ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.3, March 2010, pp.461−472
doi: 10.3724/SP.J.1001.2010.03521

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

多线程程序时序分析的隐 Markov 模型*

孔德光+, 谭小彬, 奚宏生, 帅建梅, 宫 涛

(中国科学技术大学 自动化系,安徽 合肥 230027)

Hidden Markov Model for Multi-Thread Programs Time Sequence Analysis

KONG De-Guang+, TAN Xiao-Bin, XI Hong-Sheng, SHUAI Jian-Mei, GONG Tao

(Department of Automation, University of Science and Technology of China, Hefei 230027, China)

+ Corresponding author: E-mail: kdg@mail.ustc.edu.cn

Kong DG, Tan XB, Xi HS, Shuai JM, Gong T. Hidden Markov model for multi-thread programs time sequence analysis. *Journal of Software*, 2010,21(3):461−472. http://www.jos.org.cn/1000-9825/3521.htm

Abstract: To overcome the difficulty of analyzing and detecting the data race in multithread programs, a method based on Hidden Markov model is presented for the analysis of time sequences in multithread programs. The random variable uncertainty is used to depict the mutual influence in different multithread in time sequences, and the probability distribution for random variable uncertainty is analyzed as the outcome of multithread programs on the condition of data race. Hidden Markov model is constructed to appreciate the state for the thread running according to the observed values of the running threads. The Baum-Welch and forwarding algorithm are used to simulate the real running process of the programs in the influence of context. It is proved by experiments that HMM model can quickly and effectively reflect the time sequence of the multithread programs, which can be used to instruct the detecting process of multithread programs.

Key words: multi-thread; data race; hidden Markov model; sequence

摘 要: 针对多线程程序数据竞争分析与检测困难的问题,提出一种基于隐 Markov 模型的多线程程序时序分析方法.用随机变量不确定性刻画不同线程之间时序上的交互关系,分析数据竞争条件下程序不确定结果的概率分布情况;建立多线程程序时序分析的隐 Markov 模型,使用 Baum-Welch 和前向算法仿真上下文对程序实际运行状态的影响.实验结果表明,该模型能够快速有效反映多线程执行时序,用于指导多线程程序时序竞争检测过程.

关键词: 多线程;数据竞争;hidden Markov 模型;时序

中图法分类号: TP393 **文献标识码**: A

现代复杂信息系统中,多线程程序显示出越来越广泛的应用前景.在提高系统性能和效率的同时,提高多线程程序可靠性成为一个重要问题.但是由于多线程程序本身编程复杂性和执行结果随机性与难以再现的特点,使得多线程程序非常容易出现数据竞争及死锁问题,并且检测困难[1,2].通常,调试器和运行时检测工具由于不能确定新线程开始时间而导致检测失败.多线程中数据竞争就是由于访问共享资源时缺乏或者不适当地运用

* Supported by the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z449 (国家高技术研究发展计划(863))

Received 2008-01-26; Accepted: 2008-10-30

同步机制引起.当使用较细粒度加锁机制时,保护不够便会导致数据竞争.如果对于某一时刻某一线程访问某个资源(例如变量)时间没有限定,就会出现数据竞争,赢得竞争的线程获得访问许可,运行结果不可预测.

目前,多线程程序分析技术是单线程程序分析技术的扩展和深化,有静态分析、动态分析、模型检测 3 种主要手段.其中,静态分析技术[3,4]通过对程序执行状态的抽象和静态模拟,计算程序点具有的程序属性,并在此基础上模拟并发线程间交互的影响.依据 Rice 定理,静态分析具有不可判定性,而模拟多线程程序的执行算法具有指数的时间复杂度,并且对于变量和别名信息的判断算法通常是不完备的近似算法,从而存在很多误报和漏报.Sitraka 公司单元测试工具 Jprobe Suite[5]中的 Threadalyzer 工具就是通过寻找程序中的死锁和空值来检验程序的线程安全性,存在着较多误报问题.动态分析[6−8]技术常常借助于插桩(instrumentation)代码,通过对插桩程序产生的事件流分析进行运行中(on-the-fly)和运行后(post-mortem)分析,得到依赖于特定的输入和执行路径的分析结果,系统开销较大,也同样不能保证检测效果.多线程分析中不可避免地要用到数据流分析技术,数据流分析本身可以看作是基于抽象解释框架的模型检测.模型检测[9,10]本身可以利用其较强的语义描述能力,容易描述一些程序中隐含的属性,通过状态空间搜索,发现静态分析难以发现的一些错误.但是,由于线程与运行环境抽象的困难,并发程序的模型检测使用起来比较困难,并且还存在不少问题.

要从源头上解决数据竞争问题,可以改良同步原语的实现,减少同步代价,设计更好的语言和类型系统来保证程序中不会出现竞争.而对于现实世界中大量多线程程序,这些方案无济于事.应用静态分析、动态分析、模型检测等技术也不能保证多线程程序分析的可靠性和完备性.对于多线程程序,时序分析是检测数据竞争、冗余同步或者死锁的关键性工作,正是时序的不确定性,才导致出现各种各样的不可预料有害的结果.现有的研究集中在语法、语义的层面上[6,7,11,12],使用控制流分析、数据流分析,尤其是指针和别名分析等技术手段,结合抽象解释框架,采取上下文敏感或者流敏感等策略,多线程程序分析和检测成为一个热点问题.针对多线程程序数据竞争分析与检测困难的问题,本文从多线程程序运行时的环境状态出发,提出一种基于随机分析的多线程程序数据竞争分析方法,用不确定性变量刻画不同线程之间在时序上的交互关系,分析数据竞争条件下不确定结果概率分布情况.并在此基础上对所得数据进行分析,建立反映程序对外界环境感知的 HMM 模型,预测待检测的多线程程序产生数据竞争的概率分布,从而指导检测过程.

本文第 1 节给出一个数据竞争程序的实例和分析过程.第 2 节分析多线程程序时序影响因素.第 3 节是该多线程程序实验结果分析.第 4 节建立多线程程序时序分析的隐 Markov 模型.第 5 节是模型实验结果仿真.第 6 节介绍相关工作.最后是总结和展望.

