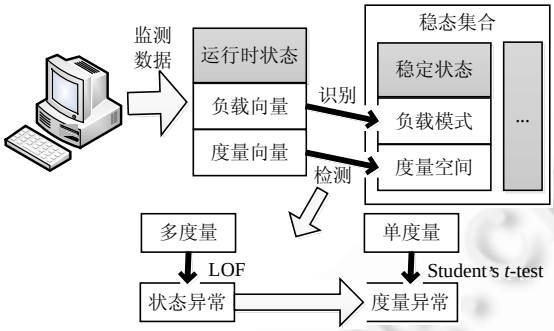


Fig.5 Anomaly detection approach

图 5 异常检测方法

3.1 基于LOF的异常状态检测

LOF 是一种基于局部密度的 *K*NN 方法,LOF 分数等于数据实例其距离最近的 *k* 个邻居所在区域的密度与其所在区域密度的比例.对于一个正常数据实例,其局部密度与其邻居的密度相似;然而对于异常的实例,其局部密度低于最近邻居,从而得到较高的 LOF 分数.

我们利用 LOF 进行异常检测,缘于其具有以下优势:

- 首先,能够表现度量间的相互关联.由于系统的运行状态是由多个度量共同刻画的,而且其相互间存在着关联关系[10]会对异常检测结果造成影响[21],这可以通过度量向量在度量空间中的相对位置来体现;
- 其次,能够适应 Web 应用负载波动的需要.在不同的负载模式下,度量向量会发生不同程度的波动,如并发 100 时,CPU 的利用率是 0.2,出现 20%的波动会造成 0.04 的变化;而并发 400 时,CPU 的利用率是 0.8,若同样出现 20%的波动,则会造成 0.16 的变化.由于 LOF 是基于局部密度的方法,可以解决该问题;
- 再次,减轻噪音数据造成的干扰.在线异常检测采用无监督学习,即无需人工标记状态正常或异常,这就有可能在建立度量空间时引入噪音数据.由于 LOF 是基于 *K*NN 的方法,检测时需要考虑到与其距离最近的 *k* 个邻居,可能引入的部分异常数据并不会对最终的检测结果造成太大影响.同时,随着度量空间中度量向量数量的增多和 *k* 值的增大,异常数据造成的影响会随之减小;
- 最后,可以量化各负载模式下度量向量的异常程度,以表示异常的严重程度.

在 Web 应用系统中,不同度量的测量单位和范围往往是不同的,为了消除变量的量纲效应,使每个变量都具有同等的表现力,我们首先利用 Z-Score 对度量向量进行标准化处理,而后利用定义 5 计算 LOF 分数.

定义 5(局部异常因数). $LOF_k(p)=\sum_{o\in N_k(p)}\frac{lrd_k(o)}{lrd_k(p)}\Big/|N_k(p)|$,其中,

(1) *k* 邻域距离(*k*-distance).对 $p\in D$,*p* 的 *k* 邻域距离定义为 $dist_k(p)$,是与 *p* 距离最近的 *k* 个点的最大距离;

(2) *k* 邻域点集.对 $p\in D$,与 *p* 距离最近的 *k* 个点的集合 $N_k(p)=\{q\in D\backslash\{p\}|d(p,q)\leqslant dist_k(p)\}$;

(3) 可达距离.对 $q\in D$,$reach\text{-}dist_k(p,q)=\max\{dist_k(q),d(p,q)\}$;

(4) 局部可达密度. $lrd_k(p)=1\Big/\frac{\sum_{o\in N_k(p)}reach\text{-}dist_k(p,o)}{|N_k(p)|}$.

当 $LOF_k(p)$在 1 附近时,*p* 认为是正常点;当 $LOF_k(p)>1$ 时,我们可以根据 LOF 来判定异常程度.在利用 LOF 进行异常检测时,相邻点数量也是需要考虑的因素,本文第 4.5 节实验结果表明,当其大于等于 5 时,检测结果波

动幅度很小.

3.2 基于学生*t*测试的异常度量定位

在检测到异常状态之后,需要计算出各度量的异常分数,以定位异常度量.从统计学角度考虑,数据的分布应符合一定的统计模型,正常数据会处于模型的高概率区域,而异常数据会处于低概率区域.由于基于统计的方法难以表示各变量之间复杂多样的关联关系,因此,我们只是在检测得到异常状态之后用以定位异常度量.学生 t 测试方法通常用来检验单变量数据是否符合已知的分布情况.假设各度量数据符合高斯分布[20],在检测出异常状态的度量向量 $MV(m_1,m_2,\ldots,m_k)$后,对于每个度量值进行 t 转换,使其均符合 t 分布,t 值由定义 6 给出.

定义 6(度量 *i* 的 *t* 值). $t(m_i)=\dfrac{m_i-\overline{m_i}}{S_i/\sqrt{L}}\sim t(L-1)$, 其中, $\overline{m_i}=\dfrac{1}{L}\sum_{j=1}^{L}m_{ij}$, $S_i=\sqrt{\dfrac{1}{n-1}\sum_{j=1}^{n}(m_{ij}-\overline{m_i})^2}$, L 为度量空间中度量向量的数量.

计算得到 $n\to\infty$时的$(1-\alpha)$置信点,当 $t(m_i)>t_{\alpha/2}(L-1)$,则认为度量出现异常,并将此作为该维度度量的度量异常值(metric anomaly value,简称 MAV).由于各个度量的异常程度有所差别,*MAV* 越大,表明其与均值的偏差越大,异常程度就越严重.

对各度量异常值进行排序,进一步确定各度量异常的严重程度:$MAV(m_i)\geqslant MAV(m_j)\geqslant\ldots\geqslant MAV(m_k)$,以定位异常原因.

