软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(4):886−903 [doi: 10.13328/j.cnki.jos.004761]

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

基于在线故障定位及自主适应提高软件可靠性*

杨晓燕[1], 周 远[2], 丁佐华[1]

[1](浙江理工大学 信息学院,浙江 杭州 310018)

[2](浙江理工大学 理学院,浙江 杭州 310018)

通讯作者: 丁佐华, E-mail: zouhuading@hotmail.com

摘 要: 可靠性是衡量软件质量的一个重要指标.在线预测和提高软件可靠性是一个重要的研究课题.目前大多数在线预测和提高软件可靠性的方法具有如下弱点:不能预测软件不同时段的可靠性,且不能定位导致可靠性下降的组件.针对服务组合软件系统,提出在线提高可靠性的方法.通过观测端口失效数据,预测在线系统在不同时段的可靠性.当预测到的可靠性低于预期时,则采用改进的基于频谱的错误定位方法,定位出导致问题的故障组件,再通过添加新组件或替换故障组件的方法对软件系统重新配置,从而在线自动提高软件系统可靠性.使用在线商店的事例来说明方法的有效性.

关键词: 可靠性预测;基于频谱定位;重新配置;自适应算法;可靠性提高

中图法分类号: TP311

中文引用格式: 杨晓燕,周远,丁佐华.基于在线故障定位及自主适应提高软件可靠性.软件学报,2015,26(4):886−903. http://www.jos.org.cn/1000-9825/4761.htm

英文引用格式: Yang XY, Zhou Y, Ding ZH. Improving software reliability based on online fault localization and self-adaption. Ruan Jian Xue Bao/Journal of Software, 2015,26(4):886−903 (in Chinese). http://www.jos.org.cn/1000-9825/4761.htm

Improving Software Reliability Based on Online Fault Localization and Self-Adaption

YANG Xiao-Yan[1], ZHOU Yuan[2], DING Zuo-Hua[1]

[1](School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China)

[2](School of Science, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: Reliability is an important index to measure the quality of software. Online predicting and improving software reliability are an important research topic. Most existing methods have the following weakness: They can neither predict software reliability on different time intervals nor locate the faulty components that cause the declining of the reliability. This paper proposes a new method to online improve reliability for service composition. The method uses the monitored failure data at ports to predict the reliabilities of service composition on different time intervals. If the predicted reliability is lower than the expected value, it then locates the faulty component that causes the declining of the reliability by using an improved spectrum-fault-localization method. The system is automatically reconfigured by adding a new component or replacing the faulty component to improve the system reliability. An online shop example is used to demonstrate the effectiveness of the proposed method.

Key words: reliability prediction; spectrum-based localization; reconfiguration; self-adaptation algorithm; reliability improvement

可靠性是衡量一个软件质量的重要指标,尤其是对于在线服务软件.例如应用于电信、金融、医疗、教育、制造、能源、物流等重要行业的软件.没有精确的可靠性预测,任何想不到的错误都可能中断服务.大多数这类系统都被期望满足严格的可靠性需求.当系统可靠性下降时,就需要提高系统可靠性,满足可靠性需求.在线服务型软件有着自己的特点:

* 基金项目: 国家自然科学基金(61210004, 61170015)
收稿时间: 2014-08-12; 修改时间: 2014-10-14; 定稿时间: 2014-11-14

- 首先,其运行环境是不确定和高度开放的,不同时刻有不同数量的用户在请求服务;
- 其次,服务组合软件系统由多个组件构成,由于并行使用,在同一时刻可能会有多个故障发生,而且会因异步服务而使故障的影响发生延迟.

因此,这类软件的可靠性随着运行时间的变化而动态发生变化.而传统的可靠性预测值在一个可控制的环境下得到,无法正确反映系统的实际可靠性,因此,我们需要对系统进行在线可靠性预测.当预测值低于期望可靠性时,我们就希望系统能够进行在线的自适应配置,从而提高可靠性.

目前已经有一些在线预测软件可靠性和提高软件可靠性的研究成果,比如:Pietrantuono 等人[1]通过程序不变量来计算在线软件的可靠性;Malek 等人[2]通过连续分析和自主配置来提高移动软件系统的可靠性;Cooray 等人[3]采用隐式马尔可夫链来预测嵌入式系统和移动系统的可靠性,并通过对过程内的组件重新分配位置以及改变组件的个数来提高软件的可靠性.但这些工作有一些共同的弱点:

1) 不能预测软件在不同时段的可靠性;
2) 不能定位导致可靠性下降的组件.

在本文中,我们使用 ARIMA 模型对运行时软件的端口数据进行失效数据预测,进而进行在线系统不同时段的可靠性预测.若预测的可靠性低于预期值,则通过改进的基于频谱的错误定位技术定位出故障组件;然后,系统再自动添加与故障组件功能相同的组件,或用功能相同的组件替换故障组件来在线自动提高其可靠性.图 1 显示了我们的方法的框架.本文的贡献主要在于成功地定位出导致系统可靠性降低的故障组件,在线自动提高了系统的可靠性.

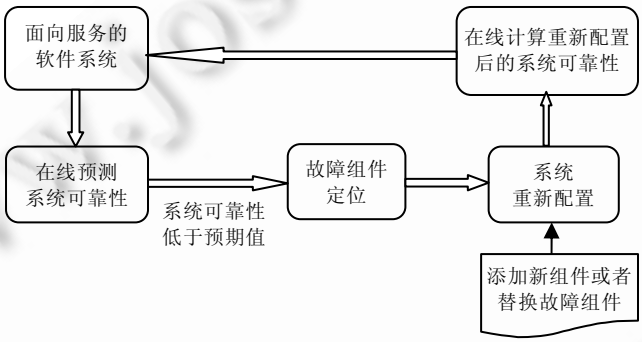

Fig.1 Framework of the proposed method

图 1 方法框架概况

本文第 1 节简要介绍我们以前的工作,即在线预测系统可靠性的方法.第 2 节介绍当系统可靠性降低时,如何定位故障组件的方法.第 3 节介绍如何对系统重新配置以提高系统可靠性,并给出系统自动提高系统可靠性的自适应算法.第 4 节通过实例演示说明我们的方法的可行性.第 5 节是相关工作的介绍.最后,对全文做出总结.

1 在线可靠性预测

在我们以前的工作中[4,5],使用基于端口的可靠性模型,通过收集日志文件中记录的端口的变量值,运用统计模型 ARIMA 预测变量的失效数据,由失效数据预测端口的可靠性与系统的可靠性.

1.1 基于端口的组件模型

在我们以前的工作中,我们运用基于组件的服务组件架构 SCA 模型计算系统的在线可靠性.SCA 提供独立于语言的方法来定义与组合服务组件,支持不同的编程语言实现组件功能.基于 SCA,我们分析每个端口的可靠性并计算服务的可靠性.对于 SCA 模型简要介绍如下.

- 端口

端口 p 是一个多元组(M,t,c),M 表示端口 p 的有限方法集,t 表示提供或需求的端口类型,c 表示同步或异步的通信类型.

- 组件

组件 Com 是一个多元组(P_p,P_r,G,W):P_p 是提供端口的有限集;P_r 是需求端口的有限集;G 是有限子组件集;$W \subseteq TP \times \bigcup_{C \in G}(C.P_p \cup C.P_r)$ 表示非自反关系的端口关系,且 $TP = P_p \cup P_r \cup \bigcup_{C \in G} C.P_r$, $C.P_p$ 和 $C.P_r$ 分别表示子组件 C 提供和需求的端口集.

