软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(8):1964−1977 [doi: 10.13328/j.cnki.jos.004845]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

串并行软件系统测试资源动态分配建模及求解*

陆 阳, 岳 峰, 张国富, 苏兆品, 王永奇

(合肥工业大学 计算机与信息学院,安徽 合肥 230009)

通讯作者: 岳峰, E-mail: yuefeng@hfut.edu.cn, http://www.hfut.edu.cn

摘 要: 系统测试是软件开发各个阶段中最消耗时间和资源的阶段,对于串并行软件系统来说,系统可靠性随着测试时间的推进会发生变化,如果再按照最初的方案分配测试资源,可能会造成测试资源的浪费,这时需要分阶段对测试资源进行再分配.在基于搜索理论的软件工程领域展开研究,首先,在分析测试资源、测试代价和系统可靠性关系的基础上构建了以测试资源为约束,以最大化可靠性和最小化测试成本为目标的测试资源多目标动态分配模型,按照测试进程的推进,动态地分配测试资源;然后,基于具有改进种群初始化策略的"一维整数向量编码"差异演化算法,提出了一种针对串并行软件系统的测试资源动态分配算法.对比实验结果表明,测试资源动态分配模型在保证系统可靠性的前提下,有效地节省了系统测试的消耗,提高了串并行软件系统的开发效率.

关键词: 串并行软件系统;测试资源动态分配;可靠性;差异演化;一维整数向量编码;种群初始化

中图法分类号: TP311

中文引用格式: 陆阳,岳峰,张国富,苏兆品,王永奇.串并行软件系统测试资源动态分配建模及求解.软件学报,2016,27(8):1964−1977. http://www.jos.org.cn/1000-9825/4845.htm

英文引用格式: Lu Y, Yue F, Zhang GF, Su ZP, Wang YQ. Model and solution to testing resource dynamic allocation for series-parallel software systems. Ruan Jian Xue Bao/Journal of Software, 2016,27(8):1964−1977 (in Chinese). http://www.jos.org.cn/1000-9825/4845.htm

Model and Solution to Testing Resource Dynamic Allocation for Series-Parallel Software Systems

LU Yang, YUE Feng, ZHANG Guo-Fu, SU Zhao-Pin, WANG Yong-Qi

(School of Computer and Information Science, Hefei University of Technology, Hefei 230009, China)

Abstract: Software testing is the most time and resource consuming stage during software development. For series-parallel software systems, as the reliability of the system changes as the testing time advancing, if the strategy of testing resource allocating is still executed in accordance with the original plan, it may lead to a vast waste of testing resource. To address the issue, this paper tackles a testing resource dynamic allocation problem for series-parallel software systems with bounded resource in the field of search based software engineering. Firstly, the definitions of testing resource, system reliability and testing cost are given. Based on these definitions, a multi-objective dynamic allocation model for testing resource is established with the objective of allocating the testing resource among different modules to maximize the reliability and minimize the testing cost subject to the available testing resource. Then, a "1-dimensional integer vector coding" differential evolution algorithm with improved colony initialization strategy is proposed for the dynamic model. Comparison results with existing models show that the proposed approach is effective and efficient for solving the testing resource

* 基金项目: 国家自然科学基金(61174170, 61100127, 61371155); 教育部博士点基金(20120111110001); 安徽省自然科学基金(1508085MF132, 1508085QF129)

Foundation item: National Natural Science Foundation of China (61174170, 61100127, 61371155); Ph.D. Programs Foundation of Ministry of Education of China (20120111110001); Anhui Provincial Natural Science Foundation of China (1508085MF132, 1508085QF129)

收稿时间: 2014-11-24; 修改时间: 2015-01-27, 2015-03-30; 采用时间: 2015-04-20

allocation problem, therefore providing a way to reduce the consumption of the testing resource and to improve the reliability and development efficiency of series-parallel software systems.

Key words: series-parallel software system; testing resource dynamic allocation; reliability; differential evolution; 1-dimensional integer vector coding; colony initialization

软件测试的目的是为了发现存在的错误和潜在的威胁,从而提高系统稳定性和可靠性.然而随着软件系统规模的显著增大,软件测试变为最为消耗时间和资源的阶段,大概50%以上的资源是在测试阶段被消耗的.因此,需要在测试阶段有效地分配测试资源,以最少的成本去找出软件中潜在的各种错误和缺陷,通过修正各种错误和缺陷来提高软件可靠性.

测试资源即指软件测试中所耗费的人力、时间等各种资源开销.在软件工程中,为了对开发资源进行量化,定义了人×时间的资源量化单位,最常见的是人月,除此之外,人时、人天、人年等也常被使用.

随着串并行软件系统的广泛应用,软件复杂度不断提高,包括的模块越来越多.在测试过程中,每个模块都需要经过测试.特别是在软件的单元测试中,不同模块的测试活动都在激烈争夺有限的测试资源,不同的测试活动也会带来不同的成本的上升与可靠性的增加.软件工程师必须清楚如何将测试资源分配给不同的模块测试.测试资源分配问题就是如何把有限的测试资源分配给每一个模块,从而保证软件系统的可靠性最大.

测试资源分配问题是系统可靠性优化中的关键和重要问题.自 20 世纪 90 年代以来,已经成为研究的热点问题.在早期的工作中,测试资源的分配目标很单一,就是为了单纯地最大化系统的可靠性[1,2].

随着研究的深入,测试代价作为一个新的因素越来越得到人们的重视.直观来说,测试代价是在测试进程中为了使系统达到一定可靠性水平所需要的代价度量.它是一个以可靠性值为自变量的函数,因此它也是一个以测试资源分配方案为自变量的函数.绝大部分的工作或者是以最小化测试代价取代最大化可靠性[3−6],或者是在测试资源和测试代价的约束下最大化系统的可靠性[7−9],均是作为一个单目标问题来研究的.其中,代表性的研究是基于搜索算法求解测试资源分配问题.Huang 等人[4]研究了在测试代价一定的情况下最大化系统可靠性的测试资源分配问题,并提出了一种启发式的求解方法.Huang 等人[5]提出了一种基于动态规划(dynamic programming,简称 DP)的测试资源分配算法.Kapur 等人[6]基于数学规划方法求解测试资源约束下可靠性最大化问题.Aggarwal[9]和 Kapur[10]团队引入遗传算法(genetic algorithms,简称 GA)求解测试资源分配问题.

在现实的系统设计进程中,可靠性和测试代价都是十分重要的.众所周知,在当测试资源一定的情况下,若想获得更高的可靠性,必然意味着需要付出更大的测试代价.所以不可能得到一个方案,这个方案在可靠度值和测试代价两个目标上都达到最优的情况,而只能寻求在可靠度值和测试代价两个目标上的平衡,从而得到一种均衡的分配方案.诚然,一些科研工作者[11−19]已经尝试了这项工作,讨论了基于测试时间、可靠性、测试资源成本为核心的多目标优化模型.其中,Kumara[10]和 Dai[11]等人基于遗传算法求解多目标测试资源分配问题;Wang 等人[14]研究了测试资源给定的情况下同时考虑可靠性最大和代测试价最小的问题,并采用多目标演化算法(multi-objective evolutionary algorithm,简称MOEA)进行求解;Yu等人[15]在Wang等人[14]工作的基础上设计了一种基于局部搜索策略的多目标演化算法进行求解;Yin 等人[16]采用混合粒子群搜索算法(hybrid particle swarm optimization,简称 HPSO)求解测试资源的多目标优化模型;Jia 等人[17]采用多目标粒子群算法(multi-objective particle swarm optimization,简称 MOPSO)求解多准则人力资源的多目标分配;Chaharsooghi 等人[18]引入蚁群算法(ant colony optimization,简称 ACO)求解测试资源的分配问题.