3.3 时间复杂度分析

本文所提出的异常检测方法的计算复杂度主要集中在负载模式识别、异常因数计算和稳定状态建立这 3 个方面.对于识别负载模式的聚类算法,当监测数据到来时,需要计算当前点与 k 个聚类中心的最小距离,时间复杂度为 $o(k)$;若加入该聚类,则计算包含 m 个点的聚类中心,时间复杂度为 $o(m)$.因此,该部分时间复杂度为 $o(n)$.对于异常检测的 LOF 算法,时间复杂度为 $o(n\times K\text{NN}$ 查找时间),文中采用 SR-tree[22]查找最近邻居,时间复杂度为 $o(\log n)$,LOF 的时间复杂度为 $o(n\log n)$.对于稳定运行状态建立,需要计算方差和协方差矩阵,时间复杂度为 $o(n^3)$,其中,n 为应用组件数量.由于负载模式数量有限,且聚类一旦形成就无需重复计算,这部分计算复杂度并不影响方法整体性能.由以上分析可知,本文提出的方法并不会带来太大的计算开销,适用于在线异常检测.

4 实验与评价

4.1 实验设置

为了评价本文所提出的异常检测方法的有效性,我们建立了基于 JEE 的典型 Web 应用实验环境,其拓扑结构如图 6 所示,软、硬件配置则在表 1 中给出.其中,负载发生器模拟动态变化的真实工作负载,Web 应用部署在 Web 应用服务器上,通过访问后端的数据库获取数据,异常检测器部署在独立的机器上,用以搜集 Web 应用服务器的监测数据并进行异常检测.负载向量采用 AOP[23]方法,利用 AspectJ[24]框架对应用服务器会话管理器注入监测点得到,度量数据由 Windows 操作系统管理工具提供的性能监控接口获得,从 CPU(% Processor time,% user time)、内存(available mbytes,page faults/sec)、网络(bytes received/sec, bytes sent/sec,segments retransmitted/sec)、磁盘(avg. disk bytes/transfer, avg. disk sec/transfer, avg. disk write queue length)这 4 个主要方面选取了 10 项度量.本方法还可以选取更多的应用服务器和数据库度量,以全面、深入地监测和分析系统状况.为了便于问题的阐述,这里仅选取 Web 应用服务器中具有代表性的几项.

本文提出的方法具有应用无关性,可以广泛应用于各种类型的 Web 应用中,这里选取 TPC-W 基准测试[19]作为实验基础.TPC-W 是被广泛使用的模拟在线电子书店的 Web 应用标准测试规范,通过在可控的环境下执行一系列电子商务典型事务来评价 Web 应用系统.TPC-W 提供了网上书店常用的功能,包括图书的浏览、查询及订购等.围绕这些功能,TPC-W 定义了 14 种事务交互,分为浏览和订单两大类.按照浏览和订单类事务的比例,TPC-W 将测试分为购物、浏览和订单这 3 种混合模式.由于可以通过调整并发数量、访问模式等参数来模拟

事务型 Web 应用多种真实的负载模式,我们用以验证负载模式识别的效果以及负载模式对异常检测的影响.同时,该基准测试是国际上产业界和学术界广泛认可和采用的测试基准,具有较好的典型性和可重现性,如 IBM 与 Oracle 等公司多次基于此标准进行测试,学术界也普遍采用其进行方法验证.因此,本实验选用了我们研发的符合 TPC-W 规范的基准测试套件 Bench4Q[25],应用服务器 OnceAS[26].数据库采用默认设置,即 10 000 件商品和 1 440 000 个用户.Bench4Q 客户端模拟器产生典型的工作负载,建立并发客户连接,每个客户模拟通用场景的会话,包括一系列请求,如创建账户、查询关键词、浏览商品、订购和购买等.在实际应用场景中,工作负载模式和并发数量是动态变化的,为了模拟这种效果,负载发生器实现 TPC-W 规定 3 种负载模式的动态切换,并且通过 EB 数的改变来实现并发用户数量的变化.

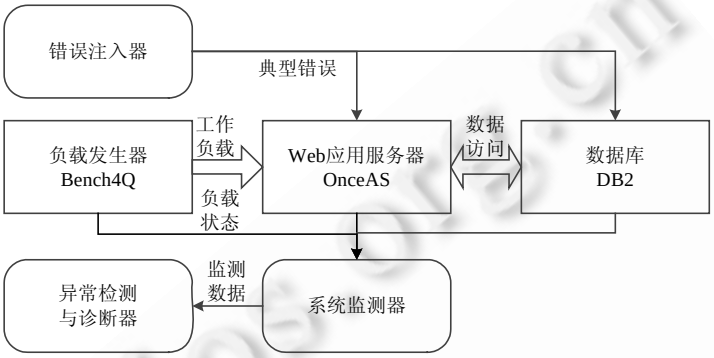


Fig.6 Experiment architecture

图 6 实验环境拓扑结构图

Table 1 Software and hardware environment

表 1 实验环境

组件	处理器	内存
Application server (OnceAS)	Intel® Xeon™ 2.5GHz (8 CPUs)	4G
Database (DB2)	Intel® Xeon™ 3.0GHz (4 CPUs)	2G
Clients (emulated browsers)	Intel® Core™ 2 Duo 2.33GHz (2 CPUs)	2G
Others (injector/monitor/detector)	Intel® Core™ 2 Duo 2.33GHz (2 CPUs)	2G

