Hadoop 集群下的并行克隆代码检测*

叶 林,姚国祥
(暨南大学 信息科学技术学院,广东 广州 510632)

摘 要:克隆代码会导致项目的维护困难,削弱项目的健壮性,并且克隆代码中所包含的 bug 会破坏整个项目。当前克隆代码检测技术或者拘泥于只能检测少数几种克隆代码,或者需要极高的检测时间。而且如果需要检测大量的源代码,一台机器的主存也许无法存储所有的信息。对克隆代码检测技术的并行运行进行了可能性研究,使用基于程序依赖图的克隆代码检测技术,这种技术不仅可以检测出语法上的克隆,也可以检测出语义上的克隆,提出了一个并行子图同构检测方法并使用 MapReduce 并行实现,实验结果极大地提高了该方法的运行速度。

关键词:克隆代码;程序依赖图;同构匹配检测;Hadoop
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1674-7720(2014)02-0069-03

Parallel clone code detector in Hadoop cluster

Ye Lin, Yao Guoxiang
(College of Information Science and Technology, Ji'nan University, Guangzhou 510632, China)

Abstract: Clone codes make the project hard to maintain and weaken the robustness, and the bugs in these codes would undermine the whole project. The state-of-the-art clone code detectors are either not able to find code with same semantics, or computationally expensive. And if clone code detector is to be performed on plenty number of code, the main memory of one machine may not able to hold all the information. In this paper we focus on the parallel of the clone code detector, we utilize the program dependence graph (PDG)-based code clone detection method, which can not only check the code in contiguous syntax, but also the code with the same semantics. We present an approach to parallel the isomorphism matching in the PDG. By using MapReduce paradigm, we dramatically enhance the searching speed of this method.

Key words: clone code; PDG; isomorphism matching; Hadoop

在软件项目的开发过程中,由于能够降低开发者的工作量,"复制粘贴"也许是最常使用的操作。但这也带来了克隆代码的问题。

克隆代码的存在给软件维护带来了困难,当开发者试图修改代码时,他们很可能修改了克隆代码中的一处而忘记了别的地方,这显然会带来代码的不一致。为了避免这个难题,大量的克隆代码检测技术被提出。但问题在于克隆代码的精确定义本身就不明确,现有的每一种方法都有其对于克隆代码自己的定义。因此,同样的源代码,如果用不同的克隆代码检测方法检测,可能会得到完全不同的结果。

基于程序依赖图的方法能够探测语义克隆代码,而且它还具有一个其他方法所不具有的能力:能够探测非连续性的克隆代码[1]。非连续性的克隆代码是被其他代码或文件所分割开来的克隆代码,克隆代码中的代码并不是连续的。而开发者往往会在粘贴克隆代码后做一些修改,这样,基于程序依赖图的检测方法就能够检测出这种克隆代码。

但是基于程序依赖图的方法有一个很大的缺点,即运行非常缓慢。程序依赖图的同构检测是著名的图同构匹配问题,该问题为 NP 完全问题,需要指数级的时间复杂度,这导致了运行时间呈指数级增长。

本文提出了一种并行执行程序依赖图同构匹配的方法。通过使用这种方法,减少了这一特定问题的图同构匹配算法所需要的时间。并使用 MapReduce 这一流行的并行框架来并行该方法。

* 基金项目:国家自然科学基金项目(61272415, 61272413, 61133014)

1 背景知识

1.1 程序依赖图

程序依赖图是一个有向图，该图的顶点代表了源代码中的代码，而边代表了两个顶点之间的依赖。在程序依赖图中只有两种边：代表控制依赖的边和代表数据依赖的边。以下展示了一个源代码的例子，图 1 为该源代码所产生的程序依赖图。

```
#include <stdio.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define BUFFER_SIZE     1024
#define DELIM   "\t"

int main(int argc, char *argv[]){
    char strLastKey[BUFFER_SIZE];
    char strLine[BUFFER_SIZE];
    int count = 0;

    *strLastKey = '\0';
    *strLine = '\0';

    while( fgets(strLine, BUFFER_SIZE - 1, stdin) ){
        char *strCurrKey = NULL;
        char *strCurrNum = NULL;

        strCurrKey  = strtok(strLine, DELIM);
        strCurrNum = strtok(NULL, DELIM);
                /* necessary to check error but.... */

        if( strLastKey[0] == '\0'){
            strcpy(strLastKey, strCurrKey);
        }
        if(strcmp(strCurrKey, strLastKey)){
            printf("%s\t%d\n", strLastKey, count);
            count = atoi(strCurrNum);
        }else{
            count += atoi(strCurrNum);
        }
        strcpy(strLastKey, strCurrKey);

    }
    printf("%s\t%d\n", strLastKey, count);
                                /* flush the count */
    return 0;
}
```