1 数据竞争程序实例和分析过程

1.1 示例程序

```
1  int main()     //Main Thread
2  {long i=0;     DWORD dwEvent;
3          for (i=0;i<2;i++)
4          {      hEvents[i]=CreateEvent(NULL,FALSE,FALSE,NULL);
5                 if (hEvents[i]==INVALID_HANDLE_VALUE)    exit(1);
6          }//创建两个信号量,每个线程在结束之前发出信号
7          for (i=0;i<lRepeatTimes;i++)//线程参数为其 ID 号,分别用 1 和 2 表示
8          {      hFirstThread=(HANDLE)(_beginthread(MyThreadProc,0,& FirstThreadID));
9                 SetThreadPriority(hFirstThread,nPriority);//设置优先级
10       hSecondThread=(HANDLE)(_beginthread(MyThreadProc,0,& SecondThreadID));
11       SetThreadPriority(hSecondThread,nPriority);//等待两个线程信号量都发出,两个线程执行完成
12       dwEvent=WaitForMultipleObjects(2,hEvents,TRUE,INFINITE);
13         if (nResult==cnSecondThreadID)
```

```
14          lFirstThreadFast++;//记录结果为第 1 个线程先执行的次数}
15    cout<<"The sum of first Thread faster than second:"<<lFirstThreadFast<<endl;
16    return 0;
17 }
18 void MyThreadProc(void*pMyID)
19 {   char*MyID=(char*)pMyID;      long i,j=0;
20    for (i= 0;i<lDelayTimes;i++){   for (j=0;j<lDelayTimes;j++)   {}   }
21    nResult=(int)(*MyID);
22    HANDLE hevent=hEvents[(int)(*MyID)−1];
23    SetEvent(hevent);
24 }
```

1.2 线程数据竞争时序图

上述存在数据竞争的 C 程序中,Main 是主线程,使用_beginthread 来创建两个标号分别为 FirstThreadID 和 SecondThreadID 的线程,分别用句柄 hFirstThread 和 hSecondThread 来表示,并且两个线程都对共享数据 nResult 进行写操作,改变其值,存在数据竞争,此时结果是不确定的.主线程中通过调用 WaitForMultipleObjects 操作,产生阻塞,等待直至两个子线程都执行完成,唤醒主线程.在子线程中,使用 pMyID 来标识不同的子线程,同样使用返回值 nResult 来标识不同的线程执行完成,并且在执行结束之前发出信号,通过 SetEvent 唤醒主线程.其中,lRepeatTimes 设置实验的次数,lDelayTimes 设置线程中时延长度,nPriority 用于设置线程的优先级.表 1 中,$t1$,$t2$;$t3$,$t4$;$t5$,$t6$ 执行先后顺序不确定,图 1 给出一种可能的时序图.

Table 1 The threads' time-event mapping relationship table

表 1 线程时刻-事件对应关系

Time point	Event
$t1$	Main thread starts the first thread
$t2$	Main thread starts the second thread
$t3$	First thread write the variable: nResult
$t4$	Second thread write the variable: nResult
$t5$	First thread finishes SetEvent
$t6$	Second thread finishes SetEvent

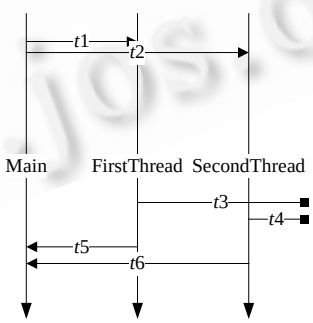

Fig.1 A time sequence diagram for a C program with data races

图 1 存在数据竞争的 C 程序时序图

2 多线程程序时序影响因素分析

2.1 时序因素分析

由于关注重点在于线程运行中的时序关系,在程序行为分析中使用的控制流、数据流以及相关的语义分析

都是必须的,这方面的研究也较多[6−9].本文将注意点聚焦在时序相关的属性分析上.例如,$t3$,$t4$ 两个时刻发生的先后顺序不确定,结果不可预测.那么 $t3$,$t4$ 可以认为是两个随机变量,而不能简单地假定先开始运行的线程总是比后开始运行的线程先访问到两者共享的数据.在不同操作系统以及体系结构上,线程调度方式也可能不同,从而使数据竞争问题更加复杂.动态执行程序,通过程序执行动态特征对线程执行状态进行分析.就最简单的两个线程 1,2 而言,其执行序列可能有多种,例如 122122211….考察两个线程在执行结束点上的情况,1 先执行完,或者 2 先执行完.对于 n 个线程,那么有 $n!$种线程执行顺序(仅考察线程结束点).这样,n 个线程数据竞争的结果最多会有 $n!$个.影响线程 i 执行时间的主要因素如下:

(1) 内部因素:线程本身负荷大小.

N_i:线程中包含操作的数目以及每条指令编译优化后生成目标代码指令的执行周期;

I_i:线程本身因素,是否有中断(例如 I/O 操作);

P_i:线程运行的优先级别.

(2) 外部因素:线程所运行的环境的影响.

A_i:操作系统对多线程程序的调度算法.无论是优先级轮转还是时间片等分,或者是其他算法,会影响多线程程序的切换与执行;

C_i:该线程运行的上下文,操作系统的负载,即当操作系统通过特定的调度算法进行进程和线程切换,该线程不可避免地受到其他线程和进程调度的影响;

O_i:编译器后端,在目标代码生成阶段进行优化的影响.

对于线程 i 而言,线程 i 的执行时间记为 K_i,那么 $K_i=F_i(N_i,I_i,P_i,A_i,C_i,O_i)$由 N_i,I_i,P_i,A_i,C_i,O_i 这些因素共同决定,记为 F_i.对于同一机器上同时运行的不同线程而言,可以认为 I_i,P_i,A_i,O_i 都是相同的,只有 N_i,C_i 不同.如果两个线程操作也完全相同,则 N_i 相同.由于各影响因素之间相互独立,那么 $K_i = F_i'(N_i, I_i, P_i, A_i, O_i)\cdot F_i(C_i)$ .C_i 取决于当前线程运行的工作环境.