我们使用端口活动描述组件的动态行为,其基本活动被认为是两个端口之间的信息交换.组件动态行为表达式的语法定义如下:

$$
\begin{array}{lll}
BE ::= & p \bullet_m & (Synchronous\ sending\ massage) \\
 & p_1 \bullet_m\ p_2 & \\
 & p \circ_m & (Synchronous\ receiving\ message) \\
 & p_1 \circ_m\ p_2 & \\
 & p \blacklozenge_m & (Asynchronous\ sending\ message) \\
 & p_1 \blacklozenge_m\ p_2 & \\
 & p \lozenge_m & (Asynchronous\ receiving\ message) \\
 & stop & (stop) \\
 & BBE & (Basic\ behavior\ expression) \\
 & BE; BE & (Sequence) \\
 & BE \lhd b \rhd BE & (Condition) \\
 & b * BE & (Loop) \\
 & BE \sqcap BE & (Non\text{-}determinism) \\
 & BE \parallel BE & (Parallel) \\
 & BE[p_1 / p_2] & (Renaming) \\
 & BE[p_1 \to p_2] & (Wiring)
\end{array}
$$

这里:BBE 代表跳过什么也不做的基本活动;分配活动 $p.x=e$ 分配表达式 e 的值给 p 的变量 x.b 表示布尔表达式,定义省略;m 代表消息;$p \bullet_m$ 代表端口 p 同步发送消息 m,但是 p 和任何参考端口都不连接;$p_1 \bullet_m p_2$ 表示端口 p_1 同步发送消息 m 给端口 p_2;其他基本活动的含义基本相似.$BE[p_1/p_2]$是语法的重命名,表示在$\sqcap BE$ 中用 p_1 替换 p_2.$BE[p_1 \to p_2]$用来指定连接操作,其语法转换定义如下:

$$
BE[p_1 \to p_2] = \begin{cases}
p_1 \bullet_m\ p_2 & BE = p \bullet_m \wedge p = p_1 \\
p_2 \circ_m\ p_1 & BE = p \circ_m \wedge p = p_2 \\
p_1 \blacklozenge_m\ p_2 & BE = p \blacklozenge_m \wedge p = p_1 \\
p_2 \lozenge_m\ p_1 & BE = p \lozenge_m \wedge p = p_2 \\
BE_1[p_1 \to p_2]; BE_2[p_1 \to p_2] & BE = BE_1;\ BE_2 \\
BE_1[p_1 \to p_2] \lhd b \rhd BE_2[p_1 \to p_2] & BE = BE_1 \lhd b \rhd BE_2, \\
b * BE_1[p_1 \to p_2] & BE = b * BE_1 \\
BE_1[p_1 \to p_2] \sqcap BE_2[p_1 \to p_2] & BE = BE_1 \sqcap BE_2 \\
BE_1[p_1 \to p_2] \parallel BE_2[p_1 \to p_2] & BE = BE_1 \parallel BE_2 \\
BE & BE = p_2 \odot p_1 \\
BBE & BE = BBE
\end{cases}
$$

其中, $\odot \in \{\bullet_m, \circ_m, \blacklozenge_m, \lozenge_m\}$.

- 轨迹

一个给定的轨迹 tr 包含 n 个活动 $a_1,a_2,\ldots,a_n$,按照前面的语法规则,用 $BE \xrightarrow{tr} BE'$ 表示由 BE 转到 BE',所

经过的序列即为活动轨迹 tr: $BE \xrightarrow{a_1} BE_1 \to ... \to BE_{n-1} \xrightarrow{a_n} BE'$.

1.2 失效数据预测

服务失败可能由服务组件的缺陷引起,也可能由所需的服务传播或执行环境等原因引起.这里,我们主要考虑与服务性能有关的错误,认为当一个服务请求的响应时间大于它的门限值时发生服务错误.我们使用 ARIMA 来预测失效数据.

ARIMA 模型是时间序列分析中广泛使用的预测模型[6],我们选择该模型进行预测的原因在于:此模型考虑了时间序列数据的变化趋势、季节性、循环、误差和随机干扰性;其次,该模型对短期变化趋势的预测具有较高的准确率.而我们收集的失效数据是一个时间序列集,需要预测的是未来短期内的失效数据,因此,ARIMA 模型是一个理想的预测模型.该模型一般形式表示为 $ARIMA(p,d,q)$,其中,p,q 分别表示自回归项和移动平均项,d 表示时间序列成为平稳时所做的差分次数.对于平稳、零均值的时间序列$\{X_t\}$,其一般表达式为

$$X_t=\phi_1X_{t-1}+\ldots+\phi_pX_{t-p}+\varepsilon_t+\theta_1\varepsilon_{t-1}+\ldots+\theta_q\varepsilon_{t-q},$$

其中,$\phi_1,\ldots,\phi_p$ 为自回归系数,$\theta_1,\ldots,\theta_q$ 为滑动平均系数,$\{\varepsilon_t\}$为白噪声序列(零均值和恒定方差σ^2).

利用 ARIMA 模型进行失效数据的预测包括如下过程:

- 第 1 步,失效数据的平稳化处理.

当失效数据序列$\{Y_1,Y_2,\ldots,Y_t\}$不平稳时,我们通过逐次差分,直到获得新的平稳序列$\{X_1,X_2,\ldots,X_{t-d}\}$,其中,所进行的差分次数即为 d 的值.然后,再将平稳序列$\{X_1,X_2,\ldots,X_{t-d}\}$零均值化处理 $X_i = X_i - \overline{X}$, 其中, $\overline{X}$ 为平稳序列的均值.

- 第 2 步,模型确定.

利用最小二乘估计法、极大似然估计法等方法对 $ARMA(p,q)$的自回归系数、滑动平均系数进行估计;再利用 AIC 准则进行模型定阶,有着最小 AIC 值的模型即为最优模型.最后判断此模型的残差序列是否为白噪声:若是,则通过检验,得出软件可靠性预测模型;否则,重新计算.

- 第 3 步,失效数据预测.

根据所得模型预测 $\{X'_t\}$,然后还原为失效数据$\{Y_t\}$的预测结果.预测过程的算法如下所示.

Algorithm 1. Pseudo-Code for the prediction of the algorithm.

```
Input: Original time series;
Output: Predict values.
Begin
Yt=read original time series
  Internal ArrayList Adjust(ArrayList Yt)
  {if (!judge(Yt))
    {differential(Yt);}
    else
      return Yt is a stationary series;}
    ModelIdentify(ACF,PACF)
      /*ACF is autocorrelation function,
      PACF is the partial autocorrelation function*/
    While (ACF trail && PACF trail)
      return ARMA(p,q);
ParameterEstimation(ARIMA(p,d,q))
    {While (differential(Yt))
  do (ADF check)
  /*ADF represents Augmented Dickey-Fuller Unit Root Test*/
```

```
{if (Y_t is a stationary)
   d=differentialtimes;
     return s;}
judge(p,q){
   while (AIC=min && p && q)}
     return ARIMA(p,d,q);}
Prediction(Y_t)
{Ipm(predict early parameters values);
   Epm(predict future parameters values);
Switch(choice)
Case 1: Ipm();
   return result;
   break;
Case 2: Emp();
   return result;
   break;      } }
ErrorAnalysis()
   {if ((|result-actualvalue|/actualvalue)<10%)
     return result;
       else ParameterEstimation(ARIMA(p,d,q))
End
```

算法中:

- 函数 *Adjust* 首先确定序列的稳定性;
- 函数 *ModelIdentify* 确定 ARIMA 模型;
- 模型 *ARIMA*(*p*,*d*,*q*)的参数通过单元根检查和函数 *ParameterEstimation* 的 AIC 的值得到;
- 函数 *Prediction* 进行预测,由两个函数组成:*Ipm* 预测早期参数值,*Epm* 预测未来参数值;
- 函数 *ErrorAnalysis* 分析预测误差.

对于算法的复杂度,函数 *Adjust* 和 *ModelIdentify* 的复杂度为 $O(n\log n)$和 $O(n)$,函数 *ParameterEstimation* 的复杂度为 $O(n^2)$,函数 *Prediction* 的时间复杂度为 $O(1)$,函数 *ErrorAnalysis* 的时间复杂度为 $O(n^2)$.所以,算法的时间复杂度为 $T(n)=O(n^2)$.

1.3 预测端口可靠性

当一个端口的方法被调用时,我们认为这个端口被访问 1 次,n_i 表示端口 p 在一段时间间隔[0,T]内被访问的次数.预测过程中一个失效数据表示一次错误,f_i 表示预测过程中错误的次数.假设端口 p 有操作 $p_1\to p_2$,则在时间 T 端口 p 的可靠性定义为

$$r(p[p_1\to p_2])=1-\frac{f_i}{n_i}.$$

因为端口 p 有一个固定操作 $p_1\to p_2$,为了方便起见,我们也记为 $r(p)\triangleq r(p[p_1\to p_2])$.

1.4 预测系统可靠性

在我们以前的工作中,已经定义了基于端口的系统可靠性.系统在执行一段时间后的可靠性定义为

$$r(system)=\frac{\sum_i r(tr_i)f(tr_i)}{\sum_i f(tr_i)},$$

其中,$r(tr_i)$是轨迹 tr_i 的可靠性,$f(tr_i)$是轨迹 tr_i 的发生频率.

