软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(10):2263−2278 [doi: 10.3724/SP.J.1001.2011.03862]

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

并发计算重复问题与控制方法*

何 倩+, 孟祥武, 陈俊亮, 沈筱彦

(北京邮电大学 网络与交换技术国家重点实验室,北京 100876)

Concurrent Redundancy Problem and Control Methods

HE Qian+, MENG Xiang-Wu, CHEN Jun-Liang, SHEN Xiao-Yan

(State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China)

+ Corresponding author: E-mail: treeqian@gmail.com

He Q, Meng XW, Chen JL, Shen XY. Concurrent redundancy problem and control methods. *Journal of Software*, 2011,22(10):2263−2278. http://www.jos.org.cn/1000-9825/3862.htm

Abstract: Data race is a difficult problem for the development and testing of the concurrent program. Research has found that data race may cause duplicate computing which may decrease a system's performance. First, the concurrent computation redundancy problem (CCRP) is defined. The related performance index and judging methods are given, and the general concurrent redundancy control mechanism is designed. When CCRP is studied, the parallel program generally be analyzed based on a producer-consumer model. In the case of the producer-consumer model with data source, CCRP is analyzed in detail. The single condition and cross condition redundancy control algorithms have different application scopes and can be used as fixed patterns to solve CCRP. Relative property proofs and simulations are given based on Petri net. The concurrent program experiments show that the concurrent redundancy control is necessary and efficient. Two control algorithms are compared in the experiments. The research has reference value for data race detection and concurrent programming.

Key words: concurrent control; Petri net; data race; concurrent computation redundancy; concurrent redundancy control

摘 要: 数据竞争问题是并发程序开发与测试难题,发现数据竞争可能导致计算重复,重复会导致系统性能下降.从实例出发定义了并发计算重复问题(concurrent computation redundancy problem,简称 CCRP),给出了相关性能指标和判断方法,设计了通用并发重复控制机制.并发程序一般都可以基于生产者-消费者模型进行 CCRP 分析.以带数据源的生产者-消费者为例详细分析了 CCRP,给出了单条件、条件交叉两种重复控制算法,算法具有不同的适用范围,都可以作为固定模式来解决 CCRP,基于 Petri 网作了相关性质的证明与仿真.并发程序实验结果说明了并发重复控制的必要性和有效性,比较了两种算法的差异.该研究对于数据竞争检测、并发程序设计具有参考价值.

关键词: 并发控制;Petri 网;数据竞争;并发计算重复;并发重复控制

中图法分类号: TP301 **文献标识码**: A

* 基金项目: 国家自然科学基金(60432010, 60872051); 国家重点基础研究发展计划(973)(2007CB307103); 国家科技支撑计划(2006BAH02A11)

收稿时间: 2009-12-23; 修改时间: 2010-03-05; 定稿时间: 2010-04-14

随着计算机系统的广泛普及以及计算在网络上的流行,特别是近年来微处理器技术的巨大发展,计算机并发问题的研究越来越受到人们的重视.使用并发程序能够充分发挥硬件的计算能力,提高系统的性能.但是由于目前缺乏良好的并发编程工具,正确编写多线程的并发程序是比较困难的[1].基于共享存储的并行程序可能会出现数据竞争情况,这种情况通常是由于错误的同步造成的,比如缺少必要的同步语句等.在共享存储程序中,这样的错误会使程序运行具有不确定性[2].

数据竞争属于海森堡错误,具有不确定和难重现的特点.目前,静态检测和动态检测技术方法都难以精确地找出并发程序中所有的数据竞争问题[3,4].传统的并发控制方法有锁、信号量、管程等,在具体的实际应用中合理地组合以达到高并发而又没有数据竞争等问题,是困扰并发程序开发的难题之一.我们在实际的内容分发网关加、解密系统开发中遇到了一种由于并发导致的特殊问题,系统触发的同一个任务有时会重复地被执行多遍,严重影响了系统性能.通过分析我们发现,这种重复的本质源于数据竞争,并发程序中各线程(本文中的线程也可以是进程)由于不确定的发生顺序,导致了同样的任务被多次重复地执行,任何并发程序出现错误的同步都有可能导致这种错误.直接使用传统的并发控制来避免这种重复计算是困难的,需要一种较为繁琐的组合机制.对于复杂程序,例如一些基于 CORBA,Web 服务等技术的分布式应用系统,实现起来容易出错.实施并发计算的一个本质目的就是为了充分提高系统的运行效率,如果并发由于重复计算而导致性能下降,采用并行计算意义就不大.另外,作为数据竞争问题的直接后果,并发计算重复问题的研究可以为并发程序中数据竞争问题的检测提供参考.重复计算的并发控制流程复杂,需要借助合理的工具和理论进行分析.目前,针对并发程序的常用分析方法有 Petri 网、自动机等.Petri 网是一种具有严格数学语义的形式化图形建模工具,适合描述并发、异步和分布式系统,相关的文献[5–11]引入 Petri 网对并发程序以及并发控制进行了有效的分析.

为了深入分析并发程序中的重复计算问题,方便并发程序的正确编写和数据竞争问题的检测,本文从实例出发定义了并发计算重复问题,提出了相关检测办法,得到通用控制模型.构建了一个轮询驱动、带数据源的生产者-消费者模型作为算法研究的具体对象,基于 Petri 网,分析证明了并发计算重复问题的存在,设计了单条件重复控制和条件交叉重复控制这两种算法,它们可以作为并发控制的固定模式来解决并发计算重复问题.

本文第 1 节介绍 Petri 网的基本概念.第 2 节定义并论述并发计算重复问题.第 3 节提出通用控制模型.第 4 节对生产者-消费者模型进行并发计算重复性分析.第 5 节给出并发计算重复控制算法.第 6 节进行实验分析与讨论.第 7 节介绍相关工作.第 8 节总结全文.

1 Petri 网基本概念

Petri 网是一种具有严格数学语义的形式化图形建模工具,本文将基于 Petri 网以及 Petri 网的图形建模能力来描述和解决这类涉及计算重复的并发控制问题.Petri 网的详细介绍请参考文献[12–14],下面我们只简单介绍与本文相关一些基本概念.

定义 1(Petri 网(库所/变迁系统)). 六元组 $PN=(P,T,F,K,W,M_0)=(N,M_0)$称作 Petri 网($P/T$-系统)[12],当且仅当:

(1) P 是库所的有限集合,T 是变迁的有限集合,且 $P\cup T\neq\varnothing$,$P\cap T\neq\varnothing$;

(2) $F\subseteq(P\times T)\cup(T\times P)$是弧的集合;

(3) $dom(F)\cup cod(F)=P\cup T$;

(4) $K:S\to N^+\cup\{\infty\}$是容量函数;

(5) $M:P\to\{0,1,2,\ldots\}$是标识函数,M_0 是初始标识;

(6) $W:F\to N^+$是每条弧所关联的权函数.

$\forall x\in P\cup T$,称${}^\bullet x=\{y|(y\in P\cup T)\wedge(y,x)\in F\}$为 x 的前置集,$x^\bullet=\{y|(y\in P\cup T)\wedge(x,y)\in F\}$为 x 的后置集.

定义 2(并发性(concurrency)). 考虑网 $PN=(P,T,F,K,W,M_0)$,变迁集 $U\subset T$,如果 $M\geqslant\sum_{t\in U}w(-,t)$, 则称变迁集 U 是并发的[12,13].

定义 3(有界性和安全性(boundedness and safeness)). 在 $PN=(N,M_0)$中,如果$\forall M\in R(M_0)$都有 $M(p)\leqslant K$,则称库所 p 为 K 有界或 K 安全.如果 PN 中每一库所都是 K 有界的,则称 PN 为 K 有界(安全)的[12].

