从曼德尔虫引起的软件故障中恢复过来

Michael Grottke[1] , *会员，IEEE*，Dong Seong Kim[2] , *会员，IEEE*，Rajesh Mansharamani[3] 。
Manoj Nambiar[4] , *IEEE高级会员*, Roberto Natella[5] , *IEEE会员*, Kishor S. Trivedi[6] , *IEEE会员*[1]
德国Friedrich-Alexander-Universität Erlangen-
Nürnberg的统计和计量经济学系，电子邮件。Michael.Grottke@fau.de
[2]*新西兰坎特伯雷大学计算机科学和软件工程系 电子邮件: dongseong.kim@canterbury.ac.nz*
[3]*性能工程的自由顾问，印度 电子邮件:*
rajesh.mansharamani@gmail.com
[4]*印度塔塔咨询服务公司，电子*
邮件：m.nambiar@tcs.com
[5]*那不勒斯费德里科二世大学，意大*
利
电子邮件：roberto.natella@unina.it
[6]*美国杜克大学电子和计算机工程系杜克高可用性保证实验室，电子邮件：ktrivedi@duke.edu*

***摘要*--**
尽管在软件测试方面花费了大量精力，但软件故障仍然是任务和企业关键背景下的一个主要问题。事实上，虽然软件测试对容易重现的错误(*Bohrbugs*)很有效，但它对处理导致难以重现的故障(*Mandelbugs*)的错误却相当不适合。在积极的一面，曼德尔虫的难以捉摸的性质为故障恢复提供了机会，本文对此进行了研究。基于在生产中运行的11个信息技术（IT）系统中的Mandelbugs的真实案例，本文提出了一个描述IT系统中恢复过程的模型。然后，它提出了平均恢复时间和软件（非）可用性的封闭式表达和数字分析。这种分析使设计者能够比较恢复策略，并确定对曼德尔虫引起的故障的恢复效率有很大影响的参数。

Index Terms -Mandelbugs, mean time to recovery, semi-Markov model, sensitivity analysis, software quality.

简称和缩略语

ATM自动取款机
DCED数据通信设备
IT信息技术
JVMJava虚拟机
MTTFM平均故障时间
MTTR 平均恢复时间
ODCOrthogonal Defect Classification
OSOperatingsystem
SSUAS稳不可用
TCP传输控制协议

注释

$E[D_{ad}]$ 自动检测故障的平均时间
$E[D_{md}]$ 人工检测故障的平均时间
$E[D_{dg}]$ 故障诊断的平均时间
$E[D_{rs}]$ 软件组件或服务重新启动的平均时间

$E[D_{rb}]$ 硬件服务器或虚拟机重新启动的 平均时间 $E[D_{rc}]$ 软件组件或服务重新配置的 平均时间 $E[D_{hf}]$ 进行热修复的 平均时间

$E[D_{bf}]$ 进行错误修复的 平均时间

p_{ad} 自动检测故障的 概率

p_i 故障是由错误类型*i*引起的 概率

p_{4hf} 正确热修复的 概率

d_{ij} 考虑到错误类型*i*，诊断出错误类型*j*的 概率

I. 简介

可靠性是信息技术（IT）软件系统的一个关键质量属性；在对任务和企业关键应用程序进行功能、集成和性能测试时，需要花费数周甚至数月时间[1]。尽管做出了这些努力，在发布给用户的软件中仍然存在一些错误，并导致IT软件的故障。这种故障影响了最终用户和操作人员，导致了长时间的中断，增加了维护成本，以及人们对质量的低认知度：最近*HealthCare.gov*门户网站的问题[2]，以及一些关键任务系统的事故[3]，都是IT软件故障的严重后果的例子。

由于这些原因，最近的研究[4]、[5]、[6]、[7]调查了软件故障的性质，以及针对这些故障的对策。作者发现，错误可以被分为两类。玻尔布格和曼德尔布格。玻尔博格是一种容易隔离的故障，其表现形式在一组明确定义的条件下是一致的。一旦玻尔博格造成了故障，在软件代码中追踪它并将其删除是一项相对简单的工作。相比之下，曼德尔虫的行为可能显得不确定，因为同一组输入数据在某些时候似乎会使它造成故障，但在其他时候却不会。因此，由曼德尔虫引起的故障通常很难重现。Mandelbug的这些特征可以追溯到它的激活或错误传播的复杂性，或者两者都是。一个可能的原因是故障激活和故障发生之间的时间滞后。另一个可能的原因是间接因素的影响，如曼德尔布格所在的软件应用程序与其系统内部环境（硬件、操作系统或其他应用程序）的相互作用，输入和操作的时间（相对于彼此），以及输入和操作的顺序。竞赛条件[8]是一个典型的例子，它是由曼德尔虫引起的问题。由于多核服务器和分布式架构的出现，目前正在高速发展的并发程序，非常容易包含曼德尔漏洞[9], [10]。

虽然玻尔布格可以通过测试和调试来有效抵制，但由于曼德尔布格难以捉摸的特性，要找到并清除它是一项艰巨的任务。当然，一旦理解了曼德尔漏洞，它就可以在代码中被修复。然而，对于一个由曼德尔布格引起的故障，通常很难确定是什么引起的故障。例如，要处理一个竞赛条件，开发者应该首先确定导致竞赛的并发进程的排序，然后通过使用适当的排序或锁定技术来修复它。

在积极的一面，曼德尔漏洞难以捉摸的性质为容错策略提供了机会：由于触发曼德尔漏洞的条件（通常与事件时间或环境有关）是不稳定的，这些漏洞通常在重新启动失败的系统或组件，以及第二次重试失败的操作（例如，当环境略有不同时）[6], [11], [12], [13], [14]之后不再表现出来。例如，在出现竞赛条件的情况下，重试一个失败的操作将大概率地成功，因为在第二次执行时，操作系统（OS）的进程调度可能会有所不同。其他类型的故障可以通过重启故障机器来容忍，重启可以清除被曼德尔虫破坏的错误数据，或者通过重新配置系统的不同参数设置，如超时和缓冲区的大小。IT系统设计者从经验中知道，他们可以利用这种*环境多样性*来快速恢复软件故障，从而以经济有效的方式提高系统的可用性，而不需要求助于昂贵的设计多样性，或修复导致故障的曼德尔虫。相比之下，重新启动、重启或重新配置一个系统将无法成功对付玻尔虫，因为这些虫子在第二次向系统提交相同的输入数据时将再次产生故障[6]。

本文提出了一个分析模型，用于评估和比较IT系统的恢复策略，考虑到了曼德尔虫和玻尔虫的不同性质。建立和分析这种模型的兴趣有三个方面。首先，我们通过组成模型就可以获得对IT系统行为方式的系统理解。第二，我们可以从模型中推导出感兴趣的措施，如平均恢复时间和稳定状态下的软件（非）可用性。第三，也是最重要的，是

该模型使我们能够(i)比较不同的恢复行动以及基于这些行动的恢复策略的相对收益；(ii)找到恢复的瓶颈，以便能够通过数字和参数敏感性分析来确定提高平均恢复时间的最关键系统参数。我们特意设计了一个容易被IT从业者理解、配置和使用的模型；并且我们努力对IT系统和故障恢复策略做出现实的假设。本文的主要贡献包括在真实的IT系统中发现的曼德尔漏洞的具体例子，由这些漏洞引起的故障恢复的详细模型，以及从这些漏洞中恢复的平均时间的闭合式解决方案，以及由此产生的软件（非）可用性。

本文的组织结构如下。在第二节中，我们讨论了在错误分析和分析模型领域的相关工作。第三节介绍了在11个IT系统运行过程中发现的几个曼德尔虫的例子，以及对曼德尔虫的分类和采取的恢复行动。在第四节中，我们根据上一节中获得的洞察力，建立了一个分析模型，用于从曼德尔布格引起的故障中恢复。在第五节中，我们得出这个模型所暗示的平均恢复时间，并进行了敏感性分析。最后，第六节是本文的结论。

II. 相关的工作

软件错误分析一直是各种IT项目实施中的一个共同主题。它的主要目标是通过避免引入错误，以及在开发的早期阶段改善错误的消除，来减少软件开发项目中错误的影响和成本。这些分析产生了从测试用例设计到根本原因分析、检查表以及编程和调试实践的准则和实践。在过去提出的许多关于错误分析的研究中，以下研究和它们的应用值得注意。

- Basili和Perricone[15]将软件故障类型分为*初始化*、*控制结构*、*接口*、*数据*和*计算*故障；并进一步将这些故障分为*遗漏*和*委托*故障；获得了它们在软件开发过程中的频率；并指出了它们与一些因素的关系，如代码复杂性（如循环复杂性）、代码重用和开发者的经验。
- Perry和Evangelist[16]将他们的分析集中在*接口故障*（即与模块的本地环境之外的结构相关的故障，但模块使用了这些结构），因为它们代表了大型系统中故障的重要部分。他们对实时系统的接口故障进行了详细的分类（例如，对模块功能的分歧、滥用接口、违反数据约束），并提供了减轻每一类接口故障的建议，包括改进对没有经验的开发人员的培训，以及设计和验证活动。
- Chillarege*等人*[17]提出的正交缺陷分类（ODC）是一种分类方案，用于从*缺陷属性*的分布中获得对开发过程的了解。属性包括缺陷类型，它反映了程序员所做的修正。缺陷类型的定义是基于缺陷和它所处的开发阶段之间的因果关系，能够识别出需要更多关注的过程阶段：例如，过多的*功能*缺陷（即系统功能缺失或不正确）表明高层设计阶段应该得到改进。
- 一些来自软件从业者的书籍和技术论文[18], [19], [20], [21]调查了发生在编码阶段的常见错误（例如，由于对编程语言结构的误解而产生的语法和语义错误），并提出了预防这些错误的编程实践。

在本文中，我们利用Grottke和Trivedi[5], [6], [7]中的分类方法，将bug分为Mandelbugs（以及它们的反义词Bohrbugs）。这种分类与虫子的*不可复制性有关*，因为由它们引起的故障具有短暂性。在他的开创性工作中，Gray[22]观察到，这种曾经归因于硬件故障的行为[23]，[24]，在软件故障中也很普遍。他甚至假设，在成熟的高可用性系统中发生的大多数软件故障都是*短暂的*，因为严格的测试活动和多年的生产使用消除了大多数错误，除了那些与复杂条件（如竞赛条件、过载和设备故障）有关的错误，这些错误在（明显的）随机时间表现出来。因此，这样的bug很难重现，也很难被开发人员调试[12]。这种瞬时行为是由错误和程序环境之间的微妙关系造成的，例如事件的时间，以及操作系统资源的状态。

为了说明这些方面，Grottke和Trivedi[5], [6], [7]将Mandelbug定义为这样的错误：(i)由该错误产生的错误（即运行系统的不正确的内部状态）的传播在转化为故障之前涉及几个错误状态或几个子系统或两者（即。(ii)
故障的发生受到间接的、难以控制的因素的影响，如程序的环境（如操作系统、其他同时执行的程序或硬件）、输入和操作的时间（相对于彼此）以及输入和操作的顺序。如果这两个条件中的一个成立，那么由错误引起的故障通常是很难重现的。这个分类方案已经在一些研究中被采用和验证（在下面讨论），并且随着时间的推移，通过考虑曼德尔漏洞的子类型得到了完善[25]。

与上面提到的错误分类研究不同，Mandelbugs-
Bohrbugs分类对容错架构的设计有重要影响，这一点在介绍中已经讨论过了。因此，最近的研究集中在了解不同类型的Mandelbugs，提出和评估技术和策略来容忍

