基于故障的布尔表达式测试技术综述

孙昌爱[1,2] 程庆顺[1]
(北京科技大学计算机通信与工程学院 北京 100083)[1]
(中国科学院软件研究所计算机科学国家重点实验室 北京 100190)[2]

摘 要 布尔表达式是软件规格说明与程序实现中的关键成份,严重影响软件的正确性。如何有效地测试布尔表达式是软件测试领域的一个重要研究方向。介绍了基于故障的软件测试的基本概念与原理,归纳了布尔表达式的各种故障类型,讨论了不同故障类型之间的检测包含关系;评述了几类代表性的面向布尔表达式的测试策略,提出了一种比较框架并比较了各种策略的适用情形、故障检测能力与测试用例精简程度。针对现有研究工作的不足,探讨了几个值得研究的问题,简要介绍了近年来此领域的研究工作。

关键词 布尔表达式,基于故障的软件测试,测试策略,测试用例生成,故障类型
中图法分类号 TP311.5 **文献标识码** A

Survey on Fault-based Testing Techniques for Boolean Expressions

SUN Chang-ai[1,2] CHENG Qing-shun[1]
(School of Computer and Communication Engineering, University of Science and Technology Beijing, Beijing 100083, China)[1]
(State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[2]

Abstract Boolean expressions play a key role in specifications and programs, and thus significantly affect the correctness of software. How to efficiently test Boolean expressions is an important issue. We first introduced the basic concepts and principle of fault-based testing. We then summarized several fault classes of Boolean expressions and discussed their detection hierarchies. Next, we reviewed several representative fault-based testing strategies for Boolean expressions, and proposed a comparison framework to compare them with respect to the applicability, fault detection capability and reduction of test suites. Finally, we pointed out several future issues based on our observations of limitations on the existing works, and briefly introduced our relevant work in this area.

Keywords Boolean expressions, Fault-based testing, Testing strategies, Test case generation, Fault types

1 引言

基于故障的软件测试(fault-based testing)[1]是近年来提出的一种新型软件测试技术。其基本思想是,假设一个被测试软件中存在特定种类的故障类型,然后根据这些故障类型设计测试用例集,用产生的测试用例集去检测被测软件中是否存在某种类型的故障。由于布尔表达式是软件规格说明与程序实现中的关键成份,基于故障的布尔表达式测试的研究非常活跃,人们分析了布尔表达式中不同故障类型之间的包含关系,开发出多种有效的测试策略。

本文试图勾勒出布尔表达式领域基于故障的测试研究线路图,总结与归纳该领域的研究进展。首先,介绍布尔表达式的相关概念与基于故障的软件测试基本原理。其次,归纳布尔表达式的常见故障类型,讨论不同故障类型之间的包含关系。然后,总结基于故障的测试策略,分析与比较各种测试策略的适用情形、故障检测能力和测试用例精简程度。最后,针对已有研究工作的不足,探讨尚需要进一步研究的问题,介绍我们的一些研究工作。

2 基本概念与术语

布尔表达式广泛存在于规格说明与程序中[2],如规格说明中大量的业务逻辑是由判断条件控制的,实现程序控制结构的一个重要组成部分是条件表达式(包括分支条件和循环条件)。所谓布尔表达式(Boolean Expression),就是通过逻辑操作符"与"、"或"、"非"将布尔变量连接形成以结果为"真"或"假"的表达式。在程序或规格说明中,布尔表达式中的布尔变量可能对应于一个关系表达式、算术表达式或逻辑表达式。例如,下述C语言分支语句中" if ((i>0) && (i<10)) i=i+m; else i=i-m;",条件表达式为"(i>0) && (i<10)",其中,将关系表达式"i>0"和"i<10"分别看作布尔变量

到稿日期:2012-05-29 返修日期:2012-08-24 本文受国家自然科学基金(60903003),北京市自然科学基金(4112037),中国科学院软件研究所计算机科学国家重点实验室开放课题(SYSKF1105),中央高校基本科研业务费资助项目(FRF-SD-12-015A),北京市优秀人才培养资助项目(2012D009006000002)资助。

孙昌爱(1974—),男,博士,副教授,CCF高级会员,主要研究方向为软件测试、程序分析、服务计算,E-mail:casun@ustb.edu.cn;程庆顺(1987—),男,硕士,主要研究方向为软件测试。

"A"和"B",则对应的布尔表达式为"A 与 B"。逻辑操作符有不同的表示方式,常用"·"、"*"、"∧"表示"与"操作符,用"+"或"∨"表示"或"操作符,用"—"或"!"表示"非"操作符,用"T"或"1"表示"真"值,用"F"或"0"代表"假"值[3-6]。部分文献也称布尔表达式为逻辑表达式、谓词表达式或布尔规格说明。本文约定:用缺省、"+"、"—"分别表示操作符"与"、"或"和"非",用"T"和"F"分别表示"真"和"假"。

2.1 不同的表示形式的布尔表达式

定义 1 一个布尔表达式仅由布尔变量及其否定构成的合取式组成,称该表达式为简单合取式,简记为 CNF。如果一个布尔表达式仅由一些简单合取式的析取组成,则称该表达式为析取式,简记为 DNF,将每个简单合取式称为一个乘积项,把乘积项中的布尔变量的一次出现称为布尔文字,布尔文字分为正文字和负文字。

定义 2 对于一个给定的 DNF,如果在不改变它的值的情况下,没有一个乘积项或者布尔文字可以被去掉,则称该 DNF 为最简析取式,简记为 IDNF。

定义 3 如果一个布尔表达式由"与"、"或"、"非"、与"扩号"将一个或多个文字连接而成,称该表达式为一般形式,简记为 GF。

定义 4 如果布尔表达式中每个布尔变量仅出现一次,而且没有公共布尔变量,则称该布尔表达式为单一表达式,简记为 SF。

2.2 布尔表达式的值集

布尔表达式 S 包含 n 个布尔变量,S 的输入空间记作 $B^n=\{X_1X_2,\cdots,X_n\}$,由于每个布尔变量 $X_i(1\leqslant i\leqslant n)$ 的取值为"真"与"假",为了对 S 进行穷尽测试,则需要产生 2^n 个测试用例,即 $\|B^n\|=2^n$,每个测试用例 $TC_j(1\leqslant j\leqslant 2^n)$ 对应于一个 n 维的布尔向量。假设布尔表达式 S 为一个包含 n 个布尔变量的最简析取式(IDNF),记为 $S=TM_1+TM_2+\cdots+TM_m$。其中,S 的第 i 个乘积项 $TM_i=X_1^iX_2^i,\cdots,X_{k_i}^i$,$m$ 为 S 中乘积项的个数,$1\leqslant i\leqslant m$,$1\leqslant K_i\leqslant n(k_i$ 是第 i 个乘积项中的布尔文字的个数)。为了便于后文讨论,对 S 的输入空间进行划分,并定义不同的值集。

定义 5 真值点 $TP=\{t|S(t)=T\}$,其中,t 表示 S 的测试输入(即测试用例),$S(t)$ 表示测试输入为 t 时 S 的布尔值。

定义 6 假值点 $FP=\{t|S(t)=F\}$。

定义 7 唯一真值点 $UTP=\bigcup_{i=1}^{m}TP_i$,其中,$TP_i=\{t\mid TM_i(t)=T\wedge(\forall j(j\neq i\wedge TM_j(t)=F)\}$,$TM_i(t)$ 表示测试输入为 t 时乘积项 TM_i 的布尔值。

