支持自适应容错的多处理器实时操作系统

Supporting Adaptive Fault-Tolerant in Multiprocessors Real-Time Operating System

陈 宇[1] 熊光泽[1] 杨 春[2]

(电子科技大学计算机科学与工程学院 成都610054)[1](四川师范大学数学与软件科学学院 成都610066)[2]

Abstract In multiprocessors real-time operating system, single fault-tolerant strategy can't adapt the variable and dynamic environment. Adaptive fault tolerance has some fault-tolerant strategies. According the change of environment, adaptive fault tolerance chooses a suitable strategy to allocate the systematic resource. We design adaptive fault tolerance as a module of a multi-processors real-time operating system. It makes the application programming easier, and enhances the reusability of the fault-tolerance management.

Keywords Software fault-tolerant, Adaptive, Multiprocessor, Real-time operating system

1 引言

实时系统的基本特性是任务响应时间的确定性和系统处理任务的高吞吐量[1]。同时，随着实时系统在安全关键领域内越来越广泛的应用，如核电站控制系统、空中交通管制系统、飞行控制系统、病人监护系统、雷达监视系统等，可靠性也逐渐成为实时系统的重要特性[2]。为提高实时系统的可靠性，主要的技术手段包括：避免错误、消除错误、容错和规避错误。在实时系统的运行过程中，容错(fault-tolerant)是最重要的可靠性保障手段。容错指如何保证在出现错误时系统仍能提供正确服务[3]。工程实际中，实时系统在出现错误时，利用冗余资源实现容错操作。

近三十年来，国内外研究人员已经提出了许多容错策略，并构建了许多实验和应用系统。然而，基于冗余的容错实时系统存在两个主要问题：一是，在系统正常运行时，冗余资源的能力被浪费了；二是，单依靠一种容错策略无法适应运行环境不断变化。当部分资源暂时或永久失效，或者系统负载提高时，若容错策略不变，则为满足关键任务的容错要求而预留的资源数不变，则留给其他任务的系统资源减少，将导致系统吞吐量的下降或任务的丢失。当新资源加入，若容错策略不变，则任务不能利用新资源提高容错能力。因此，理想的容错实时系统应该具有多种容错策略，系统根据当前存在的各种资源、系统的负载、实时任务的关键程度和容错要求，选择合适的容错策略，实现资源的最优配置，保证系统各种功能、性能指标的实现。

容错管理功能可以由应用程序自身实现，也可由实时操作系统实现。理想的，抽取容错管理中独立于任何应用的功能，由实时操作系统中的一个模块实现，而与应用相关的容错管理功能由应用自身实现。这样，既提高应用无关容错管理功能模块的可重用性，又降低了应用程序编程的复杂度。

我们通过对一种多处理器实时操作系统 RTEMS 的修改，使之具有以上特性。本文将分析实时系统的自适应容错技术；简要介绍实时操作系统——RTEMS；详细论述自适应容错管理模块的构成、基本功能和实现方法；最后我们给出结论。

2 自适应容错

资源冗余是实现容错的重要手段。在物理意义上，可以将冗余分为时域冗余和空间域冗余。在工程应用中，冗余可以分为硬件冗余、软件冗余和时间冗余[4]。在以上三种冗余技术中，硬件冗余技术最为成熟，而近年来，软件冗余和时间冗余也得到很大发展。软件冗余要求同一功能由多个独立设计的模块并行或串行实现，并通过对各模块运行结果的检测或比较决定最终的输出。目前采用的主要技术包括 P-E(Primary-Exception handling)、RB(Recovery Block)、DRB(Distributed Recovery Block)、NVP(N-version Programming)和 NSCP(N-Self-Checking Programming)[5]。时间冗余主要通过修改实时调度算法，为容错操作预留处理器空闲时间，这以降低正常情况下处理器的利用率为代价；而非精确计算的引入，以容错操作时的任务性能降级为代价减少系统对预留空闲处理器时间的要求[6]。显然，以上三种冗余技术的结合，是实现容错，提高系统可靠性的重要保证。