导致的故障，以及调查软件系统的类型和领域与曼德尔虫的发生之间的关系。

- Huang等人[26]观察到一种特殊类型的Mandelbug，影响*连续运行的*应用程序，被Grottke等人[7]称为*老化相关（Mandel）bug*，可以通过在故障发生前应用的主动方法，即*软件年轻化来缓解*。例如，导致内存泄漏和膨胀的bug就是这种情况，它导致了内存的逐渐耗尽，并导致应用程序的突然停止。在这种情况下，在一个方便的时间（例如，在预定的维护期间）执行应用程序的重启，可以释放泄漏的内存，并有可能避免故障的发生。Alonso等人[33]提出了一个软件年轻化策略的实验比较，Cotroneo等人[27]对软件老化和年轻化研究进行了全面调查。在[28]和[29]中分析了在开放源码软件中发现的与老化有关的错误。

- 基于Grottke和Trivedi[6]的定义，随后的研究通过对现场和发布前测试中发现的曼德尔虫进行分类和分析，分析了曼德尔虫在真实系统中的发生。Grottke等人[7]对NASA任务中的软件异常进行了分析，发现虽然玻尔布格占故障的大多数，但曼德尔布格也占了相当大的份额，在20%到40%之间。此外，这些作者表明，不同任务的玻尔布格（以及因此而产生的曼德尔布格）的比例似乎稳定在几乎相同的数值上。他们还发现，即使在长期运行的关键任务软件中，与老化有关的错误也占不可忽视的份额（4.4%）。最近的研究分析了不同类型和不同领域的系统中的曼德尔漏洞的比例，包括一个大型的分布式防御系统[30]；Linux、MySQL、Apache HTTPD和Apache
  AXIS开源项目[25]；和两个企业产品[31]。即使这些研究对Mandelbugs和Bohrbugs的总体趋势做出了类似的结论，他们也指出，Mandelbugs的比例受领域和软件类型的影响，其中嵌入式和操作系统软件往往表现出较高的Mandelbugs比例，而中间件和企业软件的这一比例则较低。软件生命周期的阶段对Mandelbugs的比例也有影响，因为处于发布前测试阶段的软件往往比处于发布后阶段的软件表现出更多的Bohrbugs比例。最后，经验证据[7]，[25]显示，根据严重程度等级，用户和开发人员认为玻尔布格和曼德尔布格的重要性相似，而Chillarege[32]指出，曼德尔布格比玻尔布格影响不同的产品领域；特别是，曼德尔布格显著影响感知的可用性（即服务的连续性），但它们似乎在系统的功能要求方面没有影响。

- 在已经提出的用于容忍由Mandelbugs引起的故障的几种容错技术中，我们提到了*microreboot*[11]，它通过重新启动应用程序的选定子组件来对故障做出反应，从而避免了整个应用程序的重新启动，并减少了恢复时间。NT-
  SwiFT[13]，它为开发容错集群系统提供了一些工具，包括检测进程和节点故障的工具，以及对故障进程进行检查和回滚的工具；以及*Rx*[14]，它在*修改过的*环境下重新执行故障程序（如g.,通过改变异步事件的时间、线程调度和堆内存中的数据结构分配）来增加屏蔽曼德尔漏洞的可能性。由于在不同的环境中重新执行，我们把这些方法称为*环境多样性*，与基于设计多样性的经典软件容错技术形成对比。

这些研究指出了曼德尔布格在关键任务系统中的重要性，强调了大量的曼德尔布格会影响这些系统，通过采用针对曼德尔布格引起的故障的故障恢复策略，可以大大改善可用性和容错性。

在[26]、[34]、[35]、[36]、[37]、[38]和[39]中已经发表了许多关于调整软件复兴以抵消老化相关错误影响的模型。但只有少数研究是关于更普遍的曼德尔虫类的建模，以及相关恢复策略的分析。文献中提出了针对特定平台实例的系统可用性模型。Garg等人[40]提出了由SwiFT和DOORS容错技术提供的冷复制和暖复制方案的分析模型，并得出了在瞬时软件故障情况下的可用性、吞吐量和请求丢失概率的闭式表达式。Trivedi等人[41]分析了IBM
Websphere的会话初始协议（SIP）；该论文考虑了非老化相关的Mandelbugs，以及由此类bug引起的故障的恢复。Grottke和Trivedi[42]对可以使用各种技术恢复的系统进行了详细的基于模型的研究。

本文提出的恢复模型受到[41]和[42]的启发。虽然[42]中的模型具有非常普遍的拓扑结构，但本文提出的模型是针对IT系统中曼德尔虫恢复的具体情况而定制的。与我们的初步工作[43]相比，我们提供了一个增强版的恢复模型。新的模型通过调整模型中的故障检测、故障诊断和恢复行动，使用了定制的恢复策略，从而使其有助于比较替代的恢复策略。此外，新模型更适合包括来自曼德尔布格的现场故障数据的信息（在类型和频率方面），从而使其更容易分析现实情况下的恢复策略。我们

使用该模型来比较在曼德尔虫存在下的三种恢复策略，并提供关于恢复策略的有效性和有效恢复的最关键因素的见解。分析是通过考虑[25]中描述的关于曼德尔布格的现场故障数据来进行的，从中我们得出了三种用于分析恢复策略的现实情况。本文没有考虑将恢复策略与软件年轻化相结合，而是留待未来的研究。

III. 真实IT系统中的曼德尔布格分析

在这项研究中，我们考虑了发生在11个IT系统中的由于曼德尔漏洞造成的故障。我们分析了在测试过程中没有发现的曼德尔漏洞，这些漏洞甚至在IT系统发布几个月后才影响到生产。本文的两位作者参与了这些IT系统的审查和维护，他们有第一手的资料可以接触到设计文件、测试套件以及关于IT系统故障和错误的数据。在IT人员的支持下，我们分析了他们采用的故障检测、恢复和修复流程。

这11个IT系统跨越多个领域，包括一个证券交易所系统、一个外汇交易系统、一个政府的税务信息系统、一个用于前台订单路由的产品，以及一个IT机构的在线测验应用。其余项目涉及一家大型银行、一家清算公司、两家经纪公司、一家大型药店和一家大型电信供应商。除了测验应用，这些IT系统都是他们公司核心业务的关键项目。所有的IT系统都有多达数千的并发用户，并受到高峰负荷的影响。几乎在所有的情况下，都不允许有计划外的停机，因为这将导致大量的业务损失。

我们的分析集中在IT故障的技术方面，目的是确定根本原因和IT操作人员采取的实际恢复行动。在下面的小节中，我们首先提供曼德尔布格的例子，以及各自采取的恢复行动。然后，我们提供了我们的分析所发现的曼德尔布格的分类，以及为处理这些布格所采用的恢复方法。

A. 曼德尔布格的例子

以下六个例子是我们分析中发现的曼德尔布格，是IT运营人员所执行的故障和恢复行动的代表。

- **Bug #1：证券交易系统中的请求阐述不同步。**
  - *系统描述。*该系统管理来自交易者的输入、修改和取消订单的请求。请求由两个阶段的管道来处理。首先，请求被排队验证，如果请求是有效的（例如，请求的订单是有效的），则向用户发送确认。其次，有效的请求被排队处理。
  - *错误描述和表现形式。*该系统受到一个时间问题的影响，涉及到订单输入的请求和同一订单的修改请求。故障的根本原因是，新的订单只在流水线的末端被插入订单簿，而不是在验证后才插入。因此，如果订单输入请求已被验证，但它仍在处理队列中，那么对该订单的修改请求就会被错误地拒绝，即使请求的订单已被验证。这种故障是零星发生的，例如在罕见的超载条件下，对交易者来说，它表现为修改请求的失败。
  - *恢复行动。*故障可以通过几秒钟后重新发出修改请求来恢复。

- **Bug #2：大型物流公司数据库的碎片化。**
  - *系统描述。*该系统有一个用于在线交易处理的主数据库，和一个用于保存旧数据的档案数据库。为了提高性能，每晚的批处理工作将数据从主数据库转移到存档数据库。为了避免不可用，通过在主数据库上使用删除查询，在不关闭数据库的情况下将数据归档。
  - *错误描述和表现。*在运行时执行的删除查询导致主数据库的碎片化，从而影响其性能。在某些情况下，归档工作没有按时完成，并且干扰了早上执行的工作，这些工作更新了货物的状态。
  - *恢复行动。*在每天对数据库索引进行凝聚整理后，系统从性能下降中恢复过来。

- **Bug #3：大型电信系统中的前端屏幕无反应。**
  - *系统描述。*用户通过前端屏幕访问该系统。当用户启动一个新的会话时，在服务器上创建一个小的临时文件。
  - *错误描述和表现。*临时文件在会话结束时从未被清理过，导致服务器的文件系统中积累了成千上万的临时文件。因此，服务器变得更慢，前端屏幕开始在随机时间冻结。即使在服务器重启后，屏幕冻结也不断发生，而且频率越来越高。

- *恢复行动。*开发人员引入了一个后台工具，定期将临时文件从服务器上移走，并在以后删除它们。

- **Bug #4：大型政府税务信息系统的内存耗尽。**
  - *系统描述。*该系统被公司采用来提交其雇员的收入记录。用户将带有记录的文件上传到一个Java Web应用程序。
  - *错误描述和表现。*运行应用程序的Java虚拟机（JVM）由于JVM堆内存的耗尽而崩溃。随着堆内存消耗的时间增加，崩溃的概率也随之增加，当一家公司上传一个大文件时，发生了崩溃。
  - *恢复行动。*系统管理员不得不增加JVM堆的大小以允许上传大文件。

- **Bug #5：X.25数据通信设备（DCE）崩溃。**
  - *系统描述。*X.25
    DCE是一个连接到X.25网络的网络接口。在这个案例中，DCE是由一个大型电信设备供应商生产的，它基于一个通用的CPU，运行一个专有的实时操作系统。DCE被更新了一个新的软件功能，这个功能受到高数据包流量的影响。
  - *错误描述和表现。*在DCE上运行的软件在某些多次调用新功能的负载条件下会崩溃。该新功能使用了一个复制内存区域的函数（类似于C语言中的*memcpy*()函数），但所复制的数据量是所需复制的两倍。结果，一个共享链表被破坏了。由于进程的地址空间没有被侵犯，所以没有立即出现分段故障。当另一个不相关的进程试图访问这个列表时，发生了故障，导致DCE崩溃。
  - *恢复行动。*该故障可以通过重新初始化X.25 DCE来恢复，一旦确定了根本原因，就在生产中消除了。

- **Bug #6：自动取款机（ATM）的网络通信故障。**
  - *系统描述。*一台ATM使用X.25网络连接到一个服务器。X.25 DCE接口与服务器交换确认，以控制数据流。
  - *错误描述和表现。*当一个比特变量被错误地设置为1时，DCE没有正确地发送确认，导致通信失败。该故障是随机发生的，取决于内存的状态，因为ATM软件没有明确地将位变量初始化为零。
  - *恢复行动。*该故障可以通过重启ATM软件来恢复，并最终通过初始化比特变量来修复。

B. 曼德尔虫的分类

在我们对11个IT项目的分析中，我们总共发现了38个曼德尔漏洞。我们认识到，这些Mandelbugs可以被分为几类，在表I中进行了总结。上面描述的Bug #1代表了一个*Lag*类型的Mandelbug的例子，一个时间问题导致了管道的不一致状态，失败的请求在重试几次后可以成功执行。Bug #2到#5是*与老化有关的*Mandelbug的例子，其中Bug #2和Bug #3是通过清理系统资源来处理的，而Bug #4需要重新配置系统参数来恢复操作。Bug #5和Bug #6在确定了根本原因后得到了解决，但只是在最初的停机时间后。

