运行时验证中的减少监控开销方法研究

徐 胜 叶俊民 陈 曙 金 聪 陈 盼
(华中师范大学计算机学院 武汉 430079)

摘 要 运行时验证中的一个重要研究内容就是减少监控开销,以达到运行时开销对系统影响最小化的目标。总结了近年来运行时验证中减少监控开销技术的研究发展,首先介绍了运行时开销控制的研究现状;然后详细介绍了运行时开销减少的具体方法;最后分析了运行时开销控制技术面临的主要挑战,并对该领域未来的研究方向进行了展望。

关键词 运行时验证,监控开销,开销控制

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.5.030

On Reducing Monitoring Overhead in Runtime Verification

XU Sheng YE Jun-min CHEN Shu JIN Cong CHEN Pan
(School of Computer Science, Central China Normal University, Wuhan 430079, China)

Abstract A most important issue in the field of runtime verification is reducing monitoring overhead. How to reduce the monitoring overhead to minimize its influence on the system is one of the essential objectives. In this paper, monitoring methods and research status of runtime overhead control were first introduced. Then several computation methods of overhead costs were analyzed to optimize the monitoring mode. Finally, the current challenges and the future research directions of overhead control in runtime verification were further discussed.

Keywords Runtime verification, Monitoring overhead, Overhead control

1 引言

运行时验证是一种轻量级的形式化验证方法,它通过运行时监控器来检查待验证系统的执行是否满足给定的正确性性质。对一个目标系统进行运行时验证,需要将被验证的系统置于监控下[1]。通常,一个对待验证系统进行检测的模块被称为监控器。监控器接收系统的运行踪迹,并根据给定的性质规约,判定系统执行是否满足这些规约。运行时监控开销是一系列监控器在检测抽象轨迹时所产生的,并且随着状态空间的增大而逐渐扩大。在运行时验证中,系统面临的一个最大挑战就是它包含的运行时开销。有时这种开销是巨大的,会严重影响系统性能,特别是时间敏感系统。因此,减少运行时监控开销,以满足时间敏感系统的验证需求,缓解组合空间爆炸问题,是一件非常有价值的研究工作。

传统的运行时验证中[2],通常一个系统是由待验证的系统和一个外部的观察者组成,这个外部观察者即监控器。在运行时验证中,监控器通常从高层性质规约中自动生成。从逻辑和语言的角度来看,运行时验证主要验证了 LTL 性质[3-5],特别是安全性质[6,7]。文献[8-10]中提出了其他语言和框架的持续发展,促进了时态性质规约的应用。但是这些研究并没有考虑到高效的监控器合成,在运行时监控开销方面遇到了极大的挑战。

减少监控开销方法在优化监控器的监控方式方面得到了极大的发展。优化监控器方式能够极大地减少运行时开销。在运行时验证的文献中常见的是事件触发的监控方式[11],系统状态每一次变化都会触发监控器执行。另一种方法是时间触发的监控方式[12,13],监控器以一个固定的时间间隔来对待验证的程序状态进行采样。时间触发的方法涉及到解决一个最优化问题,其目标是最小化辅助内存的大小,使得监控器能够正确地重构程序的状态变化序列,以此控制运行时开销。

减少监控开销方法在离散事件系统的控制理论中得到发展。文献[15]通过暂时减少监控器的参与来控制开销,用户通过设定的阈值来减少开销。文献[16]提出利用状态估计进行采样,利用隐马尔科夫模型估计未来可到达的状态,以决定监控器是否需要采样待验证的程序,但该方法无法保证精确地重构系统状态,因而监控器无法确保在采样周期内获取所有程序状态变化,以至于运行时开销难以控制和预测。

本文的目的是介绍减少运行时监控开销的方法以及未来开销控制的发展趋势。本文第2节是研究基础,介绍运行时验证中监控器的几种常见监控方式;第3节是减少监控开销方法研究;第4节是实验分析,从具体实验的角度来验证本文所提方法的有效性和可行性;最后总结了本文工作,并提出了进一步研究展望。

到稿日期:2015-03-18 返修日期:2015-07-22 本文受国家科技支撑计划项目(2015BAK33B00),教育部人文社会科学研究规划基金(15YJA880095),中央高校基本科研业务费专项资金科研项目(CCNU15GF003)资助。

徐 胜(1990—),男,硕士,主要研究方向为软件工程,E-mail:1025992671@qq.com;叶俊民(1965—),男,博士,教授,主要研究方向为软件可靠性,E-mail:jmye@mail.ccnu.edu.cn;陈 曙(1981—),男,博士,讲师,主要研究方向为软件工程、软件形式化;金 聪(1960—),女,博士,教授,主要研究方向为数字水印、模式识别;陈 盼(1990—),女,硕士生,主要研究方向为监控开销方法。

2 研究基础

在运行时验证中，运行时监控器主要用事件触发监控方法[11]、时间触发监控方法[12,13]和混合式监控方法[14]3种方法来对系统进行监控。下面具体介绍这3种监控方式的监控机制，并详细分析每一种监控方式在开销约减中的优缺点。

2.1 事件触发监控方法

当前运行时验证中，大多数监控器采用的是事件触发监控方法。事件触发监控方法(Event-triggered Monitoring，ETRV)[11]是当一个新的关键事件(如变量的值发生变化)发生时程序触发监控器来验证一组性质规约。事件触发监控的监控机制如图1所示。