这些模型和方法均属于静态分配测试资源的方法,即根据系统初始时刻各模块的可靠性进行测试资源的分配,不考虑各模块可靠性随着测试进程发生变化的情况.这些模型和方法的缺点是,在测试完成后,有可能有些模块的可靠性远远低于系统的可靠性,使软件测试的效果不佳.

因此,研究资源受限的、面向大规模串并行软件系统的测试资源动态分配模型和求解方法势在必行.为此,本文首先构建一种资源受限的测试资源多目标动态分配模型,在此基础上,采用基于具有改进种群初始化策略的“一维整数向量编码”差异演化算法对模型进行求解.

1 资源受限的串并行软件系统测试资源动态分配模型

串并行软件系统通常包括多个模块,这些模块之间可能是串行关系,也可能是并行关系.一个具有多个串并行模块结构的串并行软件系统模型如图 1 所示.现实中的绝大多数串并行软件系统都可以用这种模型进行描述.

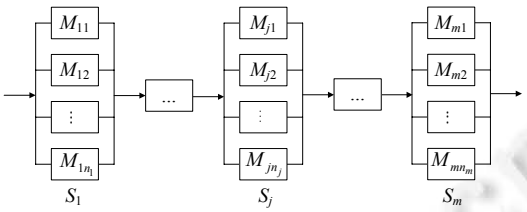

Fig.1 Series-Parallel software systems

图 1 串并行软件系统

一般情况下,测试资源分配通常在以下的前提条件下进行:

(1) 串并行软件系统由 m 个串联子系统构成,每个子系统 $S_j(j=1,2,\ldots,m)$由 n_j 个冗余模块 $M_{j1},\ldots,M_{jk},\ldots,M_{jn_j}$ 并联组成;

(2) 每个模块的测试是单独进行的;

(3) 任意模块 M_{jk} 的平均失效次数符合非齐次泊松过程分布;

(4) 每次失效(错误)的发生是相互独立的.

1.1 串并行软件系统的测试资源

在本文中,可用的测试资源指的是系统测试时间,为某一固定的有限值 T(定义了人×小时的资源量化单位).对于如图 1 所示的串并行软件系统而言,系统实际的测试时间 T^a 等于 m 个串联子系统的测试时间之和,即

$$T^a=\sum_{j=1}^{m}t_j \tag{1}$$

而对于具有多个冗余模块的子系统来说,每个模块均可以单独完成子系统的功能,即冗余模块之间是并联的关系.在这种情况下,如果再依次对每个模块进行测试,将会浪费大量的测试资源.因此,子系统的测试可以通过同时对每个冗余模块进行测试完成.也就是说,子系统 S_j 的测试时间应该为其所有冗余模块测试时间的最大值,即

$$t_j=\max_k\{t_{jk}\} \tag{2}$$

其中,t_{jk} 为模块 M_{jk} 消耗的测试资源,且 $t_{jk}>0$.

综合公式(1)和公式(2),系统测试时间可以表示为

$$T^a=\sum_{j=1}^{m}t_j=\sum_{j=1}^{m}\max_k\{t_{jk}\} \tag{3}$$

1.2 串并行软件系统的可靠性

设$\varphi_{jk}(t_{jk})$表示失效累计发生总数的均值函数,如公式(4)所示.

$$\varphi_{jk}(t_{jk})=a_{jk}\cdot b_{jk}\cdot \mathrm{e}^{-b_{jk}t_{jk}} \tag{4}$$

其中,a_{jk} 和 b_{jk} 是固定的参数:a_{jk} 是在模块 M_{jk} 在无限执行时间内可观察到的错误总数的期望值;b_{jk} 是模块 M_{jk} 的错误探测率,即在模块 M_{jk} 中发现的错误占存在错误的比例.

模块 M_{jk} 的可靠性 r_{jk} 可以表示为测试资源 t_{jk} 的函数,如公式(5)所示:

$$r_{jk}(x/t_{jk})=\mathrm{e}^{-\varphi_{jk}(t_{jk})\cdot x},x\geqslant 0 \tag{5}$$

对于如图 1 所示的串并行软件系统而言,其核心步骤为串联结构,系统总体可靠性取决于每一个串联子系统的性能.由乘法规则,$R(x/T)$可表示为公式(6)所示:

$$R(x/T)=\prod_{j=1}^{m}r_j(x/t_j) \tag{6}$$

对于每一个串联子系统来说,为了提高结构的可靠性,其内部可能存在多个模块,以并联的方式增加冗余度.在这种情况下,除非该子系统内所有模块都失效,否则,该子系统将一直保持正常工作.因此,具有 n_j 个冗余模块的第 j 个子系统的可靠性可表示为公式(7):

$$r_{jk}(x/t_{jk})=1-\prod_{k=1}^{n_j}[1-r_{jk}(x/t_{jk})] \tag{7}$$

综合公式(6)和公式(7),系统可靠性可以表示为

$$R(x/T)=\prod_{j=1}^{m}\left\{1-\prod_{k=1}^{n_j}[1-r_{jk}(x/t_{jk})]\right\} \tag{8}$$

1.3 串并行软件系统的测试代价

对于串并行软件系统而言,系统测试代价由其所有子系统的测试代价决定.假设 C 表示系统总的测试代价,C_j 表示子系统 S_j 的测试代价,则

$$C=\sum_{j=1}^{m}C_j \tag{9}$$

对于子系统 S_j,其测试代价应该为每个冗余模块的测试代价之和,即

$$C_j=\sum_{k=1}^{n_j}C_{jk} \tag{10}$$

其中,C_{jk} 为模块 M_{jk} 的测试代价.

在软件测试中,测试代价通常与可靠性有关.一般而言,可靠性要求越高,所花费的测试代价越大;反之,可靠性要求越低,测试代价越小.也就是说,测试代价可以表示为可靠性的函数,如公式(11)所示:

$$C_{jk}(r_{jk})=x_{jk}\cdot \mathrm{e}^{y_{jk}\cdot r_{jk}-z_{jk}} \tag{11}$$

其中,x_{jk},y_{jk} 和 z_{jk} 是固定的参数,其作用是控制模块 M_{jk} 测试代价的增长速度.

综合公式(9)~公式(11),系统测试代价可以表示为

$$C=\sum_{j=1}^{m}\sum_{k=1}^{n_j}C_{jk}(r_{jk})=\sum_{j=1}^{m}\sum_{k=1}^{n_j}x_{jk}\cdot \mathrm{e}^{y_{jk}\cdot r_{jk}-z_{jk}} \tag{12}$$

1.4 串并行软件系统测试资源动态分配模型

目前,已有的测试资源分配模型侧重于研究测试资源约束下最大化系统的可靠性,或者测试资源约束下最小化系统测试代价,或者测试资源约束下最大化软件可靠性和最小化测试代价.

但是对于这种模块众多的串并行软件系统来说,随着测试活动的进行,系统的可靠性和各模块的可靠性都会发生变化,如果再按照最初的方案分配测试资源,可能会造成测试资源的浪费,带来额外损失.因此,测试资源的分配需要分阶段进行再分配.为此,本文将着手构建资源受限的测试资源多目标动态分配模型.