行为表达式 BE 的轨迹可靠性为 BE 的端口可靠性的乘积.如果 BE 包含顺序、选择或循环结构,我们使用以下规则计算包含 BE 的所有轨迹的可靠性:

- [顺序规则]. p_1,p_2 为两个端口,它们形成一个顺序的表达式:$p_1;p_2$,我们有:$r(p_1;p_2)=r(p_1)r(p_2)$;
- [选择规则]. p_1 和 p_2 为两个端口,它们形成的选择表达式为 $p_1 \lhd b \rhd p_2$.若 b=True,则执行 p_1 且 $\chi(b)=1$;否则执行 p_2,$\chi(b)=0$.于是有:$r((p_1 \lhd b \rhd p_2))=r(p_1)^{\chi(b)} \times r(p_2)^{1-\chi(b)}$.

对于循环端口,我们有两种情况:(1) 循环的状况取决于用户输入或来自外部环境的数据;(2) 循环时间是一个固定的数字.

- [循环规则(1)]. p 是一个端口,b 是一个条件,它们形成一个循环表达式:$b*p$,假设退出第 i 次循环的概率为 $p(i)$,且 $\sum_i p(i)=1$.然后我们有: $r((b*p))=r(p)^{\sum_i ip(i)}$;
- [循环规则(2)]. p 是一个端口,n 为一个正整数,它们形成一个循环表达式:$n*p$.那么我们有:$r(n*p)=(r(p))^n$.

2 基于频谱的故障组件定位

上节我们讲述了在线服务系统在运行时的可靠性预测.当预测到的可靠性低于预期值时,可能由服务组件的缺陷引起,也可能由所需的服务传播或执行环境等原因引起.本文主要考虑因某个或多个组件发生故障导致预测到的系统可靠性达不到预期值,因此我们需要定位出发生故障的组件.本文在 Abreu 等人提出的错误定位方法的基础上[7−9],考虑到经过组件的次数,采用改进的基于频谱的错误定位方法进行错误组件的定位.

2.1 建立频谱矩阵

一个在线服务系统由一系列组件构成,可能会有多个错误组件.假设服务组合由 M 个组件组成,记为 C_j ($j \in \{1,\ldots,M\}$),可能有 E 个错误组件.诊断报告 $D=\langle \ldots,d_k,\ldots \rangle$为有序的可能的多个错误的组件候选集,$d_k$ 按照错误的可能性排列.

在 Abreu 等人提出的错误定位方法中,频谱矩阵表示系统动态行为中所包含的组件的标记.系统每执行一次,经过了的组件记为 1,否则记为 0.假设共执行了 N 次,频谱矩阵表示为一个 $N \times M$ 的矩阵 A.执行的结果存储在向量 e 中,表示每次执行完毕后,若运行通过则记为 0,运行失败记为 1.本文改进了这一方法,考虑了所经组件的次数,即,矩阵元素 a_{ij} 表示组件 C_j 在第 i 次执行时是否经过此组件及经过的次数.

2.2 候选集生成

由上一步建立的频谱矩阵(A,e)可知:假设每次初始输入变量是正确的,最后组件的输出值由 e 给出,我们可以推断出所经过的组件的正确性.本文使用最小命中集算法来计算诊断候选集,因为当系统的组件数目庞大,需要搜索大量命中集时,这种解决方案可以很有效地获得所需最小命中集.按照相关性排序,排除可能没有价值的后验概率非常低的候选集,因此,该解决方案既能得到需处理的候选集最小数量,又提高了效率.基于频谱的错误定位技术(SFL)能够很好地按照组件故障可能性预测故障排行.SFL 输入频谱矩阵(A,e)产生有序的组件错误可能性排行.组件以相似系数计算排名,提供了良好的准确性,即排名最高的组件往往是错误的.Abreu 等人在文献[9]中给出了相似系数的定义,但他们只考虑到是否调用组件,而我们的频谱矩阵是经过组件的次数,故需修改 n_{11},n_{10},n_{01} 的计算公式,具体相似系数的定义为

$$s(j)=\begin{cases}\dfrac{n_{11}(\{j\})}{den(j)=\sqrt{(n_{11}(\{j\})+n_{10}(\{j\}))*(n_{11}(\{j\})+n_{01}(\{j\}))}}, & den(j)\neq 0\\ 0, & \text{otherwise}\end{cases} \tag{1}$$

其中,

$$n_{11}(j)=|\{i\in\{1,2,...,N\}\mid a_{ij}>0\wedge e_i=1\}|,$$
$$n_{10}(j)=|\{i\in\{1,2,...,N\}\mid a_{ij}>0\wedge e_i=0\}|,$$

$$n_{01}(j)=|\{i\in\{1,2,...,N\}\mid a_{ij}=0\wedge e_i=1\}|.$$

最小命中集算法 STACCATO 如下:

Algorithm 2. Staccato.

Inputs: *Matrix*(*A*,*e*), number of components *M*, stop criteria λ, *L*;

Output: Minimal Hitting set *D*.

1: $T_F\leftarrow\{A_i^*\mid e_i=1\}$ ▷Collection of conflict sets
2: $R\leftarrow rank(H,A,e)$ ▷Rank according to heuristic *H*
3: $D\leftarrow\varnothing$
4: $seen\leftarrow 0$
5: for all $j\in\{1,...,M\}$ do
6: if $n_{11}(\{j\})=|TF|$ then
7: $push(D,\{j\})$
8: $A\leftarrow Strip\_Component(A,j)$
9: $R\leftarrow R\backslash\{j\}$
10: $seen\leftarrow seen+1\backslash M$
11: end if
12: end for
13: while $R\neq\varnothing\wedge seen\leqslant\lambda\wedge|D|\leqslant L$ do
14: $j\leftarrow POP(R)$
15: $seen\leftarrow seen+1\backslash M$
16: $(A',e')\leftarrow Strip(A,e,j)$
17: $D'\leftarrow Staccato(A',e',M-|\{j|n_{11}(j)=|T_F|\}|-1,\lambda,L)$
18: while $D'\neq\varnothing$ do
19: $j'\leftarrow POP(D')$
20: $j'\leftarrow\{j\}\cup j'$
21: if *is_not_subsumed* (D,j') then
22: $push(D,j)$
23: end if
24: end while
25: end while
26: return *D*

算法 STACCATO 递归地使用频谱矩阵(*A*,*e*)的子结构生成最小命中集.算法分为 3 步.

(1) 初始化阶段使用相关系数对组件进行排行;

(2) 将所有失败的集合中包含的组件加入候选集 *D* 中;

(3) 当$|D|<L$时,对排行中首先超过λ的组件作以下处理:移除组件 *j*,保留(*A*,*e*)中所有 $e_i=1\wedge a_{ij}>0$ 的 A_i^*,对新的(*A*,*e*)运行 STACCATO 算法,将返回的组件加入 *D* 并验证它是否为最小命中集.

算法 STACCATO 的时间复杂度为 $O((|D|+M\times(N+\log M))C)$,而实际中,由于搜索算法的时间复杂度只有 $O(C\times|D|+C\times M\times(N+\log M))$,对于空间复杂度,因为算法需要存储每个组件的 4 个计数器来得出排行(n_{11},n_{10},n_{01},n_{00}),所以递归深度为 *C*,算法 STACCATO 的空间复杂度为 $O(C\times M)$.

2.3 候选集排序

对于上一阶段所得出的候选集 d_k 与实际观测值一致,然而尽管候选集空间减小了,但剩余的候选集 d_k 的概

率也并非相同.因此,计算候选集概率 $P_r(d_k)$、建立候选集排行依旧至关重要.我们使用贝叶斯规则来计算候选集的概率.