1.2 MapReduce

MapReduce[2]是一个流行的编程模型，该模型能够通过一个运行在集群上的并行的、分布式的算法对大数据集进行处理。它提供了一个简单易用的并行算法编程框架，使用该框架的开发者只需要定义两个函数：Map 和 Reduce。原始数据被该框架转换成键值对，每一个 Map 进程每一次处理一个键值对(key,value)：

Map: <k1, v1> → <k2, v2>

Map 函数在集群中并行执行，MapReduce 框架将所有相同的 key 的键值对传递给一个 Reduce 函数。Reduce 函数产生最终的结果：

Reduce: <k2,v2> → <k3,v3>

2 程序设计算法

首先把源代码转换成以静态形式表示数据流和控制流的程序依赖图，将其记为 s-PDG。程序依赖图的节点代表了源代码中的语句(声明、赋值、表达式、控制逻辑等)，同时记录所有节点对应源代码的类别以便在后面的比对中使用。然后选择一段程序块所对应的 s-PDG 的子图，作为查找与图同构的样本，将这个子图记为 b-PDG。随后对 s-PDG 和 b-PDG 进行比对，以检测除了

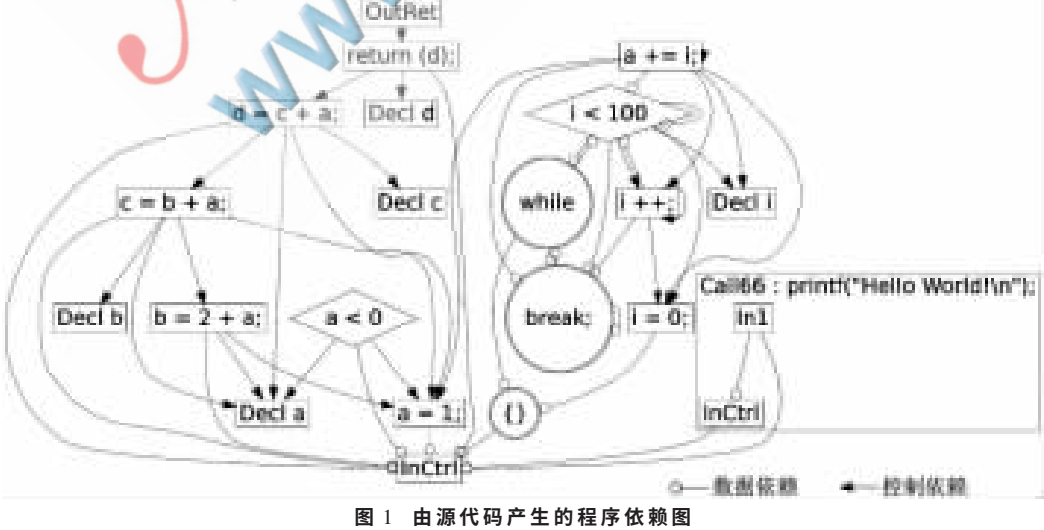

图 1 由源代码产生的程序依赖图

b-PDG 本身以外是否还有别的 s-PDG 的子图与 b-PDG 同构。如果有，则这个子图所对应的代码就与 b-PDG 对应的程序块为克隆代码。

经典的算法在检测子图同构时只能顺序执行，本文所要做的是将 s-PDG 切分成多个小图，然后并行子图同构检测。在论述切分 s-PDG 的方法之前，先给出会在切分中使用的伪圆的定义。

在图 $G=(V,E)$ 中，任给 $A \in V$，以 A 为圆心，以一个正数为半径，对于任意节点 $B \in V$，如果 AB 之间的最短路径长度(对于边无权值的图，最短路径长度为最短路径所经过的节点的个数)小于半径，则 B 位于该伪圆中。当计算最短路径时忽略边的方向。

按照参考文献[3]中提出的方法切割 s-PDG：

(1)根据 s-PDG 节点的种类分别计数。

(2)取出 s-PDG 中数量最少的节点的种类，将其记为种类 l。然后选取出 b-PDG 中属于种类 l 的节点。如果 b-PDG 中没有种类 l 的节点，则变更种类 l 为 s-PDG 中第二少种类的节点。如果种类 l 仍然在 b-PDG 中没有节点，则继续变更种类 l 为 s-PDG 中第三少种类的节点，直到 b-PDG 中存在种类 l 的节点。

