Tutorial

Revista de
Informática
Teórica e Aplicada

Fundamentos de Tolerância a Falhas

CNPq - 1.03.04.00-2

Ingrid E. S. Jansch-Pôrto
II/CPGCC – Universidade Federal do Rio Grande do Sul

Sumário

Este artigo apresenta, de forma objetiva mas didática, conceitos básicos e a terminologia da área de Tolerância a Falhas, enfatizando o conceito de segurança em sistemas de computação, os fatores que comprometem esta segurança, os meios para alcançá-la e as suas medidas usuais. Na seqüência, são discutidas as técnicas de confinamento, detecção, diagnóstico e tratamento, incluindo técnicas e mecanismos de recuperação e reconfiguração.

Abstract

In this paper, the basic definitions and the associated terminology used in fault tolerance domain are presented, in an objective but didatic way, emphasizing the dependability concept in computer systems, its impairments, the means to reach it and its usual measures. In the sequence, there are discussed fault confinement, detection, diagnostic and treatment, including techniques and mechanisms of error processing as recovery and reconfiguration.

Palavras-chave: Segurança, falha, confiabilidade, detecção de erros, confinamento de danos, recuperação de erros, tratamento de falhas, redundância em hardware e software.

1. Introdução

Tolerância a falhas se relaciona às áreas de arquitetura e projeto de sistemas de computação. Apesar de tolerância a falhas ser obtida mais efetivamente quando as técnicas usadas para alcançá-la forem suportadas por mecanismos disponíveis nos níveis mais baixos dos sistemas de computação, o tema não se restringe apenas ao hardware.

As falhas a serem toleradas podem ter sua origem tanto em hardware como em software, sendo igualmente prejudiciais nos dois casos. Assim também, as técnicas empregadas para aumentar a confiabilidade dos sistemas podem ser implementadas tanto em hardware como em software para detectar e tratar falhas de qualquer dos dois tipos.

Neste artigo são apresentados inicialmente os conceitos básicos e a terminologia da área, enfatizando o conceito de segurança em sistemas de computação, os fatores que comprometem essa segurança, os meios para alcançá-la e as suas medidas usuais. Fugindo um pouco às classificações comumente apresentadas, que consideram falhas como eventos não intencionais, programas daninhos, como vírus e vermes, são introduzidos como um dos fatores que prejudicam a segurança de sistemas de computação.

A seguir, são discutidas as técnicas de detecção, confinamento e avaliação de danos, bem como mecanismos de recuperação de erros e tratamento de falhas. As várias formas de redundância, essenciais tanto da detecção de erros como no tratamento de falhas, são enfatizadas. Ao final, são apresentados dois aspectos de ordem prática: o primeiro constitui-se em uma análise de números obtidos referentes a avaliação de confiabilidade de um caso-exemplo; e o segundo trata-se do relato de uma experiência em diversidade de programação realizada em nosso grupo de pesquisa no CPGCC.

2. Conceitos básicos e terminologia

Os conceitos apresentados neste capítulo constituem-se atualmente em conceitos clássicos na área de tolerância a falhas e foram extraídos principalmente das publicações de Anderson e Lee [AnLe81], de Laprie [Lapr85] e de Randell [Rand75].

Define-se um sistema como um conjunto de componentes, juntamente com seus inter-relacionamentos, projetado para prover um serviço específico. Esta definição é naturalmente recursiva: os componentes do sistema podem ser considerados como sistemas menores. Como um caso limite, um componente (ou sistema) pode ser considerado atômico quando a estrutura interna deste componente é desprovida de interesse e pode ser desprezada. Denomina-se algoritmo ao inter-relacionamento existente entre os componentes do sistema. Não existe necessidade de que um componente proporcione serviço a um único sistema; o componente pode pertencer a diversos sistemas distintos. Entretanto o algoritmo deve ser específico a cada sistema.

A confiabilidade de um sistema é uma medida do sucesso com o qual o sistema atende sua especificação. Quando o comportamento do sistema se desvia de sua especificação, ocorre um defeito. Diversas medidas formais relacionadas à confiabilidade de um sistema podem ser baseadas na incidência real ou estimada de defeitos, bem como em suas conseqüências. Exemplos destas medidas são: tempo médio entre defeitos (MTBF, do inglês, Mean Time Between Failures), tempo médio de reparo (MTTR, do inglês, Mean Time To Repair) e disponibilidade. Estes parâmetros de avaliação serão abordados de forma mais detalhada na seção 2.4.

Paralelamente à definição abrangente de defeito aparecem as definições de erro e falha. Um estado interno de um sistema é dito errôneo se, diante da funci-

onalidade normal especificada para utilização do sistema, existem circunstâncias nas quais o processamento posterior, a partir do estado em questão, leva a um defeito. O termo erro é utilizado para designar a parte do estado que está incorreta. Uma falha é a causa mecânica ou algorítmica do erro.

A terminologia "defeito, erro e falha", aqui utilizada, tem correspondência com a corrente predominante no inglês "failure, error e fault", respectivamente. As definições dos termos empregados na área de tolerância a falhas têm sido discutidas no âmbito de um sub-comitê designado em comum acordo entre IFIP e IEEE e estão sintetizados no artigo de Laprie apresentado no FTCS-15 [Lapr85]. O significado destes termos e os exemplos correspondentes serão detalhados na próxima seção.

2.1 Segurança em sistemas de computação

Segurança de funcionamento de um sistema computacional é a propriedade que permite depositar confiança justificada no serviço que ele fornece. O serviço que este sistema produz é correspondente ao comportamento observado por seu(s) usuário(s); o usuário de um sistema é um outro sistema que interage com ele.

Um defeito ou mau funcionamento do sistema sobrevém a partir do momento em que o serviço fornecido se desvia das condições estabelecidas na especificação. Um erro é um estado – com relação ao processo de tratamento – suscetível de conduzir a um defeito. Uma falha é a causa suposta ou reconhecida de um erro. Esta relação causa-conseqüência é ilustrada na figura 1.

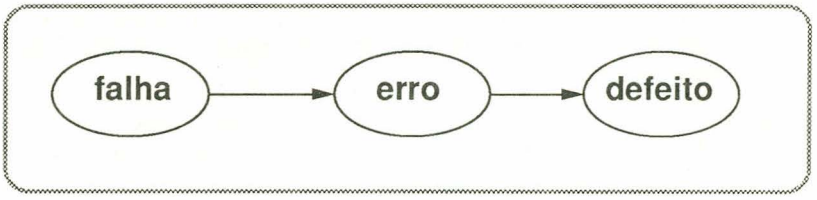

Figura 1: Entraves à segurança de funcionamento

Um erro é, portanto, a manifestação de uma falha no sistema, enquanto que um defeito é a manifestação de um erro no serviço fornecido pelo sistema. A diferença básica entre uma falha e um erro é com relação a estrutura do sistema; um erro em um sistema é causado por uma falha em um componente ou no projeto do sistema. Já a distinção entre erro e defeito não reflete apenas a estrutura do sistema. Ao invés disso, a diferença é entre uma condição (ou estado) e um evento. Um sistema contém um erro quando seu estado é errôneo, enquanto um defeito do sistema é o evento do sistema não produzir o comportamento especificado.