2.2 进一步分析

设第 i 个线程实际执行时间为 $t_i(1\le i\le n)$;线程之间的切换时间计为 m,n 个线程总的执行时间为 T,那么 $T=\sum_i t_i + m$,即 $\sum_i t_i = T - m$.不妨假设每个线程的最小执行时间和线程切换时间都为基本时间片 Toc 的整数倍,即 $t_i=k_i\cdot toc$,$m=k_{n+1}\cdot toc$,其中,$k_i(1\le i\le n+1)$为非负整数;那么

$$\sum_i k_i = \frac{T-m}{toc} \tag{1}$$

令 $w=\dfrac{T-m}{toc}$ 为正整数,易知式(1)解的总个数为 C_{w+n-1}^{n-1};至少有 k 个解大于 0 的解的个数为 $\sum_{l=k}^{w} C_n^{n-l}C_{w-1}^{l-1}$.令 $k=n$,其解则对应 n 个线程相互有数据竞争的情况.

令 $k=n=2$ 对应两个线程同时运行的情况.此时,解的总数为$(w-1)$个.当 w 为奇数时,$k_1>k_2$ 的解的个数为 $\dfrac{w-1}{2}$,$k_1<k_2$ 的解的个数为 $\dfrac{w-1}{2}$.当 w 为偶数时,$k_1>k_2$ 的解的个数为 $\dfrac{w-2}{2}$,$k_1<k_2$ 的解的个数为 $\dfrac{w-2}{2}$,$k_1=k_2$ 的解的个数为 1.但是,解的个数并不等于解出现的频度,解的个数分布能够大致反映出线程之间运行时间比较的期望值.用运行中解的个数来逼近运行中不同解出现的概率,假定运行中每个解出现的分布服从均匀分布$\mu(0,1)$,那么每个解出现的概率相同,为 $\dfrac{1}{w-1}$.

令随机变量 $X=\begin{cases}1, & k_1<k_2\\ 0, & k_1=k_2\\ -1, & k_1>k_2\end{cases}$ 反映两个线程执行快慢的差别程度,则 $E(X)=0$.那么对于两个线程而言,实际上两个线程不可能同时启动运行,在相隔 j 个时间片的启动时间间隔中,先启动的线程先运行完成的概率略微变大.这时,要使两个线程同时运行完毕,需要使后启动的线程执行时间 $k_1\ge k_2+j$,满足此条件的解的个数为

$$\max\left\{\left(\frac{w-1}{2}-\left\lfloor\frac{j}{2}\right\rfloor\right),0\right\}(w\text{ 为偶数}),\text{或者 }\max\left\{\left(\frac{w}{2}-1-\left\lfloor\frac{j}{2}\right\rfloor\right),0\right\}(w\text{ 为奇数}).$$

若仍假定每个解出现的分布服从均匀分布$\mu(0,1)$.这时, $E(k_1 \ge k_1 + j) = \max\left\{\left(\frac{w}{2}-\frac{3}{4}-\left\lfloor\frac{j}{2}\right\rfloor\right),0\right\}$.

3 多线程程序实验结果分析

以下从不同角度估计影响 $F_i(C_i)$的特征量.运行的系统参数为 Pentium 1.6G CPU,256M 内存 DDR,WinXp SP2.运行示例第 1.1 节中的程序,选取的几个关键特征为线程优先级别、系统负载、线程运行时间.

3.1 优先级别影响的运行时序分布

按照 Windows 中的约定,线程优先级别 P 的集合有 7 个元素:{THREAD_PRIORITY_IDLE,THREAD_PRIORITY_LOWEST,THREAD_PRIORITY_BELOW_NORMAL,THREAD_PRIORITY_NORMAL,THREAD_PRIORITY_ABOVE_NORMAL,THREAD_PRIORITY_HIGHEST,THREAD_PRIORITY_TIME_CRITICAL}.每个进程和线程都有相应的优先级设置,线程优先级决定何时运行和占据多少 CPU 时间,线程的实际优先级是该线程所属进程优先级和线程本身的优先级别的和.优先级相同的线程按照时间片轮流运行.设定实验次数 *lRepeatTimes* 为 1 000 次,线程中时延长度用空循环 *lDelayTimes*=10000 次来实现,系统较为轻闲(见表 2).第 2 行、第 5 行为第 1 个线程先运行结束的次数(领先次数).第 3 行、第 6 行为映射规则,将优先级 TP 映射到模糊化谓词 *Y* 上.由于 THREAD_PRIORITY_IDLE 和 THREAD_PRIORITY_TIME_CRITICAL 常用于驻留程序和紧急程序,对于一般的多线程程序出现可能性不大,这里未作讨论.

Table 2 The table of threads' priority influences for threads' running sequence

表 2 优先级对线程运行时序影响图

Priority	THREAD_PRIORITY_ ABOVE_NORMAL	THREAD_PRIORITY_ NORMAL	THREAD_PRIORITY_ LOWEST
Number of precedent	9 991	8 444	6 160
Mapping rule: *TP*→*Y*	High	Normal	Very low
Priority	THREAD_PRIORITY_HIGHEST	THREAD_PRIORITY_BELOW_NORMAL	
Number of precedent	10 000	7 510	
Mapping rule: *TP*→*Y*	Very high	Low	

3.2 系统负载影响的运行时序分布

使用当前程序运行时计算机系统的 CPU 占用率和 FP 页面使用率大小来刻画系统负载.在实验时,使用 Trends 杀毒软件进行反复系统扫描以及 Winrar 不停解压缩文件来模拟系统较忙的时刻,关闭其他不相关程序而只留取该程序运行模拟系统较轻闲的时刻,同样可以模拟出系统负载为不忙不闲的机器状态.实验次数仍为 *lRepeatTimes*=10000 次,优先级别为 THREAD_PRIORITY_NORMAL(见表 3).同时我们又进行了扰动实验,记录系统负载在发生随机扰动情况下,当其他条件不变时,线程运行次序的改变大小(见表 4).

