第43卷第5期 四川大学学报(工程科学版) Vol. 43 No. 5
2011年9月 JOURNAL OF SICHUAN UNIVERSITY (ENGINEERING SCIENCE EDITION) Sept. 2011

文章编号: 1009-3087(2011)05-0126-07

分布式自动化软件测试实现技术研究

杨秋辉,周洪宇,洪 玫,王 松,臧 康
(四川大学 计算机(软件)学院,四川 成都 610065)

摘 要: 为了解决大型软件的分布式自动化测试问题,在分布式持续软件质量保证思想基础上,通过对现有自动化测试框架的分析,提出了一个基于Internet网络资源的分布式自动化软件测试平台,并实现了原型系统。平台集成了软件测试过程中需要的一系列工具,能有效利用Internet上的空闲资源进行大型软件系统的持续集成和测试。为实现测试任务的自动划分和调度,提出了基于空闲时间约束的任务调度算法,并对集合划分问题的ACO算法进行了改进,在其中添加了动态容量监控标准和前置与后置处理过程。通过在原型系统中进行MySQL测试,验证了平台架构和设计的可行性以及算法的有效性。

关键词: 软件自动化测试; 分布式持续质量保证; 软件测试平台; 分布式任务调度; ACO算法
中图分类号: TP311 **文献标志码:** A

Implementation Technology Study of Distributed Automated Software Testing

YANG Qiu-hui, ZHOU Hong-yu, HONG Mei, WANG Song, ZANG Kang
(School of Computer Sci. (Software), Sichuan Univ., Chengdu 610065, China)

Abstract: In order to study the distributed and automated testing of large-scale software, a distributed automated software testing platform (DASTP) was presented, which was based on the analysis of an existing software testing framework and the idea of distributed continuous software quality assurance. The prototype system was implemented. The platform integrated a series of tools required in the software testing, and could use free resources in Internet to complete the continuous integration and testing of large-scale software. A task scheduling algorithm based on part-time constraint and an improved ACO algorithm for set partitioning problem were proposed. These two algorithms can partition testing task into subtasks and then schedule these subtasks automatically. By running MySQL testing in the prototype system, the feasibility of the platform architecture and the effectiveness of the algorithms were verified.

Key words: automated software testing; distributed continuous quality assurance; software testing platform; distributed task scheduling; ACO algorithm

随着计算机软件技术的发展,分布式的大型软件自动化测试成为业界的研究热点。原因有二:一是Internet的普及,基于网络的Web应用系统的规模和复杂度日益增加。这类软件大都是在分布式环境下运行,传统的集中式软件测试方式存在一定的局限性,且虚拟的测试环境和数据难于真实地模拟现实,因而测试效果不够理想。二是全球软件开发已成为一种趋势,使软件的生成从集中式到分布式,需要在任何时候、任何地点实现软件的持续集成和测试,这一软件开发的新模式,需要有新的软件测试方法和工具支持。

针对软件测试中的一系列问题,国内外学者已经提出一系列的测试工具和框架。按照实现方式划分,现有的测试框架可以分为基于本地的集中式软件测试框架、基于分布式的软件测试框架两大类。

基于本地的集中式软件测试框架,通常只是针对某种类型或某种架构方式的软件进行测试,并且该类测试框架是在本地自动生成测试用例并运行,测试成本高、效率低。例如,Li等提出了一个针对模型转换的测试框架[1],Stocks和Carrington提出了一个基于规格说明书的测试框架TTF[2],Hartman等提出了一个以模型为基础的测试工具AGEDIS[3],Marinov和Khurshid等提出了一个针对Java

收稿日期: 2010-10-17
作者简介: 杨秋辉(1970-),女,副教授,博士. 研究方向: 软件工程; 软件质量保证与测试.