根据候选集 d_k 的所有观测值,每个候选集 d_k 的概率描述了实际系统的错误情况.由贝叶斯规则得出,在所观测到的观测值下候选集 d_k 的后验概率的计算公式为

$$P_r(d_k \mid obs) = \frac{P_r(obs \mid d_k)}{P_r(obs)} \cdot P_r(d_k) \tag{2}$$

令 $P_r(j)=p$ 表示组件 C_j 错误的先验概率,假设组件错误是独立的,则候选集 d_k 的先验概率为

$$P_r(d_k) = p^{|d_k|} \cdot (1-p)^{M-|d_k|} \tag{3}$$

$Pr(obs)$是正规化因子,无需计算.因每次执行是独立的,故:

$$P_r(obs \mid d_k) = \prod_{i=1}^{N} P_r(obs_i \mid d_k) \tag{4}$$

其中,$P_r(obs_i|d_k)$如下定义:

$$P_r(obs_i \mid d_k) = \begin{cases} 1, & d_k \to obs_i \\ 0, & obs_i \wedge d_k \mid = \perp \\ \varepsilon_{ik}, & d_k \to \{obs_1, ..., obs_N\} \end{cases} \tag{5}$$

关于ε_{ik}的定义有很多种,本文的定义为

$$\varepsilon_{ik} = \begin{cases} \prod\limits_{j:(j \in d_k) \wedge (a_{ij}>0)} h_j^{a_{ij}}, & e_i = 0 \\ 1 - \prod\limits_{j:(j \in d_k) \wedge (a_{ij}>0)} h_j^{a_{ij}}, & e_i = 1 \end{cases} \tag{6}$$

其中,$h_j \in [0,1]$表示组件 j 正常工作的概率.

由公式(4)、公式(6),我们可以得出 $P_r(obs|d_k)$关于 h_j 的表达式;通过最大似然估计法,我们可以得到 h_j 的估计值,实际上就是求解如下极值问题:

$$\arg\max_{G} P_r(obs \mid d_k) \tag{7}$$

其中,$G=\{h_j \in [0,1]; j=1,2,\ldots,M\}$.将公式(7)所求的最大值代入公式(2),就可以得到候选集 d_k 的后验概率.对每个候选集作如上操作,就可以得到每个候选集的后验概率.对候选集从高到低进行排序,最前面的候选集是最有可能出错的组件集合,我们需要对它进行自适应配置.

3 系统重新配置

上节提出了定位故障组件的方法.定位出故障组件后,为了提高系统可靠性,我们可以采用两种策略来实现这个目标:

- 第一,添加新的功能相同的组件,与故障组件共同实现组件功能,从而提高系统的可靠性需求;
- 第二,更换出故障组件,连接一个功能相同的新组件,解决故障组件的影响,提高系统可靠性.

这里,所谓的功能相同的组件是指它们具有相同的端口,包括相同的函数集合、相同的端口类型以及相同的通信类型,端口间的关系也一致.

3.1 添加组件

确定了故障组件后,通过在线新添加一个相同功能的组件,从而可以实现系统的正常运行.添加组件时,除了需要保证新加入的组件正常运行外,还需要减小新组件对其他部分的影响.在正在运行的系统中添加可以正常运行的组件,此时,系统并非处于初始状态,因此,添加的组件第 1 步是检查系统的当前状态,并对新组件执行必要的初始化操作,使之与系统同步.基于端口的组件架构中,端口封装了所有所需的复杂功能,保持着内部信息,并使用多个线程来控制.端口包含 3 个属性:伙伴链接、端口类型和两个服务之间的交互操作.因此,第 2 步,我们将最后一个服务点分配给伙伴链接以获得最后的服务组合的绑定信息,以此来满足端口到端口的动态绑

定[10,11],从而支持系统的重新配置.

添加新组件后,采用第 1 节所介绍的可靠性计算方法重新计算新配置好的系统可靠性.因新添加组件后,系统可选择运行的组件增多,因此,系统执行的轨迹也随之增加,而且轨迹执行的频率也发生了改变,但因新添加的组件而增加的轨迹与经过故障组件的轨迹之和应等于初始轨迹执行的次数.在系统添加新组件后计算其可靠性过程中,新组件的各端口的可靠性很高,从而提高了系统的可靠性.

3.2 替换组件

当系统可靠性低于预期值时,将故障组件替换为提供相同功能的正常组件,从而保证系统的可靠性需求.替换新组件时要保护组件各个端口的状态信息,而且要防止通信信息等数据的丢失.我们需要执行一系列的操作:

- 第一,通过停止所有由故障组件发出的通信来停止故障组件的执行;
- 第二,解除故障组件和其他组件间的绑定;
- 第三,通过发送请求,将系统其余部分和新组件链接;
- 第四,通过调整新组件与其他组件之间的服务通信来激活新组件[12,13].

替换时先解除端口与连接器间的绑定,端口提供两种方法:一个用来发送状态信息,另一个用于替换组件时执行初始化操作.连接器可以通过将请求转发至替换组件处来解决组件失效问题.连接器将发送的先前建立的到原始组件的请求,再重新发送到新的组件.另外,因运行时改变架构,与需替换新组件的组件进行交互的组件暂时变得不可用,此时,连接器便调停其与其他组件间的交互,对服务请求排队,直到组件可用.因而,系统中其他组件不会受到改变架构时的影响,保证系统的正常运行.

替换故障组件后,再采用第 1 节所介绍的可靠性计算方法重新计算新配置好的系统可靠性.新组件的各端口可靠性很高,系统执行轨迹数目及其执行频率均未改变,进而提高了系统的可靠性.

3.3 自适应算法

我们开发出一种自适应算法来解决在线服务系统的可靠性提高问题.在系统运行过程中,通过组件端口的失效数据在线预测系统的可靠性,当预测到的系统可靠性降低时,我们认为系统某个或多个组件发生故障,导致系统可靠性低于预期.此时,通过改进的基于频谱的错误定位技术找出发生故障的组件,再自动地对系统添加新的与故障组件功能相同的组件,或用与故障组件功能相同的新组件替换出故障组件.最后重新计算系统的可靠性,从而达到提高系统可靠性的目的.算法描述如下:

Step 1:在线计算系统的可靠性 R.

Step 2:if $R<R_E$ (R_E 为系统可靠性预期值),go to Step 3;

if $R\geqslant R_E$,end.

Step 3:计算候选集概率 $P_r(d_k|obs_i)$并排序:

Step3-1:建立组件频谱矩阵(A,e);

Step3-2:确定可能错误组件候选集 D;

Step3-3:确定各候选集的后验概率(即可疑度)并排序.

Step 4:选择当前候选集 D 中可疑度最高的候选集 d_k,$D=D-\{d_k\}$.

Step 5:分别计算采用两种方案的系统可靠性:

1) 用与可疑度最高的候选集功能相同的组件集合进行替换,可靠性计算为 R_1;
2) 添加与可疑度最高的候选集功能相同的组件集合,可靠性计算为 R_2.

Step 6:重新配置系统:

- 若 $R_1\geqslant R_2$,用功能相同的组件替换出故障的组件;
- 若 $R_1<R_2$,继续添加新的功能相同的组件.

Step 7:如果 $\max(R_1,R_2)<R_E$,goto Step 4;否则结束.

在上述算法中,我们假定新组件的可靠性为 1.由以上算法分析其算法复杂度可知:第 1 节中我们已经分析

了在线预测系统可靠性算法的时间复杂度为 $O(n^2)$,当系统可靠性下降时,定位出故障组件的算法时间复杂度为 $O(C\times|D|+C\times M\times(N+\log M))+O(N\times M+M\times(N+\log M))$.自适应地对系统更换组件或添加新组件的时间复杂度为 $O(M)$.因此,该自适应算法的时间复杂度为 $O(n^2+M^2)$.

然而,这一方法还有一些不足之处.当组成系统的组件数目过于庞大时,基于频谱的错误定位算法复杂度会随组件数目的增加而增高,因而自适应算法的复杂度也相对增大.此时,该方法的有效性会大为降低,系统提高可靠性的效率相对较低.

4 实例分析

我们使用在线商店作为事例来阐明本文所提出的方法.在线商店包含组件:C_{11}=*INIT*,C_{12}=*ORDER*,C_{21}=*INVOICE*,C_{22}=*PAYMENT*,C_3=*DELIVERY*,C_{cs},C_{ds}.系统由 Java 语言和 Tomcat 服务器与 SQL 数据库服务器支持实现.在线商店的架构图如图 2 所示.

