第36卷 第14期 计 算 机 工 程 2010年7月
Vol.36 No.14 Computer Engineering July 2010
·软件技术与数据库· 文章编号：1000—3428(2010)14—0054—02 文献标识码：A 中图分类号：TP311.1

基于关键谓词的程序错误定位方法

辛 良，姜淑娟
(中国矿业大学计算机科学与技术学院，徐州 221116)

摘 要：将程序切片技术应用于程序错误定位可以大量减少需要测试的语句数。提出一种基于关键谓词的程序错误定位方法，从程序中找出能影响输出结果的关键谓词，对该谓词和错误输出语句进行数据切片，并引入代码优先技术。该方法考虑了数据依赖和控制依赖，能实现准确快速的错误定位。

关键词：错误定位；程序切片；关键谓词

Program Fault Location Method Based on Critical Predicate

XIN Liang, JIANG Shu-juan
(School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116)

【Abstract】 The statements need to be tested can be decreased by applying the program slicing technology to locate program fault. This paper presents a program fault location method based on critical predicate. The method identifies the critical predicate that can affect the output result, adopts data slicing according to the predicate and fault output statement and introduces the code prioritization technology. It uses the information of data dependences and control dependences and can locate the fault precisely and quickly.

【Key words】 fault location; program slicing; critical predicate

1 概述

在传统程序调试中，通常由程序员先设置断点，然后对失败的输入进行重运行，检查程序的运行状态并找到错误原因，其过程费时费力。程序切片的概念由文献[1]首次提出，它从源程序中抽取对程序中兴趣点上的特定变量有影响的语句和谓词，组成新的程序。将切片技术应用于程序分析和理解技术中，能大量减少需要考虑的语句。而将程序切片应用于错误定位时，必须关注2个方面的内容：(1)错误出现在切片中的几率；(2)切片的大小。通过分析多种切片，笔者发现找到影响执行路径的关键谓词是错误定位的重点所在。因此，本文提出一种依据关键谓词进行程序错误定位的技术。它先通过逐步分析错误输出结果所执行路径上的谓词确认关键谓词，然后对关键谓词和错误语句的数据切片进行分析，并根据代码优先技术获得更准确和精简的语句集。

2 相关概念

2.1 程序切片

程序切片是一种程序分解技术，它是指通过依赖关系潜在影响某一语句 n 的某一变量 v 的语句集，并把$<n, v>$称为切片准则[2]。

定义1 设 s 是程序流图CFG中的任一节点：

(1)定义集 $Def(s)=\{x/x$ 是语句 s 中值被改变了的变量$\}$；

(2)引用集 $Ref(s)=\{x/x$ 是语句 s 中引用的变量$\}$。

定义2(数据依赖) 如果节点 n、m 满足以下2个条件，则称 n 数据依赖于 m：

(1)如果存在一个变量 v 满足 $v\in Def(m)\cap Ref(n)$；

(2)G 上存在一条由节点 m 到节点 n 的路径 p，对于路径上的其他节点 $m'\in p-\{m, n\}$，有 $v|Def(m')$。

定义3(控制依赖) 如果节点 n、m 满足如下条件，则称 n 控制依赖于 m：

(1)G 上存在一条由节点 m 到节点 n 的路径 p，对于路径上的其他节点 $m'\in p-\{m, n\}$，n 是 m'的后必经点；

(2)n 不是 m 的后必经点。

程序切片不仅与兴趣点定义和使用的变量有关，还与影响该变量值的语句和谓词以及受该变量值影响的语句和谓词有关。在程序切片过程中，采用合适的结构来表示语句间的依赖关系，寻找与兴趣点具有直接或间接数据依赖和控制依赖的节点是生成切片的重点。切片算法基本过程可以概括为：先寻找语句 s 的变量 v 直接数据依赖或控制依赖的节点；然后寻找这些新节点直接数据依赖或控制依赖的节点；重复上述过程，直到没有新节点加入为止；最后将这些节点按源程序的语句顺序排列，即为程序 P 关于语句 s 的切片 S。

动态切片比静态切片小很多，不同动态切片的侧重点有很大差异。文献[3]对数据切片、完全切片和相关切片3种动态切片方法进行实验测试，比较并分析了各种切片的大小和包含错误的能力。

定义4 数据切片通过动态数据依赖链，由直接或间接影响错误输出的语句组成。

定义5 完全切片通过动态数据依赖和控制依赖链，同时考虑错误输出语句和直接影响谓词的数据切片。

定义6 相关切片同时考虑数据依赖和控制依赖，在完全切片的基础上增加了潜在影响谓词。

数据切片的语句很少，但经常不能含有错误语句。相关

基金项目：教育部科学技术研究基金资助重点项目(108063)；江苏省自然科学基金资助项目(BK2008124)；中国矿业大学研究基金资助项目(0D080310)

