doi:10.11959/j.issn.1000-436x.2016048

基于检查点的分布式软件监控与可信性评价

李珍[1]，田俊峰[1]，常卓[1]，马晓雪[2]
（1. 河北大学计算机科学与技术学院，河北 保定 071000；2. 河北大学计算机教学部，河北 保定 071000）

摘 要：基于检查点的传统软件可信性评价方法以及目前针对分布式软件的交互关联规则，对于具有复杂交互行为的分布式软件均不适用。采用伴随式分布式软件监控机制，在节点内织入 3 类检查点，引入适应复杂交互场景的交互关联规则。通过将节点分解为多层模块结构，进行基于检查点结构树的节点实例可信性以及基于节点的分布式软件可信性评价。实验表明能以较小的监控开销，更准确地评价分布式软件实例的可信性，能够处理无限路径，且不存在大状态空间问题。

关键词：分布式软件；可信性；检查点；交互；行为监控
中图分类号：TP393.08 **文献标识码：**A

Distributed software monitoring and trustworthiness evaluation based on checkpoints

LI Zhen[1], TIAN Jun-feng[1], CHANG Zhuo[1], MA Xiao-xue[2]
(1. School of Computer Science and Technology, Hebei University, Baoding 071000, China;
2. Computer Department, Hebei University, Baoding 071000, China)

Abstract: Traditional software trustworthiness evaluation based on checkpoints and the existing interaction association rules for distributed software were inapplicable to distributed software with complex interactions. According to this problem, an accompanying distributed software monitoring mechanism was used, and three types of checkpoints in nodes and interaction association rules were introduced for complex interaction scene. The trustworthiness of the node instance was evaluated based on checkpoint structure tree by dividing nodes into several modules, and then the trustworthiness of distributed software was evaluated based on nodes. The experimental results showed that the approaches could evaluate the trustworthiness of distributed software instance accurately with small monitoring cost, and could be suitable for infinitely long paths with no state explosion problem for trustworthiness evaluation of distributed software.

Key words: distributed software, trustworthiness, checkpoint, interaction, behavior monitoring

1 引言

分布式软件系统在金融、交通、航空、国防、工业控制等领域中起着越来越重要的作用，如电信和金融领域的服务系统、交通控制系统以及电子商务系统等已广泛渗透到人们的工作和生活中。随着分布式系统的规模越来越大、功能越来越复杂，人们对其可用性、可靠性和安全性等可信性质给予了更高的期望和要求。

在开放、动态的网络环境下，多个软件实体（如构件、Agent、Web 服务等）按需聚合，在满足约束的条件下相互协同来完成计算任务。大型分布式软件系统的深层次 Bug 多数是由于复杂的并发交互引起的[1]，由于功能逻辑复杂、节点间协作交互频

收稿日期：2015-01-25；修回日期：2015-11-02
基金项目：国家自然科学基金资助项目（No.61170254）；河北省自然科学基金资助项目（No.F2015201089, No.F2014201099）；河北省高等学校科学技术研究青年基金项目（No.QN2016149）；河北大学自然科学研究计划基金资助项目（No.2013-250）
Foundation Items: The National Natural Science Foundation of China (No.61170254), The Natural Science Foundation of Hebei Province (No.F2015201089, No.F2014201099), The Youth Foundation of Hebei Educational Committee (No.QN2016149), The Science Foundation of Hebei University(No.2013-250)

繁，导致软件运行时行为复杂且难控。然而分布式软件系统运行时，每个节点的行为和节点间的交互必然涉及一系列的信息和状态变化，因此，通过这些信息对分布式软件运行时行为进行分析，是监控软件自身状态的有效途径，是保证分布式软件可信运行的重要支撑方法。

检查点能够保存和恢复程序的运行状态，在进程迁移、容错、卷回调试等领域有着重要的应用。通过在软件行为轨迹中植入若干检查点，一些学者将检查点应用于软件的行为监控及可信性评价。杨晓晖等[2]以行为轨迹和检查点场景来刻画软件行为的动态特性，通过计算系统调用上下文值以及构造系统调用参数关系约束规则来评测行为轨迹和检查点场景的偏离程度，构建基于软件行为自动机的动态可信评测模型。刘玉玲等[3]通过累计多个有疑似风险的检查点，利用风险评估策略，判定有疑似风险的检查点，并采用奖惩或处罚机制来得到软件行为的可信度。田俊峰等[4]通过动态监测行为，对软件及其模块在一段时间内运行的可信状况进行研究，提出了基于马尔可夫的检查点可信评估模型。然而这些方法都不是针对分布式软件设计的，对分布式软件复杂交互行为的可信性评价无能为力。伴随式的分布式软件系统监控方法（如 MERF[5]、PIP[1]），由于系统运行和监控相互独立、可扩展性强，广泛应用到大规模的分布式软件系统。如何利用现有的分布式软件监控技术，基于检查点思想，提出适用于分布式软件行为的可信性评价方法显得尤为重要。

近年来，分布式软件的可信性尤其是对分布式软件中交互行为可信性的研究受到了学者们广泛的关注和重视。文志诚等[6]针对分布式软件运行时的外在表现特征，根据具体交互场景建立贝叶斯网模型，在上下文环境中通过监测相关的数据来对软件行为运行时可信性进行分析。彭成等[7]提出了基于网络化软件交互行为的动态建模方法，给出一种基于不变量约束规则的挖掘方法，从监控收集的软件交互行为日志中挖掘出 6 类不变模式。Wang 等[8]提出了构件交互模式软件可靠性模型，给出了不满足随机马尔可夫性质的交互模式和软件体系结构的表示方法。Yang 等[9]提出了基于随机 Petri 网的构件软件安全性模型和评价方法，在软件设计阶段对软件安全性进行预测。Tyagi 等[10]结合神经网络和模糊逻辑 2 种软计算技术来评估构件软件的可靠性。Chen 等[11]针对网构软件提出了一种基于交互的需求监控方法，通过监控网构软件运行时的行为来检测与软件需求规范的偏离，其交互规则主要考查交互执行的先后顺序。上述研究中分布式软件交互间的关联规则都局限于无条件下的因果和并发类型，规则的不准确、不全面无法适应复杂交互场景的可信性分析。

基于体系结构进行分布式软件可信性评价的方法可以借鉴基于体系结构的软件可靠性模型，主要包括基于状态和基于路径 2 种方法[12]。基于状态的模型[13,14]可以分析解决潜在的无限条路径的系统，但在对复杂系统建模时会产生难以处理的大状态空间，使计算和分析非常复杂甚至不可行。基于路径的模型[15,16]考虑程序的可能执行路径，因为路径的数目受到测试期间观察的限制，只能用来分析解决有限条路径的系统。尽管这 2 种方法存在各自的优缺点，但对复杂的分布式软件，现有的研究很少将 2 种方法有效地结合起来。

针对上述问题，本文基于文献[5]中伴随式的分布式软件监控的实现机制，在节点内植入检查点，引入适应复杂交互场景的交互关联规则，提出了检查点可信性及检查点间结构可信性的评价方法；通过将节点分解为多层模块结构，提出了基于检查点结构树的节点及其实例可信性评价方法，并在此基础上评价分布式软件及其实例的可信性。

2 分布式软件行为分析框架

定义 1(分布式软件) 分布式软件 DS 定义为在由通信网络互连的多个节点上通过通信和动作协调来执行任务的软件，表示为一个三元组 $DS=<NS, IS, RS>$，NS 为节点集合 $NS=\{Node_i|i=1,2,3,\cdots\}$，$IS$ 为节点间交互的集合 $IS=\{iaction_i|i=1,2,3,\cdots\}$，$RS$ 为节点间的交互规则集。分布式软件运行时的一条运行轨迹称为一个分布式软件实例。分布式软件运行时，可能有多个分布式软件实例并发执行。