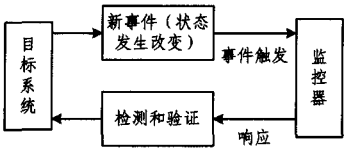

图1 事件触发监控的监控机制

当目标系统中有关键事件发生时，程序触发监控器判定性质规约是否满足，如果违背则修复目标系统。显然，目标程序中关键事件的发生是随机的，这会导致监控行为的随机性和开销边界的不可预测。例如，文献[17]中提出的事件触发监控器，采用了事件触发监控机制，其开销边界随着系统执行的变化而变化，具体如图2所示。可知，随着目标系统的持续执行，其监控开销变得不稳定。

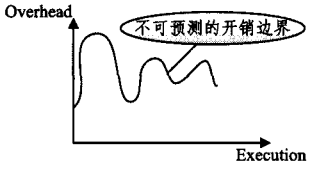

图2 事件触发监控的开销边界

事件触发监控方法的优势是简单方便，但是这种方法的监控行为预测会给系统带来不可预测的开销边界和一系列运行时中断，会导致不希望发生的瞬时过载情况和频繁调用监控器，从而产生大量监控开销。

2.2 时间触发监控方法

时间触发监控方法(Time-triggered monitoring，TTRV)[12,13]是监控器以一个固定的时间间隔对程序状态进行采样，然后判定性质规约是否违背。这个固定的周期通常被称为采样周期(Sampling Period，SP)，它使得监控器不会错过所有的关键事件。时间触发监控的监控机制如图3所示。

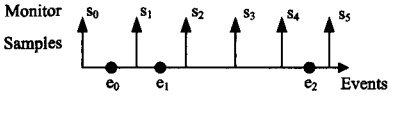

图3 时间触发监控的监控机制

这种监控方式解决了时间敏感系统中开销不可预测性问题，提供了可以预测的监控行为和开销边界，如图4所示。这使得时间触发监控特别适合设计实时嵌入式系统，因为实时可预测性在实时嵌入式系统中扮演着重要的角色。但时间触发的监控方式以固定的周期采样会产生冗余采样，会产生不必要的监控器调用，导致过多额外的监控开销。

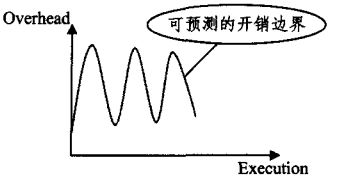

图4 时间触发监控的开销边界

Borzoo Bonakdarpour等人[12]提出的时间触发监控机制如图3所示。在程序执行时间线上，e_0、e_1、e_2 为执行路径上的关键事件，s_0，s_1，…，s_5 为监控器采样点。显然，当 s_3 和 s_4 两个监控器采样时无任何事件发生，此为冗余采样。这些冗余采样会产生过度的运行时开销。

为了减少冗余开销，在TTRV的基础上，文献[28]提出了路径敏感的时间触发监控技术(pa-TTRV)和适应性路径敏感的时间触发监控技术(adaptive pa-TTRV)，通过实时调整采样周期来减少冗余采样的数量，比单一的时间触发监控方法(TTRV)实现了更高效的监控，减少了监控开销。

2.3 混合式监控方法

混合式监控方法(Hybrid monitoring)[14]是结合事件触发监控和时间触发监控的优势来减少运行时开销的静态分析技术。监控器依赖当前所执行的路径从一个模式切换到另一个模式，以实现运行时开销对系统影响最小化的目标。

混合式监控优点在于：当关键事件比较稀疏时监控器以ET模式来监控，当关键事件比较密集时监控器则以TT模式来监控。这种混合式的监控方案结合了Event-triggered监控和Time-triggered监控的优势，避免了冗余采样，从而减少了运行时开销，具有强大的监控能力。

2.4 监控方式的比较

针对上述的3种不同的监控方式，文献[14]中分析了混合式监控与时间触发监控、事件触发监控的优缺点，并进行了详细对比。图5给出了3种不同监控方式的对比。

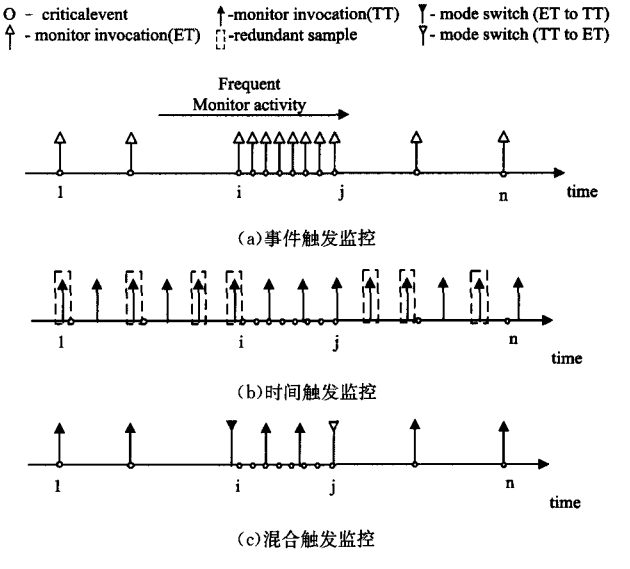

图5 不同监控方式的对比

图5(a)为事件触发监控，其中从事件 i 到事件 j 在执行的轨迹中有一系列突发事件。发生在事件 i 到事件 j 上的频繁的监控调用导致了一系列突发的系统监控，这会引起很高的执行开销和不可预测的程序行为。图5(b)为时间触发监控，图中虚线椭圆形所标记的是“冗余”采样(redundant sam-