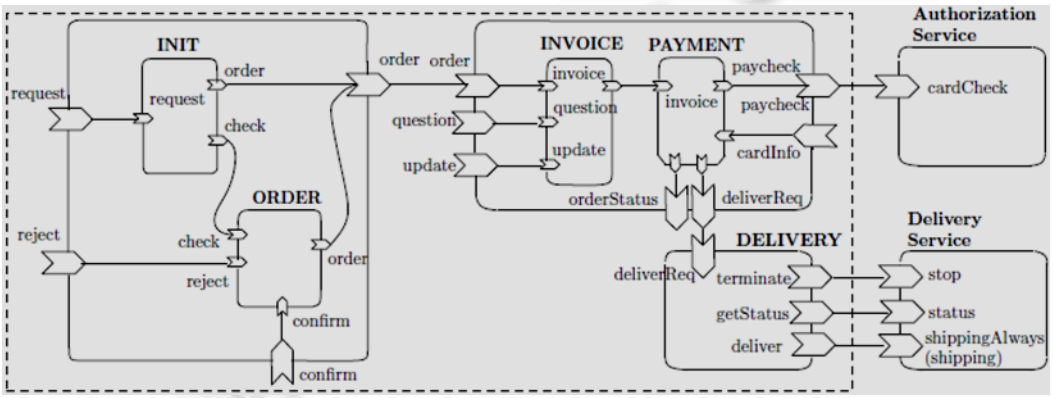

Fig.2 SCA of online shop

图 2 在线商店的服务组件体系结构

在此案例中,系统工作量由 0~200 个当前用户在 20 分钟内进行交易,每 6s 有一个用户进入系统.此外,我们定义响应时间大于 8s 为失效数据.

4.1 预测在线系统的可靠性

首先,我们以 order 端口为例,预测它在第 23 分钟时的可靠性.

通过日志文件,我们可以收集到系统所有端口的参数值.以 order 端口为例,分别预测第 21 分钟、第 22 分钟、第 23 分钟的相应时间.表 1 列举了在 20 分钟内每隔 1 分钟所收集到的 order 端口所有响应时间的值.

Table 1 Collected data for time

表 1 每分钟收集的数据

时间(min)	响应时间(ms)	时间(min)	响应时间(ms)
1	3 560	11	1 980
2	3 860	12	2 250
3	2 214	13	2 750
4	1 928	14	3 126
5	1 356	15	3 584
6	780	16	3 847
7	972	17	4 256
8	1 280	18	4 895
9	1 492	19	5 216
10	1 685	20	5 872

根据第 1.2 节所述,我们得到的预测模型为 *ARIMA*(7,2,6),用此模型来预测接下来 3 分钟的响应时间,预测结

果分别为 7 072.03,7 816.51,8 518.68.显然,端口 order 只有在第 23 分钟时响应时间大于 8s;其次,假设 order 端口共经过了 468 次.因此,我们可以得到端口 order 的可靠性为 1−1/468=0.9978.

同理,我们可以预测其余各端口在第 23 分钟时的可靠性,见表 2.

Table 2 Reliability of each port at the 23rd min

表 2 每个端口在第 23 分钟时的可靠性预测值

Port	*Reliability* (*r*)	Port	*Reliability* (*r*)
P: *request*—request	0.856 8 (r_1)	*P*: *cardInfo*—*cardInfo*	0.990 5 (r_9)
P: *check*—*check*	0.894 3 (r_2)	*P*: *payCheck*—*payCheck*	0.992 1 (r_{10})
P: *reject*—*reject*	0.995 4 (r_3)	*P*: *deliverReq*—*deliverReq*	0.994 6 (r_{11})
P: *confirm*—*confirm*	0.992 1 (r_4)	*P*: *orderStatus*—*orderStatus*	0.991 9 (r_{12})
P: *order*—*order*	0.997 8 (r_5)	*P*: *deliver*—*shippingAlways*	0.997 2 (r_{13})
P: *invoice*—*invoice*	0.996 6 (r_6)	*P*: *deliver*—*shipping*	0.995 5 (r_{14})
P: *question*—*question*	0.993 5 (r_7)	*P*: *getStatus*—*status*	0.991 8 (r_{15})
P: *update*—*update*	0.995 8 (r_8)	*P*: *terminate*—*stop*	0.996 6 (r_{16})

根据端口的可靠性预测值,我们可以计算整个系统的可靠性.根据日志文件可以得到系统共有 13 条运行轨迹,表 3 给出其中的 5 条轨迹.表 4 为对应轨迹的可靠性计算公式.

Table 3 Part of online shop system traces

表 3 在线购物系统的部分轨迹路线

轨迹编号	轨迹
tr_1	*request*→*check*→*reject*
tr_2	*request*→*check*→*confirm*→*order*→*question*→*invoice*→*carInfo*→*paycheck*→*deliverReq*→*deliver*(*shipping*)→*getStatus*→*terminate*
tr_3	*request*→*check*→*confirm*→*order*→*question*→*invoice*→*carInfo*→*paycheck*→*deliverReq*→*deliver*(*shippingAlways*)→*getStatus*→*terminate*
tr_4	*request*→*check*→*confirm*→*order*→*update*→*invoice*→*carInfo*→*paycheck*→*deliverReq*→*deliver*(*shippingAlways*)→*getStatus*→*terminate*
tr_5	*request*→*check*→*confirm*→*order*→*update*→*invoice*→*carInfo*→*paycheck*→*deliverReq*→*deliver*(*shipping*)→*getStatus*→*terminate*

Table 4 Traces reliability of online shop

表 4 在线商店的轨迹可靠性

$r(tr_i)$	Reliability
$r(tr_1)$	$r_1r_2r_3$
$r(tr_2)$	$r_1r_2r_4r_5r_6r_8r_9r_{10}^{\sum_{i=1}^{3} ip_{b_4}(i)}r_{11}r_{13}r_{15}r_{16}$
$r(tr_3)$	$r_1r_2r_4r_5r_6r_8r_9r_{10}^{\sum_{i=1}^{3} ip_{b_4}(i)}r_{11}r_{14}r_{15}r_{16}$
$r(tr_4)$	$r_1r_2r_4r_5r_7r_8r_9r_{10}^{\sum_{i=1}^{3} ip_{b_4}(i)}r_{11}r_{13}r_{15}r_{16}$
$r(tr_5)$	$r_1r_2r_4r_5r_7r_8r_9r_{10}^{\sum_{i=1}^{3} ip_{b_4}(i)}r_{11}r_{14}r_{15}r_{16}$

同时,我们得到:

- 概率:

$$p_{b_4}(1)=0.965, p_{b_4}(2)=0.024\text{ (被盗的信用卡,信用不足等)}, p_{b_4}(3)=0.011\text{ (错别字等)};$$

- 标记频率:

$$f(tr_1)=56, f(tr_2)=53, f(tr_3)=78, f(tr_4)=58, f(tr_5)=85, f(tr_6)=62, f(tr_7)=49,$$
$$f(tr_8)=65, f(tr_9)=57, f(tr_{10})=94, f(tr_{11})=56, f(tr_{12})=76, f(tr_{13})=51.$$

因此,我们预测的整个系统的可靠性为

$$r(system)=\frac{\sum_i r(tr_i)f(tr_i)}{\sum_i f(tr_i)}=0.6989.$$

可以看出,系统的可靠性低于预期值.为此,我们需要找出导致这一问题的故障组件.

4.2 定位故障组件

我们从数据库的文件信息中得到系统运行时所经过的组件信息以及运行情况,以 60s 为窗口大小,更新频谱矩阵,见表 5.

Table 5 Spectral matrix of online shop

表 5 在线商店的频谱矩阵

C_{11}	C_{12}	C_{21}	C_{22}	C_{cs}	C_3	C_{ds}	e	
1	0	1	4	3	2	1	0	obs_1
1	1	1	4	3	2	1	1	obs_2
1	1	1	4	3	2	1	0	obs_3
1	1	1	4	0	0	0	0	obs_4
1	0	1	4	0	0	0	1	obs_5
1	0	1	4	3	2	1	0	obs_6
1	0	1	4	0	0	0	0	obs_7
1	1	1	4	0	0	0	0	obs_8
1	0	1	4	3	2	1	0	obs_9
1	1	0	0	0	0	0	1	obs_{10}

由第 2 节介绍的方法得出此矩阵推出可能发生错误的候选集.相关系数计算见表 6.

Table 6 Correlation coefficient value of each component

表 6 各组件的相关系数

	C_{11}	C_{12}	C_{21}	C_{22}	C_{cs}	C_3	C_{ds}
$n_{11}(j)$	3	2	2	5	1	1	1
$n_{10}(j)$	7	3	7	4	4	4	4
$n_{01}(j)$	0	1	1	1	2	2	2
$S(j)$	0.55	0.52	0.38	0.38	0.26	0.26	0.26

因此,由相关系数大小排序可得$\langle C_{11},C_{12},C_{21},C_{22},C_{cs},C_3,C_{ds}\rangle$.由最小命中集算法 STACCATO 返回的候选集为

$$\{\{C_{11}\},\{C_{12},C_{21}\},\{C_{12},C_{22}\}\}.$$

计算每个候选集的概率值,从而确定出问题组件.对于候选集$\{C_{11}\}$可知:

C_{11}	1	1	1	1	1	1	1	1	1	1
e	0	1	0	0	1	0	0	0	0	1
$P_r(e_i\|d_k)$	h_1	$1-h_1$	h_1	h_1	$1-h_1$	h_1	h_1	h_1	h_1	$1-h_1$

故 $P_r(obs|\{C_{11}\})=h_1^7(1-h_1)^3$,对 h_1 求其最大似然估计值,得到 h_1=0.7,此时,$Pr(e|\{C_{11}\})=2.224\times10^{-3}$.