定义 4(活性(liveness)). 在 $PN=(N,M_0)$中,若$\forall M\in[M_0\rangle\exists M'\in[M\rangle$:$t$在$M'$授权发生,则说$t\in T$是活的;若所有$t\in T$都是活的,则 PN 是活的.若$\forall M\in[M_0\rangle\exists t\in T$:$t$ 在 M 授权发生,则 PN 是无死锁的[12].

定理 1. 若 N 为 $T_$图[12],则 N 为活的基本网系统的充分必要条件是:

(1) $\forall t\in T,\exists l\in L:e\in l$,即每个事件至少属于一条简单有向圈;

(2) $\forall l\in L,\exists b\in B:b\in l\cup c_{in}$,即初始情态 c_{in} 每条简单有向圈都至少有一个托肯.

在 Petri 网图中,用一个圆圈表示库所 P 元素,用一个四方形或者竖棒表示变迁 T 元素,有向直线或曲线表示有向弧,黑点表示托肯.Petri 可以形象地描述顺序、并发、分支、循环等程序逻辑,方便对问题的简化分析.本文的 Petri 网描述采用了 FMC[15]的图示方法.

2 并发计算重复问题

2.1 例子程序

我们首先看两个数据竞争的例子:

```
P1
//依状态查询数据,读
(1) A=Query(db,flagA);
//把 A 插入到 P2 的处理队列 B
(2) insetB(A);
```

例 1

```
P2
(1) b=Popup(B);
(2) While (b!=null)
(3) {
(4) Process(b);
(5) Update(db,b,succ);
(6) b=Popup(B);
(7) }
```

```
P1,P2,…,Pn
//ShareA 为共享变量
(1) if (ShareA==a)
(2) {
(3)   Process(ShareA);
(4)   ShareA=b;
(5) }
```

例 2

例 1 中:P_1,P_2 是分别处于不同线程的两段代码,这两个线程都可以循环地运行,如主动对象[16];*db* 是数据库或全局的数据结构;*B* 是 P_1 和 P_2 的共享变量.显然,P_1 和 P_2 之间缺乏必要的同步,即存在数据竞争,无法保证 P_1 写入的 *B* 是否能在 P_2 中全部处理完成后,P_1 才再次运行到行 1.这样,*B* 就具有不确定性.另外我们还可以发现,*B* 的不确定性并不会影响 *db* 的最终状态,而只会在 *B* 中插入重复的值,从而导致 P_2 代码段有不确定的执行次数,这个次数大于等于无数据竞争情况下的次数.

例 2 给出了同一段代码被多个线程 $P_1,P_2,\ldots,P_n$ 调用的情况.*ShareA* 为线程间共享变量,由于缺乏必要的同步,*ShareA* 的值具有不确定性,存在数据竞争.假设如下的一个场景:线程 P_1 调用完 *Process* 函数,还没来得及更新 *ShareA* 的值;线程 P_2 运行到第 1 行时,*ShareA* 的状态仍然等于 *a*,P_2 会进入第 3 行,再一次调用 *Process* 函数.P_2 的这个调用不会影响 *ShareA* 最后的值,但是显然是重复多余的(因为 P_1 已经完成了 1 遍).

2.2 定 义

定义 5(并发计算重复问题(concurrent computation redundancy problem,简称 CCRP)). 这是指在并发程序中,针对目标任务集 *H*,存在某种特定的并发组合 *P*,出现不必要的重复计算,同一段代码会对同一项任务进行两次或两次以上的相同处理.并发组合是调用某代码段的并发实体事务块$\{p_{1_1},p_{2_1},\ldots,p_{i_m}\}$的笛卡尔集,并发计算重复计算问题最后实际执行的任务集为 R,$H\subset R$.

定义 6(并发计算不重复问题(concurrent computation no redundancy problem,简称 CCNRP)). 这是 CCRP 问题的否命题,在任何程序并发组合下都有 $H=R$.

定义 7(并发重复控制机制(concurrent redundancy control mechanism,简称 CRCM)). 这是一种特殊的并发控制方法,利用同步、条件检测、事务等方法来实现 CCRP 到 CCNRP 的转换.

定义 8(研究代码(code studied,简称 CS)). 对应于目标任务集的程序上下文,是 CCRP 的具体研究对象,它可以是整个应用程序也可以是程序的某个部分.CS 存在如下关系:

$$ts_1 \text{ is CCRP}\wedge ts_1\subset ts_2\Rightarrow ts_2 \text{ is CCRP}.$$

为了便于分析,CS 需要满足一定的条件.不能选择那种存在创建线程调用,并且包含一部分主线程代码和

一部分新创建线程代码,但是又没有包含完整的新创建线程代码的情况.这种特殊的情况分析起来过于复杂.同时,我们可以把新创建的线程执行代码完整地包含在 CS 之中.所以,实际中对这种情况进行分析也不是必要的.

定义 9(目标代码(object code,简称 OC)). 这是指 CCRP 中那部分可能出现重复执行的代码.OC 是 CS 的一部分,OC 由于并发被不必要地重复执行就意味着 CS 是 CCRP.

根据 CCRP 的定义,出现 CCRP,说明任务执行具有不确定性.通过 CRCM 正确同步之后,CCRP 是可以避免的,这说明 CCRP 是由于错误的同步造成的.所以,出现并发计算重复问题的本质原因是由于并发实体的执行没有一个确定的顺序,具有随机性,而且程序中出现了数据竞争问题.数据竞争是 CCRP 产生的必要条件,但不是充分条件.CCRP 分析首先需要确定 CS,OS 和 H,然后研究 CS 在各种并发组合下的实际执行任务 R,再进行 H 和 R 之间的比较.“同一个任务”是指对应于 CS 和 OC 具有相同数据输入、相同处理过程、相同输出的程序请求.在实际应用中,为了简化,我们可以只考虑最影响系统性能的那一部分代码作为 CS 和 OC 来进行 CCRP 的研究.

2.3 性能指标

表 1 给出了文章中用到的主要符号.计算效率的一个直接体现就是时耗,完成同样的任务耗时越多说明效率越低.我们讨论 CCRP 时可以考虑以下 3 项性能指标:

(1) 并发计算重复有效度(Q)描述目标任务 H 的耗时和实际耗时之比,$Q \leqslant 1$,越接近 1 越好.对于 CCNRP 问题,$Q=1$;

$$Q=T_G(H)/T(H) \tag{1}$$

(2) 并发计算重复计算度(CR)描述重复任务的耗时占实际耗时的比率,$CR=T_G(R-H)/T_G(R)$,且 $CR=1-Q$.对于 CCNRP 问题,$CR=0$;

(3) 效率提高比($E(n)$),指的是完成相同任务集 H(H 中有 n 个子任务)并发系统中引入 CRCM 后在总耗时上减少的比例,$E(n)=CR,n=|H|,E(n)\geqslant 0$.

Table 1 Semantics of notations

表 1 符号语义表

Symbol	Definition
$T_G(K)$	The consumed time of completing task set K in the ideal condition
$T_{CR}(K)$	The consumed time of completing objective task K, when CRCM is added
$T(K)$	The actual consumed time of completing objective task K
$t_b^{(j)}$	The consumed time of completing single task j
t_c	The polling interval of producer thread
$T^{(i)}$	The consumed time of consumer completing all the tasks in the ith polling

2.4 判断方法

判断并发程序中是否存在重复计算是一件比较困难的事情,根据定义,首先需要确定目标任务集 H,然后可以从以下 4 个方面检测并发重复计算.

(1) 模式检测法

把可能出现并发重复计算的程序结构用 XML 或其他模式来描述,然后在源代码中扫描需要检测的代码,如果出现对应的规则说明并发程序存在问题.这种方法使用方便、准确,其难点在于如何得到合理的规则库.