定义 8 重叠真值点 $OTP=TP\backslash UTP$。

定义 9 邻近假值点 $NFP=\bigcup_{i=1}^{m}\bigcup_{j=1}^{n}FP_{i,j}$,其中,$FP_{i,j}=\{t\mid t\in FP_i\wedge TM_i(x_j^i/\overline{X_j^i})(t)=T\}$,$FP_i=\{t\mid TM_i(t)=F\}$,$TM_i(x_j^i/\overline{X_j^i})$ 表示用 $\overline{X_j^i}$ 替换乘积项 TM_i 中的 x_j^i 得到的乘积项。

定义 10 剩余假值点 $RFP=FP\backslash NFP$。

2.3 基于故障的软件测试

软件测试是实践中广泛使用的质量保证手段。早期,软件测试被错误地定位为"试图证明程序是正确",著名计算机科学家 Edsger Dijkstra 指出软件测试"不能证明程序是正确的,而只能表明存在错误"[7]。为了提高软件的可靠性,应尽可能地检测并消除软件中潜藏的各种故障,人们发现"一般情况下,实施充分的测试也是不可能的,而只能实施相对充分的测试"。换言之,针对某些种类的故障进行相对充分的测试是可能的,这导致了基于故障的软件测试技术的出现。

目前,基于故障的软件测试是一种比较高效的测试手段[1]。对于一个给定的被测程序 p,如何判定一组测试用例集合 ts 是否充分呢?变异分析是一种典型的基于故障的测试技术[8,9],广泛用来评价测试用例集的充分性。另外,基于常见故障类型的程序结构或行为特征,提出了各种程序故障识别与检测技术[10],如检测程序内存泄露、安全性、可信性等[11]。基于软件故障在输入域上的群束特征,开发了各种有效的软件测试技术,如适应性随机测试[12]、动态随机测试[13]。

3 布尔表达式的常见故障类型及其检测包含关系

基于故障的软件测试在布尔表达式领域的研究历史较长,已经取得丰富的成果。

3.1 故障类型

表 1 总结了布尔表达式的常见故障类型。由于不同文献对同一故障类型的称谓不同,所讨论的布尔表达式的表示形式与假设也有所不同,本文遵循文献[3,15]的定义与术语规范,用 S_F^E 表示布尔表达式 S 的错误执行[15],其中 F 表示故障类型,包括逻辑非、插入、缺失、引用 4 种,E 标识故障发生的对象和位置,包括表达式、乘积项、文字、运算符、变量和扩号。表 1 中以"—"表示此类故障类型不存在或不适用。例如,可以认为 ERF、ONF 等故障不存在,而 OIF 不适用(因为该故障无法通过语法检查)。除扩号相关的故障类型之外,表 1 中的各种故障类型都是针对精简析取式 IDNF 而言的。

表 1 布尔表达式的故障类型

Set / Parenthesis	Expression	Term	Literal	Operator	Variable
Negation	ENF	TNF	LNT	—	VNF	—
Insert	—	TIF	LIF	—	—	PIF
Omission	—	TOF	LOF	—	VOF	POF
Reference	—	——	LRF	ORF	VRF	—

各类故障类型简要说明如下:

(1)ENF[3,15]:整个表达式或其子表达式被错误地替换成其否定形式。

(2)TNF[3,15,16]:布尔表达式中的一个乘积项 TM_i 错误地替换成其否定形式。

(3)TIF:这类故障比较复杂,出现的概率也比较小。

(4)TOF[3,15,17]:布尔表达式中的一个乘积项 TM_i 缺失。

(5)LNF[3,15]:布尔表达式中的某个乘积项 TM_i 中的某个文字 x_j^i 被错误地替换成其否定形式。

(6)LOF[5,6,18]:布尔表达式中的某个乘积项 TM_i 中的某个文字 x_j^i 缺失。

(7)LIF[3,15]:布尔表达式中的某个乘积项 TM_i 中被错误地插入了其它布尔变量 x,存在以下 3 种情况:

(a)若 x 在 TM_i 中已经存在,该情形不构成故障;

(b)若 x 的否定形式在 TM_i 中已经存在,该类故障归入到 TOF 类型中;

(c)若 TM_i 中不存在 x,也不存在其否定形式 $\overline{x}$。如果没有特别指明,后文提到的 LIF 专指 7(c)情形。

(8)LRF[3,15]:布尔表达式中某个乘积项 TM_i 中某个文

字 x_j^i 错误地替换成了其它布尔变量 x，分为 4 种情形：

(a)若 x_j^i 被替换成了它的否定形式，即 $x=\overline{x_j^i}$，这种情形归入到 LNF 中；

(b)若 x 已经存在于 TM_i 中，且 $x\neq x_j^i$，这种情形归入到 LOF 中；

(c)若 x 的否定形式 $\overline{x}$ 已经存在于 TM_i 中，且 $x\neq x_j^i$，这种情形归入到 TOF 中；

(d)x 和 $\overline{x}$ 都不存在于 TM_i 中。如果没有特别指明，后文提到 LRF 专指 8(d)情形。

(9)ORF[3,15,19]：布尔表达式中的布尔操作符故障，分两种情形：

(a)某个“AND”操作符替换成了“OR”操作符，这种情形记作 ORF[·]；

(b)某个“OR”操作符替换成了“AND”操作符，这种情形记作 ORF[+]。

(10)VNF[5,14,15]：布尔表达式中的某个变量的每一次出现都被替换成了它的否定形式。部分文献混淆了 VNF 和 LNF。与 VNF 不同，LNF 是指布尔表达式中的某个变量的某一次出现都被替换成了它的否定形式。

(11)VRF[5,14,15]：布尔表达式中某个变量的每一次出现都被替换成了其它的变量。STF[0]和 STF[1]是两种特殊的 VRF(请参见后文讨论)。

(12)VOF：布尔表达式中某个变量缺失。部分文献也称该类故障为 MCF[20-22]。

(13)POF[3]：布尔表达式中的一对括号缺失。

(14)PIF[3]：在布尔表达式中错误地插入了一对括号。故障类型 POF 和 PIF 都是针对一般形式的布尔表达式 GF 而言的，这两种故障类型统称为 ASF[3,14]。

(15)STF[0][2,23,24]：布尔表达式中某个乘积项中的一个文字 x_j^i 错误地置为了 0 (False)，这种情形相当于 TOF。该故障也记作 SA0[20,21]。

(16)STF[1][2,23,24]：布尔表达式中的某个乘积项中的一个文字 x_j^i 错误地置为了 1：

(a)当 x_j^i 所在的乘积项中只有一个文字 x_j^i 时，$S^{x_j^i}_{STF[1]}=1$，只要有一个测试用例的值等于 0，这种故障就能检测出来。

(b)当 x_j^i 所在的乘积项中不只一个文字时，这种情形相当于 LOF。

该故障也记作 SA1[20,21]。不难看出，STF 这类故障可以转换为其它的故障类型讨论。

3.2 不同故障之间的检测包含关系

针对特定形式的布尔表达式，不同故障类型之间是否存在某些检测包含关系？所说的检测包含关系是指，如果故障类型 A 包含故障类型 B(记作 $A\Rightarrow B$)，意味着测试用例集合 S 如果能检测出故障类型 A，则 S 也一定能检测出故障类型 B。换言之，$TS(A)\subseteq TS(B)$，这里 $TS(A)$ 和 $TS(B)$ 分别表示检测故障类型 A 和 B 的测试用例集合。理解不同软件故障类型之间的检测包含关系(强弱关系)，不仅有助于解释实验结果，而且有助于减小测试代价(集中精力测试较强类型的故障)。