Table 3 The table of system overload influence on threads' running sequence

表 3 系统负载对线程运行时序影响图

System load (CPU,FP)	92%, 264M	49%, 182M	2%, 160M	34%, 168M	78%, 225M
Number of precedent	6 273	6 756	8 997	8 553	6 457

Table 4 The table of system overload invariance influence on threads' running sequence

表 4 系统负载扰动对线程运行时序影响图

System load (CPU,FP)	54%, 180M	46%, 178M	4%, 162M	6%, 164M	9%, 154M
Number of precedent	6 746	6 638	8 800	9 234	8 754

3.3 系统运行时间影响的运行时序分布

对于多线程程序选取不同的运行时间,使用增加空循坏次数的方式来增加每个线程的运行次数和运行时间(表 5 中第 1 行为空循环的次数).实验次数仍为 *lRepeatTimes*=10000 次,优先级别为 THREAD_PRIORITY_NORMAL,系统状态为轻闲(见表 5).同时对运行时间进行扰动调整,其他条件不变,得到运行时间分布图(见表 6).

Table 5 The table of running time influence on thread running sequence

表 5 运行时间对线程运行时序影响图

Running time	100	1 000	3 000	6 000	10 000
Number of precedent	10 000	10 000	9 993	8 952	8 444

Table 6 The table of running time invariance influence on thread running sequence

表 6 运行时间扰动对线程运行时序影响图

Running time	80	1 300	3 400	5 900	9 910
Number of precedent	10 000	10 000	9 990	8 742	8 237

3.4 数据结果的分析和比较

通过运行多线程来感知机器状态行为对数据分布的影响,做出下列推断:1) 系统负载越大,不确定性越强,状态条件相同的线程之间运行先后次序差别数目越小.线程之间次序差别并不是严格按照负载增大顺序而线性递减的;2) 线程优先级越低,不确定性越强,状态条件相同的线程之间运行次序差别数目越小.线程之间次序差别并不是严格按照优先级别的增大顺序而线性增加的;3) 每个线程运行时间越长,不确定性越强,状态条件相同的线程之间运行次序差别数目越小.线程之间次序差别并不是严格按照运行时间增大的顺序而线性递减的; 4) 对于运行时间的改变、负载的改变、优先级别的改变,线程运行次序的改变程度不同.这种改变,是体现在趋势上,而不仅仅是一个数字量的改变.因此,第 4 节使用模糊化谓词来描述.

4 多线程程序时序分析的隐 Markov 模型

对多线程程序时序分析建立模型,通过对机器状态、运行时间、优先级别等数据的采集感知多线程程序的运行结果,辅助分析解决多线程程序的数据竞争或者死锁问题.

4.1 隐Markov模型建立

设 X_n 是取值于状态空间 S={0,1}的 Markov 链,其样本不能被实际测量到.对于两个线程的多线程程序而言,能够测量到的是如下的 O_n 样本,即 $O_n=g(X_n)+w_n$.其中,g 是一个未知函数,$\{w_n\}$是独立同分布的随机干扰,只能取有限个值,且与随机过程$\{X_n\}$独立.那么,$\{X_n,O_n\}$就是一个二维的隐 Markov 模型[18,19].

HMM 模型为$\lambda=\{\pi,A,B\}$.

(1) 状态空间:设隐 Markov 链为 $X_n,n=1,2,\ldots,T.X_n$ 对应于不同时刻两个线程运行的先后顺序所得到的随机过程序列.状态空间 $S=\{q_0,q_1\}$,状态数 $N=2.q_0$ 代表 1 线程在前,q_1 代表 2 线程在前.两个线程运行的先后顺序,难以观测,只能通过程序运行的某些数据竞争的结果或者是动态调试过程中程序的输出值推断出来.线程运行过程对用户来说一般是透明的.状态转移概率矩阵:

$$A=(a_{ij})_{N\times N}=\begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix},$$

其中,$a_{ij}=P\{$下一步处于状态 $q_j|$当前处于状态 $q_i\}$,$a_{ij}:\{S\times S\to[0,1]\}$,$a_{ij}$ 满足 $\sum_i a_{ij}=1;\sum_j a_{ij}=1$.

(2) 系统可见符号的集合 $O=\{\{x,y,z\}:x\in X,y\in Y,z\in Z\}$,$O=\{O_1,O_2,\ldots,O_t\}$为可见符号的集合.$X$ 代表系统运行过程中的负载,Y 代表线程运行的优先级别,Z 代表线程的运行时间(或程序的规模).由前面的实验得到,对于系统运行中的负载而言,是 85%CPU 占据还是 86%CPU 占据,不会对多线程程序产生较大的影响.因此,关注重点是

CPU 的使用量级.取模糊化后集合 X={繁忙,忙,中,闲};Y 是线程优先级别的影响,Y={高,较高,中,低,很低};Z 是线程的运行时间,Z={短,中,长}.那么,O 的状态空间大小为 60.那么,$B=\{b_i(k,m,l)\}$(其中,$b_i(k,m,l)=P$\{系统负载为 X_k,该线程优先级为 Y_m,线程的运行时间为 Z_l|当前处于状态 i\},$i\in S$,$1\le k\le K$,$1\le m\le M$,$1\le l\le L$)表示在模型中可见符号的概率分布.$b_i(k,m,l)$:$\{X\times Y\times Z\}\to[0,1]$,满足 $\sum_{\{x,y,z\}} b_i(x,y,z)=1$,$\sum_{k\in X} b_i(k)=1$,$\sum_{m\in Y} b_i(m)=1$,$\sum_{l\in Z} b_i(l)=1$.由于 X,Y,Z 相互独立,易知 $b_i(k,m,l)=b_{ik}b_{im}b_{il}$.

(3) 初始状态分布:$\pi=\{\pi_0,\pi_1\}$,其中,$\pi_i=p$\{第 1 步处于状态 q_i\},$i\in S$.

4.2 可见符号的模糊化过程

给定以下 6 条 if-then 规则,很容易将观测到的 CPU 利用率和 PF 使用率转化为系统忙闲状态的模糊标记(见表 7).模糊标记尺度的选取利用了前面实验中所得的数据,为了能够较好地达到区分系统不同状态目的.线程优先级别的影响,可以使用映射规则 $f:TP\to Y$ 很容易地将观测到的线程优先级别映射到可见符号的输入上面(见表 2).TP 为线程优先级别的集合,Y={高,较高,中,低,很低}.线程的执行时间是一个比较难以衡量的性能指标.我们使用程序的运行时间 t 来衡量.运行时间 Z 集合为{短,中,长}.规则如下:

If $t\le$30s, then 运行时间{短}; if 30s<t<90s, then 运行时间{中}; if $t\ge$90s,运行时间{长}.