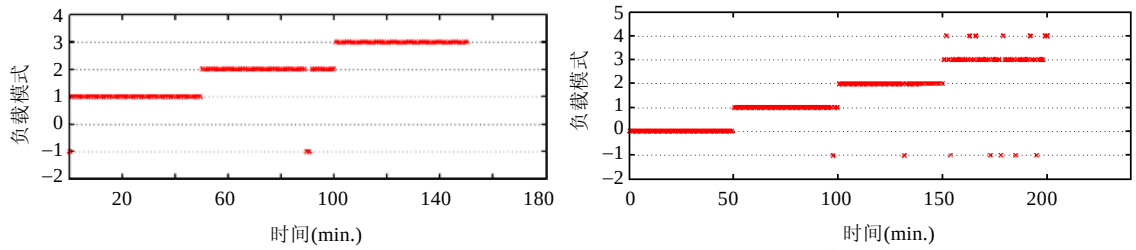
4.2 负载模式的识别

负载模式由访问模式和并发用户数量两个方面决定,我们对其分别考虑以评价基于增量式聚类的在线识别负载模式的有效性.

首先,负载发生器模拟并发数量不变、访问模式改变的情况.负载并发为 200,在 Browsing,Shopping 和 Ordering 这 3 种访问模式间动态变化.如图 7(a)所示,聚类 1 表示 Browsing 模式,聚类 2 表示 Shopping 模式,聚类 3 表示 Ordering 模式,其中,归为–1 的点为判定的负载异常.

而后,负载发生器模拟访问模式不变、并发数量变化的情况.负载在 Browsing 访问模式下,并发以 100,200,300,400 动态变化,每种负载持续 50 分钟,如图 7(b)所示,方法能够正确地将负载模式划分为 0,1,2,3,4 这 5 种聚类.并发在 400 时,大部分点在聚类 3,而部分点归到了聚类 4,同时产生了负载异常.这是由于,并发 400 时,系统性能即将达到瓶颈,共享资源竞争加剧而造成请求处理速度波动较大,部分偏离中心较大的点形成聚类 4,如图 8 所示.因此,聚类 0~聚类 2 分别表示并发为 100,200,300 的负载模式,聚类 3 和聚类 4 共同表示并发为 400 的负载模式.将一种工作负载划分为多个聚类,使得异常检测以细粒度进行,并不影响检测结果.

实验结果表明,我们所提出的聚类算法能够正确识别出访问模式和并发用户数量变化所产生的不同负载模式.



(a) 访问模式变化情况下的负载模式识别 (b) 并发用户变化情况下的负载模式识别

Fig.7 Workload pattern recognition

图 7 负载模式识别

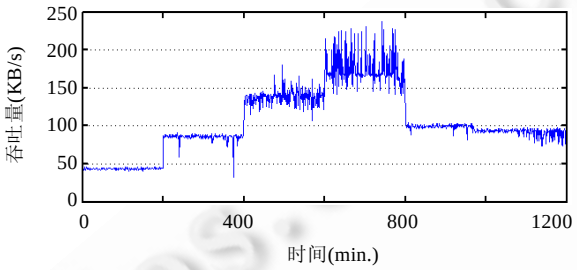


Fig.8 Dynamic workload generation

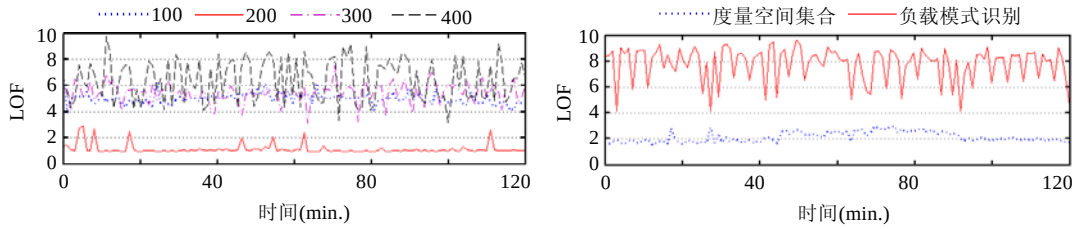
图 8 工作负载的动态变化

4.3 负载模式对异常检测的影响

为了评价负载模式识别对异常检测效果的影响,分为两个阶段进行实验.在正常状态下运行 Bench4Q 应用,模拟产生负载以建立负载模式及其度量空间.图 8 描述了负载动态变化情况,y 轴为负载发生器单位时间发送请求的吞吐率,表示并发数量的变化.负载由 Browsing 访问模式下并发数量分别为 100,200,300,400 的负载和并发 200 情况下 Shopping 和 Ordering 访问模式的负载组成,每组负载运行 100 分钟.由于 Bench4Q 基于 Close 模式,即客户端发送请求速率受服务器端请求处理速率制约,并发 400 时,服务器端的性能接近瓶颈,从而影响到客户端负载的波动.

首先,评价仅考虑一种负载模式的异常检测效果.这里采用 Browsing 并发 200 的负载模式所对应的度量空间进行检测.系统运行在正常状态下,负载发生器模拟产生 Browsing 访问模式的负载,改变并发数量.如图 9(a)所示,并发 200 时,LOF 在 1 附近;并发 100 和 300 时,LOF 在 4~6 之间波动;并发 400 时,LOF 在 4~8 之间波动.因为训练过程未能考虑所有负载模式下产生的度量空间,新负载模式下的正常度量向量也会偏离训练得到的度量空间,使得 LOF 较高从而造成误判.由此可见,采用基于在线负载模式识别的方法由于考虑了多种负载,因而可以提高准确率(正确辨别的错误数/总共辨别的错误数).