Kuhn[25]最早探索了不同故障类型之间的检测包含关系。假设布尔表达式表示为析取式 DNF，针对某个故障类型，计算能够保证检测出该类故障的测试用例集合所满足的条件。通过计算不同故障类型的检测条件，可以分析出几类故障的检测包含关系(也称层次关系)。Kuhn 得到的故障检测包含关系为：VRF⇒VNF⇒ENF。即故障类型 VRF 强于故障类型 VNF，故障类型 VNF 强于故障类型 ENF。换言之，能够检测出 VRF 的测试用例集一定也能检测出 VNF，能够检测出 VNF 的测试用例集一定也能检测出 ENF。其中，故障类型 VRF 分为 MCF(Missing Condition Faults)和 ICF(Incorrect Condition Faults)两种情况。

在 Kuhn 提出的故障关系中，没有充分研究故障类型 MCF 和故障类型 VRF 之间的检测包含关系。2002 年，Tsuchiya 和 Kikuno 进一步扩展了 Kuhn 提出的故障检测包含关系[22]。具体说来：(1)当故障类型 MCF 和 VNF 发生在同一变量时，MCF⇒VNF；(2)当故障类型 MCF 和 VRF 发生在同一变量而且发生故障的乘积项包含多个变量时，VRF⇒MCF；(3)其他情况下，MCF⇒VRF 和 VRF⇒MCF 均不成立。该扩展细化了 VRF、MCF 和 VNF 3 类故障类型的包含关系。为了提高测试效率并保证测试充分性，应选择能够检测包含单变量的故障类型 VRF 和 MCF 的测试用例集(即 TS(VRF) 与 TS(MCF)的交集)进行测试。

已知的故障检测包含关系仅仅考虑了变量相关的 3 类故障类型(VRF、VNF 和 ENF)之间的关系[22,25]。其中，故障类型 VRF 是指替换的所有变量均出现，然而故障可能仅与变量的某一次出现有关(即 LRF)。另外，由于布尔表达式由多个乘积项构成，因此故障可能发生在某个乘积项中(如 TOF、TNF)，而不是整个表达式中(如 ENF)。2005 年，Lau 和 Yu 进一步将故障类型的检测包含关系拓展到乘积项和文字层次上[15]。扩展后的故障检测包含关系为：(1)LIF⇒LRF⇒LNF⇒TNF⇒ENF；(2)LIF⇒TOF⇒ORF[+]⇒ENF；(3)LOF⇒ORF[·]⇒TNF；(4)TOF⇒LNF；(5)LOF⇒LNF。扩展后的故障类型检测包含关系，可以更全面地解释经验研究的结果，有助于设计基于故障的布尔表达式的测试策略。

之前讨论的不同故障类型的检测包含关系，假设被测的布尔表达式采用析取范式。2007 年，Kapoor 与 Bowen 两位学者将不同故障类型之间的检测包含关系拓展为一般形式的布尔表达式[21]。在拓展的故障检测包含关系中，涉及的 10 种故障类型分为两大类：与操作符相关的故障类型包括 ORF、ENF、VNF 和 ASF；与操作数相关的故障类型包括 MVF(Missing Variable Fault)、VRF、CCF(Clause Conjunction Fault)、CDF(Clause Disjunction Fault)、SA0 和 SA1。得到的故障包含关系为：VNF⇒ENF；MVF⇒VNF；VRF⇒VNF；CCF⇒VRF；CDF⇒VRF；CCF⇒SA0⇒VNF；CDF⇒SA1⇒VNF。需要指出的是，这里讨论的变量相关的故障类型仅考虑变量某一次出现所发生的故障，因此，VNF、ASF、MVF、VRF 和 CCF 分别对应于表 1 中的 LNF、POF、LOF、LRF 和 LIF。

Kapoor 与 Bowen 在讨论故障类型的检测包含关系时，由于忽视了等价类的存在，因此分析得到的检测包含关系存在错误。2011 年，Chen 等[20]指出了 Kapoor 与 Bowen 提出的故障检测包含关系中的错误，如下检测包含关系是不正确的：MVF⇒VNF，VNF⇒ENF，SA0⇒VNF，SA1⇒VNF，CCF⇒VRF 和 CDF⇒VRF。

图 1 总结了目前已证明的故障检测包含关系[3,26-28]。其

中，图 1(a)假设布尔表达式表示为析取范式(DNF)，图 1(b)假设布尔表达式采用一般形式(GF)；$A \rightarrow B$ 表示故障类型 A 包含故障类型 B(即 $A \Rightarrow B$)，或者故障类型 A 强于故障类型 B；⑯与⑰成立的前提条件是故障与同一变量有关；按表 1 的定义的故障类型，图 1(b)中的 VRF 对应于 LRF。理解不同故障类型之间的包含关系有助于设计高效的测试用例生成策略。

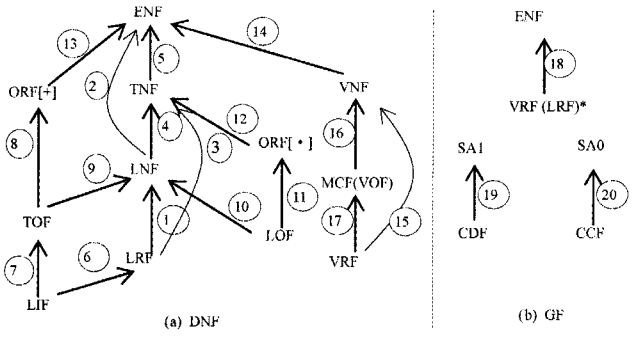

图 1　不同故障类型之间的包含关系

4　基于故障的测试策略

基于故障的软件测试的关键问题是如何产生与选择高效的测试用例集，即生成与选择的测试用例集中测试用例的数量尽可能小，同时可以检测尽可能多的故障类型。为此，人们提出了各种面向布尔表达式的测试用例生成策略。

4.1　逻辑覆盖准则

为了对构成程序的每个逻辑成份进行覆盖测试，人们提出了面向条件与判定的逻辑覆盖技术。1979 年，Myers[7] 提出了条件覆盖 CC 和判定覆盖 DC 测试准则。对于复杂的布尔表达式，CC 和 CD 无法对其进行充分的测试，人们又提出了一些更强的覆盖准则，如条件/判定覆盖 C/DC、修改的条件/判定覆盖等等[29-31]。具体说来：

(1)CC：程序中的每个条件的可能值都应覆盖到(即“真”与“假”至少出现一次)。

(2)DC：程序中的每个判定的可能值都应覆盖到。

(3)C/DC：程序中的每个条件和每个判定的可能值都覆盖到。

(4)MC/DC：程序中的每个条件和每个判定的可能值都应覆盖到，并且每个条件都独立地影响判定的结果。该覆盖准则强调了单个变量对于表达式值的影响能力。

(5)M-CC：程序中的每个条件的可能值的所有组合都应覆盖到。

4.2　有意义影响(Meaningful Impact)策略

1994 年，Weyuker、Goradia 和 Singh 提出了 BASIC 策略[14]。BASIC 策略与 MC/DC 准则的思想相似，生成的测试用例集应使得每个条件变量能够独立影响布尔表达式的值。在 BASIC 策略的基础上，提出了 MIN、ONE、MANY-A、MANY-B、MAX-A 和 MAX-B 策略[14,32]。