(2) 时间估计比较法

选取若干任务,固定任务的输入,在非并发情况下单独完成,得到完成的时间作为理想完成时间,构成基本任务集.然后把基本任务集多次执行,保证其在并发程序里的分布密度,监视任务的完成时间 $T(R)$.因为目标任务 H 的总耗时 $T(H)=\sum t_b^j$,比较 $T(R)$和 $T(H)$的差,这种差异如果显著则存在重复计算.差异显著性的确定算法可以采用置信区间法.该方法直观且易于实现,考虑到多线程的加速比,只适应于消费者为单线程的情况.另外,对于一些耗时短的任务,如果分布的密度不够,也不一定能够检测出来.

(3) 理论分析法

基于 Petri 网、自动机等理论描述并发程序,然后对于并发程序中的各种场景进行抽象、假设,分析比较 H 和 R.可以假定在程序中不存在任务丢失的情况,那么只需比较$|H|$和$|R|$.本文将基于 Petri 网作理论分析.

(4) 程序跟踪法

构建一个独立的观察者程序,对于定义的任务集 H 在程序中的执行过程进行监视,如果发现出现重复则可以立即报告.这种方法与后面在并发程序控制中提出的通用控制模型有类似之处.

3 通用并发重复控制模型

3.1 基于生产者-消费者的CCRP分析方法

生产者-消费者描述的是有一群生产者进程在生产产品,并将这些产品提供给消费者进程去消费[17].在生产者-消费者模型中,既有生产者或消费者自身进程的并发,又有生产者和消费者之间的并发,具备并发程序的典型特征.在研究并发程序 CCRP 问题时,一般都可以基于生产者-消费者来进行分析.假设对于 CCRP 分析选择的研究代码满足第 2.1 节中提出的条件,那么可以分成如下两种情况进行讨论:

(1) CS 存在线程创建.这种情况下,我们可以线程创建为界,把主线程的代码视为生产者,把新创建线程的执行代码视为消费者;

(2) CS 不存在线程创建.如果 CS 不被并发调用,显然,CS 不是 CCRP;如果 CS 被并发调用,则可以目标代码为界,把 OC 的前趋、后继视为生产者,OC 视为消费者,具体分解转化过程参见第 3.2 节.

3.2 程序分解转化

假设某一段代码 C 由 $C_1,C_2,C_3,\ldots$,等几个部分构成,我们需要监视 C_2 代码段的计算重复性.C 就是 CCRP 的 CS,C_2 为 CCRP 的 OC.显然,C_1 是 C_2 任务的产生源(生产者),在 C_1,C_2 之间插入 C_{12} 做检测控制操作;C_2 就是任务的处理者(消费者),在 C_2 的完成点,C_2,C_3 之间插入 C_{23} 做检测控制操作,得到新的程序 $C_1,C_{12},C_2,C_{23},C_3$,如图 1 所示.考虑这段程序属于并发程序里的一个部分,则可以把 C_1 和 C_2 视为分别处于不同线程的两个部分,把插入检测控制块后的程序分解成生产者-消费者模型,用 Petri 网描述,如图 2 所示.其中,C_{12} 分解到生产者和消费者各一部分(C_{12_1},C_{12_2}).

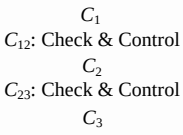


Fig.1 Program flow chart after inserting control block

图 1 插入控制块后的程序流程图

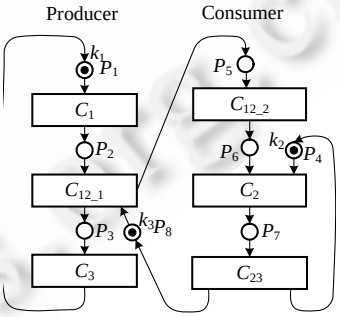


Fig.2 Petri net of converted PCM (producer-consumer model)

图 2 转化的生产者-消费者 Petri 网图

3.3 控制流程

控制流程主要分成 3 个部分:检测条件定义、检测条件检查、检测条件更新.检测条件定义可以创建每一个任务的一个控制块,包括任务的标识、状态;检测条件检查点有 3 个:生产者的任务放入(如 C_{12_1})、消费者的任务取出(如 C_{12_2})、消费者的任务完成(如 C_{23}),条件检查基于任务控制块判断是否进行下一步的操作;检测条件更新是指在必要的时候更新任务控制块,通常伴随检测条件检查而发生.第 5 节我们将详细讨论控制流程的

实现.

3.4 检测条件

检测条件的产生主要有两种方法:

(1) 基于数据竞争变量自身.并发计算重复问题是由于数据竞争造成的,直接基于共享变量构建任务控制块来保证生产者不生产重复的任务给消费者,直观且不需要额外增加数据结构.但是这种方法在“检测条件更新”时需要对数据竞争变量进行直接的操作,这是一个限制;

(2) 增加任务监视控制块.对每个进入消费者的任务进行记录,通过比较去除生产者产生的重复任务来避免消费者重复操作.增加的控制块可以是一个最大序号计数器,也可以是记录当前消费者任务队列的 Hash 表.实际应用中,需要结合已有程序自身的特点选择合适的数据结构.

对条件检测的判断可基于关系运算符,关系运算符主要有 3 种“>,<,=”.所以,在进行条件检测产生的时候可以构建增序序列或降序序列,通过比较运算“>”或“<”来实现条件检查.这种方法可以避免条件的更新,如面包师算法[17];也可以为每个任务选择一个主键,然后通过“=”来实现条件检查.如果任务本身的主键不易选定,则可以使用 Hash 函数基于任务的关注特征生成一个主键.

4 生产者-消费者模型分析

生产者-消费者模型(producer-consumer model,简称 PCM)是 6 大并行算法模型之一,常用来讨论并发程序、并发控制问题[17,18].PCM 为使生产者线程和消费者线程并发进行,在它们之间设置了一个具有多个缓冲区的缓冲池,生产者线程可以将其所生产的产品放入一个缓冲区中,消费者线程可以从一个缓冲区中取得产品去消费[17].在进行 CCRP 问题研究的时候,并发程序一般都可以基于 PCM 进行分析,所以对 PCM 的研究很有代表性.本文后续的工作将围绕 PCM 展开,PCM 为研究代码,消费者为目标代码.

为了让 PCM 能够触发和执行,给 PCM 增加一个数据源,生产者从数据源中得到有意义的数据,数据源是以意义标识为主键的一组数据,是并发程序中的全局数据结构,决定了消费者的目标任务.这类变形的生产者-消费者问题如图 3 所示,这种设计在使用了并发流水线模型的网络应用系统是比较常见的.例如一些面向消息的持久通信系统、数据库应用系统的服务器端等.针对这种并发应用,数据源 A 决定了目标任务集,生产者根据条件 p 查询过程 $f_p(A),f_p(A)\rightarrow B$,然后消费者过程 $f_c(B),f_c(B)\rightarrow A',A'$就是实际完成的任务.$A'$是否与 A 一一对应就决定了程序中是否存在重复计算,并且有$|H|=|A|,|R|=|A'|$.