程序的新型自动化测试框架，实现了测试用例的自动生成和运行[4]。

基于分布式的软件测试框架，将测试任务分发给各测试终端，然后再收集测试结果并处理。Porter[5]等提出了一个分布式软件测试框架 Skoll，利用 Internet 上的空闲客户端资源建立软件自动化测试平台。在业界，已经有很多公司开发了该类测试框架。如 Cloud Testing、Kite 和 SOASTA 等，但这些框架只能用于 Web 应用程序的性能测试和负载测试。STAF 是由 IBM 开发的开源、跨平台、支持多语言并且基于可重用组件来构建的自动化测试框架，但 STAF 对持续软件质量保证的支持有限，且生成的测试报告和日志不够直观[6]。Zhu 等[7]提出了分布式环境下实时软件的自动化回归测试框架，该框架能根据特定需求动态的改变系统配置信息，自动完成软件回归测试。

针对现有软件测试工具和框架在解决大型软件的分布式自动化测试问题时的不足，作者基于分布式持续软件质量保证思想[5]，在现有的软件自动化测试平台原型 Skoll 的基础上，设计和实现了一个基于 Internet 的分布式自动化软件测试平台 DASTP (distributed automated software testing platform)。该平台提供软件生成、集成和测试的集成环境，支持大型软件系统的持续集成和测试的自动化，并利用网络上的空闲客户端资源，实现了在真实环境下进行软件测试的目标，解决了 Web 应用软件的配置测试、兼容性测试和分布式测试问题。这种测试方式既节约了软件测试成本，又提高了测试效率，使测试可以在全球 24 小时进行。

1　分布式自动化软件测试平台的构建

1.1　分布式自动化软件测试平台概述

Skoll 系统[5]是一个系统原型，它基于持续软件质量保证思想，针对软件配置测试提供了一系列的解决方案。Skoll 系统主要功能包括标识 QA 任务、划分并分配子任务到网络节点、收集和分析测试结果、根据结果调整测试计划。系统采用 C/S 结构，用户通过 Web 注册并构建测试客户端，然后客户端发送任务请求，服务器根据客户端信息将测试任务分配到测试客户端运行，最后服务器收集、分析测试结果并调整测试计划。系统运行环境如图 1 所示。

在 Skoll 系统的基础上，DASTP 采用基于脚本的实现方式，为其他软件测试类型和软件测试工具

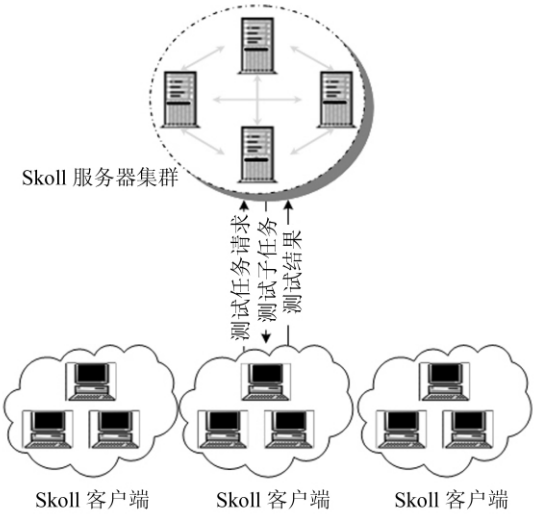

图 1　Skoll 运行环境

Fig. 1　Skoll running environment

的集成提供了解决方案；增加了基于 Web 的测试人员客户端，使测试人员能实时提交测试任务、查看测试结果和任务执行情况；增加了客户端状态监控模块，实现了对客户端的实时监控和管理；为测试任务的划分和调度给出了具体算法。

DASTP 平台运行环境如图 2 所示，平台基于 Internet 网络，测试人员通过 Web 浏览器提交测试任务，平台接收测试任务，并进行任务划分，生成测试子任务，然后将子任务分配到测试任务运行客户端运行，最后收集测试记录和结果，进行测试结果分析和测试报告生成。任务运行客户端是 Internet 上的终端志愿者，申请注册到该测试平台上，在其本机的空闲时段参与运行测试任务，服务器对注册的运行客户端进行管理。在测试执行过程中，服务器将动态监控任务运行客户端的状态，动态管理测试任务执行过程。

1.2　分布式自动化软件测试平台体系结构设计

DASTP 平台体系结构如图 3 所示，平台由测试人员客户端、任务运行客户端和服务器 3 部分组成。服务器与测试人员客户端之间采用 B/S 架构，服务器与任务运行客户端之间采用 C/S 架构。测试人员客户端提供测试任务提交、测试过程监控和测试结果查看等交互式页面。任务运行客户端负责测试任务的运行和测试结果的上传。服务器端是测试平台的核心，由 Web 服务器、测试服务器和数据库服务器组成，负责测试任务的接收和管理、测试子任务的生成与调度、对任务运行客户端的动态监控、测试结果收集与分析、测试计划调整等。