定义 2(节点) 节点 $Node$ 定义为分布式软件 DS 中通过通信网络互连来协同工作的计算机，表示为 $Node=<CP, TR, R, S_0, E, P>$，其中，$CP=\{cp_i|i=1,2,3,\cdots\}$为检查点集合；$TR=\{tr_{I,J}|I,J\subseteq\{1,2,\cdots\}\}$为转移集合，$tr_{I,J}$表示 cp_I向 cp_J的转移（$cp_I=\{cp_i|i\in I\}$，$cp_J=\{cp_j|j\in J\}$），I 和 J 中元素用空格相间；R 为节点中检查点间的结构，由顺序 S、并行 P、选择 B 和循环 L 组合而成；$S_0\subseteq CP$ 为起始检查点集合；$E\subseteq CP$ 为终止检查点集合；$P=\{p_{I,J}|I,J\subseteq\{1,2,\cdots\}\}$为转移 $tr_{I,J}$ 发生概率的集合，$p_{I,J}\in[0,1]$。节

点 *Node* 内的一条运行轨迹称为一个节点实例，节点实例可表示为一个顺序或并发执行的检查点序列。分布式软件运行时，可能有多个相同或不同的节点实例并发执行。

节点中设置若干检查点，通过对分布式软件运行时检查点及其属性的监控，来对节点行为的可信性进行评价。检查点包括以下 3 类。

1) 功能检查点。在节点的一个独立基本功能结束处设置，需记录功能检查点的属性信息，包括检查点功能、上下文、参数策略、时间间距、内存占用率、CPU 占用率等。功能检查点的设置粒度决定了分布式软件行为检查的粒度，也就决定了分布式软件行为可信的程度，需要和运行效率进行平衡。

2) 分支检查点。在节点行为分支处设置，属性主要是分支的条件信息。

3) 交互检查点。当分布式软件运行需要节点间交互时，在节点每个交互发送的起始位置设置交互检查点，来捕获当前交互的属性信息（交互的属性详见定义 3）。

本文对分布式软件节点中的检查点结构进行抽象，采用 Petri 网中的库所表示检查点，用迁移表示检查点间的转移。为了构造检查点间的结构，定义如下基本组合：sequence、AND-split、AND-join、OR-split、OR-join，如图 1 所示。若 cp_i 为 cp_j 的前一检查点，则称 cp_i 为 cp_j 的前向检查点，cp_j 为 cp_i 的后向检查点。设检查点 cp_f, cp_k 分别是 cp_i, cp_j 的共同前向和后向检查点。

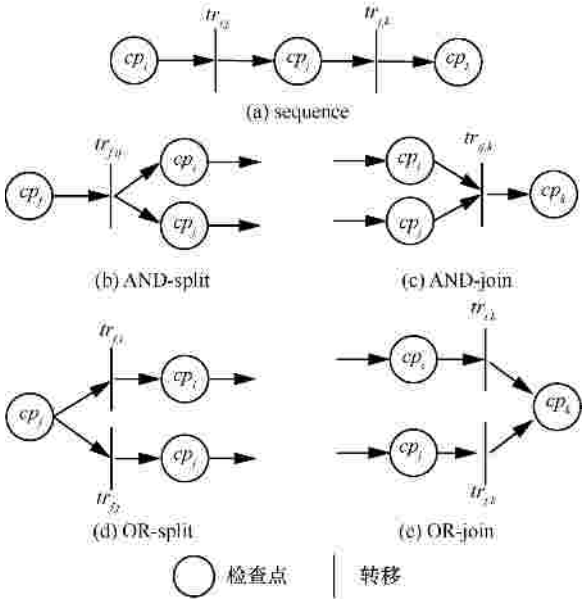

图 1　检查点的基本组合

sequence：检查点 cp_i, cp_j, cp_k 按顺序依次执行；

AND-split：通过 $tr_{f,ij}$ 由 cp_f 转移到 cp_i 和 cp_j；

AND-join：通过 tr_{ijk} 由 cp_i 和 cp_j 转移到 cp_k；

OR-split： cp_f 通过 $tr_{f,i}$ 转移到 cp_i，或通过 tr_{fj} 转移到 cp_j；

OR-join：cp_i 通过 $tr_{i,k}$ 转移到 cp_k，或 cp_j 通过 tr_{jk} 转移到 cp_k。

定义 3(交互) 交互定义为分布式软件 *DS* 中节点间的一次通信，表示为 c=<*sender*, *receiver*, *msgName*, *msgSort*, *arguPolicy*, *condition*, *sendTime*, *recvTime* >。其中，每个元素为交互的一个属性。*sender* 和 *receiver* 分别表示发送方和接收方；*msgName* 为交互时发送的消息名称；*msgSort* 为交互的类型，包括 2 种交互：同步交互 *synch* 和异步交互 *asynch*，同步交互后都跟着一个返回消息，返回接收方接收该交互的时刻，发送方收到返回消息后才继续后面的操作，而异步消息后无返回消息；*arguPolicy* 为参数策略，可以是关于单一参数的关系，采用枚举方式列出允许出现的参数值，也可以是关于 2 个交互参数之间的关系；*condition* 为交互触发的条件；*sendTime* 和 *recvTime* 分别表示交互的发送时刻和接收时刻，异步消息无需记录接收时刻。一次分布式软件运行时节点间的交互称为交互实例。本文假定节点间交互开始后即可顺利结束。

下面给出分布式软件行为可信性分析的框架，如图 2 所示。对每个节点，设置一个节点监控器，在节点中部署监控探针（probe），在运行时获取检查点的属性值（对交互检查点，还需获取交互间关联），缓存在本地存储介质（例如磁盘）中，经过本地 Agent 异步汇集到节点分析器。对每个节点，设置一个节点分析器，负责从各节点监控器收集该节点内的监控信息，实时地生成该节点实例的可信性报告，并通过记录器将收集的监控信息保存起来，当进行节点可信性分析时再从记录器中提取多次运行的监控信息。对整个分布式软件，设置一个软件分析器，负责从节点分析器收集节点实例可信性报告或节点可信性报告，生成分布式软件实例可信性报告或分布式软件可信性分析报告。节点分析器或软件分析器都可以将可信性分析结果通过显示/报警器显示给系统管理员，当某节点或软件分析结果为不可信时，通过显示/报警器报警，系统管理和维护人员可依据分析结果对节点实施调控。

分布式软件行为的监控是通过编写监控探针，

采用面向方面编程 AOP 编织工具[5]将监控方面代码注入分布式软件检查点处，生成具备被监控能力的系统。探针代码集中在一个独立的文件中，较好地实现了方面分割，降低软件维护的代价。同时，通过对检查点的监控与原软件并发执行，降低监控带来的时间开销。分布式软件行为的可信性分析过程由节点分析器和软件分析器负责完成，流程如图 3 所示。首先评价分布式软件中各节点的检查点可信性，在此基础上可以得到各节点实例的可信性以及检查点间结构的可信性。基于节点实例可信性即可评价分布式软件实例的可信性；基于检查点间结构的可信性来评价节点的可信性，据此进行分布式软件的可信性评价。

3 分布式软件行为的可信性分析

3.1 检查点的可信性

检查点的属性根据对检查点可信性的判断是否确定分为 2 种：确定性属性和模糊属性。确定性属性对检查点可信性的判断是确定的，一旦偏离正常值，则能直接决定检查点的可信性不在可接受范围，可信性的值为 0 或 1；而模糊属性无法用准确的数字表示，允许存在一定范围的误差，具有一定的模糊性，可信性的值为[0,1]。

功能检查点中的检查点功能、上下文、参数策略属性属于确定性属性，时间间距、内存占用率、CPU 占用率属性属于模糊属性；分支检查点和交互检查点的属性均属于确定性属性。有关根据确定性属性和模糊属性评价检查点可信性的方法，详见文献[17,18]。

交互检查点的可信性除了取决于交互检查点的属性外，还取决于交互关联的可信性。只有当交互检查点属性和交互关联均可信时，该交互检查点才可信，因此交互检查点的可信性可表示为交互检查点属性的可信性和交互关联可信性的乘积。下面重点介绍交互关联可信性的评价方法。交互关联规则有以下 4 种，如图 4 所示。