2.2 Fatores que comprometem a segurança

Falhas, erros e defeitos constituem entraves à segurança de funcionamento. São circunstâncias indesejáveis mas não imprevistas que causam ou resultam na insegurança de funcionamento de sistemas. As falhas podem ser de diversos tipos; antes, porém, do exame destes aspectos, serão apresentados alguns exemplos ilustrativos referentes aos conceitos de falhas, erros e defeitos.

2.2.1 Falha, erro e defeito na prática

Em geral, a definição de falha é intuitivamente associada a componentes físicos. Esta, entretanto, é uma noção restritiva. A fim de facilitar a interpretação de aspectos conceituais da "cadeia" falha-erro-defeito para a prática, serão transcritos a seguir alguns exemplos apresentados por Laprie [Lapr85].

Exemplo 1 : O engano de um programador é uma falha; a conseqüência é um erro latente no software escrito (uma instrução errada ou um segmento de dados); sob ativação, causada pela utilização combinada do módulo onde reside o erro com um padrão de entrada adequado que ativa o segmento de programa ou de dados errado, o erro se torna efetivo. Quando este erro produz dados errados que afetam, em valores ou tempos, o serviço fornecido, ocorre um defeito.

Exemplo 2 : Um curto-circuito que ocorre em um circuito integrado é uma falha; a conseqüência é um erro, e pode ser modelado através de grampeamento em um dos valores booleanos, pela modificação da função executada pelo circuito, ou outro modelo. O erro que pode permanecer latente até que algum evento cause sua ativação, continuando o processo como no exemplo anterior.

Exemplo 3 : Uma perturbação eletromagnética pode se constituir em uma falha; quando age sobre as entradas de uma memória, poderá criar um erro se a memória estiver habilitada para escrita. O erro permanecerá latente até que a(s) palavra(s) incorreta(s) de memória for(em) lida(s), propagando-se até a ocorrência do defeito como no exemplo 1.

Exemplo 4 : Uma interação inadequada homem-máquina, realizada de forma intencional ou inadvertidamente, por um operador durante a operação do sistema é uma falha; os dados alterados resultantes constituem-se em erro, e assim por diante.

2.2.2 Tipos e origem de falhas

Um erro pode resultar da ativação de uma falha interna ao sistema, que anteriormente encontrava-se latente, ou da ocorrência de uma falha externa. Uma falha interna pode resultar de um defeito físico de um componente físico (visto

então como um sistema), referida como falha física, ou de uma falha de concepção afetando o software ou o hardware (o termo "concepção" é empregado com sentido lato, podendo estas falhas ocorrerem desde o início do projeto propriamente dito até a realização de atividades de manutenção posteriores). Uma falha externa pode referir-se ao ambiente físico do sistema ou a uma falha humana de interação, existindo assim um erro de entrada (i.e., a seleção de um ponto de entrada fora do domínio de entrada especificado). Segundo Avizienis [Aviz78], as falhas podem ser enquadradas em classes, a saber: falhas físicas e falhas humanas. Enquadram-se no primeiro grupo os fenômenos físicos adversos internos (desordens físico-químicas) ou externos (perturbações do ambiente). As falhas humanas são imperfeições do tipo falhas de projeto ou falhas de interação. A origem e as características de cada um dos diferentes tipos de falhas serão examinadas mais detalhadamente a seguir: Um esquema correspondente aos diversos tipos de falhas é mostrado na figura 2.

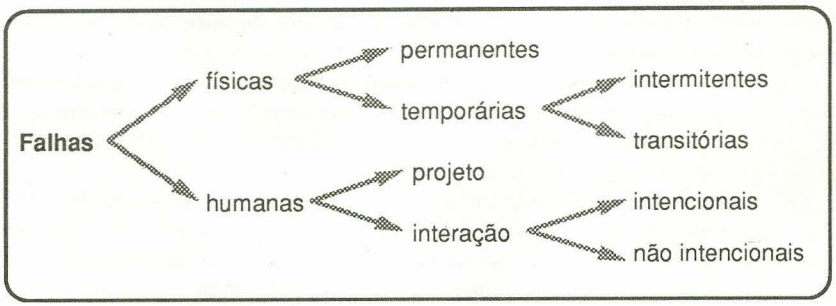

Figura 2: Representação esquemática dos tipos de falhas

Falhas físicas:

As falhas físicas são ditas permanentes ou temporárias.

Falhas permanentes ocorrem em componentes afetados por ruptura mecânica ou algum outro fenômeno de desgaste tal como uma migração metálica, interdifusão ("purple plague"), defeitos no óxido, etc.. As falhas permanentes são, em geral, localizáveis. Sua ocorrência pode ser minimizada por um processamento cuidadoso e por testes de inspeção visual iniciais. Jamais podem ser eliminadas completamente.

As falhas temporárias podem ser transitórias ou intermitentes. Falhas transitórias são causadas por perturbações de sinal induzidas externamente causadas, em geral, por radiações eletromagnéticas ou de partículas alfa, ou flutuações da fonte de alimentação. Podem ser controladas por cuidados especiais

no isolamento e desacoplamento durante o projeto do equipamento. As falhas intermitentes ocorrem normalmente quando um componente está em vias de desenvolver uma falha permanente. A degradação de algum parâmetro leva à diminuição da diferença entre os níveis de sinal, as margens de ruído diminuem e valores falsos de sinal aparecem de forma aparentemente indeterminada.

O controle das falhas permanentes minimiza a ocorrência de falhas intermitentes mas, como não é possível assegurar-se a não ocorrência de falhas permanentes, as falhas intermitentes são potencialmente possíveis.

Não há dados precisos sobre a freqüência relativa de falhas permanentes e temporárias; entretanto parece haver consenso sobre uma avaliação prática de que as falhas permanentes seriam aproximadamente dez vezes menos freqüentes que as temporárias [McCl82].

Falhas humanas:

Conforme exposto anteriormente, as falhas humanas têm, como origem, procedimentos adotados em fases associadas ao projeto do sistema ou durante a interação com o sistema.

As falhas de projeto podem ser cometidas a) durante o projeto inicial do sistema, em qualquer de suas etapas, desde a especificação até a implementação, ou em suas modificações subseqüentes, ou b) durante o estabelecimento de procedimentos de operação ou manutenção.

As falhas de interação constituem-se em violações, inadvertidas ou deliberadas, dos procedimentos de operação e manutenção. Anderson & Lee [AnLe81] discordam desta subdivisão na taxonomia de falhas humanas, alegando que uma interação com o sistema, independente de ser permitida ou não, ou de ser correta ou incorreta, não pode ser causa de um defeito. Ou a especificação do sistema deve determinar explicitamente a resposta à interação, e neste caso qualquer defeito deve ser atribuído a uma falha interna; ou, se a especificação não determina resposta, não há defeito uma vez que o sistema tem liberdade de produzir uma resposta arbitrária.

