一种面向软件仓库挖掘的动态作业配置框架

史殿习 尹 刚 米海波 袁 霖 王怀民
(国防科学技术大学计算机学院 长沙 410073)

摘 要 构造面向软件仓库挖掘的数据中心,是目前软件工程领域的研究热点。软件仓库数据处理作业的执行时间差异明显、资源消耗大等特点为其作业配置带来诸多挑战。提出一种面向软件仓库挖掘的作业配置框架TrustieSDC,该框架支持一种新型远程作业部署和服务模式,采用一种基于软件版本划分的动态作业配置算法以缩短长作业响应时间并提高系统资源利用率。基于Gnome项目SVN库的实验表明,TrustieSDC的性能和资源利用率与并行后的Alitheia相比有明显改进。

关键词 软件仓库挖掘,数据中心,作业配置,开发者贡献度,开发者网络

Research on Dynamic Job Configuration Framework for Mining of Software Repositories

SHI Dian-xi YIN Gang MI Hai-bo YUAN Lin WANG Huai-min
(School of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract Construction of datacenters for mining of software repositories(MSR) is a hot topic in current software engineering area. Data processing jobs for software repositories are highly diverse in execution-time and resource-consuming, which bring many challenges for the job configuration in such environments. This paper proposed a job configuration framework named TrustieSDC for mining of software repositories. TrustieSDC supports a new paradigm for remote deployment and execution of MSR jobs, and proposes a software-subversion-partition based job configuration algorithm to cut the response time of long jobs and increase the resource utilization. The experiments based on SVN repositories of Gnome projects shows that compared with paralleled Alitheia system, TrustieSDC gains remarkable improvement on both performance and resource utilization.

Keywords Ming of software repositories, Data center, Job configuration, Developer contribution, Developer network

1 引言

近年来,软件工程领域研究人员逐渐意识到软件开发过程中产生的大量数据对更好地理解软件开发过程、提高软件开发效率和质量具有重要意义。软件仓库数据挖掘(Mining of Software Repository, MSR)技术通过对软件项目版本库、邮件列表、缺陷库等软件仓库数据进行统计和分析实现软件评估或预测,是目前软件工程领域的研究热点。

但是异构多样、复杂海量的软件仓库数据为其管理和应用带来了巨大挑战。例如,Mockus花费6年时间统计了世界各大主要开源社区的项目数据[1],其中SourceForge社区包含了121389个项目版本库、超过2600万个文件以及8100万个版本信息,总数据量超过820G;而git.kernel.org社区中包含595个版本库,却有超过1200万个文件及970万版本信息。此外,单个软件项目SVN库的数据量往往从10M到10G不等,版本数目从1000至100万不等。又例如,规模较大的Eclipse项目的SVN数据约2~3G,而一个中等规模的Apache-Http-Server项目则有986168个版本。同时,SVN库等软件仓库数据并不是结构化数据,常用的数据挖掘算法难以对其执行检索、观察和分析等操作。目前国际多个研究团队已开始关注面向MSR的海量数据处理问题。Flossmole[2]和Alitheia[3,4]等研究平台提供了初步的软件仓库数据存储和管理能力,得到了广泛关注。但建立公共的软件数据中心尚面临两方面挑战:(1)对软件仓库数据的获取、解析和存储(如解析并存储到关系数据库)等工作是采用常用数据挖掘方法和工具对其进行分析的前提,但对这类规模大且结构复杂的数据的处理任务往往需要大量的计算资源和处理时间;(2)由于不同领域、不同用户的MSR算法具有多样性而且差别很大,因此软件数据中心不可能提供所有挖掘应用,也难以合理配置资源进而高效地完成各类挖掘请求,因此软件数据中心采用何种资源共享模式实现灵活的应用部署和请求调度,以有效支持科学实验和未来可能的商用软件数据挖掘,是该方向面临的挑战。针对以上问题,本文试图研究一种新的软件数据中心体系结构及其应用模式,并研究一种高效的MSR