如图 2 所示,将测试资源 T 分为 p 个测试阶段:

$$(T_0=0,T_1],\ldots,(T_{i-1},T_i],\ldots,(T_{p-1},T_p=T] \tag{13}$$

对于第 i 个测试阶段$(T_{i-1},T_i](i=1,2,\ldots,p)$,同时考虑系统可靠性和测试代价,把测试资源分配问题定义为如下的多目标优化问题(multi-objective optimization problem,简称 MOP).

- 目标函数:

$$\text{Max } R_i(x/(T_i-T_{i-1}))=\prod_{j=1}^{m}\left\{1-\prod_{k=1}^{n_j}[1-r_{ijk}(x/t_{ijk})]\right\} \tag{14}$$

$$\text{Min } C_i=\sum_{j=1}^{m}\sum_{k=1}^{n_j}C_{ijk}(r_{ijk}) \tag{15}$$

- 约束条件:

$$T_i^a=\sum_{j=1}^{m}t_{ij}=\sum_{j=1}^{m}\max_k\{t_{ijk}\}\leqslant T_i-T_{i-1},t_{ijk}>0 \tag{16}$$

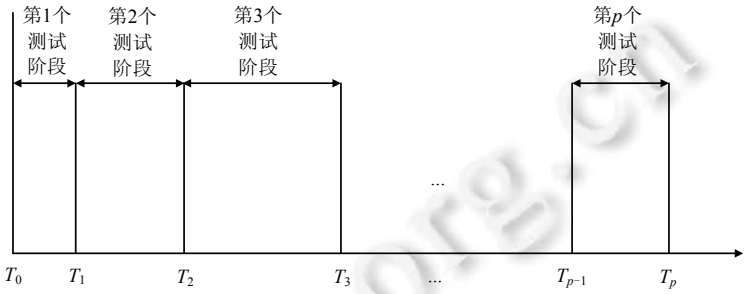

Fig.2 Testing phase

图 2 测试阶段

在公式(14)中,$R_i(x/(T_i-T_{i-1}))$表示该系统在第 i 个测试阶段的总体可靠性,t_{ijk}表示模块 M_{jk}在第 i 个测试阶段所消耗的测试资源,$r_{ijk}(x/t_{ijk})$表示模块 M_{jk}在第 i 个测试阶段的可靠性.对于同一个模块 M_{jk}来说,其平均失效次数是测试资源的函数,服从泊松分布,这一函数关系在每个测试阶段均是相同的.换句话说,平均失效次数与测试资源的函数关系是与所处的测试阶段无关的.由公式(5),模块的可靠性可表示为测试资源的函数,这种函数关系也是固定的,与其所在的测试阶段无关.因此,本文用 $r_{jk}(x/t_{ijk})$来代替 $r_{ijk}(x/t_{ijk})$,即认为模块的可靠性只与其所消耗的测试资源有关.因此,公式(14)表明,测试资源分配问题要追求系统在每个测试阶段可靠性的最大化.

在公式(15)中,C_i表示第 i 个测试阶段的系统测试代价,$r_{ijk}=r_{jk}(x/t_{ijk})$表示模块 M_{jk}在第 i 个测试阶段的可靠性,$C_{ijk}(r_{ijk})$表示模块 M_{jk}在第 i 个测试阶段的测试代价.同理,对于同一个模块 M_{jk}来说,测试代价是其所要求的可靠性的函数,这种函数关系也是固定的,而与其所在的测试阶段无关.因此,用 $C_{jk}(r_{ijk})$来代替 $C_{ijk}(r_{ijk})$.公式(15)表明,测试资源分配问题还要追求每个测试阶段测试代价的最小化.

在公式(16)中,T_i^a 表示第 i 个测试阶段的系统实际消耗的测试资源,t_{ij}表示第 i 个测试阶段的子系统 j 实际消耗的测试资源,t_{ijk}表示模块 M_{jk}在第 i 个测试阶段实际消耗的测试资源.公式(16)表明,系统在第 i 个测试阶段实际消耗的测试资源不应该超过总资源量 T_i-T_{i-1}.

综上,资源受限的测试资源多目标动态分配模型可以描述为

$$\left.\begin{aligned}&\text{Max } R_i(x/(T_i-T_{i-1}))=\prod_{j=1}^{m}\left\{1-\prod_{k=1}^{n_j}[1-r_{jk}(x/t_{ijk})]\right\}\\&\text{Min } C_i=\sum_{j=1}^{m}\sum_{k=1}^{n_j}C_{jk}(r_{ijk})\\&\text{s.t.}\\&\quad T_i^a=\sum_{j=1}^{m}t_{ij}=\sum_{j=1}^{m}\max_k\{t_{ijk}\}\leqslant T_i-T_{i-1}\\&\quad 0<t_{ijk}\leqslant T_i-T_{i-1}\end{aligned}\right\} \tag{17}$$

1.5 模型分析

根据第 1.1 节~第 1.4 节,本节同时考虑系统可靠性和测试代价,把测试资源分配问题定义为多目标优化问题.多目标优化问题又称为多属性优化问题、多准则优化问题和多指标优化问题.

对于公式(14)描述的多目标优化问题,其 2 个目标函数之间存在相互冲突的关系,很难在解空间中寻找到一个解,使其在所有目标函数上达到最优.因此,解决多目标优化的目标应该是寻找到一组相互之间具有很好权衡关系的解,Pareto 将这种解定义为 Pareto 最优解.

定义 1(多目标优化问题)[19,20]. 设目标函数的个数为 g,约束条件的个数为 c,决策变量的维数为 d,则以最小化为例,带约束的多目标优化问题可描述为

$$\left.\begin{aligned}&\min\ F(x)=(f_1(x),f_2(x),...,f_g(x))^T\\&\text{s.t.}\\&\quad h_i(x)\geqslant 0,i=1,2,...,c\end{aligned}\right\}\tag{18}$$

其中,$\boldsymbol{x}=(x_1,x_2,\ldots,x_d)^T$ 为决策向量,$f_1(\boldsymbol{x}),f_2(\boldsymbol{x}),\ldots,f_g(\boldsymbol{x})$为目标函数,$h_1(\boldsymbol{x}),h_2(\boldsymbol{x}),\ldots,h_c(\boldsymbol{x})$为约束条件.

定义 2(可行解)[19,20]. 如果一个决策变量 $\boldsymbol{x}$ 能够满足公式(18)中的 c 个约束条件,则决策变量 $\boldsymbol{x}$ 称为可行解.全部可行解构成的集合成为可行解集,记为 X.

定义 3(Pareto 支配)[19,20]. 在可行解集 X 中,对于任意两个可行解 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$,如果满足下面的条件:

$$\left.\begin{aligned}&\forall j=1,2,...,g,f_i(\boldsymbol{x}_1)\leqslant f_i(\boldsymbol{x}_2)\\&\exists k=1,2,...,g,f_i(\boldsymbol{x}_1)<f_i(\boldsymbol{x}_2)\end{aligned}\right\}\tag{19}$$

则称 $\boldsymbol{x}_1$ 支配 $\boldsymbol{x}_2$,记为 $\boldsymbol{x}_1\succ\boldsymbol{x}_2$.

定义 4(Pareto 最优解)[19,20]. 若可行解集 X 中的一个解 $\boldsymbol{x}^*$是 Pareto 最优解,当且仅当在 X 中不存在其他的解支配 $\boldsymbol{x}^*$.全部 Pareto 最优解组成的集合称为 Pareto 最优解集、非劣解集或非支配解集,记为 X_P.