表一
曼德尔虫的分类（从[43]的表I扩展而来）

曼德尔布格的类型	解释	虫子的数量
老化。	失败率增加或性能随时间下降，例如，由于资源被耗尽了。	
• 记忆泄漏	即使不需要，也不释放内存分配和对象。	3
• 光标泄漏	打开的数据库游标没有关闭。	2
• TCP老龄化	传输控制协议（TCP）达到一定数量后性能下降连接已被打开。	1
• 数值溢出	像序列号溢出的数字量。	1
• 分裂	由于频繁的插入和插入而在数据库文件中产生漏洞，导致性能下降。数据清理。	1
• 记忆践行者	共享的数据结构被参与者的进程破坏。	1
• 网络设备	网络交换机或路由器出现故障，如随机乱码一些比特或发送重复数据包的发生率越来越高。	2
比赛情况	由于并发和没有适当的同步基元，访问顺序发生变化。	5

滞后	一个特定组件的状态与另一个组件的状态在一小段时间内不同步，通常为 因为是异步处理。	7
超负荷工作	组件故障或由于一小段时间内工作量激增而导致的非常差的性能 时间。	2
边界	当达到一个限制时，如分配的最大内存量或当 负载增加超过某个配置的阈值而不老化。	5
超时	一个交易或会话或TCP连接超时，导致用户请求被中断。 被拒绝。	4
中止	当正在处理的某一请求被中止时，正常操作就会恢复。的原因是 耽误系统处理的请求可能不为人知。	2
重试	失败原因不明，但重试操作成功。	1
未初始化的位	由于假设未初始化的位被设置为零而导致的故障。	1

C. 恢复方法

从分析的38个Mandelbugs的描述中，我们确定了为处理它们而采取的恢复行动。这些恢复行动可以分为几类，它们是IT运营人员为减少关键业务应用程序的意外停机而采取的代表性行动。根据我们的经验，这些恢复行动在这11个IT系统之外也被采用。在Mandelbugs引起的故障后，最常采用的四种方法来恢复系统，如下：[1]。

- **重新启动。**这种恢复行动重新启动一个软件组件或服务，可能需要几秒钟，而且只能影响被重新启动的组件正在处理的事务。此外，这种恢复行动可以使用IT管理系统自动进行[44], [45]。根据我们的经验，大多数由老化和非老化相关的曼德尔布格引起的故障都可以通过软件重启来恢复。
- **重新启动。**这个恢复行动涉及到硬件或虚拟机的重启，可能需要几分钟的时间来完成。当故障涉及整个系统的资源（例如，一个共享的数据结构），简单的组件或服务重启是不够的，这种行动是必要的。
- **重新配置。**这种恢复动作在执行重新启动或重启之前改变硬件、虚拟机或应用程序的参数。参数的改变通常会增加系统资源或超时值的限制，或者涉及到系统资源的清理以提高性能。表I中列出的*Abort*类型的Mandelbug构成了一个特殊情况：在这里，重新配置涉及到删除当前处理的请求。重新配置可能需要长达几分钟的时间，这取决于参数调整是在应用程序（例如，增加软件服务的超时），还是在系统层面（例如，增加虚拟机的内存和存储，或将服务迁移到另一个具有不同硬件的物理机），以及恢复是否由自动化软件管理工具监督。
- **热修复。**热修复是对生产中的IT系统的源代码或系统软件的微小改变（例如，操作系统、运行时系统和库代码的更新）。热修复是在很短的时间内（最多几个小时）创建和应用的，不需要对变化进行广泛的测试。IT系统源代码中的前一种热修复可以消除故障的根源（例如，通过修复一个错误），或者可以避免其影响（例如，通过重试失败的操作）；系统软件中的后一种热修复也可以减轻曼德尔漏洞的影响（例如，通过引入操作系统资源的定期垃圾收集）。

在这四种方法都不能解决的故障情况下，需要通过彻底的测试和代码调试进行常规的错误修复，以避免故障的发生。修复错误需要几天的时间，这取决于故障背后的错误的复杂性。修复的错误与其他变化和新功能一起被包括在下一个软件版本中。

IV. 曼陀罗虫的恢复模式

在我们的恢复模型中，我们考虑的是一个生产中的IT系统，它由一套部署在底层硬件或虚拟机技术上的软件组件（应用程序或平台）组成。假设(i)软件组件可以被重新启动，(ii)软件组件可以通过不同的参数设置重新配置，如缓冲池

[1]对于*重试*类型的错误，用户的行为在重试几次后就会恢复正常，不需要对IT系统做什么。

数据库系统中的大小或分类区域的大小，以及（iii）服务器可以自动或手动重启。这些假设是非常现实的，并构成了恢复模型的基础。

我们通过流程图来描述IT系统在运行阶段的故障恢复行为，如图1所示，然后对该流程图进行解释。流程图描述了在故障发生后为恢复而采取的行动。故障可以由系统自动检测（我们称之为*自动检测*），通常是通过企业系统管理工具，也可以由人工检测（我们称之为*人工检测*）。检测之后是对问题的调查（我们称之为*诊断*）；最后是通过诊断选择的一个或多个行动来恢复故障（我们称之为*恢复行动*）。根据造成故障的错误类型（Bohrbug或Mandelbug），恢复过程遵循我们模型的四个不同分支之一。该模型对四种类型的bug进行了区分，具体如下。

- **可重启掩蔽的曼德尔布格**。可以通过重启受故障影响的软件组件或服务来掩盖的曼德尔故障。其他恢复行动，如重新配置或重启，也会掩盖故障，因为它们涉及重启。
- **可重新启动的曼德尔布格**。需要硬件或虚拟机重启以避免故障再次发生的曼德尔故障。
- **可重新屏蔽的曼德尔布格**。需要调整参数以避免故障再次发生的曼德尔故障；在这种情况下，简单的重新启动或重启是不够的。
- **Bohrbugs，和其他类型的Mandelbugs**。在重新启动、重启或重新配置后，不能通过重试失败的操作来掩盖的错误。它们包括在重试操作时持续表现出来的Bohrbugs，以及受环境条件影响的Mandelbugs，在重启、重新启动或重新配置后仍然持续存在。这些错误需要对问题进行深入调查，并进行人为干预，以消除故障的根本原因，如错误修复。

该模型为这四种类型的错误中的每一种都包括一个不同的分支。在每个分支中，都会尝试III.C小节中描述的恢复行动（根据一些恢复策略，如本节后面所讨论的）；而且，根据故障背后的错误类型，这些恢复行动会导致不同的结果。例如，在可重启掩盖的Mandelbugs（流程图中最左边的节点）的情况下，每个恢复行动最终都会将系统带入一个正确的状态，因为每个行动（重启、重启、重新配置、修复）都足以掩盖这种错误。在玻尔虫的情况下（流程图中最右边的节点），只有修复可以掩盖故障，而其他的恢复行动（重启、重启、重新配置）对这种类型的错误没有好处，最终应该由开发人员进行修复。

请注意，在重启、重启或可重新配置的曼德尔漏洞的情况下，热修复足以掩盖故障，因为热修复应该包括重启或重启以应用变化，同时改变重新配置的代码中的参数（如资源阈值和超时）。因此，热修复包括重新配置或重启，以重新初始化系统并掩盖故障。相反，在玻尔漏洞的情况下，热修复有可能不足以纠正漏洞，而故障仍然存在（在其他情况下，热修复所执行的重启、重新启动或重新配置足以保证故障不会持续存在，即使热修复无法消除问题的根源）。因此，在波尔布格的情况下，可能需要开发、测试和部署一个常规的错误修复程序，这可能需要几天的工作时间。

在恢复行动之前，我们引入了一个*诊断阶段*，在这个阶段，开发者或自动化工具在重启、重新配置、重新启动或修复中选择一个恢复行动。然后，根据错误的类型，该行动可以成功地将系统带到一个正确的状态，或者恢复行动可能不成功，而系统可能仍然表现出故障。在后一种情况下，我们假设尝试另一个恢复行动，通过执行另一个持续时间较长、避免失败可能性较大的行动（例如，如果重启失败，则尝试重启；重启需要更多的时间，但可以从可重启掩码和可重启掩码的曼德尔漏洞造成的故障中恢复过来）。

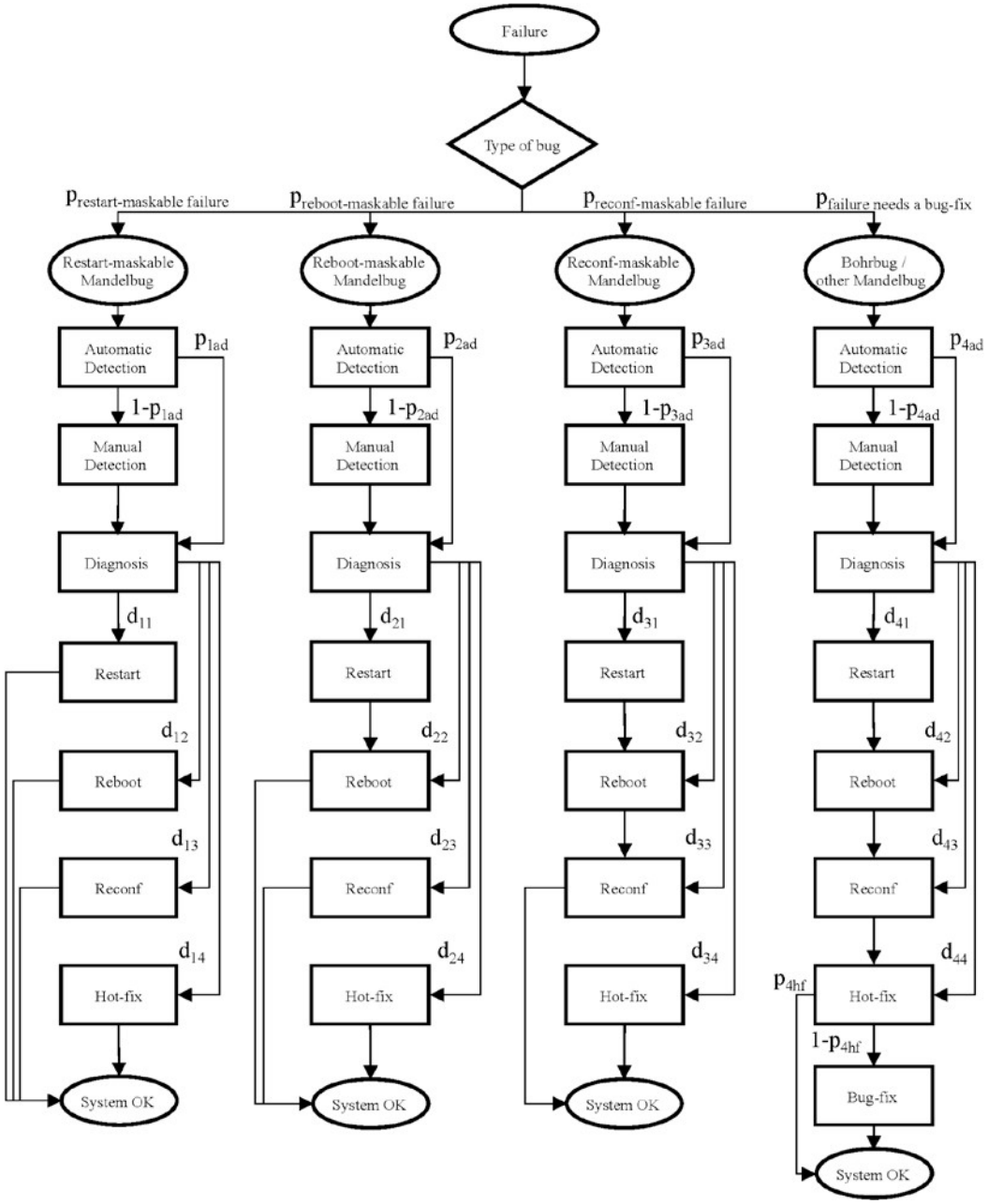


图1.故障后的恢复流程图。

流程图使我们能够模拟IT组织在处理软件故障时可以采取的不同策略。第一种策略（可以称为*基于诊断的恢复*）是调查故障原因，例如通过查看应用程序和操作系统日志，诊断故障的根本原因，并根据诊断结果执行恢复行动。诊断阶段可以包括收集和分析故障数据的时间，以及决定和批准恢复行动的时间。需要注意的是，在我们的模型中，在诊断和恢复发生之前，基本的错误类型是不知道的，而且有可能诊断并没有选择最适合抵消故障的恢复行动。换句话说，在某些情况下，诊断选择的恢复行动并不成功，而在其他情况下，选择的行动可能比其他同样有效的行动需要更多时间。