到稿日期:2010-08-31 返修日期:2010-12-07 本文受国家863课题(2007AA010301),国家自然科学基金项目(60903043)资助。
史殿习(1966-),男,博士,副研究员,主要研究方向为分布计算、普适计算、信息安全,E-mail:dxshi@nudt.edu.cn;**尹 刚**(1975-),男,博士,助理研究员,主要研究方向为分布计算、信息安全、可信软件;**米海波**(1982-),男,博士生,主要研究方向为云计算和自主计算;**袁 霖**(1982-),男,博士生,讲师,主要研究方向为安全协议分析、可信软件;**王怀民**(1962-),男,教授,博士生导师,主要研究方向为分布计算、人工智能、信息安全、可信软件。

作业动态配置方法，为 MSR 数据平台和应用平台提供可行的方案。

本文第 2 节介绍软件仓库挖掘领域和分布式系统领域的相关工作；第 3 节介绍 TrustieSDC 的软件体系结构及其动态作业配置框架，并提出一种基于 SVN 版本划分和作业资源占用率的动态作业配置算法；第 4 节基于实验对 TrustieSDC 性能和应用效果进行分析。

2 相关工作

目前国际上 MSR 研究的软件仓库数据源主要包括非公开和公开两种。非公开软件仓库主要是企业内部的开发环境积累的软件数据，但是很难有足量的项目开发数据，缺乏有效的数据挖掘样本；一些研究将多个企业的软件项目数据库进行集成[5]，有效增强了知识总量，但其数据一般不公开，研究结论难以得到验证。在这种背景下，自由/开源软件的开发数据成为 MSR 领域研究的重点，这类软件仓库不仅可以公开获得，而且数据量巨大且相对完整。

近年来，许多研究团队开始关注开源软件仓库中相关元数据的收集。Flossmole[2] 利用网络爬虫收集各大开源社区的项目网页中的相关信息，并按社区将数据存储到 MySQL 数据库中，定期通过 SQL 文件的形式在互联网上发布。世界最大开源社区 Sourceforge 也将其所有项目的数据公开给研究人员，注册用户可以通过在线 SQL 查询获取所需信息[6]。SourceForge 提供了丰富的开源软件的软件开发数据，包括一些软件的源代码、项目缺陷数据、文档、开发成员之间的交流信息等，很多学者的软件工程研究均是在其数据集的基础上完成的。但是，Flossmole 的数据库中缺少反映项目开发细节的过程数据（如开发日志和缺陷等），Sourceforge 的 SQL 查询模式会降低数据挖掘研究的效率，普通的数据挖掘过程往往需要大量的人工 SQL 查询。

软件仓库数据的结构化存储方面，很多研究将软件仓库中开发活动的日志和代码文件换为结构化或半结构化的数据。例如，Walker 等人分析软件仓库（CVS）中软件代码文件的内部结构的关联关系和代码修改轨迹，并以图的格式存储[7]；Zimmermann 等人将 CVS 库的版本数据（文件、目录、发布的不同版本以及版本分支）转换为关系数据库表[8]。Alitheia 是初步实现了对软件仓库数据进行自动化解析和存储的开源平台[3,4]，使研究人员可以把精力放在问题研究本身。但是，Alitheia 的性能难以满足大规模软件仓库数据处理的需要。例如，在一台 8 核服务器上对 Gnome 社区中的 48 个软件项目（共 2.5GB，相当于 1 个大型软件项目的版本数据）的解析过程总共花费 12h[3]。此外，Shang 等人研究了基于 MapReduce 的非结构化软件数据的高效索引问题[9]。

