一种基于前向计算的动态程序切片方法

王兴亚[1] 姜淑娟[1] 鞠小林[1,2] 邵浩然[2]

(中国矿业大学计算机科学与技术学院 徐州 221116)[1] (南通大学计算机科学与技术学院 南通 226019)[2]

摘 要 动态程序切片技术是一种重要的程序分析技术,在软件分析、测试与调试过程中有着广泛的应用。给出一种基于前向计算的动态程序切片方法,该方法首先在对当前执行语句进行定义使用分析的基础上计算该语句定义变量的影响集,其次计算该语句的直接动态依赖关系,最后计算当前执行语句中变量的动态切片。根据该方法设计并实现了一个Java动态程序切片系统,基于一组基准测试程序开展了切片实验,并与已有的切片方法进行了比较。实验结果表明,该方法可以得到比较精确的动态程序切片结果。

关键词 动态切片,前向计算,程序依赖性,三地址码

中图法分类号 TP311 **文献标识码** A

Dynamic Program Slicing Based on Forward Computation

WANG Xing-ya[1] JIANG Shu-juan[1] JU Xiao-lin[1,2] SHAO Hao-ran[2]

(School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)[1]

(School of Computer Science and Technology, Nantong University, Nantong 226019, China)[2]

Abstract Dynamic program slicing is widely used in software analyzing, testing and debugging. This paper proposed a novel forward computation approach for computing dynamic slices. Firstly our approach computes the influenced set of a defined variable in current executing statement based on the defined and used variables. Secondly it computes the direct dynamic dependence relationship of current statement. Finally it computes the dynamic slice of the variables. Applying this method, we designed and implemented a dynamic program slicing prototype for Java, and performed an experimental study on several open source programs. The experimental results show that the size of program slices by our approach is less than other methods.

Keywords Dynamic slicing, Forward computation, Program dependence, Three-address code

1 引言

动态程序切片技术是一种重要的软件分析技术,其概念最早由Korel和Laski于1988年提出[1]。其后的20余年,研究人员对动态程序切片的理论和方法开展了大量的研究工作[2]。目前,动态程序切片技术已经广泛应用于各类软件工程任务中,如:程序调试、软件测试、软件维护以及错误定位等[3,4]。

动态程序切片方法主要有两种:基于前向计算的动态程序切片方法[5,6]和基于后向分析的动态程序切片方法[7,8]。前向计算的动态程序切片方法根据兴趣点直接动态依赖节点计算兴趣点切片,此类方法需要维护程序执行过程中每一个节点的切片。后向分析的动态程序切片方法通过回溯动态依赖关系找到兴趣点的直接和间接依赖节点集合生成兴趣点切片,此类方法需要维护在程序一次执行过程中产生的所有动态依赖关系。这两类方法都需要维护大量的中间信息。与后者相比,前者具有很好的交互性,更适合于程序调试过程。基于此,本文对基于前向计算的动态程序切片方法进行研究。

传统前向计算切片方法很少考虑变量在定义过程中实际使用的变量,导致部分无关变量的切片结果包含在切片中,造成切片结果冗余。本文通过引入定义变量的影响集对传统前向计算切片方法进行改进,提出一种基于定义变量影响集的动态程序切片方法。

将本文提出的动态程序切片方法和传统前向计算方法应用于实际的切片环境中。实验结果表明通过使用本文提出的方法计算出的切片结果更加精确,切片规模大约缩小15%。

2 传统的动态切片前向计算方法

对于程序P,给定切片准则$C=\langle I,s,V\rangle$,动态程序切片是输入为I时程序P中所有直接和间接影响兴趣点s中变量集合V的语句构成的集合。在介绍本文提出的切片方法前,本节给出一组程序切片相关定义,并对传统基于前向计算动

到稿日期:2013-03-12 返修日期:2013-05-22 本文受国家自然科学基金(61202006,60970032),江苏省青蓝工程,江苏省自然科学研究项目(12KJB520014),江苏省研究生培养创新工程(CXZZ12_0935),南通市应用研究计划(BK2011025,BK2012023)资助。

王兴亚(1990—),男,硕士生,主要研究领域为软件分析与测试,E-mail:xingyawang@cumt.edu.cn;姜淑娟(1966—),女,博士,教授,博士生导师,CCF会员,主要研究领域为编译技术、软件工程等;鞠小林(1976—),男,博士生,讲师,主要研究领域为软件分析与测试;邵浩然(1975—),男,硕士,副教授,主要研究领域为软件工程与网格计算。

