# 开题报告

**一、课题来源、目的、意义和要求**

现今，随着计算机技术的发展和用户对软件系统性能要求的提高，并发技术得到了广泛的应用。虽然并发程序的使用提高了系统的性能，但也给程序员带来了开发和调试的困难。并发程序的不确定性、错误的不可再现性，使得传统的顺序程序调试方法不再可用。

早期的顺序程序都是一条指令接一条指令地顺序执行。然而在客观世界的问题空间中，存在着逻辑上同时发生的活动。为了描述和处理同时发生的活动，必须使用并发机制。并发程序的设计比顺序程序困难得多，程序员需要处理好各个进程之间的同步和通信关系。这些问题处理得不恰当，会带来并发程序所特有的一些错误（如死锁）。即便是各个进程之间的同步和通信关系都是正确的，排除出现在单个进程中的逻辑错误也是很困难的。同时，并发程序的执行取决于程序中各个独立事件发生的先后顺序，是动态的、不确定的。这种不确定性使得程序中的错误不能够按照程序员安排的环境出现，给程序的动态调试带来很大困难。对于这类问题，传统的顺序程序调试技术显得有些无能为力。在调试和测试程序时，程序员通常采用循环调试的方法，即以相同的输入反复执行程序，借助断点和观测点观察程序的运行状态，从而找出问题的所在。使用循环调试方法的前提是：对于同样的输入，程序的运行结果（输出结果和执行路径）都相同，即程序的运行是确定的。并发程序的执行由于进程的并发性、进程间的通信与同步等原因具有不确定性，即在相同的输入下，程序每次正常执行的结果都可能不同。这种不确定性直接导致了错误的不可再现性：如果程序的某次调试运行出现了一个错误，在程序的后续多次运行中，这个错误可能不再出现，如果这个错误出现的几率很小，那么程序员可能始终无法发现这个错误或查出导致这个错误的原因。

为了能够继续使用循环调试的方法调试并发程序，必须能够使并发程序的执行确定化。我们采用的方法分为两个阶段：第一个阶段是记录阶段，记录并发程序的原始运行并且记录必要的执行信息；第二个阶段是重放阶段，在保证输入给程序的信息与追踪阶段相同的前提下，根据记录阶段记录的信息控制程序的执行，重现原始的执行状态。在能够重现并发程序的某次执行的条件下，循环调试方法将可以用于并发程序的调试，从而降低并发程序测试和调试的难度。

基于以上提出的追踪重放并发程序执行的必要性和可行性，我们研制和开发了基于重放的并发程序调试工具，该工具用于对 Linux 下编写的并发程序的调试。

1. **课题综述/国内外发展状况**

关于并发程序的调试方法，国内外已有研究者做过一些工作。陈振强通过分析并发程序的行为，构造程序流图、检测同步错误和数据竞争。张斌在分析并发程序依赖关系的基础上，提取出任务间的通信关系，建立并图形化显示通信关系模型。以上两者均是通过静态分析程序的方法进行测试和调试，不会引入探针效应，但是该类方法随着程序规模的增大，算法的复杂度较高，而且无法了解程序的执行细节。Baiardi 设计的 Debugger中，编程人员可以通过行为规范描述，预先提供一个 ECSP 程序的期望行为；在执行中，判断运行时的行为是否符合预先提供的行为规范，若不相同，根据不符的类型，执行相应的预定义动作，分析程序执行结果，因此它的主要目标在于程序的验证。Jason 提出的 CBUG，它在源代码中植入调试钩子，然后将源代码和调试器编译，一边执行一边调试，然而由于程序执行的不确定性，单次执行时错误是随机的，所以这种一步式的调试方法效果不是很好。

在并发程序的依赖性分析上，徐勇在毕业论文中简单讨论了Ada并发程序的依赖关系,但没有考虑由同步引起的依赖关系及可静态检测的死锁问题;试图用图的可达性来分析并发程序中语句的依赖关系,但其所构造的并发程序间的依赖关系是不可传递的,使得结果不够精确;Krinke[9]通过附加的判断条件,构造了比较精确的、不包含同步的并发程序的依赖图,由于同步会引起程序执行顺序的变化,而并发程序中同步是广泛存在的,该算法无法广泛应用;Nanda[10]将Krinke方法推广到包含同步的并发程序中,这两种方法本质上采用了类似Petri网可达性分析的方法,最坏情况下时间复杂度是指数级。

1. **总体方案设计**

之前其他人实现的基于重放的并发程序调试，都是在进程的级别上，涉及到了进程间的通信。而这次我要实现的是基于线程的多线程环境下的时序分析。

此系统预期设计出这样一个程序：它可以读取一个文件，获得某个程序的全局变量的相关信息，如：该全局变量在该程序的第几行，做了读或者写的操作。通过这个文件的信息，就可以对该程序的源代码进行插装。将对变量的读或者写，改为一个模块的入口。该模块的主要作用就是记录当前是哪个进程，对哪个变量做了什么样的操作，并将这些信息输出成文档。之后，通过这个文档，就可以设计一个重放器，根据文档的信息，再一次按之前的执行路径执行，这样就可以让程序员通过这些执行信息来调试程序。

该系统预计有三个模块：

* 1. 模块一：读取记录变量信息的文件，并对源代码进行插装和替换，将对变量的操作改为模块二的入口。
  2. 模块二：记录以下信息：

1. 哪一个线程进入到该模块；
2. 对哪个变量进行了操作；
3. 对变量进行了什么操作；
4. 将记录生成文档，输出；
   1. 模块三：根据模块二的文档，重放线程的执行。