定义 5(Pareto 最优前沿)[19,20]. Pareto 最优集相对应的目标函数向量所构成的集合被称为 Pareto 前沿.

多目标优化问题的最终目的是寻找或逼近问题的 Pareto 最优集以及 Pareto 前沿.通常的求解方法是:将需要优化的各个子目标通过系数加权的方式转化为一个单目标函数,再借鉴单目标优化问题的求解算法进行求解.其中,加权系数的取值由使用者设定或者由优化算法进行自适应调节.传统多目标优化方法包括约束法、线性加权和法、极大极小法、目标规划法等.这些方法的优点是简单,但是系数很难确定,而且待优化目标排列的次序对结果的影响也很大.因此,在求解之前需要足够的先验知识.而实际中,需要求解的多目标优化问题的先验知识往往是未知的、难以获取的,且有些目标函数是非线性的、不连续的或不可微的,传统的方法往往会出现无法收敛或收敛速度非常慢的情况.所以对目标函数复杂的多目标优化问题,传统方法的求解效果往往不够理想,很难达到 Pareto 最优解.

进化算法(evolutionary algorithm,简称 EA)[21]的出现,为多目标优化问题的求解提供了新的思路和方法.进化算法是通过模拟物种在自然环境选择作用下的进化更新过程进行随机搜索的一种优化算法,进化算法解决多目标优化问题的优势在于:

(1) 进化算法是启发式算法,能够处理各种形式的优化目标函数,而且由于其采用了随机搜索策略,算法在使用时不需要先验知识,也不涉及复杂的数学推导.

(2) 进化算法是基于种群思想寻找最优解的,而且种群之间互相操作形成新的种群.这样可以提高解的多样性以及搜索的全局性.

因此,国内外的许多文献中都提出了比较有效的多目标优化算法,其中经典的多目标进化算法包括 Fonseca 和 Fleming 等人在 1993 年提出的多目标遗传算法(MOGA)[19]、Deb 和 Srinivas 等人在 1994 年提出的非劣排序遗传算法(NSGA)[20]、Horn 和 Nafpliotis 等人在 1994 年提出的小生境 Pareto 遗传算法(NPGA)[22]等,被称为第一代进化多目标优化算法.这些方法提出了基于 Pareto 支配等级来划分解的优劣性、通过适应度共享来保持群体多样性等思想,为进化算法在多目标优化领域的应用奠定了基础.

Zitzler 等人在 1999 年提出的强度 Pareto 进化算法(SPEA)[23]、Zitzler 等人在 2001 年提出的 SPEA 改进算法 SPEA-II[24]、Knowles 和 Corne 等人在 2000 年提出的 Pareto 存档进化策略(PAES)[25]及其改进算法 PAES-II[26]、Deb 等人在 2002 年提出的 NSGA 的改进算法 NSGA-II[27]等,被称为第二代进化多目标优化算法.这些方法都是基于精英保留思想以加快进化速度,提高收敛性能.

随着粒子群算法(particle swarm optimization,简称 PSO)[28]和差异演化算法(differential evolution,简称 DE)[29]的出现,学者们开始在多目标优化领域引入 PSO 和 DE,提出了一些可借鉴的方法.

因此,本文引入差异演化算法求解串并行软件系统测试资源动态分配模型,设计了一维整数向量编码表示模型的可行性解,设计了种群初始化策略提高算法的性能.

2 基于差异演化算法的测试资源动态分配算法

由于本文考虑的是测试资源的动态分配,在每个测试阶段$(T_{i-1},T_i]$中,都需要完成把测试资源 T_i-T_{i-1} 分配给每一个模块 M_{jk} 的任务.但是在实际分配过程中,系统测试可能不需要这么多的资源,也就是说,实际测试时间 $T_i^a < T_i - T_{i-1}$.这时,下一个测试阶段可能提前.换句话说,下一个测试阶段的测试资源由 $T_{i+1}-T_i$ 变为 $T_{i+1}-T_i+(T_i - T_{i-1} - T_i^a)$.因此,在每一测试阶段找到最优解之后需要解码,获得实际的测试时间 T_i^a.基于差异演化算法的测试资源动态分配算法(DE-TRDAA)总体流程如图 3 所示.本文下面将对算法中的关键问题进行详细说明.

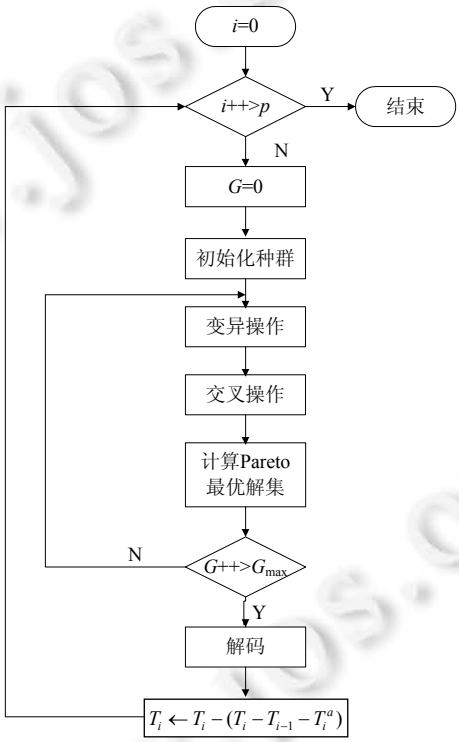

Fig.3 Flowchart of DE-TRDAA

图 3 基于差异演化算法的测试资源动态分配算法流程图

2.1 编 码

采用一维整数向量编码表示一个个体,如图 4 所示.在每个编码中,有 m 个整数向量,Γ_j 表示子系统 S_j 中每个模块分配到的测试资源向量,且

$$\Gamma_j = [t_{j1}, t_{j2}, ..., t_{jn_j}],\ t_{jk} > 0,\ k = 1, 2, ..., n_j \tag{20}$$

设每个种群中有 N 个个体,$\mathbb{L}_G$ 为第 G 代种群,$\mathbb{l}_l^G=[\Gamma_{l1}^G,...,\Gamma_{l2}^G,...,\Gamma_{lm}^G]$ 表示第 G 代种群 $\mathbb{L}_G$ 中第 l 个个体 (l=1,2,…,N). $\tilde{\mathbb{l}}_l=[\tilde{\Gamma}_{l1},...,\tilde{\Gamma}_{lj},...,\tilde{\Gamma}_{lm}]$ 是变异后的个体,$\tilde{\mathbb{l}}_{l'}=[\tilde{\Gamma}_{l'1},...,\tilde{\Gamma}_{l'j},...,\tilde{\Gamma}_{l'm}]$ 和 $\tilde{\mathbb{l}}_{l''}=[\tilde{\Gamma}_{l''1},...,\tilde{\Gamma}_{l''j},...,\tilde{\Gamma}_{l''m}]$ 是交叉后的个体.

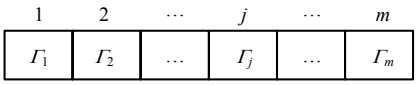

Fig.4 One-Dimensional integer vector coding

图 4 一维整数向量编码

2.2 种群初始化策略

由于本文采用的是测试资源的动态分配思想,希望把测试资源尽可能多地分配给可靠性低的模块,从而提高系统总体可靠性,那么在种群初始化的时候就需要优先考虑可靠性低的模块,以提高进化的速度.首先,针对如图 5 所示的一个包含 10 模块的串并行软件系统进行测试,观察模块可靠性、测试代价与测试时间的关系.