近年来，多处理器实时系统向着大规模、高复杂度的方向发展，其状态的变化不再是可以由简单的有限状态机加以描述的。例如，地空反导弹系统的雷达跟踪系统，在和平时期，由于跟踪监视的目标少，因而系统负载低；而战争时期，由于敌人导弹的饱和攻击，雷达跟踪系统需要跟踪大量来袭目标，其系统负载高，同时可能出现部分资源受到攻击而被暂时或永久破坏。显然，在出现硬件或软件的暂时或永久故障的情况下，为了保证关键任务仍能在规定的时限范围内完成运算，并输出正确的结果，容错是必不可少的。通常，为保证系统的容错要求，系统的设计往往以系统可能出现的最坏情况为基础。然而，最坏情况在系统的生命周期中所占的比例很小，所以为这种情况而预留的冗余资源在大多数时候被浪费了。此外，由于实时系统对空间和质量的要求十分苛刻，引入大量冗余资源也是不现实的。若减少系统的冗余资源数，任务可能因为无法及时获得足够的冗余资源而无法达到指定容错级别或直接被放弃。另一方面，不同的容错策略在不同的运行环境中，其效果也是不同的。例如，在冗余资源丰富，任务的容错可以采用多个模块并行的方式实现，如 NVP，DRB 和 NSCP。若系统

陈 宇 博士研究生，主要研究方向：实时操作系统，软件容错技术。熊光泽 教授，博士生导师，主要研究方向：实时计算机系统、实时软件可靠性评测。杨 春 讲师，主要研究方向：分布式系统与网络安全。

负载较高，则可采用模块串行运行实现容错，如 RB 和 P-E 等。综上所述，在系统运行的不同阶段采用同一种容错策略是不合理的。我们认为，在实时系统中应该有多种容错策略并存，根据运行环境的变化，选择合适的容错策略，在保证系统可靠性要求的前提下，灵活地配置系统资源，提高资源的利用率和系统的任务吞吐量。这就是所谓的自适应容错。

自适应容错的主要目的是提高实时系统的可靠性、性能和在恶劣环境下可生存能力。因此，自适应容错应该具有以下几个主要特征[7]：

高效：自适应容错机制应该能够提高系统的可靠性，同时其运行开销必须保持在可以接受的范围内；

高的资源利用率：自适应容错机制应该通过提高资源的利用率降低系统对冗余资源的需求，并能够保证在部分资源暂时无法使用的情况下，满足关键任务的可靠性要求；

通用性：自适应容错机制必须能够应付不同类型的系统错误，如外部设备错误、处理器错误、系统软件错误和应用软件错误，同时也应该能够处理暂时错误和永久错误；

响应时间的确定性：自适应容错机制必须能够在规定的时间内响应外部环境的变化，在不同的容错策略间实现切换；

可配置：自适应容错机制必须能够接受通过参数形式传递的应用程序对于容错的特殊要求，并基于这些参数进行容错策略的选择；

多样性：自适应容错机制应该包含多种不同的容错策略，以响应运行环境的变化；

界面友好：自适应容错机制作为多处理器实时操作系统的一个模块，应该具有简单、清晰的编程接口，以减轻应用程序设计者的负担。

目前，国外多家研究机构已经展开对自适应容错的实现机制和应用的研究。在二十世纪九十年代的前半段，对于自适应容错的研究主要围绕其基本概念和实现机制进行。Kim 和 Lawrence 对自适应容错在复杂的实时分布式应用进行了分析[8]，并提出了 action-level 容错的概念[9]，Bondavalli 等在分布式实时操作系统 Spring 上提出了自适应容错模型 FERT[10]，Gong 等提出了异类错误容错的自适应容错算法 BOD－1和 BOD-2[11]。基于以上对自适应容错原理和实现机制的研究，建立了一些具有自适应容错功能的实验系统，并逐步走向实用，其中包括 Electra[12]、Proteus[13]、ROAFS[14,15]和 Chameleon[14]等。但是，以上系统主要基于普通的分布式操作系统，为分布式应用提供自适应容错，只具有有限的软实时服务能力，不能满足硬实时系统的需求。

我们通过对多处理器硬实时操作系统 RTEMS 的修改，为其增加自适应容错管理模块，包含几种使用的容错策略、容错策略选择机制、系统资源监控机制等，使 RTEMS 既能保证硬实时任务的时限要求，又能适时地选择合理的容错策略，提高系统的资源利用率，保证系统的可靠性。

3 RTEMS 概述[17]

RTEMS 全称是多处理器系统实时执行体。作为一种自由软件，RTEMS 的未经验证的源码可以通过互联网免费获得。RTEMS 以 LINUX 作为开发平台，代码的绝大部分由 C 实现，少量代码由汇编实现，并且以 GNU 的 GCC 和 GDB 作为编译和调试工具。两年来，我们分析 RTEMS 的源码，修改其中的错误，并在此基础上对其进行全面的实验。我们认为：RTEMS 作为一种源码开放的实时执行体，其性能和功能已经达到目前在市场上广泛应用的商用实时操作系统，如 VxWorks、pSOS 等的水平。