态程序切片方法进行深入分析，同时指出该方法存在的问题。

2.1 基本定义

定义 1(使用集) 给定语句 s，使用集 $Use[s]=\{v|v$ 是 s 中值被使用的变量$\}$。

定义 2(定义集) 给定语句 s，定义集 $Def[s]=\{v|v$ 是 s 中值被改变的变量$\}$。

定义 3(执行轨迹) 在给定输入下，程序 P 的执行轨迹 T 就是语句$\langle s_1,s_2,\cdots,s_n\rangle$的有序集合。为 T 中的每个元素给定一个唯一的时间戳 timestamp，表示语句的执行次序，记为 $s^{timestamp}$。$T[i,j]$是 T 的子序列，表示 T 中次序 i 到次序 j 的元素序列。

当程序中存在循环或方法调用时，程序中的一条语句在一次输入下可能会被多次执行。通过时间戳可以区分出同一条语句在一次输入下的不同执行。

控制依赖(Control Dependence，CD)用于表示处于同一控制流上程序实体之间的关系。若语句 s'决定语句 s 的执行与否，则语句 s 控制依赖于语句 s'。在静态分析程序的源代码时，受跳转语句或循环语句的影响，一条语句可能控制依赖于多条语句。而在程序的实际执行过程中，语句 s 执行与否只会受一条谓词语句的直接影响，该谓词语句即为语句 s 的动态控制依赖。数据依赖(Data Dependence，DD)用于表示由于数据定义和使用形成的实体之间的关系。在一条执行序列中，如果语句 s 使用了定义在语句 s'中的变量，则称语句 s 数据依赖于语句 s'。由于存在多条执行序列，一条语句可能数据依赖于多条语句。而在程序的实际执行过程中，语句 s 使用的变量只会在一条语句中被定义，该定义语句即是语句 s 的动态数据依赖。下面分别给出动态控制依赖和数据依赖的定义。

定义 4(动态控制依赖) 设 s^i,s^j 是执行轨迹 T 中的两个元素，其中 $i<j$。s^j 动态控制依赖于 s^i 当且仅当

$$(s^j\ \mathrm{CD}\ s^i)\wedge(s^j\overline{\mathrm{CD}}\ s^k),\forall k\in(i,j),$$

记作 s^j dynCD s^i。dynCD(s^j)表示 s^j 的动态控制依赖集。

定义 5(动态数据依赖) 设 s^i,s^j 是执行轨迹 T 中的两个元素，其中 $i<j$。s^j 动态数据依赖于 s^i 当且仅当

$$Def[s^i]\cap Use[s^j]-\sum_{k=i+1}^{j-1}Def(T(k))\neq\phi,$$

记作 s^j dynDD s^i。dynDD(s^j)表示 s^j 的动态数据依赖集。

2.2 传统前向计算方法中存在的问题

计算某一执行实例的动态程序切片时，该实例语句使用集中变量并未受到影响，其切片可通过该实例的动态数据依赖获得，且与其在最后定义处相同[6]。因此，只需要计算实例中定义变量的动态程序切片。传统的前向计算方法首先在计算定义集中变量的动态程序切片时，将所有使用变量的切片结果加入到定义变量的切片集合中。

在传统的前向计算切片方法中没有考虑定义变量发生数据定义时实际使用的变量，而是将所有的使用变量默认为定义变量定义时使用的变量。如图 1 所示，第 15 行语句的定义集 $Def[15]=\{i\}$，使用集 $Use[15]=\{$ret_sub2，$i,j\}$。采用传统前向计算切片方法计算得到变量 i 的切片结果包含了当前使用集中所有变量的切片结果，而实际上在该语句中变量 i 的值仅仅受子过程 sub2()的返回值 ret_sub2 的影响，只需包含 ret_sub2 的切片结果即可。因此，采用传统的前向计算法计算得到的切片结果存在冗余。

```
1. sub1(x){
2.   return x+1;
3. }
4. sub2(p,q){
5.   if(q>0){
6.     q++;
     }
7.   a=1;
8.   p+=a;
9.   retunr p;
10. }
11. main(){
12.   i=read();
13.   j=read();
14.   while(sub1(i)≤1){
15.     i=sub2(i,j);
     }
16.   print(i,j);
17. }
```