3 TrustieSDC 动态作业配置框架

3.1 体系结构

TrustieSDC（Trustie Software Data Center）是一种面向 Trustie 平台的软件数据中心动态作业配置框架，主要面向常用软件仓库数据源的处理和分析，并最终能够提供开放、免费的在线软件数据挖掘服务。Trustie* 是国家 863 计划“十一五”重点项目“高可信软件生产工具及集成环境”的研发成果，是集软件协同开发平台、软件资源库、软件可信分级模型及可信证据框架、软件生产线框架、多谱系软件工具及软件生产线等多项关键技术与核心软件生产要素于一体的网络化可信软件公共生产服务环境，能够以服务形式支持可信软件的大规模协作生产。

TrustieSDC 的设计目标是支持常见软件版本管理系统（如 SVN 和 CVS）、常用邮件列表（如 Mailman 和 Nabble）和缺陷库（如 Bugzilla 和 Mantis）中的开发过程数据和软件代码的分析和挖掘以及结果数据的展现。TrustieSDC 的软件结构基于 OSGi，目前包含 3 类构件（遵循 OSGi 的 bundle 模型的软件模块）：ETL（Extract Transform Load）构件、MSR 构件、UI（用户界面）构件，如图 1 所示。其中 ETL 构件负责将软件项目的数据仓库（如 SVN 库）解析、转换并存储到关系数据库（如 MySQL）中；MSR 构件负责基于关系数据库中的软件数据进行挖掘，产生各种分析结果数据；UI 构件按照不同应用的需求综合显示 MSR 构件的分析结果。每一种构件都有相应的框架设施，支持此类构件的共性操作。作业调度器（Job Scheduler）负责管理各类构件的在线注册、加载、执行以及作业请求的动态调度和配置，是负责作业动态配置框架的核心构件，其主要负责如下两类过程。

（1）构件的远程注册和加载：TrustieSDC 的系统管理员或者远程用户都可以通过 Web 界面将 3 类构件注册到系统中，并可以通过 Web 请求远程执行已注册构件的各种功能。TrustieSDC 可成为一种公开、可成长的 MSR 服务平台。

（2）作业请求的动态调度和配置：TrustieSDC 可能面对大量数据解析、数据存储和数据挖掘的作业请求。这些请求的执行时间变化幅度大，可能从几秒到几天；而且请求密集时系统需要较为准确地根据资源使用情况将请求队列合理地配置到不同的节点上执行。作业调度器将对作业请求队列中的作业进行分析，对长作业进行切片，然后成批地将作业配置到节点集合。

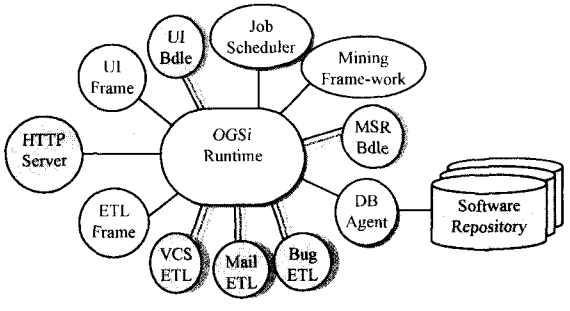

图 1 TrustieSDC 静态软件体系结构

目前 TrustieSDC 基于 Alitheia 的 SVN 解析引擎实现了适合大规模数据处理（如转换和挖掘）作业的并行解析引擎和作业请求的动态配置机制，并在此基础上开发了人员贡献度、提交者网络等 MSR 构件。

3.2 系统模型

本文将基于上述软件体系结构建立的一个系统实例记为 TrustieSDC。该系统实例可以接收用户的各种作业请求，并将请求分配到不同节点上执行，同时允许用户开发新的作业

* Trustie 平台网址：http://www.trustie.net

构件并加入到系统中。针对系统的动态作业配置问题，TrustieSDC 可以抽象为两个基本集合：

TrustieSDC=(Jobs,SubJobs,Nodes)