RTEMS 是一个多任务、模块化、支持多处理器的实时操作系统，能为实时应用提供高性能的应用支撑平台。RTEMS 由多个相互独立的功能模块组成，模块间通过接口相互调用，因此修改模块内部代码不会影响其他模块。RTEMS 内部采用层次化的结构（见图1），实现核心模块功能的代码放在执行体核心中，每个 API 的功能通过执行体核心中多个核心函数组合加以实现，而核心函数间也可以相互调用。CPU 相关代码实际上是硬件抽象层，使执行体能够在不同芯片集间方便地移植。图2所示为 RTEMS 的应用体系结构。

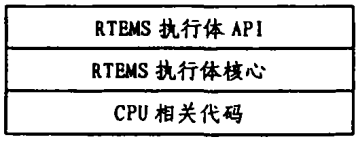

图1 RTEMS 的结构

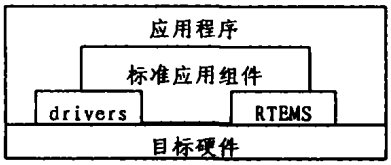

图2 RTEMS 的应用体系结构

RTEMS 支持同构和异构多处理器硬件平台。RTEMS 将可供任意结点访问的任务、队列、事件、信号、信号量和内存块等实体定义为全局对象，内核通过对象 ID 信息确定其所在的结点。应用程序对全局对象的访问只需知道其 ID。当应用程序访问目标对象时，内核首先通过对象 ID 确定对象位置。若为本地对象，则直接访问之。否则，内核首先在本结点上建立一个代理（proxy），并由代理将对象 ID 和访问要求作为消息发往对象所在结点。目标结点的服务器（server）接收并处理该消息，创建一个任务，并由该任务完成访问工作。最终，结果通过 server/proxy 返回给应用程序。在消息传输和访问操作期间，应用程序可以选择等待或继续运行。因此，同一个应用程序可能发起多个远程操作。RTEMS 对多处理器的连接介质和拓扑结构不做假设，消息的发送由发送函数调用物理硬件的设备驱动程序实现。

4 自适应容错管理模块（AFTM）的结构分析

自适应容错的管理将作为一个模块存在于 RTEMS 的核心执行体，该模块的核心是自适应机制。该机制是根据系统存在的资源和任务的资源、容错要求，确定任务采用的容错策略，并为任务分配相应的资源。当系统可用资源数或系统负载出现变化时，自适应机制调整任务的容错策略，重新分配相应的资源。在这一节中，我们将简要介绍本模块采用的几种基本容错策略，分析模块的各组成部分及其功能，分析为实现 AFTM 而需对 RTEMS 原有模块进行的修改。

4.1 基本容错策略

理论上，AFTM 可以包容任何容错策略。但是，容错策略实际上可以分为并行、串行和简单错误处理三类。并行容错指同时在多个结点上运行同一任务的多个不同版本，并通过比较决定最终的输出，如 DRB、NVP 和 NSCP。这一策略可以在较短时间内获得运算结果，但同时占有多个结点并协调工作

是十分复杂，且开销较大。串行容错指任务的不同版本在一个结点上串行执行，仅当前一个版本无法通过测试才运行下一个版本，如RB。在出错情况下，任务的最终响应较并行容错迟缓，但正常情况下，两者几乎一致。同时运行于单一结点使串行操作具有简单、开销小的特点。简单错误处理通常指异常处理，当任务运行出错时，转而运行简单的错误处理小程序，如记录错误状态等，将系统恢复到任务执行前的正确状态，然后直接返回到父任务，由父任务决定随后的操作。简单错误处理不能获得任务的正确输出，但能够保证系统始终处于正常状态。因此，AFTM从上述三类容错策略中分别选出一种，作为基本容错策略，其中包括P-E、RB和DRB，由自适应决策机制根据当前系统状态选择合适的容错策略。在多种并行容错策略中选择DRB是因为：DRB是RB的扩展，是在RB的基础上实现的，选择DRB有利于代码的重用。