图 1 一个例子程序

3 基于影响集的动态切片前向计算方法

本文研究动态程序切片前向计算方法，在一条语句执行后计算该语句中各个变量的动态程序切片。给定一个执行实例 s^i，计算 s^i 中变量的动态切片分为 4 步：首先对语句 s 中的变量进行定义使用分析，然后对语句 s 中的定义变量进行影响集分析，接着计算 s^i 的动态控制依赖关系，最后进行变量动态程序切片计算。

3.1 定义使用分析

定义使用分析判断语句中变量的类型，生成定义集和使用集。在分析过程中，对谓词语句和调用语句进行变量抽象处理，分别对应生成谓词类型变量和返回值类型变量。谓词类型变量添加到定义集，返回值类型变量加入到使用集。

对语句进行定义使用分析时，首先生成语句的三地址码(Three-Address Code)中间表达式，对其中的每一条三地址码进行定义使用分析，最后将所有的中间表示分析结果进行整合，得到语句 s 的定义使用分析结果。由于源代码结构的复杂性，基于源代码的分析非常困难。使用三地址码进行定义使用分析，再通过映射得到源码中该语句中变量的定义使用分析结果，避免直接对源码进行分析，降低了分析的难度。

三地址代码是由如下形式组成的指令序列：$x=y$ op z。其中 x、y 和 z 可以是变量名、常量或编译器生成的临时量，op 表示一个运算符。在三地址代码中，一条指令的右侧最多有一个运算符，不允许出现组合的算术表达式。三地址代码拆分了多运算符算术表达式以及控制流语句的嵌套结构，适用于目标代码的生成和优化[9]。

针对语句 s 进行定义使用分析可以获得变量定义集 $Def[s]$和使用集 $Use[s]$。以图 1 例子程序中行 14 为例，$Def[14]=\{$pred_14$\}$，$Use[14]=\{$ret_sub1，$i\}$。由于行 14 是谓词语句，抽象行 14 为谓词类型变量 pred_14；行 14 调用了子过程 sub1，抽象 sub1 为返回值类型变量 ret_sub1。

3.2 影响集分析

如 2.2 节所述，在计算动态程序切片时，若不考虑定义变

量的使用变量集合，切片结果可能存在冗余。因此，需要对定义变量做影响集分析，找到变量在定义过程中实际使用的变量。为此，本文引入一个新的集合 $Inf[def]$，用于表示执行语句中影响定义变量 def 的变量集合。其定义如下：

定义 6(影响集) 给定语句 $s, v \in Def[s]$，影响集 $Inf[v]=\{v' | v'$ 是 v 定义时使用的变量，$v' \in Use[s]\}$。

三地址码的四元式(Quaduple)是具有 4 个域的记录结构，这 4 个域为：(op, arg1, arg2, result)。其中，op 表示一个运算符，arg1、arg2、result 为指针，它们指向有关名字在符号表中的登记项或一临时变量(可以空缺)。由于 result 的值肯定受到 arg1 和 arg2 的影响，因此

$Inf[result]=\{arg1, arg2\}$

如三地址指令 $x=y+z$，op 存放+，arg1 存放 y，arg2 存放 z，result 存放 x，由此得到 $Inf[x]=\{y,z\}$。图 1 例子程序的三地址码和三地址码四元式及定义变量的影响集如表 1 所列。

表 1 例子程序中定义变量的影响集

Source No.	Three-Address Code	Quadruple	def	Inf[def]
1	sub1(x) {	(par,@par1,_,x)	x	{@par1}
2	$i0=x+1; return $i0;	(+,x,1,$i0) (return, $i0, _, @ret)	@ret	{x}
3	}			
4	sub2(p,q) {	(par,@par1,_,p) (par,@par2,_,q)	p q	{@par1} {@par2}
5	if q >0 goto label0;	(j>,q,0,label0)	pred_5	{q}
6	q=q+1; label0:	(+,q,1,q) label0:	q	{q}
7	a=1;	(=,1,_,a)	a	∅
8	p=p+a;	(+,p,a,p)	p	{p,a}
9	return p;	(return,p,_,@ret)	@ret	{p}
10	}			
11	main() {			
12	i=invoke〈read()〉();	(call,read,0,@ret) (=,@ret,_,i)	i	{@ret}
13	j=invoke〈read()〉();	(call,read,0,@ret) (=,@ret,_,j)	j	{@ret}
	label0:	label0:		
15	i=invoke〈sub2()〉(i,j);	(par,i,_,@par1) (par,j,_,@par2) (call,sub2,2,@ret) (=,@ret,_,i)	@par1 @par2 i	{i} {j} {@ret}
	label1:	label1:		
14	$i0 = invoke〈sub1()〉(i); if $i0 <=1 goto lable0;	(par,i,_,@par1) (call,sub1,1,@ret) (=,@ret,_,$i0) (j<=,$i0,1,label0)	@par1 pred_14	{i} {@ret}
16	invoke〈print()〉(i,j);	(par,i,_,@par1) (par,j,_,@par2) (call,print,2,@ret)	@par1 @par2	{i} {j}
17	}			