(1) BASIC 策略：从每个乘积项 TM_i 的唯一真值点 UTP_i 中选出一个测试用例，再从每个乘积项 TM_i 中的每个布尔文字 x_j^i 的邻近假值点 $NFP_{i,j}$ 中选择一个测试用例。

(2)MIN：该策略是对 BASIC 策略的简化。从每个乘积项 TM_i 的唯一真值点 UTP_i 中选出一个测试用例，再从每个乘积项 TM_i 中的每个布尔文字 x_j^i 的邻近假值点 $NFP_{i,j}$ 中选择一个测试用例，且要求选出的测试用例数尽可能少。

(3)MANY-A：如果乘积项 TM_i 的唯一真值点 UTP_i 中有 2^x 个测试用例，则从中选出$\lceil x \rceil$个测试用例；同样，如果乘积项 TM_i 中的布尔文字 x_j^i 的邻近假值点 $NFP_{i,j}$ 中有 2^x 个测试用例，则从中选出$\lceil x \rceil$个测试用例。如果 $x=0$，则选择一个测试用例。$\lceil x \rceil$表示上取整。

(4)MANY-B：该策略是对 MANY-A 策略的一种演化。除了从每个唯一真值点 UTP_i 和邻近假值点 $NFP_{i,j}$ 中选择$\lceil x \rceil$个测试用例，如果重叠真值点 OTP 中有 2^x 个测试用例，则从中选出$\lceil x \rceil$个测试用例；同样，如果剩余假值点 RFP 中有 2^x 个测试用例，则从中选出$\lceil x \rceil$个测试用例。如果 $x=0$，则选择一个测试用例。

(5)MAX-A：从每个乘积项 TM_i 的唯一真值点 UTP_i 中选出所有的测试用例，再从每个乘积项 TM_i 中的每个布尔文字 x_j^i 的邻近假值点 $NFP_{i,j}$ 中选择所有的测试用例。

(6)MAX-B：该策略是对 MAX-A 策略的一种演化。除了从每个唯一真值点 UTP_i 和邻近假值点 $NFP_{i,j}$ 中选择所有的测试用例，如果重叠真值点 OTP 中有 2^x 个测试用例，则从中选出$\lceil x \rceil$个测试用例；同样，如果剩余假值点 RFP 中有 2^x 个测试用例，则从中选出$\lceil x \rceil$个测试用例。如果 $x=0$，则选择一个测试用例。

4.3　MUMCUT 策略

1999 年，Chen、Lau 和 Yu 提出了 MUMCUT 策略[21]，它是 MUTP、MNFP 和 CUTPNFP 3 种策略的组合[18,33,34]。采用 MUMCUT 策略生成的测试用例集中的测试用例数量比 MC/DC 准则要多，但比 MAX-A 和 MAX-B 要少，而且故障检测能力强。与 BASIC 策略类似，MUMCUT 策略主要面向 IDNF 形式的布尔表达式。

(1)MUTP：从每个唯一真值点 UTP_i 中选出测试用例，使其满足对于那些不在乘积项 TM_i 中出现的变量要取到 F 和 T[3,6,18]。

(2)MNFP：从每个邻近假值点 $NFP_{i,j}$ 中选出测试用例，使其满足对于那些不在乘积项 TM_i 中出现的变量要取到 F 和 T[3,6,18]。

(3)CUTPNFP：若测试集 TS 满足 CUTPNFP，则对于任意的 i 和 j，TS 包含了从唯一真值点 UTP_i 选择的测试用例 t_1 和从邻近假值点 $NFP_{i,j}$ 中选择的测试用例 t_2，且 t_1 和 t_2 的差异只在于在 TM_i 中对应的 x_j^i。

(4)MUMCUT：同时满足 MUTP、MNFP 和 CUTPNFP 3 种策略[6,35,36]。

4.4　其它策略

除了上述几类代表性的测试用例生成策略外，Tai 等提出的 BRO 策略[37,38] 和 Foster 提出的敏感数据测试策略[39] 也广受关注。

(1)BRO 测试策略：包含 n 个简单条件组成的条件 C 的条件约束定义为$(D_1, D_2, \cdots, D_n)$，其中 $D_i(0<i\leqslant n)$ 表示条件 C 中第 i 个简单条件的输出约束。如果在条件 C 的一次执行过程中，C 中每个简单条件的输出都满足 D 中对应的约

束，则称 C 的这次执行覆盖了 C 的条件约束 D。对于一个布尔变量 B 来说，B 的输出约束指出，B 的值可以是“真”、“假”或者不确定。该策略可以检测出操作符相关的故障类型，包括“与”和“或”操作符的错误使用、增加或缺少“非”操作符。该策略适用于单一形式的布尔表达式。

(2)敏感数据测试策略：如果逻辑上可能，测试数据应能区分每个布尔表达式成份。具体说来，(I)对每个变量出现的地方，如果以“与”(或“或”)操作符连接时，该变量赋值为 F(或 T)，其它变量赋值为 T(或 F)；(II)对每个表达式而言，以“与”(或“或”)操作符号连接时，其中所有变量赋值 T(或 F)；(III)对每一次赋值而言，与(I)或(II)中变量或表达式的关系为“与”(或者“或”)的其它表达式或变量赋值为 T(或 F)。该策略面向一般形式的布尔表达式，其基本思想与 MC/DC 准则、BASIC 策略、MUTP 策略相似。

4.5 各种基于故障的测试策略的关系与比较

为了进一步比较上述几类具有代表性的面向布尔表达式的测试用例生成策略，本文提出一个比较框架。该框架从适用性、检错能力和测试用例精简 3 个方面对相关的测试用例生成策略进行分析与比较。其中，适用性指测试策略适用的布尔表达式的形式；检错能力指测试策略所能检测出的故障类型；测试用例精简指测试策略产生的测试用例集与穷尽测试产生的测试用例集的比较。一般说来，检错能力愈强，测试用例精简程度愈差；适用性较差，则测试用例精简程度较好。因此，基于故障的测试策略的最终研究目标是在提高适用性的前提下，以较少的测试用例(提高测试用例精简程度)，尽可能多地检测出故障(即提高检错能力)。表 2 总结了上述不同测试策略的适用性。

表 2 不同测试策略的适用性

策略类型	适用性
逻辑覆盖准则	一般形式(GF)
有意义影响策略	最简析取式(IDNF)
MUMCUT 策略	最简析取式(IDNF)
BRO 测试策略	单一表达式(SF)
敏感数据测试策略	一般形式(GF)

表 3 总结了不同测试策略的检错能力。“√”表示能够完全检测出该种故障类型，“P”表示能够部分检测出该种故障类型。其中，CC、DC、C/DC、MC/DC、CUTPNFP、MUMCUT 和 M-CC 测试策略对应于 ENF、TNF、TOF、ORF、LNF、LOF、LIF、LRF 和 STF，这 9 种故障类型的检错能力的相关结果来自于文献[5]中的表 4。有影响意义策略和敏感数据测试策略对应于 LNF、ENF、LRF、ORF、POF 和 PIF，这 6 种故障类型的检错能力的相关结果来自于文献[14]中的表 8—表 12(文献[14]中的 VNF 和 VRF 实际上为 LNF 和 LRF)。MUTP 和 MNFP 对应于 ENF、TNF、TOF、LNF、LIF、LOF，这 6 种故障类型的检错能力的相关结果自于文献[3]。根据图 1 不同故障类型的检测包含关系，我们可以补全表中各种策略对应于各种故障类型的检错能力。例如，如果某个测试策略 S 能够完全检测出故障类型 LRF，那么该 S 一定能够完全检测出故障类型 LNF、TNF 和 ENF；反之，如果某个测试策略 S 能够部分检测出故障类型 ENF，那么该 S 也能够部分完全检测出故障类型 TNF、LNF 和 LRF。为了区分直接与间接检错能力并使比较结果更加简明，我们没有标注出通过推理而得的检错能力(图中未标注的部分均为“P”)。