带数据源的生产者-消费者模型可以用 Petri 描述,如图 4 所示.生产者第 i 次执行,根据 F_1 条件从数据源 A 中取出某个符合条件的数据 A^i,设 $FA=\{a|a$ 为满足条件 F_1 的数据$\}$,则 $A^i=\{a_j|a_j\in FA\}$,然后把 A^i 中的每一个元素 a_j 放入消费者数据队列 B 中,$B^i=B_r^{i-1}+A^i$,B_r^{i-1} 是上一次尚未处理完的任务;消费者对 B 中的元素进行处理,处理完后修改对应数据源 a_j 的状态,a_j 不再满足条件 F_1.生产者定时产生数据,消费者只要 B 中有数据就会不断循环工作.图中 P_1,P_2,P_3 的容量为生产者的并发数 k_1,P_4,P_6 的容量为消费者的并发数 k_2,P_5 和 P_7 的容量为缓冲区的大小 k_3,P_7 可以用来控制消费者数据队列的长度,避免数据溢出和 Petri 网无界.变迁 T_1 实现数据查询;变迁 T_2 把数据放入到消费者数据队列;变迁 T_3 是一个定时器操作;变迁 T_4 是消费者消费和处理数据的操作;变迁 T_5 实现消费者完成数据消费后的状态更新.

命题 1. BDPN 属于 CCRP.

证明:当 $t_c<T^{(i)}$,生产者进行第 i 次数据查询时,消费者数据队列 B 中还有一部分数据 B_r^{i-1} 未处理完毕.$\forall b_i, b_i\in B_r^{(i-1)}$ 都有 $b_i\in FA$,所以 b_i 会被 T_1 再次选中而被 T_2 放入 B.B 中出现重复的数据,使得$|R|>|H|$.生产者在进行数据查询时$(R_1(T_1))$,只要存在处于 P_5,P_6 库所和 T_4,T_5 变迁$(R_2(P_5,T_4,P_6,T_5))$的消费者就会导致重复计算,其可能区域是$R_1(T_1)\cap R_2(P_5,T_4,P_6,T_5)$.同理,如果$k_1>1$,且执行 T_1 操作时 B_r^{i-1} 不为空,那么也会导致计算重复.此时,重复计算的可能区域是消费者的整个生命周期.所以,BDPN 属于 CCRP. □

定理 2. 单线程生产者 PCM 进行并发重复控制的必要性定理.

$$\forall i,\ \begin{cases}\text{if } (T^{(i)} > t_c),\ \text{necessary}\\ \text{if } (T^{(i)} \leqslant t_c),\ \text{not necessary}\end{cases}.$$

证明:如果 $T^{(i)} \leqslant t_c$,那么在每次生产者线程定时周期里,消费者都可以完成所有的任务处理,当生产者执行 T_1 操作时, B_r^{i-1} 为空,故不会重复计算,并发重复控制不是必须的;如果 $T^{(i)} > t_c$,由命题 1 可知会重复计算任务,所以需要进行并发重复控制.

对于多线程(k_1>1)的生产者,由于一个生产线程可能在另一个生产者 T_1 发生之后马上进行 T_1 操作,即使 $t_c < T^{(i)}$也可能产生重复的计算.所以,多线程的生产者 PCM 必须进行并发重复控制. □

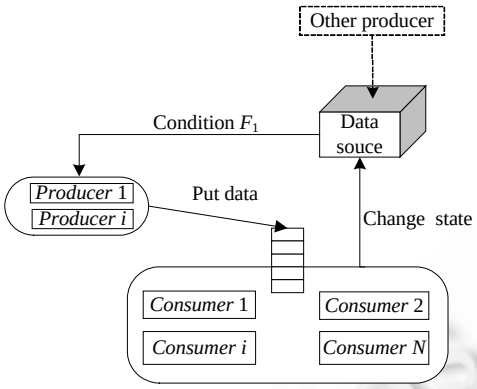


Fig.3 Producer-Consumer model with data source

图 3 带数据源的生产者-消费者模型

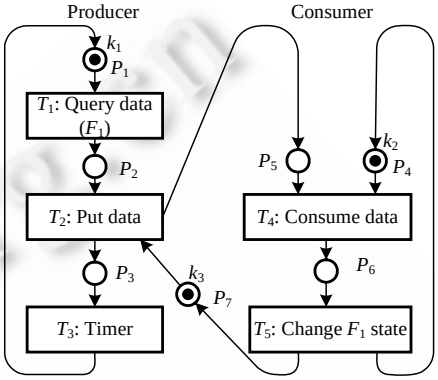


Fig.4 Basic PCM Petri net with data source (BDPN)

图 4 带数据源的基本生产者-消费者 Petri 网图

5 并发重复控制算法

并发计算重复控制的难点与数据竞争问题一样,关键在于如何正确地同步.以下我们将介绍两种并发重复控制算法.

5.1 单条件重复控制算法

BDPN 网 T_1 查询数据后,这些数据的状态在数据源上并没有发生改变,在生产者轮询时间足够快,而消费者处理速度又较慢的情况下,生产者就会把已经推入消费者队列的数据再次放入.这样,可以在 T_1 后、T_2 前增加一个操作 T_{1_1},把当次查询得到所有数据的数据源更改到一个临时状态,并且通过同步锁(P_8)来保证在同一时刻只有一个生产者执行 T_1 和 T_{1_1}.单条件并发重复控制的 Petri 网(SCPN)描述如图 5 所示.

显然,SCPN 网中因为生产者在完成数据查询后及时更新了数据源的状态,所有进入消费者数据队列的数据 a_i($a_i \notin FA$)都处于中间状态,不会被 T_1 选中,而且同步锁保证了同一时刻只有一个生产者执行 T_1 和 T_{1_1}.这样,即使消费者迟一些更新这个状态也不会导致重复计算.SCPN 的生产者和消费者处理过程可描述如下:

算法 1. 生产者执行函数 *Producer*().

输入:任务源 *H*,时间 *T*;

输出:消费者任务队列.

1) While (true)

{ 2) 加同步锁 *m*1;

3) 按查询条件 F_1 到数据源获取数据 A^i 集;

4) 修改 A^i 的数据源状态,不再被 F_1 所查到;

5) 释放同步锁 *m*1;

6) 把 A^i 放入消费者任务队列,如果任务队列满,则需要等待;

7) 休眠一定的时间;}

算法 2. 消费者执行函数 *Consumer*().

输入:消费者任务队列;

输出:完成的任务.

1) While (true)

{ 2) 从消费者任务队列取出一个任务 $Task_i$;

3) 完成任务 $Task_i$;

4) 修改 $Task_i$ 对应的数据源状态为完成状态;}

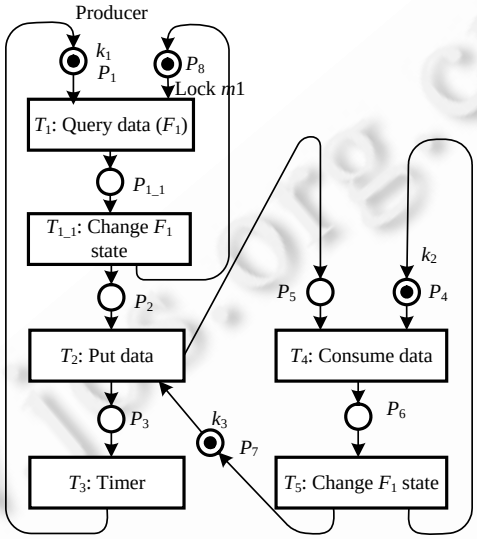


Fig.5 Single condition redundancy control Petri net

图 5 单条件重复控制算法 Petri 网图

5.2 SCPN性质

并发程序设计需要保证算法的有界性、活性、无死锁,它们是 Petri 网系统的重要性质.SCPN 只有具备这些性质才能说明算法是正确的,这里我们给出相关证明.

命题 2. SCPN 网是安全有界的.

证明:因为 SCPN 网中所有变迁均无输入变迁,且初始标识 $M_0(P_1,P_3,P_4,P_5,P_6,P_7,P_4,P_2,P_{1\_1},P_8)=M_0(k_1,0,0,0,k_2,k_3,0,0,1)$,从 SCPN 可达图可知,$\forall M\in R(M_0)$都有 $M\leqslant \mathrm{Max}(k_1,k_2,k_3)$,其中:$M(P_i)\leqslant k_1,i=1,2,3$;$M(P_i)\leqslant 1,i=1\_18$;$M(P_i)\leqslant k_2,i=4,6$;$M(P_i)\leqslant k_3,i=5,7$.由定义 3 可知,SCPN 网是 K 安全有界的. □

命题 3. SCPN 网是活的.