1) 因果

交互 $iaction_1$ 发生后的下一交互为 $iaction_2$，则两交互为因果关联，包括同步因果（$<_S$）和异步因果（$<_A$）。同步因果：当 $iaction_1 \cdot recvTime < iaction_2 \cdot sendTime$ 时，$iaction_1 <_S iaction_2$；异步因果：当 $iaction_1 \cdot sendTime < iaction_2 \cdot sendTime$ 时，$iaction_1 <_A iaction_2$。

2) 并发

2 个或多个交互同时执行，或至少重叠执行。当 $iaction1 \cdot sendTime \in [iaction2,\ sendTime,\ iaction_2 \cdot$

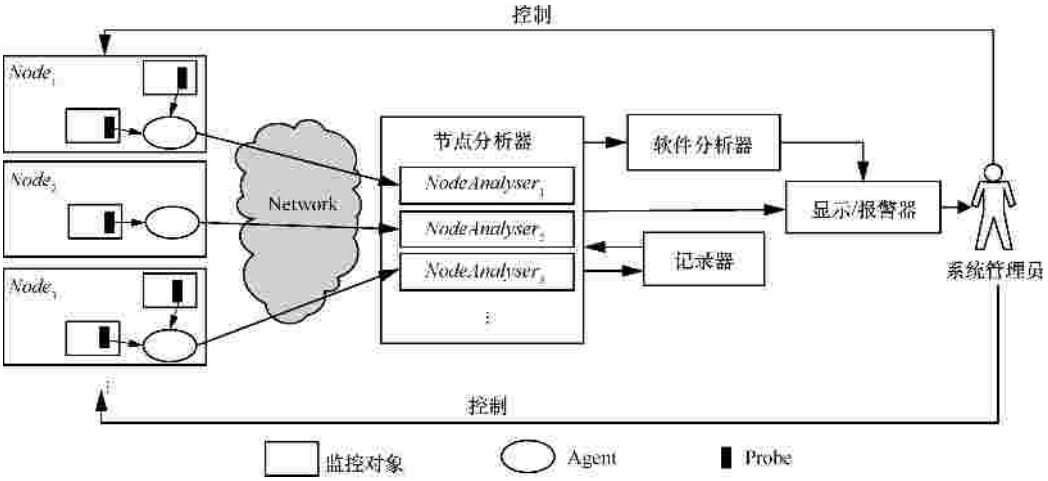

图 2 分布式软件行为可信性分析框架

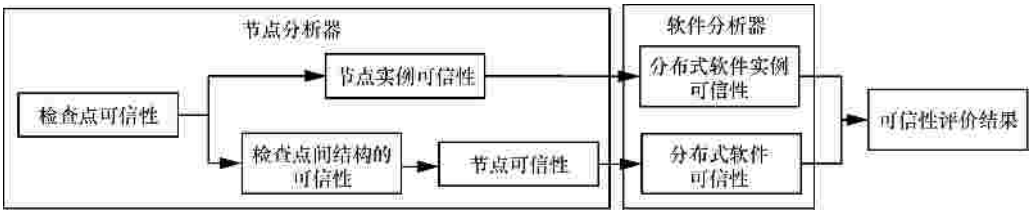

图 3 分布式软件行为可信性分析流程

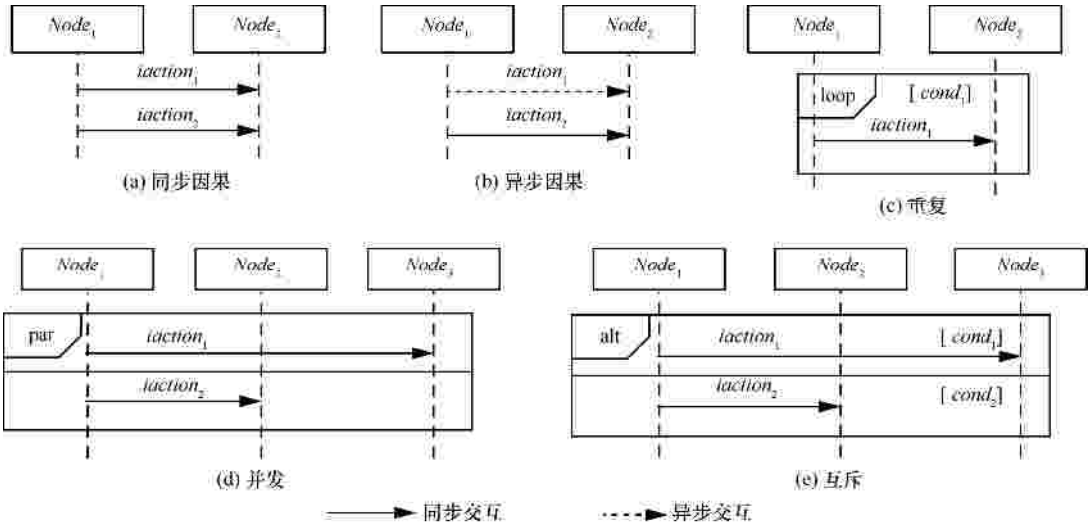

图 4　交互间的关联

$recvTime$]或 $iaction_2 \cdot sendTime \in [\ iaction_1 \cdot sendTime, iaction_1 \cdot recvTime]$时，交互 $iaction_1$ 与交互 $iaction_2$ 为并发关联，表示为 $iaction_1 \| iaction_2$。

3) 互斥

当条件 $cond_1$ 满足时，执行交互 $iaction_1$；当条件 $cond_2$ 满足时，执行交互 $iaction_2$，则两交互为互斥关联，表示为$+([cond_1]iaction_1, [cond_2]iaction_2)$。

4) 重复

当条件 $cond_1$ 满足时，循环执行交互 $iaction_1$，直到条件 $cond_1$ 不满足为止，循环执行的交互为重复关联，表示为$*[cond_1]\ (iaction_1)$。

分布式软件运行时，可能有多个分布式软件实例并发执行。每个节点动态维护一张分布式软件实例表，表中的一条记录对应一个分布式软件实例，该表的记录包括如下表项：标识、以该节点为接收方的交互和以该节点为发送方的交互。当一个分布式软件实例运行时，经过某个节点，则在该节点的分布式软件实例表中增加一条记录，当分布式软件实例运行结束时，删除各节点中对应标识的记录。多个节点中具有相同标识的记录构成一个分布式软件实例。在同一分布式软件实例上的交互，需满足对应的交互关联规则。若与交互关联规则不匹配，则交互关联的可信性值为 0，判为不可信，并给出原因；否则交互间关联的可信性值为 1，判为可信。交互关联的可信性分析算法如下。

算法 1　交互间关联的可信性分析算法

输入：交互检查点 ic_i 的交互集合 $IS_i=\{\ iaction_j \mid j=1,2,3,\cdots\}$，交互规则集 RS 和一次分布式软件运行中的交互实例集合 $IS_i'=\{\ iaction_j' \mid j=1,2,3,\cdots\}$。

输出：交互检查点 ic_i 的交互间关联的可信性值（0 或 1），若可信性值为 0，报告不可信的原因。

1) FOR EACH $iaction_j \in IS_i$

2) 查找 RS 中含 $iaction_j$ 的规则子集 RS_j；

3) FOR EACH $rul_s \in RS_j : iaction_j <_S iaction_k$ (or $iaction_k <_S iaction_j$)

4) 查找与 $iaction_j'$在同一分布式软件实例中的 $iaction_k'$；

5) IF $iaction_j' \cdot recvTime > iaction_k' \cdot sendTime$ (or $iaction_k' \cdot recvTime > iaction_j' \cdot sendTime$) THEN

6) 报告 $iaction_j'$和 $iaction_k'$执行顺序异常；

7) RETURN 0；

8) END FOR

9) FOR EACH $rul_s \in RS_j : iaction_j <_A iaction_k$ (or $iaction_k <_A iaction_j$)