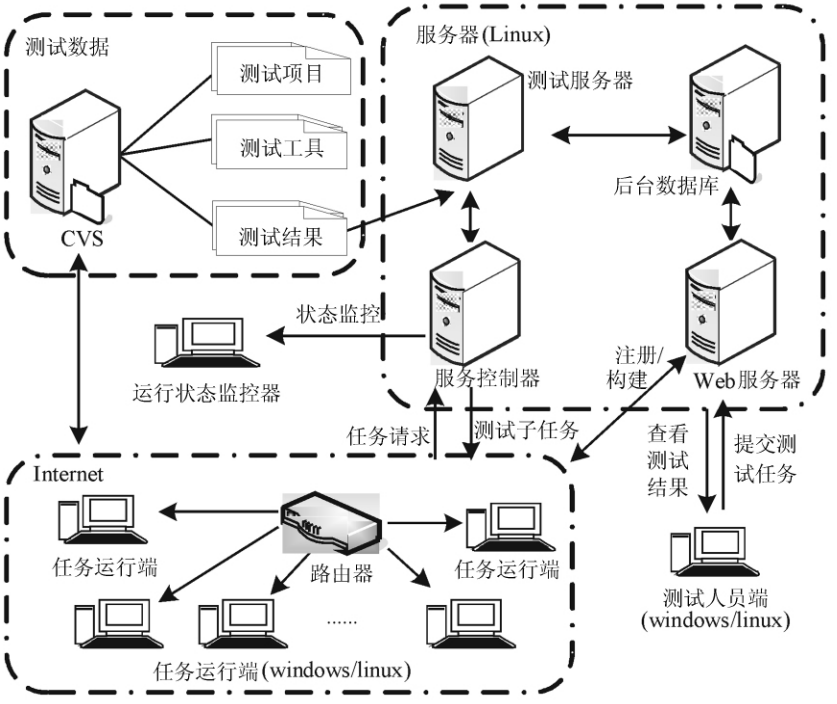

图2 DASTP平台运行环境

Fig. 2 DASTP platformrunning environment

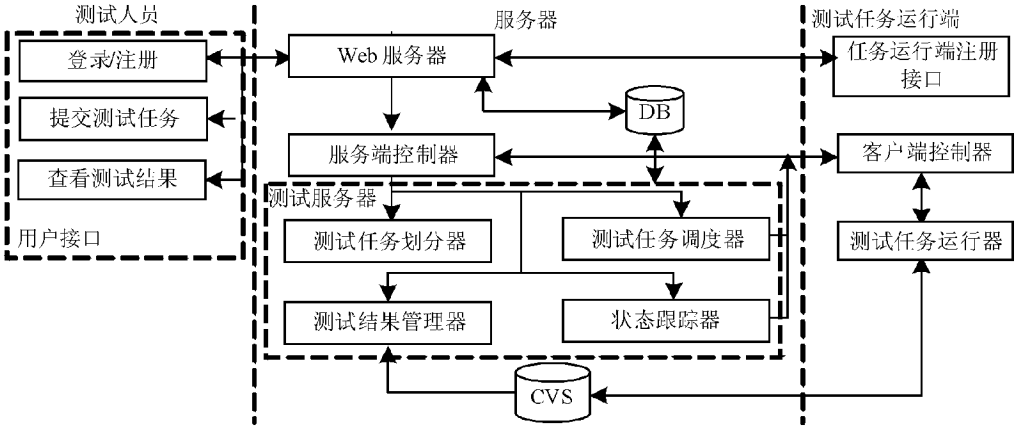

图3 DASTP体系结构图

Fig. 3 DASTP architecture

在服务器端,Web服务器提供测试人员注册登陆、测试任务提交、测试结果查询和任务运行客户端注册等4个用户交互接口;数据库服务器负责存储系统中的静态和动态数据资源,是系统的资源共享平台;测试服务器由测试任务划分器、测试任务调度器、测试结果管理器和测试任务状态跟踪器组成。测试任务划分器使用集合划分问题的蚁群算法对测试任务进行划分,生成测试子任务和脚本。测试任务调度器利用基于空闲时间约束的任务调度算法将测试子任务分配到合适的任务运行端。测试结果管理器分析运行端上传的测试结果,利用软件测试的统计度量准则(如故障率和覆盖率等)对测试结果进行分析,生成测试报告。状态跟踪器负责实时跟踪任务运行端的状态并根据其状态信息调用相应的异常处理模块。