3.3 动态依赖分析

动态依赖分析用于计算当前执行实例的动态依赖关系，包括动态数据依赖关系计算和动态控制依赖关系计算。

用 $Ld[v]$ 表示变量 v 最近被定义的位置，$LD[v]$ 表示变量 v 最近被定义的执行实例，则 $LD[v]=T(Ld[v])$。

给定执行实例 s^i，计算 s^i 的动态数据依赖。对于 $Use[s]$ 中的每一个元素 use，$LD[use]$ 是最近定义变量 use 的执行实例，因此 $LD[use]$ 是 s^i 的一个动态数据依赖。$Use[s]$ 中所有元素(使用变量)的 $LD[v]$ 构成了 s^i 的动态数据依赖。

给定执行实例 s^i，计算 s^i 的动态控制依赖。首先判断语句 s 的控制依赖(用 $CD(s)$ 表示)是否为空。如果 $CD(s)$ 为空，则语句 s 不存在过程内控制依赖，此时 s^i 动态控制依赖于当前调用点的动态控制依赖；如果 $CD(s)$ 不为空，则语句 s 存在过程内控制依赖，此时 s^i 动态控制依赖于所有控制依赖中最近执行的执行实例。比较 $CD(s)$ 中所有元素最近的执行位置，执行位置值最高的执行实例即为 s^i 的动态控制依赖。

完成执行实例 s^i 动态依赖关系的计算后，更新 $Def(s)$ 中每一个元素 def 的最近定义执行位置 $Ld(def)$ 为 i，最近定义执行实例 $LD(def)$ 为 s^i。计算执行实例动态依赖关系与更新定义元素执行位置/执行实例的顺序不能颠倒，否则会覆盖部分变量的最近执行位置/执行实例信息，造成动态依赖关系的丢失。

3.4 改进的动态切片前向计算算法

改进的动态程序切片前向计算算法如图 2 所示。语句不会影响使用集中元素的值，因此使用集中元素的切片结果不会发生改变。

```
algorithm: Dynamic program slicing based on forward computation
input: s^i—current execution instance
       call—call statement
       dynCD[s^i]—dynamic control dependence of s^i
       dynDD[s^i]—dynamic data dependence of s^i
       Inf[def]—the set of variables which influence
variable def
output: slice[def]—dynamic slice of variable def
begin
1. find slicePred by dynCD[s^i];
2. foreach use: Use[s] do
3.    find slice[use] by dynDD[s^i];
4. end for
5. foreach def: Def[s] do
6.    sliceTemp={s,call}∪slicePred;
7.    foreach inf: Inf[def] do
8.       sliceTemp=sliceTemp∪slice[inf];
9.    end for
10.   slice[def]=sliceTemp;
11. end for
end
```

图 2 改进的动态程序切片前向计算算法

图 2 中算法的 1—4 行根据当前执行实例的动态控制依赖和动态数据依赖分别获得控制依赖谓词变量和使用集中变量的动态切片。5—11 行计算定义集中所有变量的动态切片。定义变量的切片由 4 部分组成，分别是当前执行语句、调用语句、动态控制依赖谓词切片以及定义变量使用集中变量的切片。

3.5 实例分析

结合图 1 中的例子程序，说明本文提出的基于前向计算的动态程序切片方法。给定输入 $I=\{12: i=0; 13: j=1\}$，程序的执行轨迹 $T=\langle 11^1, 12^2, 13^3, 1^4, 2^5, 14^6, 4^7, 5^8, 6^9, 7^{10}, 8^{11}, 9^{12}, 15^{13}, 1^{14}, 2^{15}, 14^{16}, 16^{17}\rangle$。给定切片准则 $(I, 16^{17}, i)$，应用本文提出的算法计算程序的动态切片，实例分析过程及结果如表 2 所列。