作者简介：辛 良(1984－)，男，硕士研究生，主研方向：软件分析与测试；姜淑娟，教授、博士、博士生导师

收稿日期：2010-02-22 **E-mail**：free_xl715@163.com

切片和完全切片由于考虑了控制依赖的原因，因此比数据切片大很多，包含错误语句的能力也强很多。相关切片和完全切片不同，前者增加了潜在影响谓词的概念。总体而言，上述 3 种切片方法各有不同的优缺点。数据切片较小(语句较少)，但仅考虑数据依赖，包含错误的能力最低。完全切片和相关切片比数据切片大，同时考虑了数据和控制依赖，有强大的包含错误能力，但它们比数据切片耗费更多时间和空间。

2.2 代码优先方法

代码优先方法是一种与程序切片思想不同的错误定位方法。某一测试用例如果有错误输出，则说明本次执行路径上必然存在错误语句。对于多个产生错误输出的测试用例，可以肯定这些测试的执行路径上都有错误语句存在。针对该问题，基于语句错误可能性的定义，可以得到以下性质[4]:

(1)执行到错误语句时，不一定有错误输出。

(2)若有错误的输出，则一定执行了错误语句。

(3)某一语句错误的可能性与执行了该语句并且有错误输出的测试用例数量呈正比。

假定 *T*1、*T*2、*T*3 都是错误的测试用例，根据以上性质可知，*T*1∩*T*2∩*T*3 集合中语句错误的可能性较大。

2.3 基本错误的分类

依靠错误输出语句的数据切片方法进行错误定位，错误语句通常不能包含在切片中。原因是错误语句影响了选择谓词的值，改变了程序的执行路径。这属于控制依赖的范畴，仅考虑错误输出语句的数据依赖路径不能够找到错误语句。

从软件中程序错误输出结果的观点来看，可以大体分为 2 类错误：数据依赖部分的错误和谓词选择部分的错误。前者可以通过对错误输出语句数据切片进行查找来发现。对于后者则需要找到改变执行路径的选择谓词。

3 基于关键谓词的错误定位方法

3.1 关键谓词错误定位问题分析

基于测试用例和程序切片的代码优先方法能很好地应用于选择谓词分支上的错误，但不适用于每个测试都要经过的必经节点。

文献[5]提出关键谓词概念，通过改变谓词当前状态，根据能否得到正确的输出结果判定其是否为关键谓词。一条含有错误的输出测试用例，如果其执行路径当前考虑谓词为 true(false)，则改变状态为 false(true)，判断是否能得到正确的输出结果。如果能，则判定此谓词为关键谓词，否则根据优先级继续寻找关键谓词。

3.2 关键谓词错误定位问题的具体解决方案

本文算法分为 2 步，第 1 步是寻找关键谓词，第 2 步是进一步的错误定位，具体算法描述如下:

```
//寻找关键谓词
Procedure FindCPredicate(the wrong point)
{将经过的谓词节点都填入到PSet(n)中;
 将经过的谓词节点与错误输出点的距离排序;
  while(优先级最高的谓词节点)
     {   If(存在谓词状态转换 == 正确的输出)
                { 这个节点就是关键谓词节点; }
                  Else   { continue; }
     }}
//进一步错误定位
Procedure   FindFault()
{求关键谓词变量的后向动态数据切片S1;
 求错误输出变量的后向动态数据切片S2;
 切片集合  St = S1∪S2;
 对错误输出语句的另一测试用例求执行路径E;
 集合S = St ∩ E;
 }
```

对于一次错误输出结果测试用例的运行，将执行的谓词节点放入集合 *PSet*()，当前错误输出的数据依赖语句用集合 *DSet*()表示。

在寻找关键谓词的过程中，必须考虑 2 个问题：每次只考虑一个谓词和关键词的优先级。每次考虑一个关键谓词避免了多个关键谓词组合的情况，因为同时考虑多个关键谓词会增加分析问题的复杂度。通过大量实验证明，关键谓词通常发现错误的点是很近的[4]。基于该理论，可以简单地根据谓词和错误点的距离进行排序。距离越短，优先级越高。依据谓词的优先级，逐个考虑本条执行路径上的谓词是否为关键谓词。对于当前考虑的谓词节点，使用的测试用例必须保证本谓词节点前面的执行路径与原来错误输出结果的测试用例的执行路径相同。改变当前谓词节点的执行状态，对于错误语句 *S* 和它的执行实例 *Se*，寻找关键谓词时存在 3 种情况:

