基于程序控制流的静态软件胎记算法研究

赵雅霖 雷聚超 唐俊勇
(西安工业大学计算机科学与工程学院 陕西 西安 710021)

摘 要 针对传统 SKB 软件胎记检测程序抄袭结果不准确的问题,提出一种基于程序控制流的软件胎记方法进行抄袭检测。使用 Java 程序静态分析的结果作为元信息,通过分析元信息得到字节流指令。分析字节指令流以及程序的控制流结构,并将其中的外部引用替换为外部控制流结构,以该控制流结构作为软件胎记。使用 VF2 算法计算两胎记之间的相似度,从而判定两程序之间是否存在抄袭行为。实验结果表明,所提胎记较传统 SKB 胎记更具可信性。

关键词 软件胎记 程序控制流结构 VF2 算法 相似度 抄袭检测

中图分类号 TP3 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2020.03.006

STATIC SOFTWARE BIRTHMARK ALGORITHM BASED ON PROGRAM CONTROL FLOW

Zhao Yalin Lei Juchao Tang Junyong
(*School of Computer Science and Engineering, Xi'an Technological University, Xi'an 710021, Shaanxi, China*)

Abstract Aiming at the inaccurate program plagiarism results detected by traditional SKB software birthmark, this paper proposes a software birthmark method based on program control flow for plagiarism detection. It used the static analysis result of Java program as the meta information, and the byte stream instruction was obtained by analyzing the meta information. The byte instruction stream and program's control flow structure were analyzed, and the external reference was replaced with an external control flow structure, which was taken as a software birthmark. We used VF2 algorithm to calculate the similarity between the two birthmark, so as to determine whether there was plagiarism between two programs. The experimental results show that the birthmark proposed in this paper is more reliable than the traditional SKB birthmark.

Keywords Software birthmark Program control flow structure VF2 algorithm Similarity Plagiarism detection

0 引 言

随着网络高速发展,人们对各种软件的使用也越来越频繁,软件给人们生活带来了极大的便利和巨大的经济效益。在这些利益的背后,很多商家或者个人通过不正当手段来盗取他人的软件以获取非法利益,给软件著作者带来了很大的损失。商业软件联盟(BSA)在 2018 年全球软件调查报告中指出,全球 PC 的软件盗版率为 37%,而中国的软件盗版率更是高达 66%。盗版软件每年造成的损失高达 3 590 亿美元。可见,盗版软件严重危害了版权所有者的利益。

针对软件抄袭问题,目前主要有两种解决方法。一种是对软件本身的保护,保证软件不被盗取;另一种是在软件被盗取之后,通过找到软件中可以证明软件版权信息的方法来维权。由于 Java 具有跨平台运行和开源的特点,利用逆向分析手段将 Java 指令恢复为源代码很容易,因此无法做到对软件的绝对保护。在软件被盗取后,目前常见的证明手段是通过软件水印[1]或软件胎记[2]的方法。软件水印是指向软件代码中嵌入唯一的标识符的技术手段,尽管在嵌入的时候已经对这些信息做了一些隐蔽处理,但目前可以通

收稿日期:2019-04-25。西安工业大学教育改革项目(18jgz01)。**赵雅霖**,硕士生,主研领域:计算机软件与理论。**雷聚超**,教授。**唐俊勇**,副教授。

过多种方法对这些水印进行篡改和移除。

软件胎记是指程序中一些与生俱来的,固有的特征,它具有很强的抗攻击能力,无论代码被进行了怎样的篡改和迷惑,其胎记依然保持不变。在软件胎记的研究中,陈林等[3]提出一种将程序中所有 k-gram 碎片频率构成向量生成的软件胎记算法;Xie 等[4]则在计算 k-gram 碎片对单种语义保持变换的频数变化率的基础上,统计出各碎片的抗攻击值构建胎记向量,以此来判定抄袭行为。以上两种软件胎记一定程度上克服了指令集胎记健壮性不强的问题,但并没有从本质上解决问题,因为造成语义丢失的根本原因在于 k-gram 算法本身。Schuler 等[5]提出了基于 Java 平台的 API 动态软件胎记算法。Park 等[6]提出了一种由静态分析程序后得到的 API 踪迹的集合生成的软件胎记算法,但是目前已有研究者提出了针对系统调用的攻击方法使该软件胎记算法失效。