1.3 分布式自动化软件测试平台处理流程

平台处理流程如图4所示。测试人员登录并上传测试任务;服务器调用任务划分器划分并生成子任务;任务运行端注册并安装客户端软件,在系统空闲时发送测试任务请求;服务器为任务运行端分配测试子任务;任务运行端构建测试环境、运行测试任务并上传测试结果;服务器分析收集的测试结果生成最终测试报告;测试人员通过浏览器查看测试结果报告。

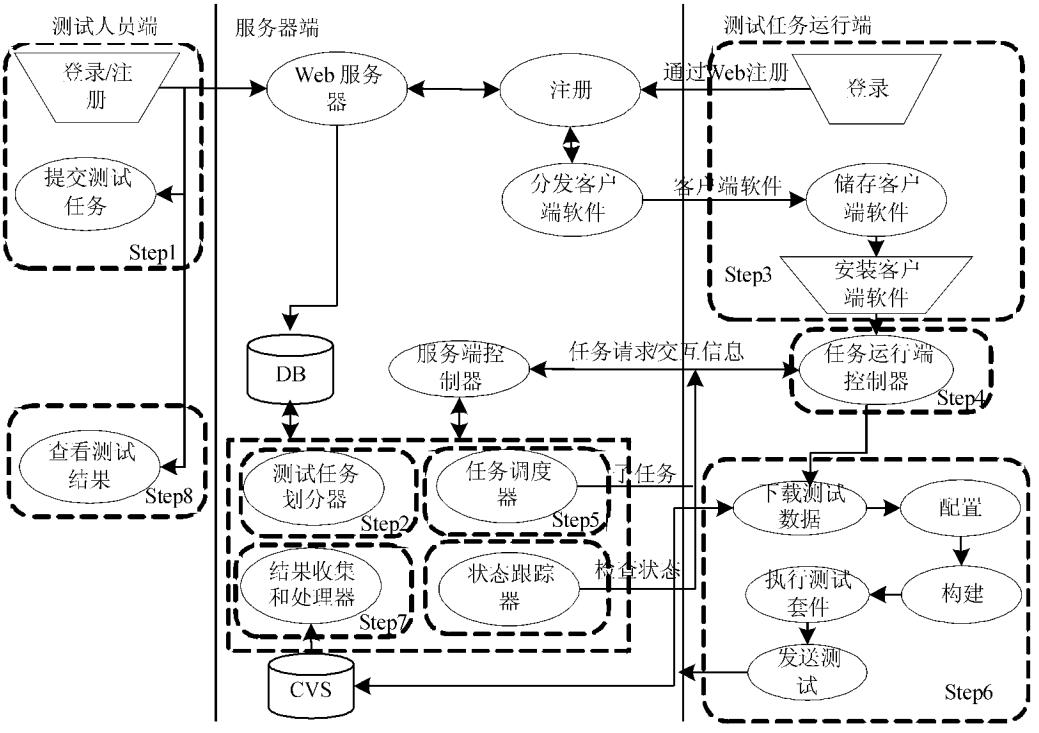

图 4　系统处理流程图

Fig. 4　System processing flow

2　分布式自动化软件测试任务管理

2.1　基于蚁群优化算法的软件测试任务划分

为了合理、高效的利用服务器与客户端资源,在划分软件测试任务时,应尽可能使各个子任务的执行时间相对均衡。如果各子任务执行时间差别很大,则调度算法很容易使执行时间相对较短或较长的任务被分配执行,而其他测试任务得不到执行,造成客户端资源浪费和测试进度的延误。若各子任务相对均衡,各测试任务要么都被分配执行,要么大多都不能被分配执行。当任务不能被分配执行时,就会激发任务生成模块重新划分子任务,最终使任务能被分配执行,实现资源的合理利用。