式中，Jobs 是系统接收到的所有作业请求组成的集合；SubJobs 是 Jobs 拆分得到的子作业集合；Nodes 是系统中所有节点组成的集合。Jobs 中的每一个作业 $job \in$ Jobs 可以表示为元组 $job(cpu,svn,start)$，其中 $job.cpu$ 是作业 job 的 CPU 占用率(即此类作业在执行时的 CPU 使用率平均值，一般是根据此类作业先前执行情况的统计值)；$job.svn$ 是作业 job 需要处理的软件版本库中的标准软件版本数(即根据版本库包含的版本数和数据量综合计算得到的一种相对的版本数)；$job.start$ 是其首个版本号。为了提高系统性能和 CPU 利用率，TrustieSDC 需要将长作业(执行时间特别长的作业)拆分为较小的子作业；从而将 Jobs 转换为子作业集合 SubJobs。其中 SubJobs-Jobs 是 Jobs 中的长作业拆分得到的新的子作业，Jobs-SubJobs 是 Jobs 中的长作业集合；Jobs∩SubJobs 是没有经过拆分的短作业。SubJobs 中的每个子作业 $sub \in$ SubJobs 可以表示为 $sub(cpu,start)$，其中 $sub.cpu$ 是子作业 sub 的 CPU 占用率，该值等于其对应作业的 CPU 占用率；$sub.start$ 是其首个版本号。Nodes 中的每一个节点 $n \in$ Nodes 表示为元组 $n(usage,subs)$，其中 n 是节点的序号(或名称)，$n.usage$ 表示节点 n 的 CPU 利用率，$n.subs$ 表示节点 n 正在执行的子作业列表。

定义 1(作业配置) 软件数据中心系统实例 TrustieSDC 的作业配置记为 JobCon：

JobCon⊆SubJobs×Nodes

JobCon 中的每个元组 $(subs,node)$ 表示子作业集合 $subs$ 将被配置到节点 $node$ 上执行。JobCon 由 TrustieSDC 的作业配置算法计算得到。当 SubJobs 中的子作业都完成配置后，有 SubJobs=$\bigcup_{n \in Nodes} n.subs$。

定义 2(作业拆分) 给定一个作业 $job \in$ Jobs、大作业阈值 α 和作业划分均值 β。如果 $job.svn > \alpha$，则 job 将按如下步骤被拆分为子作业集合 $subs$：

1)如果 $job.svn > \beta$，则创建子作业 sub；

2)$sub.start = job.start$；$sub.cpu = job.cpu$；

3)$job.start += \beta$；$job.svn -= \beta$；

4)返回第 1)。

3.3 作业配置算法

针对 Gnome 的大量开源项目 SVN 库的数据统计和初步实验发现，SVN 库版本数目及其数据量与 SVN 作业的执行时间存在一定的正比关系，而且不同软件数据处理作业的 CPU 利用率基本保持在较为稳定的水平。这些发现为缩短长作业的响应时间和提高 CPU 资源利用率提供了重要参考。TrustieSDC 将作业按照其覆盖软件版本的数目进行划分，并根据不同类型作业的 CPU 占用率在同一节点并发执行不同的作业。此方法由 SPCA 算法实现。首先给出 SPCA 算法描述中使用到的符号：MaxLEN 是作业中项目数据相对版本数的临界值(对应于定义 1 中 α)；AvgLEN 是拆分作业时采用的标准平均版本数(对应于定义 1 中 β)；MaxCPU 是节点 CPU 的利用率的上限，只有当 CPU 的利用率小于 MaxCPU 时才可能在该节点上继续配置作业。SPCA 算法的基本步骤如下。

SPCA：Subversion-Partition based Configuration Algorithm

1. 对 Nodes 按当前 CPU 利用率降序排列；
2. 对 Jobs 按作业请求的 CPU 利用率降序排列；
3. 对 Jobs 中的每个作业 job，执行 4—7：
4. 　如果 $job.svn$ > MaxLEN，则：
5. 　　将 job 按 AvgLEN 拆分为 $jobs$；
6. 　　将 $jobs$ 加入 SubJobs；
7. 　否则将 job 加入 SubJobs；
8. 对 SubJobs 中的每个子作业 sub，执行 9—12：
9. 对 Nodes 中的每个节点 n，执行 10—12：
10. 　　如果 $n.usage + sub.cpu <$ MaxCPU，则：
11. 　　$n.usage$ += $sub.cpu$；
12. 　　将 sub 加入 $n.subs$；break；