10) 查找与 $iaction_j'$在同一分布式软件实例中的 $iaction_k'$；

11) IF $iaction_j' \cdot sendTime > iaction_k' \cdot sendTime$ (or $iaction_k' \cdot sendTime > iaction_j' \cdot sendTime$) THEN

12) 报告 $iaction_j'$和 $iaction_k'$执行顺序异常；

13) RETURN 0；

14) END FOR

15) FOR EACH $rul_s \in RS_j : iaction_j \| iaction_k$

16) 查找与 $iaction_j'$在同一分布式软件实例中的 $iaction_k'$；

17) IF $iaction_j'\cdot recvTime < iaction_k'\cdot sendTime$ or $iaction_j'\cdot sendTime > iaction_k'\cdot recvTime$ THEN

18) **报告 $iaction_j'$和 $iaction_k'$没有并发执行；**

19) RETURN 0；

20) END FOR

21) FOR EACH $rul_s \in RS_j : +([cond]iaction_j)$

22) IF *cond*==FALSE THEN

23) **报告 $iaction_j'$在错误的条件下执行；**

24) RETURN 0；

25) IF **执行次数** *?* >1 THEN

26) **报告执行次数异常；**

27) RETURN 0；

28) END FOR

29) FOR EACH $rul_s \in RS_j : *([cond]iaction_j)$

30) IF *cond*== FALSE THEN

31) **报告 $iaction_j'$在错误的条件下执行；**

32) RETURN 0；

33) END FOR

34) END FOR

35) RETURN 1；

36) END

3.2 检查点间的结构及其可信性

一个分布式软件节点中，检查点存在以下结构：顺序、并行、分支和循环。对于顺序、并行和分支结构，即具有有限条路径的结构，本文采用基于路径的模型[15,16]计算结构的可信性；对于具有潜在的无限路径的循环结构，采用基于状态的模型[13,14]计算循环结构的可信性。检查点 cp_i (i=1,2,…,n)的可信性记为 t_i 。

1) **顺序结构**（sequence）

具有顺序结构的检查点按顺序依次执行，如图 5(a)**所示。检查点序列的可信性为各检查点可信性的乘积，该序列发生的概率为各转移概率的乘积，则顺序结构的可信性可表示为检查点序列的可信性与该序列转移概率的乘积。图** 5(a)**所示的顺序结构可信性 T_S 为**

$$T_S = p_{1,2}\cdots p_{n-1,n}t_1t_2\cdots t_{n-1}t_n = \prod_{k=1}^{n-1}p_{k,k+1}\prod_{j=1}^{n}t_j \quad (1)$$

2) **并行结构**（parallel）

在并发环境下，常通过多个模块并发执行来提高系统的性能。并发结构的可信性可表示为并发执行的各检查点可信性的乘积。如图 5(b)**中的点划线框所示的并行结构的可信性 T_P 为**

$$T_P = t_2t_3\cdots t_{n-1} = \prod_{i=2}^{n-1}t_i \quad (2)$$

3) **分支结构**（branch）

具有分支结构的检查点某一时刻只有一个分支执行，如图 5(c)**中的点划线框所示的分支结构。每条分支是一个顺序结构，依据式**(1)**来计算各分支的可信性，而分支结构的可信性为各分支可信性的和。点划线框表示的分支结构的可信性 T_B 为**

$$T_B = p_{1,2}p_{2,n}t_2 + p_{1,3}p_{3,n}t_3 + \cdots + p_{1,n-1}p_{n-1,n}t_{n-1} = \sum_{i=2}^{n-1}p_{1,i}p_{i,n}t_i \quad (3)$$

4) **循环结构**（loop）

具有循环结构的检查点循环执行的次数大于等于 1**。为了更方便计算循环结构的可信性，本文在图** 5(d)**所示的循环结构前后分别增加入口检查点** entry **和出口检查点** exit**，这** 2 **个检查点的可信性 $t_{entry}=t_{exit}=1$，图** 6 **所示循环结构的可信性 T_L 为**

$$T_L = \begin{cases} \dfrac{(1-p_{1,1})t_1}{1-p_{1,1}t_1}, & n=1 \\ \dfrac{(1-p_{n,1})\prod_{i=1}^{n-1}p_{i,i+1}\prod_{j=1}^{n}t_j}{1-p_{n,1}\prod_{i=1}^{n-1}p_{i,i+1}\prod_{j=1}^{n}t_j}, & n>1 \end{cases} \quad (4)$$

下面给出循环结构可信性的计算过程。假设检查点间的交互具有 Markov **性质，软件体系结构采用离散时间** Markov **链（**DTMC**）表示。增加可信状态** T **和不可信状态** U**，如图** 7 **所示。令 $p_{i,j}t_i$ 表示检查点 cp_i 可信，并由 cp_i 到 cp_j 的转移概率。**

转移概率矩阵 Q 为

	entry	cp_1	cp_2	…	cp_{n-1}	cp_n	exit
entry	0	1	0	...	0	0	0
cp_1	0	0	$p_{1,2}t_1$	...	0	0	0
cp_2	0	0	0	...	0	0	0
…	...	...	...	...	...	...	...
cp_{n-1}	0	0	0	...	0	$p_{n-1,n}t_{n-1}$	0
cp_n	0	$p_{n,1}t_n$	0	...	0	0	$(1-p_{n,1})t_n$
exit	0	0	0	...	0	0	0