本文提出了一种基于控制流的静态软件胎记算法。首先,提取中间文件中的控制流结构,并以图的形式进行存储。其次,对软件胎记中的可达方法引用指令进行解引用处理,并对其中的冗余指令进行约减处理,为软件胎记的比较降低一定的复杂度,以提高胎记的可信性。最后通过一种基于 VF2[7]算法的胎记对比模型得出胎记的相似度,进而判定两软件是否抄袭。

1　传统软件胎记算法

软件胎记的概念是由 Grover[8]提出的,一个完整的软件胎记抄袭检测模型主要由两个函数组成:

$$\begin{cases} B_p = extract(P) \\ similarity = compare(B_P, B_Q) \end{cases} \quad (1)$$

式中:B_P表示程序 P 的胎记,*extract*( )表示软件胎记的特征提取函数,*similarity* 为[0,1]区间上的实数。式(1)表明了软件胎记是由一个软件胎记提取算法对软件的一方面或多方面特征提取的产物,软件胎记的相似度是由软件胎记比较模型对比两个软件胎记计算出的数值,该数值在 0 到 1 之间,通过该数值确定软件是否抄袭。

1.1　使用软件胎记检测程序抄袭

使用静态软件胎记进行程序检测分为以下三步:首先,对源程序和可疑程序的源码或中间文件及可执行码进行静态分析,获取源程序及可疑程序中的一项或多项特征,通过对特征的加工,获得源程序和可疑程序的软件胎记。其次,针对提取出的软件胎记构建一个比较模型,该模型以源程序及可疑程序的软件胎记为输入,输出两胎记之间的相似值。最后通过相似值判定程序是否抄袭,设 α 为阈值,P、Q 为两个程序,B_P、B_Q分别为两个程序的胎记,则软件抄袭的判定标准为:

(1) 若 $comp(B_P, B_Q) \geqslant 1 - \alpha$,则 P、Q 为抄袭关系;

(2) 若 $comp(B_P, B_Q) \leqslant \alpha$,则 P、Q 为非抄袭关系;

(3) 若 $comp(B_P, B_Q) < 1 - \alpha$ 或 $comp(B_P, B_Q) > \alpha$,则 P、Q 关系不确定。

1.2　传统 k-gram 胎记

k-gram 软件胎记是由 Myles 等[9]提出的,主要思想是从程序指令中提取 k-gram 碎片,将所提取的所有碎片作为一个集合来表示软件胎记。在提取 k-gram 碎片时,主要以滑动窗口的方式去提取,即按照其物理相邻地址去划分。仅考虑物理关系而不考虑其逻辑关系或内部的控制流情况。

1.3　软件胎记的评价体系

软件胎记的属性有两种,分别为可信性和健壮性。

定义 1　软件胎记的可信性:设 P 和 Q 是非抄袭关系的两个程序,*extract* 为软件胎记的提取函数,B_p与 B_Q分别为两个软件的软件胎记,$B_P = extract(P)$,$B_Q = extract(Q)$,如果 $B_P \neq B_Q$,则称 *extract* 函数具有可信性。

定义 2　软件胎记的健壮性:设程序 P'是程序 P 的抄袭程序,*extract* 为软件胎记的提取函数,B_p与 $B_{P'}$分别为两个程序的软件胎记,$B_P = extract(P)$,$B_{P'} = extract(P')$,如果 $B_P = B_{P'}$,则称 *extract* 函数具有健壮性。

在系统完成建模之后,通过精确率(Precision)、召回率(Recall)、F 度量值(F-Measure)进行评价。

在样本中,设 T 为检测结果正确,F 为检测结果错误。T_P表示将盗版程序对检测为盗版程序对,T_N表示将非盗版程序对检测为非盗版程序对,F_P表示将非盗版程序对检测为盗版程序对,F_N表示将盗版检测为非盗版程序对。