同理,对于候选集$\{C_{12},C_{21}\}$可知:

C_{12}	C_{21}	e	$P_r(e_i\|d_k)$
0	1	0	h_2
1	1	1	$1-h_1h_2$
1	1	0	h_1h_2
1	1	0	h_1h_2
0	1	1	$1-h_2$
0	1	0	h_2
0	1	0	h_2
1	1	0	h_1h_2
0	1	0	h_2
1	0	1	$1-h_1$

$P_r(obs|\{C_{12},C_{21}\})=h_1^3h_2^7(1-h_1)(1-h_2)(1-h_1h_2)$,对 h_1,h_2 求最大似然估计值为 h_1=0.6417,h_2=0.8528,故:

$$P_r(obs|\{C_{12},C_{21}\})=2.07\times10^{-3}.$$

对于候选集$\{C_{12},C_{22}\}$:

C_{12}	C_{22}	e	$Pr(e_i\|d_k)$
0	4	0	h_2^4
1	4	1	$1-h_1h_2^4$
1	4	0	$h_1h_2^4$
1	4	0	$h_1h_2^4$
0	4	1	$1-h_2^4$
0	4	0	h_2^4
0	4	0	h_2^4
1	4	0	h_1h^4
0	4	0	h^4
1	0	1	$1-h_1$

$P_r(obs|\{C_{12},C_{22}\})=h_1^3h_2^{28}(1-h_1)(1-h_2^4)(1-h_1h_2^4)$, 对 h_1,h_2 求最大似然估计值为 h_1=0.6417,h_2=0.9610,故:

$$P_r(obs|\{C_{12},C_{22}\})=2.07\times10^{-3}.$$

利用公式(7)后,可知 $P_r(d_1|obs)=1.86\times10^{-9}$,$P_r(d_2|obs)=1.92\times10^{-10}$,$P_r(d_3|obs)=1.92\times10^{-10}$.所以,候选集排行为 $\{\{C_{11}\},\{C_{12},C_{21}\},\{C_{12},C_{22}\}\}$,即,组件 C_{11}=*INIT*,出错的可能性最大.

定位出出错组件后,系统将进行重新配置.

4.3 计算添加新组件后的系统可靠性

采用系统重新配置的第 1 个方案,添加与组件 C_{11}=*INIT* 功能相同的新组件 C'_{11}=*INIT'*,重新计算系统可靠性.图 3 所示为添加新组件 C'_{11}=*INIT'* 后系统部分的架构图.

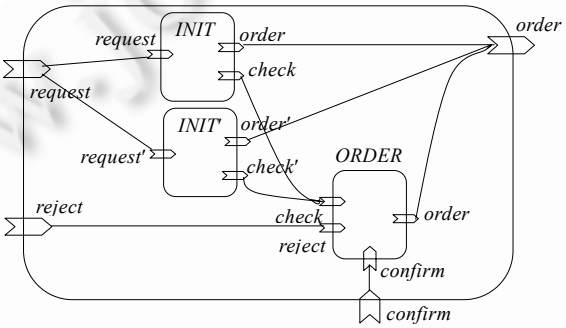

Fig.3 Adding a new component of part online shop SCA

图 3 添加新组件的部分在线商店 SCA

添加新组件 C'_{11}=*INIT* 后,系统可选择组件 C_{11}=*INIT* 与 C'_{11}=*INIT* 两条路径开始执行,且两条开始路径互斥.因此,系统的轨迹变为原来的 2 倍.各端口可靠性见表 7.

Table 7 Values of reliability for each port

表 7 每个端口的可靠性的值

Port	*Reliability* (r)	Port	*Reliability* (r)
P: *request*—*request*	0.856 8 (r_1)	*P*: *update*—*update*	0.995 8 (r_8)
P: *check*—*check*	0.894 3 (r_2)	*P*: *cardInfo*—*cardInfo*	0.990 5 (r_9)
P: *request*—*request'*	1.000 0 (r'_1)	*P*: *payCheck*—*payCheck*	0.992 1 (r_{10})
P: *check'*—*check*	1.000 0 (r'_2)	*P*: *deliverReq*—*deliverReq*	0.994 6 (r_{11})
P: *reject*—*reject*	0.995 4 (r_3)	*P*: *orderStatus*—*orderStatus*	0.991 9 (r_{12})
P: *confirm*—*confirm*	0.992 1 (r_4)	*P*: *deliver*—*shippingAlways*	0.997 2 (r_{13})
P: *order*—*order*	0.897 8 (r_5)	*P*: *deliver*—*shipping*	0.995 5 (r_{14})
P: *order'*—*order*	1.000 0 (r'_5)	*P*: *getStatus*—*status*	0.991 8 (r_{15})
P: *invoice*—*invoice*	0.996 6 (r_6)	*P*: *terminate*—*stop*	0.996 6 (r_{16})
P: *question*—*question*	0.993 5 (r_7)		

添加新组件后,整个系统的可靠性为

$$r(system)=\frac{\sum_i r(tr_i)f(tr_i)}{\sum_i f(tr_i)}=0.8693.$$

4.4 计算替换出故障组件后的系统可靠性

采用系统重新配置的第 2 个方案,用功能相同的新组件 C''_{11}=*INIT*″替换故障组件 C_{11}=*INIT*,重新计算系统可靠性.各端口可靠性见表 8.

Table 8 Values of reliability for each port

表 8 每个端口的可靠性的值

Port	*Reliability* (*r*)	Port	*Reliability* (*r*)
P: *request*—*request*	1.000 0 (r_1)	*P*: *cardInfo*—*cardInfo*	0.990 5 (r_9)
P: *check*—*check*	1.000 0 (r_2)	*P*: *payCheck*—*paycheck*	0.992 1 (r_{10})
P: *reject*—*reject*	0.995 4 (r_3)	*P*: *deliverReq*—*deliverReq*	0.994 6 (r_{11})
P: *confirm*—*confirm*	0.992 1 (r_4)	*P*: *orderStatus*—*orderStatus*	0.991 9 (r_{12})
P: *order*—*order*	1.000 0 (r_5)	*P*: *deliver*—*shippingAlways*	0.997 2 (r_{13})
P: *invoice*—*invoice*	0.996 6 (r_6)	*P*: *deliver*—*shipping*	0.995 5 (r_{14})
P: *question*—*question*	0.993 5 (r_7)	*P*: *getStatus*—*status*	0.991 8 (r_{15})
P: *update*—*update*	0.995 8 (r_8)	*P*: *terminate*—*stop*	0.996 6 (r_{16})

替换出故障组件后整个系统的可靠性为

$$r(system)=\frac{\sum_i r(tr_i)f(tr_i)}{\sum_i f(tr_i)}=0.9546.$$

比较添加新组件与替换出故障组件两种方案的实验结果可得:在该实验中,当系统可靠性达不到预期值时,更换组件提高的系统可靠性效果要更好.因此,本实例中采取更换故障组件 C_{11}=*INIT* 的策略来自动地提高系统的可靠性.

5 相关工作

本文的工作与下面的研究领域有关.

- 系统可靠性预测

在过去的 30 多年间有很多模型和方法被提出[14−27],然而其中大多数方法都是从系统的软件架构考虑出发[16−22,27],这些方法中的基础假设使得它们不适用于动态服务系统领域.此外,目前许多方法是收集静态数据来预测系统可靠性,并未考虑系统的动态性问题.例如:Hu 等人[28]在 Nelson 模型基础上考虑了从用户角度获得关于故障的严重性的信息,提出了基于多粒度故障严重性的软件可靠性模型;Gunawan[29]提出了利用蒙特卡洛方法来预测分布式系统的可靠性.目前也有一些关于在线可靠性预测的研究,例如 Epifani,Ghezzi 和 Mirandola 提出的通过运行时参数自适应模型演化的 KAMI(keep alive models with implementations)方法[30],它利用运行时数据来更新可靠性模型的参数,提供连续的在线可靠性分析.Pietrantuono,Russo 和 Trivedi[1]结合静态可靠性模型与动态分析方法来预估运行时系统可靠性趋势.Wang 等人[31]通过运行时数据以及结合动态分析方法与软件架构来预测系统可靠性.这些方法考虑了软件运行时系统可靠性的动态变化,运用变化来调整可靠性预测.然而这些方法只是预测系统实时的可靠性,并未使用在线数据进行从系统实时状态到相隔一个时间段后系统的可靠性预测.