ples)。从图中可以看出,监控器有很多"冗余"采样;一个"冗余"采样,就是一次监控器调用,并且缓冲区中没有处理任何关键事件。尽管时间触发的"冗余"采样会导致额外的监控开销,但我们观察到这种方式能够有效地减少运行时的累积开销。图 5(c)为混合监控,通过减少监控活动来减少整体监控开销。

3 运行时监控中减少监控开销的方法

减少运行时开销的关键在于分析监控开销的来源、监控器具体监控过程,以及解决监控开销的技术方案。下面从这几个方面来具体分析产生运行时开销的一系列过程。运行时开销产生于监控器监控待验证目标系统过程中。待验证的目标系统可以看成一条程序执行轨迹,通常将程序执行轨迹抽象为一条抽象轨迹,用从高层规约中合成的一系列监控器来验证这条抽象轨迹,具体监控模型如图 6 所示。

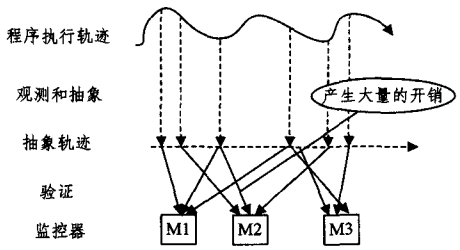

图 6 运行时验证的监控模型

在运行时验证中,运行时监控开销取决于许多因素,包括需观察的程序位置个数、不同的数据值需要监控器副本的个数,以及更新监控器状态和验证性质规约的开销等。特别是对复杂性质(如实时系统性质)进行运行时验证会产生很大的开销,从而影响系统的性能。下面从监控器的监控过程方面来介绍如何减少监控开销。

典型的运行时验证系统包括两部分:待监控的系统和监控器,如图 7 所示。待监控的系统即是目标程序,监控器分为两个组成部分:事件接收器和属性接收器。监控器首先接收系统产生的事件踪迹,然后监控事件,包括监控变量、函数、关键事件等,再通过属性接收器对踪迹是否满足规约进行判定,如果违约则修正程序。

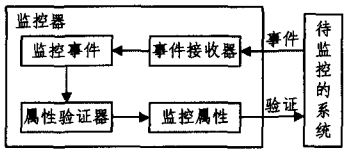

图 7 典型运行时监控机制

由图 7 可知,可通过运行时监控机制来分析运行时监控开销。因此,减少运行时监控过程的开销可具体细分为两部分:减少监控事件的开销和减少监控属性的开销。下面具体分析这两部分的开销减少方法。

3.1 减少监控事件开销

监控事件包括获取程序中关键事件、对关键事件分析以及监控器本身的插装机制。减少监控事件开销可从优化关键事件获取方式、事件分析方式以及插装机制来展开。

(1)采用混合式监控方式获取事件

由 2.3 节可知,监控器采用混合式监控方法获取关键事件,结合了事件触发监控和时间触发监控的优势特点。因此,基于混合式验证的监控开销包含 3 个部分:处理关键事件所发生的开销、监控切换的开销以及采样的开销。

文献[14]提出了一种混合式运行时验证开销模型,并用整数线性规划求解该模型,以此实现一种最优化的监控方式,获取程序的关键事件。具体分析如下。

设 $G=\langle V,v^0,A,\omega,v^f\rangle$ 是程序 P 的控制流图,$V_c\subseteq V$ 是计算最大采样周期 LSP 后的关键顶点集合,C_{event}、C_{switch}、C_{sample} 分别为处理关键事件所发生的开销、监控切换的开销、采样的开销。对于 G 的一条路径 $\pi=v^0\rightarrow v_1\rightarrow\cdots\rightarrow v^f$,$\Delta_\pi$ 是路径 π 的最长子路径集,则文献[14]的监控方案的开销定义如式(1)所示。

$$Q_0(\pi)=\sum_{v\in V_c}[c_{ET}\cdot(1-M(v))+c_{hist}\cdot M(v)]+\sum_{(v_1,v_2)\in A}[c_{E\rightarrow T}\cdot(1-M(v_1))\cdot M(v_2)+c_{T\rightarrow E}\cdot M(v_1)\cdot(1-M(v_2))]+\sum_{\delta=[v_i\rightarrow\cdots\rightarrow v_j],\delta\in\Delta_\pi}[c_{TT}\cdot(|\frac{\sum_{k=i}^{k=j}\omega(v_k)}{LSP}|)] \tag{1}$$

其中,3 个和式分别对应 C_{event}、C_{switch}、C_{sample}。其中,第一个和式为处理关键事件所发生的总开销,是 c_{ET} 与 c_{hist} 的加权平均值。当控制流图的顶点采用 ET 监控时,$M(v)$为 0,表明此时只产生处理事件开销,而不会产生缓冲区的开销成本;当控制流图的顶点采用 TT 监控时,$M(v)$为 1,表明此时只产生将关键事件保存到缓冲区中的开销,而不会产生处理关键事件的开销。同理,第二个和式为在 ET 到 TT 模式之间切换所发生的总开销,第三个和式为 TT 模式中采样所发生的总开销。