而后,评价对负载模式不加区分的异常检测效果.在 Bench4Q 中注入 CPU 密集循环的错误,负载发生器模拟 Browsing 并发 200 的负载模式,将基于负载模式识别与不区分负载模式的检测效果进行比较.如图 9(b)所示,基于负载模式识别的方法检测得到的 LOF 在 6~9 之间,异常效果明显;而根据度量空间集合检测得到的 LOF 在 2 附近,异常不明显.这是因为度量空间集合扩大了正常度量空间的范围,使得原本异常的度量向量也被包含在了正常的空间当中,造成异常难以被检测出来.由此可见,采用基于负载模式识别的方法由于区别负载可以提高召回率(正确辨别的错误数/总共的错数).



(a) 不同并发情况下的 LOF (b) 不同度量空间情况下的 LOF

Fig.9 Impact of workload pattern on anomaly detection

图 9 负载模式对异常检测效果的影响

4.4 异常检测效果

根据如图 8 所示的负载,建立负载模式及与之对应的度量空间进行异常检测,从系统状态异常检测和度量异常定位两个角度来验证方法的有效性.由于实验涉及太多度量值,不易在图中全部清晰地表示,我们仅从内存、网络、磁盘、处理器这 4 个方面考察,每方面各选取一个度量,包括可用内存、每秒接收字节数、每秒磁盘传输数、处理器利用率等.采用 Browsing 访问模式,并发为 300 的负载模式进行测试,每组测试进行 2 个小时,每分钟进行 1 次检测.首先是系统在正常状态下,即运行未注入错误的 Bench4Q 应用,以验证该方法不会将正常状态错误地认为异常状态.而后,在 Bench4Q 应用中注入 Web 应用的典型错误,实时搜集监测数据并进行分析.实验中,LOF 门限值设定为经验值 2,在 t 测试中,$n\to\infty$的 95%置信点为 $t_{0.05}(n)=1.645$,因此,当度量的 MAV 大于此值时认为异常.

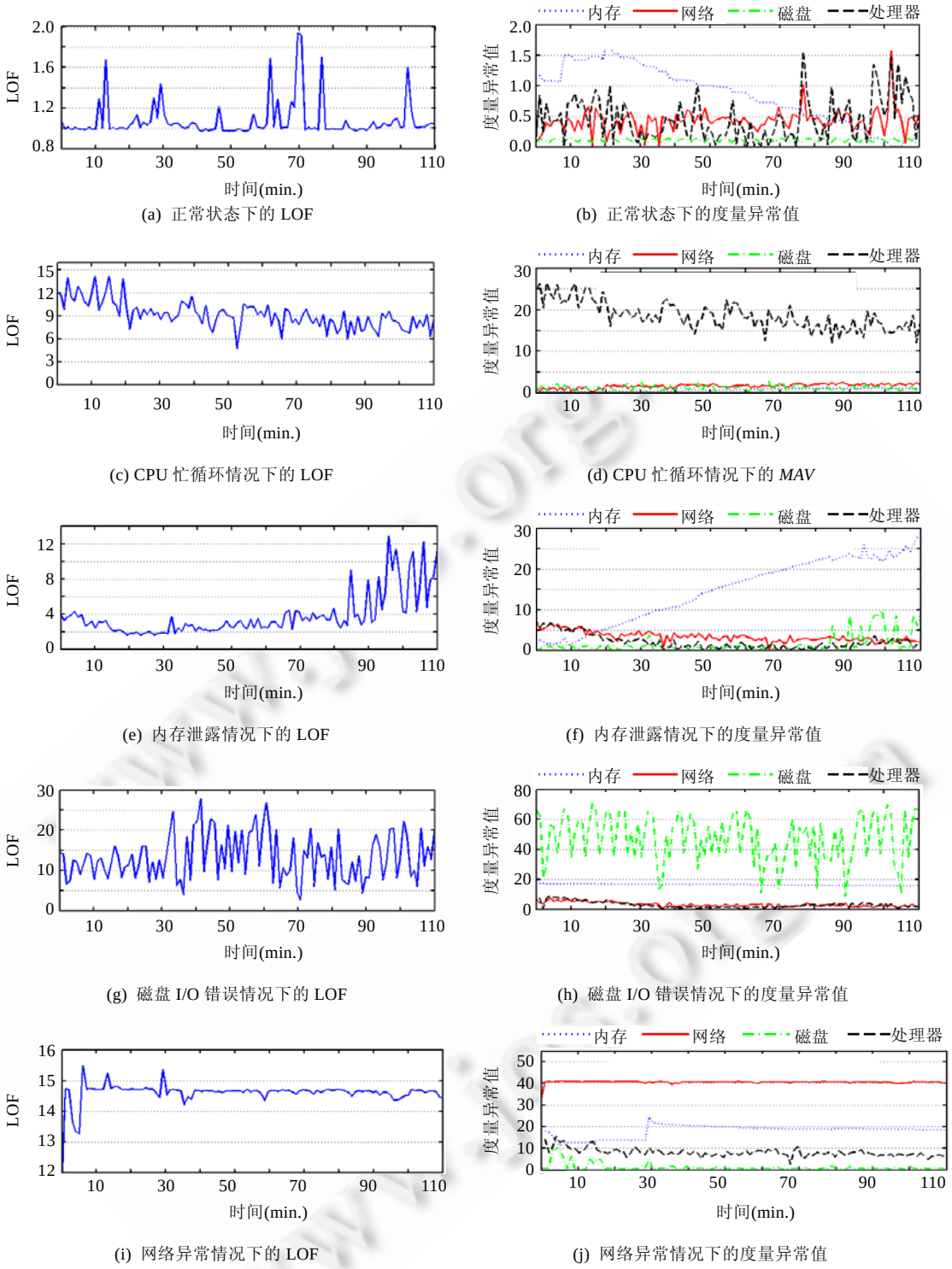
在错误注入类型方面,由于目前尚无评价 Web 应用异常检测方法有效性的 Benchmark,为了保证注入错误的典型性,我们参考已有的研究成果.文献[27]通过对众多真实 Web 应用故障进行分析,研究了 Web 应用出现故障的常见原因,我们从中选取典型错误.据此,我们选取内存泄露、资源密集处理、磁盘 I/O 错误和网络异常这 4 种错误进行实验,在 Bench4Q 应用的 Servlet 中分别随机注入这些错误.受篇幅限制,这些错误仅是示例,我们的方法也同样适用于检测引起的系统度量变化的其他异常.