诊断过程（包括选择正确的和不正确的恢复行动）由概率d_{ij}，它表示在故障是由第*i种*类型的bug引起的情况下，选择第*j*种恢复行动的条件概率。其中*i*={ 1=*重启-可屏蔽的Mandelbug*, 2=*重启-可屏蔽的Mandelbug*, 3=*重新配置-可屏蔽的Mandelbug*, 4=*Bohrbug* }代表故障背后的bug类型，*j*={ 1=*重启*, 2=*重启*, 3=*重新配置*, 4=*热修复* }代表选定的

恢复行动。例如，考虑i=2（即一个可重启掩盖的Mandelbug引起了故障）：诊断将选择最佳的恢复行动（即重启，由j=2表示），概率为d_{22}。概率为d_{21}，诊断将选择重启，这将不能从故障中恢复，并且需要接着重启。而在概率为$d_{23}+d_{24}$的情况下，诊断将选择一个能从故障中恢复的行动，但与重启相比，成本增加。

这种恢复策略可以进一步分为两种策略：*基于人工诊断的恢复*和*基于自动诊断的恢复*。在手动诊断的情况下，开发人员或管理员手动检查故障证据，然后在诊断出故障背后的错误类型后选择最合适的恢复技术。这种策略需要一些时间来执行恢复，因为在开始恢复之前要对故障过程进行调查。然而，鉴于诊断是由仔细的分析和故障证据支持的，错误诊断的概率可以忽略不计。

在自动诊断的情况下，管理工具会自动收集和分析故障证据（例如，日志），并决定执行的恢复行动[46]。这种方法得到了IT管理系统的支持，比如IBM Tivoli[45]和HP Business Service Management软件[44]，它们为问题的确定提供了广泛的监控，以及基于策略的自动化和恢复。有理由认为，这种恢复策略比人工诊断要快，但恢复行动有时会出错（即对于某些故障，管理工具可能不会选择有效的恢复行动，在系统管理员没有为该类型的故障提供策略的情况下）。

另一种可能的策略被称为*升级恢复*，即每次发生故障时首先尝试重启，然后在重启不成功的情况下进行重启，然后在重启不成功的情况下进行重新配置，最后在其他所有恢复行动都失败时才对故障进行调试并制定修复方案。使用升级的恢复策略，没有诊断阶段：不分析故障的影响和根本原因（因此不考虑基本错误的性质），这种策略在每次故障时都会确定地选择相同的恢复行动序列（重启、重启、重新配置、修复）。

建议的流程图可以被采用来推导出故障恢复时间的半马尔可夫模型[47]，该模型在图2中描述。恢复行动可以有一个普遍分布的时间，唯一的假设是其平均值应该是有限的。该模型代表了一个通用IT系统从故障中恢复的时间分布。我们采用这个模型来推导出平均恢复时间（MTTR）的闭式表达，它也可以用来计算系统的稳定状态可用性。需要注意的是，这个模型没有考虑到由于故障而导致的处理损失，也不包括曼德尔漏洞（比如*重试*类型），对于这种类型，用户的重试会使故障大概率消失；这个模型不应该从用户的角度来解释，而应该从系统的角度来解释。还必须指出的是，该模型侧重于故障恢复策略，而不是故障预防策略，如软件年轻化；虽然该模型考虑了与老化有关的故障，但故障恢复和软件年轻化之间的相互作用会使该模型明显变得更加复杂，使其偏离主要焦点，因此留待未来研究。

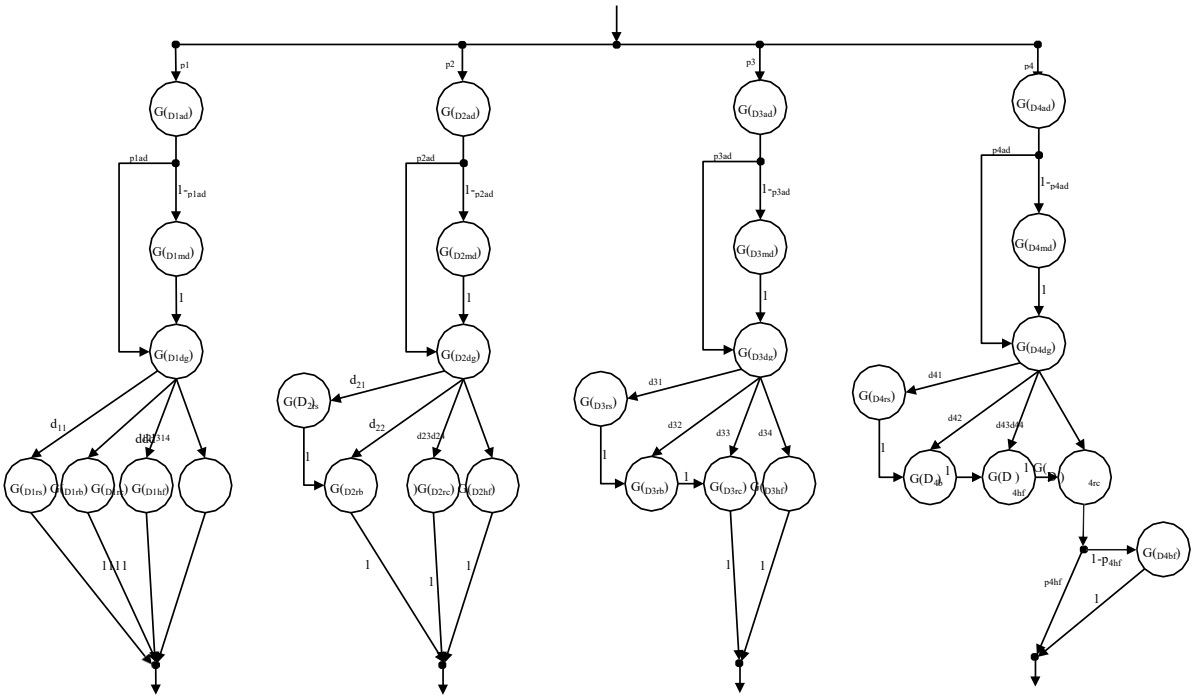


图2.故障的恢复时间分布。

该模型的一个重要方面是，错误类型的比例在不同的失败中保持不变。可能导致错误类型比例变化的两个因素是：(i)移除故障背后的错误的修复；以及(ii)
软件的新版本，这可能导致新的错误。我们用不随新版本变化的比例来模拟这种现象。这一假设得到了现场故障研究[7]、[25]的支持，这些研究观察到，一旦项目成熟，Mandelbugs和Bohrbugs之间的比例就会稳定在一个恒定的数值上，这一方面是由于在项目生命周期中逐渐清除旧的bug，另一方面是随着项目的新版本而引入新bug。虽然这种选择对于某些系统来说可能并不完全准确，但作为处理从曼德尔漏洞中恢复的具体情况的第一批模型之一，我们希望保持模型的简单。此外，对于大型的、复杂的系统来说，把这个模型表现为包括软件发布管理在内的稳定状态的操作是合理的。

从图2所示的恢复时间分布图中，我们得到以下MTTR的闭式解。

$$
\begin{aligned}
MTTP = {} & p_1 E[D_{1ad}] + (1-p_{1ad}) E[D_{1\%d}] + E[D_{1dg}] + d_{111rs} E[D]\, d_{12} + E[D_{1b}] + d_{131rc} E[D]\, d_{14} + E[D] \\
& + p_2 E[D_{2ad}] + (1-p_{2ad}) E[D_{2md}] + E[D_{2dg}] + d_{21} E[D_{2rs}] + E[D_{2rb}] + d_{22} E[D_{2rb}] + d23 E[D_{2rc}] + d_{24} E[D_{2hf}] \\
& + p_3 E[D_{3ad}] + (1-p_{3ad}) E[D_{3md}] + E[D_{3dg}] + d_{31} E[D_{3rs}] + E[D_{3rb}] + E[D_{3rc}] + d_{32} E[D_{3r\,b}] + E[D_{3r\,c}] + d33 E[D_{3rc}] + d_{34} E[D_{3hf}] \\
& + \pi_4 E[D_{4ad}] + (1-p_{4ad}) E[D_{4md}] + E[D_{4dg}] + d_{41} E[D_{4rs}] + E[D_{4rb}] + E[D_{4rc}] + E[D_{4hf}] \\
& + d_{42} E[D_{4rb}] + E[D_{4rc}] + E[D_{4hf}] + d_{43} (E[D_{4rc}] + E[D_{4hf}]) + d_{44} E[D_{4hf}] + (1-p_{4hf}) E[D_{4hf}]
\end{aligned}
\tag{1}
$$

V. 模型分析和推论

我们采用提出的模型来分析和比较恢复策略，并评估影响故障恢复效果的权衡和关键因素。我们的分析是根据以下研究问题组织的。

1) **哪种恢复策略是最有效的？**我们通过考虑不同的恢复策略，即（i）基于人工诊断的恢复，（ii）基于自动诊断的恢复，以及（iii）自动升级的恢复，分析了几种情况下的平均恢复时间和稳定状态（非）可用性。此外，我们对具有不同错误比例的不同类型的系统进行了分析，包括(i)高复杂度、低级别的系统，具有高比例的Mandelbugs；(ii)中等复杂度的系统；以及(iii)低复杂度的系统，具有高比例的Bohrbugs。

2) **哪些因素对故障恢复的影响最大，需要进行优化以提高可用性？**因为有几个因素涉及到故障恢复，正如建议的流程图中的参数和步骤所代表的那样，我们有兴趣确定那些对恢复影响最大的因素。了解这些因素将使开发人员能够调整故障检测、诊断和恢复行动；此外，设置自动恢复行动（如重新配置）可能会有成本，并增加系统的复杂性，因此从业人员可能对分析这些行动的成本和有效性之间的权衡感兴趣。