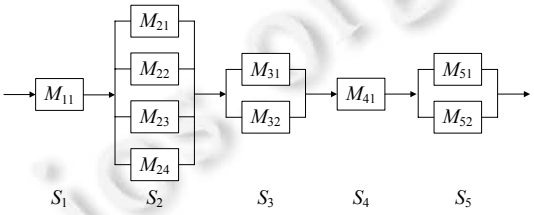

Fig.5 A series-parallel software systems of 10 modules

图 5 10 模块的串并行软件系统

以模块 M_{21} 为例,分析模块可靠性与测试资源的关系.图 6 给出了公式(5)中模块可靠性与测试时间的函数曲线.

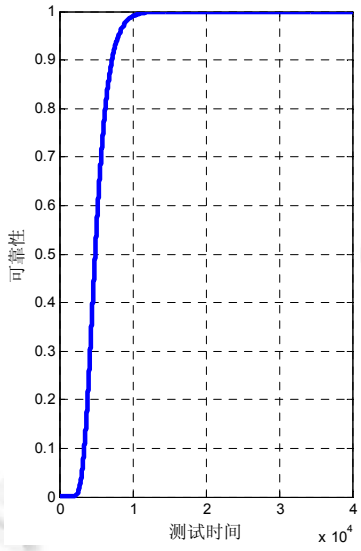

Fig.6 For module M_{21}, the relations of reliability and testing source

图 6 模块 M_{21} 可靠性与测试资源的关系

从图 6 可以看出,当一个模块的测试资源达到某一数量之后,其可靠性趋向于某一定值.也就是说,在模块的可靠性达到某一值之后,即使再分配大量的测试资源,也不会使模块的可靠性得到大幅度的提升.因此,从节省测试资源的角度出发,我们应该在不同的测试阶段采用不同的种群初始化方法.

种群的初始化方法具体如下:

(1) 对于第 i=1 个测试阶段(T_0=0,T_1].

这时,系统各个模块的可靠性都较低.可以按照公式(21)的方式随机初始化每个模块的测试资源.初始种群表示为 $\dot{\mathbf{T}}_l^0=[\Gamma_{l1}^0,...,\Gamma_{l2}^0,...,\Gamma_{lm}^0]$,其中,$\Gamma_{lj}^0=[t_{j1},t_{j2},...,t_{jn_j}]$.$\Gamma_{lj}^0=[t_{j1},t_{j2},...,t_{jn_j}]$ 的初始化如公式(21)所示:

$$t_{jk}=\begin{cases}rand(0,T_1), & k=1\\ rand\left(0,T_1-\sum_{j'=1}^{j-1}\max_{k'}\{t_{j'k'}\}\right), & k=2,3,...,n_j\end{cases} \tag{21}$$

其中,$rand(u_1,u_2)$函数表示在区间(u_1,u_2)内随机取整数.

以图 5 所示的 10 模块系统为例,说明种群初始化的过程.也就是说,对于模块 M_{11},在(T_0=0,T_1]之间随机生成测试资源数 t_{11};对于子系统 S_2,分别对模块 M_{21},M_{22},M_{23} 和 M_{24} 在剩余的资源区间(0,T_1−t_{11})内分别生成随机资源数 t_{21},t_{22},t_{23} 和 t_{24};对于子系统 S_3,分别对模块 M_{31} 和 M_{32} 在剩余的资源区间 $\left(0,T_1-t_{11}-\max_{k'=1}^{4}\{t_{2k'}\}\right)$ 内分别生成随机资源数 t_{31} 和 t_{32};依次类推,为每一个模块生成一个测试资源.

(2) 对于第 i 个(i≥2)测试阶段(T_{i-1},T_i].

因为经过在此之前测试阶段的资源分配,有些模块的可靠性比较高,根据前面的结论,即使再分配大量的测试资源,也不会使系统的可靠性得到大幅度的提升.

因此,本文的思路是把测试资源尽可能多地分配给可靠性低的模块,从而提高系统的整体可靠性.同情形(1),假设初始种群表示为 $\dot{\mathbf{T}}_l^0=[\Gamma_{l1}^0,...,\Gamma_{l2}^0,...,\Gamma_{lm}^0]$,其中,$\Gamma_{lj}^0=[t_{j1},t_{j2},...,t_{jn_j}]$.$\Gamma_{lj}^0=[t_{j1},t_{j2},...,t_{jn_j}]$ 的初始化如公式(22)所示:

$$t_{jk}=\begin{cases}rand\left(0,T_i-T_{i-1}-\sum_{j'=1}^{j-1}\max_{k'}\{t_{ij'k'}\}\right), & r_{jk}<\theta\\ 0, & r_{jk}\geqslant\theta\end{cases} \tag{22}$$

其中,r_{jk} 表示模块 M_{jk} 在进入测试阶段(T_{i-1},T_i]之前已经达到的可靠性;θ是阈值,表示用户希望达到的可靠性值.

2.3 变 异

通过改进 DE 算法的变异操作方式,人们提出了许多模式.DE 算法的不同模式可以表示为 DE/a/b/c 的形式.其中,a 表示当前被变异的个体是“随机的”或“最佳的”,b 代表所利用的差向量的个数,c 指示交叉程序的操作方法.其表达形式分别如下[30]:

1) DE/rand/1/bin:

$$\tilde{\dot{\mathbf{T}}}=\dot{\mathbf{T}}_{x_1}+F\cdot(\dot{\mathbf{T}}_{x_2}-\dot{\mathbf{T}}_{x_3}) \tag{23}$$

2) DE/best/1/bin:

$$\tilde{\dot{\mathbf{T}}}=\dot{\mathbf{T}}_{best}+F\cdot(\dot{\mathbf{T}}_{x_2}-\dot{\mathbf{T}}_{x_3}) \tag{24}$$

3) DE/rand-to-best/1/bin:

$$\tilde{\dot{\mathbf{T}}}=\dot{\mathbf{T}}_{x_1}+F\cdot(\dot{\mathbf{T}}_{best}-\dot{\mathbf{T}}_{x_1})+F\cdot(\dot{\mathbf{T}}_{x_2}-\dot{\mathbf{T}}_{x_3}) \tag{25}$$

4) DE/rand/2/bin:

$$\tilde{\dot{\mathbf{T}}}=\dot{\mathbf{T}}_{x_5}+F\cdot(\dot{\mathbf{T}}_{x_1}-\dot{\mathbf{T}}_{x_2})+F\cdot(\dot{\mathbf{T}}_{x_3}-\dot{\mathbf{T}}_{x_4}) \tag{26}$$

5) DE/best/2/bin:

$$\tilde{\dot{\mathbf{T}}}=\dot{\mathbf{T}}_{best}+F\cdot(\dot{\mathbf{T}}_{x_1}-\dot{\mathbf{T}}_{x_2})+F\cdot(\dot{\mathbf{T}}_{x_3}-\dot{\mathbf{T}}_{x_4}) \tag{27}$$

其中,F 为缩放因子,$F\in[0,2]$;$\tilde{\mathbf{I}}$ 是变异后的个体;x_1,x_2,x_3,x_4,x_5 是随机整数,表示个体在种群中的序号;$\tilde{\mathbf{I}}_{best}$ 是当代最优个体.

针对模型(17)进行大量的实验,发现模式 DE/best/2/bin 的 DE 算法效果最好.因此,本文采用如公式(27)所示的变异操作.