正常操作:如图 10(a)所示,Web 应用处在正常运行状态的 120 个监测点中,LOF 大多处于 1.4 以下,均低于设定值.如图 10(b)所示,度量异常值(MAV)大多小于 1.6,只有 1 个度量点可被认为异常,其他均在正常范围之内,因此该方法具有较低的误报率.

CPU 密集循环:一段时间内,线程陷入循环等待或死循环占用 CPU 资源,造成请求得不到及时处理,如自旋锁等.每次调用注入该错误的 Web 应用组件,则执行多次循环计算操作.如图 10(c)所示,我们可以看到,LOF 大致处于 6~12 而判定为异常.同样,如图 10(d)所示,我们可以看到度量向量中各 MAV,其中,CPU MAV 处于 15~25 之间,其他度量明显低于 CPU,因此可以确定是与处理器相关的问题造成了 Web 应用出现异常.

内存泄露:这将会逐渐耗尽系统资源,并最终导致崩溃.在实验中,每次调用注入该错误的 Web 应用组件则分配 1K Bytes 内存空间且引用静态变量,造成该对象不被垃圾回收.如图 10(e)所示,总体上随着时间的增加,LOF 值逐渐增加且均高于 2 而判定为异常,这是因为内存泄露使问题逐渐严重的过程.同样,如图 10(f)所示,度量向量中各 MAV 中内存 MAV 逐渐增加,并且均明显高于其他度量,因此可以确定是内存问题造成系统异常的出现.

磁盘 I/O 错误:在 Web 应用中,磁盘访问通常由负载密度和用户行为所决定.然而,某些应用层错误也会引入在 I/O 访问方面的性能瓶颈,并且在很大程度上改变磁盘访问模式.每次调用注入该错误的 Web 应用组件就会引入额外的磁盘读写操作.如图 10(g)所示,可以注意到,LOF 值大致在 5~25 之间波动较大,这是由于操作系统对磁盘并不是立即进行读写操作.如图 10(h)所示,磁盘 I/O MAV 明显高于其他度量,除了磁盘 I/O 度量以外,内存度量也会受到影响.这是由于,频繁磁盘操作所读写的数据需要缓存在内存中因而占用更多内存空间.



(a) 正常状态下的 LOF (b) 正常状态下的度量异常值

(c) CPU 忙循环情况下的 LOF (d) CPU 忙循环情况下的 *MAV*

(e) 内存泄露情况下的 LOF (f) 内存泄露情况下的度量异常值

(g) 磁盘 I/O 错误情况下的 LOF (h) 磁盘 I/O 错误情况下的度量异常值

(i) 网络异常情况下的 LOF (j) 网络异常情况下的度量异常值

Fig.10 Effectiveness of anomaly detection

图 10 异常检测效果

网络异常:由于服务器必须从众多不明的客户端接受请求,网络服务对于攻击是相当脆弱的.例如,Web 应用

中存在漏洞或被注入恶意代码而向外发送报文,从而占用网络带宽等.每次调用注入该错误的 Web 应用组件就会执行多次循环,向局域网中任意 IP 的主机发送 UDP 报文的操作,结果由于网络带宽被占用而造成请求处理迟缓.如图 10(i)所示,LOF 在 15 左右,明显高于正常值,系统可判定为异常状态.从图 10(j)中我们可以观察到,网络、内存和处理器均具有较高的 *MAV*,而网络 *MAV* 最高.由此可见,一种错误往往也会引起其他度量的异常,我们的方法不仅判定度量是否处于异常状态,而且量化异常的严重程度.

通过对 Bench4Q 应用组件中注入典型的 4 种错误,可以看到,我们提出的异常检测方法能够成功地检测,并且准确定位异常度量.通过粗粒度的监测数据是难以准确定位到具体的错误的,如软件中的一行代码,但通过异常检测可以及时发现异常现象以便采取措施,并缩小错误发生原因的范围.而后,可以借助细粒度错误分析方法,如程序分析或日志追踪等,进一步确定错误的具体位置.异常检测的准确程度与监测的粒度相关,而本文所提出的方法基于局部密度算法,具有较好的可扩展性,可以考虑更多的度量,以提供更多的信息帮助错误定位.同时,该方法同样适用于 Web 应用服务器集群,如第 3.3 节中的分析,本方法引入较小的开销适合在线检测场景,因此只需在集群中每个节点上分别进行检测.每个 Web 应用服务器会建立自己独立的度量空间,而不会受到软、硬件环境异构性、其他节点状态和负载均衡算法的影响,从而具有较好的可扩展性和鲁棒性.

4.5 相邻点数量对异常检测的影响

基于局部密度进行异常检测需要考虑与之相邻点所处区域的密度,相邻点数量的选取成为重要的问题.因此,本节通过实验分析相邻点的数量对检测结果造成的影响.采用正常操作、内存泄露、CPU 忙循环、磁盘 I/O 异常、网络异常这 5 类场景,在每种场景中,相邻节点数量取 1,3,5,7,9,11,13 这 7 种值分别进行实验,每次实验结果的 LOF 取均值进行对比.如图 11 所示,相邻点数在 1 和 3 之间差别明显,各种场景下,随着相邻节点数量的增加,LOF 趋于稳定,而大于 5 后的 LOF 差别并不大.因此,检测过程中我们取 5 个相邻节点.