表 2 实例分析

$s^{timestamp}$	Def[s]	Use[s]	call	pred	dynCD[s^j]	def	Inf[def]	inf	dynDD[inf]	slice[def]
11^1	∅	∅	∅	∅	∅	∅	∅	∅	∅	∅
12^2	{i}	{ret_read}	∅	∅	∅	i	{ret_read}	ret_read	∅	{12}
13^3	{j}	{ret_read}	∅	∅	∅	j	{ret_read}	ret_read	∅	{13}
1^4	{x}	{i}	14	∅	∅	x	{i}	i	12^2	{1,12,14}
2^5	{ret_sub1}	{x}	14	∅	∅	ret_sub1	{x}	x	1^4	{1,2,12,14}
14^6	{pred_14}	{ret_sub1,i}	∅	∅	∅	pred_14	{ret_sub1}	ret_sub1	2^5	{1,2,12,14}
4^7	{p,q}	{i,j}	15	pred_14	14^6	p q	{i} {j}	i j	12^2 13^3	{1,2,4,12,14,15} {1,2,4,12,13,14,15}
5^8	{pred_5}	{q}	15	pred_14	14^6	pred_5	{q}	q	4^7	{1,2,4,5,12,13,14,15}
6^9	{q}	{q}	15	pred_5	5^8	q	{q}	q	4^7	{1,2,4,5,6,12,13,14,15}
7^{10}	{a}	∅	15	pred_14	14^6	a	∅	∅	∅	{1,2,7,12,14,15}
8^{11}	{p}	{p,a}	15	pred_14	14^6	p	{p,a}	p a	4^7 7^{10}	{1,2,4,7,8,12,14,15}
9^{12}	{ret_sub2}	{p}	15	pred_14	14^6	ret_sub2	{p}	p	8^{11}	{1,2,4,7,8,9,12,14,15}
15^{13}	{i}	{ret_sub2,i,j}	∅	pred_14	14^6	i	{ret_sub2}	ret_sub2	9^{12}	{1,2,4,7,8,9,12,14,15}
1^{14}	{x}	{i}	14	pred_14	14^6	x	{i}	i	15^{13}	{1,2,4,7,8,9,12,14,15}
2^{15}	{ret_sub1}	{x}	14	pred_14	14^6	ret_sub1	{x}	x	1^{14}	{1,2,4,7,8,9,12,14,15}
14^{16}	{pred_14}	{ret_sub1,i}	∅	pred_14	14^6	pred_13	{ret_sub1}	ret_sub1	2^{15}	{1,2,4,7,8,9,12,14,15}
16^{17}	∅	{i,j}	∅	∅	∅	∅	∅	∅	∅	∅

由表 2 可知，由于考虑了实际影响定义变量 *def* 的变量集合 *Inf*[*def*]，本文方法生成的动态程序切片更加精确。例如，15^{13}处变量 i 的切片结果 *slice*[i]={15}∪*slice*[pred_14]∪*slice*[ret_rub2]，并没有包含使用变量 i 和 j 的切片结果。而传统方法要包含使用变量 i 和 j 的切片结果。

4 实验分析

我们在 Java 优化框架 Soot 生成的三地址码(Jimple)的基础上开发出一个动态程序切片工具(DSlicer4J)，对 Java 程序进行动态程序切片，其框架如图 3 所示。

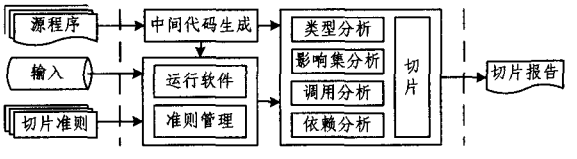

图 3 DSlicer4J 框架

表 3 实验对比结果

Subject	Description	LOC	Traditional	This paper
Jtcas	collision avoidance	181	43	32
Sorting	five sorting algorithms	222	22	19
Schedule1	priority scheduler	290	37	33
Schedule2	priority scheduler	317	48	46
Print_tokens1	lexical analyser	478	56	47
Print_tokens2	lexical analyser	410	40	31