Table 7 The table of system overload's state mapping rules

表 7 系统负载状态映射规则图

CPU	75%~100%	35%~75%	0~35%
$PF\ge$200M	Very busy	Busy	Normal
PF<200M	Busy	Normal	Free

4.3 HMM模型λ计算[19]

在识别算法给定后,HMM 模型运转相位的好坏取决于 HMM 模型的学习相位[18,19].设采集的 k 个观察值序列集合为 O:$\{O^1,O^2,\ldots,O^k\}$,其中,$O^k=(O_1^k,O_2^k,\ldots,O_{T_k}^k)$ 是第 k 个序列.假定不同观测序列之间相互独立,目标是调整模型λ的参数,使得 $P(O\mid\lambda)=\prod_{k=1}^{K}P(O^k\mid\lambda)$ 最大.如果训练样本不足,则模型训练不够准确;而训练次数过多,同样有 over-fitting 问题.样本数据使用多线程程序运行中采集的数据,对模型训练采用离线算法进行训练(如图 2 所示).这时,在给定模型和观察序列 O^k 条件下,从 i 到 j 的转移概率定义为

$$\xi_t^k(i,j)=P(s_t=i,s_{t+1}=j\mid O^k,\lambda)==\frac{\alpha_t(i)a_{ij}b_j(O_{t+1}^k)\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(x_{t+1})\beta_{t+1}(j)}.$$

令 $\gamma_t^k(i)=\sum_{j=1}^{N}\xi_t^k(i,j)$ 为 t 时刻处于状态 S_i 的概率,则估计公式如下:

转移概率 $\hat{a}_{ij}=\dfrac{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}\xi_t^k(i,j)}{\sum_{k=1}^{K}\sum_{t=1}^{T_k-1}\gamma_t^k(i)}$,可见符号分布为 $\hat{b}_i(x,y,z)=\dfrac{\sum_{k=1}^{K}\sum_{t=1,O_t^k=\{x,y,z\}}^{T_k}\gamma_t^k(i)}{\sum_{k=1}^{K}\sum_{t=1}^{T_k}\gamma_t^k(i)}$,初始分布为 $\hat{\pi}_i=\dfrac{1}{K}\sum_{k=1}^{K}\gamma_1^k(i)$.

上述模型中,$P(O^k|\lambda)$的求解可借助于使用动态规划策略的前向算法,在定义前向变量 $\alpha_t^k(i)=P(O_1^k,O_2^k,\ldots,O_t^k,q_t=S_i\mid\lambda)$ $(1\le t\le T_k)$后,初始化 $\alpha_1^k(i)=\pi_i b_i(O_1^k)$ $(1\le t\le T_k)$,利用递推公式 $\alpha_{t+1}^k(j)=\left(\sum_{i=1}^{n}\alpha_t^k(i)a_{ij}\right)b_j(O_{t+1}^k)$ $(1\le t\le T_k-1)$,即得 $P(O^k\mid\lambda)=\sum_{i=1}^{N}\alpha_{T_k}^k(i)$.递推计算中,该动态规划算法的时间复杂度为 $O(N^2T_k)$.模型参数计算中,用于重新估计模型参数的时间复杂度为 $O(N\cdot T_k\cdot l\cdot num)$.其中,$N$ 为状态空间大小,l 为可见符号中离散符号的最大个数,num 为用于训练的观测值集合中 O^k 的数目:$|O|$.

Step1: initial model to be trained: λ_0;
Step2: based on λ_0 and observing sequence O, train the new model λ;
Step3: If $\log P(O|\lambda)-\log P(O|\lambda_0)<cv$, end;
Step4: or else, let $\lambda_0=\lambda$, goto Step 2.
(cv is the threshold).

Fig.2 A parameter training algorithm for hidden Markov model

图 2 隐 Markov 模型参数训练算法

5 实验结果仿真

5.1 两个线程程序仿真结果

测试环境为 Visual studio6.0,选取第 1.1 节中取得的样本数据,建立 HMM 模型.实验随机采集了 5,10,20,40,80,120 组观测序列进行训练,观察序列 T_k 长度分别取值 2,4,6.例如,采集长度为 6 的观察序列模糊化后为:{(闲,中,中),(忙,中,中),(中,中,中),(忙,中,长),(忙,中,长),(中,中,长)}}.由于运算量较大,训练样本采集后离线方式计算.按照第 4 节所述算法,采用多组观测序列训练的 Baum-Welch 算法,估计 HMM 模型的参数.由于 Baum- Welch 算法是局部最优算法,不能保证运算结果一定达到全局最优,不同的初始分布会影响训练后的模型参数.算法时间复杂度为 $\max\{O(m\cdot N^2T_k),O(m\cdot N\cdot T_k\cdot l\cdot num)\}$,由算法迭代次数 m 和每次迭代的计算时间共同决定,每次迭代中算法的计算时间为 $\max\{O(N^2T_k),O(N\cdot T_k\cdot l\cdot num)\}$(见第 4.3 节),迭代次数 m 受到收敛参数 δ、观测序列 O^k 和模型初始参数的影响.图 2~图 7 为调节不同的收敛参数 cv=0.001,0.01,0.1,0.2 后,模型训练算法的仿真结果.图 3~图 8 中横轴为观测序列数目(num of training dataset),分别为 5,10,20,40,80,120;在图 3、图 5、图 7 中,纵轴为算法收敛的迭代次数(num of recursive time),在图 4、图 6、图 8 中,纵轴为算法收敛所需的时间(recursive time before convergence/s).可见,该 Baum-Welch 算法随着收敛参数 cv 的减小,其迭代次数和计算时间都不断增大;取不同的观察序列长度发现,该算法的收敛时间随着观测序列的增长而增大.但是在实验中,最坏情况下的收敛时间也都在 1.5s 以内,比较高效.计算时初始分布 $\pi=\{\pi_0,\pi_1\}$ 随机产生,初始令 $A=\begin{pmatrix}0.5 & 0.5\\ 0.5 & 0.5\end{pmatrix}$,而在求解 $\alpha_t^k(i)$ 时出现了下溢问题,从而导致其值随着序列长度 t 的增大而逐渐趋向于 0.采用增加比例因子 S_t^k 的方式对第 4.4 节中的算法进行修正.在定义 $\alpha_1^k(i)=\pi_i b_i(O_1^k)$ $(1\le t\le T_k)$后,令 $S_1^k=\sum_{i=1}^{n}\alpha_1^k(i)$,那么 $\alpha_1^{k^*}(i)=\dfrac{\alpha_1^k(i)}{S_1^k}$.递推公式为

$$\alpha_{t+1}^k(j)=\left(\sum_{i=1}^{N}\alpha_t^{k^*}(i)a_{ij}\right)b_j(O_{t+1}^k),S_t^k=\sum_{i=1}^{n}\alpha_t^k(i),\alpha_t^{k^*}(i)=\frac{\alpha_t^k(i)}{S_t^k}.$$