为了达到上述任务划分要求,将测试任务中的测试用例映射成独立的元素,测试用例的执行时间映射为元素的权值,相应的测试任务则映射为包含有限元素的集合,这样测试任务的划分问题就转化成了集合划分问题(SPP)[8]。集合划分问题是将给定集合划分成相互独立的子集,使产生的各子集内元素权值之和最小,并且各子集权值差异最小。SPP 的目标可以保证生成的各个子任务相对均衡。集合划分问题的数学描述如下[9]。

将一个包含 n 个元素的集合 S 划分成 m 个子集,用一个 $m \times n$ 的矩阵 $\boldsymbol{A} = (a_{ij})$ 表示元素分配情况,若第 j 个元素在第 i 个子集中则 $a_{ij} = 1$,否则 $a_{ij} = 0$。c_j 代表第 j 个元素的权值,x_j^k 是一个二进制变量,如第 j 个元素被第 k 子集选则取值为 1,否则为 0。SPP 的数学模型为:

$$\mathrm{Min}\, f^k(x) = \sum_{j=1}^{n} c_j x_j^k, \qquad k = 1, \cdots, m \tag{1}$$

约束条件:

$$\sum_{j=1}^{n} a_{ij} \times x_j^i = 1, \qquad i = 1, \cdots, m \tag{2}$$

2.1.1　传统的 ACO 算法

对于 SPP 问题,可以采用蚁群优化算法求解问题空间[8-10]。下面是蚁群优化算法的描述。蚂蚁在进行集合划分时,以概率 p 决定下一步所选元素,其值为:

$$p_j^k(t) = \frac{\tau_j^{\alpha} \times \eta_j^{\beta}}{\sum_{l \in N^k} \tau_l (\eta_l)^{\beta}}, \text{ if } \quad j \in N^k \tag{3}$$

其中,τ_j 为蚂蚁在搜索过程中在第 j 个元素上留下的信息素含量,η_j 为搜索过程中的启发式信息,α、β 为动态调整的参数,α 决定信息素在决策中所占的比重,β 决定启发式信息在决策中所占的比重,N^k 为允许加入子集的元素。启发式信息 η_j 求值表达式为:

$$\eta_j = \frac{1}{c_j} \tag{4}$$

其中，c_j 为元素 j 的权值。

当蚂蚁完成了一次集合划分后，元素上的信息素含量按式(5)进行更新:

$$\tau_i(t+\Delta t) = (1-\rho)\tau_i(t) + \Delta\tau_i(t, t+\Delta t) \quad (5)$$

其中，$\rho(0<\rho<1)$ 为信息素的挥发系数，$1-\rho$ 表示随着时间的推移信息素浓度的残留程度，$\tau_i(t)$ 为 t 时刻节点 i 处的信息素含量，Δt 表示时间间隔。在集合划分的蚁群算法中，蚂蚁是根据元素上的信息素含量来进行决策，而不是以路径上的信息素来决策[11]。$\Delta\tau_i$ 为一只蚂蚁扫描完一遍后在节点上留下的信息素，其计算公式为:

$$\Delta\tau_i = \begin{cases} \dfrac{Q}{L_k}, & \text{if } i \in tabuList^k \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

其中，L_k 为该次划分所产生的子集权值之和，Q 为系统给定的常量。$tabuList^k$ 是蚂蚁 k 已访问的元素集合，代表蚂蚁行进的路径，用以控制每个元素只被访问一次。

2.1.2 改进的 ACO 算法

Crawford 和 Castr[9] 给出了利用上述 ACO 算法解决集合划分问题的算法，并利用 AS 和 ACS 两种蚁群优化算法验证了算法的可行性。但其算法只能解决元素之间有约束条件的集合划分问题，对解决离散元素的 SPP 问题还存在一定的缺陷。这些缺陷主要包括，蚂蚁何时停止子集的构建，在集合中元素的权值差别较大时如何构建更加合理的划分结果。针对上面两点，从两个方面对文献[9]的 ACO 算法进行了改进，一是为子集设定动态容量监控标准，二是在蚁群周游前后增加预处理和后置处理过程。改进后，用蚁群优化算法实现集合划分的伪代码描述如下:

```
Procedure ACO for SPP
  Begin
    InitParameters( );
    While ( remain iterations) do
      j = PriorProcessingProcedures( );
      For k : = 1 to nants - j do
        While ( solution is not completed)
          SelectElement( );
          If Sum < D then
            AddElementToSolution( election)
            AddToTabuList( k);
          Else
            Break;
          endif
        EndWhile
      EndFor
      PostProcessingProcedures( );
      UpdateOptimum( );
      UpdatePheromone( );
    EndWhile
    Return best_solution_founded
  End
```