矩阵 Q 的 k 次幂的第 1 **行第** n+2 **列元素**

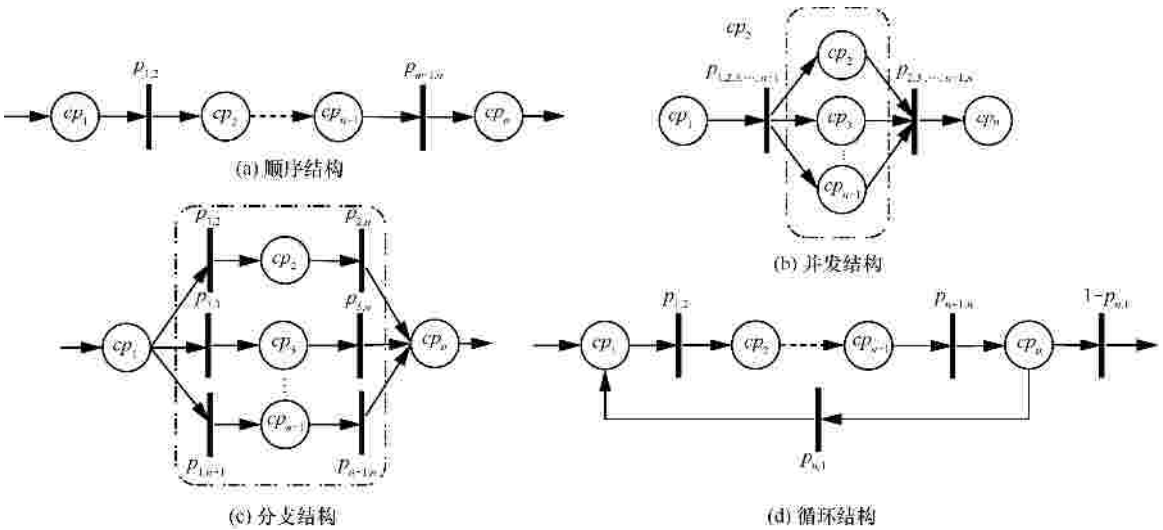

图 5　检查点间的结构

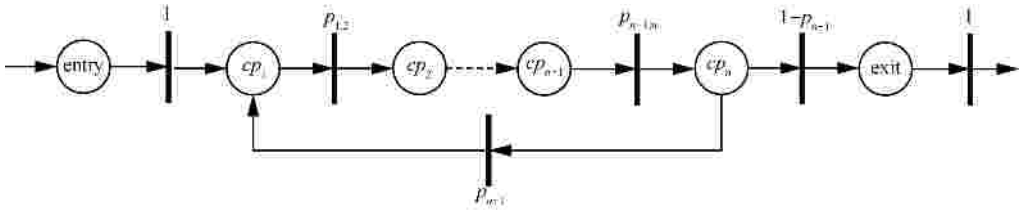

图 6　增加入口和出口检查点后的循环结构

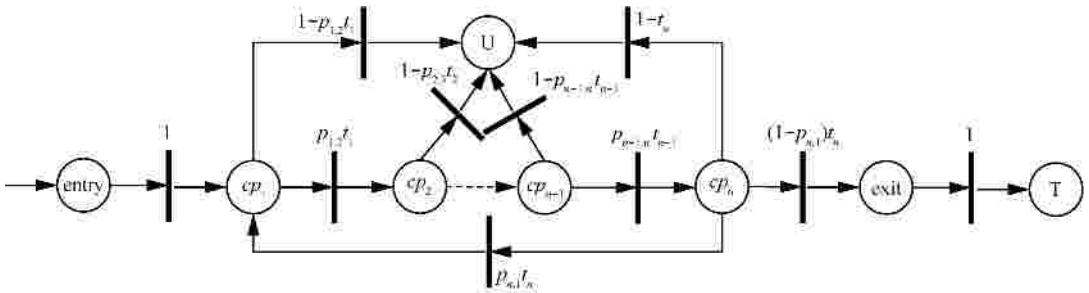

图 7　增加状态后的循环结构

$Q^k(1,n+2)$ 为从检查点 entry 经 k 步转移到检查点 exit 的概率。步数 k 的可能取值范围为 $[0,\infty)$。当至少有一个检查点 cp_1 时，有 $Q^0(1,n+2)=Q^1(1,n+2)=0$。

矩阵 $\boldsymbol{M}=\sum_{k=0}^{\infty}\boldsymbol{Q}^k=(\boldsymbol{I}-\boldsymbol{Q})^{-1}$，则矩阵 $\boldsymbol{M}$ 的第 1 行第 n+2 列元素 $\boldsymbol{M}(1,n+2)$ 为从检查点 entry 到检查点 exit 的概率

$$\boldsymbol{M}(1,n+1)=\begin{cases}\dfrac{(1-p_{1,1})t_1}{1-p_{1,1}t_1}, & n=1\\[2ex] \dfrac{(1-p_{n,1})\prod_{i=1}^{n-1}p_{i,i+1}\prod_{j=1}^{n}t_j}{1-p_{n,1}\prod_{i=1}^{n-1}p_{i,i+1}\prod_{j=1}^{n}t_j}, & n>1\end{cases}$$

由此得

$$T_{\mathrm{L}}=\boldsymbol{M}(1,n)t_{\mathrm{exit}}=\begin{cases}\dfrac{(1-p_{1,1})t_1}{1-p_{1,1}t_1}, & n=1\\[2ex] \dfrac{(1-p_{n,1})\prod_{i=1}^{n-1}p_{i,i+1}\prod_{j=1}^{n}t_j}{1-p_{n,1}\prod_{i=1}^{n-1}p_{i,i+1}\prod_{j=1}^{n}t_j}, & n>1\end{cases}\tag{5}$$

3.3　分布式软件的可信性

下面以图 8 所示的一个分布式软件在某节点上的检查点结构为例，进行节点的可信性分析。首先将节点按如下规则划分为若干层，每层包含若干模块：1）一个模块中包含 2 个或 2 个以上检查点（具有一个检查点的循环结构需多次执行该检查点，此

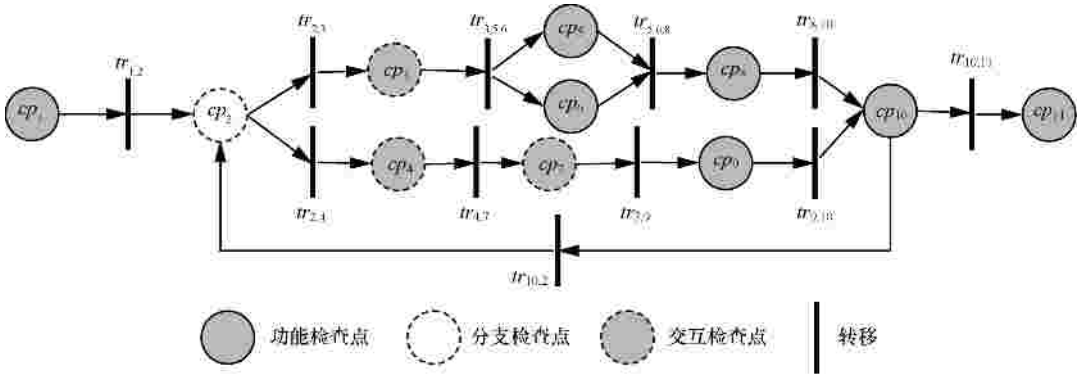

图 8 某节点的检查点结构

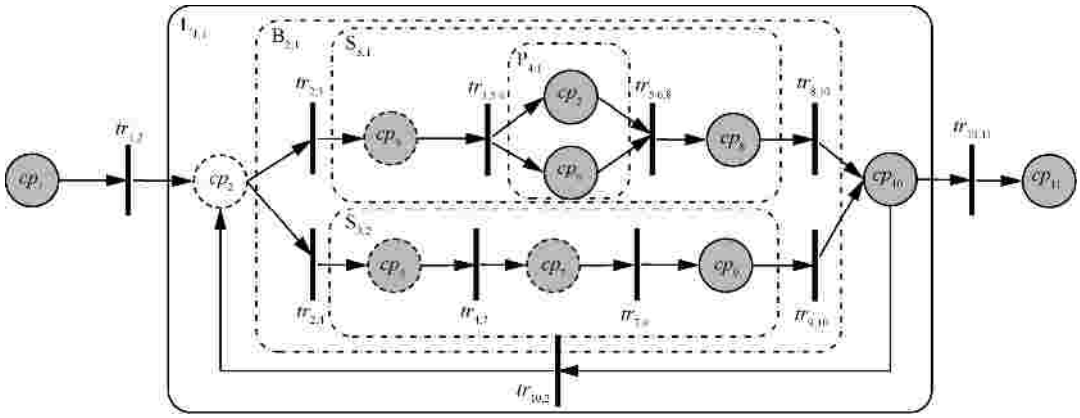

图 9 由模块表示的检查点结构

种情况认为包含多个检查点）；2）一个模块直接包含（即相邻下层）的模块或检查点仅以一种结构存在。图 9 用点划线标出了各层模块。模块或检查点间的结构 S：顺序，P：并发，B：分支，L：循环。为方便起见，本文将第 0 层记为 S_0，将第 i 层中具有顺序结构的第 j 个模块记为 $S_{i,j}$ $(i, j = 1, 2, \text{L})$，下标的第 1 位表示层数，第 2 位表示该层的模块号。其他结构类似表示。对于分支和循环结构，需在相邻层检查点对应连接边上标明条件。图 9 对应的检查点结构树如图 10 所示。

在节点的一次运行过程中，各检查点要么是顺序，要么是并发到达的。只有保证每个检查点都可信时，整个节点才可信。因此，对于 $Node_i$，节点实例的可信性 $T_{\text{ins}i}$ 等于节点一次运行过程中经过的所有检查点（叶子）的可信性的乘积。基于检查点结构树的节点实例可信性评价算法如下。