那么,此时 $P(O^k\mid\lambda)=\sum_{i=1}^{N}\alpha_{T_k}^{k^*}(i)\cdot\prod_{t=1}^{T_k}S_t^k$,故 $\log P(O^k\mid\lambda)=\sum_{t=1}^{T_k}\log S_t^k$.再使用第 4.3 节的算法,计算给出测试样本轨道 O^k 后所得到的 $\log P(O^k|\lambda)$值.对于每条样本轨道,算法的计算时间为 $O(N^2T_k)$.取 T_k=2,4,6 不同序列长度的样本轨道进行测试发现,运行时间在 0.1s 内,满足实时检测的需要.如图 9 所示,横轴为不同的 T_k 取值,纵轴为 $\log P(O^k|\lambda)$的计算时间.分别选取不同长度的样本轨道进行测试(如图 10 所示),取 5 组测试数据进行测试,每组测试数据集大小为 10 个,其中,数据集 data1,data3 中 T_k=2;data2,data4 中 T_k=4;data5 中 T_k=6.图 10 中的横轴表示程序实际执行结果(用第 1 个线程领先的次数标识),纵轴表示使用前向算法的 $\log P(O^k|\lambda)$计算结果.可以看出,相同程序的不同运行结果可以通过前向算法的概率计算值来预测.$\log P(O^k|\lambda)$值的变化大致反映了实际执行结果的分布情况,实际执行结果大小与概率对数值大小大致呈非严格非线性正比例增长,实际运行结果相似的样本轨道预测结果也较为集中.由于受到参数选择和样本轨道长度的影响,再加之系统运行中的噪声因素,上述结果比较满意,说明了模型有效性.那么通过概率计算值,容易有效地快速预测线程执行结果.