2.4 交 叉

在 DE 算法中,交叉操作是对原来个体 $\mathbf{I}_l=[\Gamma_{l1},...,\Gamma_{lj},...,\Gamma_{lm}]$ 和变异后的个体 $\tilde{\mathbf{I}}_l=[\tilde{\Gamma}_{l1},...,\tilde{\Gamma}_{lj},...,\tilde{\Gamma}_{lm}]$ 进行的,常采用的交叉操作是位交叉.设交叉后的个体 $\tilde{\mathbf{I}}_{l'}=[\tilde{\Gamma}_{l'1},...,\tilde{\Gamma}_{l'j},...,\tilde{\Gamma}_{l'm}]$ 和 $\tilde{\mathbf{I}}_{l''}=[\tilde{\Gamma}_{l''1},...,\tilde{\Gamma}_{l''j},...,\tilde{\Gamma}_{l''m}]$,则

$$\tilde{\Gamma}_{l'j}=\begin{cases}\Gamma_{lj}, rand_j(0,1)\leqslant CR\\ \tilde{\Gamma}_{lj}, rand_j(0,1)>CR\end{cases} \tag{28}$$

$$\tilde{\Gamma}_{l''j}=\begin{cases}\tilde{\Gamma}_{lj}, rand_j(0,1)\leqslant CR\\ \Gamma_{lj}, rand_j(0,1)>CR\end{cases} \tag{29}$$

其中,CR 是交叉因子,$CR\in[0,1]$,控制个体间交叉的程度;$rand(0,1)$函数表示在区间(0,1)内取得随机数.

2.5 选 择

选择操作是在第 G 代原个体 $\mathbf{I}_l^G$、交叉后的个体 $\tilde{\mathbf{I}}_{l'}$ 和 $\tilde{\mathbf{I}}_{l''}$ 之中选出最优的 Pareto 个体进入下一代,具体的选择方法如定义 3.

2.6 解 码

对于串并行软件系统的测试资源动态分配问题,进行解码的原因有两个:

- 第一,获取每个测试阶段最优的编码结果,即获得在系统达到 Pareto 最优时各模块实际消耗的测试资源、可靠性和成本.
- 第二,在实际分配过程中,系统测试可能不需要这么多的资源,也就是说,实际测试时间 $T_i^a<T_i-T_{i-1}$.这时,下一个测试阶段可能提前,即下一个测试阶段的测试资源由 $T_{i+1}-T_i$ 变为 $T_{i+1}-T_i+(T_i-T_{i-1}-T_i^a)$.因此,在每一测试阶段找到最优解之后需要解码,获得实际的测试时间 T_i^a.

因此,需要在每个测试阶段 DE 进化到最优解之后进行解码操作.具体操作可以描述如下:

Step 1. 根据公式(5)计算每个模块的可靠性 r_{jk}.

Step 2. 根据公式(11)计算每个模块的测试代价 C_{jk}.

Step 3. 根据公式(16)计算系统的实际测试时间 T_i^a.

3 对比实验

本节将本文的测试资源动态分配模型与文献[14]中采用的模型进行对比,验证模型的有效性;并且,观察不同的测试阶段数目对最优解的影响.公式(30)为文献[14]的测试资源分配模型:

$$\left.\begin{aligned}&\text{Max } R=\prod_{l=1}^{n_p}\left\{1-\prod_{i=1}^{k_l}[1-r_{li}(x/t_{li})]\right\}\prod_{j=1}^{n_s}r_j(x/t_j)\\&\text{Min } C_i=\sum_{l=1}^{n_p}\sum_{i=1}^{k_l}C_{li}(r_{li})+\sum_{j=1}^{n_s}C_j(r_j)\\&\text{s.t.}\\&\sum_{l=1}^{n_p}\sum_{i=1}^{k_l}t_{li}+\sum_{j=1}^{n_s}t_j\leqslant T\\&t_{li},t_j\geqslant 0\end{aligned}\right\} \tag{30}$$

在此模型中,n_p,n_s 分别为系统中并行子系统和串行子系统的数目,k_l 为第 l 个并行子系统所包含的冗余模块

数目,r_{li},t_{li},C_{li} 分别为第 l 个并行子系统的第 i 个冗余模块的可靠性、测试资源和测试代价,r_j,t_j,C_j 分别为第 j 个串行子系统的可靠性、测试资源和测试代价.

3.1 实验环境

假设现有 23 人从事软件测试工作,每个人可以使用的测试时间为 1 000 小时,在如图 5 所示的串并行软件系统上比较测试资源动态分配模型与文献[14]中的模型.为了对比的公平性,本文参考文献[14]的参数设置方法,在不同的区间内采用随机取值的方式进行,每个参数的生成区间见表 1.文献[14]中的模型是不考虑测试阶段的,本文称为静态模型,也可以认为是只有一个测试阶段的动态分配模型,即 p=1.

Table 1 Model parameter generation interval

表 1 模型参数生成区间

	a_{jk}	b_{jk}	x_{jk}	y_{jk}	z_{jk}
M_{11}, M_{41}	[30,35]	[5.8×10^{-3},6.2×10^{-3}]	[3.4,3.55]	[6.0,6.2]	[4.0,4.1]
M_{21}, M_{22}, M_{23}, M_{24}, M_{31}, M_{32}, M_{51}, M_{52}	[200,350]	[3×10^{-4},9×10^{-4}]	[3.4,3.55]	[6.0,6.2]	[4.9,5.1]

采用 DE 算法求解具有不同测试阶段数目的动态分配模型.DE 的参数设置如下:种群规模 N=20,迭代次数 G=200,缩放系数 F=0.5,交叉概率 CR=0.5.

3.2 实验结果与分析

我们首先进行一次随机实验,图 7 给出了不同测试阶段数的 Pareto 解集的分布图.这里的系统可靠性是指在整个测试完成后系统的总体可靠性,系统测试代价是指整个测试完成后系统总体所付出的测试代价,系统测试时间是指整个测试完成后系统实际消耗的测试时间.

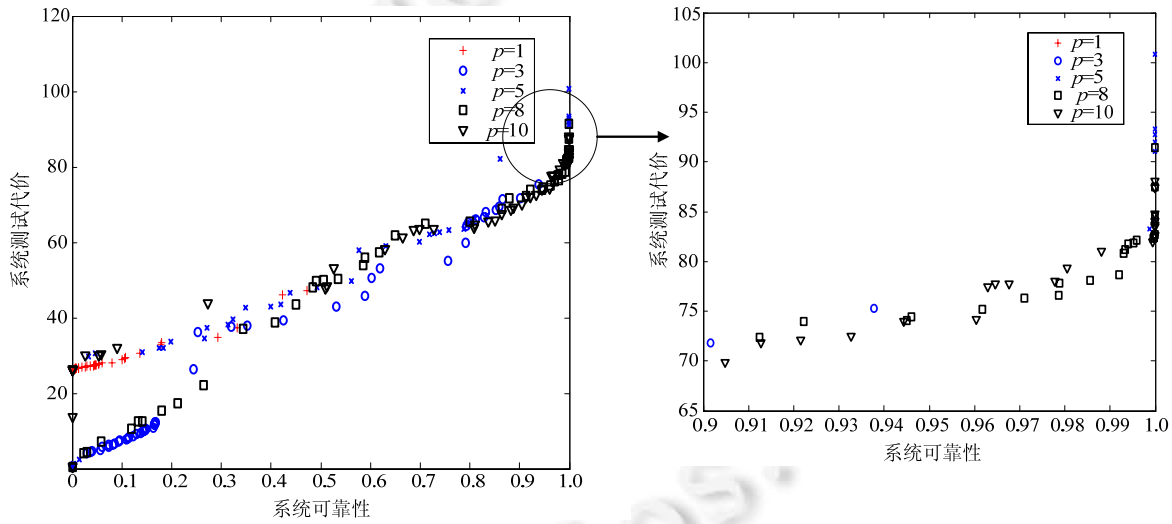

Fig.7 Distribution of Pareto solutions of different numbers of test phases in a randomized experiment