A. 模型参数

在这项研究中，我们使用所选的参数来分析恢复模型，以适当地反映运行中的IT系统。大多数参数可以由开发人员和IT运营人员设置，以反映他们的现实生活经验，如执行重启和重新启动的平均时间（取决于系统中采用的操作系统、中间件和虚拟机技术），以及故障检测所需的平均时间（取决于用户定义的参数，如超时值）。相反，鉴于故障是相对罕见的事件，而且关于故障的测量数据量往往有限，开发人员和IT人员很难估计与故障的概率和类型有关的参数。不幸的是，在IT行业中，由于缺乏出版物，以及缺乏数据共享，没有标准值可言。因此，我们根据科学文献中的定量数据（如果有的话）和我们在IT系统故障恢复方面的经验来设定模型参数，包括第三节中讨论的11个项目。模型参数见表二至表四，并在本节的下文中讨论。我们的恢复模型以及MTTR分析的一个优点是，它*不依赖于曼德尔虫引起的平均故障时间（MTTF）*，因为这很难估计。我们还指出，即使参数在不同的项目中可能有所不同，本文提出的模型是通用的，它允许从业者根据他们的具体项目轻松调整其参数。

我们分析的一个重要方面是故障类型的比例，因为恢复行动的有效性取决于导致故障的错误。开发人员可以通过利用被分析系统的问题跟踪和管理工具（以及其他现场故障数据源）或IT组织内的类似系统来获得这种信息，开发人员可以从中知道哪些是最常见的故障及其根本原因。在[25]中，我们提出了一个基于问题报告的这种分析的例子，其中我们研究了流行的开源软件项目中的Mandelbugs和Bohrbugs，这些项目经常在关键业务的背景下被采用。

表二

常见输入参数及其值的解释

参数	解释	价值
$E[D_{ad}]$	自动检测故障的平均时间	30秒
$E[D_{md}]$	人工检测故障的平均时间	5分钟
$E[D_{rs}]$	软件组件或服务重新启动的平均时间	10秒
$E[D_{rb}]$	硬件服务器或虚拟机重启的平均时间	1分钟
$E[D_{rc}]$	软件组件或服务重新配置的平均时间	5分钟
$E[D_{hf}]$	进行热修复的平均时间	30分钟
$E[D_{bf}]$	进行错误修复的平均时间	1天
垫子	自动检测故障的概率	0.9
p4hf	正确热修复的概率	0.9

我们将恢复模型的参数建立在我们之前对bug的经验分析上[25]。在该研究中，我们对错误的性质进行了深入的分析，包括错误是否受输入或操作的时间或顺序的影响，或受环境的影响，是否涉及资源泄漏等。这些信息为我们提供了关于这些项目中玻尔布格和曼德尔布格的相对比例的估计。特别是，经验分析表明，曼德尔布格的存在受到一个特定项目的规模、领域和技术的影响[25]。我们发现（根据每行代码的错误密度、软件结构和软件复杂度指标），可能有三种不同的情况。

- **高复杂度的**软件，有相当份额的曼德尔漏洞（~40.2%），影响硬件和操作系统相关的软件，如Linux内核和MySQL数据库管理系统。
- **中等复杂度的**软件，有~17.7%的曼德尔漏洞，如Apache HTTPD服务器；和
- **低复杂度**的软件，具有相对较低的Mandelbugs百分比（~7.5%），如Apache AXIS项目。

然后，对于每一种情况，我们把Mandelbugs的比例分为三部分（分别是可重新启动的Mandelbugs、可重启的Mandelbugs和可重启的Mandelbugs的子比例）。为此，我们分析了第三节中讨论的11个IT系统中的Mandelbugs，以确定有多少Mandelbugs可以使用三种恢复行动（重启、重启和重新配置）来恢复。从这个分析中，我们估计了可重启掩码、可重启掩码和可重新配置掩码的Mandelbugs的比例。表三总结了三种情况下每种错误类型的比例。我们观察到有相当比例的情况下，简单的重启就足够了（通常对几个与老化有关的bug来说是如此），而较小比例的bug，重新配置就足够了（像过载、限制、超时和中止类型的bug），或者属于重启类别（对一些老化的情况来说是如此，特别是与操作系统有关，也是在冒险进行bug修复前的最后手段）。

表三

三种复杂情况下的输入参数及其值的解释

伞兵	解释	**高**复杂度 (曼德尔布格的份额：40.2%)	**中等**复杂度 (曼德尔布格的份额：17.7%)	**低**复杂度 (曼德尔布格的份额：7.5%)
p_1	重新启动的概率- 可蒙蔽的曼德勒布格	0.2644	0.1166	0.0496
p_2	侦察的概率 可蒙蔽的曼德勒布格	0.0635	0.0280	0.0119
p_3	重启的概率- 可蒙蔽的曼德勒布格	0.0740	0.0327	0.0139
p_4	玻尔博格的概率	0.5981	0.8227	0.9246

至于诊断，我们选择了模型参数来反映第四节中讨论的三种恢复策略。与诊断有关的参数，在表四中作了总结，其选择如下。

- **基于人工诊断的恢复**。在这种情况下，恢复是由人类操作员监督的，他总是选择最好的恢复方案（错误诊断的概率可以忽略不计），同时花费较长的时间。对于这种策略，考虑到故障是由第*i*种错误类型引起的，选择第*j*种恢复行动的概率d_{ij} 为d_{ij} =1 iff i=*j*（即所选的恢复行动*j*与错误类型*i*相匹配），否则d_{ij} =0。例如，对于可重启屏蔽的错误，d_{22} =1，而d =d_{2123} =d_{24} =0。
- **升级的恢复**。在这种情况下，重启总是被用作发生故障时的第一个恢复行动，而不管实际导致故障的错误类型是什么。因此，我们有d_{i1} =1（即总是选择第一个恢复行动，即重新启动），以及d =d_{i2i3} =d_{i4} =0，对于每个错误类型*i*。
- **基于诊断的自动恢复**。在这种情况下，一个自动工具试图根据导致故障的错误类型来确定最佳的恢复行动。诊断会在短时间内自动选择恢复行动，但在某些情况下，所选择的恢复行动可能不是最佳行动（即，要么该行动不足以恢复故障，因此需要新的行动来恢复；要么该行动比足以恢复故障的行动成本更高）。对于这个策略，我们假设对故障的诊断是准确的，但不是完美的；根据以前对自动诊断技术的研究的实验数据[46]，正确的恢复行动是在95%的情况下被选中。因此，我们有d_{ij} =0.95 iff i=*j*，d_{ij} =0.05/3=0.016，否则（选择后一个值是为了确保d_{i1} +d_{i2} +d_{i3} +d_{i4} =1，对于每个错误类型*i*）。

表四
三种诊断情况下的输入参数及其数值的解释

伞兵	解释	手动诊断	升级的恢复	自动诊断
$E[D_{dg}]$	故障诊断的平均时间	30分钟	1秒	1秒
d_{11}	诊断出*重启*的概率，给定一个***可重启的***曼德尔布格病毒	1	1	0.95
d_{12}	诊断*重启*的概率，给定一个***可重启掩码的***Mandelbug	0	0	0.0167
d_{13}	诊断*重新配置*的概率，给定一个***可重启掩码的***Mandelbug	0	0	0.0167
d_{14}	诊断*出热修复*的概率，给定一个***可重启掩码的***曼德尔虫	0	0	0.0167
d_{21}	诊断*重启*的概率，给定一个***可重启掩码的***Mandelbug	0	1	0.0167
d_{22}	诊断*重启*的概率，给定一个***可重启屏蔽的***曼德尔虫	1	0	0.95
d_{23}	诊断出*重新配置*的概率，给定一个***可重启掩码的***Mandelbug	0	0	0.0167
d_{24}	诊断*出热修复*的概率，给定一个***可重启掩码的***曼德尔虫	0	0	0.0167
d_{31}	诊断*重启*的概率，给定一个***可重组的***Mandelbug	0	1	0.0167
d_{32}	诊断*重启*的概率，给定一个***可重组的***Mandelbug	0	0	0.0167
d_{33}	诊断*重组的*概率，给定一个***可重组掩码的***Mandelbug	1	0	0.95
d_{34}	诊断出一个*热修复*的概率，给定一个***可重组的***曼德勒虫	0	0	0.0167
d_{41}	诊断*重启*的概率，给定一个***波尔布格***	0	1	0.0167
d_{42}	诊断*重启*的概率，给定一个***波尔布格***	0	0	0.0167
d_{43}	诊断*重新配置*的概率，给定一个***玻尔巴格***	0	0	0.0167
d_{44}	诊断出一个*热修复*的概率，给定一个***玻尔巴格***	1	0	0.95

表五
从三种复杂情况和三种诊断情况下获得的九个案例

	案例1	案例2	案例3	案例4	案例5	案例6	案例7	案例8	案例9
复杂情况	高	高	高	中型	中型	中型	低	低	低
诊断情况	手动诊断	升级的恢复	自动诊断	手动诊断	升级的恢复	自动诊断	手动诊断	升级的恢复	自动诊断

总的来说，所考虑的情景和恢复策略导致了总共9种情况：三种恢复策略的组合，以及三种可供选择的虫子类型比例的组合。它们被列在表五中。

人工诊断可能需要大量的时间，甚至在关键业务系统中。我们看到在得出需要做什么的结论之前，诊断时间长达 30分钟。在某些情况下，诊断包括将系统、应用和技术领域的专家聚集在一起的延迟。相反，自动诊断是基于恢复策略的，并且可以在几秒钟内完成。

至于检测，关键业务系统的IT运营人员必须对所有已知的故障类型进行自动化的大部分故障检测和恢复程序。我们的保守估计（基于我们在IT系统方面的经验，如第三节所讨论的，以及以前的实验研究[48]）是90%的故障是自动检测的。自动故障检测将在几秒钟内发生，使用定期检查系统健康状况的IT管理工具。在手动检测故障的情况下，IT人员经常通过监测系统资源注意到故障症状，或者被客户投诉警告，而故障检测可能需要几分钟时间。

至于自动恢复，重新启动一个软件服务或其组件往往需要几秒钟，而重启则需要几分钟。根据我们的经验，重启Windows、Unix和Linux服务器需要两到十五分钟。重新配置需要几秒钟到几分钟，这取决于之后是重启（使用新的参数值）还是重启（可能在迁移到新机器之后）。热修复可能需要几分钟到两三个小时，而普通的错误修复需要严格的测试，如果立即发布，可以在几天内应用；或者，根据错误的严重程度，如果与软件的下一个版本一起发布，可能需要一周或更长时间。