(1)执行了 *Se*，能够正确输出，就找到了关键谓词。

(2)执行了 *Se*，但不能正确输出，则继续寻找关键谓词。

(3)未执行 *Se*，能正确输出，但由于无法判定谓词实例转换是否能解决上述问题，因此继续寻找关键谓词。

上述错误定位算法按照选择谓词的优先级，逐个考虑当前谓词是否为关键谓词。确定关键谓词后，对此关键谓词进行后向数据切片 *S*1。而对于错误输出语句，进行后向数据切片 *S*2。*S*1 和 *S*2 的并集作为切片集合 *St*，即同时考虑了错误输出语句的控制依赖和数据依赖。

某一错误输出语句的测试用例，其执行路径 *E* 上必然存在错误语句。通过与切片集合 *St* 的求交集 *S*=*St*∩*E*，语句集合 *S* 中错误语句的可能性更大。

3.3 实例分析

一个简单的程序如下:

```
S1      Cin >> a>> n ;
S2      int i = 0;
S3      for ( ;i < n;++i){
S4              cin >> x>> y;
S5              int b = a – x;
S6              int c = y;
S7              if   (a > 1){
S8                      b = a / x;
S9                       --y;   }
S10             if (b > 0){
S11                     c = y + 1;   }
S12             if (c > 4){
S13                     z = x + y;   }
          else {
S14                     z = x - y; }
S15             printf(z);
    }
```

针对某一含有错误输出语句的测试用例，可以根据其执行路径，确定谓词集合。并根据与发现错误语句 *S*15 的远近距离进行排序。*PSet*(*S*15)={*S*3,*S*7,*S*10,*S*12}，*S*12 的优先级最高，*S*3 的优先级最低。最先考虑 *S*12，如果不是关键谓词点，则继续分析 *S*10，以此类推，直到 *S*3。

假如错误在语句 *S*11，则 *c*=*y*+1 被改为 *c*=*y*+5。表 1 描述了测试用例，*PSet*()、*DSet*()分别表示当前执行实例谓词集合和输出语句的数据切片集合。*T*1、*T*2、*T*3 是本文的测试用例，

(下转第 58 页)

算法 4

EVMS init;//启动 EVMS

add(device);//存储设备添加到 SN 中

update(SNS.bitmap);//更新 SN 的位矢图

update(SNS);//更新 SN

update(SNMM);//更新 SNMM

3.4 负载平衡算法

传统的带外虚拟网络存储系统只有一个 MS，当有多个客户端请求存储系统服务时，MS 容易成为系统的性能瓶颈，单个 MS 也会产生单点故障问题。为了解决上述问题，系统中采用 MSG 管理多个 MS 的方式。当有新的客户端加入存储系统时，MSG 的相应操作如下：

算法 5

设 N 是 MS 中客户端个数，f_i 是第 i 个客户端申请空闲空间的频度，F_j 是第 j 个 MS 中所有客户端申请存储空间的平均频度。

$F_j=\sum_{1}^{N} f_i/N$;//计算每个 MS 中的 F_j

$F_{min}=\min(F_1,F_2,\cdots,F_n)$;//选取平均频度最小的 MS

add($MS_{F_{min}}$);//把客户端加入平均频度最小的 MS

create(SSRDS);//创建 SSRDS

MSG 的形式比较灵活，当大规模的客户端加入存储系统时，MSMM 根据实时的客户端情况申请添加 MS，来解决 MS 过载问题。元数据对客户端和存储系统都极为重要，为了保证各个客户端元数据的可靠性，采用“模备份”方式将一个 MS 上的元数据备份到另一个 MS 中，当其中一个 MS 出现故障时，可以通过恢复找到故障 MS 中的元数据，提高了系统可靠性，具体操作如下：

算法 6

receive($WDACG_i$);//客户端 i 的元数据

findupdate(MS_i);//找到客户端对应的 MS_i，更新其客户端信息

$MS_{copy}=MS_i \mod N+1$;//计算备份的 MS

copy(MS_{copy});//$WDACG_i$ 数据备份到 MS_{copy}

存储系统在考虑 MSG 负载均衡的同时也考虑了部分 SN 过载的问题，如果一个 SN 同时有大量客户端在访问，则必然会使这个节点处于繁忙状态，读/写性能将受到严重影响，且 SN 会因为过载而产生故障。为了防止部分 SN 过载，在每次客户端申请 WDACG 时都要通过 $UR_{\min}=\min(UR_1,UR_2,\cdots,UR_n)$ 计算来选取使用率最小的 SN 为客户端分配空闲数据块，其中，UR_i 是第 i 个 SN 的使用率；n 是存储节点的个数。

3.5 元数据异步更新

整个存储系统在客户端与 MS 中各存在一个 LPT，为了达到这 2 个 LPT 的数据一致性和元数据服务器的传输信息和处理信息的压力较小，需要采用异步组更新元数据的方法。