该问题模型是一个 NP 问题,文献[14]中利用数学线性规划对该问题进行求解,从而实现了运行时开销的最小化,并且实验分析证明了其有效性和可行性。

(2)结合程序静、动态分析技术设置监控插装点

程序静态分析是先获取所有的程序事件,然后监控器再对这些事件进行检测。动态分析在监控的同时进行检测。静态分析和动态分析相结合的概念源于编程语言中的 typestate 分析环境中,其结合了两者的优势。静态 typestate 分析往往会导致不确定的结果,产生一些有价值的信息,从而可以简化运行时的状态模型。研究者又提出了残留 typestate 分析[19]。其后,Bodden 等人提出通过快速检查和基于指针分析来发现和删除不必要的监控插装点[23],此技术已用于 tracematches 工具。伊利诺伊大学的 Chen Feng 和 Rosu G 提出一种 AspectJ 语言的扩展[21],其依赖通知且使用上述方法的本质,通过三阶段的分析确定不相关的程序点,进而禁用这些程序点,减少运行时监控的程序点,从而减少运行时监控开销。

本节的关键在于:监控器在分析程序的抽象轨迹时,可根据关键事件的特点,实时地删除或修改程序中与监控相关的插装点,从而减少运行时开销。

(3)优化监控器插装

优化插装是从减少不必要的插装点这个方面来减少监控开销的,是开销控制的重要方式。近年来很多学者都是用优化插装方式来进行运行时开销控制。文献[22]中提出的 Java-MOP 是一个可用于验证实时性质的运行时验证框架,它采用面向方面编程技术实现自动插装,监控器嵌入在目标系统中,从而实现高效的监控器插装,但该方法不能将监控器从目标程序中独立出来。此外,还可以使用一些插装包,例如文献[12]中的 Java-MaC 就是采用 JTrek 验证工具,根据 PEDL

和 MEDL 规约对目标系统进行高效的插装。

3.2 减少验证属性开销

在监控器监控属性过程中，减少验证属性包括降低验证算法的复杂度、减少需要监控的属性值等。

(1)降低验证算法的复杂度

监控器是通过验证工具(如 Java-MOP、MaC)来实现自动验证的，而验证的效率取决于验证算法的复杂度。降低验证算法的复杂度能够有效减少运行时监控中的验证开销。文献[30]提出一种使用有穷自动结构表示无穷结构的监控算法，从而使得算法的空间复杂度降为了多项式级，使得监控器验证开销变得可控制，具体监控算法如图 8 所示。

```
i←0
q←0
Q←{(α,0,waitfor(α))|α temporal subformula of Ø}
loop
   Carry over constants and relations of D_i to D_i
   For all(α,j,Ø)∈Q do
   Build relations for α in D_j
   Discard auxiliary relations for α in D_{j-1} if j−1 >= 0
   Discard relations p_δ^DØ, where δ is a temporal subformula of α
while all relations p_δ^DØ are built for α∈tsub(Ø) do
   Output valuations violating Ø at time point q
Discard structure D_{q-1} if q−1>= 0
q←q+1
Q←{(α,i+1,waitfor(α)) | α temporal subformula of Ø}∪
   {(α,j, ⋃_{θ∈update(S,τ_{i+1}−τ_i)} waitfor(θ) )|(α,j,S)∈Q and S≠Ø}
i←i+1
end loop
```

图 8 有穷结构替换无穷结构的监控算法

除此之外，文献[29]中 Drusinsky 等人只使用了 MTL 限制性子集，将 MTL 公式进行化简，从而大幅度地减少了监控的复杂度，减少了监控开销，如图 9 所示。

```
算法功能:化简 MTL 公式
输入:MTL 公式的前缀表达式形式 prefixFomula
输出:化简后 MTL 公式的前缀表达式形式
Begin
    while(prefixFormula 中包含字符"G")
    {
    if("G"后面的一个字符为"[")
    {//表明该时态算子带有时钟约束~d
        得到该时态算子的完整形式 G[~d];
        将 G[~d]替换成 R[~d]f;
    }else 将 G 替换成 Rf;
    }
    while(prefixFormula 中包含字符"F")
    {
    if("F"后面的一个字符为"[")
    {//表明该时态算子带有时钟约束~d
        得到该时态算子的完整形式 F[~d];
        将 F[~d]替换成 U[~d]t;
    } else 将 F 替换成 Ut;
    }
    将 prefixFormula 中所有的"→"替换成"‖!";
End
```

图 9 MTL 公式约减算法

上述公式经过化简后，复杂度大大降低，有效地控制了验证算法开销。在此基础上，通过静态分析和动态分析相结合的方法来进一步减少运行时验证过程中的开销，如将演绎验证工具 KEY 和运行时验证工具结合[31]，分离出不需在运行时进行验证的部分公式，从而简化运行时验证过程，从而降低验证算法的复杂度以减少验证开销。