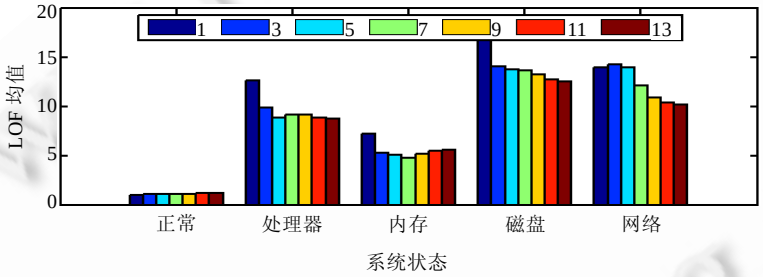


Fig.11 Impact of the number of neighbors

图 11 相邻节点数对 LOF 的影响

5 相关工作

基于信号的方法针对特定错误所表现的异常现象进行检测.该方法事先定义错误发生时出现的异常信号,在运行过程中将观测到的现象与异常信号进行匹配,如果匹配成功,则认为检测到了异常.文献[7]提出基于规则的分类方法预测异常,通过分析在异常发生前频繁出现的事件,推断异常规则.文献[5,6]首先分析属性间的关联关系,基于此建立属性网络图.当失败发生时,属性间的关联关系被打破,将此时的失败状态信息作为失败信号,当失败再次发生时,提取相似实例以分析问题原因.文献[6]基于 Bayesian 模型辨别特定的错误,这种方法要求系统组件以及相互依赖关系编码在 Bayesian 网络.如果已知系统错误及其表现,该方法将会具有较高的准确性和及时性.但是,该方法需要有专家系统辅助分析错误及其异常;同时,对于此前未曾出现的错误或未定义的异常现象则不能够识别.

基于执行路径的异常检测方法应用较广.文献[28]采用上下文无关文法描述请求执行路径,语法符号表示提供服务的组件,语法规则表示请求在组件间的转移概率,当执行路径不能被语法解析时,执行路径即认为是异常的.文献[29]利用聚类,通过 String-edit 距离将请求路径划分为组,请求路径不符合已有聚类时认为是异常.文

献[9]统计组件间调用行为,将组件实例间调用频率作为χ^2测试的参考分布,若组件实际调用与参考分布不符,则表明组件异常.这种方法能够发现应用层异常,但应用场景和用户访问模式的变化会导致组件交互行为的改变,因此这种方法并不适应负载模式动态变化的网络环境.

基于度量的方法无需系统的先验知识因而被广泛关注.文献[30]描述页面点击次数和失败周期的分布,应用χ^2测试和 Naïve 贝叶斯模型检测系统异常.文献[9]提出多变量统计方法,多维度度量被削减为对一个维度的统计,从度量数据中抽取信号和噪音,并且采用 Hotelling T^2和平方预测错误(squared prediction error,简称 SPE)的方法进行测试,当统计值与期望值产生较大偏差时判定系统异常,但此方法在负载模式变化时不具备鲁棒性.

基于度量间稳定关联的异常检测方法近年来有较多研究.文献[10,31]分析在正常状态时度量间存在着稳定的关联关系,当关系被打破时则认为出现了异常,并且提出了基于自动线性回归获得度量间关联关系的方法.文献[32]为了解决建立关联时间复杂度较高的问题,讨论了两种算法以加快发现稳定的度量关联关系.文献[33]比较分析了多种建立度量间关联的线性回归方法.文献[34]针对度量间存在非线性关系的问题,研究了采用高斯混合模型建立度量间关联关系.由于模型参数使用期望最大算法进行估计,两个变量关联性搜索的复杂度是$O(n^2)$.这类方法采用无监督学习系统正常状态,并且可以广泛应用到众多的系统当中.但是,这种方法只能建立两个度量间的关联,而事实上关联关系较为复杂,因而多度量关联的问题无法解决,并且某些独立度量和其他度量间并不存在关联关系.同样,正如文献[11]分析的那样,随着时间的变化或用户行为模式的改变,这种稳定关系也将会被打破.

网络安全中,入侵检测技术用于检测对计算机系统的恶意行为,而异常检测是一种主要方法,它根据用户访问行为的正常程度来判断系统是否发生入侵.基于统计和机器学习的入侵检测方法,对用户正常行为建立模型,当监测到的行为与模型发生偏差时,则认为入侵发生,如文献[35−37].与本文提出的方法思想相近的是 Minnesota 入侵检测系统[38],该系统监测一定时间窗口内的网络流,通过分析网络数据包的包头,根据与最近流比较,计算每个流的 LOF 值作为流的异常值,但该方法只适用于同时存在多个流的情况.文献[39]提出基于 Commute 距离的方法检测异常网络流量,由于其能够同时考虑到监测数据点间距离、点密度和点分布等特点,具有较好的鲁棒性.但其入侵检测的目标是检测用户恶意行为,而难以检测 Web 应用本身错误所引起的异常;监测和分析的对象是网络数据包,如 IP 地址、网络流量、进程状态等与网络请求相关的低层度量,而不关注事务型 Web 应用的负载特点,如应用组件的请求序列、事务完成情况等.