当 TrustieSDC 系统累计接收到一批作业请求后(此前系统处理上一批作业队列)，SPCA 算法将由作业调度器执行，结果是 Jobs 中作业被分配(可能先拆分)到 Nodes 中执行。

4 实验分析

4.1 系统配置

实验环境由 9 台主机模拟一个小型的数据中心，其中 8 台作业服务器、1 台 Web 服务器和作业调度器。作业服务器中，4 台是 CPU 分别为 Intel(R) Core(TM)2 Duo 2.83GHz 的 Dell 主机，另外 4 台是 CPU 为 AMD Athlon(tm) 64 X2 3600+ 1.9GHz 的 lenovo 主机。作业服务器的操作系统皆为 Red Hat 2.6.24.3，TrustieSDC 的作业拆分机制和子作业处理实验机制基于 Alitheia 0.95 开发完成。

实验中用到的 10 个作业构件包括 Alitheia 开发的软件项目数据加载构件、软件数据测度分析构件以及 TrustieSDC 新开发的两种构件(分别为人员贡献度挖掘和开发者网络分析)。为了增强实验结果的说服力，本实验的软件仓库数据采用 Gnome 社区的 12 个开源软件项目的 SVN 库，项目数据的详细信息见表 1。实验作业请求的构造采用作业构件与 Gnome 项目随机匹配的方法产生，每次实验使用包含 100 个作业的队列。图 2 为一种基本的 TRUSTIE-SDC 实验系统配置实例。

表 1 用于实验的开源软件项目

序号	软件项目名	版本数	SVN 库大小/MB
1	Gnome-Doc-Utils	1122	8.84
2	Gnome-VFS	5550	72.0
3	Gnome-Utils	8123	99.8
4	Gnome-Menus	959	5.57
5	Eel	2175	21.7
6	Evolution-Data-Server	9569	188
7	Gnome-Edit	6556	97
8	Gnome-Applets	11008	219
9	Gnome-Control-Center	8982	103
10	Gnome-Desktop	5242	69.9
11	Gnome-Media	3993	44.5
12	Gnome-Terminal	3050	31.3

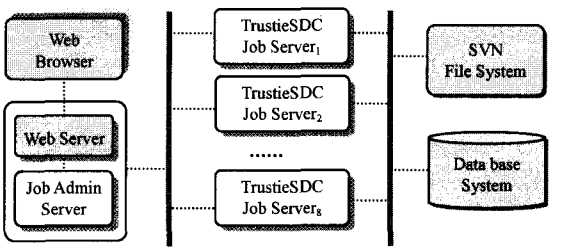

图 2 一种基本的 TRUSTIE-SDC 实验系统配置实例

为了比较分析 TrustieSDC 的性能和 CPU 利用率，本文将 SPCA 与一种经过并行处理的 Alitheia 系统进行比较，该系统简称为 EPSA(Event-based Parallel Scheduled Alitheia)。EPSA 作业调度的基本思路是：将 Alitheia 并行地部署在多个节点上，每当一个节点的作业执行完毕后，就把作业队列中的位于队首的作业分配到该节点执行。

4.2 实验 1：作业响应时间分析

图 3 显示了 SPCA 和 EPSA 处理不同规模作业的响应时间，其中横坐标表示按作业规模排列的作业序号，纵坐标表示作业的响应时间。为了充分说明在处理长作业时 SPCA 的优势，图 3 将 Jobs 重新按规模升序排序，例如图中第 91 个作业在 Jobs 中的顺序可能是 20。从中可以看出，当作业规模较小(如版本数大约小于 5000)时，每个作业的执行方式本质上为串行，所以响应时间 SPCA 与 EPSA 没有明显差别；但当作业规模较大时，SPCA 将其分裂为若干个子作业并行执行，所以其执行时间较 EPSA 有了大幅度减少。