(2)减少需要检测的属性

减少需要检测的属性的目的在于有效地减少运行时验证的开销。对一个产品线属性进行运行时监控时，可使用静态分析方法对组合之后的产品特性进行检查，将不会发生冲突的属性删掉，不对其进行监控，从而达到减少开销的目的[31]。

例如，假设[Inside]、[File]、[Local]之间的约束为 Inside、File 和 Local，均包含了 Base，因此在检测属性时，可以只验证 Base。实验证明，采用这种思路监控 read 函数和 print 函数时运行时开销分别下降了 44%和 22%。另外，Bodden E 等提出的 Clara 工具通过减少需要检测的事件来减少运行时开销[32]，即不需要检测的事件则不进行验证，可以对使用 Tracecuts、JavaMOP、PQL、QVM、J-LO 等工具生成的 AspectJ 监控器进行优化，从而实现运行时监控开销的控制。

4 实验分析

4.1 实验描述

本实验通过 HyRV 的插装工具链来实现实验的数据分析过程，该工具链输入是 C 语言程序和成本数据配置，输出是插装后的优化方案。HyRV 的插装工具栏如图 10 所示，其输入为程序源代码，输出为带监控模式的插装优化方案。

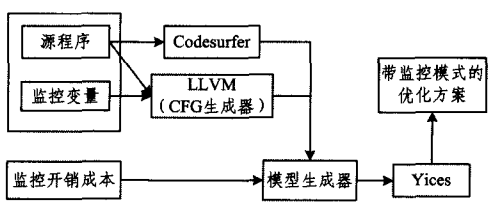

图 10 HyRV 插装工具链的基本框架

针对 HyRV 插装工具链基本框架，源程序和监控变量是该工具链的输入，LLVM 是 CFG 生成器，能将源程序代码转化为对应的控制流图。Codesurfer 是 C 或者 C++程序代码的理解工具，能够定位源程序中关键变量的位置。源程序首先通过 LLVM 转化为程序的 CFG 图，然后通过 Codesurfer 代码理解器定位到程序中的关键变量的位置，并将这些关键变量设置为监控变量，嵌入在程序的 CFG 图中。带有监控变量的 CFG 图和监控开销成本则通过模型生成器导入到工具 Yices 中，通过将 CFG 图的优化问题转化为整数线性规划模型，并用整数线性规划解析器(ILP Solver)来求解该模型，最终在 CFG 图中生成带有插装监控模式的优化方案。

生成插装方案包含单一的事件触发监控模式(ET-only)、单一的时间触发监控模式(TT-only)、混合式触发监控模式(HyRV)。

其中，根据采样周期的不同配置，单一的时间触发监控模式(TT-only)包含两种类型，分别为 TT-only(SP=10LSP) 和 TT-only(SP=20LSP)。根据采样周期和切换开销的不同配置，混合式触发监控模式(HyRV)包含 4 种类型，分别为 HyRV(SP = 10LSP, switch = 100)，HyRV(SP = 10LSP,

switch＝150)，HyRV(SP＝20LSP，switch＝100)，HyRV(SP=20LSP,switch=150)。

在本实验中，关于监控器在监控过程采用的监控方式，给出了如表1所列的统一说明。

表1　监控器采用的监控机制说明

监控方式	说明(单位：时钟周期)
ET-only	采用单一事件触发监控
TT-only(SP=10LSP)	采用单一时间触发监控，且采样周期为10×LSP；
HyRV(SP=10LSP,switch=100)	采用混合式监控，且采样周期为10×LSP，切换开销为100
HyRV(SP=10LSP,switch=150)	采用混合式监控，且采样周期为10×LSP，切换开销为150
TT-only(SP=20LSP)	采用单一时间触发监控，且采样周期为20×LSP；
HyRV(SP=20LSP,switch=100)	采用混合式监控，且采样周期为20×LSP，切换开销为100
HyRV(SP=20LSP,switch=150)	采用混合式监控，且采样周期为20×LSP，切换开销为150

4.2　实验结果

本文实验中所用到的基准集数据源于SNU-RT(首尔大学实时研究组)基准集的数据[25]。本实验分别用单一事件触发监控、单一时间触发监控、混合式监控3种方式进行监控，计算每一种监控方式的总开销，收集数据并进行对比分析，从而分析混合式监控与单独模式监控的开销优化对比。

我们从基准集中选取3组程序：Blowfish、Dijkstra和Quadratic。这3组程序均为C语言编写的程序，其程序中关键事件、关键变量分布特点如表2所列。

表2　3个基准集程序的CFG特点

程序	代码量(行)	CFG中关键顶点(个)	CFG中的弧(条)	监控变量数(个)
Blowfish	745	169	213	20
Dijkstra	171	39	78	11
Quadratic	476	105	156	19

由表2可知，在程序Blowfish中，关键顶点在代码量中占比为22.6%；在程序Dijkstra中，关键顶点在代码量中占比为22.8%；在程序Quadratic中，关键顶点在代码量中占比为20.8%。三者的关键顶点在CFG图中所占的比例差不多，表明两组基准集具有相同特点。

针对同一特点的基准集，选定基准集后，再对6个基本开销成本 c_{ET}、c_{TT}、c_{hist}、$c_{E\to T}$、$c_{T\to E}$ 进行开销配置，其基本成本开销配置如表3所列。

表3　基本成本开销配置(单位：时钟周期)