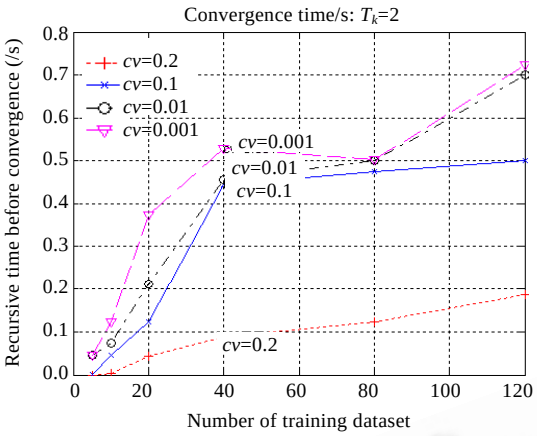

Fig.3 Number of recursive time for B-W algorithm

图 3 B-W 算法收敛迭代次数

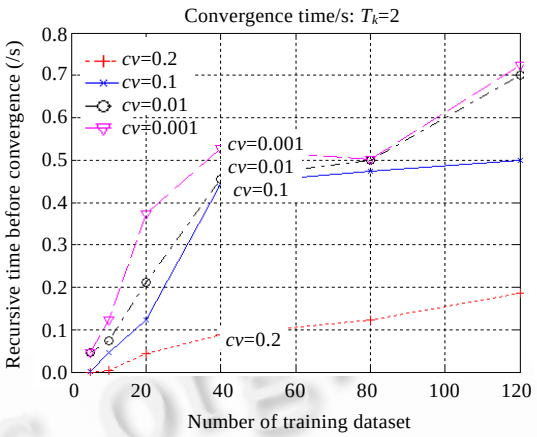

Fig.4 Recursive time(/s) for B-W algorithm

图 4 B-W 算法收敛迭代时间

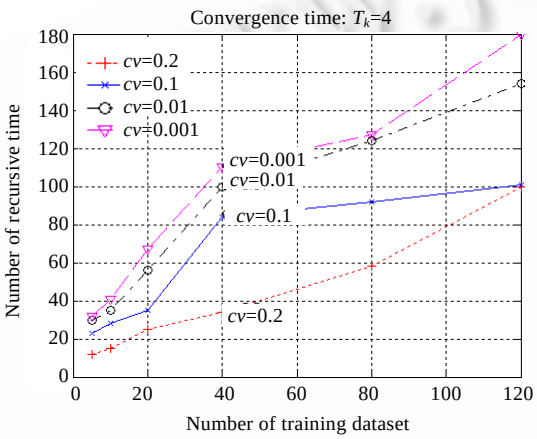

Fig.5 Number of recursive time for B-W algorithm

图 5 B-W 算法收敛迭代次数

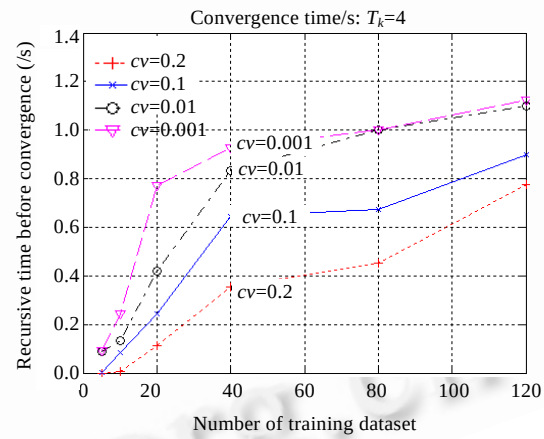

Fig.6 Recursive time(/s) for B-W algorithm

图 6 B-W 算法收敛迭代时间

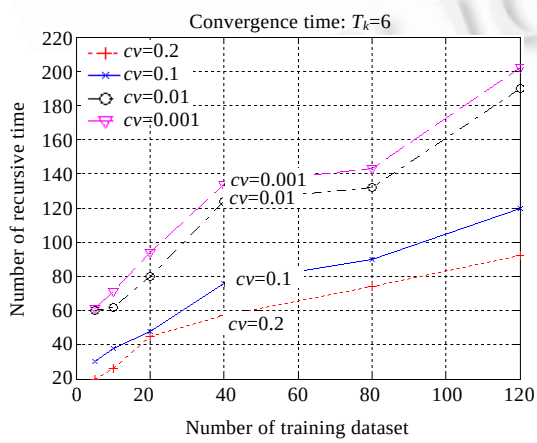

Fig.7 Number of recursive time for B-W algorithm

图 7 B-W 算法收敛迭代次数

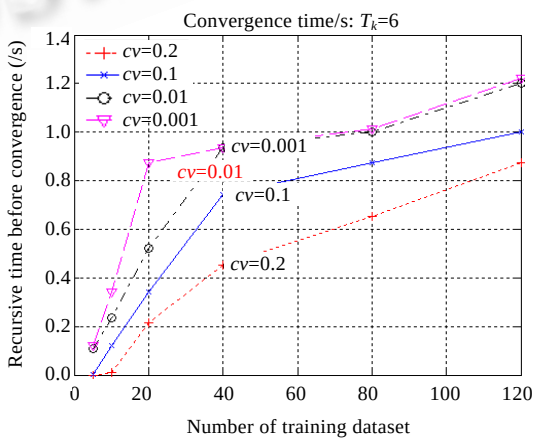

Fig.8 Recursive time(/s) for B-W algorithm

图 8 B-W 算法收敛迭代时间

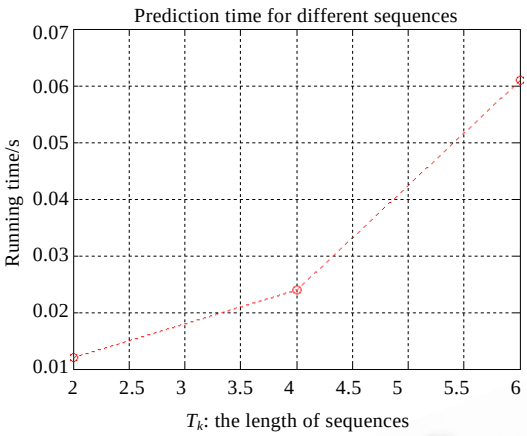

Fig.9 Calculate time (/s) for forward algorithm

图 9 前向算法计算时间(/s)

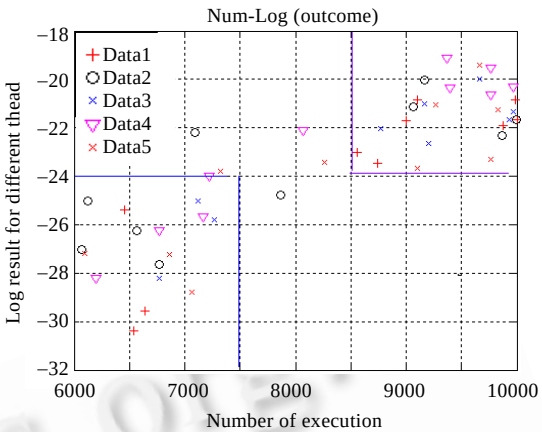

Fig10 $\log P(O^k|\lambda)$ value and true result of the program

图 10 前向算法运行结果 $\log P(O^k|\lambda)$与程序实际执行结果

5.2 对n个线程的推广和不足之处

当推广到 n 个线程时,建立 HMM 模型的状态空间变为 $n!$,随着 n 的增大而大规模增长.对于建立模型 B-W 算法而言,由于其时间复杂度为 $\max\{O(m\cdot N^2T_k),O(m\cdot N\cdot T_k\cdot l\cdot num)\}$(第 4.3 节),而计算概率的前向算法每条样本轨道的计算时间为 $O(N^2T_k)$,可见,影响算法的主要因素就是状态空间 $n!$的大小.在给定模型参数下,经实验发现,n=3,T_k=4 时,每条样本轨道的计算时间仅为 0.247s,而 n=4,T_k=4 时,每条样本轨道的计算时间增大到 11.423s.可以说,$n\leq 3$ 时基本满足实时检测的需要.在 $n>3$ 时,必然要对状态空间使用等价类划分或者状态空间削减以及过大避免等技术,从而使其在 n 较大时变得可行.该讨论比较复杂,我们将在后续的工作中给出详细地论述,以便更好地推广到实际程序分析中.

导致该算法不够精确的原因在于,HMM 模型建立中参数的选择取决于样本数据的采集.由于样本采集与系统运行的随机性,有时训练结果并不能完全真实反映系统各种因素的综合影响,这必然会影响到后面的检测过程,尽可能净化训练样本数据和采用不同时间和空间范围内的表征数据是其解决方案.另外,由于现有时序竞争检测算法都是从程序语义出发,在程序静态或动态分析过程记录和收集影响程序执行变量读写和访问的各种操作[13−16],给出潜在的冲突实例和数据竞争点.而本文分析的角度在于上下文对多线程时序的影响上,重点在于分析和预测数据竞争的结果,这正是原有工作所没有涉及的,所以无法在相同背景下进行比较.

6 相关工作

多线程程序同步意味着同时运行的几个线程之间的协调[1−3]来获得正确的运行顺序,从而避免不可预料的并发相关的错误.3 种主要的同步错误是死锁、数据竞争和原子性被破坏.影响同步的一个重要因素就是时序关系的影响.其中,对于数据竞争错误的研究已经有很多[1−12],归纳起来主要有两种方式检测潜在的数据竞争错误:一种是基于 Lockset 的检测方式[13],当两个或者多个线程访问同一个共享变量而没有加锁时,潜在的数据竞争被报告;另一种是基于 happen-before 分析[14],当两个或多个线程访问相同的共享变量,而访问次序是不确定的.Wang 提出了一种 Multi-lockset 算法[15,16],同时也考虑 happen-before 关系,用于运行时条件竞争检测.陈意云[17]利用静态检测框架,应用精确别名分析算法来静态模拟访问事件发生序,给出了以对象为中心的数据竞争静态检测算法,取得较好的效果.本文没有探讨各种基于语法和语义的时序与同步分析,而是将关注的重点放在了时序分析上,这并不是说语义分析不重要.所提出的基于随机因素的分析方法要借助于多线程程序中体现的语义信息,两者相互结合,能够产生更好的效果.