证明:由图 5 可知,$\forall p\in P:|{}^\bullet p|=|p^\bullet|=1$,则 SCPN 网属于 $T_$图.SCPN 中存在 4 个简单有向圈:$[P_1,T_1,P_{1\_1},T_{1\_1},P_2,T_2,P_3,T_3]$,$[P_8,T_1,P_{1\_1},T_{1\_1}]$,$[P_7,T_2,P_5,T_4,P_6,T_5]$和$[P_4,T_4,P_6,T_5]$.显然,$\forall T_i\in T,\exists l\in L:e\in l$,每个变迁 T_i 至少属于一条简单有向圈;又 SCPN 的初始情态为 $M_0(P_1,P_3,P_4,P_5,P_6,P_7,P_4,P_2,P_{1\_1},P_8)=M_0(k_1,0,0,0,k_2,k_3,0,0,1)$,所以每个简单有向圈都存在托肯.由定理 1 可知,SCPN 网是活的. □

5.3 Petri网性质仿真分析

使用 GSPN 工具 PIPE[19]对图 5 中的 SCPN 进行 Petri 网相关实验,实验中增加 k_1,k_2,k_3 的托肯数,SCPN 的可达状态会急剧增加,由于 k_1,k_2,k_3 分别代表了并发线程数和缓冲区大小,这充分说明了并发程序状态分析是非常困难的.在 $k_1=1,k_2=2,k_3=2$ 时有可达状态 23 个,当 $k_1=1,k_2=2,k_3=3$ 时有可达状态 35 个,当 $k_1=2,k_2=2,k_3=3$ 时,有可

达状态 53 个.在所有仿真实验中,SCPN 都能够得到可达标识图,进行结构分析都显示安全、活性无死锁等性质,这进一步验证了命题 2 和命题 3.当 k_1=1,k_2=2,k_3=2 时,SCPN 可达图如图 6 所示,可达标识集及稳态概率见表 2.S_0~S_3 的稳态概率较高,这 4 种标识状态占到所有可能的 50%以上.对于并发计算重复问题,我们要优先避免稳态概率高的状态下出现计算重复.

Table 2 Reachable markings of SCPN and stable probability

表 2 SCPN 中的可达标识集及其稳态概率

	P_1	P_3	P_5	P_6	P_7	P_4	P_2	P_{1_1}	P_8	Prob.
S_0	1	0	0	0	2	2	0	0	1	0.115 59
S_1	0	0	0	0	2	2	1	0	0	0.162 15
S_2	0	0	0	0	2	2	0	1	1	0.196 79
S_3	0	1	1	0	1	2	0	0	1	0.105 48
…	…	…	…	…	…	…	…	…	…	…	…
S_{23}	0	0	0	2	0	0	0	1	1	0.006 94

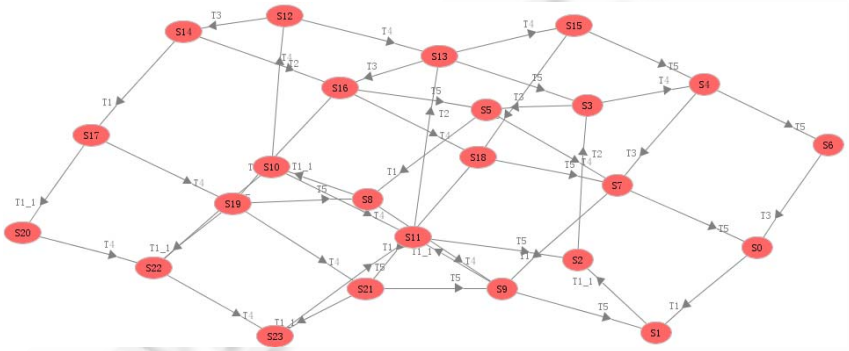


Fig.6 Reachable graph of SCPN (k_1=1, k_2=2, k_3=2)

图 6 SCPN 可达图(k_1=1,k_2=2,k_3=2)

5.4 条件交叉重复控制算法

条件交叉重复控制算法增加了一个新的数据结构 B 来记录进入消费者的所有任务,在 BDPN 网的基础上把数据查询 T_1 和状态更新 T_5 进行了拆分,条件交叉重复控制 Petri 网(CCPN)描述如图 7 所示.在 CCPN 网中有两个检测条件,$a_i \in FA$ or $a_i \notin FA(F_1)$和 $a_i \in B$ or $a_i \notin B(F_2)$,其中,F_2 条件使用的 B 是一个模型内局部变量,对 B 的操作不会影响全局数据源.释放检测条件 F_1 和 F_2 的顺序与检测 F_1 和 F_2 的顺序相同,呈交叉状.在执行 T_1,T_2 操作的同时,通过同步锁 $m1$ 保证不会发生 T_6.显然,$m1$ 锁的获取和释放不满足死锁发生的条件[20].为了简便起见,在图 7 中同步锁 $m1$ 对于变迁 T_6 的控制不特别进行标识.在 CCPN 网中,P_2,P_{10} 的容量为 1,P_1,P_3,P_4 的容量为 k_1,P_5,P_7,P_8 的容量为 k_2,P_6 和 P_9 的容量为 k_3.

CCPN 的生产者和消费者处理过程可描述如下:

算法 3. 生产者执行函数 *Producer*().

输入:任务源 H,时间 T;

输出:消费者任务队列.

1) While (true)

{ 2) 加同步锁 $m1$;

3) 按查询条件 F_1 到数据源获取数据集 A^i;

4) 从任务控制块,根据条件 F_2 判断 A^i 中是否存在重复任务;剔除重复任务,得到任务集 A_2^i;维护条件 F_2,在 B 中记录 A_2^i;

5) 释放同步锁 $m1$;

6) 把 A_2^i 交给消费者执行,如果任务队列满,则需要等待;

7) 休眠一定的时间;}

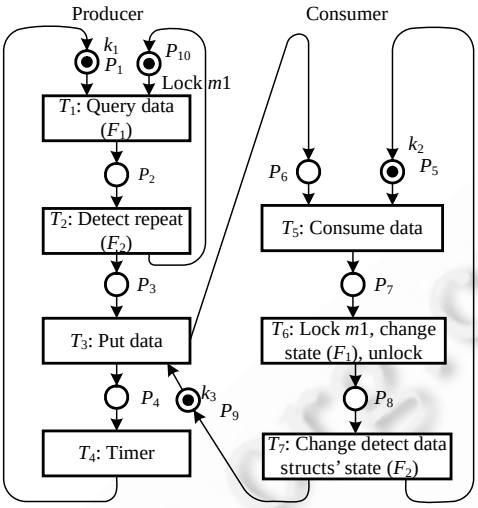


Fig.7 Cross condition redundancy control Petri net

图 7 条件交叉重复控制 Petri 网图

算法 4. 消费者执行函数 *Consumer*().

输入:消费者任务队列;

输出:完成的任务.

1) While (true)

{ 2) 从消费者任务队列取出一个任务 $Task_i$;

3) 完成任务 $Task_i$;

4) 加同步锁 $m1$;

5) 维护条件 F_1,修改 $Task_i$ 对应的数据源为完成状态;

6) 释放同步锁 $m1$;

7) 删除任务控制块 B 中对于 $Task_i$ 的相关记录;}

5.5 CCPN网性质

命题 4. CCPN 网属于并发计算不重复问题.