算法 2 基于检查点结构树的节点实例可信性评价算法

输入：节点 $Node_i$ 的检查点结构树的根指针 *pRoot*。

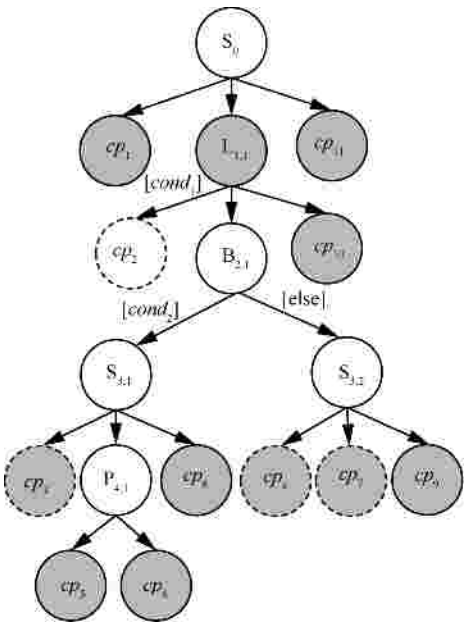

图 10 节点的检查点结构树

输出：节点实例的可信性 $T_{\text{ins}i}$，并报告不可信的检查点。

1) $T_{\text{ins}i}$=1；//初始化 $Node_i$ 节点实例的可信性

2) InitStack(*S*)；//初始化栈 *S*
3) Push(*S*, *pRoot*)； // *pRoot* 入栈 *S*
4) WHILE StackEmpty(*S*)==FALSE DO // 栈 *S* 不空
5) Pop(*S*, *lp*)；//栈顶指针 *lp* 出栈
6) IF *lp*->*type*==’S’ THEN //顺序结构
7) FOR EACH *lp*->*child*$_i$ 以从右到左顺序；
8) Push(*S*, *lp*->*child*$_i$)；
9) END FOR
10) IF *lp*->*type*==’P’ THEN //并发结构
11) FOR EACH *lp*->*child*$_i$ 以任一指定顺序；
12) Push(*S*, *lp*->*child*$_i$)；
13) END FOR
14) IF *lp*->*type*==’B’ THEN //分支结构
15) 查找满足当前条件 $cond_r$ 的孩子 *lp*->*child*$_k$；
16) Push(*S*, *lp*->*child*$_k$)；
17) IF *lp*->*type*==’L’ THEN //循环结构
18) WHILE $cond_s$==TRUE DO
19) FOR EACH *lp*->*child*$_i$ 以从右到左顺序；
20) Push(*S*, *lp*->*child*$_i$)；
21) END FOR
22) END WHILE
23) ELSE //检查点
24) 根据 3.1 节方法评价当前检查点的可信性 *t*；
25) IF *t* <*d* THEN //*d* 为检查点可信阈值
26) 报告当前检查点不可信；
27) T_{insi}=$T_{insi}t$；
28) END WHILE
29) RETURN T_{insi}；
30) END

基于检查点结构树，节点 $Node_i$ 的可信性 T_i 可递归的表示为

$$T_i = T_{S_0}(t_1, T_{L_{1,1}}[t_2, T_{B_{2,1}}(T_{S_{3,1}}[t_3, T_{P_{4,1}}(t_5, t_6), t_8], T_{S_{3,2}}[t_4, t_7, t_9]), t_{10}], t_{11}) \quad (6)$$

其中，检查点 cp_i 的可信性 t_i 可以用多次运行得到的检查点可信性的均值 $\bar{t}_i$ 来进行估计，得到节点的可信性估计 $\hat{T}_i$ 为

$$\hat{T}_i = T_{S_0}(\bar{t}_1, T_{L_{1,1}}[\bar{t}_2, T_{B_{2,1}}(T_{S_{3,1}}[\bar{t}_3, T_{P_{4,1}}(\bar{t}_5, \bar{t}_6), \bar{t}_8], T_{S_{3,2}}[\bar{t}_4, \bar{t}_7, \bar{t}_9]), \bar{t}_{10}], \bar{t}_{11}) \quad (7)$$

根据木桶理论[19]，分布式软件实例的可信性 T_{ins} 取决于分布式软件一次运行过程中最小的节点实例可信性，即 $T_{ins} = \min(T_{insi})$。分布式软件的可信性 T 为各节点的可信性的最小值，即 $T = \min(T_i)$。

4 实验与分析

以一个典型的分布式电子商务应用——网上购物系统为例进行实验。该系统是一个分布式客户端-服务器系统，采用 C#语言，基于 MVC（model-view-controller）架构模式开发。本文以网上购物系统中“下订单”功能模块为例，本文方法对系统其他模块均适用。“下订单”功能模块涉及到 4 个构件：Customer、Controller、Model 和 Supplier，它们分布在不同的节点上，节点间通过交互实现下订单功能，其顺序如图 11 所示。为了区分各交互，图中在交互名前加了序号。包括 3 个场景。

场景 1　客户下订单，供销商发货。

场景 2　客户下订单后，在供销商发货前修改订单，订单成功修改，供销商发货。

场景 3　客户下订单，供销商发货后客户修改订单，修改被拒绝。

实验中共有如下节点并发执行：运行 Customer 的节点 30 个，运行 Controller 和 Model 的节点各 2 个，运行 Supplier 的节点 5 个，节点的系统配置为 CPU 为 Intel Core 2 Duo E7500 2.93 GHz，内存为 2 GB。

4.1 交互间关联的可信性分析方法比较

以构件 Controller 所在节点为例，由于本文重点是分布式软件的交互，因此仅考虑交互检查点，有功能检查点和分支检查点时类似，检查点结构树如图 12 所示。当多个 Customer 并发执行时，形成多个分布式软件实例，下面以一个分布式软件为例，依次访问的交互为：*placeOrder*, *writeToDB*, *notifySupplier*, ([change order after shipment]) *changeOrder*, *writeToDB*, *notifySupplier*, *notifyUserOfShipment*, *writeToDB*, *notifyUser*。