基本开销配置	c_{hist}	c_{ET}	c_{TT}	$c_{E\to T}$	$c_{T\to E}$
1	50	100	100	100	100
2	50	150	150	150	150
3	50	200	200	100	100
4	50	250	250	150	150
5	50	300	300	100	100
6	50	350	350	150	150
7	50	400	400	100	100

确定了基准集和基本开销配置后，再通过开销计算公式进行计算，得出总开销 $M_0(G)$。对Blowfish、Dijkstra和Quadratic 3组程序分别用单一事件触发监控方式、单一时间触发监控方式、混合式监控方式3种不同方法监控其总开销，得出不同监控模式下的总开销数据量。

通过文献[14]对应的开销公式进行计算后，分析实验数据结果，绘成折线对比分析图，Blowfish、Dijkstra的监控总开销结果分别如图11、图12所示。

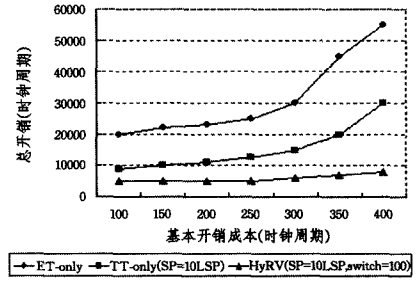

图11　对Blowfish的监控开销对比

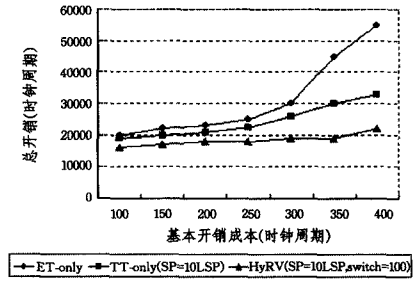

图12　对Dijkstra的监控开销对比

图11和图12表明，对Blowfish和Dijkstra中的关键事件进行监控，事件触发监控方式所产生的监控开销明显高于时间触发监控方式产生的监控开销，相对于混合式监控，两种监控方式都产生巨大的监控开销。当监控器采用混合式触发监控方式时，运行时开销有明显降低。

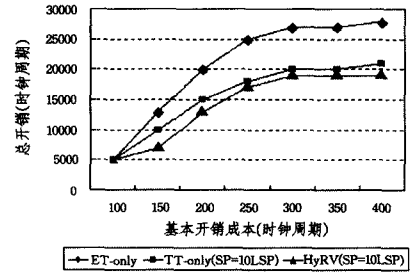

图13　对Quadratic的监控开销对比

对Quadratic的监控总开销结果如图13所示，由图可知，对Quadratic程序中的关键事件进行监控，事件触发监控方式所产生的监控开销明显高于时间触发监控方式产生的监控开销。而混合式触发监控方式与时间触发监控方式产生的开销相差较小。综合考虑，选择混合式监控是最优的。

5　主要挑战及发展趋势

目前，运行时验证技术正处于快速发展的时期，因此运行时开销控制方法也在不断发展。如何在不同的验证工具(如JavaMOP、J-LO等验证工具)中进行开销控制，如何在不同的领域(如Java程序的运行时验证、硬件运行时监控、恶意攻击及病毒检测、网络协议的验证、列车运行控制系统的控制等)进行开销控制，都是研究的热点和难点。

运行时验证技术的发展趋势是在线的、自适应的、可自我学习和自我修复的系统。Bartocci E等人对此进行了初步研究，提出自适应地、动态地对运行时开销进行控制的系统。如何实现这一类运行时验证系统的开销控制是一大挑战。

在线监控系统的开销控制是一个重要挑战。将所有事件都记录下来会产生巨大的系统开销，而使用采样的方法又面临数据不完整的问题，如何根据待验证的属性在这两个方向上进行折中是控制监控开销的关键。

我们认为，运行时开销控制的未来研究方向中有以下问题值得进行深入研究：

(1)研究复杂系统开销控制最优化监控方式。

混合式监控方法是事件触发监控和时间触发监控的结合，弥补了两者在监控开销过程中的诸多缺陷。然而，混合式监控方法只适合用在小型的时间敏感系统中，对于复杂的时间敏感系统(如实时嵌入式系统等)，混合式监控方法并不是最优化的监控方式，其产生的开销也是巨大的。因此研究最优化监控方式具有重大的意义。

(2)研究参数化监控过程中的开销控制方法。

参数化运行时监控是针对程序中的动态性质，监控和分析系统执行，检查其是否满足指定的参数化性质的技术。随着软件技术的发展，通过运行时验证技术来验证系统的动态属性以保证软件可靠性已经越来越普遍，因此研究参数化监控开销控制具有重大的价值。

(3)研究如何采用符号执行结合动态分析技术来调整运行时采样周期。

符号执行技术能够实现更精确和更具体的预测功能。将符号执行技术和混合式监控方式结合，根据执行路径预测来控制关键事件采集的频率，调整采样周期，保证在尽可能小的采样频率下获得足够的所需要的事件，以达到控制运行时开销的目的。