证明:因为消费者在执行 T_7 以前 a_i 一直属于 B,由于 $a_i \in B$,所有由 T_1 查询得到的重复数据都不会通过 T_2 的 F_2 检测,所以 a_i 不会被再次交给消费者;在执行完 T_7 之后,因为 T_6 已经发生,a_i 已不再属于 A,则 a_i 不能通过 F_1 检测.另外,因为同步锁 $m1$ 的作用,当消费者做 T_6 操作,修改 a_i 的 F_2 检测条件时,生产者线程不处于 P_2,即不存在处于通过 F_1 检测尚未通过 F_2 检测的遗留数据.所以,CCPN 网不会重复计算任务.由定义 6 可知,CCPN 网属于并发计算不重复问题. □

命题 5. CCPN 网是安全有界的.

证明:略,证明过程类似于命题 2. □

命题 6. CCPN 网是活的.

证明:略,证明过程类似于命题 3. □

综上所述,CCPN 属于 CCNRP,具备安全有界和活性,能够正确地解决程序中的 CCRP 问题.使用 PIPE 工具对图 7 中的 CCPN 网进行 Petri 网仿真,可以得到对应的可达标识图和可达标识集.仿真结果与图 6 和表 2 类似,

实验结果可以进一步验证交叉控制算法的相关性质.

5.6 算法比较

单条件并发重复控制算法是一种简单而有效的实现,但是这种方法需要把数据源的状态修改到一个临时状态.在并发环境下,可能其他线程正在使用这个数据源的状态,不允许修改;另外,修改一个全局的数据结构需要较大的访问代价,比如说同步或者是延时较大的远程连接等等.因此,SCPN 网适用于解决数据源的状态允许编辑且数据源访问代价较小的情况.

条件交叉重复控制算法不需要修改数据源为临时状态,比 SCPN 少一次数据源修改,具有通用性,但是增加了新的数据结构,需要更多的内存空间;另外,为了保证生产者处于 P_2 时不能进行 T_6 操作,增加了生产者和消费者之间的同步,这种内部同步开销与全局变量同步相比代价较小.CCPN 的缺点是实现过程相对较为复杂(见表 3).

Table 3 SCPN vs. CCPN

表 3 SCPN 与 CCPN 比较

	Temporary modification of data source	Added data structure	Complexity	Adaptation range
SCPN	Need	No	Simple	Changeable data source and low cost
CCPN	No need	Yes	Relatively complicated	All the range, good generality

6 实验结果与分析讨论

6.1 实验环境

采用 C++语言实现了一个完成解密任务的基于数据源的生产者-消费者程序,程序包含 3 种算法工作模式.其中,无并发重复控制对应于 BDPN,单条件并发重复控制对应于 SCPN,交叉重复控制对应于 CCPN.用 Sqlite3-3.5.6[21]存储数据源,使用 ACE 5.6[16]和 OpenSSL 0.9.8[22]底层库实现线程管理和加解密,条件交叉重复控制算法的控制块基于 Hash 表实现.在一台 Intel 双核 E2160(1.8G),2G 内存,安装 Windows XP SP2 操作系统的 PC 机上完成实验.通过改变待解密文件的大小,构造了两组各 10 个耗时不同的基本任务,见表 4.第 1 组任务文件较大,耗时 $t_b^{(i)}$ 从 3.3s~13.8s 不等,累计耗时 75.732s;第 2 组任务文件较小,耗时从 0.031s~0.156s 不等,累计耗时 1.014s.在每一组中,10 个基本任务通过复制和改变任务标识可多次在数据源中使用,构成任务测试集.考虑到任务耗时具有一定的随机性,做了 10 次实验,取平均值.在后续实验中,除第 6.4 节外均使用第 1 组数据作测试.

Table 4 Basic testing tasks

表 4 基本测试任务

Group 1										
Task No.	1	2	3	4	5	6	7	8	9	10
File size (M)	55.4	157.3	49	137.4	88.9	136	123	142.4	55.8	201.5
Consuming time (s)	3.339 2	11.177 9	2.832 8	9.090 5	5.914 2	8.798 2	7.995 5	9.329 6	3.451 5	13.803
Group 2										
Task No.	1	2	3	4	5	6	7	8	9	10
File size (M)	4.355	4.107	2.412	1.47	0.881	2.976	2.464	7.051	2.731	1.815
Consuming time (s)	0.156	0.125	0.078	0.047	0.031	0.109	0.094	0.234	0.078	0.062

6.2 批量任务产生间隔与并发重复控制

为了方便比较,设置单线程的生产者和单线程的消费者,任务队列足够大,分别完成 10,20,30 个基本任务,生产者线程批量任务的产生间隔即轮询间隔 t_c 从 30s 到 300s,依次增加 30s.然后,分别对无并发重复控制(no control)、单条件并发重复控制(single control)、交叉重复控制(cross control)这 3 种算法进行测试,程序在 3 种算法执行完后都能够正确地把加密文件解密,完成不同任务的耗时情况如图 8 所示.

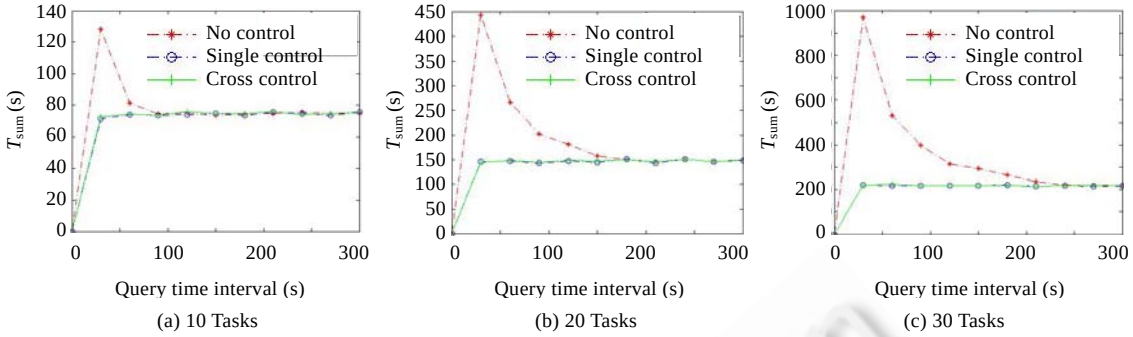


Fig.8 Consumed time of completing different based tasks

图 8 完成不同任务的耗时图

从图 8 可以看出,当批量任务的产生间隔 t_c 小于需要进行并发重复检查的时间 $T^{(i)}(\sum_j t_b^j$,定理 3)时,Single Control 和 Cross Control 算法较 No Control 算法节约比较多的计算时间,并发重复控制算法维持在理想耗时的水平($\sum_j t_b^j$),引入控制所增加的开销要远小于节约的时间,可以忽略不计.Cross Control 新增数据结构和同步带来的开销也很小,二者耗时差别不大.当任务数达到 30,t_c=30s 时,重复控制算法节约的时间(效率提高比)有 3.6 倍之多.当 $t_c>T^{(i)}$时,No Control 算法也会收敛到一个理想耗时的水平.图 9 给出了不同任务情况下各算法在实验中出现的重复计算次数.当 $t_c<T^{(i)}$时,No Control 算法存在明显的重复计算,在图 9(c)中,当 t_c=30s 时重复计算次数多达 94 次;而 Single Control 和 Cross Control 算法没有出现一次重复计算.