- 系统可靠性在线提高

目前也有一些研究来实现此目标,比如 Cooray[3]等人提出的通过在执行时加入包括系统的执行环境等各种信息资源来不断地提供精确的可靠性预测.通过对组件的再分配进程和改变组件副本的数量对系统进行重新配置,进而得到系统的最优配置,从而提高系统的在线可靠性.

他们将软件可靠性提高问题转化为一个优化问题,该优化问题是一个 NP 问题.该优化问题的解空间呈指

数增长:$O((\max\{w_i\}\times h)^t)$,$w_i$为组件$i$最大可复制数,$h$表示过程数,$t$为组件数.Cooray等人首先通过最大变异优先策略(即,能产生最大可靠性的改变)获得可以达到可靠性要求的配置方案,然后再通过比较邻近配置方案,选择一个较好的配置方案.Caporuscio 等人[32]研究了基于组件的软件系统的性能提高问题,通过系统配置(增加组件、移除组件、替代组件、修改参数等)来提高性能指标,并用贪婪策略选择最优配置方案.

- 软件自适应

研究者对软件自适应研究较多,如Chen等人在2008年举办的自适应系统软件工程讨论班上总结了自适应软件模型应具备的特征或维(dimensions),分别从目标(goal)、变化(change)、机制(mechanism)、效果(effects)这4个方面进行了讨论[33];Epifani 等人通过在运行阶段调整模型的参数来更新模型,从而自动地提高软件的非功能性特性,如可靠性和性能[34];Mohammad 等人[12]提出了在动态配置过程保证系统可靠性的方法——并行转换,即:通过部署一个空闲资源,同时允许对新组件和旧组件进行并发操作.Magee 等人[35]根据 3 层模型,依据高层目标设计自适应软件架构,对任务进行合成.Garlan和Schmerl[36]通过描述架构类型和修复策略来实现动态变化.Cheng[16]设计了一个叫做 Rainbow 的框架,它具备在运行时监视、检测、决定、实施的自适应能力.孙熙等人[37]将软件 Agent 技术和构件技术结合起来,使得构件能够根据环境的状态调整自己的行为.面向服务的架构通过服务组合也支持实现自适应软件,如:Irmert 等人[38]利用面向服务组件模型提出了(CoBRA)架构,用来支持动态适应;马晓星等人[39]提出了一种面向服务的动态协同架构,让应用系统能够灵活地动态演化,以适应底层Internet 计算环境和用户需求的变化;Wang[40]提出了一个规则模型,该模型可以用来提取程序中离散的规则,从而提高软件的自适应能力;Ma 等人[41]提出了一种用可自动地从构件的实施中提取的米利机来管理构件之间的动态依赖性,从而便于基于构件的系统进行实时重构.

然而,上述研究有如下两个方面的不足.

(1) 可靠性预测方面

这些方法只是预测系统的可靠性,而无法根据实时运行数据和系统当前状态来预测系统在未来某个具体时间间隔的可靠性.

(2) 可靠性提高方面

当系统的可靠性降低时,上述相关研究者只是提出了一些通用的系统重新配置的方法,并未具体分析、定位引起可靠性降低的组件.

因此,本文首先根据以前工作中通过时间序列分析模型 ARIMA 来预测系统在未来某个具体时间间隔的可靠性.当系统可靠性降低时,我们首先通过基于频谱的错误定位技术定位出导致系统可靠性下降的故障组件,进而对故障组件进行在线重新配置,从而在线提高了系统可靠性.

6 结 论

在本文中,我们利用端口失效数据来预测在线服务组合系统的可靠性,通过改进的基于频谱的错误定位技术来定位出故障组件,系统自动重新配置:或者添加新组件,或者替换出故障组件,从而在线提高系统的可靠性,这样可以解决面向服务的软件的可靠性降低的问题.这种方法与现有技术的不同之处在于:它不仅可以在线预测系统运行不同时段的可靠性,而且还可以在系统可靠性降低而不满足可靠性需求时,定位出故障组件,重新配置系统以在线自动提高系统可靠性.

本文主要进行了如下两个方面的创新.

- 目前已有一些关于可靠性预测和可靠性提高的研究,但是这些研究没有考虑引起可靠性降低的原因.本文提出,通过故障组件定位来对系统进行自适应配置以提高系统的可靠性.因此,系统配置具有针对性;
- 改进了 Abreu 等人提出的基于频谱的组件定位方法.我们考虑各个组件被执行的次数,而不仅仅是简单的是否被执行,因为不同组件被执行的次数将对定位的准确性具有较大的影响.

在将来的工作中,我们将针对如下4个方面进行更进一步的研究.

(1) 由于实时收集的数据是与时间有关的,我们要考虑研究参数值的变化对可靠性的影响问题;

(2) 采用更多的例子来验证预测模型的准确度;

(3) 考虑组件间的交互而引起的假性正确问题;

(4) 综合考虑系统自适应配置的成本(例如时间、内存、可靠性提高效果),从而选择合适的系统配置方法.

References:

[1] Pietrantuono R, Russo S, Trivedi KS. Online monitoring of software system reliability. In: Proc. of the 2010 European Dependable Computing Conf. IEEE, 2010. 209−218. [doi: 10.1109/EDCC.2010.33]

[2] Malek S, Roshandel R, Kilgore D, Elhag I. Improving the reliability of mobile software systems through continuous analysis and proactive reconfiguration. In: Proc. of the 31st Int'l Conf. on Software Engineering-Companion, Vol. 2009. 2009. 275−278. [doi: 10.1109/ICSE-COMPANION.2009.5071000]

[3] Cooray D, Kouroshfar E, Malek S, Roshandel R. Proactive self-adaptation for improving the reliability of mission-critical, embedded, and mobile software. IEEE Trans. on Software Engineering, 2013,39(12):1714−1735. [doi: 10.1109/TSE.2013.36]

[4] Ding ZH, Xu T, Chen MH. Online reliability prediction of services composition. In: Proc. of the 2014 3rd Int'l Workshop on Evidential Assessment of Software Technologies, Vol.264. 2014. 340−348. [doi: 10.1145/2627508.2627509]

[5] Ding ZH, Jiang MY. Port based reliability computing for service composition. IEEE Trans. on Service Computing, 2011,5(3): 422−436. [doi: 10.1109/TSC.2011.17]

[6] Jia ZY, Kang R. Forecasting method of software reliability based on ARIMA model. Computer Engineering and Applications, 2008, 44(35):17−19 (in Chinese with English abstract).

[7] Abreu R, Zoeteweij P, van Gemund AJC. A new bayesian approach to multiple intermittent fault diagnosis. In: Proc. of the Int'l Joint Conf. on Artificial Intelligence. 2009. 653−658.

[8] Abreu R, Zoeteweij P, van Gemund AJC. On the accuracy of spectrum-based fault localization. In: Proc. of the Testing: Academic and Industrial Conf. on Practice and Research Techniques——MUTATION 2007. IEEE, 2007. 89−98. [doi: 10.1109/TAIC.PART.2007.13]

[9] Abreu R, van Gemund AJC. Diagnosing multiple intermittent failures using maximum likelihood estimation. Artificial Intelligence, 2010,174(18):1481−1497. [doi: 10.1016/j.artint.2010.09.003]

[10] Yan GH, Han YH, Li XW. ReviveNet: A self-adaptive architecture for improving lifetime reliability via localized timing adaptation. IEEE Trans. on Computers, 2011,60(9):1219−1232. [doi: 10.1109/TC.2011.33]

[11] Peyman O, Nenad M, Richard NT. Architecture-Based runtime software evolution. In: Proc. of the 20th Int'l Conf. on Software Engineering. IEEE, 1998. 177−186. [doi: 10.1109/ICSE.1998.671114]

[12] Mohammad G, Abbas H. Partial scalability to ensure reliable dynamic reconfiguration. In: Proc. of the 2013 IEEE 7th Int'l Conf. on Self-Adaptation and Self-Organizing Systems Workshops. IEEE, 2013. 83−88. [doi: 10.1109/SASOW.2013.14]

[13] Cao K, Jiang H, Chen GH, Cui P, Xiong T. Self-Adaptive induced mutation algorithm for reconfigurable antenna system. IEEE Antennas and Wireless Propagation Letters, 2014,13:237−240. [doi: 10.1109/LAWP.2014.2302315]

[14] Xie CL, Li BX, Su ZY. WSDG-Based reliability prediction approach for Web service composition. Journal of Southeast University (Natural Science Edition), 2012,42(6):1074−1079 (in Chinese with English abstract).