精确率:盗版检测结果的正确率,即检测结果为盗版的程序对中实际结果确实是盗版的比率,计算公式为:

$$Precision = \frac{T_P}{T_P + F_P}$$

召回率:盗版检测结果的正确率,即检测结果为盗版的程序对中实际结果确实是盗版的比率。计算公式为:

$$Recall = \frac{T_P}{T_P + F_N}$$

F 值计算公式为：

$$F_{measure} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

单以检测结果的正确率来评价抄袭检测模型的好坏是不准确的，在精确率和召回率都很高的情况下，抄袭检测模型的检测结果才有参考意义。

2 改进的软件胎记算法

本文提出一种基于程序控制流的软件胎记，使用该软件胎记对程序对进行抄袭检测分为以下三步：

首先，对 Java 源程序 P 与 Java 可疑程序 Q 通过分析各自对应的 Class 中间文件获取其对应的控制流特征，将两程序的控制流特征加工为各自的静态软件胎记。

其次，构建一种基于 VF2 算法的控制流特征静态胎记对比模型，计算两胎记之间的相似度作为源程序与可疑程序之间的相似度。

最后，设置检测阈值 α，使用式(2)所示的判定标准得出两程序是否抄袭的结论。

其流程如图 1 所示。

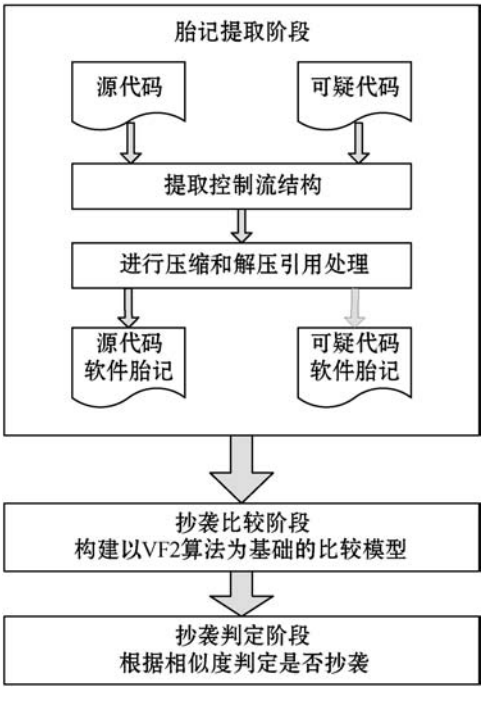

图 1 抄袭检测流程图

目前，Java 的字节码指令共有 203 个，按照功能可分为 10 种[10]，但从控制流的角度来说，本文将指令分为 3 种：

单后继指令：该种指令在读取完该指令及操作数后将下一个字节翻译为指令。

双后继指令：该种指令必定附带操作数，并且在指令附带的操作数中必定带有一个 offset 部分。

多后继指令：该种指令在整个体系中只有两条，分别是 lookupswitch 与 tableswitch，这两条指令都是由 switch 结构编译而来。当 switch 结构中的 case 值比较稀疏时，会生成 lookupswitch，反之则会生成 tableswitch。这类指令的后继指令数目不定，由 case 分支的数目决定。

Java 语言用这三类指令即可构成程序中最基本的三种控制流结构：分支结构、循环结构和顺序结构。其中双后继指令和多后继指令构成了分支结构中的 if 和 switch 结构，顺序结构和循环结构均由单后继指令构成，稍有不同的是循环结构中一定会有一个 goto 指令，其后继是指向之前的某条指令。所有程序的控制流结构均由以上三种基本控制流结构组合而成。

2.1 生成基于控制流的静态程序胎记

Class 文件作为编译的中间文件，为 JVM 提供了所有运行期必要的符号信息。一个 Class 文件中最主要信息的就是 Method 数组中的 CodeAttribute，该属性表记录了源程序经编译后产生的指令数组，JVM 在运行期通过遍历该数组来执行一个方法。由于 JVM 的字节码指令体系是基于栈的变长指令体系，所以本文使用 map 的结构来存储程序的控制流信息。

本文获取方法 M 的程序控制流胎记步骤如下：

(1) 指定 jar 文件 J 和迭代深度 $R_{\max}$，解压后获得工程 P，设置 $R_{\text{temp}} = 1$。

(2) 根据给定的方法签名 M_{sign} 和 class 文件 C，在工程 P 中获取方法 M 的指令数组，记为 *code*，*code* 首元素必定为一个指令。