表 3 各种测试策略的故障检测能力

故障类型 \ 测试策略	逻辑覆盖准则					有影响意义策略						MUMCUT 策略				其他	
	CC	DC	C/DC	MC/DC	M/CC	BASIC	MIN	MANY-A	MANY-B	MAX-A	MAX-B	MUTP	MNFP	CUTPNFP	MUMCUT	BRO	敏感数据测试策略
ENF	P	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√
TNF	P	√	√	√	√							√	√	√	√	√	
TOF	P	P	P	P	√	P	P	P	P	P	P	√		√	√		P
LNF	P	P	P	P	√	P	P	P	P	P	P	√	√	√	√	√	P
LOF	P	P	P	P	√	P	P	P	P	P	P		√	√	√		P
LIF	P	P	P	P	√	P	P	P	P	P	P	√		P	√		P
LRF	P	P	P	P	√	P	P	P	P	P	P			√	√		P
ORF	P	P	P	√	√	P	P	P	P	P	P			√	√	√	P
VNF	P				√												
VRF	P				√												
VOF	P				√												
POF	P				√	P	P	P	√	P	√						P
PIF	P				√	P	P	P	√	P	√						P
STF	P	P	P	P	√									√	√	√	

评估不同测试策略的故障检测能力和测试用例集精简程度的一个可行方法是进行经验研究。Weyuker 等从一个经过修改的战斗机碰撞检测系统(TCAS II)中提取出 20 个布尔表达式[14]，采用变异分析的方法向原始布尔表达式中植入各种类型的故障，然后采用有意义影响策略产生测试用例集来检测这些故障，可以得到不同测试策略的故障检测能力和测试用例集精简程度(通过比较该策略(A)与穷尽测试方法(ET)产生的测试用例集的大小(即|TS(A)|/|TS(ET)|x100%))。后续开发的测试策略都采用类似的方法来评估故障检测能力与测试用例集精简程度。因此，上述 20 个布尔表达式成为评估与比较测试策略的性能的基准。表 4 总结了部分测试策略的测试用例集精简程度。由于相关布尔表达式不满足单一表达式的条件，因此 BRO 策略不适用。通过计算每种测试策略产生的测试用例的数量与采用穷尽测试方法需要的测试用例数量的比值可以得到相应的精简程度。MNFP、CUTPNFP 策略的相关结果采用 BEAT 工具[42]计算得到，由于上述两策略适用与 IDNF 形式的布尔表达式，因此需要将 GF 形式的布尔表达式转换为 IDNF 形式。表中的 3、10—13、16 表达式非常复杂，已有的布尔表达式化简工具无法在短时间内完成转换，其结果用“N/A”表示。

表 4　部分测试策略的测试用例精简程度比较

测试策略 / D01－D20	逻辑覆盖准则(%)					有影响意义策略(%)						MUMCUT 策略(%)			其他(%)
	CC	DC	C/DC*	MC/DC	M/CC	BASIC	MIN	MANY--A	MANY-B	MAX-A	MAX-B	MUTP	MNFP	CUTP-NFP	MUM-CUT	敏感数据测试策略
1	1.56	1.56	2.34	7.81	100	23.4	20.3	26.6	29.7	39.1	45.3	4.69	25.0	20.3	30.1	23.4
2	0.39	0.39	0.59	2.73	100	18.8	11.7	18.8	20.1	22.7	25.6	3.12	19.5	16.4	22.7	18.8
3	0.05	0.05	0.07	0.39	100	5.4	1.3	23.4	23.8	62.0	62.5	1.22	N/A	N/A	6.0	5.4
4	6.25	6.25	9.38	18.6	100	28.1	18.8	46.9	53.1	87.5	96.9	12.5	25.0	25.0	36.5	28.1
5	0.39	0.39	0.59	2.34	100	6.4	3.3	14.5	16.0	69.7	87.3	2.73	4.88	5.47	8.4	6.4
6	0.10	0.10	0.15	0.63	100	2.9	2.2	4.6	4.8	6.3	6.8	0.49	4.88	3.52	4.3	2.9
7	0.20	0.20	0.29	1.37	100	6.4	2.5	13.6	14.3	20.3	22.1	1.56	7.23	6.64	9.9	6.4
8	0.78	0.78	1.17	3.90	100	22.3	12.5	31.3	33.2	43.8	49.2	1.56	12.5	14.1	14.1	22.3
9	1.56	1.56	2.34	6.25	100	12.5	12.5	12.5	15.6	12.5	20.3	1.56	10.9	12.5	12.5	12.5
10	0.02	0.02	0.04	0.21	100	1.1	0.5	2.1	2.2	2.9	3.2	0.15	N/A	N/A	1.0	1.1
11	0.02	0.02	0.04	0.21	100	2.2	0.6	7.7	8.0	22.9	25.3	0.22	N/A	N/A	1.5	2.2
12	0.01	0.01	0.02	0.13	100	0.4	0.1	2.4	2.5	24.7	25.6	N/A	N/A	N/A	0.7	0.4
13	0.05	0.05	0.07	0.37	100	0.5	0.3	3.0	3.3	41.5	41.8	0.29	N/A	N/A	0.9	0.5
14	1.56	1.56	2.34	7.03	100	17.2	8.6	48.4	54.7	71.1	89.1	9.38	12.5	13.3	25.7	17.2
15	0.39	0.39	0.59	2.34	100	8.4	2.7	28.9	31.1	58.0	93.0	4.10	4.49	7.42	11.8	8.4
16	0.05	0.05	0.07	0.39	100	3.1	0.8	15.6	16.1	47.8	48.5	1.12	N/A	N/A	3.9	3.1
17	0.10	0.10	0.15	0.73	100	1.9	0.6	9.5	10.1	46.2	49.6	0.59	1.71	1.51	3.7	1.9
18	0.20	0.20	0.29	1.46	100	4.5	1.3	19.0	20.3	48.2	56.6	1.56	3.13	3.71	7.4	4.5
19	0.78	0.78	1.17	4.30	100	18.8	6.3	46.5	49.6	67.2	89.1	3.14	10.9	8.59	17.3	18.8
20	1.56	1.56	2.34	7.03	100	10.9	9.4	10.9	13.3	17.8	26.6	3.12	15.6	10.9	18.8	10.9
avg	0.07	0.07	0.11	0.48	100	9.8	5.8	19.3	21.1	40.6	48.2	0.76	N/A	N/A	11.9	9.8

综上所述，可以得到如下结论：

(1)逻辑覆盖准则中的几种测试策略(除 M/CC 外)的检错能力比较弱，无法检测出布尔表达式中复杂的故障类型。

(2)有意义影响策略无法兼顾检错能力和测试用例集的精简，在增强检错能力的同时，产生的测试用例集(测试开销)快速增加。具体说来，MIN 和 ONE 策略选择的测试用例少，但检错能力差；MAX-A 和 MAX-B 策略的检错能力强，但选择的测试用例又太多。