[15] Cheung RC. A user-oriented software reliability model. IEEE Trans. on Software Engineering, 1980,6(2):118−125. [doi: 10.1109/TSE.1980.234477]

[16] Garlan D, Cheng S, Huang A, Schmerl B, Steenkiste P. Rainbow: Architecture-Based self-adaptation with reusable infrastructure. Computer, 2004,37(10):46−54. [doi: 10.1109/MC.2004.175]

[17] Gokhale SS. Architecture-Based software reliability analysis: Overview and limitations. IEEE Trans. on Dependable and Secure Computing, 2007,4(1):32−40. [doi: 10.1109/TDSC.2007.4]

[18] Goseva-Popstojanova K, Hassan A, Guedem A, Abdelmoez W, Nassar DEM, Ammar H, Mili A. Architectural level risk analysis using UML. IEEE Trans. on Software Engineering, 2003,29(10):946−960. [doi: 10.1109/TSE.2003.1237174]

[19] Goseva-Popstojanova K, Trivedi KS. Architecture-Based approaches to software reliability prediction. Computer & Mathematics with Applications, 2003,46(7):1023−1036. [doi: 10.1016/S0898-1221(03)90116-7]

[20] Immonen A, Niemela E. Survey of reliability and availability prediction methods from the viewpoint of software architecture. Software and Systems Modeling, 2008,7(1):49−65. [doi: 10.1007/s10270-006-0040-x]

[21] Kramer J, Magee J. Self-Managed systems: An architectural challenge. In: Proc. of the IEEE Future of Software Engineering. 2007. 259−268. [doi: 10.1109/FOSE.2007.19]

[22] Krishnamurthy S, Mathur A. On the estimation of reliability of a software system using reliabilities of its components. In: Proc. of the 8th Int'l Symp. on Software Reliability Engineering. IEEE, 1997. 146−155. [doi: 10.1109/ISSRE.1997.630860]

[23] Reussner RH, Schmidt HW, Poernomo IH. Reliability prediction for component-based software architectures. Journal of Systems and Software, 2003,66(3):241−252. [doi: 10.1016/S0164-1212(02)00080-8]

[24] Rodrigues G, Rosenblum D, Uchitel S. Using scenarios to predict the reliability of concurrent component-based software systems. In: Cerioli M, ed. Proc. of the Fundamental Approaches to Software Engineering. Berlin, Heidelberg: Springer-Verlag, 2005. 111−126. [doi: 10.1007/978-3-540-31984-9_9]

[25] Roshandel R, Medvidovic N, Golubchik L. A Bayesian model for predicting reliability of software systems at the architectural level. In: Overhage S, ed. Proc. of the Software Architectures, Components, and Applications. Berlin, Heidelberg: Springer-Verlag, 2007. 108−126. [doi: 10.1007/978-3-540-77619-2_7]

[26] Singh H, Cortellessa V, Cukic B, Gunel E, Bharadwaj V. A Bayesian approach to reliability prediction and assessment of component based systems. In: Proc. of the 12th Int'l Symp. on Software Reliability Engineering. IEEE, 2001. 12−21. [doi: 10.1109/ISSRE.2001.989454]

[27] Wang WL, Pan D, Chen MH. Architecture-Based software reliability modeling. Journal of Systems and Software, 2006,79(1): 132−146. [doi: 10.1016/j.jss.2005.09.004]

[28] Hu H, Jiang CH, Cai KY, Wong WE, Mathur AP. Enhancing software reliability estimates using modified adaptive testing. Information and Software Technology, 2013,55(2):288−300. [doi: 10.1016/j.infsof.2012.08.012]

[29] Gunawan I. Reliability prediction of distributed systems using Monte Carlo method. Int'l Journal of Reliability and Safety, 2013, 7(3):235−248. [doi: 10.1504/IJRS.2013.057092]

[30] Epifani I, Ghezzi C, Mirandola R, Tamburrelli G. Model evolution by run-time parameter adaptation. In: Proc. of the 2009 31st Int'l Conf. on Software Engineering. IEEE, 2009. 111−121. [doi: 10.1109/ICSE.2009.5070513]

[31] Wang WL, Hemminger TL, Tang MH. A moving average modeling approach for computing component-based software reliability growth trends. INFOCOMP Journal of Computer Science, 2006,5(3):9−18.

[32] Caporuscio M, Marco AD, Inverardi P. Model-Based system reconfiguration for dynamic performance management. The Journal of Systems and Software, 2007,80:455−473. [doi: 10.1016/j.jss.2006.07.039]

[33] Cheng BHC, de Lemos R, Giese H, Inverardi P, Magee J. Software engineering for self-adaptive systems: A research roadmap. In: Proc. of the Software Engineering for Self- Adaptive Systems. Springer-Verlag, 2009. 1−26. [doi: 10.1007/978-3-642-02161-9_1]

[34] Epifani I, Ghezzi C, Mirandola R. Model evolution by run-time parameter adaptation. In: Proc. of the 21st Int'l Conf. on Software Engineering. IEEE, 2009. 111−121. [doi: 10.1109/ICSE.2009.5070513]

[35] Kramer J, Magee J. Self-Managed systems: An architectural challenge. In: Proc. of the 2007 Future of Software Engineering. IEEE, 2007. 259−268. [doi: 10.1109/FOSE.2007.19]

[36] Garlan D, Schmerl B. RAINBOW: Architecture-Based adaptation of complex systems. Technical Report, AFRL-IF-RS-TR-2005-121, New York: Carnegie-Mellon Univ Pittsburgh Pa, 2005. 1−43.

[37] Sun X, Zhuang L, Liu W, Jiao WP, Mei H. A customizable running support framework for autonomous components. Ruan Jian Xue Bao/Journal of Software, 2008,19(3):1340−1349 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1340.htm

[38] Irmert F, Fischer T, Meyer-Wegener K. Runtime adaptation in a service-oriented component model. In: Proc. of the 2008 Int'l Workshop on Software Engineering for Adaptive and Self-Managing Systems. ACM Press, 2008. 97−104. [doi: 10.1145/1370018.1370036]

[39] Ma XX, Yu P, Tao XP, Lü J. A service-oriented dynamic coordination architecture and its supporting system. Chinese Journal of Computers, 2005,28(4):467−477 (in Chinese with English abstract).

[40] Wang QX. Towards a rule model for self-adaptive software. ACM SIGSOFT Software Engineering Notes, 2005,30(1):1−8. [doi: 10.1145/1039174.1039198]

[41] Su P, Cao C, Ma XX. Automated management of dynamic component dependency for runtime system reconfiguration. In: Proc. of the 2013 20th Asia-Pacific Software Engineering Conf. IEEE, 2013. 450−458. [doi: 10.1109/APSEC.2013.66]

附中文参考文献:

[6] 贾治宇,康锐.软件可靠性预测的 ARIMA 方法研究.计算机工程与应用,2008,44(35):17−19.

[14] 谢春丽,李必信,苏志勇.基于 WSDG 的 Web 服务组合可靠性预测.东南大学学报:自然科学版,2012,42(6):1074−1079.

[37] 孙熙,庄磊,刘文,焦文品,梅宏.一种可定制的自主构件运行支撑框架.软件学报,2008,19(6):1340−1349. http://www.jos.org.cn/1000-9825/19/1340.htm

[39] 马晓星,余萍,陶先平,吕建.一种面向服务的动态协同架构及其支撑平台.计算机学报,2005,28(4):467−477.

杨晓燕(1990－),女,河南信阳人,硕士生,主要研究领域为错误定位,软件测试和可靠性,软件自适应控制系统.

周远(1989－),男,硕士生,主要研究领域为软件建模,模型检查,软件测试和可靠性.

丁佐华(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件测试与可靠性,软件建模与分析,软件自适应控制系统,智能计算及应用.