As falhas de projeto necessitam de técnicas de tolerância a falhas mais potentes do que as necessárias para lidar com as falhas de componentes. As falhas de projeto são imprevisíveis, sua manifestação é inesperada e geram erros dificilmente preveníveis. As falhas de componentes, pelo contrário, podem ser freqüentemente modeladas; sua manifestação é, de certo modo, esperada, e produzem erros que podem ser prevenidos. Há maior experiência e documentação a respeito das falhas físicas e respectivas conseqüências, o que facilita o modelamento das situações.

As falhas humanas intencionais, embora possam ser tratadas em princípio como as demais falhas de interação, despertam interesse para comentários adicionais. Pequenos descuidos dos projetistas dos sistemas, ou as próprias características destes sistemas, têm servido a terceiros como suporte para intrusão em sistemas alheios e realização de operações indevidas, por exemplo.

2.2.3 Falhas intencionais

Todas as ações que visem explicitamente causar danos a um sistema de computação podem ser classificadas como falhas intencionais. O termo dano é aqui utilizado no seu sentido mais amplo, englobando dano físico ao hardware, dano lógico ao software e inclusive dano ao desempenho do computador, como o consumo de tempo de processador ou espaço em memória, por exemplo.

Existem três grupos principais de pessoas cuja atividade principal é causar dano a um sistema:

hackers : este nome caracteriza principalmente pessoas que se infiltram ilegalmente em redes de computadores, procurando acessar, manipular ou mesmo destruir dados e programas. É interessante notar que quando este termo surgiu, se referia a pessoas que se dedicavam quase que fanaticamente à informática e à programação, chegando ao ponto de se isolarem da sociedade. Este segundo significado é mais utilizado na Europa, e está rapidamente caindo em desuso.

crackers : este nome é aplicado a pessoas que se ocupam em "quebrar" esquemas de proteção contra cópia ilegal. Um dos maiores incentivadores das cópias piratas, os crackers possuem a mesma obsessão doentia dos hackers.

sabotadores : normalmente causada por programadores de uma empresa, a sabotagem já nasce junto com o programa fonte. Um programador pode, nestes casos, incluir rotinas em benefício próprio (como desvios para sua própria conta bancária) ou rotinas de represália caso venha a ser demitido. Nesta categoria se enquadram a maioria dos "crimes por computador".

Nos últimos cinco anos, entretanto, tem-se observado uma crescente proliferação de programas daninhos. De acordo com o seu modo de operação, estes programas podem ser divididos em diversas categorias [Webe89]:

programa traidor : aquele que é formado unicamente por uma rotina de ataque; sua ação danosa ao sistema seria realizada toda vez que o programa fosse ativado.

cavalo de Tróia : um programa caracterizado por apresentar um módulo normal e outro de ataque. Podem portanto executar tarefas úteis, mas ao mesmo tempo também possuem a capacidade de realizar dano. Existem dois tipos correntes de cavalos de Tróia: aqueles que usam seu módulo normal apenas para mascarar o ataque daninho; e aqueles com funcionamento perfeitamente normal mas cujo módulo de ataque coleta informações sobre o sistema hospedeiro. Este segundo tipo de cavalos de Tróia são mais comuns em redes de computadores e computadores de grande porte, onde sua principal função é a coleta de "passwords" ou espionagem e coleta de informações.

Note-se que enquanto estes programas aparentam ser aplicativos legítimos, mas causam destruição de dados quando disparados, eles são incapazes de auto-reprodução e não infectam outros programas. Os danos causados cessam assim que eles são retirados do sistema ou não mais ativados.

verme : um programa auto-contido, capaz de se auto-propagar. É constituído unicamente de um módulo de reprodução e não necessita de um programa hospedeiro. Vermes são comuns em redes de computadores, onde se propagam de nó para nó da rede. O caso mais famoso é o verme que atacou em novembro de 1988 a rede Internet. Em questão de horas o verme se replicou para cerca de seis mil computadores, congestionando completamente a rede. Nenhum arquivo foi destruído, mas calcula-se que o dano em termos de horas de processamento atinja a casa dos milhões de dólares.

vírus : um programa ou um fragmento de programa inserido em outro programa. O programa resultante é então constituído de três módulos (reprodução e ataque, representados pelo vírus, e normal, representado pelo programa hospedeiro). Note-se que um vírus obtém o controle do computador quando o seu hospedeiro é executado. Como o sistema normalmente não distingue entre vírus e hospedeiro, isto faz com que o vírus ganhe todos os benefícios e privilégios que o seu hospedeiro possui.

cupins eletrônicos : uma variação dos vírus, estes são programas extremamente curtos, que se propagam na memória assim como cupins na madeira, deixando atrás de si somente bytes inúteis. Eles também não possuem capacidade de infectar outros programas, e sua ação sobre um sistema é imediata mas restrita à memória.

Diversas fontes consideram todos os tipos de programas com módulos de ataque e/ou reprodução como vírus. Num sentido amplo todos seriam capazes de se reproduzirem e serem ativados, embora muitos necessitem da ajuda direta do usuário para isto. Neste trabalho adota-se a definição mais restrita de que um vírus necessita ter a capacidade de auto-propagação.

O termo vírus remonta ao trabalho de doutorado do informata americano Fred Cohen, publicado em 1983, e que tratava sobre segurança de sistemas. Segundo Cohen, "define-se um vírus como sendo um programa que pode infectar outros programas, modificando-os de maneira a permitir a inclusão de uma cópia sua. Com esta capacidade de infecção, um vírus pode se propagar em um sistema ou rede, utilizando as autorizações e privilégios dos seus usuários em proveito próprio".

Esta definição enfoca em primeira linha a capacidade de cópia que um vírus possui. É através deste mecanismo de propagação que ele se espalha, eventualmente contaminando todos os programas executáveis ou áreas de armazenamento existentes em um dado sistema. A segunda parte da definição simplesmente afirma que um vírus atua utilizando os próprios recursos do sistema; a

função que ele realiza quando entra em ação pode assim variar enormemente de um vírus para outro.

Um vírus é ativado quando um programa infectado é disparado, ou quando o sistema é inicializado a partir de um meio magnético contaminado. Um vez ativo, o vírus passa a executar duas rotinas distintas:

- Uma rotina de propagação (ou reprodução), que procura programas não contaminados e, ao achar um, copia nele o vírus, infectando-o.
- Uma rotina de atuação (ou ataque), que realiza uma função qualquer, mas sempre prejudicial ao sistema. Essa função é ativada quando certa condição é satisfeita.

A rotina de atuação pode tomar as mais variadas formas, e é em função desta rotina que os vírus podem ser qualificados em benignos e malignos:

benigno : um vírus deste tipo não causa maiores danos ao computador ou ao programa hospedeiro. Nenhuma informação é destruída. A rotina de atuação de um vírus benigno normalmente se caracteriza pela emissão de mensagens, avisos sonoros ou outros distúrbios desta espécie. O único dano causado é ao desempenho, consumindo espaço de memória ou tempo de processador.

maligno : um vírus maligno quase sempre destrói informação quando entra em ação. Nos casos mais brandos somente o programa hospedeiro é removido do disco; ataques mais sérios podem ser efetivados contra arquivos de dados, textos ou planilhas eletrônicas, fazendo com que o usuário perca dados valiosos. Um vírus maligno sempre visa causar dano à informação ou dano físico.