其中模块二的设计要注意以下几点：

1、模块的功能要足够简单，执行时间要短，避免产生“探针效应”；

2、在对变量操作改为模块入口的操作中，要注意对模块入口的前后上锁，避免在模块中执行的时候发生抢占，影响最终结果；

1. **设计环境**

软件环境

操作系统：Ubuntu

硬件环境

处理器：i5-2410m 2.3GHz

内存：4GB

硬盘：500GGB

1. **进度计划**

1、2014.1.1~2014.1.13 开始收集资料，了解课题背景，完成论文翻译和开题报告。

2、2014.1.13~2014.3.13 着手开始完成系统的需求分析和概要设计。3、2014.3.13~2014.5.15 开始进行系统的程序设计，并且完成测试工作。

4、2014.5.15~2014.5.25 撰写论文

5、2014.5.25~2014.5.30 整理毕业设计所有资料，完成论文。

1. **参考文献**

[1] Moc J, carr D A. Understanding Distributed Systems via Execution Trace Data. Proceedings of IWPC. 2001: 60-67.

[2] C.E Mcdowell, D.P.Helmbold. Debugging Concurrent Programs. ACM Computing Surveys, 1989, 21(4): 593-622.

[3] E.H.Paik, Y.S.Chung, B.S.Lee, Chae-Woo Yoo. A Concurrent Program Debugging Environment Using Real-time

Replay. International Conference on Parallel and Distributed Systems. 1997: 460-465.

[4] Fubin Zuo, Baowen Xu. Event-based Visualization Debugging on Concurrent Program. Proceedings of the 16th IASTED International Conference on Parallel and Distributed Computing and Systems Cambridge, MA, USA. Nov 2004, 827-832.

[5] Zhenqiang Chen, Baowen Xu, Hongji Yang, Jianjun Zhao. Static Dependency Analysis for Concurrent Ada 95 Programs. In Proceedings of AdaEurope 2002: 219-230.

[6] Zhenqiang Chen, Baowen Xu, Huiming Yu. Detecting Dead Statements for Concurrent Programs. Proceedings of the SCAM2001, Florence, Italy, IEEE Computer Society Press, 2001, 65-72.

[7] 陈振强,徐宝文. 一种并发程序依赖性分析方法. 计算机研究与发展. 2002, 39(2): 159-164.

[8] 张斌,卢虹,陈振强,徐宝文,黄曙萍. 并发程序中通信关系的图形化描述. 软件学报. 2002,13(13): 286-290.

[9] F. Baiardi, N. De Francesco, G. Vagliani. Development of a Debugger for a Concurrent Language. IEEE Trans. On Software Engineering, 1986, 12(4): 547-553.

[10] Jason Gait. A Debugger for Concurrent Programs. Software Practice and Experience, 1985, 15(6): 539-554.

[11] 张钦伍. 并行调试环境中的追踪和重放. 国防科技大学学报. 1998, 20(6): 33-36.

[12] Lee B, Park T, Yeom H Y, etal. An efficient algorithm for causal message logging. Proceedings of the 17th Symposium on Reliable Distributed Systems. 1998, 19-25.

[13] Zambonelli F. An Efficient Logging Algorithm for Incremental Replay of MessagePassing Applications. 13th international Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing. 1999,392-400.

[14] 章隆兵. 软件分布式共享存储系统的编程支持环境研究: [学位论文]. 合肥: 中 学技术大学计算机系,2002.

[15] Ronsse M, De Bosschere K D. RecPlay: A Fully Integrated Practical Record/ReplaySystem. ACM Transactions oncomputer Systems. 1999, 17(2): 133-152.

[16] Ronsse M et al. Execution replay and debugging Proceedings of the Fourth InternationalWorkshop on Automated Debugging (AADEBUG2000). 2000, 5-18.

[17] Weiser M. Program Slicing. IEEE Trans. Software Engineering, 1984, 16(5): 498-509.

[18] Beizer B. Software Testing Techniques, 2nd ed., New York: Van Nostrand Reinhold, 1992.

[19] 卢虹. pSos 并发软件的分析与理解技术研究: [学位论文]. 南京: 东南大学计算系, 2002.

[20] Chen Z, Xu B. Slicing Concurrent Java Programs, ACM SIGPLAN Notices, 2001, 36(4):41-47.

[21] Chen Z, Xu B, Yang, etal. Concurrent Ada Dead Statements Detection. Information and Software Technology. 2002,44(13): 733-741.

[22] 陈振强,徐宝文. 一种并发程序依赖性分析方法. 计算机研究与发展. 2002, 39(2):159-164.

[23] K.C.Tai, R.H Carver, E.E.Obaid. Debugging Concurrent Ada Programs by DeterministicExecution. IEEE Trans. onSoftware Engineering, 1991, 17(1): 45-63.

[24] Michel Raynal. About Logical Clocks for Distributed Systems. ACM Operating SystemsReview, 1992, 26(1): 41-48.

[25] Michel Raynal, Mukesh Singhal. Logical Clocks: Capturing Causality in DistributedSystems. IEEE Computer, Feb1996, 49-56.

[26] Friedemann Mattern. Virtual Time and Global States of Distributed System. Parallel and distributed algorithms,North-Holland: Elsevier Science Publisher B.V. 1989:215-226.