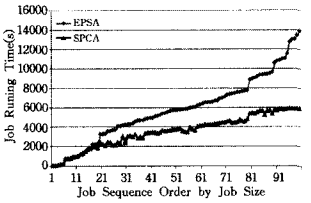

图 3 作业响应时间对比图

4.3 实验 2：资源利用率分析

图 4 显示了分别采用 EPSA 与 SPCA 时集群 CPU 的平均利用率，其中横坐标表示实际作业排列中的作业序号，纵坐标表示该作业在执行时集群的 CPU 平均利用率。从中可以看出，除了在作业处理的开始阶段，SPCA 的 CPU 利用率一直是 EPSA 的两倍多。在整个 Job 处理时间内，采用 SPCA 集群 CPU 平均利用率为 72.9%，而采用 EPSA 的 CPU 利用率仅为 34.1%。这说明了 SPCA 可以根据项目数据相对版本数分解作业规模，充分利用集群资源加速并行化执行的优势。这也意味着，处理临等量作业请求时 SPCA 可更快地完成作业队列，而且较高的资源利用率往往能够有效降低数据中心的能耗。

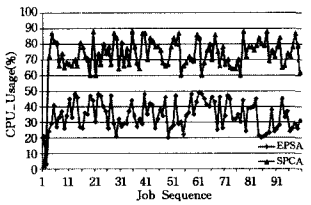

图 4 CPU 利用率对比图

4.4 实验 3：数据挖掘实例分析

图 5—图 8 分别显示了基于 TrustieSDC 新增的人员贡献度挖掘作业和开发者网络分析作业执行的结果。图 5 表明 Gnome-menus 项目代码的修改大部分工作由某几个少数开发人员完成，这印证了 Mockus 等人的“软件项目开发中超过 80%的工作由不到 4%的少数开发人员完成”的结论[10]。图 6 表明翻译文件的提交行为也有类似特征。图 7 和图 8 是两个项目的协同开发关系图。其中顶点代表开发人员，边表示人员间协同关系，边上的权重代表协同值。其中 Gnome-Doc-Utils 的提交者网络中边的最大权值(协同次数)为 18，而在 Gnome-Menus 中为 87。该结果可用于快速发现项目开发中的核心人员。