隐 Markov 模型是一种双重嵌入式随机过程,广泛地应用于语音信号识别、DNA 序列分析、文本信息提取、

软件可靠性分析等领域[18−21].未见有文献关于统计学习模型在多线程程序分析中的应用.隐 Markov 模型的相关算法包括计算序列出现概率的前向后向算法、计算最优状态序列的 Viterbi 算法以及单序列和多观测序列训练的 Baum-Welch 算法[18,19].本文首次尝试将该模型用于时序相关的多线程程序快速有效分析中.

7 结 论

本文对多线程程序进行了时序分析,用随机变量的不确定性特征刻画不同线程之间在时序上的交互关系,分析数据竞争条件下程序不确定结果概率分布情况.并提出一种基于隐 Markov 模型的多线程程序时序分析方法,通过线程运行上下文来感知线程运行的状态,具有较好的效果.进一步的工作包括将 2 个多线程的工作推广到 n 个多线程分析中,提出新的状态空间削减算法,并结合语义分析与时序分析的特点,用于分析指导复杂的多线程程序数据竞争检测.

References:

[1] Netzer RHB, Miller BP. What are race conditions? Some Issues and Formalization. ACM Letters on Programming Languages and Systems, 1992,1(1):74−88.

[2] Carr S, Mayo J, Shene CK. Race conditions: A case study. The Journal of Computing in Small College, 2001,1:88−102.

[3] Choi JD, Loginov A, Sarkar V. Static datarace analysis for multithreaded object-oriented programs. Report, RC 22146, IBM Research, 2001.

[4] Landi W, Undecidability of static analysis. ACM Letters on Programming Languages and Systems (LOPLAS), 1992,1(4):323−337.

[5] JProbe suite. 2007. http://www.componentsource.com/products/jprobe-suite/index.html

[6] Koushik S, Rosu G, Agha G. Online efficient predictive safety analysis of multithreaded programs. In: Proc. of the 10th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004). LNCS 2988, 2004. 123−138. http://www.springerlink.com/content/8b3typ69rv1mce7u/

[7] Rosu G, Koushik S. An instrumentation technique for online analysis of multithreaded programs. Concurrency and Computation: Practice and Experience, 2006,19(3):311−325.

[8] Koushik S, Rosu G, Agha G. Runtime safety analysis of multithreaded programs. In: Proc. of the 9th European Software Engineering Conf. and 11th ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering (ESEC/FSE 2003). 2003. 337−346.http://portal.acm.org/citation.cfm?id=940071.940116&coll=GUIDE&dl=GUIDE&CFID=72158575&CFTOKEN=48265571

[9] Engler D, Musuvathi M. Static analysis versus software model checking for bug finding. In: Proc. of the 5th Int'l Conf. on Verification. LNCS 2937, Berlin, Heidelberg: Springer-Verlag, 2004. 191−210.

[10] Koushik S, Viswanathan M. Model checking multithreaded programs with asynchronous atomic methods. LNCS 4144, Berlin, Heidelberg: Springer-Verlag, 2006. 300−314.

[11] Rmalin G, Context-Sensitive synchronization-sensitive analysis is undecidable. ACM Trans. on Programming Languages and Systems (TOPLAS), 2000,22(2):416−430.

[12] Rinard MC. Analysis of multithreaded programs. Lecture Notes in Computer Science 2126, Berlin, Heidelberg: Springer-Verlag, 2001. 1−19.

[13] Savage S, Burrows M, Nelson G, Sobalvarro P, Anderson TE. Eraser: A dynamic data race detector for multi-threaded programs. ACM Trans. on Computer Systems, 1997,15(4):391−411.

[14] Schonberg E. On-the-Fly detection of access anomalies. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). ACM Press, 1991. 285−297. http://portal.acm.org/citation.cfm?id=74818.74844

[15] Wang LQ, Stoller SD. Runtime analysis of atomicity for multi-threaded programs. IEEE Trans. on Software Engineering, 2006, 32(2):93−110.

[16] Wang LQ. Analysis of synchronization errors for multithreaded programs [Ph.D. Thesis]. New York: Stony Brook University, 2006.

[17] Wu P, Chen YY, Zhang J. Static data-race detection for multithread programs. Journal of Computer Research and Development, 2006,43(2):329−335 (in Chinese with English abstract).

[18] Rabiner L. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. of the IEEE, 1989,77(2): 257−286.

[19] Li XL, Plamon R. Training hidden Markov models with multiple observations—A combi-natorial method. IEEE Trans. on Pattern Analysis and Machine Intelligence, 2000,22(4):371−377.

[20] Gao DB, Reiter MK, Song D. Behavioral distance measurement using hidden Markov models. In: Proc. of the 9th Int'l Symp. on Recent Advances in Intrusion Detection (RAID 2006). 2006. 19−40. http://www.springerlink.com/content/ cr2425318t240n4l/

[21] Tan XB, Wang WP, Xi HS, Yin BQ. A hidden Markov model used in intrusion detection. Journal of Computer Research and Development, 2003,40(2):245−250 (in Chinese with English abstract).

附中文参考文献:

[17] 吴萍,陈意云,张健.多线程程序数据竞争的静态检测.计算机研究与发展,2006,43(2):329−335.

[21] 谭小彬,王卫平,奚宏生,殷保群.计算机系统入侵检测的隐马尔可夫模型.计算机研究与发展,2003,40(2):245−250.

孔德光(1983－),男,山东邹城人,博士生,主要研究领域为软件安全,信息安全.

帅建梅(1961－),女,高级工程师,主要研究领域为网络安全.

谭小彬(1973－),男,博士,副教授,主要研究领域为信息安全.

宫涛(1982－),男,博士生,主要研究领域为网络安全.

奚宏生(1950－),男,教授,博士生导师,主要研究领域为离散事件动态系统,网络性能建模与分析,信息安全.