A rotina de atuação dos vírus pode ser ativada (ou "gatilhada") pelas mais diversas condições. Estas podem ser temporais (baseadas em datas ou tempo decorrido), ou dependentes de eventos físicos (como a área ocupada em um disco magnético) ou lógicos (como o número de vezes que um programa é executado). Diversas condições distintas podem ser combinadas para formar uma condição mais complexa. É importante salientar que, ao contrário da rotina de propagação, a rotina de atuação normalmente não é ativada imediatamente. Isto caracteriza o tempo de incubação do vírus, e visa possibilitar um grande número de infecções.

Vírus, cavalos de Tróia e cupins eletrônicos são ativados desavisadamente pelos usuários de um sistema de computação, que se tornam vítimas da ação desses programas. Os vermes eletrônicos, tais como descritos anteriormente, estão normalmente associados a ação de hackers em redes de computadores. O surgimento e a proliferação de vermes em computadores americanos deu início a uma discussão internacional sobre a segurança de sistemas interconectados.

Muitos dos danos reportados na imprensa nos últimos tempos não se relacionam com vírus, mas são devidos a hackers ou sabotadores. Por outro lado, vírus

e vermes se proliferam com rapidez em ambientes onde exista troca intensiva de programas, como redes de computadores, programas de domínio público (que podem ser legalmente copiados e distribuídos), e cópias piratas. É justamente dessas últimas que os vírus retiram seu grande potencial de propagação: um sistema que não utilize cópias piratas tem baixas possibilidades de ser atingido. Nestes casos a infecção praticamente só ocorre se os programas legalmente adquiridos forem previamente infectados no produtor ou no distribuidor (casos já ocorridos nos Estados Unidos e Europa). Em um ambiente "promíscuo", como clubes de usuários, universidades, centros de pesquisa e grandes empresas, os vírus obtêm as condições ideais de propagação.

É interessante notar-se que os programas daninhos são em parte possíveis devido às poucas características de segurança dos sistemas de computação atuais. Nos computadores pessoais e domésticos estas características foram praticamente eliminadas por opção de projeto. Nos computadores e grande porte ou multi-usuários, entretanto, as medidas de segurança atualmente existentes contra intrusão provaram ser insuficientes para evitar ou mesmo restringir os danos intencionais.

2.3 Meios de obtenção de segurança

Dotar sistemas computacionais da propriedade de segurança de funcionamento, ou seja, capacitá-los a executar um serviço de acordo com a especificação, exige o emprego de procedimentos e/ou métodos a nível de realização, que podem ser classificados em dois grandes grupos:

- Técnicas de prevenção de falhas: visam impedir, por construção, a ocorrência ou introdução de falhas;

- Técnicas de tolerância a falhas: têm por objetivo o fornecimento, por meio de redundância, de um serviço que atenda às especificações a despeito da ocorrência de falhas.

A validação ou análise da segurança de funcionamento obtida, que corresponde à perspectiva de confiança na aptidão do sistema em executar um serviço de acordo com a especificação, é executada através de técnicas de remoção de falhas e de previsão de falhas (ou erros). A remoção de erros está relacionada a como minimizar, por verificação, a presença de falhas. A previsão de erros diz respeito a como estimar, por avaliação, a presença, a criação, e as conseqüências de falhas.

2.3.1 Prevenção de falhas

Inclui metodologias de projeto e a seleção de técnicas e tecnologias que visam evitar a introdução de falhas durante o projeto e a construção de um sistema.

Para o hardware, são empregadas como técnicas de prevenção: uso de componentes confiáveis e tecnologias de interconexão testadas, metodologias para lidar com complexidades de projeto, bem como métodos formais para verificação dos projetos lógicos. Do ponto de vista do software, o emprego de prevenção de falhas inclui a definição de necessidades, uso de especificações precisas, de metodologias de projeto, de técnicas de programação estruturada, técnicas de teste e documentação, o desenvolvimento de linguagens de alto nível e o gerenciamento de software com o objetivo de prevenir enganos.

Considerando-se por objetivo a eliminação de falhas de projeto, a aplicação de técnicas de prevenção de falhas ao hardware tem atingido certo sucesso, mas isto não é inteiramente verdadeiro com relação ao software. A causa provável é a alta complexidade do software, com um número altíssimo de estados possíveis, impossibilitando o teste exaustivo (na etapa de remoção). Como conseqüência, é consenso o fato de que as falhas no software são a maior causa de defeitos em sistemas computacionais.

Para [AnLe81], a prevenção de falhas apresenta dois aspectos interligados: o primeiro corresponde aos procedimentos que visam evitar falhas em tempo de projeto, recém citados, e recebe a denominação de evitação de falhas e o segundo corresponde à remoção de falhas. Mesmo com o uso de técnicas de evitação, as falhas estão usualmente presentes em sistemas implementados devido, por exemplo, à indisponibilidade ou ao custo de componentes de hardware livres de falhas, ou devido à enorme complexidade inerente a sistemas computacionais. A remoção de falhas existe, nestes casos, para testar a implementação dos sistemas e remover quaisquer falhas que puderem ser assim detectadas. O teste exaustivo de um sistema, visando revelar componentes de hardware defeituosos ou falhas de projeto tanto no hardware como no software, é um exemplo de técnica de remoção.

2.3.2 Tolerância a falhas

A aplicação de técnicas de prevenção de falhas aliadas a testes em final de fabricação, não tem demonstrado suficiência quando o objetivo é atingir alta confiabilidade. Um motivo é que os componentes físicos deterioram-se ao longo do tempo e tornam-se faltosos. Assim a tolerância a falhas torna-se necessária para proteger os sistemas, pelo menos contra falhas em componentes de hardware. Além disto, pelas razões já expostas no ítem anterior, também mostra-se útil o uso de técnicas de tolerância para proteger os sistemas contra falhas de software.

A proposta do uso de tolerância a falhas por meio de redundância física data de 1956, em um trabalho de Von Newmann [VonN56], tendo por objetivo a obtenção de um sistema altamente confiável, construído a partir de componentes não confiáveis. Desde aquela época até hoje, foram obtidos grandes progressos no desenvolvimento nas tecnologias de semicondutores, fazendo com que os componentes se tornassem mais confiáveis. Entretanto, o uso de tolerância a

falhas ainda é imprescindível para vários tipos de aplicações. As técnicas usadas são objeto do capítulo 3.

A previsão de falhas será utilizada aliada tanto às técnicas de prevenção como as de tolerância, para elaboração das hipóteses de falhas e avaliação de parâmetros de cobertura.

2.4 Medidas de segurança

A vida de um sistema é vista pelos seus usuários como uma alternância entre dois estados do serviço realizado com relação ao serviço especificado: adequado ou inadequado. A transição do estado adequado ao inadequado configura um defeito. A quantificação desta alternância conduz às duas medidas principais de segurança de funcionamento:

- a confiabilidade, que corresponde à medida do fornecimento contínuo de um serviço adequado, i.e., medida do tempo que transcorre até a manifestação da falha ocorrida sob forma de um defeito;
- a disponibilidade, que corresponde à medida do fornecimento de um serviço adequado com relação à alternância "serviço adequado / serviço inadequado".