4.2 结构分析

AFTM应具有以下基本功能和数据结构：自适应决策(AD)、容错策略管理(FTS)、资源管理(RM)、系统状态管理(SSM)和系统状态库(SDB)。以下，我们将就各组成部分做详细分析。

自适应决策(AD)：AD是AFTM的核心，它根据实时任务的自身属性、容错要求，利用资源管理确定本地资源是否满足其采用某种特定的容错策略时的资源需求。为帮助AD确定新任务的容错策略，我们为RTEMS的任务增加新的属性——重要性，并根据该属性将任务分为两类：关键任务和普通任务。对于新到达的关键任务，AD必须保证其在规定时限内获得正确输出。当关键任务在本地结点无法获得足够的资源时，AD通过RTEMS的任务管理模块在就绪任务队列中选择尚未被调度过的部分普通任务放入资源等待队列，释放这些普通任务的资源供关键任务使用。若通过释放普通任务占用的资源仍无法满足关键任务的需要，或关键任务选择并行容错策略，AD将根据系统状态库中的系统状态信息，选择关键任务的可执行结点。当新到达的普通任务在本地结点无法获得足够的资源时，则将其放入决策等待队列。为避免关键任务的执行代码在结点间的传递消耗大量时间，降低任务的响应速度，关键任务的执行代码均存放在系统的共享内存中。

容错策略管理(FTS)：AFTM将实时任务的容错结构的实现与功能的实现分离。当AD为任务选定容错策略后，FTS调用RTEMS的任务管理模块，按照指定的容错策略，创建实际运行的任务，并将其插入本结点的就绪任务队列。若任务选择并行容错策略，FTS根据AD选定的结点，利用远端结点的任务管理模块创建实际运行的任务，并插入就绪任务队列。除构建任务的容错结构外，FTS还负责与任务功能实现无关的容错管理，如任务运行环境状态信息的保存，以便在任务进行恢复操作时使用。由于FTS实现的是独立于实时任务的通用容错操作，因而可以通过RTEMS提供的容错实时运行库实现[17]。

资源管理(RM)：RM通过调度分析确定新到达的任务是否满足RTEMS的调度算法的可调度条件，AD以此为根据决定所选容错策略是否合适。当本地结点因任务运行结束而出现新的空闲资源时，RM调用RTEMS的任务管理模块，从资源等待队列中选择合适的普通任务重新插入就绪任务队列。若资源等待队列为空或空闲资源无法满足任意普通任务的要求，RM激活AD，对决策等待队列中的普通任务进行自适应容错决策。需要指出的是，当处于资源等待队列和决策等待队列中的普通任务超越其最大延迟时间(完成任务操作最迟开始执行时间)，该任务的运行将被放弃。

系统状态管理(SSM)*和系统状态数据库*(SDB)：AFTM通过SSM交换系统中各结点的状态信息，并将其存入SDB。RTEMS中，每个结点均有独一无二的ID。当一个新的结点加入系统时，将利用SSM广播其ID，而其他结点通过SSM接收并更新自己的SDB。当系统正常运行时，每个结点的SSM是一个周期任务，根据结点ID信息，周期地向其直接后继发送状态信息并由该后继的SSM接收。当某一SSM在规定时限内没有接到其直接前驱的状态信息，则认为该结点崩溃。在这种情况下，SSM将更新本结点的SDB，并在整个系统内部广播该消息。

综上所述，基于AFTM的多处理器实时系统的每个结点都应具有图3所示的结构。

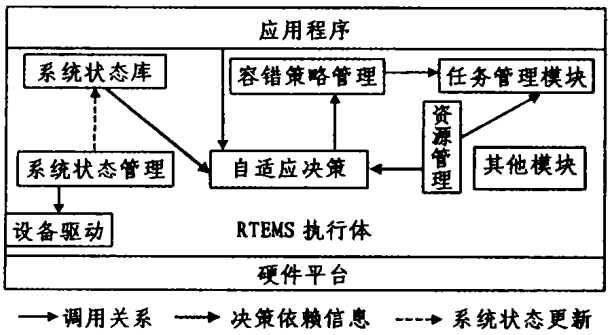

图3 基于AFTM的实时系统的结构(一个结点)

4.3 几点说明

以下，我们将对AFTM实现过程中的一些特殊之处加以说明。

1)因为实时任务优先级的先后次序不足以表示任务间相对的重要性，例如，RM调度以任务的周期确定优先级，而关键任务的周期可能大于普通任务。所以增加重要性作为区分任务关键程度的属性。