本文方法与文献[7]和文献[11]中交互间关联的可信性分析比较如表 1 所示。该分布式软件实例满足文献[7]和文献[11]的交互关联规则，交互关联可

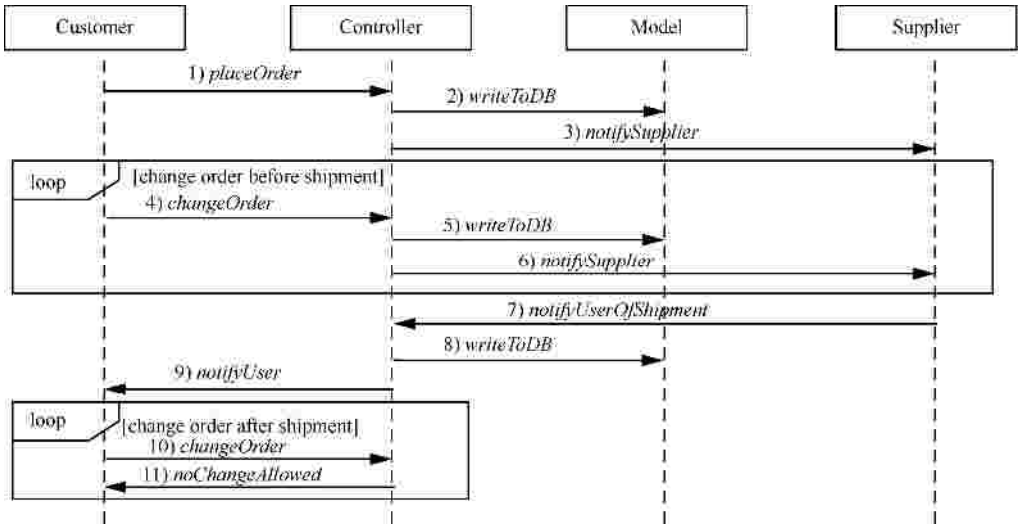

图 11 “下订单”顺序

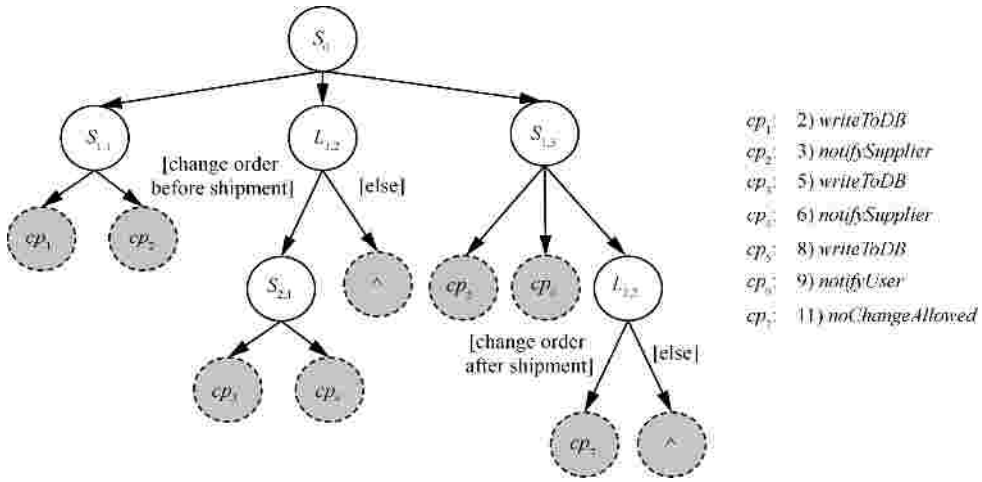

图 12 构件 Controller 所在节点的检查点结构树

信性值误判为 1（可信）。实际上，在[change order after shipment]条件下，*changeOrder* 的后续交互应为 *noChangeAllowed*。这个轨迹的偏离可以由本文算法 1 根据交互规则* [change order after shipment](*changeOrder* $<_S$ *noChangeAllowed*)检测出，得到交互关联可信性值为 0（不可信）。文献[7]和文献[11]除了没有考虑交互关联的条件外，在因果关联中也没有区分交互的同步和异步类型，将它们简单的视为一种情况，因此描述不准确。而本文方法能够区分这 2 种类型，例如，对于交互规则 *placeOrder* $<_S$ *writeToDB*，表示 *placeOrder*·*recvTime* $<_S$ *writeToDB*·*sendTime*，而 *placeOrder* $<_A$ *writeToDB*，表示 *placeOrder*·*sendTime* $<_A$ *writeToDB.sendTime*，从而能检测出这 2 种交互引起的行为偏离。此外，尽管文献[7]中给出了重复类型，但没有考虑交互重复的条件，本质上也属于因果类型，而本文方法考虑了交互重复的条件，能更准确地处理这些交互引起的行为偏离。

4.2 节点可信性评价方法比较

表 2 给出了基于体系结构的各类分布式软件节点可信性评价方法（基于路径的方法[15,16]、基于状态的方法[13,14]和本文基于检查点结构树方法）的比较，下面从以下几方面进行分析。

1) 容纳全部节点实例和处理无限路径

基于路径的方法由于路径的数目受到测试期间观察的限制，对于无限条路径的系统，无法覆盖全部路径，不能容纳全部的节点实例。基于状态的方法和基于检查点结构树的方法能够容纳全部节点运行轨迹，能够处理潜在的无限路径（即循环结构），不存在基于路径的方法只能用来分析解决有限条路径系统的问题。

表 1　交互间关联的可信性分析方法比较

方法	交互关联类型	符号	关联规则	交互关联可信性值
文献[7]	因果 并发 重复	$\rightarrow$ $\|\|$ $\leftrightarrow$	*placeOrder* $\rightarrow$ *writeToDB*, *writeToDB* $\rightarrow$ *notifySupplier*, *changeOrder* $\rightarrow$ *writeToDB*, *notifyUserOfShipment* $\rightarrow$ *writeToDB*, *writeToDB* $\rightarrow$ *notifyUser*, *changeOrder* $\rightarrow$ *noChangeAllowed*	1
文献[11]	因果	$<_p$	*placeOrder* $<_p$ *writeToDB* $<_p$ *notifySupplier*, *changeOrder* $<_p$ *writeToDB* $<_p$ *notifySupplier*, *notifyUserOfShipment* $<_p$ *writeToDB* $<_p$ *notifyUser*, *changeOrder* $<_p$ *noChangeAllowed*	1
本文方法	同步因果 异步因果 并发 互斥 重复	$<_S$ $<_A$ $\|\|$ + *	*placeOrder* $<_S$ *writeToDB* $<_S$ *notifySupplier*, *[change order before shipment] (*changeOrder* $<_S$ *writeToDB* $<_S$ *notifySupplier*), *notifyUserOfShipment* $<_S$ *writeToDB* $<_S$ *notifyUser*, *[change order after shipment] (*changeOrder* $<_S$ *noChangeAllowed*)	0

对于构件 Controller 所在节点，由于节点中存在循环结构，基于路径的方法无法计算节点的可信性，本文基于检查点结构树，节点可信性可由式(8)得出，其中，$t_i(i=1,2,\cdots,7)$为检查点 cp_i 的可信性，$T_{S_0}, T_{S_{1,1}}, T_{L_{1,2}}, T_{S_{2,1}}, T_{S_{1,3}}, T_{L_{2,2}}$ 可依据 3.2 节给出的检查点结构公式计算得出

$$T_{\text{Controller}} = T_{S_0}(T_{S_{1,1}}[t_1,t_2], T_{L_{1,2}}[T_{S_{2,1}}(t_3,t_4)], T_{S_{1,3}}[t_5,t_6,T_{L_{2,2}}(t_7)]) \tag{8}$$

2) 处理大状态空间

基于状态的方法在对复杂系统建模时会产生难以处理的大状态空间，使计算和分析非常复杂甚至不可行。本文基于检查点结构树的方法通过将节点划分模块分层，基于检查点间结构计算节点可信性，不存在大状态空间问题。

对于图 8 所示的节点的检查点结构，基于状态的方法中状态数为检查点的数目即 11，而本文方法只有循环结构中使用基于状态的模型，其他结构采用基于路径的模型。如图 10 所示，$L_{1,1}$ 有 3 个孩子，因此基于状态模型计算时状态数为 3，远远小于整个分布式软件基于状态方法计算的状态数。

3) 适用场合

基于路径的方法能够方便地进行节点实例的可信性评价，但由于对无限路径的系统无法覆盖全部路径，使基于各节点实例进行节点可信性的评价不再适用。基于状态的方法可以分析解决潜在的无限条路径的系统，适用于评价节点的可信性，但由于不针对某条运行轨迹，所以不适用于进行节点实例的可信性评价。而本文提出的检查点结构树能够容纳并表征节点内的全部运行轨迹，即所有的节点实例，因此不仅可以用于进行节点实例的可信性评价，还可以基于检查点结构树进行节点的可信性评价。

对 4.1 节提到的分布式软件实例，交互检查点 cp_3 的可信性值为 0，由基于检查点结构树的节点实例可信性评价算法（算法 2）得到 Controller 的节点实例可信性值为 0，准确判为不可信。对于构件 Controller 所在节点的可信性，可以依据式(8)计算得出。

4.3　运行效率分析

为了对检查点进行监控，会在检查点处织入探针，相应地增加了运行时开销，主要包括：1）获

表 2　分布式软件节点可信性评价方法的比较

方法	容纳全部节点实例	处理无限路径	处理大状态空间	适用场合	
				节点实例可信性评价	节点可信性评价
基于路径[15,16]	×	×	v	v	×
基于状态[13,14]	v	v	×	×	v
基于检查点结构树	v	v	v	v	v

取检查点属性信息的开销；2）将检查点属性信息经网络传输到节点分析器的开销。在运行平台一定的情况下，探针所消耗的时间基本是常量，不会改变原有程序的计算复杂度。探针获取监控信息后，传输给节点分析器，这部分处理可以在不同的处理器上完成，并且处理逻辑和软件的运行逻辑没有关联性，所以不会给程序功能逻辑的运行带来额外开销。

以“下订单”功能模块场景1中构件Controller的交互检查点为例，测试织入获取交互检查点属性的探针后各交互检查点带来的时间开销。假设织入获取交互检查点属性的探针前，“下订单”功能模块场景1运行一次的时间为a，织入后带来的时间开销为$ß$，相对未监控情况，监控所带来的额外开销百分比为$\frac{b}{a+b}\times 100\%$。对某一交互，令获取该交互检查点属性的时间开销为$t_{get}$，交互的时间开销为$t_{exe}$。本文将获取交互检查点属性和节点交互并发执行，当$t_{get}\leq t_{exe}$时，获取交互检查点属性引入的额外时间开销为0；当$t_{get}>t_{exe}$时，获取交互检查点属性引入的额外时间开销为$t_{get}-t_{exe}$。实验中共进行了5次测试，结果如表3所示。$ß$=2.83，即各交互额外时间开销均值的和，$a$ = 1108.93，引入交互检查点监控带来的额外开销为0.25%。可见，通过对交互检查点的监控与节点交互并发执行，获取交互检查点属性的时间开销基本淹没在执行交互的运行时间内，对分布式软件的整体运行造成的影响很小。