结束语 运行时验证技术已成为一种重要的验证技术，已经被应用到多个领域。减少运行时监控开销是运行时验证技术中至关重要的内容。本文对运行时验证技术中减少监控开销的基本方法进行了阐述，对比分析了目前该领域常见的解决方案，并且对该领域的研究热点进行了深入的阐述，对该领域的研究方向进行了展望。运行时监控开销控制在安全攸关系统和时间敏感系统中都非常重要，随着运行时验证技术的发展，开销控制也必将在医疗器材、航空航天、国防服务等领域发挥其重大的价值。

参 考 文 献

[1] Zhang Shuo, He Fei. Research Advances in runtime verification [J]. Computer Science, 2014, 41(11A): 359-363(in Chinese)
张硕，贺飞. 运行时验证技术的研究进展[J]. 计算机科学，2014，41(11A)：359-363

[2] Pnueli A, Zaks A. PSL Model Checking and Run-Time Verification via Testers[C]// 14th International Symposium on Formal Methods(FM). 2006: 573-586

[3] Giannakopoulou D, Havelund K. Automata-Based Verification of Temporal Properties on Running Programs[C]// IEEE International Conference on Automated Software Engineering(ASE). 2002: 412-416

[4] Chang E Y, Manna Z, Pnueli A. Characterization of Temporal Property Classes[C]// International Colloquium on Automata, Languages and Programming(ICALP). 1992: 474-486

[5] Stolz V, Bodden E. Temporal Assertions using AspectJ[J]. Electronic Notes in Theoretical Computer Science, 2015, 144(4): 109-124

[6] Havelund K, Rosu G. Efficient Monitoring of Safety Properties [J]. Software Tools and Technology Transfer(STTT), 2004, 6(2): 158-173

[7] Falcone Y, Fernandez J C, Mounier L. Runtime Verification of Safety-Progress Properties[M]// Runtime Verification(RV). 2009: 40-59

[8] Kim M, Lee I, Sammapun U, et al. Monitoring, Checking, and Steering of Real-Time Systems[J]. Electronic. Notes in Theoretical Computer Science, 2002, 70(4): 95-111

[9] Kim M, Viswanathan M, Kannan S. Java-MaC: A Run-Time Assurance Approach for Java Programs[J]. Formal Methods in System Design(FMSD), 2004, 24(2): 129-155

[10] d'Amorim M, Rosu G. Efficient Monitoring of omega-Languages[M]// Computer Aided Verification(CAV). 2005: 364-378

[11] Kupferman O, Vardi M Y. Model Checking of Safety Properties [J]. Formal Methods in Systems Design, 2001, 19(3): 291-314

[12] Bonakdarpour B, Navabpour S, Fischmeister S. Sampling-based runtime verification[M]// FM 2011: Formal Methods. 2011: 88-102

[13] Bonakdarpour B, Navabpour S, Fischmeister S. Time-triggered runtime verification[J]. Formal Methods in Systems Design (FMSD), 2013, 43(1): 29-60

[14] Wu C W W, Kumar D, Bonakdarpour B. Reducing Monitoring Overhead by Integrating Event and time triggered techniques [M]// Runtime Verification(RV). 2013: 40-59

[15] Huang X, Seyster J, Callanan S, et al. Software monitoring with controllable overhead[J]. International Journal on Software Tools for Technology Transfer(STTT), 2012, 14(3): 327-347

[16] Stoller S, Bartocci E, Seyster J, et al. Runtime verification with state estimation[C]// International Conference on Run-time Verification(RV). 2011: 193-207

[17] Havelund K, Rosu G. Synthesizing Monitors for Safety Properties[M]// Tools and Algorithms for the Construction and Analysis of Systems(TACAS). 2002: 342-356

[18] Lattner C, Adve V. LLVM: A compilation framework for lifelong program analysis and transformation[C]// International Symposium on Code Generation and Optimization: Feedback Directed and Runtime Optimization. 2004: 75-86

[19] Dwyer M B, Purandare R. Residual dynamic typestate analysis exploiting Static analysis: results to reformulate and reduce the cost of dynamic analysis[C]// ACM ICASE. New York, USA, 2007: 124-133

[20] Manna Z, Pnueli A. A Hierarchy of Temporal Properties[C]// Proceedings of ACM Symposium on Principles of Distributed Computing. 1990: 377-410

[21] Bodden E, Feng Chen, Rosu G. Dependent Advice: A General Approach to Optimizing History-based Aspects[C]// ACM ICAOSD. Virginia, USA, 2009: 3-14

[22] Chen F, Rosu G. Java-MOP: A Monitoring Oriented Programming Environment for Java[C]// Proceedings of Conferences on Theory and Practice of Software, Volume 3440 of LNCS. Edinburgh, UK, 2005: 546-550

[23] Bodden E, Hendren L, Lhotak O. A staged static program analysis to improve the performance of runtime monitoring[C]// Proceedings of the 21st European Conference on Object-oriented Programming. 2007: 525-549

[24] Zhao Chang-zhi, Dong Wei, Sui Ping, et al. Construction Tech-

niques of Anticipatory Monitors for Parameterized LTL[J]. Journal of Software,2010,21(2):318-333

[25] SNU Real-Time Benchmarks[OL]. http://www.cprover.org/goto-cc/examples/snu

[26] Zee K, Kuncak V, Taylor M, et al. Runtime checking for program verification[C]//Proceedings of the 7th International Conference on Runtime Verification(RV'07). Berlin, Heidelberg: Springer-Verlag,2007:202-213