2)任务管理模块需要维护两个新的任务队列——资源等待队列和决策等待队列。决策等待队列实质上是一个请求队列，因为该队列中所有项并非真正的任务，而是任务创建的请求。

3)感知系统中出现错误结点的最大延迟是SSM的运行周期加消息发送的最大延迟。为避免选择出错结点作为并行操作的结点，发起容错决策的结点总是首先选择其直接前驱作为并行操作的结点。因为，每个结点总是最先获得它的直接前驱的状态信息。

4)在这里，我们假设结点间的物理链路是无错的。但在系统实际运行时，通信链路的出错和容错是必须考虑的。

结论 随着多处理器实时系统在安全领域内的越来越多应用，可靠性已经成为衡量系统优劣的一个十分重要的因素。在系统运行过程中，为提高系统的可靠性，容错技术被广泛的应用。传统的实时系统往往在应用程序设计时，将功能与容错作为一个整体实现。因而，采取何种容错策略，在系统构成之初就已经确定，并将在该应用的整个生命周期中存在。然而，随着大型、高复杂度的多处理器实时系统的出现，仅依靠某一种固定容错策略无法适应多变的运行环境和系统负载。自适应容错成为解决这一问题的重要手段。

自适应容错将几种适用于不同运行环境和系统负载情况

的容错策略结合起来，根据实时任务本身的属性、系统资源现状和外部环境，动态地为任务选择当前状态下最佳的容错策略，在保证系统可靠性要求的前提下，灵活地配置系统资源，提高资源的利用率和系统的任务吞吐量。自适应容错作为实时操作系统的一个组成部分，为应用提供通用的容错支持，减轻了应用程序编程的工作量。自适应容错的实现必须高效(开销小)、通用(可以应付不同类型的错误)、确定(及时响应外部环境的变化)，否则将得不偿失。

在本文中，我们建立了基于 RTEMS 的 AFTM 模型，并对其中的每个部分的构成、基本功能和实现方法进行了详细的描述。我们希望通过对该模型的研究和实现，深入地研究自适应容错的原理及其相关实现技术，推动其在工程实际中的应用。

参 考 文 献

1 Furth B, Halang W A. A Survey of Real-Time Computing Systems. International Journal of Mini and Microcomputers, 1994, 16 (3)
2 Cristian H. Understanding fault-tolerant distributed systems. Communications of the ACM, 1991, 34(2): 56～78
3 Jahanian F. State restoration in real-time fault-tolerant system, Complex System Engineering Synthesis and Assessment Technology Workshop, July 1992. 21～29
4 Thuel S R, Strosnider J K. Enhancing Fault Tolerant of Real-Time Systems through Time Redundancy, G. M. Koob and C. G. Lau, ed., Kluwer, 1994. 265～318
5 Lyu M R. Software Fault Tolerance, Wiley, 1995
6 陈宇，熊光泽. 基于非精确计算模型的容错单调比率调度. 计算机科学，已录
7 Hecht M, Hecht H, Shokri E. Adaptive Fault Tolerance for Spscecraft, IEEE 2000
8 Kim K, Lawrence T. Adaptive Fault Tolerance: Issues and Approaches. In: Proc. IEEE Workshop on Future Trends of Distributed Computing Systems, 1990. 38～46
9 Kim K. Action-level fault tolerance, in Advances in Real-Time Systems, S. H. Sang ed., Prentice Hall, 1994. 415～434
10 Bondavalli A, Stankovic J, Strigini L. Adaptive Fault Tolerance for Real-Time Systems, in Responsive Computer Systems: Steps toward Fault-Tolerant Real-Time System, D. S. Fussell and M. Malek ed., Kluwer, 1995. 187～208
11 Li Gong, Goldberg J. Implementing Adaptive Fault-Tolerant Services for Hybrid Faults: [Technique Report]. SRI International, 1994
12 Landis S, Maffeis S. Building Reliable Distributed Systems With CORBA, in Theory and Practice of Object Systems, Wiley, 1997
13 Sabnis C, Cukier M, et al. Proteus: A Flexible Infrastructure to Implement Adaptive Fault-Tolerance in AquA. In: Proc. of the 7th IFIP IWC in DCCA, 1999. 137～156
14 Shokri E, Crane P. Architecture of ROAFTS/Solaris: A Solaris-Based Middleware for Real-time OO Adaptive Fault Tolerance Support. In: Proc. COMPSAC98, 1998. 90～98
15 Shokri E, Hecht H, Crane P, et al. An Approach for Adaptive Fault Tolerance in OO Open Distributed Systems. International Journal of Software Engineering and Knowledge Engineering, 1998, 8(3)
16 Kalbarczyk Z, Iyer R K, Bagchi S, et al. Chameleon: a Software infrastructure for Adaptive Fault Tolerance, IEEE Transactions on Parallel and Distributed Systems, 1999, 10(6): 560～579
17 陈宇，罗蕾，陈红蓉，蔡建平. 一个外军嵌入式实时操作系统-RTEMS. 见：第十届全国抗恶劣环境计算机学术年会论文集. 2000. 67～72
18 陈宇，熊光泽. 支持应用软件容错的实时运行库技术