B. 结果

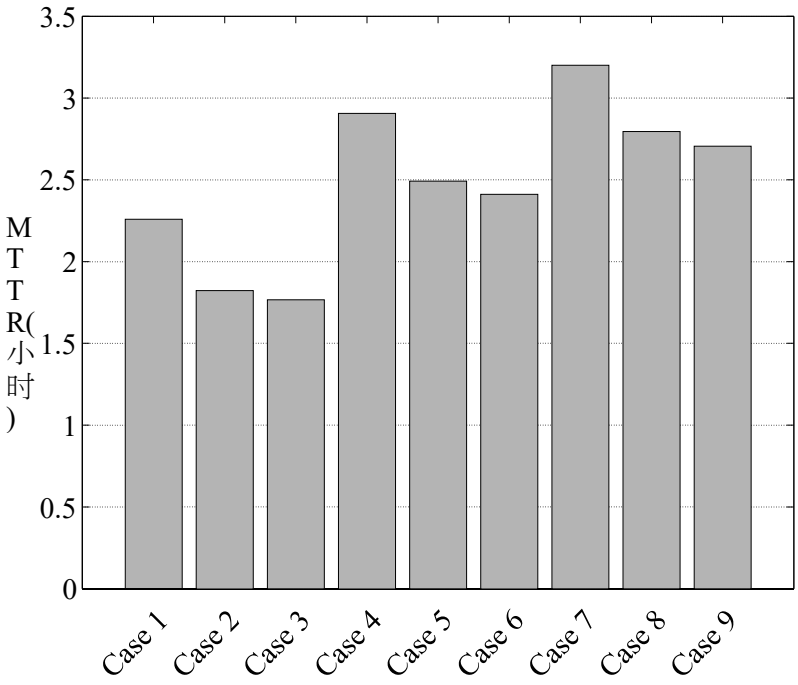
我们使用表二、三和四所示的输入参数来计算MTTR。此外，我们使用[49]计算了稳态不可用性（SSUA）。

$$\text{SSUA} = \frac{\text{!""\#}}{\text{!""\#!!""\#}} \qquad .(2)$$

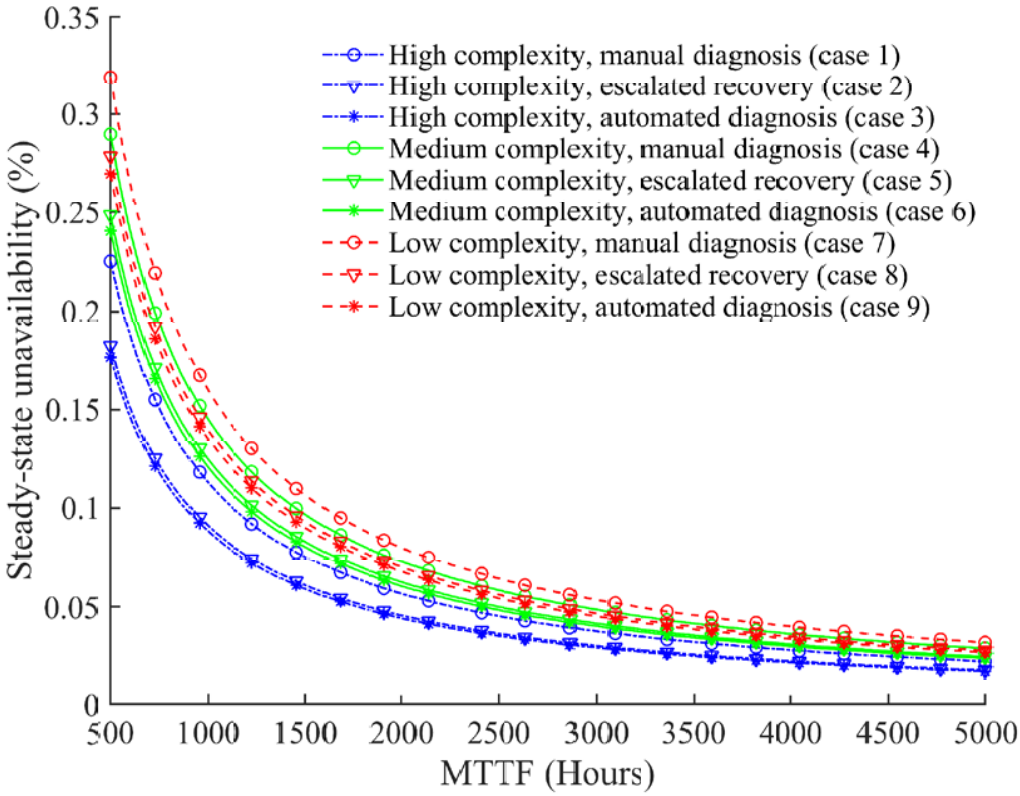
表六列出了根据我们的参数设置在所有九种情况下获得的平均恢复时间；这些值也在图3(a)中描述。由于稳态不可用性取决于平均故障时间，这很难估计，图3(b)显示了SSUA在不同的MTTF值下的变化情况。当然，各自的稳态可用性可以通过用SSUA减去1来轻松计算。值得注意的是，在具有高可用性要求的关键业务应用中，即使是几分钟的停机时间也会产生巨大的成本，并导致明显的服务中断。因此，尽管表VI和图3(a)中的MTTR值处于相同的数量级，但所分析的案例之间的差异对IT系统运营商和用户来说可能非常重要。

表六
九个案例的平均恢复时间

	高复杂度的系统			**中等**复杂度的系统			**低**复杂度系统		
	手工诊断（案例1）	**升级的恢复（案例2）**	**自动诊断（案例3）**	**手册诊断（案例4）**	**已升级恢复（案例5）**	**自动化的诊断（案例6）**	**手册诊断（案例7）**	**已升级恢复（案例8）**	**自动化的诊断（案例9）**
MTTR(小时)	2.2591	1.8224	1.7660	2.9060	2.4915	2.4119	3.1996	2.7952	2.7050



(a) 平均恢复时间（MTTR）。



(b) 稳态不可用性（SSUA）是MTTF的一个函数。图3.本研究中考虑的九个案例的MTTR和SSUA。

请注意，故障和恢复策略的影响取决于所考虑的具体IT系统，对于不同的系统，不同的参数（例如，不同比例的Mandel bugs，和不同的MTTF）可能会有所不同。虽然结果的范围局限于所考虑的情景，但由于采用了来自真实世界项目的故障数据，使这些情景对许多IT系统具有代表性，并使我们能够提出以下意见。

*观察1：人工故障诊断不如升级恢复有效。*在所有考虑的系统中（高、中、低复杂度），升级恢复的平均恢复时间低于人工诊断。这种差异表明，通过重新启动、重启和重新配置自动恢复Mandelbugs的收益高于在存在Bohrbugs的情况下尝试无用的重新启动、重启和重新配置所带来的惩罚。即使博尔布克是大多数的错误（在所考虑的情况下约60%或更多），自动恢复行动可以避免，至少在某些情况下，执行全面的问题诊断和错误修复，从而节省大量的精力，同时提高IT系统的可用性。如图4所示，影响人工诊断的主要因素是执行人工问题判断所需的时间。为了和升级恢复一样有效，手动问题确定应该在5分钟之内完成。然而，根据作者的经验，确定一个问题往往需要5分钟以上，最快的解决方案是尝试升级恢复并掩盖故障；只有在问题持续存在的情况下，才有必要对问题的根源进行深入调查。

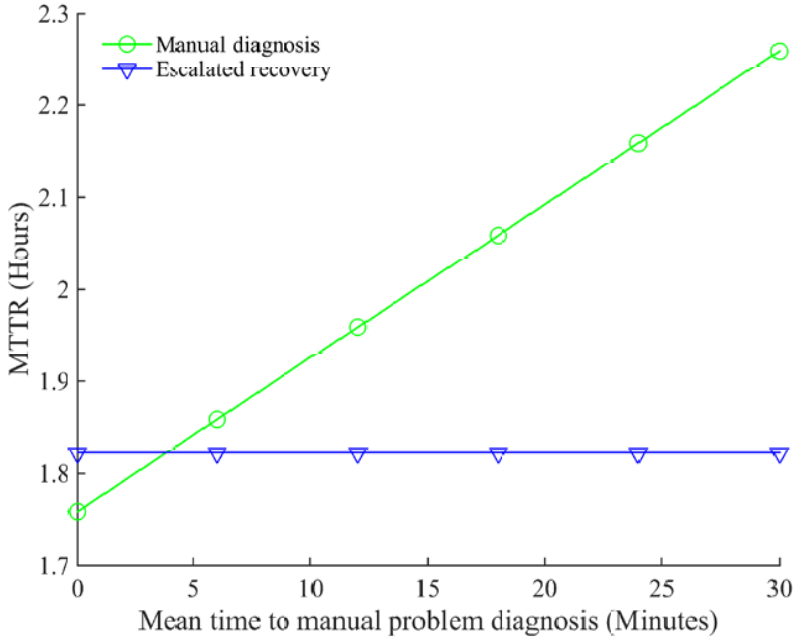


图4.在问题诊断平均时间的不同值下，人工诊断的MTTR与升级恢复（在高复杂度软件中）相比。

*意见2：自动诊断比人工诊断要好，但它并没有比升级恢复的效果好很多。*这个结果似乎并不直观，因为读者可以期望自动诊断能够快速、最佳地利用自动恢复行动，从而提供比升级恢复更好的结果。然而，自动诊断的收益往往是有限的。即使每种策略的平均恢复时间取决于它所应用的特定系统，三种恢复策略的相对有效性在所有考虑的系统中似乎是一致的。这个结果强调了在将自动恢复策略投入生产之前应该仔细评估其采用情况。特别是，影响自动诊断的有用性的因素是错误诊断的概率。如果错误的概率足够高，那么自动诊断就会触发无用的恢复行动（例如，在需要重新配置的时候重启系统）；或者更糟糕的是，会触发比必要的成本更高的恢复行动（例如，在重启就足以恢复故障的时候，需要IT管理员或开发人员的干预），增加恢复的总体成本，降低可用性。因此，高错误率会使自动诊断不值得带来额外的复杂性，甚至会使它产生反作用。因此，只有当开发者能够保证诊断策略的高准确性时，才应该采用它。例如，从图5可以看出，在高复杂度系统的情况下，当正确诊断的概率接近50%时，自动诊断的收益就变得不那么可观了，自动诊断在这个百分比以下甚至会产生反作用。如果不能保证准确的诊断，那么升级恢复是最好的策略，因为自动恢复行动的速度非常快，如重新启动、重启和重新配置。这一结果表明，IT管理员和开发人员应该对诊断的准确性进行测试，以获得这种信心，例如通过故障注入测试。

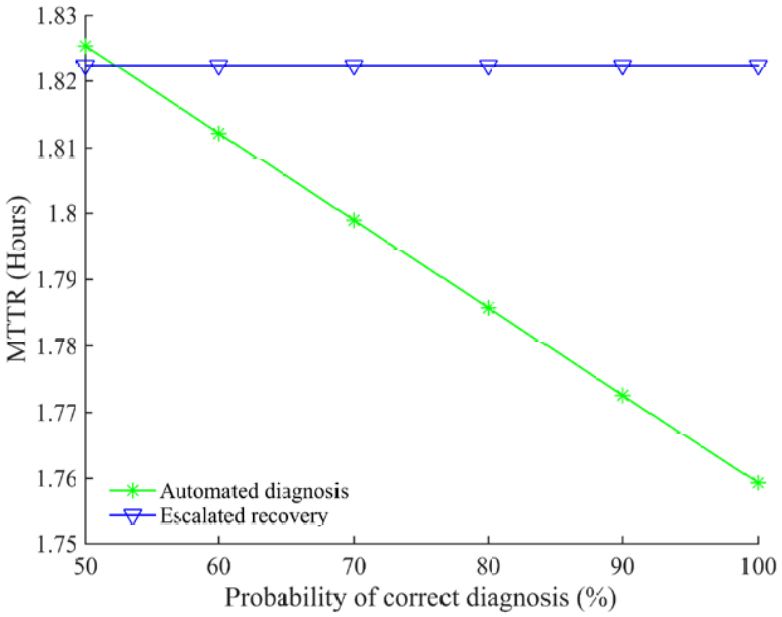


图5.在不同的正确概率值下，自动诊断的MTTR与升级恢复相比（在高复杂度软件中）。诊断。

请注意，即使低复杂度系统的MTTR较高，这一条件并不意味着它们的可用性一定比高复杂度系统低。低复杂度系统的MTTR较高，因为大多数故障是由布尔布格引起的；因此，较少的故障可以被自动恢复行动（重启、重启、重新配置）所掩盖，而其中较高比例的故障需要修复，这往往需要更多的时间。然而，（非）可用性是MTTR和MTTF的一个函数。从图3(b)中可以看出，如果低复杂度系统的MTTF足够高，其可用性就会高于高复杂度系统。例如，考虑一个具有升级恢复功能的高复杂度系统，MTTF为1000小时；从图3(b)可以看出，如果一个具有升级恢复功能的低复杂度系统的MTTF超过1545小时，其SSUA就会低于这个高复杂度系统。