(3)MUMCUT 策略是最高效的、基于故障的测试策略，较好地兼顾了故障检测能力和测试用例集的精简。换言之，MUMCUT 策略用最少的测试用例检测出最多的故障。然而，MUMCUT 策略主要面向 IDNF 形式的布尔表达式[43]，针对一般形式的布尔表达式的研究较少。

(4)其它策略中，BRO 测试策略面向单一形式的布尔表达式，可用于检测布尔运算符相关的故障类型；敏感数据测试策略可操作性较差；MUMCUT 策略和有意义影响策略是其新发展。

5　布尔表达式测试尚未解决的几个问题

基于故障的布尔表达式测试研究工作可以划分为两类：(1)从理论分析的角度，逐步探索了不同故障类型之间的包含关系；(2)从经验研究的角度，开发出多种有效的测试策略。无论故障类型之间的包含关系还是各种测试策略，都是基于布尔表达式的某种表示形式和某些故障类型。图 2 归纳了该领域的研究路线图。经过 30 多年的努力，基于故障的布尔表达式测试研究取得了丰硕的成果。该领域仍然存在几个需要进一步研究的问题。

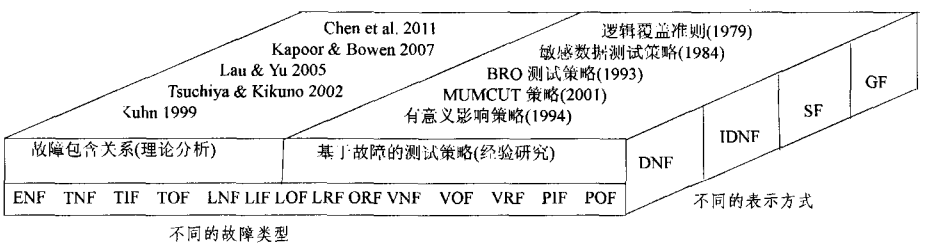

图 2　面向布尔表达式的基于故障的测试技术研究路线图

(1)面向一般形式的布尔表达式的故障类型关系与测试策略：在故障类型的包含关系方面，已有研究成果主要建立在 DNF 基础上(见图 1)。然而，程序员编写程序的条件表达式或设计人员编写规格说明时，所写的布尔表达式通常采取一般形式(GF)，而不是析取范式(DNF)或最简析取范式(IDNF)[3,4]。部分学者试图将故障类型之间的包含关系拓展到一般形式的布尔表达式[20,21]，这方面的扩展仍然有待进一步研究(见图 1(b))。类似地，在基于故障的测试策略研究方面，已有研究成果建立在不同形式的布尔表达式上(见表 2)，逻辑覆盖准则和敏感数据测试策略可以适用于一般形式，然而故障检测能力或测试用例精简程度不足。因此，仍需进一步研究面向一般形式的高效测试策略。

(2)面向测试策略的大规模经验研究：已有研究采用某个软件系统中抽取出来的 20 个布尔表达式作为基准[14]，并采用变异分析技术评估了不同测试策略的故障检测能力和测试用例的精简程度。由于基准中的布尔表达式存在一定的局限性而且数量过少，相应的评估结果的有效性和可靠性不足。如何增加布尔表达式的样本空间以增加测试策略性能评估的可靠性和有效性，有待进一步研究。

(3)基于故障的测试策略的工具支持：尽管存在多种测试

策略，但有些测试策略很难应用于实践（如 BRO 测试策略），有些测试策略（如敏感数据测试策略、逻辑覆盖策略）不存在工具支持。这导致了已有测试策略在实践中应用并不成功。因此，如何开发有效的测试策略及其工具支持，需要进一步探索。

(4)如何与单元测试有效结合：基于故障的布尔表达式测试技术借鉴了逻辑电路的测试思想[25]。与逻辑电路不同的是，程序或规格说明中的布尔表达式更加复杂。例如，布尔变量可能是一个关系表达式、算术表达式、函数调用等。如何支持程序单元测试（特别是循环结构与分支结构的测试）是基于故障的软件测试技术在实践中成功应用有待解决的一个重要问题。

6 我们的研究工作

近年来，我们围绕上述问题进行探索，取得了一些成果。

(1)面向一般形式的布尔表达式的测试策略

MUMCUT 是目前已知的测试策略中最为高效的。由于 MUMCUT 策略主要面向 IDNF 形式的布尔表达式，并且假设被测的 IDNF 中只有单一故障，当 MUMCUT 策略应用于 GF 形式的布尔表达式时，其故障检测能力如何呢？采用了随机生成大规模 GF 形式的布尔表达式，我们采用变异分析的方法植入 8 种常见的故障类型（ENF、LNF、TOF、TNF、ORF、LOF、LIF 和 LRF），将 GF 的布尔表达式转换为 IDNF 形式的布尔表达式后应用 MUMCUT 测试策略产生测试用例。实验果表明，MUMCUT 策略的故障检测能力高于 98.2%[43]。进一步分析无法检测（未杀死）的故障类型（Mutant）后，发现了 5 类 MUMCUT 策略不能检测的故障模式。

针对几类不能检测出的故障模式的特点，进一步对 MUMCUT 策略产生测试用例集以及 MUMCUT 策略的失效原因做了分析，对 MUMCUT 策略进行了扩展，从理论上验证了扩展后的 MUMCUT 可以检测出其中一些故障模式[4]。

MUMCUT 策略可以确保检测出 IDNF 形式的布尔表达式的 8 种故障类型[44]，前提条件是植入的故障必须是单一的（即布尔表达式中只能存在一个故障）。MUMCUT 无法直接应用于 GF 形式的布尔表达式，那么，布尔表达式由 GF 形式转换为 IDNF 形式时故障类型是如何传播的呢？是否存在一些 GF 形式中故障类型在转换成 IDNF 时消失的情形[2]？通过经验研究发现，一个 GF 形式的单一故障类型可以导致 IDNF 形式中的若干个故障类型同时出现[3]。这意味着，当 MUMCUT 应用于一般形式的布尔表达式时，其故障检测能力无法确定。我们采用了随机生成大规模 GF 形式的布尔表达式，并采用变异分析的方法评估了 MUMCUT 的性能。与先前的经验研究相比，增加了两类与扩号相关的故障类型（即 POF 和 PIF）；考虑到可能基于规格说明或程序产生测试用例，采用了两种方式评估了 MUMCUT 的性能。经验研究结果表明：尽管 MUMCUT 不是面向 GF 形式的布尔表达式设计的，但其故障检测能力仍然处于 91.6%与 100%之间。

(2)基于随机生成的大规模布尔表达式的经验研究

以随机生成的布尔表达式为基准，评估了 MUMCUT 应用于 GF 的布尔表达式的性能，分析了 GF 形式转换为 IDNF 形式的故障传播规律。开发了相关的工具集，包括基于有限自动机的参数化 GF 的布尔表达式的生成工具[45]、变异分析辅助工具、GF 到 IDNF 的转换工具、测试分析与判定工具。这些工具与 MUMCUT 策略支持工具 BEAT[42]一起构成了非常有效的实验环境，可以快速产生大量 GF 形式的布尔表达式，植入各种故障类型实现 GF 到 IDNF 的自动转换、测试用例的自动生成、测试的执行与判定。以 20 个布尔表达式为基准对 MUMCUT 进行性能评估时，得到的故障检测率为 99.8%[44]；以随机生成的 1000 个布尔表达式为基准的评估结果为 98.2%[43]。通过增加布尔表达式的数量以及引入随机性，可以解决以 20 个布尔表达式为基准的性能评估中评估结果不可靠的问题。