图 7 某次随机实验,不同数目测试阶段的 Pareto 解集分布图

从图 7 可以看出,当 p=1 时,即参考文献[14]中的静态分配模型,一次随机实验的 pareto 解均集中在可靠性低的区域,可靠性最大值也只有 0.471 577;当 $p\neq1$ 时,随着 p 的增加,Pareto 解集越来越向可靠性高的区域偏移;当 $p\geqslant8$ 时,Pareto 解集平均分布在 0.999 9 周围.

表 2 给出了不同测试阶段数可靠性达到最大时对应的 Pareto 最优解,同时给出了 Pareto 最优解对应的测试代价、测试时间和程序运行时间.由于可靠性和测试代价与测试时间的关系并不是严格的递增函数(如图 6 所示),最优解中可能存在测试时间浪费的情况,因此,最优解对应的可靠性和测试代价与测试时间并没有明显的递增或递减关系.但是由于测试阶段数目的增加,每个测试阶段均迭代 200 次,那么程序运行时间出现增加的现

象.因此,在实际的测试资源分配问题中,可以根据需要选择合适的 p,以满足实际需求.

Table 2 Solution of the maximum system reliability of different numbers of test phases in a randomized experiment

表 2 某次随机实验,不同测试阶段数目,系统可靠性最大时的解

	p=1	p=3	p=5	p=8	p=10
系统可靠性	0.471 577	0.937 797	0.999 976	0.999 998	0.999 999
系统测试代价	47.210 878	75.270 063	100.833 471	91.433 065	88.092 116
系统测试时间(人*小时)	8 900	9 392	8 887	9 616	9 463
程序运行时间(s)	0.121 000	0.394 000	0.775 000	1.186 000	1.288 000

为了进一步验证模型的有效性,本文进行了 10 次重复实验,图 8 给出了不同测试阶段数的 Pareto 解集平均分布图.从图 8 可以看出,即使经过 10 次实验,采用文献[14]中的模型,即 p=1 时,Pareto 解集也分布在 0.7 左右;而当 $p\geqslant 3$ 时,Pareto 解集分布在 0.9~1 之间.也就是说,采用本文的测试资源分配模型对测试资源进行动态分配,可以改善测试的性能,尽可能地提高系统的可靠性.

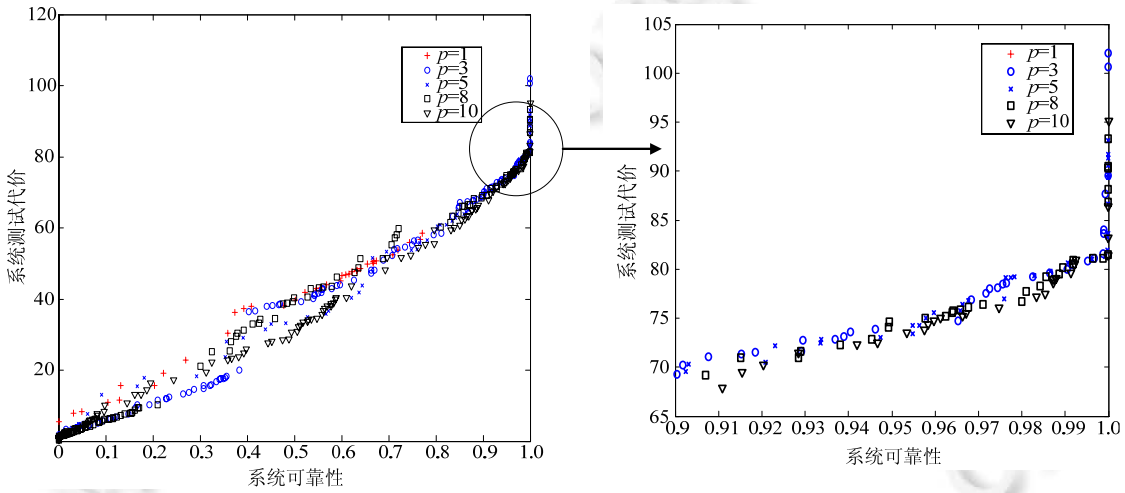

Fig.8 Distribution of Pareto solutions of different numbers of test phases for 10 repeated times randomized experiments

图 8 重复 10 次实验,不同数目测试阶段的 Pareto 解集分布图

表 3 给出了不同测试阶段数可靠性达到最大时对应的 Pareto 最优解的平均解.由表 3 可知,随着测试阶段数目的增加,系统的可靠性将不断增加,而消耗的测试资源却在减少,进一步验证了本文动态分配模型可以节约测试资源、提高测试性能的结论.但由于测试阶段增加,在每一个测试阶段都需要进行寻优操作,会在一定程度上增加程序运行的时间.

Table 3 Average solution of the maximum system reliability of different numbers of test phases in 10 repeated times randomized experiments

表 3 10 次随机实验,不同测试阶段数目,系统可靠性最大时的平均解

	p=1	p=3	p=5	p=8	p=10
系统可靠性	0.692 356	0.999 983	0.999 988	0.999 999	0.999 999
系统测试代价	54.340 058	80.890 534	83.873 532	84.367 851	83.898 993
系统测试时间(人×小时)	9 508	9 487	9 463	9 397	9 373
程序运行时间(s)	0.121 900	0.393 900	0.776 400	1.190 100	1.298 100

综上,本文的测试资源动态分配模型及其求解方法在一定程度上可以节省测试资源,提高测试性能,增加系

统的可靠性.

4 结束语

系统测试作为软件开发各个阶段中最消耗时间和资源的阶段,对于串并行软件系统来说,系统可靠性随着测试进程的推进会发生变化,如果再按照最初的方案分配测试资源,可能会造成测试资源的浪费,这时需要分阶段对测试资源进行再分配.本文针对复杂串并行软件系统中测试资源分配问题,首先构建了考虑可靠性时变的测试资源多目标动态分配模型,以测试资源(测试时间)为约束、以最大化可靠性和最小化测试代价为目标;然后,基于"一维整数向量编码"DE 算法进行求解,并设计了新的种群初始化策略和解码机制.对比实验结果表明,测试资源的动态分配在一定程度上可以节省系统测试资源的消耗,提高复杂软件系统的可靠性,从而在测试资源分配问题研究领域,对深化测试资源分配问题的研究起到了一定的推动作用.