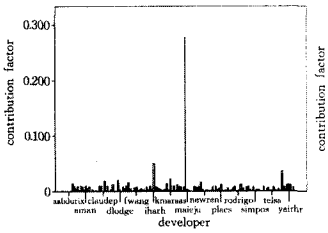

图 5 Gnome-Menus 代码修改行为柱状图

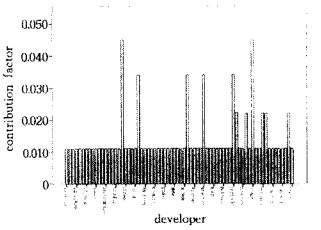

图 6 Gnome-VFS 翻译提交行为柱状图

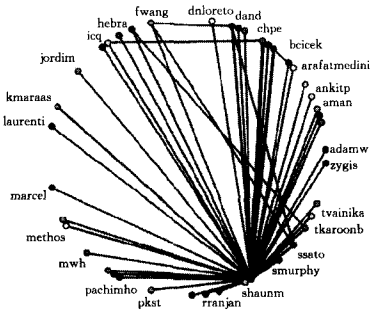

图 7 Gnome-Doc-Utils 协同开发关系图

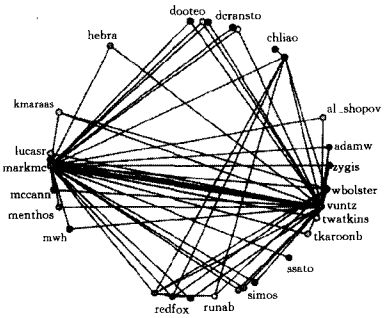

图 8 Gnome-Menus 协同开发关系图

结束语 互联网中海量的开源软件版本数据和开发过程数据为软件的可信度量提供了宝贵的数据资源。但是，如何有效利用这些数据，是目前 MSR 领域面临的瓶颈问题，也对现有数据中心动态配置方法提出了挑战。本文结合国际最新研究进展，基于国家 863 重点项目科技成果 Trustie 协同平台，设计了一种面向软件仓库挖掘的数据中心体系结构 TrustieSDC 及其作业管理框架，提出了一种高效的动态作业配置算法。实验结果表明，相比经过并行处理后的 Alitheia 引擎架构，TrustieSDC 在 CPU 利用率、作业响应时间等方面有明显改进，为软件数据挖掘的科学实验和未来实际应用提供了具有前景的运行平台构造方法。

参 考 文 献

[1] Mockus A. Amassing and indexing a large sample of version control systems: towards the census of public source code history [C]// 6th IEEE Working Conference on Mining Software Repositories. May 2009: 16-17

[2] Howison J, Conklin M, Crowston K. Flossmole: A collaborative repository for floss research data and analyses[J]. International Journal of Information Technology and Web Engineering, 2006, 1(3): 17-26

(下转第 133 页)

确结果表决通过有15.3557%(IAW),不正确结果没有通过表决的有84.6443%(IAF);输出结果表决通过的占78.1832%,表决没有通过的占21.8168%。

分析实验数据可知,在输出结果空间基数 r 较小时,通过表决而不正确的结果占比比较大(18.8931%),误报率(false positive rate)会比较高(23.5460%)。通过算法的改进,误报率降低,即产生 IAW 的可能性降低了 8.1903%。通过实验表明改进算法是有效的,提高了系统的安全性和可靠性。

表 1 是实验 1 和实验 2 的一致表决算法和自适应一致表决算法实验数据。两个实验都表明算法改进后误报率下降,说明通过改进算法提高了系统的安全性和可靠性。而 CAA 在实验 1 中有所下降,在实验 2 中则有所上升,说明改进后的算法更适应正确结果占比更高的情况。

表 1 一致表决算法和自适应一致表决算法实验数据

试验	实验 1			实验 2		
	CVA	ACVA	Incr.	CVA	ACVA	Incr.
Correct %	81.1069	81.1069	0.0000	93.0615	93.0615	0.0000
Incorrect %	18.8931	18.8931	0.0000	6.9385	6.9385	0.0000
CAA %	95.0683	92.8182	−2.2501	99.7875	99.8086	0.0211
IAW %	23.5460	15.3557	−8.1903	2.4410	1.2940	−1.1470
CAF %	4.9317	7.1818	2.2501	0.2125	0.1914	−0.0211
IAF %	76.4540	84.6443	8.1903	97.5590	98.7060	1.1470
Agreement %	81.5556	78.1832	−3.3724	93.0331	92.9731	−0.0599
Disagreement %	18.4444	21.8168	3.3724	6.9669	7.0269	0.0599

结束语 一致性表决算法在 N 版本程序设计中得到了广泛的应用。一致表决算法适合输出结果空间基数小的情况,因此产生 IAW 错误的可能性会更高。针对这个问题,本文提出了自适应一致性表决算法。实验表明,改进后的算法降低了不正确结果通过表决的概率,提高了系统的安全性和可靠性。

参 考 文 献

[1] Lorczak P R, Caglayan A K, Eckhardt D E. A Theoretical Investigation of Generalised Voters[C]// IEEE 19th Int. Ann. Symp. on Fault-Tolerant Computing Systems. Chicago, June 1989: 444-451