(3)测试策略支持工具开发方面

扩展了 MUMCUT 测试策略，并在 BEAT 系统的基础上进行了实现，同时还集成了其它测试策略。基于所开发的测试策略支持工具，不仅可以进行大规模的经验研究（以随机生成的布尔表达式为基准，对扩展的 MUMCUT 策略和有意义影响策略进行性能评估），而且可以在单元测试阶段辅助生成测试用例。

结束语 本文系统地归纳与总结基于故障的布尔表达式测试技术。在回顾基于故障的软件测试的基本概念和原理的基础上，对布尔表达式的各种故障类型进行了归纳和定义，统一了不同文献对同一故障类型的不同称呼，纠正了部分文献中对某些故障类型不恰当的定义方式。系统总结了不同故障类型之间的检测包含关系，得到较全面的故障检测包含关系图，这有助于解释经验研究的结果，并为布尔表达式的测试提供了理论指导。在评述与回顾各种测试策略的基础上，提出了一种比较框架来比较各种测试策略的适用性、故障检测能力和测试用例精简程度。总结了该领域的研究线路图，指出了在面向一般布尔表达式的故障检测包含关系及测试策略等方面存在且尚需研究的一些问题，简要介绍了课题组近年来针对这些问题开展的一些研究工作。

参 考 文 献

[1] Morell L J. A Theory of Fault-Based Testing [J]. IEEE Transactions on Software Engineering, 1990, 16(8): 844- 857

[2] Okun V, Black P E, Yesha Y. Comparison of Fault Classes in Specification-Based Testing [J]. Information and Software Technology, Elsevier, 2004, 46(8): 525-533

[3] Chen T Y, Lau M F, Sim K Y, et al. On detecting faults for Boolean expressions [J]. Software Quality Journal, 2009, 17(3): 245-261

[4] Sun C A, Dong Y W, Lai R, et al. Analyzing and Extending MUMCUT for Fault-based Testing of General Boolean Expressions [C]// Proceedings of the Sixth IEEE International Conference on Computer and Information Technology (CIT'06). IEEE Computer Society, 2006: 184-190

[5] Yu Y T, Lau M F. A comparison of MC/DC, MUMCUT and several other coverage criteria for logical decisions [J]. Journal of Systems and Software, 2006, 79(6): 577-590

[6] Yu Y T, Lau M F, Chen T Y. Automatic generation of test cases from Boolean specifications using the MUMCUT strategy [J]. Journal of Systems and Software, 2006, 79(6): 820-840

[7] Myers G J. The Art of Software Testing [M]. John Wiley&Sons, Inc., New York, 1979

[8] DeMillo R A, Lipton R J, Sayward F G. Hints on test data selection: Help for the practicing programmer [J]. Computer, 1978, 11(4): 34-41

[9] Offutt A J, Untch R H. Mutation 2000: Uniting the orthogonal [C]// Proceedings of Mutation Testing for the New Century. 2001: 34-44

[10] 胡璇,刘斌,陆民燕. 软件代码缺陷分类及其应用[J]. 计算机工程,2009,35(2):30-33

[11] 肖庆,宫云战,杨朝红,等. 一种路径敏感的静态缺陷检测方法[J]. 软件学报,2010,21(2):209-217

[12] Chen T Y, Leung H, Mak I K. Adaptive Random Testing [C]// Proceedings of the 9th Asian Computing Science Conference, LNCS 3321. Springer, 2004: 320-329

[13] Cai K Y, Jing T, Bai C G. Partition Testing with Dynamic Partitioning [C] // Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC' 05). IEEE Computer Society, 2005: 113-116

[14] Weyuker E, Goradia T, Aingh S. Automatically generating test data from a Boolean specification [J]. IEEE Transactions on Software Engineering, 1994, 20(5): 353-363

[15] Lau M F, Yu Y T. An Extended Fault Class Hierarchy for Specification-Based Testing [J]. ACM Transactions on Software Engineering and Methodology, 2005, 14(3): 1-27

[16] Lau M F, Liu Y, Yu Y T. On detection conditions of double faults related to terms in Boolean expressions [C]// Proceedings of the 30th Annual International Computer Software and Application Conference (COMPSAC 2006). Chicago: IEEE Computer Society, 2006: 403-410

[17] Lau M F, Liu Y, Yu Y T. On detection double faults with term and literal in Boolean Expressions [C]// Proceedings of the 7th International Conference on Quality Software (QSIC 2007). Hong Kong: IEEE Computer Society, 2007: 117-126

[18] Chen T Y, Lau M F, Yu Y T. MUMCUT: A Fault-Based Strategy for Testing Boolean Specifications [C]// Proceedings of Sixth Asia-Pacific Software Engineering Conference (APSEC' 99). Asia: IEEE Computer Society, 1999: 606-613

[19] Kobayashi N, Tsuchiya T, Kikuno T. Non-specification-based approaches to logic testing for software [J]. Information and Software Technology, 2002, 44(2): 113-121

[20] Chen Z Y, Chen T Y, Xu B W. A Revisit of Fault Class Hierarchies in General Boolean Specifications [J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(2): 1-12

[21] Kapoor K, Bowen J. Test conditions for fault classes in Boolean specifications [J]. ACM Transactions on Software Engineering and Methodology, 2007, 16(3): 1-12

[22] Tsuchiya T, Kikuno T. On Fault Classes and Error Detection Capability of Specification-Based Testing [J]. ACM Transactions on Software Engineering and Methodology, 2002, 11(1): 58-62

[23] Wang C Q, Wang C H. A Method for Logic Circuit Test Generation Based on Boolean Partial Derivative and BDD [C]// Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering. IEEE Computer Society, 2009: 499-504

[24] Lau M F, Liu Y, Yu Y T. On the detection conditions of double faults related to literals in Boolean expressions [C]// Proceedings of the 12th International Conference on Reliable Software Technologies. Berlin: Springer, 2007: 55-68

[25] Kuhn D R. Fault classes and error detection capability of specification-based testing [J]. ACM Transactions on Software Engineering and Methodology, 1999, 8(4): 411-424

[26] Chen T Y, Yu Y T. On the Relationship between Partition and Random Testing [J]. IEEE Transactions on Software Engineering, 1994, 20(12): 977-980

[27] Lau M F, Yu Y T. On the Relationships of Faults for Boolean Specification Based Testing [C]// Proceedings of 13th Australian Software Engineering Conference (ASWEC' 01). Australian: IEEE Computer Society, 2001: 21-28

[28] Kaminski G, Ammann P. Using a Fault Hierarchy to Improve the Efficiency of DNF Logic Mutation Testing [C]// Proceedings of 2009 International Conference on Software Testing Verification and Validation. Denver: IEEE Computer Society, 2009: 386-395

[29] Chilenski J J, Miller S P. Applicability of modified condition decision coverage to software testing [J]. Software Engineering Journal, 1994, 9(5): 193-200

[30] Dupuy A, Leveson N. An empirical evaluation of the MC/DC coverage criterion on the HETE-2 satellite software [C]// Digital Aviation Systems Conference. Phildelphia: DASC, 2000: 438-444

[31] Jones J A, Harrold M J. Test-suite reduction and prioritization for modified condition/decision coverage [J]. IEEE Transactions on Software Engineering, 2003, 29(3): 195-209