Do ponto de vista do projetista, a confiabilidade é um dos critérios fundamentais a ser considerado no desenvolvimento de um sistema complexo. As conseqüências sobre os custos de uma empresa por fornecer equipamentos não confiáveis podem ser bastante severas; elas podem traduzir-se por aumentos nos custos de manutenção, de produção, insatisfação do cliente, entre outros.

Por outro lado, é necessário administrar cooperativamente critérios de desempenho do sistema, critérios de confiabilidade e critérios de custo, os quais, se considerados independentemente, conduziriam a opções diferenciadas. O desempenho relaciona-se aos tempos de resposta, à capacidade das unidades componentes do sistema, às propriedades e facilidades disponíveis. A confiabilidade está relacionada à probabilidade de que o sistema esteja operando corretamente em um dado instante, ao tempo médio entre falhas, aos tempos de manutenção e reparo. O custo a nível de concepção engloba aqueles envolvidos para a concepção propriamente dita bem como a implementação, custo de manutenção, de estoque, etc.

2.4.1 Confiabilidade

A confiabilidade é definida como sendo a capacidade que um sistema tem em responder a uma dada especificação dentro de condições definidas e durante um certo tempo de funcionamento [SoMa87]. Os três parâmetros envolvidos, grifados na definição, precisam ser caracterizados, a fim de que se atinjam os

requisitos de confiabilidade desejados. Isto envolve a caracterização quantitativa das especificações e do tempo de funcionamento (duração da missão) e a definição das condições de funcionamento.

Caracterização quantitativa das especificações:

Sendo a falha um fenômeno aleatório, utiliza-se o conceito de variáveis aleatórias para descrição das grandezas relacionadas; a quantificação destas grandezas pode ser pelo uso de probabilidades. Assim, sendo a confiabilidade expressa pela probabilidade de que o sistema funcione corretamente durante o intervalo $[0,t]$ e X a variável que representa o tempo de funcionamento sem falhas de um sistema,

Confiabilidade = $R(t) = P\ \{\ X > t\ \}$

Portanto, a confiabilidade é a função de sobrevivência da variável aleatória X.

Tempo de funcionamento (ou duração da missão):

A especificação do tempo de funcionamento ou duração da missão dependerá das exigências da aplicação. Em termos gerais, este intervalo corresponde ao tempo em que o sistema deve funcionar sem que ocorram defeitos, ou em que o sistema deva funcionar ininterruptamente, sem manutenções.

Exemplos clássicos são: satélites, onde a duração da missão corresponde a toda vida útil do equipamento; aviões, onde a duração da missão representa o tempo de vôo que é em média dez horas. Segundo Wensley [Wens78], o requisito de confiabilidade para o sistema de controle de um avião é expresso por uma probabilidade de defeito menor do que 10^{-9} por hora em um vôo de dez horas de duração.

Em outros casos, nos quais o tempo de vida útil é bastante longo e a aplicação não exige requisitos de confiabilidade tão rígidos, a duração da missão é especificada em termos de indisponibilidade, ou seja, pelo somatório dos tempos em que o sistema pode permanecer fora de operação. É o caso de centrais telefônicas, cujo tempo de vida útil é de 20 a 40 anos. Uma medida de indisponibilidade admitida para uma central poderia ser expressa pelo período de duas horas durante o seu tempo de vida útil [SoMa87]. A exigência de operação ininterrupta neste caso implicaria em custo excessivamente elevado para o sistema. Outras aplicações da mesma classe são centros de processamento de dados comerciais nos quais a ênfase de projeto visa, cada vez mais, permitir reparos fáceis e rápidos.

Condições de funcionamento:

Neste nível, define-se condições de funcionamento dos componentes e do sistema, determinando assim exigências a serem satisfeitas na escolha destes. Exemplos

de condições referentes aos componentes são: especificações ambientais para instalação do equipamento, especificações elétricas, nível de qualidade do produto. Estes parâmetros influirão sobre a escolha dos tipos e famílias de componentes e sobre as taxas de falhas admitidas a nível dos componentes e dos módulos envolvidos na construção do sistema. A nível do sistema estas condições especificam se o sistema é reparável ou não, o tipo de redundância a ser utilizado, e os tipos de falhas a serem considerados. A definição dessas características dependem da aplicação prevista conforme será visto no próximo capítulo.

A confiabilidade pode ser melhorada pelo uso de técnicas de prevenção de falhas, a nível de componentes, e pelo uso de técnicas de tolerância a falhas, a nível do sistema. As técnicas de tolerância a falhas serão examinadas na seção 3.

2.4.2 Disponibilidade

Considere-se a operação de um sistema onde há alternância entre os períodos de funcionamento normal, interrompidos pela ocorrência de um defeito, e períodos de reparo, utilizados para detecção das falhas e substituição de componentes ou módulos. Associando-se a variável x_i aos períodos de funcionamento normal (sem defeitos) e a variável r_j aos períodos de reparo, teremos a operação do sistema representada pela seqüência $x_1, r_1, x_2, r_2, \ldots$ a qual corresponde a um processo aleatório. A disponibilidade será expressa pela probabilidade de que o sistema esteja funcionando no instante t.

Uma medida prática bastante utilizada para avaliar a segurança de um sistema é o MTBF (Mean Time Between Failures), que mede o intervalo médio de ocorrência de defeitos. É obtida pela realização de medidas durante o funcionamento do sistema ou pode ser estimada. Esta medida associa-se fortemente ao parâmetro disponibilidade, uma vez que mede os intervalos x_i.

3. Técnicas de tolerância a falhas

3.1 Razões e objetivos

Em diversas situações, apenas o uso de prevenção a falhas como técnica de controle de falhas no sistema não é suficiente. Dentre estas situações pode-se citar os casos onde a freqüência e duração dos tempos de reparo são inaceitáveis ou onde o sistema encontra-se inacessível para atividades de manutenção e reparo manual. Um tratamento alternativo à prevenção de falhas nesta situação é baseado em técnicas de tolerância a falhas.

Um sistema pode ser projetado para ser tolerante a falhas pela incorporação de componentes adicionais e uso de algoritmos especiais, os quais tentam limitar os efeitos resultantes de falhas sobre as saídas do sistema. A cobertura, ou grau de tolerância a falhas, depende do sucesso na detecção e identificação de estados errôneos correspondentes a falhas e do sucesso no tratamento destes estados e de suas causas. A definição dos níveis requeridos de cobertura dependem do tipo

de resposta aceitável, que é uma função da aplicação. As diversas possibilidades de resposta do sistema a falhas são:

- erro na saída;
- falha mascarada: as saídas estão corretas, embora tenha ocorrido falha no sistema;
- seguro a falhas (fault secure): as saídas estão corretas ou há indicação de erro;
- livre de falhas (fail safe): as saídas estão corretas ou em um valor seguro, geralmente indicativo da condição.