用本文设计的 DSlicer4J 工具对 6 个基准测试程序分别进行实验并与传统的方法进行了对比。其中 Jtcas 和 Sorting 来源于 Subject Infrastructure Repository(SIR)[10]，其余 4 个程序分别是西门子程序集中 C 语言程序的 Java 版本[1)]。实验结果如表 3 所列。由实验结果可知，利用本文方法计算得到的切片结果比传统方法更加精确，切片数量平均减少 15%。

5 相关工作

Korel 等人[5]最早提出了基于前向计算的动态程序切片生成算法。他们将程序细化分为可移动块，当一个可移动块执行结束后，判断当前可移动块是否加入到动态程序切片。赵瑞莲等人[11]在 Korel 切片方法的基础上，提出了以块之间的支配关系表示程序中块的嵌套包含关系，使程序结构更加清晰，能计算出更精确的动态程序切片。他们的方法可以对过程内程序进行动态程序切片，不能完成过程间程序的动态程序切片。

Beszedes 等人[12]考虑潜在依赖关系，提出了一种基于前向计算的相关程序切片方法。同时，他们对 C 程序中的指针和函数调用进行处理，使该方法可以应用于大型 C 程序[13]。Masri 等人[14,15]总结了 5 种动态依赖关系，根据 5 种直接依赖关系求解动态程序切片。Zhang 等人[6]利用约简的有序二叉判定图来表示切片，有效减少了变量切片结果维护的时间和空间。他们的方法考虑所有的动态依赖关系，没有分辨出实际影响定义变量的变量集合，计算的切片结果存在冗余。

结束语 本文提出了一种基于前向计算的动态程序切片方法，通过分析定义变量定义时实际使用的变量，提出基于定义变量影响集的动态程序切片方法。设计实现了一个动态程序切片原型工具 DSlicer4J，并在一组基准测试集上开展切片实验验证。我们的理论分析和实验结果表明，本文提出的基于定义变量影响集的前向切片算法与传统的切片方法相比，得到的切片结果更精确。下一步工作是：首先在我们的方法中应用数据流分析以提升切片算法的精度；其次研究海量运行信息的处理方法使得我们的方法可以应用于实际大规模的应用程序的切片；最后将我们的方法推广用于更多的其它语言的程序。

参 考 文 献

[1] Korel B, Laski J. Dynamic program slicing[J]. Information Processing Letters, 1988, 29(3): 155-163

(下转第 278 页)

1) http://www.cc.gatech.edu/raul/siemens/

的变化，在这个方面可以体现 DSFLC 算法具有较好的可扩展性。

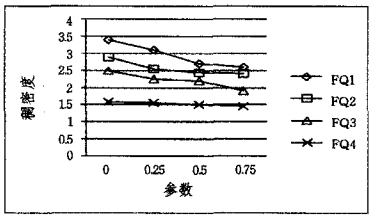

图 8　随着 ε 变化，稠密度的变化

结束语　本文研究了在大规模图数据集中发现最稠密子图问题，提出了一个近似算法 DSFLC：对于任何参数 $\varepsilon(\varepsilon>0)$，DSFLC 算法只需要扫描大规模数据集 $O(\log_{1+\varepsilon}n)$ 次，便可以获得最稠密子图问题最优解的 $2(1+\varepsilon)$ 近似解。基于 Twitter Storm 流式框架，将 DSFLC 算法进行并行化处理，大大提高了算法的执行效率。通过实验测试，验证了算法的有效性和可扩展性，确保了近似算法的解具有较好的理论保证。综上所述，基于 DSFLC 算法，配合使用 Storm 框架可以很好地在大规模流式图数据集中发现最稠密子图。将算法完全进行并行化处理是亟待解决的一个难题，也是我们今后的研究方向。

参 考 文 献

[1] Yon D, Filippo G, Marco P. Extraction and classification of dense communities in the Web[C]//WWW 2007. 2007: 461-470

[2] Newman M E J. Modularity and community structure in networks[C]//PNAS 2006. 2006: 8577-8582

[3] William F G, Steve L, Lee G C. Efficient identification of Web communities[C]//KDD 2000. 2000: 150-160

[4] https://github.com/nathanmarz/storm

[5] http://www.dblp.org/db/

[6] Goldberg A V. Finding a maximum density Subgraph[R]. UCB/CSD-84-171. EECS Department, University of California, Berkeley, 1984

[7] Charikar M. Greedy approximation algorithms for finding dense components in a graph[C]//APPROX 2000. 2000: 84-90