(上接第111页)

中共享。本算法同时也产生用于刷新协议和恢复协议执行时所必须得验证信息，并将其放入各服务器的 ROM 中。

周期 t 的刷新协议 refresh(t)

在服务器 i 上执行：

1. 产生签名密钥对($S_i(t)$, $V_i(t)$)和加密密钥对($E_i(t)$, $D_i(t)$)，产生签名 $S_i(t)(E_i(t))$。广播 $K_i=(V_i(t), S_i(t)(E_i(t)))$。若有 $S_i(t-1)$，则产生签名 $S_i(t-1)(K_i)$。

2. 服务器联合产生一个签名 Scert(KeyTable)，其中，$KeyTable=[K_1, \cdots, K_n]$。

3. 验证 Scert(KeyTable)的有效性。

4. 为需要恢复影子的服务器运行重构协议。

5. 对包括 Scert 在内的长效秘密运行刷新协议。

本算法的目的是刷新所有的长效秘密，包括 Scert。其中步骤1中，如果收到了同一个服务器发送的两个或两个以上的经过加密的不同信息，则丢弃，一个也不要；如果收到一个加密的信息，一个明文信息，则接受加密的信息；如果收到两个明文信息，则丢弃，一个也不要。步骤2中，要恢复的服务器不参与签名的产生。步骤4中的重构协议可以选择文[7]中的重构协议。步骤5的刷新协议可以选择文[2]中的算法。

清除攻击后的周期性恢复协议

设对服务器 i 进行恢复

1. 从 ROM 中读出有关信息，包括 Vcert，并验证正在执行的主动安全程序 P_i 是否未被修改，即程序是否与 Scert(P_i)吻合。若一切无误，则等待进行刷新协议的恢复。否则，恢复失败。

本算法用于侦测到成功的攻击后的恢复。此时，假定攻击者已被清除掉，而且主动安全程序从磁盘上重新调入运行。

结束语 主动安全技术是一种新型的安全技术，必将在电子商务和开放网络中起到关键性的作用。本文给出的主动安全技术的实现算法对架构主动安全系统至关重要，它拉近了主动安全技术与应用的距离。主动安全的功能计算与具体算法相关，因此有必要进一步将给出更多的针对具体密码学算法的主动实现。

参 考 文 献

1 Ostrovsky R, Yung M. How to withstand mobile virus attacks. In: Proc. of the 10th ACM Symposium on the Principles of Distributed Computing, 1991, 51～61
2 Herzberg A, et al. PROACTIVE SECRET SHARING Or: How to Cope With Perpetual Leakage? Advances in Cryptology——Crypto 95 Proceedings, Lecture Notes in Computer Science, 1995, 963: 339～352
3 Gennaro R, et al. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. In Eurocrypt '99, pages 295～310
4 Frankel Y, et al. Proactive RSA. In Advances in Cryptology——Crypto'97, 1997, 440～454
5 Rabin T. A Simplified Approach to Threshold and Proactive RSA. Crypto'98, LNCS, Springer-Verlag, 1998, 1462: 89～104
6 Frankel Y, MacKenzie P, Yung M. Adaptively-Secure Optimal-Resilience Proactive RSA. ASIACRYPT'99. 180～194
7 Herzberg A, et al. Proactive Public Key and Signature Systems. In: 1997 ACM Conf. on Computers and Communication Security, 1997
8 Canetti R, Halevi S, Herzberg A. Maintaining Authenticated Communication in the Presence of Break-ins. Journal of Cryptology, 2000, 13: 61～105
9 Barak B, et al. The Proactive Security Toolkit and Applications. In: ACM Conf. on Computer and Communications Security 1999. 18～27