[2] Kanekawa K, Maejima H, Kato H, et al. Dependable On-board Computer Systems with a New Method: Stepwise Negotiated Voting[C] // IEEE 19th Int. Ann. Symp. on Fault-Tolerant Computing Systems. Chicago, June 1989: 13-19

[3] Bass J M. Voting in Real-tTime Distributed Computer Control Systems[D]. Sheffield, UK: Department of Automatic Control and System Engineering, The University of Sheffield

[4] Gersting J L, Nist R L, Roberts D B, et al. A Comparison of Voting Algorithms for N-version Programming[C]// IEEE 24th Ann. Hawaii Int. Conf. on Systems Sciences. Kauai, HI, USA, Jan. 1991, 2: 253-62

[5] Scott R K, Gault J W, McAllister D F, et al. Investigating version dependence in fault-tolerant software[R]. AGARD 361. 1984: 201-210

[6] Knight J C, Leveson N G. An experimental evaluation of the assumption of independence in multisession programming [J]. IEEE Trans. Software Engineering, 1986, SE-12: 96-109

[7] Bishop E, Humphreys B, Lahti D. PODS-A project on diverse software[J]. IEEE Trans. Software Engineering, 1986, SE-12: 929-940

[8] Eckhardt K, Knight C, McAllister V. A large scale second generation experiment in multi-version software: Description and early results[C]// Proc. FTCS 18. June 1988: 9-14

[9] McAllister D F, Sun C-E, Vouk M A. Reliability of Voting in Fault-tolerant Software Systems for Small Output-spaces[J]. IEEE Log, 1990, 39(5): 524-534

[10] Latif-Shabgahi C, Bennett S. Adaptive Majority Voter: A Novel Voting Algorithm for Real-time Fault-tolerant Control Systems [C]// Proceedings of 25th EUROMICRO Conference. Milan, Italy, September 1999, 2: 113-120

(上接第 116 页)

[3] Gousios G, Spinellis D. A Platform for Software Engineering Research[C] // 6th IEEE International Working Conference on Mining Software Repositories. 2009: 31-40

[4] Gousios G, Spinellis D. Alitheia Core: An extensible software quality monitoring platform [C]// Proceedings of the 31st International Conference on Software Engineering(ICSE '09)—Formal Research Demonstrations Track. 2009: 579-582

[5] Tsunoda M, Monden A, Yadohisa H. Productivity Analysis of Japanese Enterprise Software Development Projects [C]// Proceedings of the International Workshop on Mining Software Repositories. 2006: 14-17

[6] Van Antwerp M, Madey G. Advances in the SourceForge Research Data Archive [C]// The 4th International Conference on Open Source Systems(WoPDaSD 2008). Milan, Italy, September 2008

[7] Walker R J, Holmes R, Hedgeland I, et al. A Lightweight Approach to Technical Risk Estimation via Probabilistic Impact Analysis [C]// Proceedings of International Workshop on Mining Software Repositories. 2006: 98-104

[8] Zimmermann T, Zeller A, Weissgerber P, et al. Mining version histories to guide software changes [J]. IEEE Transactions on Software Engineering, 2005, 31(6): 429-445

[9] Shang Wei-yi, Jiang Zhen-ming, Adams B, et al. MapReduce as a general framework to support research in mining software repositories(MSR) [C]// 6th IEEE International Working Conference on Mining Software Repositories. 2009: 21-30

[10] Mockus A, Fielding R F, Herbsleb J. A case study of open source development: The apache server [C]// 22nd International Conference on Software Engineering. Limerick, Ireland, June 2000: 263-272

[11] Lim S L, Quercia D, Finkelstein A. StakeNet: Using Social Networks to Analyse the Stakeholders of Large-scale Software Projects[C]// 32nd International Conference on Software Engineering. Cape Town, South Africa, 2010: 285-294

[12] Ossher J, Bajracharya S, Lopes C. Automated Dependency Resolution for Open Source Software[C]// 7th IEEE International Working Conference on Mining Software Repositories. Cape Town, South Africa, 2010: 130-140