(3) 遍历 *code*，根据虚拟机字节码指令规范(下称指令规范)将字节数组 *code* 中的信息翻译到 *instruction* 中，步骤如下：

① 建立一个跟 *code* 等长的数组，记为 *instruction*，建立游标 $index = 0$。

② 将 $code_{index}$ 位置元素拷贝至 $instruction_{index}$ 中，查找指令规范翻译 $code_{index}$ 为 i_0，查找指令规范得到 i_0 的附带操作数有 *len* 个字节，$code_{index+1}$ 至 $code_{index+len}$ 即为 i_0 指令的操作数，将操作数拷贝至 $instruction_{index}$ 元素。

③ 将 *index* 向后移动 $len + 1$ 长度，重复②，直至遍历完 *code*。

步骤(3)过程示意图如图 2 所示。

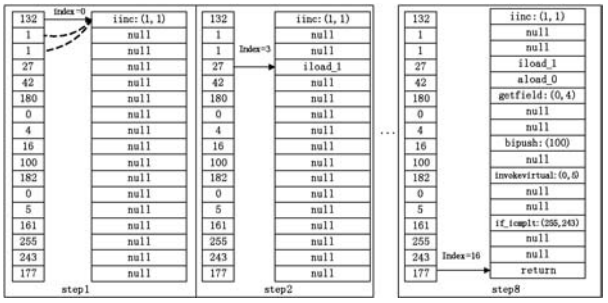

图 2　instruction 生成过程示意图

(4) 遍历 *instruction*,初始化其中所有不为空的元素的前驱和后继,前驱为 0 的元素记为起始节点 *start*,后继为 0 的元素记为终止节点 *end*。记初始化前驱后继后的 *instruction* 为 $feature^{\text{struct}}$。

(5) 遍历 *instruction*,记正在遍历的指令为 i,检查 i 是否为 invoke 系列指令,若为 invoke 指令且 $R_{\text{temp}} \leqslant R_{\max}$,则检查 i 调用的方法 M'是否存在于工程 P 中,若存在则对 M'生成签名 M'_{sign},记 M'所在的 class 文件为 C',$R_{\text{temp}} = R_{\text{temp}} + 1$,以 M'_{sign}和 C'作为输入,重复步骤(2),生成 M'的控制流特征 *instruction'*,将 *instruction'* 的首元素添加到 i 的后继。遍历 *instruction* 完成后 $R_{\text{temp}} = R_{\text{temp}} - 1$。

对如图 3 所示的示例程序使用本文提出的胎记提取算法提取出的胎记如图 4 所示。

```
public class TestCase {
    public void Entrance(int input) {
        testFor(input);
        testIf(input);
    }

    public void testFor(int input) {
        for (int i = 0; i < input; testSwitch(i), i++);
    }

    public int testIf(int input) {
        return (input < 100) ? 99 : 101;
    }

    public int testSwitch(int input) {
        switch (input) {
            case 0: return 0;
            case 1: return 2;
            case 2: return 4;
            case 3: return 6;
            default: return -1;
        }
    }
}
```

图 3　示例程序

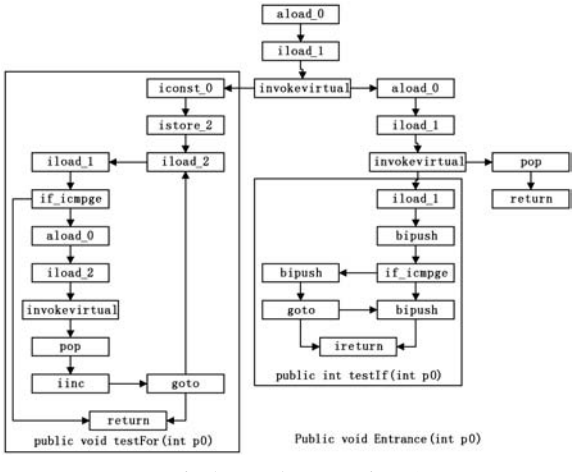

图 4　程序胎记示意图(迭代深度为 2)

2.2　程序控制流胎记的约简

为了对源程序和可疑程序的胎记进行对比,需要建立一种以图同构算法为核心的对比模型,而图同构问题是 NP 问题,为此,本文提出一种约简程序控制流胎记的方法。

(1) 将胎记 B 中 *instruction* 的所有元素的状态初始化为 MEANINGLESS。

(2) 迭代遍历胎记 B 中的 *instruction*,设置所有前驱数目不为 1 或后继数目不为 1 的元素状态为 MEANINGFUL,特别地,若元素 i 的后继数目大于 1,则将 i 的每个后继元素的状态也设置为 MEANINGFUL。