(3)计算 s-PDG 中所有这些种类 l 的节点与其他节点的距离，将最大值定为伪半径。

(4)以上面计算出的伪半径，以 s-PDG 中种类为 l 的节点为圆心，可以得到一些伪圆。这些伪圆就是切割 s-PDG 的最终结果。将它们记为 c-PDG 的集合。

在查找同构子图的过程中必须检查节点的种类，对应的节点必须有同样的种类。所以同构子图必须有种类为 l 的节点。考虑到 b-PDG 的尺寸大小，在 s-PDG 中的节点如果距步骤(4)中选取的圆心距离过大，则这些节点不可能处于同构子图中，因此可以把这些节点切除不再考虑。

该算法的基本流程如图 2 所示。

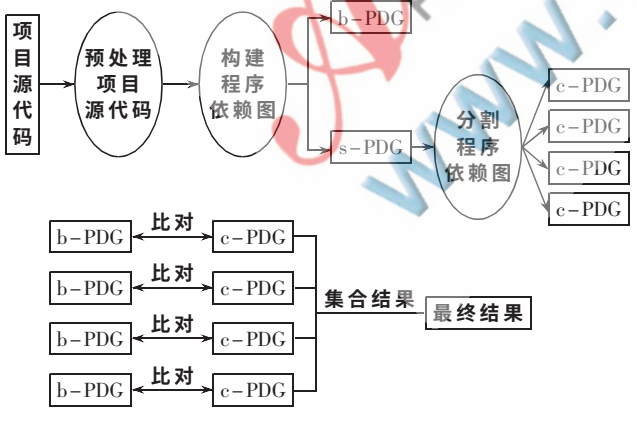

图 2 基本流程图

3 算法的实现

使用 JavaPDG[4]生成整个项目的程序依赖图。JavaPDG 是一个静态的 Java 字节码分析器。这个工具能够产生各种不同的对源代码的图形展示，例如系统依赖图、程序依赖图、控制流图和函数调用图。

使用 Hadoop [5](一个 MapReduce 框架的开源实现)来并行这个子图集的同构匹配。

使用 Igraph[6]来检测子图同构匹配。Igraph 是一个针对图的操作的开源软件包，由于 Igraph 是用 C 语言写成的，必须通过 Hadoop 流来将这个软件包用于并行同构检测。

4 实验与评价

通过对两个开源项目的检测来评价本文的算法，结果如表 1 所示。通过代码行数和对应程序依赖图的节点和边的个数来对比项目的大小。将经典 PDG 匹配算法与以 3 台机器组成的集群上并行为例的本文算法所消耗的时间进行了比较。

表 1 实验结果

开源项目	节点粒度	经典算法用时/min	本文并行算法用时/s
Spring-context-3.2.1	10	625	70
	50	568	78
	100	500	85
	200	420	88
	400	334	100
Gitblit	10	630	76
	50	593	82
	100	582	84
	200	542	100
	400	523	120

结果显示，本文算法极大地提高了同构匹配的性能，经典的程序依赖图同构匹配算法需要花费几个小时，而本文并行算法仅仅花费几分钟。这是因为并行算法移除了程序依赖图中的部分节点，而且并行了同构匹配的过程。

本文提出了一种提高基于程序依赖图的克隆代码检测性能的方法。把程序依赖图分割成若干个小图并使用 Hadoop 并行执行子图同构检测，使得算法的性能得到了提高。使用两个得到广泛使用的开源项目来测试本文算法，测试结果显示该算法显著地提高了克隆代码检测的性能。

参考文献

[1] BELLON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools[J]. IEEE Transactions on Software Engineering, 2007, 33(9): 577-591.

[2] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[3] LI J, ERNST M D. CBCD: cloned buggy code detector[C]. ICSE 34th International Conference on Software Engineering, 2012: 310-320.

[4] SHU G, SUN B, HENDERSON T A, et al. JavaPDG: a new platform for program dependence analysis[C]. In Proceedings

of the 6th IEEE International Conference on Software Testing, Verification and Validation, Testing Tools Track, Luxembourg, 2013: 18-22.

[5] Hadoop. The apache software foundation[EB/OL]. [2013-09-10]. http://hadoop.apache.org/.

[6] CSARDI G, NEPUSZ T. The igraph software package for complex network research[C]. InterJournal, Complex Systems, 1695. 2006.

(收稿日期: 2013-09-22)

作者简介:

叶林, 男, 1987 年生, 硕士在读, 主要研究方向: 云计算。

姚国祥, 男, 1959 年生, 教授, 主要研究方向: 计算机网络。