Em sistemas pequenos empregados em aplicações não críticas tais como jogos, editores de texto, ou similares, admite-se a ocorrência de erros na saída, isto é, admite-se que o sistema não seja tolerante a falhas. Aliás, em sistemas deste tipo, em geral, não se justifica o custo adicional que seria acarretado pelo uso de técnicas de tolerância a falhas.

No extremo oposto, em termos de possibilidades, estão sistemas onde qualquer saída errônea não pode ser admitida. Usualmente, este é o caso das aplicações críticas tempo-real, cujas saídas causam ações diretas (e imediatas) como em plantas de controle de usinas de energia, em controle de aeronaves, sistemas médicos, etc. Neste caso, as falhas devem ser mascaradas.

Em algumas aplicações, é suficiente ter uma indicação de saída que pode estar incorreta. Estas são situações em que se deve prevenir a utilização de resultados incorretos, mas onde é possível recalcular resultados suspeitos, como em aplicações bancárias e em telefonia. Este tipo de resposta do sistema é denominada segura a falhas. A indicação de erro pode ser usada para definir parada de operação diante da ocorrência de falha (referido na literatura como fail-stop).

O caso restante corresponde aos sistemas livres de falhas, nos quais, se a saída é incorreta, ela estará sempre em um estado seguro. É usado em aplicações nas quais um tipo de resposta anormal é aceitável, como em controle de sinaleiras (com o laranja intermitente como estado seguro), em caixa automático (que susta as operações), ou mesmo em trens metroviários (onde o sistema de freios pode ser acionado em caso de perigo ou perda de controle).

3.2 Mecanismos de controle de falhas

Os sistemas tolerantes a falhas diferem com respeito ao seu modo de operação na presença de falhas e com relação aos tipos de falhas que devem ser toleradas: em alguns casos, o objetivo é continuar a proporcionar o desempenho e capacidade funcional total do sistema; noutros, o desempenho degradado ou a capacidade funcional reduzida são aceitáveis até a remoção da falha. Em ambos casos, os

mecanismos de controle de falhas são limitados pelo número de defeitos que podem ser tolerados.

Os métodos de tolerância a falhas são realizados basicamente através de técnicas de tratamento de erros e técnicas de tratamento de falhas [LCGP89]. As técnicas de tratamento de erros destinam-se a eliminá-los se possível antes que ocorra um defeito. O tratamento de falhas destina-se a evitar que falhas sejam reativadas. De forma geral, procedimentos de manutenção ou reparo estão associados ao tratamento de falhas.

Segundo Anderson & Lee [AnLe81], as atividades relacionadas à tolerância a falhas podem ser subdivididas em quatro fases: detecção de erros, confinamento e avaliação de danos, recuperação de erros e tratamento de falhas. Quando consideradas em conjunto, constituem-se nas etapas através das quais a tolerância a falhas é implementada, evitando que falhas conduzam o sistema a uma situação de funcionamento inadequado. Estas fases serão conceituadas e exemplificadas a seguir.

3.2.1 Detecção de erros

Consiste no reconhecimento da existência de um estado errôneo, o qual é efeito ou resultante da manifestação de uma falha. Este estado errôneo consiste em um estado diverso do previsto na especificação inicial do sistema. Assim, a detecção é o ponto de partida para as técnicas de tolerância a falhas.

Esta fase de detecção de erros pode ser processada em diversos momentos, podendo ser concorrente com a operação normal do sistema ou processar-se até a avaliação dos resultados. Estes testes podem ser complementares, assim como a opção entre mecanismos de software ou hardware, que podem ser usados simultaneamente. De modo geral, os diferentes tipos de teste têm objetivos diversos e seu uso concomitante resulta no somatório das vantagens de cada um.

Os mecanismos de detecção de erros mais usados são listados a seguir.

- Uso de cópias do sistema (replication checks): esta técnica envolve algum tipo de replicação na atividade do sistema a ser testado. A replicação pode ser vista como uma implementação alternativa do sistema, através da qual a atividade do sistema pode ser testada sob o aspecto de consistência. O tipo de replicação a ser definida (de componentes, software, ou temporal) depende das hipóteses de falhas consideradas, dos fatores econômicos envolvidos e do grau de segurança desejado.

- Testes de limite de tempo (timing checks): avaliam possíveis existências de falhas relacionadas a restrições de temporização de circuitos, ou seja, verificam o cumprimento de especificações temporais dos sistemas. Estes testes podem revelar a existência de falhas, mas não são capazes de assegurar sua inexistência. Por isto, é recomendado o uso deste tipo de teste associado a outro que proporcione a indicação complementar. A forma

de implementação mais usada para estes testes é através dos chamados cão-de-guarda (watchdog timers).

- Testes reversos (reversal checks): tenta reconstituir as entradas do sistema a partir das saídas geradas; a seguir, compara as entradas calculadas com as entradas reais, a fim de verificar se existe erro. Evidentemente, este tipo de teste só pode ser aplicado quando a relação entre entradas e saídas é biunívoca. Uma variante deste teste pode ser usada em sistemas onde há uma relação fixa e verificável entre as entradas e saídas. É o caso de funções matemáticas onde pode-se aplicar a função inversa, recalcular a função por substituição de valores, etc.

- Codificação (coding checks): baseia-se no uso de redundância relacionada à informação processada no sistema. Os dados redundantes mantêm uma relação fixa com os dados não redundantes, e a ocorrência de falhas causa a alteração da informação, situando-a fora do espaço de código.

- Testes de razoabilidade (resonableness checks): estes testes verificam se o estado de diversos objetos abstratos no sistema é razoável, com base nos objetivos e utilização visados para estes objetos, conforme especificado pelo projetista do sistema. Envolve conhecimento da estrutura interna do sistema. Exemplos típicos são avaliação de objetos que deveriam estar situados em determinados limites ou intervalos numéricos assim como a verificação de compatibilidade entre tipos.

- Testes estruturais (structural checks): são particularmente aplicáveis a estruturas de dados complexas, que consistem de um conjunto de elementos ligados por ponteiros, como listas, filas e árvores. Estes testes verificam a integridade estrutural notadamente no que se refere a sua consistência. A redundância usada nestas estruturas assume uma de três formas principais: contagem do número de elementos que deveriam estar em uma estrutura; uso de ponteiros redundantes incorporados na estrutura; ou os elementos integrados a uma estrutura podem conter informações de estados e/ou tipo.

- Programas de diagnóstico (diagnostics checks): têm por objetivo o exercitamento dos componentes com conjuntos de valores de entrada, para os quais conhece-se as saídas correspondentes. As saídas obtidas a partir dos componentes são comparadas com os valores esperados: a detecção de divergências indica a existência de falhas. Diferem dos procedimentos anteriormente citados pelo fato de terem em vista a análise de componentes específicos ou isolados e não a análise do sistema como um todo.

3.2.2 Confinamento e avaliação de danos

O tempo que transcorre desde a ocorrência de uma falha até a sua manifestação através de um erro (detectável) é chamado tempo de latência ou latência de erro.