(3) 迭代遍历胎记 B 中的 *instruction*,记正在遍历的元素为 i,若 i 的状态为 MEANINGLESS,则将 i 的唯一后继 *suc* 加入至其前驱 *pre* 的后继集中,将 i 从其前驱 *pre* 的后继集中删除,再将 i 删除。

2.3　使用程序控制流胎记进行抄袭检测

本文提出一种使用程序控制流胎记进行抄袭检测的方法:

(1) 用户选定源工程 P_{src}与可疑工程 P_d,源类 C_{src}和源方法签名 M_{src}^{sign},可疑类 C_d和可疑方法签名 M_d^{sign},迭代深度 $R_{\max}$,检测阈值 α,净输入的放大率 θ 和净输入的偏置 *bias*。

(2) 根据 2.1 节所述方法及步骤(1)中参数,生成源胎记 B_{src}与可疑胎记 B_d。

(3) 根据 2.2 节所述方法及 B_{src}与 B_d生成约简胎记 B_{src}^{reduce}与 B_d^{reduce}。

(4) 选定 B_{src}^{reduce}与 B_d^{reduce}中节点较少的作为 G_{query},若两图节点数目一样,则选 B_{src}^{reduce}作为 G_{query},使用 VF2 算法匹配 B_{src}^{reduce}与 B_d^{reduce}的最大同构子图,记为 G_{common}。

(5) 使用式(2)计算 M_{src}与 M_d的相似度 $sim_{src,d}$,按照抄袭判定标准和检测阈值 α 给出最终结论[11]。

$$\begin{cases} sim_{src,d} = \dfrac{1}{1 + e^{\theta \cdot net - bias}} \\ net = \dfrac{N_{\text{common}}}{(N_{src} + N_d)} \end{cases} \tag{2}$$

3　实验验证

本文选择了 4 款在业界广泛使用的开源测试与分析工具 Junit、JMeter、selenium、findbugs,设置迭代深度 $R_{\max} = 3$,检测阈值 α 设置为 0.3,放大率 θ 为 10,偏置 *bias* 为 -2。

本次实验检测同一软件不同版本之间的对应方法对与不同软件之间的随机抽取方法,对于某软件,在由

版本 1 演进至版本 2 时,较多数类及方法将会保留,故同一软件对应不同版本间的对应方法默认结果为抄袭,即阳性。对不同软件间的随机抽取方法,默认其为独立开发,即阴性。

为验证本文提出的胎记可信性,抽取 8 对阳性程序与 4 对阴性程序检测其实验结果,如表 1 所示。

表 1 程序控制流胎记可信性结果验证

受试程序对	本文	SKB
assertEquals(Junit4.0&4.3)	0.97	0.94
runBare(Junit4.1&4.2)	0.93	0.92
makeAlias(Jmeter3.1&3.2)	0.95	0.90
writeHeader(Jmeter3.2&3.3)	0.86	0.83
getCurrent(selenium3.11&3.12)	0.68	0.65
Merge(selenium3.12&3.14)	0.93	0.93
Parse(findbugs4.4&4.6)	0.90	0.90
perform(findbugs4.5&4.7)	0.73	0.68
runBare_junit4.2&makeAlias_Jmeter3.1	0.26	0.28
writeHeader_Jmeter3.2&merge_selenium3.12	0.19	0.23
getCurrent_selenium3.12&parse_findbugs4.4	0.31	0.34
Perform_findbugs4.5&assertEquals_Junit4.0	0.17	0.21

由表 1 可以看出:本文提出的胎记对受试的 14 对程序出现 2 个误判,正确率为 85.7%,精确率为 87.5%,召回率为 87.5%,F 度量为 87.5;而 SKB 对受试的 14 对程序出现 3 个误判,正确率为 78.6%,精确率为 85.7%,召回率为 75%,F 度量为 0.8,两种胎记受试结果对比如图 5 所示。