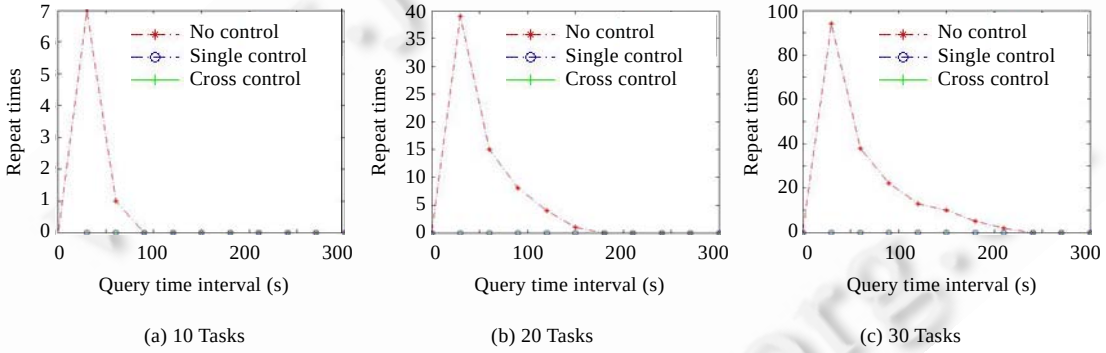


Fig.9 Repeat times of completing different based tasks

图 9 完成不同任务的重复计算次数

6.3 任务强度与并发重复控制

通过变化生产者的目标任务数和轮询间隔时间,可以控制任务强度 S,$S=T(H)/t_c$,任务强度 S 反映了单位时间内的任务量.然后,分别对无并发重复控制、单条件并发重复控制、交叉重复控制这 3 种情况进行不同任务强度下的趋势分析,测试结果如图 10、图 11 所示.

从图 10 可以看出:No Control 除了在强度很小情况下能够保持一个理想的时耗外,其余情况下都随着任务强度呈线性增长.这说明在并发频繁的应用系统中,就更需要作并发重复控制,否则浪费严重;Single Control 和 Cross Control 算法能够一直保持在一个理想的时耗,其大小只与任务的总量有关.图 11 显示,在不同任务强度条件下,No Control 会出现比较明显的重复计算,而 Single Control 和 Cross Control 算法能够有效地解决并发重复计算,没有出现一次重复.

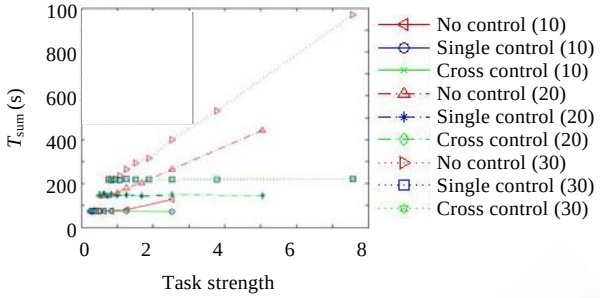


Fig.10 Consumed time under different task strength

图 10 不同任务强度下的耗时图

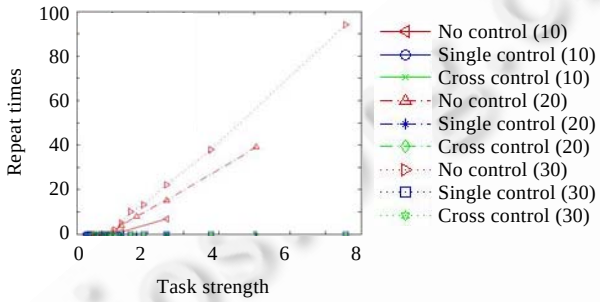


Fig.11 Repeat times under different task strength

图 11 不同任务强度下计算重复次数

6.4 单条件与条件交叉重复控制算法

考虑到第 1 组数据任务耗时较大,不便于分析两种算法的细微差别,我们选用第 2 组数据,固定任务数为 20.首先比较了 Single Control 和 Cross Control 算法在不同生产者和消费者线程数下的时耗,如图 12 所示.其中,Cross Control 算法因为增加了新的数据结构和同步控制,在多数情况下耗时要略高于 Single Control 算法,但是二者相差不太明显.另外,在本次多线程的生产者(k_1>1)和多线程的消费者(k_2>1)的实验中,Single Control 和 Cross Control 算法都没有出现计算重复现象.

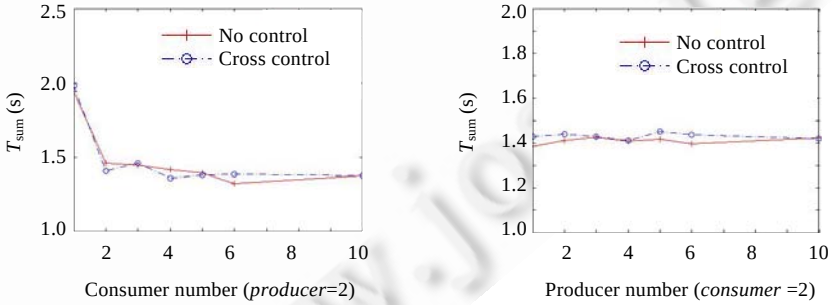


Fig.12 Consumed time under different thread number

图 12 不同线程数下的耗时图

在本实验的默认情况下,数据源的访问速度很快.为了比较两种算法在不同数据源访问代价下的时耗,我们人为地增加了访问数据源的延时,得到两种算法在不同代价下的时耗,如图 13 所示.两种算法受数据源访问代

价的影响都比较大,随着访问延时的增加,任务的完成时间迅速增加;但是,Single Control 算法的增长速度要明显高于 Cross Control 算法,说明条件交叉重复控制算法在数据源访问代价较高的情况下具有一定的优势.

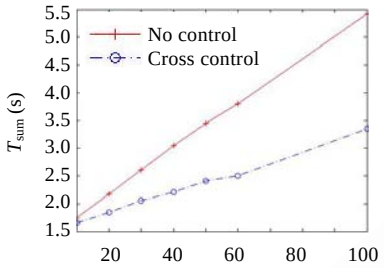


Fig.13 Consumed time under various accessing costs

图 13 不同数据源访问代价下的耗时图

7 相关工作

并发程序、多线程可以提高硬件系统的运行效率,同时也比串行程序的编写要更复杂,更难于掌握.文献[1]指出,我们需要建立一个更确定、便于预测和理解的并发编程模型.文献[18]系统地对并行计算进行了介绍,给出了若干并发程序性能优化的方法和模型,并发计算重复性控制属于并发程序性能优化的范畴.文献[23,24]从静态分析的角度来检测数据竞争,静态分析的方法由于在所有可能的路径上进行最大范围的覆盖,会带来大量误报[4];文献[3,4,25]分别基于维护发生序和锁集合对数据竞争问题进行动态分析,取得了一定的效果.但是数据竞争问题的检测和调试尚未得到根本性解决,仍然是并发计算的一大难题.本文提出的 CCRP 问题是数据竞争问题导致的结果,出现 CCRP 则意味着存在数据竞争,其检测方法可以用于数据竞争问题.

文献[17]对临界区、信号量、管程等并发控制方法进行了介绍,文献[8]使用 Petri 网对多线程应用程序进行建模,详细分析了常见并发控制方法,开发了 C2Petri 工具实现从 C 多线程程序自动生产 Petri 网;文献[6]证明了给定结构复杂的 Petri 网都可通过 S-网的同步合成运算而得到,文献[9]则给出如何保持同步合成中操作活性和无死锁的方法;Petri 网进行并发程序相关研究的还有文献[7,10,11]等.本文参考以上文献的 Petri 网描述方式和方法,对于并发计算重复问题以及控制机制作了分析.

面包师算法[17]通过比较客户的排号,只允许大于最大已服务的排号被调度,从而避免了对同一个客户进行多次服务.面包师算法解决的是 CCRP 的一个特例,适合拥有唯一有序的客户名称的情况.本文对并发程序因线程间并发无序导致的重复计算现象进行了系统研究,给出了相应控制算法.对于维护数据的同步和一致性、避免死锁的并发控制研究比较多,例如文献[9,26,27]等,并发计算重复控制比同步和一致性的控制更上层一些,因为并发计算重复控制机制仍然要用到同步.