Este tempo é explicado pelo fato de que apenas a excitação da área afetada com determinado valor ou combinação de valores lógicos resultará em erro. Por outro lado, dependendo do nível onde se dá esta detecção, é possível que informações inválidas já estejam espalhadas pelo sistema, e o estado geral desse seja passível de suspeita. Assim, antes de efetuar os procedimentos previstos para o tratamento do erro detectado, faz-se necessário verificar a extensão de danos provocados sobre o restante do sistema.

O confinamento de falhas e a extensão dos danos resultantes dependem de decisões tomadas a nível de projeto. As estratégias relacionadas dependem da estrutura que o projetista incorporou ao sistema operacional, uma vez que esta estrutura lhe permitirá identificar as atividades possíveis do sistema que podem ter seguido uma transação incorreta, bem como os principais fluxos de informação.

Assim, os mecanismos importantes para avaliação dos danos são os que reforçam a estrutura em tempo de execução, impondo restrições no fluxo de informações do sistema. Estas restrições visam basicamente evitar fluxos acidentais de informações, o que é essencial com relação ao confinamento, e podem atuar como interfaces de verificação para a detecção de erros.

A avaliação dos danos é executada através de estratégias referidas como estáticas ou dinâmicas. São englobadas na classificação como técnicas estáticas as especificadas no projeto inicial do sistema. De forma geral, são bastante amplas, embasadas em falhas de hardware, e servem para uma primeira estimativa da extensão dos danos causados. As técnicas dinâmicas são usadas no refinamento das estimativas iniciais obtidas pelo uso de técnicas estáticas; atuam através da exploração de estruturas de dados do sistema. Basicamente, utilizam-se dos mecanismos explicados na seção anterior destinados à detecção de erros e de técnicas de diagnóstico.

Há bastante inter-relação entre as medidas e mecanismos usados para lidar com confinamento e avaliação de danos. Os quatro princípios básicos são conceituados a seguir.

- Ações atômicas: a atividade do sistema é constituída de operações primitivas que são executadas pelos componentes do sistema. Assim, a atividade de um grupo de componentes constitui uma ação atômica se não há interação entre este grupo e o resto do sistema, durante o seu desenvolvimento. O uso do conceito de ações atômicas permite ao projetista indicar as seções de atividade, isto é, as seqüências de operações primitivas de um processo ou de um conjunto de processos que podem ser executadas atomicamente. Tal execução atômica tem a propriedade de não permitir fluxo de informações entre o processo (ou conjunto de processos) e o resto do sistema.

- Isolamento de processos: é o princípio de que cada processo não deve ter nenhuma capacidade além da necessária para cumprir sua tarefa; toda ação é proibida, a menos que seja permitida explicitamente. Isto significa

que os processos de um sistema não podem interagir exceto segundo modos pré-arranjados; nenhuma forma inesperada de interferência pode ocorrer. As informações descrevendo as capacidades de um processo devem ser protegidas contra alterações.

- Hierarquia de processos: permite a decomposição de um sistema complexo em termos de uma hierarquia aninhada de máquinas abstratas, tornando o sistema mais claro conceitualmente e de maior confiabilidade. A principal razão da melhoria destas características relaciona-se à integração do conceito de modularidade ao sistema, com todas as suas vantagens inerentes.

- Controle de recursos: o controle de recursos implementa a atribuição de unidades de recursos físicos a objetos computacionais como, por exemplo, de processadores a processos ou de page frames a segmentos. O princípio fundamental define que, quando uma unidade é retirada (preempted, no original, em inglês) de um objeto, o estado da unidade deve ser salvo, e a unidade colocada em um estado nulo; quando for reatribuída ao objeto, deve ser restaurado o estado da unidade correspondente àquele da última preempção do objeto. Portanto, nenhuma unidade que contenha vestígios de uso anterior por um objeto deve passar ao controle de um objeto diferente.

Os princípios de ações atômicas, isolamento de processos, hierarquia de processos e controle de recursos são alcançados pelo uso de mecanismos de proteção como: a definição de modos supervisor e usuário, o uso de anéis de proteção, o emprego do conceito de máquinas virtuais e a definição de capacidades. A escolha dos mecanismos depende do tipo de proteção que se quer fornecer ao sistema; de modo geral, mecanismos implementados para proteger o sistema contra processos maliciosos – que tentam romper restrições que supostamente deveriam ser impostas ao fluxo de dados – também são efetivos contra processos falhos.

Na prática, a avaliação de danos é um aspecto que deve ser considerado em conjunto com outros aspectos de tolerância a falhas, como as técnicas de detecção de erros e de recuperação. Na maior parte dos sistemas, é mais fácil prevenir o espalhamento dos danos do que determinar, após a ocorrência da falha, as alterações que possam ter sido provocadas sobre o fluxo de dados.

À medida que os sistemas computacionais vêm se tornando mais distribuídos e descentralizados, com múltiplos processadores operando de forma autônoma, torna-se mais importante prevenir o espalhamento dos danos e mais difícil de projetar estratégias amplas para a avaliação dinâmica de danos. Assim, é necessário prover o sistema de interfaces de verificação adequadas, que limitem a extensão dos danos, pela detecção dos erros o mais cedo possível.

3.2.3 Recuperação de erros

As técnicas de recuperação de erros visam transformar o estado atual incorreto do sistema em um estado livre de falhas, escolhido e definido durante a especificação, a partir do qual pode ser retomada a operação normal do sistema. Esta fase deve suceder as de detecção de erros e de confinamento e avaliação de danos, a fim de evitar que sobrevenha a ocorrência de um defeito no sistema.

As técnicas de recuperação são usualmente classificadas em dois grupos: são as técnicas de retorno e as de avanço. Estes grupos são complementares e não exclusivos, de tal forma que pode-se optar por usá-los de forma combinada. O princípio de funcionamento dos métodos consiste na busca de um novo estado livre de falhas. As diferenças e aplicabilidade de cada um destes métodos são explicados a seguir, de forma suscinta.

Técnicas de avanço (forward error recovery):

Baseiam-se na transformação do estado errôneo, efetuando correções para remover erros. Assim o sistema é conduzido a um novo estado, não ocorrido anteriormente, ou a um estado por onde o sistema não passou desde a última manifestação de erro. Este enfoque:

- pode ser bastante eficiente, mas é específico a cada sistema;
- não pode ser implementado através de mecanismos;
- depende da extensão dos danos;
- e mostra-se adequado apenas quando os danos podem ser previstos antecipadamente, de forma razoavelmente acurada.

Técnicas de retorno (backward error recovery):

O sistema é conduzido para um estado pelo qual passou antes da ocorrência de falha, descartando toda informação referente ao estado atual. Portanto, as técnicas pressupõem o estabelecimento de pontos de retorno, que são instantes durante a execução de um processo, a partir dos quais o estado corrente pode ser restaurado. A recuperação por retorno:

- pode ser bastante onerosa do ponto-de-vista de implementação;
- pode apresentar problemas em casos onde há manipulação de objetos que não podem ser restaurados.