5 结束语

针对现有的分布式软件交互关联规则无法适应复杂交互场景可信性分析，以及传统的基于软件体系结构来评价分布式软件可信性的方法存在的问题，本文在节点内织入3种检查点，引入了更准确的交互关联规则，通过对检查点属性及节点间交互的监控，来对检查点可信性进行评价；通过将节点分解为多层结构，实现了基于检查点结构树的分布式软件及其实例的可信性分析；并以小型分布式软件为例进行了实验，验证了方法的可行性及有效性。

本文方法与基于检查点的传统软件可信性评价机制相比，增加了交互检查点，引入了适应复杂交互场景的交互关联规则，因此对于节点间协作交互频繁的分布式软件，如电信和金融领域的服务系统、交通控制系统以及电子商务系统等，更能突出本文方法的优势。对于节点间交互不频繁的分布式软件，监控的处理则退化为针对传统软件的处理方法。

下一步的工作是在大规模分布式软件上测试各类检查点的设置粒度，针对具体需求在监控开销和监控粒度上权衡；并测试本文方法是否随分布式软件规模的增大存在一定的局限性。

参考文献：

[1] REYNOLDS P, KILLIAN C, WIENER J L, et al. Pip: detecting the unexpected in distributed systems[C]//The 3rd Symposium on Networked Systems Design and Implementation. USENIX Assoc ion Berkeley, CA, USA, c2006: 115-128.

[2] 杨晓晖, 周学海, 田俊峰, 等. 一个新的软件行为动态可信评测模型[J]. 小型微型计算机系统, 2010,31(11): 2113-2120.
YANG X H, ZHOU X H, TIAN J F, et al. Novel dynamic trusted evaluation model of software behavior [J]. Journal of inese Computer Systems, 2010,31(11): 2113-2120.

[3] 刘玉玲, 杜瑞忠, 冯建磊, 等. 基于软件行为的检查点风险评估信任模型[J]. 西安电子科技大学学报（自然科学版）, 2012,39(1):179-184.
LIU Y L, DU R Z, FENG J L, et al. Trust model of software behaviors based on check point risk assessment [J]. Journal of Xidian University, 2012,39(1):179-184.

表3 交互检查点带来的时间开销(ms)

测试序号	2) *writeToDB*		3) *notifySupplier*		8) *writeToDB*		9) *notifyUser*		场景1
	t_{get}	t_{exe}	t_{get}	t_{exe}	t_{get}	t_{exe}	t_{get}	t_{exe}	
1	10.35	15.74	10.05	8.25	10.23	15.87	8.98	7.65	1 124.34
2	9.98	14.95	9.30	7.83	10.11	15.44	9.24	7.87	1 098.35
3	9.57	14.80	9.91	8.13	9.76	14.32	8.65	7.54	1 087.65
4	10.12	15.56	9.54	7.95	9.99	15.34	8.50	7.39	1 128.72
5	9.76	14.55	9.58	8.01	9.74	14.78	8.78	7.78	1 105.60
平均值	9.96	15.12	9.68	8.03	9.97	15.15	8.83	7.65	1 108.93
额外开销均值	0		1.65		0		1.18		2.83

[4] **田俊峰，张亚姣. 基于马尔可夫的检查点可信评估方法**[J]. **通信学报，**2015,36(1): 230-236.
TIAN J F, ZHANG Y J. Checkpoint trust evaluation method based on Markov [J]. Journal on Communications, 2015,36(1): 230-236.

[5] **刘东红，郭长国，王怀民，等. 监控使能的分布式软件系统构造方法**[J]. **软件学报，**2011,22(11):2610-2624.
LIU D H, GUO C G, WANG H M, et al. Monitoring enabled istributed software construction method[J]. Journal of Software, 2011, 22(11): 2610-2624.

[6] **文志诚，李长云，满君丰. 基于贝叶斯网的分布式软件行为运行时可信性分析**[J]. **小型微型计算机系统，**2012,33(3):504-511
WEN Z C, LI C Y, MAN J F. Analyzing running-time behavioral credibility for distributed software based on Bayesian network [J]. Journal of Chinese Computer Systems, 2012,33(3):504-511.

[7] **彭成，杨路明，满君丰. 网络化软件交互行为动态建模**[J]. **电子学报，**2013,41(2):314-320.
PENG C, YANG L M, MAN J F. Dynamic modeling of networked software interactive behavior[J]. Acta Electronica Sin 2013,41(2): 314-320.

[8] WANG Q, LU Y, XU Z J, et al. Software reliability mode for component interaction mode[J]. Journal of Electronics(Chi 2011, 28(4/5/6):632-642.

[9] YANG N H, YU H Q, QIAN Z L, et al. Modeling and quantitatively predicting software security based on stochastic Petri nets[J]. Mathematical and Computer Modeling, 2012,55(1-2):102-112.

[10] TYAGI K, SHARMA A. An adaptive neuro fuzzy model for estimating the reliability of component-based software systems[J]. Applied Computing and Informatics, 2014,10(s1-2):38-51.

[11] CHEN X H, LIU J, LIU Z M. Requirements monitoring for nternetware: an interaction based approach[J]. Science China: Information Science, 2013,56(8):1-15.

[12] GOKHALE S S. Architecture-based software reliability analysis: overview and limitations[J]. IEEE Transactions on Dependable and Secure Computing, 2007,4(1): 32-40.

[13] CHEUNG R C. A user-oriented software reliability model[J]. IEEE Transactions on Software Engineering, 1980,SE-6(2):118-125.

[14] GOKHALE S S, WONG W E, HORGAN J R, et al. An analytical approach to architecture-based software performance and reliability prediction[J]. Performance Evaluation, 2004,58(4):391-412.

[15] HSU C J, HUANG C Y. An adaptive reliability analysis using path testing for complex component-based software systems[J]. IEEE Transactions on Reliability, 2011,60(1):158-170.

[16] YACOUB S, CUKIC B, AMMAR H H. A scenario-based reliability analysis approach for component-based software [J]. IEEE Transactions Reliability, 2004,53(4):465-480.

[17] **李珍，田俊峰，赵鹏远. 基于分级属性的软件监控点可信行为模型**[J]. **电子与信息学报，**2012,34(6):1445-1451.
LI Z, TIAN J F, ZHAO P Y. A trustworthy behavior model for software monitoring point based on classification attributes[J]. Journal of Electronics & Information Technology, 2012,34(6):1445-1451.

[18] **李珍，田俊峰，杨晓晖. 基于检查点分级属性的软件动态可信评测模型**[J]. **计算机研究与发展，**2013,50(11):2397-2405.
LI Z, TIAN J F, YANG X H. Dynamic trustworthiness eval ion model of software based on checkpoint's classification attributes [J]. Journal of Computer Research and Development, 2013,50(11): 2397-2405.

[19] **邵建平，王玲. "拉长板"木桶原理及其运用研究**[J]. **科技管理研究，**2005,25(4):159-161.
SHAO J P, WANG L. Research on "stretched panels" bucket theory and its application [J]. Science and Technology Manage nt Research, 2005,25(4):159-161.

作者简介：

李珍（1981-），女，河北保定人，河北大学副教授，主要研究方向为软件安全、可信计算。

田俊峰（1965-），男，河北保定人，博士，河北大学教授、博士生导师，主要研究方向为信息安全、分布式计算、网络技术。

常卓（1979-），男，河北保定人，河北大学讲师，主要研究方向为可信计算、网络技术。

马晓雪（1974-），女，河北蔚县人，河北大学副教授，主要研究方向为可信计算、网络技术。