为了更深入地了解恢复过程，我们对模型进行了参数化的敏感性分析。敏感性分析是一种确定对模型结果影响最大的因素的方法。它可以用来寻找系统中的恢复或可用性瓶颈，从而指导改进和优化。参数敏感性分析是通过计算每个参数的弹性来进行的。这里，弹性代表了由相应参数的百分比变化所导致的MTTR（或SSUA）的百分比变化。这个措施提供了一个统一的方法来比较不同测量单位的不同参数的影响。MTTR相对于一个参数p的弹性可以从相对于该参数的偏导数计算出来。

$$\text{弹性}\quad {}_{!""\#}(pp)=\frac{!}{!""\#(!)}\text{MTTR}^{!}(pp)\ 。\qquad (3)$$

SSUA的弹性可以用与MTTR类似的方法来计算。使用SSUA导数的连锁规则，其弹性可以表示为

$$\text{弹性}\quad {}_{!!"\#}(pp)=\frac{!}{!!"\#(!)}\text{SSUA}^{!}(pp)=\frac{!}{!""\#(!)}\text{MTTR}^{!}(pp)(1-\text{SSUA}(pp))。\ (4)$$

为了计算弹性，我们将所有参数固定为默认值（如表二至表四所示），并计算部分导数（相对于评估中的参数p），以及MTTR和SSUA函数。我们考虑了表二到表四中的所有参数，除了错误比例（我们根据定量的现场故障数据，考虑三种情况）和诊断参数d_{ij}（因为它们是由恢复策略强加的，我们之前已经讨论过）。

利用上述公式，我们计算出各参数的弹性，如表七所示。参数的排名是根据弹性的绝对值来排序的，排名在括号中显示。由于SSUA非常接近于0，MTTR的弹性与SSUA的弹性非常相似，对于所有参数和所有9种情况，MTTR的排名与SSUA的排名总是一样的。因此，表七只显示了MTTR的弹性值。正的弹性表示一个参数的增加会导致MTTR的增加，而负的弹性表示，如果一个参数值增加，MTTR会减少。

表七
MTTR相对于模型参数的弹性（以及括号内的排名）

(a) 高复杂度的软件系统

参数	案例1（自动诊断）	案例2(升级的恢复)	案例3（人工诊断）
垫子	-3.3199e-02 (5)	-4.1154e-02 (4)	-4.2469e-02 (4)
p4hf	-5.7186e+00 (1)	-7.0887e+00 (1)	-7.3152e+00 (1)
$E_{[爸爸]}$	3.6888e-03 (6)	4.5726e-03 (7)	4.7188e-03 (6)
$E_{[Dmd]}$	3.6888e-03 (7)	4.5726e-03 (8)	4.7188e-03 (7)
$E_{[Ddg]}$	2.2133e-01 (3)	1.5242e-04 (10)	1.5729e-04 (10)
邓小平	3.2513e-04 (10)	1.5242e-03 (9)	4.1443e-04 (9)
$E_{[Drb]}$	4.6819e-04 (9)	6.7271e-03 (6)	8.3250e-04 (8)
$E_{[Drc]}$	2.7311e-03 (8)	3.0734e-02 (5)	5.1079e-03 (5)
$E_{[Dhf]}$	1.3237e-01 (4)	1.6409e-01 (3)	1.7125e-01 (3)
$E_{[Dbf]}$	6.3539e-01 (2)	7.8763e-01 (2)	8.1280e-01 (2)

(b) 中等复杂度的软件系统

参数	案例4（自动诊断）	案例5 (升级的恢复)	案例6（人工诊断）
垫子	-2.5809e-02 (5)	-3.0102e-02 (4)	-3.1096e-02 (4)
p4hf	-6.1150e+00 (1)	-7.1322e+00 (1)	-7.3677e+00 (1)
$E_{[爸爸]}$	2.8676e-03 (6)	3.3447e-03 (7)	3.4551e-03 (5)
$E_{[Dmd]}$	2.8676e-03 (7)	3.3447e-03 (8)	3.4551e-03 (6)
$E_{[Ddg]}$	1.7206e-01 (3)	1.1149e-04 (10)	1.1517e-04 (10)
邓小平	1.1150e-04 (10)	1.1149e-03 (9)	1.4462e-04 (9)
$E_{[Drb]}$	1.6056e-04 (9)	5.9090e-03 (6)	3.9789e-04 (8)
$E_{[Drc]}$	9.3661e-04 (8)	2.8609e-02 (5)	2.6173e-03 (7)
$E_{[Dhf]}$	1.4155e-01 (4)	1.6510e-01 (3)	1.7118e-01 (3)
$E_{[Dbf]}$	6.7945e-01 (2)	7.9247e-01 (2)	8.1864e-01 (2)

(c) 低复杂度的软件系统

参数	案例7（自动诊断）	案例8 (升级的恢复)	案例9（人工诊断）
垫子	-2.3441e-02 (5)	-2.6832e-02 (5)	-2.7726e-02 (4)
p4hf	-6.2421e+00 (1)	-7.1451e+00 (1)	-7.3833e+00 (1)
$E_{[爸爸]}$	2.6045e-03 (6)	2.9813e-03 (7)	3.0807e-03 (5)
$E_{[Dmd]}$	2.6045e-03 (7)	2.9813e-03 (8)	3.0807e-03 (6)
$E_{[Ddg]}$	1.5627e-01 (3)	9.9377e-05 (10)	1.0269e-04 (9)
邓小平	4.3053e-05 (10)	9.9377e-04 (9)	6.4677e-05 (10)
$E_{[Drb]}$	6.1996e-05 (9)	5.6670e-03 (6)	2.6913e-04 (8)
$E_{[Drc]}$	3.6164e-04 (8)	2.7980e-02 (4)	1.8794e-03 (7)
$E_{[Dhf]}$	1.4449e-01 (4)	1.6540e-01 (3)	1.7116e-01 (3)
$E_{[Dbf]}$	6.9356e-01 (2)	7.9390e-01 (2)	8.2036e-01 (2)

弹性的排名显示了以下观察结果。

*观察3：正确的热修复的概率是最有影响的参数。*避免错误的热修复需要后续的错误修复，这是至关重要的。这个结果是真实的，因为如果一个IT系统进入了修复错误的步骤，那么它就需要更多的时间来恢复。在所有九个案例中，正确热修复的概率被排在第一位，其弹性比其他参数高几个数量级。因此，优化这个参数是很重要的，在应用热修复恢复故障时要尽可能多地注意。即使提高这个概率增加了热修复的持续时间，错误的热修复对MTTR和SSUA的影响也会大大增加。因此，花额外的时间来保证热修复的正确性是值得的。当然，不能在所有情况下都避免完全的错误修复，因为这种情况也取决于复杂性和错误的严重程度，但是在不是严格要求的情况下避免错误修复对可用性是有好处的。

*观察4：自动检测故障的概率有明显的影响。*在热修复和错误修复时间之后，这个参数一直被列为最有影响的参数之一。这个参数很重要，因为故障检测是所有类型的恢复行动的先决条件，无论发现故障后将采取何种行动，都必须执行。因此，避免在故障发生和发现之间的延迟，可以快速恢复，并减少不可用性。这个参数可以通过采用先进的IT管理系统进行优化，该系统提供了精确监控系统资源、进程和服务的设施；并且可以通过系统管理员提供的故障检测策略自动发现异常情况。

VI. 结论

在本文中，我们首先介绍了实际案例、曼德尔布格的类型，以及从曼德尔布格引起的故障中恢复的方法。然后，我们提出了一个使用流程图的恢复模型，并在流程图的基础上建立了一个半马尔可夫模型，以推导出IT系统MTTR的闭式解。该模型的目的是简单的，便于从业人员使用。最后，我们采用该模型来评估恢复行动和策略的相对价值，考虑了9种有代表性的情况，基于本研究和以前的故障数据。对该模型的分析表明，该模型有助于比较不同的恢复策略，获得关于一个策略何时最有效的见解，并确定关键参数。通过计算每个参数的弹性，进行了参数敏感性分析，以确定对曼德尔虫引起的故障的恢复影响最大的参数，因此IT管理员和开发人员应该对这些参数进行优化。这一领域的未来工作包括调查反应式方法（如本工作中分析的故障恢复策略）和主动式方法（如软件年轻化）之间可能的相互作用，这些方法的目的是在故障发生前预防故障，以支持调整这种主动式方法以优化系统可用性。

鸣谢

这项工作得到了Theo和Friedl
Schoeller博士商业和社会研究中心以及MIUR项目DISPLAY（PON02_00485_3487784）的支持。

参考文献

[1] J.D. Musa, *软件可靠性工程。更可靠的软件，更快，更便宜*。Tata McGraw-Hill Education, 2nd edition, 2004.
[2] Gizmodo.com，"全面回顾HealthCare.gov的问题"，见http://gizmodo.com/a-comprehensive-review-of-what- went-down-with-healthcar-1479224003，2013（最后访问时间：2014-08-29）。
[3] E. Wong, V. Debroy, A. Surampudi, K. HyeonJeong, and M. F. Siok, "Recent catastrophic accidents:Investigating how software was responsible," in *Proc. IEEE Intl. Conf. on Secure Software Integration and Reliability Improvement (SSIRI)*, 2010, pp.14-22.
[4] M.Grottke和K. S. Trivedi，"软件故障、软件老化和软件年轻化"，*J. 日本可靠性工程协会*，第27卷，第7期，第425-438页，2005。
[5] M.Grottke和K. S. Trivedi, "A classification of software faults," in *Supplemental Proc. IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2005, pp.
[6] M.Grottke和K.S.Trivedi，"对抗错误。Remove, retry, replicate, and rejuvenate," *IEEE Computer*, vol. 40, no. 2, pp.107-109, 2007.
[7] M.Grottke, A. P. Nikora, and K. S. Trivedi."空间任务系统软件中故障类型的经验调查"，在*Proc.IEEE/IFIP可靠系统和网络国际会议（DSN）*，2010年，第447-456页。
[8] A.Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*.John Wiley & Sons，第7版，2004年。
[9] M.Cavage，"无法绕过它：你正在构建一个分布式系统，" *《ACM通讯》*，第56卷，第6期，第63-70页，2013年。
[10] S.Moore，"多核是超级计算机的坏消息"，*IEEE Spectrum*，第45卷，第11期，第15页，2008。
[11] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot-A technique for cheap recovery, " in *Proc.USENIX操作系统设计与实现研讨会（OSDI）*，2004，第31-44页。
[12] D.Cavezza, R. Pietrantuono, J. Alonso, S. Russo, and K. Trivedi, "A study of the reproducibility of environment-dependent software failureures," in *Proc. IEEE Intl. Symposium on Software Reliability Engineering (ISSRE)*, 2014, pp.267-276。
[13] Y.Huang, P.E. Chung, C. Kintala, D. Liang, and C. Wang, "NT-SwiFT: Software implemented fault tolerance on Windows NT," in *Proc.USENIX Windows NT Symposium*, 1998.
[14] F.Qin, J. Tucek, J. Sundaresan, and Y.Y. Zhou, "Rx: Treating bugs as allergies-A safe method to survive software failures," *ACM SIGOPS Operating Systems Review*, Vol. 39, no.5, pp. 235-248, 2005.
[15] V.R. Basili, and B.T. Perricone, "Software errors and complexity: an empirical investigation," *Communications of the ACM*, vol. 27, no. 1, pp. 42-52, 1984。
[16] D.Perry和W.Evangelist, "An empirical study of software interface faults," in *Proc. of New Directions in Computing Conference*, 1985, pp.32-38.
[17] R.Chillarege, I.S., Bhandari, J.K., Chaar, M.J., Halliday, D.S., Moebus, B.K., Ray, and M.Y. Wong, "Orthogonal defect classification-a concept for in-process measurements, " *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp.943-956, 1992.
[18] B.Beizer, *Software Testing Techniques*.Van Nostrand Reinhold Co, 2nd edition, 1990.