[27] Zhou W, Sokolsky O, Loo B T, et al. DMaC: Distributed Monitoring and Checking[M]//Runtime Verification(RV). 2009:184-201

[28] Navabpour S, Bonakdarpour B, Fischmeister S. Path-aware time-triggered Runtime Verification [M] // Runtime Verification (RV2012). 2012:184-201

[29] Drusinsky D. On-line monitoring of metric temporal logic with time-series constraints using alternating finite automata[J]. JUCS,2006,12(5):482-498

[30] Basin D, Klaedtke F, Miiller S, et al. Runtime monitoring of metric first-order temporal properties[J]. Best Practive & Research Clinical Obstetrics & Gynaecology,2010,22(5):899-916

[31] Bodden E, Lamp. Clara: Partially Evaluating Runtime Monitors at CompileTime[M] // Runtime Verification. Springer Berlin Heidelberg,2010:74-88

[32] Kim C H P, Bodden E, Batory D, et al. Reducing configurations to monitor in a software productline[M] // Runtime Verification. Springer Berlin Heidelberg,2010:285-299

(上接第 149 页)

实例中,用于处理失效的操作都没有给出,它们不是本文讨论的重点。针对不同用途的分布式系统,系统设计开发者可以在此基础上设计默认的相关处理机制,帮助实现系统容错功能的半自动化甚至全自动化。注意,实例中的虚拟差错变量监控声明都是用于监控某个虚拟差错变量为真的情况,而实际上,还可以监控虚拟差错变量为假的情况。在后一种情况下,"某个虚拟差错变量为假"变成了相关指定操作的一个前置条件。这种机制能够防止该操作在某种失效发生后被执行,除非该失效被修复。

结束语 本文针对传统分布式容错系统设计在开发大规模容错系统或者容错能力较强的分布式系统时存在的不足,设计了一种抽象的容错描述语言 FTDL。该语言在解释器层面处理故障、差错或者失效的定义、检测和修复。这种解决方案试图在更抽象的层次以比面向过程程序设计语言和面向对象程序设计语言更加简练的方式描述一个分布式计算的规范。从发展的眼光看,这种程序设计思想可望有效提高分布式容错系统设计的效率。

限于篇幅,本文简要阐述了新的构想、科学意义和应用前景,给出了一些语言成分及其语义描述,并用这种语言针对典型实例给出了分布式容错计算描述的规范,通报了这项研究的最新进展。有关这种语言的详细设计、语法与语义定义、多种应用,作者将另文发表。

参 考 文 献

[1] Laprie J C. Dependable computing and fault tolerance: Concepts and terminology [C]// Proceedings of the 15th International Symposium on Fault-Tolerant Computing. 1985:2-11

[2] Elena D. Fault-Tolerant Design [M]. New York: Springer, 2013:15-16

[3] Herlihy Maurice P, Jeannette M W. Specifying Graceful Degradation [J]. IEEE Transactions on Parallel and Distributed System,1991,2(1):93-104

[4] Gärtner C. FELIX, Fundamentals of Fault-Tolerant Distributed Computing in Asynchronous Environments [J]. ACM Computing Surveys,1999,31(1):1-26

[5] Avižienis A. Design of Fault-Tolerant Computers [C]// Proceedings of the AFIPS'67 Fall Joint Computer Conference, 1967. Washington: Thompson Books,1967:733-743

[6] Alpern B, Schneider F B. Defining liveness [J]. Information Processing Letters,1985,21(4):181-185

[7] Ajay D K, Mukesh S. Distributed Computing(Principles, Algorithms, and Systems) [M]. New York: Cambridge University Press,2008

[8] Needham R M, Herbert A J. The Cambridge Distributed System [M]. Addison Wesley International Computer Science Series, 1982

[9] Hoare C A R. Communicating Sequential Processes [M]. Prentice Hall International,2004

[10] Peter W. The Vienna Definition Language [J]. ACM Computing Surveys,1972,4(1):5-63

[11] Hoare C A R. Communicating sequential processes [J]. Communications of the ACM,1978,21(8):666-677

[12] Lamport L, Shostak R, Pease M. The Byzantine generals problems [J]. ACM Transaction on Programming Languages and Systems,1982,4(3):382-401

[13] Pierre A. The practical importance of formal semantics [M]. Liber Amicorum,1989:31-40

[14] Larsen P G, Lausdahl K, Battle N. The VDM-10 Lauguage Manual: TR-2010-06 [R]. the Overture Open Source Initiative, April 2010

[15] Lausdahl K, Coleman J W, Larsen P G. Semantics of the VDM Real-time Dialect: ECE-TR-13 [R]. Adarhus University, April 2013

[16] Birrell A D, Nelson B J. Implementing Remote Procedure Calls [J]. ACM Transaction on Computer System,1984,2(1):39-59

[17] Wang Yan-yan, Liu Wei, Wang Zhi-ming. Networked fault tolerant control for uncertain singular systems with a packet dropout[J]. Journal of Chongqing University of Posts and Telecommunications(Natural Science Edition),2013,25(3):379-383(in Chinese)
王岩岩,刘伟,汪志鸣. 数据包丢失的不确定奇异系统网络化容错控制[J]. 重庆邮电大学学报(自然科学版),2013,25(3):379-383