[8] Lawler F. Combinatorial Optimization: Networks and Matroids [M]. Holt, Rinehart, and Winston, 1976

[9] Gibson D, Kumar R, Tomkins A. Discovering large dense subgraphs in massive graphs[C]//VLDB 2005. 2005: 721-732

[10] Bahmani B, umar R. Sergei Vassilvitskii: D ensest Subgraph in Streaming and MapReduce[C]//VLDB 2012. 2012: 454-465

[11] Silva A, Meira W Jr, Zaki M J. Mining Attribute-structure Correlated Patterns in Large Attributed Graphs[C]//VLDB. 2012: 466-477

[12] Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters[C]//ACM 2008. 2008: 107-113

[13] Bu Ying-yi, Howe B, Balazinska M, et al. HaLoop: Efficient Iterative Data Processing on Large Clusters [C]//VLDB 2010. 2010: 285-296

[14] Abouzeid A, Bajda-Pawlikowski K, Abadi D J, et al. HadoopDB: An architectural hybrid of MapReduce and DBMS technologies for analytical workloads[C]//VLDB 2009. 2009: 922-933

[15] Condie T, Conway N, Alvaro P, et al. MapReduce Online[C]// NSDI'10 Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation. USENIX Association Berkeley, CA, USA, 2010: 21

(上接第 253 页)

[2] Xu Bao-wen, Qian Ju, Zhang Xiao-fang, et al. A brief survey of program slicing[J]. SIGSOFT Softw. Eng. Notes, 2005, 30(2): 1-36

[3] Abadi A, Ettinger R, Feldman Y A. Fine slicing: theory and applications for computation extraction[C]// Proceedings of the 15th international conference on Fundamental Approaches to Software Engineering, 2012. Tallinn: Springer-Verlag, 2012: 471-485

[4] Zhang Xiang-yu, Gupta N, Gupta R. A study of effectiveness of dynamic slicing in locating real faults[J]. Empirical Softw. Engg., 2007, 12(2): 143-160

[5] Korel B, Yalamanchili S. Forward computation of dynamic program slices[C]//Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis, 1994. Washington: ACM, 1994: 66-79

[6] Zhang Xiang-yu, Gupta R, Zhang You-tao. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams[C]// Proceedings of the 26th International Conference on Software Engineering, 2004. Edinburgh: IEEE Computer Society, 2004: 502-511

[7] Zhang Xiang-yu, Gupta R, Zhang You-tao. Precise dynamic slicing algorithms[C]//Proceedings of the 25th International Conference on Software Engineering, 2003. Portland: IEEE Computer Society, 2003: 319-329

[8] Nagarajan V, Jeffrey D, Gupta R, et al. A system for debugging via online tracing and dynamic slicing[J]. Software-Practice & Experience, 2012, 42(11): 1431-1431

[9] Alfred V A, Monica S L, Ravi S, et al. Compilers Principles, Techniques and Tools(第 2 版)[M]. 赵建华, 郑滔, 戴新宇, 译. 北京: 机械工业出版社, 2009: 382-402

[10] Do H, Elbaum S, Rothermel G. Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact[J]. Empirical Softw. Engg., 2005, 10(4): 405-435

[11] 王雪莲, 赵瑞莲, 李立健. 一种用于测试数据生成的动态程序切片算法[J]. 计算机应用, 2005, 25(6): 1445-1447

[12] Gyimothy T, Beszedes A, Forgacs I. An efficient relevant slicing method for debugging[J]. SIGSOFT Softw. Eng. Notes, 1999, 24(6): 303-321

[13] Beszedes A, Gergely T, Szabo Z M, et al. Dynamic slicing method for maintenance of large C programs[C]//Proceedings of the fifth European Conference on Software Maintenance and Reengineering, 2001. Lisbon: IEEE Computer Society, 2001: 105-113

[14] Masri W, Podgurski A, Leon D. Detecting and Debugging Insecure Information Flows[C]// Proceedings of the 15th International Symposium on Software Reliability Engineering, 2004. Saint-Malo: IEEE Computer Society, 2004: 198-209

[15] Masri W, Nahas N, Podgurski A. Memoized Forward Computation of Dynamic Slices[C]// Proceedings of the 17th International Symposium on Software Reliability Engineering, 2006. Raleigh: IEEE Computer Society, 2006: 23-32