外层循环的迭代次数是系统设定，而内层 while 循环在集合中所有元素都被分配后或者子集中元素的权值之和大于阈值 D 时终止循环，$TabuList$ 表示已访问的元素列表，Sum 为子集权值之和。D 是为子集权值设定的动态容量监控标准，其初值为总的任务执行时间 / 划分后的子集数 $\times M\%$，其中参数 M 的取值范围为 $100 \le M \le 150$，其值可以根据实际情况设定。当完成一次子集划分后，若所有子集中权值的最大值小于当前的权值 D，则用该值替换 D，否则不做任何处理。PriorProcessingProcedures( ) 是预处理过程，用于去掉单个元素的权值大于给定阈值 D 的元素，并将其划分成单独的子集。PostProcessing-Procedures( ) 是后置处理过程，用于将剩余元素加入到当前权值最小的子集中。

算法实际运行时各参数设定如下: 信息素影响参数 $\alpha = 1$，启发式信息影响参数 $\beta = 0.5$，信息素蒸发率 $\rho = 0.1$，常数 Q 为任务总的执行时间，分组数 m 由系统根据测试任务运行端长时间的统计信息确定[10-12]。

实验证明，改进后的 ACO 算法能很好的满足平台对任务划分性能的要求。该算法比粒子群优化算法有更高的有效性[13]，比遗传算法有更低的时空复杂度[14]。

2.2 基于空闲时间约束的任务调度

为解决测试任务调度模块的任务动态分配问题，在综合现有的分布式任务调度算法[15-16]基础上提出了一个基于空闲时间约束的任务调度算法。该算法先根据测试任务和任务运行端的配置信息选出符合条件的一系列任务运行端，然后再根据任务执行时间和运行端空闲时间选出最合适的运行客户端。

下面是基于空闲时间约束的任务调度算法的描述: 定义任务配置空间模型 $T_k = (A, B, D, E, R)$，任务运行客户端配置空间模型 $C_k = (A, B, D, E, F, I)$。其中，A 为操作系统类型，B 为操作系统版本，D

为编译器类型，E 为操作系统配置，R 为任务的预计执行时间，F 为任务运行端的空闲时间; I 为任务运行端 ID。算法中，“=”表示等号右边和左边匹配，如操作系统类型 Win7 和 Windows 可以表示为 Windows =Win7; ≥为比较运算符，表示任务运行端空闲时间大于任务执行时间。

然后根据这两个模型，分别定义一个任务集合 $\boldsymbol{T}$、一个任务运行端集合 $\boldsymbol{C}$。

$$\boldsymbol{T}=\begin{bmatrix}T_1\\ \vdots\\ T_n\end{bmatrix}=\begin{bmatrix}A_1 & B_1 & D_1 & E_1 & R_1\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ A_n & B_n & D_n & E_n & R_n\end{bmatrix},$$

$$\boldsymbol{C}=\begin{bmatrix}C_1\\ \vdots\\ C_m\end{bmatrix}=\begin{bmatrix}A_1 & B_1 & D_1 & E_1 & F_1\\ \vdots & \vdots & \vdots & \vdots & \vdots\\ A_m & B_m & D_m & E_m & F_m\end{bmatrix}。$$

对于 $\boldsymbol{T}$ 中的任一任务 $T_p=(A_p,B_p,D_p,E_p,R_p)$，根据其配置信息，在集合 $\boldsymbol{C}$ 中运行匹配规则得到客户端集合 G:

$$G=\left\{G_k\left|\begin{array}{l}G_k\in C\&\&B_{G_k}=B_{T_p}\&\&A_{G_k}=A_{T_p}\&\&\\ D_{G_k}=D_{T_p}\&\&E_{G_k}=E_{T_p}\&\&F_{G_k}\geq R_{T_p}\end{array}\right.\right\}。$$