[19] P.卡特，"常见的C语言错误，"，可在http://www.drpaulcarter.com/cs/common-c-errors.php（最后访问时间为2014-08-29）。

[20] A.Koenig, *C Traps and Pitfalls*.Addison-Wesley, 1989.

[21] V.Vipindeep和P. Jalote，"常见错误和避免错误的编程实践清单"，技术报告，IIT Kanpur，2005年，可在http://www.cse.iitk.ac.in/users/jalote/papers/CommonBugs.pdf，（最后访问时间：2014-08-29）。

[22] J.格雷，"为什么计算机会停止，对此可以做些什么？"Tandem Computers Tech.Rep. 85.7, 1985.

[23] C.C. Liaw, S. Y. H. Su, and Y. K. Malaiya."使用卡住故障测试集的延迟故障测试生成"，在*Proc. International Test Conf.*，1980年，第167-175页。

[24] S.Y. H. Su, I. Koren, and Y. K. Malaiya, "A continuous-parameter Markov model and detection procedures for intermittent faults, " *IEEE Trans.计算机*，第27卷，第6期，第567-570页，1978。

[25] Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K.S. Trivedi, "Fault triggers in open-source software:An experience report," in *Proc. IEEE International Symposium on Software Reliability Engineering (ISSRE)*, 2013, pp.178-187.

[26] Y.Huang, C. Kintala, N. Kolettis, and N. Fulton, "Software rejuvenation:分析、模块和应用"，在*Proc.容错计算年度国际研讨会（FTCS）*，1995，第381-390页。

[27] D.Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "A survey of software aging and rejuvenation studies," *ACM Journal on Emerging Technologies in Computing Systems*, vol. 10, no. 1, 2014.

[28] D.Cotroneo, R. Natella, and R. Pietrantuono, "Predicting aging-related bugs using software complexity metrics, " *Performance Evaluation*, vol. 70, no.3, pp. 163-178, 2013.

[29] F.Machida, J. Xiang, K. Tadano, and Y. Maeno, "Aging-related bugs in cloud computing software," in *Proc. Intl. Workshop on Software Aging and Rejuvenation（WoSAR）*, 2012.

[30] G. Carrozza, D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Analysis and prediction of Mandelbugs in an industrial software system, " in *Proc. IEEE Intl.软件测试、验证和确认会议（ICST）*，2013年，第262-271页。

[31] R.Chillarege, "Comparing four case studies on Bohr-Mandel characteristics using ODC," in *Proc. Intl. Workshop on Software Aging and Rejuvenation（WoSAR）*, 2013.

[32] R.Chillarege, "通过ODC触发器了解Bohr-Mandel错误，以及对其现场比例的经验估计的案例研究，" 在*Proc.Int. Workshop on Software Aging and Rejuvenation (WoSAR)*, 2011.

[33] Alonso, R. Matias, E. Vicente, A. Maria, and K.S. Trivedi, "A comparative experimental study of software rejuvenation overhead, " *Performance Evaluation*, vol. 70, no.39, pp. 231-250, 2012.

[34] S.Distefano和K. S. Trivedi, "Non-Markovian statespace models in dependability evaluation," *Quality and Reliability Engineering International*, vol. 29, no. 2, pp.

[35] T.Dohi, K. Goševa-Popstojanova, and K. S. Trivedi, "统计非参数算法来估计最佳的软件年轻化时间表", in *Proc. IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2000, pp.77-84.

[36] M.Grottke和B. Schleich，"测试如何影响老化软件系统的可用性？3, pp. 179- 196, 2013.

[37] Y.Liu, K. S. Trivedi, Y. Ma, J. J. Han, and H. Levendel, "电缆调制解调器终端系统中软件年轻化的建模和分析，" in *Proc. IEEE国际软件可靠性工程研讨会（ISSRE）*，2002年，第159-170页。

[38] K.S. Trivedi, G. Ciardo, B. Dasarathy, M. Grottke, A. Rindos, and B. Vashaw, "Achieving and assuring high availability," in *Proc. IEEE Workshop on Dependable Parallel, Distributed and Network-Central Systems (DPDNS)*, 2008, pp.

[39] K.Vaidyanathan和K. S. Trivedi，"软件年轻化的综合模型"，*IEEE Trans.Dependable and Secure Computing*, vol. 2, no. 2, pp.

[40] S.Garg, Y. Huang, C.M. Kintala, K.S. Trivedi, and S. Yajnik, "Performance and reliability evaluation of passive replication schemes in application level fault tolerance, " in *Proc.年度国际容错计算研讨会（FTCS）*，1999，第322-329页。

[41] K.S. Trivedi, D. Wang, D. J. Hunt, A. Rindos, W. E. Smith, and B. Vashaw, "IBM© Websphere©上SIP协议的可用性建模", in *Proc. IEEE Pacific Rim International Symposium on Dependable Computing（PRDC）*, 2008, pp.

[42] M.Grottke和K. S. Trivedi，"升级的故障恢复方法分析"，技术报告，2011。

[43] K.S. Trivedi, R. Mansharamani, D.S. Kim, M. Grottke, M. Nambiar, "Recovery from failures due to Mandelbugs in IT systems," in *Proc. IEEE Pacific Rim International Symposium on Dependable Computing（PRDC）*, 2011, pp.

[44] 惠普公司。*业务服务管理（BSM）*，可在http://www8.hp.com/us/en/software-solutions/business-service- management-overview.html（最后访问时间：2014-08-29）。

[45] IBM公司，Tivoli System Automation for Multiplatforms，，可在http://www.ibm.com/software/products/en/tivosystautoformult（最后访问时间为2014-08-29）。

[46] I.Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "将仪表数据与系统状态相联系。自动诊断和控制的构建块"，在*Proc.USENIX操作系统设计与实现研讨会（OSDI）*，2004，第231-244页。

[47] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*.John Wiley & Sons，第二版，2001。

[48] G. Candea, E. Kiciman, S. Zhang, P. Keyani, and A. Fox, "JAGR: An autonomous self-recovering application server," in *Proc.自主计算研讨会*，2003年，第168-177页。

[49] R.Sahner, K. Trivedi, and A. Puliafito, *计算机系统的性能和可靠性分析*。Kluwer Academic Publishers, 1996.

迈克尔-格罗特克在美国韦恩州立大学获得经济学硕士学位；在德国弗里德里希-亚历山大-
纽伦堡大学（FAU）获得工商管理文凭；并获得博士学位。2004年至2007年，他在美国杜克大学电子和计算机工程系担任副研究员和助理研究教授。2010年，他获得了德国联邦大学的Habilitation学位。他的研究兴趣包括统计学、计算机科学和商业管理之间的中间领域，如软件老化和年轻化、软件性能、软件工程经济学和随机建模。他在国际会议以及国际期刊上发表了关于这些主题的论文，包括IEEE Computer, IEEE Transactions on Dependable and Secure Computing, IEEE Transactions on Reliability, Journal of Systems and Software, and Performance Evaluation。他是IEEE和德国统计学会的成员。

Dong Seong Kim目前是新西兰基督城坎特伯雷大学计算机科学和软件工程系的讲师（相当于北美体系中的助理教授），自2011年8月起。他于2008年在韩国航空航天大学获得计算机工程博士学位。2007年，他曾在美国马里兰大学学院公园分校担任访问研究员。2008年6月至2011年7月，他在美国北卡罗来纳州达勒姆的杜克大学担任博士后研究员。他的研究兴趣是系统和网络的可依赖性和安全性。特别是异常入侵检测系统，无线特设和传感器网络的安全，物联网和云计算系统的可靠性，以及网络安全建模和分析。他是IEEE的成员。

Rajesh Mansharamani是一名自由职业的性能工程顾问，自2011年以来，为IT服务公司和证券交易所服务。他目前还是印度计算机测量集团的主席，该集团是一个由1500多名性能工程和容量规划专业人士组成的社区。从2006年到2011年，Rajesh是塔塔咨询服务公司（TCS）的副总裁和性能工程研究中心的首席科学家。从1999年到2006年，Rajesh在TCS领导企业性能工程小组，在此之前，他从1994年开始在塔塔研究设计和开发中心担任研究员。在他的职业生涯中，他曾从事过一些全国性系统的架构、设计和性能工程，涉及银行、金融服务、政府和药房等部门。拉杰什拥有威斯康星大学麦迪逊分校的计算机科学硕士和博士学位，以及IIT-Bombay的理工科学士学位。

Manoj Nambiar目前作为首席科学家在TCS工作，领导性能工程研究中心（PERC）。他还领导平行化和优化卓越中心，作为公司HPC计划的一部分。直到2011年，Manoj一直是PERC的高性能信息传递、网络和操作系统的研究负责人。在此之前，他一直是性能工程领域的顾问，专门研究网络和系统性能。他在国际会议和期刊上发表了关于测量、建模和性能分析的论文；还曾在会议的技术程序委员会任职。他的研究兴趣包括性能和可用性建模，高性能计算，容错计算，以及算法的设计和分析。Manoj拥有孟买大学的计算机工程学士学位（1994年），以及印度C-
DAC的VLSI设计研究生文凭（2001年）。他是IEEE的高级会员。

Roberto Natella目前是意大利那不勒斯联邦第二大学的博士后研究员，他于2011年获得该校计算机工程博士学位，同时也是Critiware
s.r.l.大学衍生公司的联合创始人。他的研究方向是关键任务系统的软件可靠性保证方法，包括可靠性基准测试、软件故障注入、软件老化和年轻化，重点是这些方法在工业项目中的实际应用。他曾为软件工程和容错计算领域的主要会议和期刊撰写论文，并担任审稿人。他是IEEE的成员。

Kishor S. Trivedi在北卡罗来纳州达勒姆的杜克大学电子和计算机工程系担任哈德逊主席。他拥有孟买理工学院的技术学士学位。(他拥有孟买理工学院的技术学士学位，以及伊利诺伊大学厄巴纳-
香槟分校的硕士和博士（CS）学位。他自1975年以来一直在杜克大学任教。他是一本著名的教科书的作者，题为《*可靠性、排队和计算机科学应用的概率和统计*》；约翰-
威利公司已经出版了修订的第二版（包括其印度版）。他还出版了另外两本书，题为《*计算机系统的性能和可靠性分析*》和《*排队网络和马尔科夫链*》。他是IEEE的研究员，也是IEEE计算机协会的黄金核心成员。他已经发表了500多篇文章，并指导了45篇博士论文。他是IEEE计算机协会技术成就奖的获得者，因为他对软件老化和年轻化的研究。他的研究兴趣是计算机和通信系统的可靠性、可用性、性能和生存能力以及软件的可靠性。他与工业界紧密合作，开展我们的可靠性和可用性研究。

可利用性分析，提供短期课程，以及开发和传播软件包，如HARP、SHARPE、SREPT和SPNP。