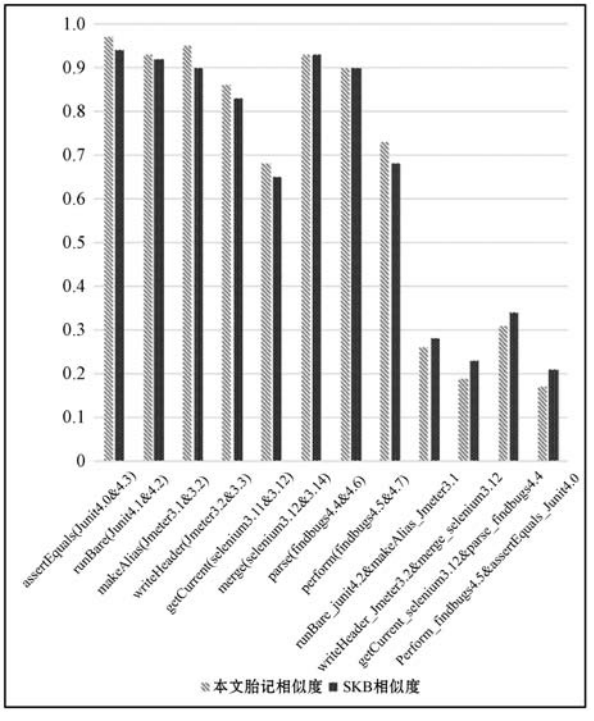

图 5 程序控制流胎记与 SKB 效果对比图

为了验证本文提出的胎记的健壮性,从 4 个软件中随机寻找各 50 个方法,使用 proGuard[12] 对共计 200 个受试对象进行迷惑,使用本文提出的胎记对这 200 对程序对进行抄袭检测,结果如表 2 所示。

表 2 软件胎记健壮性结果验证

源程序	相似度均值	正确率	F 度量
Junit	0.88	0.90	0.95
Jmeter	0.78	0.86	0.92
FindBugs	0.73	0.84	0.91
selenium	0.91	0.94	0.97

由于是健壮性检测,故受试程序对的实际结果都是阳性,不存在 FP 与 FN,对共计 200 对经过迷惑的程序对出现 23 对误判,整体正确率为 88.5%,说明本文提出的胎记具有一定的健壮性。

4 结 语

本文通过静态分析程序的指令流获取程序的控制流软件胎记,使用 VF2 算法来计算胎记的相似度,对常见的四种代码检测和分析工具进行胎记可信性和健壮性对比实验。实验结果表明,本文提出的胎记较传统 SKB 胎记更具可信性,由于本文的胎记是以中间指令分析为基础,所以更具广泛适用性,但由于本方法未考虑节点间的标签,所以仍存在语义丢失的可能。下一步将在两节点匹配时考虑节点内部指令序列的相似对比因素。

参 考 文 献

[1] Collberg C, Thomborson C D. Software watermarking: Models and dynamic embeddings[C]//Proceedings of Symposium on Principles of Programming Languages,1999:311 - 324.

[2] Tamada H, Nakamura M. Detecting the theft of programs using birthmarks[R]. Information Science Technical Report NAIST-IS-TR2003014 ISSN 0919 - 9527, Graduate School of Information Science, Nara Institute of Science and Technology, 2003.

[3] 陈林,刘粉林,芦斌,等. 基于 k-gram 频数的静态软件胎记[J]. 计算机工程,2011,37(4):46 - 48.

[4] Xie X, Liu F L, Lu B, et al. A software birthmark based on weighted k-gram[C]//IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2010), Xiamen, China, 2010: 400 - 405.

[5] Schuler D, Dallmeier V, Lindig C. A dynamic birthmark for java[C]//Proceedings of the 22nd IEEE/ACM International

Conference on Automated Software Engineering. ACM,2007: 274 - 283.

[6] Park H, Choi S, Lim H I, et al. Detecting Java theft based on static API trace birthmark[M]//Advances in Information and Computer Security. Springer Berlin Heidelberg, 2008.

[7] Cordella L P, Foggia P, Sansone C, et al. A(sub)graph isomorphism algorithm for matching large graphs[J]. IEEE Trans Pattern Anal Mach Intell,2004,26(10):1367 - 1372.

[8] Grover D. Program Identification [M]. The Protection of Computer Software-Its Technology and Applications, Cambridge University Press, Cambridge, 1989: 119 - 150.

[9] Myles G, Collberg C. K-gram based software birthmarks [C]//Proceeding of ACM Symposium on Applied Computing. ACM, 2005: 314 - 318.