在集合 G 中找出空闲时间最接近任务 T_p 的估计执行时间的客户端，即为测试任务 T_p 的最佳执行客户端。

实验证明，在平台对于时效性要求较低的情况下，该算法能根据测试任务的运行环境和执行时间，以最小的时空复杂度，实现对测试任务的合理分配。并且该算法与集合划分问题的蚁群算法相结合，能很好的解决一般任务调度算法所产生的“饿死现象”和分配不均衡等现象。

3　分布式自动化软件测试实现与验证

3.1　系统平台的实现与运行

DASTP 是一个基于 Web 的系统，以 MyEclipse 为开发环境、Java 为开发语言，系统服务器运行在 Linux 环境下，测试人员客户端和测试任务运行端运行在 Linux 或 Windows 环境下。整个系统以 MySQL 作为后台数据库，用于存放系统的运行时动态信息，如测试任务信息、测试任务运行端信息等; 以 CVS 作为外部资源库，用于获取系统运行的静态信息，如被测程序、测试工具、测试用例等。为了完成自动化的软件测试任务，系统通过向测试脚本中添加测试环境构建脚本实现了对一系列现有软件构建和测试工具的集成，例如 ANT、Junit 等。

3.2　系统平台可行性验证

为了验证平台可行性，选取 MySQL 为被测软件，用 MySQL 自带的测试脚本作为测试数据。由于目标是验证平台架构和设计的可行性，因此在实验时模拟了一些处理部分，忽略了一些在实际运行环境应该考虑的问题，例如保密性和安全性。

用测试 MySQL 来验证平台可行性是一个很好的实验方案，因为 Linux 下的 MySQL 版本自带了一个自动化测试框架(MySQL Test Framework) ，要验证 MySQL 服务器端和客户端的功能只需在平台中集成该框架并运行相应测试脚本即可，并且该测试框架包含了一个含 600 个测试用例的测试集，每个测试用例都能独立完成一项测试任务。

实验环境如下: 1 台基于 Linux 的服务器，Windows 和 Linux 环境下不同配置和空闲时间的任务运行端各 5 台，2 台供测试人员使用的空闲 PC。测试开始前做如下准备:

测试人员端: 测试人员通过测试任务提交接口上传测试任务，包括测试任务名、被测软件或代码、测试数据、测试任务运行的环境信息、测试用例列表等信息。其中，测试数据包含测试运行脚本和测试用例，测试任务运行的环境信息包含测试任务配置信息(如操作系统类型、版本等) 、测试所需工具以及工具安装脚本，测试用例列表包含任务中所有用例的名字、估计执行时间。由于 MySQL 自带的测试脚本是在 MySQL Test Frameworks 框架上运行，而 MySQL Test Frameworks 仅在 Linux 环境下运行且对编译器、操作系统版本等没有明确要求，因此在实验中仅设定操作系统类型为 Linux、测试工具为 MySQL Test Frameworks、工具安装脚本为 MySQL 的安装脚本，忽略其他测试任务运行的环境信息。其他参数设置为: 任务名 mysql，测试软件为 MySQL 5.1，测试用例为 MySQL 自带的 600 个测试用例。

测试任务运行客户端: 通过 Web 页面注册客户端。在注册客户端时，仅设定客户端的操作系统版本和空闲时间，每种操作系统类型的客户端的空闲时间依次为 24、27、30、33、36 min，忽略其他配置信息。

完成上述操作后，登录各测试任务运行端，执行测试任务，待测试结束后查看最终测试结果如图 5 所示。

由实验结果可知，测试人员只需提交测试任务，系统便能根据用户提交的测试任务自动进行任务划分、生成测试子任务，并将子任务分配到合适的运行

端去运行,运行结束后收集测试结果、生成最终测试报告。分析最终测试报告可以清楚了解整个测试的执行结果,包括每个测试用例的通过与否、测试统计信息(如故障率和覆盖率等)。因此,该实验很好地验证了本文测试平台架构和设计的可行性以及平台对其他测试工具的支持。

Pass Standard List

Pass Standard	Expected Result	Actual value	Task Name
Failure Density	0:0.1	0.1875	mysql
Coverage Index	0.6:1	0.8125	mysql

Test Case List

Test Case Name	Result
im_options_set	failed
im_options_unset	failed
im_utils	failed
index_merge	pass
index_merge_innodb	pass