当客户端的 WDACG 被写满时，客户端把 WDACG 信息发送给相应的 MS，MS 接收到客户端发送过来的 WDACG 后，更新对应的 LPT 表并进行元数据模备份，确保不会影响客户端读/写的性能。

4 结束语

本文提出的存储系统结构解决了带外虚拟网络存储的按需分配存储空间、高性能的读/写、在线存储容量扩展、负载平衡、元数据可靠性和元数据更新等问题，目前，基于 EVMS 技术的带外虚拟网络存储系统原型处于研制阶段，而现有存储系统存在一个问题，即一个客户端或多个客户端重复存储相同的数据块。重复存储相同的数据块降低了存储系统的性能和空间利用率，笔者考虑采用重复数据删除技术处理存储系统中数据块重复的问题，而有关存储系统的容错、备份、数据迁移和数据安全等问题有待进一步讨论并解决。

参考文献

[1] 赵文辉, 徐 俊, 周加林, 等. 网络存储技术[M]. 北京: 清华大学出版社, 2005.

[2] 王 迪, 舒继武, 薛 巍, 等. 块级别的海量存储虚拟化系统[J]. 清华大学学报: 自然科学版, 2007, 47(1): 108-111.

[3] 韦 理, 周悦芝, 夏 楠. 用于网络存储系统的存储空间动态分配方法[J]. 计算机工程, 2008, 34(5): 33-35.

[4] Enterprise Volume Management System(EVMS)[Z]. (2003-05-06). http://evms.sourceforge.net/.

[5] 赵跃龙, 戴祖雄, 王志刚, 等. 一种智能网络磁盘系统结构[J]. 计算机学报, 2008, 31(5): 858-867.

编辑 陈 晖

(上接第 55 页)

其中，测试用例的 4 个数字分别为(a, n, x, y)。这 4 个数字是需要用户输入的变量值。通过测试用例 $T1$ 和 $T2$，可以得到关键谓词为 $S12$。对于错误输出的测试用例 $T2$，$S12$ 后向切片为{$S4$, $S9$, $S11$, $S12$}，错误输出语句 $S15$ 后向数据切片为{$S4$, $S9$, $S13$, $S15$}。两切片的并集 St 为{$S4$, $S9$, $S11$, $S12$, $S13$, $S15$}。对于测试用例 $T3$，求出执行语句 E 和 St 的交集{$S4$, $S9$, $S11$, $S12$, $S13$, $S15$}。该集合含有错误的可能性很高，且比原程序集合小。通过进一步分析，可以确定错误发生在语句 $S11$。

表 1 测试用例列表

编号	测试用例	*PSet*()	*DSet*()	错误数	正确数
*T*1	(8, 1, 2, -2)	*S*3, *S*7, *S*10, *S*12	*S*4, *S*9, *S*14, *S*15	5	5
*T*2	(6, 1, 2, 2)	*S*3, *S*7, *S*10, *S*12	*S*4, *S*9, *S*13, *S*15	3	1
*T*3	(8, 1, 4, 2)	*S*3, *S*7, *S*10, *S*12	*S*4, *S*9, *S*13, *S*15	5	3

4 结束语

本文提出一种基于关键谓词并同时考虑代码优先的算法。该方法求出 2 个切片的并集，并按照代码优先的思想将错误输出语句的测试用例执行路径与此并集相比，得到需要测试的集合。它减少了分析语句的数量，得到的语句集合包含错误的可能性极大增加，降低了错误定位难度。

本文中确定关键谓词的方法在多重谓词结构中适用性不高，这是下一步工作需要研究的内容。

参考文献

[1] Weiser M. Programmers Use Slices When Debugging[J]. Communications of the ACM, 1982, 25(7): 446-452.

[2] 李必信. 一种分析和理解程序的方法——程序切片[J]. 计算机研究与发展, 2000, 37(3): 284-291.

[3] Zhang Xiangyu, He Haifeng. Experimental Evaluation of Using Dynamic Slices for Fault Location[C]//Proc. of the 6th International Symposium on Automated and Analysis-driven Debugging. California, USA: [s. n.], 2005.

[4] Sun Jirong, Li Zhshu, Ni Jianchen, et al. Software Fault Localization Based on Testing Requirement and Program Slice[C]//Proc. of International Conference on Networking, Architecture and Storage. Guilin, China: [s. n.], 2007.

[5] Zhang Xiangyu, Gupta N. Locating Faults Through Automated Predicate Switching[C]//Proc. of the 28th International Conference on Software Engineering. Shanghai, China: [s. n.], 2006.

编辑 陈 晖