CASE-DDB的设计和不确定性问题的解决*

桑 楠** 龚天富 陈文宇
(电子科技大学计算机系 成都 610054)

【摘要】 分布式程序执行行为的不确定性导致其调试工作复杂化,以至于无法用传统的调试工具进行调试。CASE-DDB是一种基于事件的调试工具,用于调试用分布式 C++语言编写的分布式应用程序。文中介绍了 CASE-DDB的设计思想和基本结构,提出了一种调试执行行为不确定的分布式应用程序的方法。

关 键 词 分布式; 不确定性; 事件; 调试工具; 再现
中图分类号 TP316.2

随着计算机系统的迅速发展,计算机硬件的价格变得越来越低。利用多台计算机联网组成一个分布式系统,解决日益复杂的问题,已变得越来越实际。然而,由于分布式程序并发执行,因而程序的每次执行过程基本上无法重复出现。这种现象称为执行行为的不确定性,来源于两个因素:分布式语言中选择会合机制等不确定因素和分布式系统中各道进程执行次序不确定(受时间影响)。

不确定性使得分布式调试工作复杂化。传统的顺序调试工具无法适应分布式程序的调试要求,迫切需要研制专门的分布式调试工具。为适应这种需要,国外已设计出多种分布式调试模型,各自针对一些具体的应用对象,但在国内尚未见到有关报道。现有分布式调试模型可归成三类:静态分析方法、传统的并行调试和基于事件的调试[1]。前两种方法对依赖时间的错误很难解决,而分布式系统,特别是实时系统,有许多与时间相关的错误。因此,彻底解决分布式调试问题,一般趋向于采用基于事件的调试。

基于事件的调试方法在具体设计时,根据解决问题的重点,分成两大类。一类依所收集的信息量,以及是否可循环调试来分,有三种方法:基于执行期间获取的程序状态之瞬象——BA方法;基于再现程序执行的 BUGNET方法和 Instant Replay方法[2,3]。相比之下,后两种方法较好,但又各有优缺点:前者允许再现、循环调试、有全局时钟、允许对程序的某一部分进行调试等,但它要求系统的通信不频繁,且占用的存贮空间极大,很难检查进程交互作用的全局影响;后者对通信要求不高且不需要全局时钟,但它未保存消息的具体内容,只能再现整个系统的执行过程。另一类根据对实现分布式软件的语言的依赖情况,分成基于实现和基于语言两种方法[4,5]。前者直接修改分布式语言的成分,使用它开发出的分布式程序可确定性调试,这种方法依赖于语言的编译器、所使用的操作系统,以及运行时的系统环境来收集调试信息,控制程序的执行。后者则直接利用开发分布式程序的源语言 L,将源程序 P改写成一个执行结果等同的程序 $\dot{P}$,在 $\dot{P}$的执行过程中收集有关的调试信息,用于程序的确定性再现,这种方法操作简便,但"探针"影响较大。

本文描述了一个用基于事件和语言的策略设计的分布式调式工具(简称 CASE-DDB)的设计

① 1996年 12月 5日收稿,1997年 2月 26日修改定稿
* 电子部"八五"重点科研项目
** 男 32岁 硕士 讲师

原理,阐述了不确定性问题的处理过程。它用于调试用分布式 C++ 语言编写的分布式程序。

1 分布式 C++ 语言简介

分布式 C++ 语言是在面向对象的程序设计语言 C++ 的基础上,扩展有关分布式处理的机制而成的[6]。分布式 C++ 中,增加了进程类型、进程组成分、进程创建和撤消机制,还提供了入口调用、接收语句、选择语句等消息通信机制。使用该语言,可以较方便的描述分布式应用问题。

2 设计思想

基于事件的调试策略的基本思想是:将分布式系统的一个状态变化看成是一个事件发生,整个分布式程序的执行过程看成若干个并行执行的事件序列。分布式程序的调试则是调试每个监控事件的状态变化,从而掌握分布式程序的执行过程和执行结果。按这种思想,CASE-DDB的设计分成五个部分:事件定义、收集运行信息、信息整理、运行再现,以及调试信息的体现和图形用户界面。事件定义用于设置调试系统所用的监控原语集,图形用户界面是为用户提供一个良好的使用环境,其他各部分则反映了整个调试过程。

CASE-DDB的设计中,将分布式程序执行过程中发生的所有事件分成两类:全局事件和局部事件。前者是与分布式系统状态有关,可能导致不确定性行为产生,如进程创建、撤消、入口调用等;后者局限于一道进程内,本身不影响其他进程的行为,如变量赋值、函数调用等。CASEA-DDB将调试过程分成两个阶段。第一阶段收集涉及分布式并发执行的运行信息,即监控分布式程序中的所有全局事件,收集有关信息并初步整理,保证调试时可确定性再现此执行过程。在此阶段内不做任何调试工作,也不收集任何局部事件信息,以减少调试对用户原程序执行的干扰。第二阶段进行程序的再现调试,它根据用户的调试要求,对收集的事件信息进行加工处理,用于在模拟环境下确定性再现分布式程序的原始执行过程,并将有关调试信息告知用户。在此阶段,用户可在程序中设置断点和局部事件,而其执行不再受分布式系统的影响,相当于顺序程序调试。