图5 最终测试结果

Fig. 5 Final testing results

4 结论与进一步工作

基于分布式的持续软件质量保证思想提出了一个分布式自动化测试平台,该平台支持大型软件系统的持续集成和测试,能利用 Internet 网络上的空闲客户端资源实现自动化分布式测试,包括测试任务划分和调度、测试任务自动运行、测试结果收集和分析等功能,为分布式自动化软件测试提供了解决方案。通过实验验证了原型设计和架构的可行性。进一步研究主要是针对以下几个方面:测试脚本的自动生成、任务划分与调度算法的优化、测试结果处理部分相应判定准则的优化和平台安全性、可靠性研究。

参考文献:

[1] Lin Yuehua, Zhang Jing, Gray J. A testing framework for model transformations [M] //Beydeda S, Matthias B, Gruhn V. Modeldriven Software Development, Research and Practice in Software Engineering. Springer, 2005: 219 – 236.

[2] Stocks P A, Carrington D A. Test templates: A specification-based testing framework [C] //Proceedings of 15th International Conference on Software Engineering. 1993: 405 – 414.

[3] Hartman A, Nagin K. The AGEDIS tools for model based testing [C]. International Symposium on Software Testing and Analysis, 2004, 29(4): 129 – 132.

[4] Marinov D, Khurshid S. TestEra: A novel framework for automated testing of Java programs [C] //Proc 16th IEEE International Conference on Automated Software Engineering (ASE). San Diego, CA, 2001: 22 – 32.

[5] Porter A, Yilmaz C, Memon A M. Skoll: Distributed continuous quality assurance [C] //Proceedings of the 26th IEEE/ACM International Conference on Software Engineering. Edinburgh, Scotland, IEEE/ACM, 2004.

[6] Rankin C. The software testing automation framework [J]. IBM Systems Journal: Software Testing and Verification, 2002, 41(1): 126 – 139.

[7] Zhu F, Rayadurgam S, Tsai W T. Automating regression testing for real-time software in a distributed environment [C] //Proceedings of First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 98). 1998: 373 – 382.

[8] Maniezzo V, Milandri M. An ant-based framework for very strongly constrained problems [C] //Dorigo M, Di Caro G, Sampels M. Ant algorithms—Proceedings of ANTS 2002—Third International Workshop. Lecture Notes in Comput Sci, Berlin: Springer, 2002, 2463: 222 – 227.

[9] Crawford B, Castro C. Integrating lookahead and post processing procedures with ACO for solving set partitioning and covering problems [C] //Rutkowski L, Tadeusiewicz R, Zadeh L A, et al. ICAISC 2006. LNCS (LNAI), Heidelberg: Springer, 2006, 4029: 1082 – 1090.

[10] Dorigo M, Stutzle T. Ant colony optimization [M]. Cambridge, MA: MIT Press, 2004.

[11] Leguizamon G, Michalewicz Z. A new version of ant system for subset problems [C] //Congress on Evolutionary Computation. CEC'99, Piscataway, NJ, USA, IEEE Press, 1999: 1459 – 1464.

[12] Lessing L, Dumitrescu I, Stutzle T. A comparison between ACO algorithms for the set covering problem [C] //ANTS 2004. Lecture Notes in Comput Sci, SV, 2004, 3172: 1 – 12.

[13] Chu P C, Beasley J E. Constraint handling in genetic algorithms: the set partitioning problem [J]. Journal of Heuristics, 1998(4): 323 – 357.

[14] Gao Shang, Hou Zhiyuan. Particle swarm optimization algorithm for set partition problem [J]. Journal of Jiangsu University of Science and Technology: Natural Science Edition, 2005(12): 42 – 45. [高尚,侯志远. 集合划分问题的粒子群优化算法 [J]. 江苏科技大学学报:自然科学版, 2005(12): 42 – 45.]

[15] Dandamudi S P. A comparison of task scheduling strategies for multiprocessor systems [C] //Proceedings of the IEEE Symposium on Parallel and Distributed Processing. Dallas, TX, 1991: 423 – 426.

[16] Aguilar J, Gelenbe E. Task assignment and transaction clustering heuristics for distributed systems [J]. Information Sciences, 1997, 97: 199 – 219.

(编辑 杨 蓓)