References:

[1] Lo JH, Sy-yen K, Lyu MR, Huang CY. Optimal resource allocation and reliability analysis for component-based software applications. In: Proc. of the 26th Annual Int'l Computer Software and Applications Conf. Los Alamitos: IEEE Computer Society, 2002. 7−12. [doi: 10.1109/CMPSAC.2002.1044526]

[2] Kapur PK, Jha PC, Bardhan AK. Optimal allocation of testing resource for a modular software. Asia-Pacific Journal of Operational Research, 2004,21(3):333−354. [doi: 10.1142/S0217595904000278]

[3] Li X, Xie M, Ng SH. A general formulation of optimal testing-time allocation for modular systems. In: Proc. of the IEEE Int'l Conf. on Industrial Engineering and Engineering Management (IEEM 2009). Hong Kong, 2009. 252−256. [doi: 10.1109/IEEM.2009.5373369]

[4] Huang CY, Lyu MR. Optimal testing resource allocation, and sensitivity analysis in software development. IEEE Trans. on Reliability, 2005,54(4):592−603. [doi: 10.1109/TR.2005.858099]

[5] Huang CY, Lo JH. Optimal resource allocation for cost and reliability of modular software systems in the testing phase. Journal of Systems and Software, 2006,79(5):653−664. [doi: 10.1016/j.jss.2005.06.039]

[6] Kapur PK, Bardhan AK, Yadavalli VS. On allocation of resources during testing phase of a modular software. Int'l Journal of Systems Science, 2007,38(6):493−499. [doi: 10.1080/00207720701353504]

[7] Kamel R, Xin Y, Thompson HH. An efficient sampling scheme for estimating software reliability with associated cost. Far East Journal of Mathematical Sciences, 2008,28(2):353−366.

[8] Turner R. Optimized allocation of testing budget for missile defense vehicle. In: Proc. of the Annual Reliability and Maintainability Symp. Orlando, 2007. 250−253. [doi: 10.1109/RAMS.2007.328063]

[9] Aggarwal AG, Kapur PK, Kaur G, Kumar R. Genetic algorithm based optimal testing effort allocation problem for modular software. BVICAM's Int'l Journal of Information Technology, 2012,4(1):445−451.

[10] Kapur PK, Aggarwal AG, Kapoor K, Kaur G. Optimal testing resource allocation for modular software considering cost, testing effort and reliability using genetic algorithm. Int'l Journal of Reliability, Quality and Safety Engineering, 2009,16(06):495−508. [doi: 10.1142/S0218539309003538]

[11] Jha PC, Gupta D, Bo Y, Kapur PK. Optimal testing resource allocation during module testing considering cost, testing effort and reliability. Computers & Industrial Engineering, 2009,57(3):1122−1130. [doi: 10.1016/j.cie.2009.05.001]

[12] Kumar V, Arora HD, Taneja N, Sahni R. On allocation of resources during testing and debugging phase using flexible SRGM: A genetic algorithm approach. Int'l Journal of Information & Computation Technology, 2014,4(11):1085−1086.

[13] Dai YS, Xie M, Poh KL, Yang B. Optimal testing-resource allocation with genetic algorithm for modular software systems. Journal of Systems and Software, 2003,66(1):47−55. [doi: 10.1016/S0164-1212(02)00062-6]

[14] Wang Z, Tang K, Yao X. A multi-objective approach to testing resource allocation in modular software systems. In: Proc. of the 2008 IEEE Congress on Evolutionary Computation (CEC 2008). Hong Kong, 2008. 1148−1153. [doi: 10.1109/CEC.2008.4630941]

[15] Yu S, Fei D, Bin L. Optimal testing resource allocation for modular software systems based-on multi-objective evolutionary algorithms with effective local search strategy. In: Proc. of the 2013 IEEE Workshop on Memetic Computing (MC). Singapore, 2013. 1−8. [doi: 10.1109/MC.2013.6608200]

[16] Yin PY, Wang JY. Optimal multiple-objective resource allocation using hybrid particle swarm optimization and adaptive resource bounds technique. Journal of Computational and Applied Mathematics, 2008,216(1):73−86. [doi: 10.1016/j.cam.2007.04.018]

[17] Jia Z, Gong L. Multi-Criteria human resource allocation for optimization problems using multi-objective particle swarm optimization algorithm. In: Proc. of the 2008 Int'l Conf. on Computer Science and Software Engineering. Wuhan, 2008. 1187−1190. [doi: 10.1109/CSSE.2008.1506]

[18] Chaharsooghi SK, Meimand Kermani AH. An effective ant colony optimization algorithm (ACO) for multi-objective resource allocation problem (MORAP). Applied Mathematics and Computation, 2008,200(1):167−177. [doi: 10.1016/j.amc.2007.09.070]

[19] Fonseca CM, Fleming PJ. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: Proc. of the 5th Int'l Conf. on Genetic Algorithms (ICGA'93). Urbana-Champaign, 1993. 416−423.

[20] Srinivas N, Deb K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation, 1994,2(3):221−248. [doi: 10.1162/evco.1994.2.3.221]

[21] Holland JH. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Cambridge: MIT Press, 1992.

[22] Horn J, Nafpliotis N, Goldberg DE. A niched Pareto genetic algorithm for multiobjective optimization. In: Proc. of the 1st IEEE Conf. on Evolutionary Computation, IEEE World Congress on Computational Intelligence. Orlando, 1994. 82−87. [doi: 10.1109/ICEC.1994.350037]

[23] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. IEEE Trans. on Evolutionary Computation, 1999,3(4):257−271. [doi: 10.1109/4235.797969]

[24] Eckart Z, Laumanns M, Thiele L. SPEA2: Improving the strength pareto evolutionary algorithm. Technical Report, TIK-Report 103, Zurich: Swiss Federal Institute of Technology Zurich (ETH), Computer Engineering and Networks Laboratory (TIK), 2001. 1−21. [doi: 10.3929/ethz-a-004284029]

[25] Knowles J, Corne D. The Pareto archived evolution strategy: A new baseline algorithm for Pareto multiobjective optimisation. In: Proc. of the 1999 Congress on Evolutionary Computation. Washington, 1999. 98−105. [doi: 10.1109/CEC.1999.781913]

[26] Knowles JD, Corne DW. M-PAES: A memetic algorithm for multiobjective optimization. In: Proc. of the 2000 Congress on Evolutionary Computation. La Jolla, 2000. 325−332. [doi: 10.1109/CEC.2000.870313]

[27] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. on Evolutionary Computation, 2002,6(2):182−197. [doi: 10.1109/4235.996017]

[28] Zhang GF, Jiang JG, Lu CH, Su ZP, Fang H, Liu Y. A revision algorithm for invalid encodings in concurrent formation of overlapping coalitions. Applied Soft Computing Journal, 2011,11(2):2164−2172. [doi: 10.1016/j.asoc.2010.07.015]

[29] Zhang GF, Jiang JG, Su ZP, Qi MB, Fang H. Searching for overlapping coalitions in multiple virtual organizations. Information Sciences, 2010,180(17):3140−3156. [doi: 10.1016/j.ins.2010.04.028]

[30] Liang JJ, Qu BY, Mao XB, Niu B, Wang DY. Differential evolution based on fitness Euclidean-distance ratio for multimodal optimization. Neurocomputing, 2014,137(2014):252−260. [doi: 10.1016/j.neucom.2013.03.069]

陆阳(1967－),男,安徽合肥人,博士,教授,博士生导师,CCF 高级会员,主要研究领域为可靠性工程,分布式控制技术.

苏兆品(1983－),女,博士,副教授,主要研究领域为智能计算,多媒体安全.

岳峰(1981－),男,博士,副研究员,主要研究领域为可靠性设计,智能计算.

王永奇(1992－),男,硕士,主要研究领域为智能计算,灾害应急管理.

张国富(1979－),男,博士,副教授,主要研究领域为智能计算,MAS 理论.