3 不确定性问题的解决

对一道进程的执行而言,只要确定了进程创建时间、由其他进程传递来的信息和获取时间、进程撤消时间等全局事件信息,整个进程的执行过程和结果就确定了。一旦各道进程的执行过程和结果都确定,随之而得的若干并发事件序列(依时间的偏序)就确定了整个分布式程序的执行行为。因此,不确定性问题解决的关键是全局事件的设置和全局事件信息是否完整反映分布式程序的进行。

按上述原理,CASE-DDB对分布式程序执行行为不确定性的解决方法分成以下几方面:

1) 根据分布式 C++ 语言的特点,将与进程有关的操作均定义成全局事件并实现成相应的监控原语,如 CreatProcArricval、StopProc、Entry Arrival、Entry Finish、Accepi Arriival、Accept Finish 等。

2) 改写分布式程序 P,在与全局事件有关的操作发生前后加入监控原语,收集该操作对分布式系统状态的影响。如将入口调用语句

〈进程名〉.〈调用名〉(〈实参表〉)

改写成

〈监控进程〉. Entry Arrival(〈发生时间〉,〈所在结点〉);

〈进程名〉.〈调用名〉(〈实参表〉);

〈监控进程〉. EntryFinish(〈发生时间〉,〈信息〉,〈所在结点〉);

改写后的程序 P′的执行结果与 P相同。

3) 一旦程序执行有错,整个分布式程序自动停止运行。

4) 将收集的事件信息按事件发生的时间偏序进行整理,获得各结点上事件发生序列,存放在相应的事件信息库 EventDB-NodeNo中,格式为:

〈发生时间〉〈信息数目〉〈信息表〉

5) 重新改写 P,删除 P中所有与全局事件有关的定义和语句,代以读入事件信息的定义和原语,将 P改写成一个顺序程序 P″。如将进程说明

Process〈进程类名〉〈进程名〉;

改写成对象说明

〈C++ 类名〉〈对象名〉;

当然,P中的所有进程类必须先改写成相应的 C++ 类。

6) 在模拟环境下,再现 P的执行过程: P″执行到全局事件发生处时,以相应的事件信息作为输入数据,驱动程序向前执行。如前面的入口调用改写成输入语句

GetMessage(〈事件信息库名〉,〈参数表〉);

于是,程序 P″的执行和调试与一般顺序程序相同,不再存在不确定性行为。

【例 1】(1)源程序 P

```
Process Productor {
    private:
      void body ( );
    …
};
…
Productor:: body ( )
  { do
      {
        ch= getchar ( );
        B. put(ch);
      }
    while (ch = = '\' );
  }
Process Buffer {
    entry;
      void put(char ch);
      …
  };
…
main( )
  {
    Process Buffer B(node1);
    Process Productor P1(node2);
…
```

(2)等效的执行程序 P′

```
Process Productor {
    private;
      void body ( );
    …
  };
…
Productor:: body( )
  { do
    { ch= getcharo ( );
      M. EntryArrival(… );
      B. put(ch);
      M. EntryFinish(… );
    } while (ch= = '\' );
  }
Process Buffer {
    entry:
      void put(char ch);
    …
  };
…
main ( )
  { Process Monitor M;
    M. CreatProcArrival(… );
    Process Buffer B( node1);
    M. CreatProcArrical(… );
    Process Productor P1(node2);
    …
```

```
}
```

(3) 调试进程 P1的顺序程序 P1"

```
class Productor
  {
    privater
      void body ( )
    ...
  };
Productor::: Productor ( )
  { body ( )
  }
 ...
}
Productor:: body ( )
  { do
    { D. GetMessage(node2,& ch);
    } while (ch= = '\');
  }
...
main( )
  { class Debugger D;
    D. GetMessage(… );
    class Productor P1;
    ...
```

一般来说,即使有全局时钟,分布式程序的执行也不可能象顺序程序那样暂停,否则全局状态必然发生变化。CASE-DDB的设计不依赖于系统全局时钟,各事件发生的时间相对于一个事件。这样,各结点上的事件偏序容易确定。为协调分布式系统中不同结点上的事件序列,只需获取各结点上的起始执行时间,再依消息发送、接收次序、父子进程关系等确定进程间的事件发生次序。

此外,为减少监控原语对用户原程序执行的影响,全局事件信息的收集过程尽量简略。全局事件信息中,发生地点等作为静态信息保存,运行时只需记录发生时间和接收的信息,其他的可在再现时自动产生。