6 总结及未来工作

本文提出一种基于负载模式识别的 Web 应用在线异常检测方法.该方法基于增量式聚类在线识别 Web 应用负载模式,针对负载模式进行异常检测,有效解决了负载动态变化引起异常检测困难的问题.提出了利用 LOF 检测 Web 应用异常,其能够考虑度量间关联,并且量化异常严重程度.实验结果表明,本文所提方法能够准确识别负载模式的变化,有效检测出 Web 应用典型错误所造成的异常状态并定位异常度量.目前,我们的方法仅适用于事务型 Web 应用,能够检测异常状态而难以准确定位错误代码.进一步地,我们将对于由若干台微机、集群系统、盘阵、复杂网络等组成的巨型复杂系统进行有针对性的研究,对所提出的方法进行扩展,使其可应于更为复杂的软件系统.另一方面,结合执行路径分析方法,以应用组件为粒度来定位错误的具体位置.

References:

[1] Chandola V, Banerjee A, Kumar V. Anomaly detection: A survey. ACM Computing Surveys, 2009,41(3):1−58. [doi: 10.1145/1541880.1541882]

[2] Patterson D. A simple way to estimate the cost of downtime. In: Proc. of the 16th Int'l Conf. on System Administration. Philadelphia: Berkeley: USENIX Association, 2002. 185−188.

[3] Chen MY, Accardi A, Kiciman E, Lloyd J, Patterson D, Fox A, Brewer E. Path-Based faliure and evolution management. In: Proc. of the 1st Int'l Symp. on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2004. 23−36.

[4] Oppenheimer D, Ganapathi A, Patterson DA. Why do Internet services fail, and what can be done about it? In: Proc. of the 4th Int'l

Conf. on Internet Technologies and Systems. Seattle: Berkeley: USENIX Association, 2003. 1−16.

[5] Chen HF, Jiang GF, Yoshihira K, Saxena A. Invariants based failure diagnosis in distributed computing systems. In: Proc. of the 29th Int'l Symp. on Reliable Distributed Systems. Washington: IEEE Computer Society Press, 2010. 160−166. [doi: 10.1109/SRDS.2010.26]

[6] Ghanbari S, Amza C. Semantic-Driven model composition for accurate anomaly diagnosis. In: Proc. of the 5th Int'l Conf. on Autonomic Computing. Washington: IEEE Computer Society Press, 2008. 35−44. [doi: 10.1109/ICAC.2008.33]

[7] Sahoo RK, Oliner AJ, Rish I, Gupta M, Moreira JE, Ma S, Vilalta R, Sivasubramaniam A. Critical event prediction for proactive management in large-scale computer clusters. In: Proc. of the 9th Int'l Conf. on Knowledge Discovery and Data Mining. New York: ACM Press, 2003. 426−435. [doi: 10.1145/956750.956799]

[8] Fonseca R, Porter G, Katz RH, Shenker S, Stoica I. X-Trace: A pervasive network tracing framework. In: Proc. of the 4th Int'l Conf. on Networked Systems Design & Implementation. Berkeley: USENIX Association, 2007. 20−33.

[9] Chen HF, Jiang GF, Ungureanu C, Yoshihira K. Failure detection and localization in component based systems by online tracking. In: Proc. of the 11th Int'l Conf. on Knowledge Discovery in Data Mining. New York: ACM Press, 2005. 750−755. [doi: 10.1145/1081870.1081968]

[10] Jiang GF, Chen HF, Yoshihira K. Discovering likely invariants of distributed transaction systems for autonomic system management. In: Proc. of the 3rd Int'l Conf. on Autonomic Computing. Washington: IEEE Computer Society Press, 2006. 199−208. [doi: 10.1109/ICAC.2006.1662399]

[11] Jiang M, Munawar MA, Reidemeister T, Ward PAS. System monitoring with metric-correlation models: Problems and solutions. In: Proc. of the 6th Int'l Conf. on Autonomic Computing. New York: ACM Press, 2009. 13−22. [doi: 10.1145/1555228.1555233]

[12] Pitkow JE. Summary of WWW characterizations. Computer Networks and ISDN Systems, 1998,30(1-7):551−558. [doi: 10.1016/S0169-7552(98)00066-X]

[13] Hellerstein JL, Zhang F, Shahabuddin P. Characterizing normal operation of a Web server: Application to workload forecasting and problem detection. In: Proc. of the Computer Measurement Group. New York, 1998. 150−160.

[14] Williams A, Arlitt M, Williamson C, Barker K. Web workload characterization: Ten years later. Web Content Delivery, 2005, 2(1):3−21.

[15] Arlitt M, Krishnamurthy D, Rolia J. Characterizing the scalability of a large Web-based shopping system. Trans. on Internet Technology, 2001,1(1):44−69. [doi: 10.1145/383034.383036]

[16] Gao J, Jiang GF, Chen HF, Han JW. Modeling probabilistic measurement correlations for problem determination in large-scale distributed systems. In: Proc. of the 29th IEEE Int'l Conf. on Distributed Computing Systems. Washington: IEEE Computer Society Press, 2009. 623−630. [doi: 10.1109/ICDCS.2009.56]

[17] Menascé DA, Almeida VAF, Fonseca R, Mendes MA. A methodology for workload characterization of e-commerce sites. In: Proc. of the 1st Int'l Conf. on Electronic Commerce. New York: ACM Press, 1999. 119−128. [doi: 10.1145/336992.337024]

[18] Mahalanobis PC. On the generalized distance in statistics. National Institute of Sciences of India, 1936,2(1):35−49.