[10] Asadollah S A, Sundmark D, Eldh S, et al. 10 Years of research on debugging concurrent and multicore software: a systematic mapping study[J]. Software Quality Journal,2016, 25(1): 1 - 34.

[11] 王曙燕,赵鹏飞,孙家泽. 基于多特征的静态软件胎记提取算法[J]. 计算机应用,2018,38(3):806 - 811.

[12] Kim D, Gokhale A, Ganapathy V, et al. Detecting plagiarized mobile apps using API birthmarks[J]. Automated Software Engineering, 2015, 23(4): 1 - 28.

(上接第 27 页)

去除压缩解压的时间,同步时间与文件大小成线性关系,传输速度在 10 MB/s 以上,测试环境为百兆带宽。

对资源集成框架中服务调用以及数据、文件同步进行了性能测试,结果表明:SCF 在并发性和响应时间方面要优于 RMI 框架,而且 SCF 可以整合多种语言开发的服务,不仅限于 Java。在资源占用率方面 SCF 也有着出色的表现。在数据同步和文件传输方面,可以支持海量的时空数据同步和文件传输,数据条数和文件大小不构成传输瓶颈,而且传输带宽利用率可达 90% 以上。

3　结　语

本文根据信息资源共享平台的需求,提出了一种面向时空大数据资源集成框架,为实现大规模时空数据资源和服务应用的整合提供了理论依据和技术基础。实验表明各系统应用可以利用此套资源集成框架高效稳定地完成数据交换和服务互操作,实现系统的互联互通。此外,通过性能测试将服务集成模块的通信框架 SCF 与 RMI 框架进行比较分析,结果表明基于 SCF 在应用于实际工程时可支持较高的并发而且响应更为快速。通过对数据资源集成模块的性能测试,资源集成框架可以提供高效安全的数据库及文件同步方式。本文研究成果已投入卫星地面系统等大型工程实践,时空大数据资源集成框架能够满足系统需求且性能良好。

参　考　文　献

[1] 夏日,王宗宝. 近十年来我国信息资源整合研究综述[J]. 情报科学,2015(2):154 - 160.

[2] 李为为,孙玉光,周汝胜. 省级交通地理信息数据资源整合方案探讨[J]. 中国交通信息化,2016(6):16 - 22.

[3] 王海. 探讨 GIS 技术在数字城市建设中的应用[J]. 建筑工程技术与设计,2016(14):433.

[4] Martinek P, Szikora B. SOA based web service adaptation in enterprise application integration[J]. Electrical Engineering, 2009, 53(3/4).

[5] 麻志毅,陈泓婕. 一种面向服务的体系结构参考模型[J]. 计算机学报,2006,29(7):1011 - 1019.

[6] Lee J, Park M S. Integration and composition of web service-based business processes[J]. Journal of Computer Information Systems, 2003,44(1):82 - 92.

[7] Sessions R . COM and DCOM-Microsoft's vision for distributed objects[M]. John Wiley & Sons, Inc. 1997.

[8] Downing T B . Java RMI: remote method invocation[M]. IDG Books Worldwide, Inc. 1998.

[9] 沈敏. 基于 XML 的分布式异构数据库数据同步系统的研究与开发[D]. 上海:上海大学,2005.

[10] 刘永毅. 异构分布式数据库远程数据同步的研究与设计[D]. 长春:吉林大学,2010.

[11] Leader-us. ZeroC Ice 权威指南[M]. 北京:电子工业出版社,2015.

[12] Juric M B, Rozman I, Brumen B, et al. Comparison of performance of Web services, WS-Security, RMI, and RMI - SSL[J]. Journal of Systems and Software,2006,79(5):689 - 700.

[13] Kaushik M. Research of load testing and result based on loadrunner[J]. International Journal for Multiscale Computational Engineering, 2014, 6(4):301 - 310.

[14] Quang D N, See O H, Nga D V, et al. Performance testing framework in a heterogeneous and hybrid smart grid communication network[J]. Research Journal of Applied Sciences Engineering & Technology, 2013, 6(23):4506 - 4518.

[15] Yi L I, Zhou G X. New flow of performance testing based on LoadRunner[J]. Application Research of Computers, 2009, 26(11):4143 - 4145.