4 CASE-DDB的实现

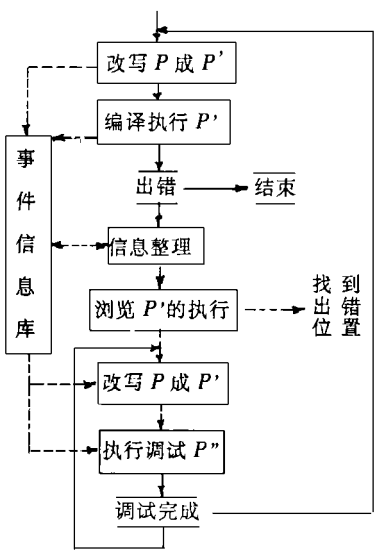

图 1 分布式程序 P的调试过程

CASE-DDB的设计分成五个部分。在实现时,除事件定义作成一个监控原语库外,其他各部分用四个相关模块完成。

1) 收集运行信息 信息的收集分成三部分: 静态信息、全局事件信息和局部事件信息。静态信息在程序 P'的编译过程中获得,全局信息大多在程序 P'的运行过程中收集,其余的在程序再现时(运行 P")完成。

2) 信息整理 分布式程序执行产生大量信息,其中很大一部分与出现的错识无关。为此,不但在收集运行信息时要有所选择,而且需要按再现执行的实际要求对信息库中的信息进行整理,获取所需要的各类静态信息和动态信息。同时,也为信息体现作准备。

3) 运行再现 再现分成再现整个执行过程、一个结点上的执行、一道进程、一个函数等多种方式。在再现过程中,除可体现信息库中已有的分布式程序信息外,还可根据不同的要求,进一步收集全局和局部事件信息。整个再现过程是在模拟运行环境控制下进行的。

模拟运行环境 整个再现过程的控制器。它依再现方式,由

信息库中找出与当前再现程序段有关的全局事件序列,同时,将当前系统状态告知用户。

调试手段　同顺序程序的调试一样,再现调试分成单步执行、依断点执行、浏览等多种方式。事件是再现执行的基本单元,执行过程就是事件发生的过程(依时间)。每执行一步(一个或多个事件),都要输出有关的事件信息。

4) 调试界面　这是 CASE-DDB的控制模块。它一方面接收用户的各种预置条件,用于修改原程序 P;另一方面将系统的运行情况和相关信息体现给用户,以便用户找出 P中的错误。

静态信息　在原程序 P编译后,用户就可查询诸如进程名、进程原形、进程交互情况、对象继承性关系等信息。

事件信息　事件发生前后系统状态变化情况。包括发生的时间、地点、导致哪些被监控变量发生变化等等。若同时有多个事件发生,则相应地要体现多项信息(在多个窗口内)。

时间进程图　为方便用户理解多道进程的并发执行过程,使用时间进程图来体现各道进程的活跃周期,即依时间排序的进程执行情况。

5 结 束 语

我们已描述了 CASE-DDB的设计思想和实现,其模型已用 C++ 语言基本实现,在以 SUN 工作站为结点机的分布式网络上运行通过。虽然它是为分布式 C++ 语言设计的,但由于是用面向对象程序设计方法完成的,所以很容易进行移植,使其适应其他由 C++ 扩展而来的分布式语言。

分布式调试工具尚属于研究阶段,国外的研究也大多属于模型,因此有待于继续进行下去。

参 考 文 献

1　Mcdowell C E, Helmbold D P. Debugging concurrent programs, ACM Computing Surveys, 1989, 21(4): 600~622

2　Leblanc T J, Mellor-Crummey J M. Debugging parallel programs with instant replay. IEEE Trans on Computers, 1987, 36(4): 471~482

3　Jones S T, Barkan R H, Wittie L D. BUGNET: A real time distributed debugging system. 1987 IEEE Real Time System Symposium, 1987: 56~55

4　Baiard E, De Francesco N, Matteoli E. Development of a debugger for a concurrent language. ACM Sigplan Notices, 1983, 18(8): 98~106

5　Goldszmidt G S, Katz S, Yemini S. Interactive blockbox debugging for concuttent languages. Proc of the ACM SIGPLAN/SIGOPS Workshop on Parallel& Distribuied Debugging, 1988: 271~282

6　分布式 C++ 语言(工作报告).南京大学计算机系, 1993

7　Tokuda H, Kotera M. A real-time tool set for the ARTS kernel. Proc of the 9th IEEE Real-Time System Symposium, 1988: 289~299

Design of CASE-DDB and Resolution of Nondetermination

Sang Nan　　Gong Tianfu　　Chen Wenyu

(Dept. of Computer, UEST of China　Chengdu　610054)

Abstract　It is very difficult to debug distributed program because of the nondetermiantion of distributed program execution. So, it is almost impossible to debug distributed program with traditional tools. CASE-DDB is an event-based debugger for debugging distributed program written in distributed C++. This paper introduces its rationale and realization, and provides a method to debug nondeterminative distributed program.

Key words　distribute;　nondetermination;　event;　debugger;　replay

编辑　黄　辛