[19] Menascé DA. TPC-W: A benchmark for e-commerce. Internet Computing, 2002,6(3):83−87. [doi: 10.1109/MIC.2002.1003136]

[20] Breunig MM, Kriegel HP, Ng RT, Sander J. LOF: Identifying density-based local outliers. In: Proc. of the SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 2000. 93−104. [doi: 10.1145/335191.335388]

[21] Stehle E, Lynch K, Shevertalov M, Rorres C, Mancoridis S. On the use of computational geometry to detect software faults at runtime. In: Proc. of the 7th Int'l Conf. on Autonomic Computing. New York: ACM Press, 2010. 109−118. [doi: 10.1145/1809049.1809069]

[22] Katayama N, Satoh S. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In: Proc. of the SIGMOD Int'l Conf. on Management of Data. New York: ACM Press, 1997. 369−380. [doi: 10.1145/253260.253347]

[23] Kiczales G, Lamping J, Mendhekar A, Maeda C, Lopes C, Loingtier JM, Irwin J. Aspect-Oriented programming. In: Proc. of the 11th European Conf. on Object-Oriented Programming. Berlin: Springer-Verlag, 1997. 220−242.

[24] Kiczales G, Hilsdale E, Hugunin J, Kersten M, Palm J, Griswold W. An overview of AspectJ. In: Proc. of the 15th European Conf. on Object-Oriented Programming. Berlin: Springer-Verlag, 2001. 327−354.

[25] Zhang WB, Wang S, Wang W, Zhong H. Bench4Q: A QoS-oriented e-commerce benchmark. In: Proc. of the 35th Int'l Conf. on Computer Software and Applications. Washington: IEEE Computer Society Press, 2011. 38−47. [doi: 10.1109/COMPSAC.2011.14]

[26] Huang T, Chen NJ, Wei J, Zhang WB, Zhang Y. OnceAS/Q: A QoS-enabled Web application server. Journal of Software, 2004, 15(12):1787−1799 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/15/1787.htm

[27] Pertet S, Narsimhan P. Causes of failures in Web applications. Technical Report, CMU-PDL-05-109, Berkeley: Carnegie Mellon University, 2005.

[28] Kiciman E, Fox A. Detecting application-level failures in component-based Internet services. IEEE Trans. on Neural Networks, 2005,16(5):1027−1041. [doi: 10.1109/TNN.2005.853411]

[29] Barham P, Donnelly A, Isaacs R, Mortier R. Using Magpie for request extraction and workload modeling. In: Proc. of the 6th Int'l Symp. on Operating Systems Design & Implementation. Berkeley: USENIX Association, 2004. 18−31.

[30] Bodic P, Friedman G, Biewald L, Levine H, Candea G, Patel K, Tolle G, Hui J, Fox A, Jordan MI, Patterson D. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In: Proc. of the 2nd Int'l Conf. on Automatic Computing. Washington: IEEE Computer Society Press, 2005. 89−100. [doi: 10.1109/ICAC.2005.18]

[31] Jiang GF, Chen HF, Kenji Y. Modeling and tracking of transaction flow dynamics for fault detection in complex systems. IEEE Trans. on Dependable and Secure Computing, 2006,3(4):312−326. [doi: 10.1109/TDSC.2006.52]

[32] Jiang GF, Chen HF, Yoshihira K. Efficient and scalable algorithms for inferring likely invariants in distributed systems. IEEE Trans. on Knowledge and Data Engineering, 2007,19(11):1508−1523. [doi: 10.1109/TKDE.2007.190648]

[33] Munawar MA, Ward PAS. A comparative study of pairwise regression techniques for problem determination. In: Proc. of the Int'l Conf. of the Center for Advanced Studies on Collaborative Research. New York: ACM Press, 2007. 152−166. [doi: 10.1145/1321211.1321227]

[34] Guo Z, Jiang GF, Chen HF, Yoshihira K. Tracking probabilistic correlation of monitoring data for fault detection in complex systems. In: Proc. of the 36th Int'l Conf. on Dependable Systems and Networks. Washington: IEEE Computer Society Press, 2006. 259−268. [doi: 10.1109/DSN.2006.70]

[35] Faraoun KM, Boukelif A. Neural networks learning improvement using the *k*-means clustering algorithm to detect network intrusions. Int'l Journal of Computational Intelligence, 2007,3(2):28−36.

[36] Abraham A, Grosan C, Martin-Vide C. Evolutionary design of intrusion detection programs. Int'l Journal of Network Security, 2007,4(3):328−339.

[37] Tajbakhsh A, Rahmati M, Mirzaei A. Intrusion detection using fuzzy association rules. Applied Soft Computing, 2009,9(2):462−469. [doi: 10.1016/j.asoc.2008.06.001]

[38] Ertoz L, Eilertson E, Lazarevic A, Tan PN, Kumar V, Srivastava J, Dokas P. The MINDS—Minnesota Intrusion Detection System, Next Generation Data Mining. Boston: MIT Press, 2004.

[39] Nguyen LDK, Tahereh B, Sanjay C, Zainab Z. Network anomaly detection using a commute distance based approach. In: Proc. of the 2nd Int'l Conf. on Data Mining Workshops. Washington: IEEE Computer Society Press, 2010. 943−950. [doi: 10.1109/ICDMW.2010.90]

附中文参考文献:

[26] 黄涛,陈宁江,魏峻,张文博,张勇.OnceAS/Q:一个面向 QoS 的 Web 应用服务器.软件学报,2004,15(12):1787−1799. http://www.jos.org.cn/1000-9825/15/1787.htm

王焘(1982－),男,河南开封人,博士生,主要研究领域为分布式系统故障诊断,自主计算,软件可靠性.

张文博(1976－),男,博士,副研究员,CCF 会员,主要研究领域为网络分布式计算,软件工程.

魏峻(1970－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.

钟华(1971－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.