8 总 结

充分发挥多核、高性能计算机硬件的效率,需要多进程、多线程的并发软件.本文研究了并发程序中的重复计算问题,该问题源于数据竞争,是由于并发线程发生的顺序不确定和错误同步所造成的;该问题不一定导致程序结果错误,但会影响系统的性能.CRCM 是一类新的并发控制机制,它可以解决 CCRP.我们设计、比较了单条件和条件交叉两种并发重复控制算法,基于 Petri 网作了相应的并发特性分析.实验结果表明:CRCM 可以明显地改善系统性能去除重复计算;单条件和条件交叉控制算法在数据源访问快的时候性能相当,但在数据源访问代价较大的情况下,条件交叉控制算法具有一定的优势.并发计算重复问题的研究有利于数据竞争问题的检测,有助于正确编写并发程序,对于提高并发程序的可控性和计算性能具有一定的理论和现实意义.下一步我们将继续研究 CCRP 的自动检测与转化、多机并发 CRCM 的实现、基于 CCRP 的数据竞争测试等相关课题.

致谢 感谢匿名评阅人对本文工作提出的宝贵意见以及项目组张玉洁、商彦磊老师和陈声扬的帮助.

References:

[1] Lee EA. The problem with threads. Computer, 2006,39(5):33−42. [doi: 10.1109/MC.2006.180]

[2] Zhang LB, Zhang FX, Wu SG, Chen YY. A lockset-based dynamic data race detection approach. Chinese Journal of Computers, 2003,26(10):1217−1223 (in Chinese with English abstract).

[3] Fu H, Cai M, Dong JX, Jin X, Gong Y. Enhanced data race detection approach based on lockset algorithm. Journal of Zhejiang University (Engineering Science), 2009,43(2):328−333 (in Chinese with English abstract).

[4] Marino D, Musuvathi M, Narayanasamy S. LiteRace: Effective sampling for lightweight data-race detection. ACM SIGPLAN Notices, 2009,44(6):134−143. [doi: 10.1145/1542476.1542491]

[5] Li XO, Medina JM, Chapa SV. Applying Petri nets in active database systems. IEEE Trans. on Systems, Man and Cybernetics, Part C: Applications and Reviews, 2007,37(4):482−493. [doi: 10.1109/TSMCC.2007.897329]

[6] Zeng QT. Behavior descriptions of structure-complex Petri nets based on synchronous composition. Journal of Software, 2004, 15(3):327−337. http://www.jos.org.cn/1000-9825/15/327.htm

[7] Yamaguchi H, El-Fakih K, Bochmann GV, Higashino T. Deriving protocol specifications from service specifications written as predicate/transition-nets. Computer Networks, 2007,51(1):258−284. [doi: 10.1016/j.comnet.2006.03.011]

[8] Kavi KM, Moshtaghi A, Chen DJ. Modeling multithreaded applications using Petri nets. Int'l Journal of Parallel Programming, 2002,30(5):353−371. [doi: 10.1023/A:1019917329895]

[9] Pu F, Lu WM. Preservation of liveness and deadlock-freeness in synchronous synthesis of Petri net systems. Journal of Software, 2003,14(12):1977−1988 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1977.htm

[10] Llorens M, Oliver J. Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets. IEEE Trans. on Computers, 2004,53(9):1147−1158. [doi: 10.1109/TC.2004.66]

[11] Ding ZJ, Jiang CJ. Verification of concurrent programs by temporal Petri nets. Chinese Journal of Computers, 2002,25(5):467−475 (in Chinese with English abstract).

[12] Yuan CY. Principles of Petri Nets. Beijing: Electronic Industrial Press, 2005 (in Chinese).

[13] Wehler J. Petri nets. 1999. http://www.pst.ifi.lmu.de/personen/wehler/

[14] Reisig W. Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets. Berlin: Springer-Verlag, 1998.

[15] Home of fundamental modeling concepts. http://www.fmc-modeling.org/

[16] Douglas CS, Stephen DH. C++ Network Programming, Vol.1. Massachusetts: Addison-Wesley Press, 2001.

[17] Silberschatz A, Galvin PB, Gagne G. Operating System Concepts. 7th ed., New York: John Wiley & Sons Press, 2004.

[18] Gramma A, Gupta A, Karypis G, Kumar V. Introduction to Parallel Computing. 2nd ed., Massachusetts: Addison-Wesley Press, 2003.

[19] Bonet P, Llado CM, Puijaner R, Knottenbelt WJ. PIPE v2.5: A Petri net tool for performance modeling. In: Proc. of the 23rd Latin American Conf. on Informatics (CLEI 2007). San Jose, 2007. http://pipe2.sourceforge.net

[20] Coffman EG, Elphick MJ, Shoshani A. System deadlocks. ACM Computing Surveys, 1971,3(2):67−78. [doi: 10.1145/356586.356588]

[21] Home of SQLite. http://www.sqlite.org

[22] Home of OpenSSL. http://www.openssl.org

[23] Voung JW, Jhala R, Lerner S. RELAY: Static race detection on millions of lines of code. In: Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering (ESEC-FSE 2007). Dubrovnik: ACM Press, 2007. 205−214. http://dl.acm.org/citation.cfm?id=1287654

[24] Boyapati C, Lee R, Rinard M. Ownership types for safe programming: Preventing data races and deadlocks. In: Proc. of the Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). Seattle, 2002. http://dl.acm.org/citation.cfm?id=582440 [doi: 10.1145/582419.582440]

[25] Prvulovic M. CORD: Cost-Effective (and nearly overhead-free) order-recording and data race detection. In: Proc. of the 12th Int'l Symp. on High-Performance Computer Architecture. Austin, 2006. http://doi.ieeecomputersociety.org/10.1109/HPCA.2006.1598132 [doi: 10.1109/HPCA.2006.1598132]

[26] Palm R. Synchronization of decentralized multiple-model systems by market-based optimization. IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics, 2004,34(1):665−672. [doi: 10.1109/TSMCB.2002.806488]

[27] Raghunathan S. Extending inter-process synchronization with robust mutex and variants in condition wait. In: Proc. of the 2008 14th IEEE Int'l Conf. on Parallel and Distributed Systems (ICPADS 2008). Melbourne: IEEE Press, 2008. 121−128. [doi: 10.1109/ICPADS.2008.98]

附中文参考文献:

[2] 章隆兵,张福新,吴少刚,陈意云.基于锁集合的动态数据竞争检测方法.计算机学报,2003,26(10):1217−1223.

[3] 富浩,蔡铭,董金祥,金星,龚宜.基于锁集合算法的增强型数据竞争检测方法.浙江大学学报(工学版),2009,43(2):328−333.

[9] 蒲飞,陆维明.同步合成 Petri 网系统活性与无死锁性的保持性.软件学报,2003,14(12):1977−1988. http://www.jos.org.cn/1000-9825/14/1977.htm

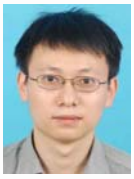
[11] 丁志军,蒋昌俊.并发程序验证的时序 Petri 网方法.计算机学报,2002,25(5):467−475.

[12] 袁崇义.Petri 网原理与应用.北京:电子工业出版社,2005.

何倩(1979－),男,湖南安仁县人,博士生,CCF 学生会员,主要研究领域为分布式计算,Petri 网应用,P2P 优化.

陈俊亮(1933－),男,教授,博士生导师,中国科学院院士,中国工程院院士,CCF 高级会员,主要研究领域为网络智能化.

孟祥武(1966－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为通信软件,下一代互联网,语义 Web.



沈筱彦(1981－),男,博士,主要研究领域为智能搜索.