[32] Chen T Y, Lau M F. Two Test Data Selection Strategies towards Testing of Boolean Specifications [C]// Proceedings of the 21th Annual International Computer Software and Applications Conference (COMPSAC' 97). IEEE Computer Society, 1997: 608-611

[33] Chen T Y, Lau M F. Prioritization of Test Cases in MUMCUT Test Sets: An Empirical Study[M]. Berlin Heidelberg: Springer, 2002: 245-256

[34] Kaminski G, Ammann P. Using Logic Criterion Feasibility to Reduce Test Set Size While Guaranteeing Fault Detection [C]// Proceedings of International Conference on Software Testing Verification and Validation. Denver: IEEE Computer Society, 2009: 356-565

[35] Yu Y T, Lau M F. Comparing Several Coverage Criteria for Detecting Faults in Logical Decisions [C]// Proceedings of the 4th International Conference on Quality Software (QSIC'04). Hong Kong: IEEE Computer Society, 2004: 14-21

[36] Yu Y T, Lau M F, Chen T Y. Using the Incremental Approach to Generate Test Sets: A Case Study [C]// Proceedings of the 3rd International Conference On Quality Software (QSIC'03). Hong Kong: IEEE Computer Society, 2003: 263-271

[37] Tai K C. Predicate-Based Test Generation for Computer Programs [C] // Proceedings of the International Conference on Software Engineering (ICSE 1993). IEEE Computer Society, 1993: 267-276

[38] Tai K C, Vouk M A, Paradkar A M, et al. Evaluation of a predicate-based software testing strategy [J]. IBM systems journal, 1994, 33(3): 445-457

(下转第 40 页)

境、同一个实验平台进行这两个实验，两个 Job 的 Reduce 数量都设置为 4。默认 Partition 方案将不做任务处理而直接进行该实验。本方案第一次运行不带任何权重信息，实际上就是 TeraSort 所采用的方案。所以我们实际上是先随意地运行一些其他数据生成一个初始的权重信息，再利用给定的实验数据进行加权方案的测试。实验结果如表 2 和表 3 所列。

表 2　默认 hadoop 实验结果数据(2012. 6. 18)

Task	开始时间	结束时间	消耗时间
Task021_00	12:48:45	13:48:37	59mins,52sec
Task021_01	13:01:45	14:07:34	65mins,49sec
Task021_02	13:07:45	14:21:37	73min,52sec
Task021_03	12:58:45	15:15:36	196min,50sec
总时间	12:48:35	15:15:51	207mins,16sec

表 3　本方案实验结果数据(2012. 6. 18)

Task	开始时间	结束时间	消耗时间
task_r_022_00	15:50:07	16:03:30	73mins,23sec
task_r_022_01	15:55:07	16:15:06	89mins,59sec
task_r_022_02	15:53:07	16:20:15	87mins,7sec
task_r_022_03	15:59:07	16:28:42	89mins34sec
总时间	15:49:57	16:29:00	99mins,3sec

从图 2 可以看出，优化前后效率提升接近一倍。所以对于同一个 Job 中存在处理复杂度不一样的 Key，采用文中给出的 Partition 方案，效率会有明显的提升，处理的复杂度差别越大，则效果就越明显。通过对上表两种实验场景各个 Reducer 运行的时间求标准差可明显看出，采用本方案以后负载比默认方案要均衡，整个 Job 效率也因为 Reduce 阶段的负载均衡而得到明显提升。

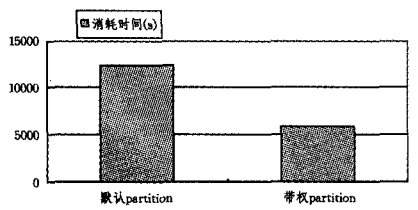

图 2　实验消耗时间对比图

结束语　在实际的业务中，周期性的海量数据计算是非常普遍的，而且周期性的数据在一定时间内其数据分布具有相似性。本文设计的策略充分利用周期性任务的这个特点，通过历史的统计运行信息来预测将来数据的分布，后续可以在历史数据基础上设计出一些更为精准的预测算法来提升负载的均衡度。本策略具有可调性、记忆性以及沉淀性，运行的时间越长，其统计出来的每种类型的数据的权重信息越丰富、越准确。

参 考 文 献

[1] White T. Hadoop: The definitive guide [OL]. http://books.google.com,2010

[2] Borthakur D. TheHadoop Distributed File System: Architecture and Design[OL]. http://cloudcomputing.googlecode.com,2007

[3] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[C]//OSDI'04, Proceedings of the 6th Coference on Symposium Opearting Systems Design & Implementation. Sep. 2004

[4] Lammel M R. Google's MapReduce programming model—Revisited[J]. Data Programmability Team,2007,68(3):208-237

[5] Armbrust M, Fox A, Griffith R. Above the Clouds: A Berkeley View of Cloud Computing[M]. ACM,2010

[6] Seo S, et al. HPMR: Prefetching and Pre-shuffling SharedMapReduce Computation Environment[C]//the Proceedings of 11th IEEEInternational Conference on Cluster Computing. Sep. 2009

[7] Jiang D,Ooi B C,Shi L,et al. The Performance of MapReduce: An Indepth Study[C]//Int'l Conference on Very Large Data Bases (VLDB). 2010

[8] Dittrich J,Jindal A. Schad Hadoop++:Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)[J]. VLDB 2010/PVLDB,2010,34(1/2):515-529

[9] Liu Xu-hui,Han Ji-zhong. Implementing WebGIS on Hadoop:A case study of improving small file I/O performance on HDFS[C]//Cluster Computing and Workshops, 2009. IEEE International Conference on. 2009:1-8

[10] Lee K-H,Lee Y-J,Choi H,et al. Parallel data processing with MapReduce:a survey[J]. ACM SIGMOD Record,2011,40(4):11-20

[11] O'Malley O. TeraByte Sort on Apache Hadoop[Z]. Yahoo!, May 2008

(上接第 23 页)

[39] Foster K A. Sensitive test data for logic expressions [J]. ACM SIGSOFT Software Engineering Notes,1984,9(2):120-125

[40] Yu L, Tsai W T, Zhao W, et al. Towards Selecting Test Data Using Topological Structure of Boolean Expression [C]//Proceedings of 9th International Conference on Quality Software (QSIC 2009). IEEE Computer Society,2009:31-40

[41] Chen T Y, Lau M F. An Empirical Study on the Effectiveness of the Greedy MUTP Strategy [C]//Proceedings of International Conference on Software Engineering: Education and Practice (SEEP'98). California: IEEE Computer Society,1998:338-344

[42] Chen T Y,Grant D D,Lau M F,et al. BEAT:Boolean Expression Fault-based Test Case Generator [C]//Proceedings of International Conference on Information Technology: Research and Education (ITRE 2003). California: IEEE Computer Society, 2003:64-69

[43] Sun C A,Sim K Y,Tse T H,et al. An Empirical Evaluation and Analysis of the Fault-Detection Capability of MUMCUT for General Boolean Expressions [C]//Proceedings of International Computer Symposium (ICS 2004). Taipei,2004:926-932

[44] Chen T Y, Lau M F. Test case selection strategies based on Boolean specifications [J]. Software Testing, Verification and Reliability,2001,11(3):165-180

[45] Sun C A,Sim K Y. An FSM-based Parameterized Generator of General Boolean Expressions [C]//Proceedings of International Computer Engineering Conference (ICENCO). 2004:119-126