Entretanto apresenta como vantagens:

- a independência da extensão dos danos;
- a aplicabilidade geral a todos os sistemas;

- ser facilmente implementável como mecanismo;
- além da possibilidade de recuperação mesmo para falhas não previstas, incluindo as de projeto.

Devido às dificuldades de estabelecimento de procedimentos estruturados para a realização de recuperação por técnicas de avanço, encontra-se na literatura apenas mecanismos para implementação de recuperação por retorno. A seguir, são listadas as técnicas mais representativas neste contexto.

- Pontos de verificação (checkpoints): constitui-se no mecanismo mais direto, para assegurar que o estado de um processo pode ser restaurado. Consiste em efetuar o salvamento de uma cópia de todo ou parte do estado do sistema, quando um ponto de recuperação é estabelecido, sendo este conteúdo referido como ponto de verificação.
- Pistas de auditoria (audit trails): apresenta procedimentos de certa forma contrários aos adotados para os pontos de verificação. A partir do estabelecimento de um ponto de recuperação, monitora cuidadosamente as atividades subseqüentes do processo e preserva registros suficientemente detalhados da atividade. Assim, assegura que todas as modificações feitas no estado do processo podem ser revertidas quando for necessária a restauração do ponto de recuperação. O registro histórico da atividade é referida como pista ou trilha de auditoria ou log.
- Recuperação encoberta (recovery cache): armazena valores correntes de objetos que estão prestes a serem modificados, preservando-os como dados de recuperação. Assim, os dados de recuperação são acumulados de forma incremental, compondo pontos de verificação parciais. Observe que estes pontos referem-se à recuperação em arquivos e não em processos, sendo empregados principalmente para recuperação em memórias de computadores.

Cabe acrescentar que a implementação de mecanismos de recuperação é simples e direta em sistemas com um único processo ou com processos independentes entre si; torna-se mais complexa em sistemas com múltiplos processos concorrentes, sejam competitivos ou cooperativos. Neste segundo caso, são adotadas soluções que assumem a diminuição dos graus de liberdade na comunicação entre processos, ou embutem overheads adicionais na operação do sistema.

3.2.4 Tratamento de falhas

Procedimentos de tolerância a falhas que não envolvam a identificação da unidade ou componente afetado estão deixando uma abertura para que se repita o problema anterior, a qualquer momento. Portanto, é necessário providenciar mecanismos que diagnostiquem a origem do erro e atuem na correção de suas causas. Dois pressupostos básicos estão relacionados à questão:

- as três fases anteriormente expostas (detecção de erros, confinamento e avaliação de danos e a recuperação de erros) não garantem a identificação da falha que originou o erro ou erros subseqüentes. Um dado erro gerado pode ter diferentes causas, não existindo um mapeamento direto entre os conjuntos de falhas e erros.

- as hipóteses de falhas utilizadas correntemente supõem a ocorrência de apenas uma falha por vez: o prosseguimento da operação sem a eliminação da causa original, pode ter por consequência o surgimento de nova falha, eventualmente não prevista nas listas ou classes de falhas iniciais.

Deste modo, o primeiro aspecto do tratamento de falhas consiste em tentar localizar a falha de forma precisa. Os objetivos seguintes referem-se a efetuar o reparo da falha e/ou a reconfigurar o restante do sistema para evitar a falha, quando o erro tiver sido causado por falha temporária. Em caso de falha temporária, não é necessário procedimento de reparo ou reconfiguração.

De acordo com [AnLe81], a localização da falha é realizada em duas fases, que diferem em nível de precisão ou detalhamento. Na primeira fase, a localização pode restringir-se à definição do módulo a ser substituído. Assim, até este momento, a ênfase reside na execução de procedimentos rápidos para definição do módulo e da estratégia de substituição, tendo por objetivo básico a manutenção do sistema operacional, ou a rápida retomada da operação.

Na segunda fase, são efetuados procedimentos que visam localizar a falha de forma mais precisa, permitindo a execução de reparos econômicos e a retomada da operação integral do sistema. Nesta fase, a ênfase reside na qualidade dos mecanismos utilizados para realizar a localização das falhas e a reconfiguração do sistema. Os testes de diagnóstico para a definição dos componentes a serem substituídos, bem como a remoção dos componentes falhos, podem ser aplicados manual ou automaticamente. Em procedimentos automáticos, os componentes críticos do sistema, e portanto aqueles que devem ser implementados com o melhor nível de confiabilidade, são os responsáveis pela aplicação dos testes de diagnóstico e pelas decisões de reconfiguração necessárias para o reparo do sistema.

Os procedimentos de tolerância a falhas associados a cada sistema, estão limitados quanto ao número de falhas que eles têm capacidade de manipular. Assim há uma certa provisão para remover componentes defeituosos. Quando a remoção de um componente defeituoso é feita automaticamente, o sistema pode ser reconfigurado de modo a atuar com menor número de componentes, referido como falha suave (ou degradação suave); ou o componente defeituoso pode ser imediatamente substituído, o que é referido como auto-reparo.

A opção de auto-reparo merece consideração especial para sistemas que operam sem atendimento por longos períodos de tempo e necessitam de reposição dos componentes falhos após sua reconfiguração. Baseia-se em técnicas tais como [McCl82]:

- redundância dinâmica, onde a unidade falha é substituída por uma estepe logo após a detecção do erro. Há a disponibilidade de um certo número de módulos-estepe para substituição dos defeituosos, sendo que estes módulos-estepe podem ser alimentados apenas quando entram em funcionamento. As vantagens da desconexão dizem respeito não só à economia de energia, mas têm relação com hipóteses de falhas vinculadas à vida útil dos equipamentos. Os módulos a serem substituídos são indicados por técnicas de detecção de erros, conforme visto na seção 3.2.1.

- redundância híbrida, onde a lógica de votação tripla ou múltipla (conforme será visto na seção 3.3.3) é usada para mascarar falhas e fornecer a detecção, sendo a unidade defeituosa substituída por uma estepe. Quando a saída de um módulo discorda do votador, ele é considerado falho, sendo portanto desconectado através do circuito de chaveamento e substituído. O ponto crítico do circuito situa-se sobre o circuito de chaveamento, que atua como elemento limitante da confiabilidade do sistema.

- redundância auto-eliminadora, na qual usa-se lógica de votação no mascaramento e detecção de falhas, seguida pelo isolamento da unidade defeituosa. Neste caso, diferentemente da redundância híbrida, todos os módulos estão inicialmente conectados, sendo eliminados ao longo da operação os módulos falhos. O circuito de controle torna-se mais simples.

Sistemas auto-reparadores usando as técnicas de redundância híbrida e auto-eliminadora são mostrados esquematicamente nas figuras 3 e 4, respectivamente.

3.3 Técnicas de controle de respostas sob falha

O princípio fundamental para obtenção de tolerância a falhas está associado ao uso de redundância, podendo esta existir através de recursos materiais ou software adicionais, ou ainda pela repetição de procedimentos (redundância temporal). Vinculados a estes, devem existir programas ou elementos de gerenciamento, os quais irão coordenar a realização destes procedimentos e/ou verificar resultados de testes.

A escolha entre diferentes técnicas é dependente da aplicação e características do sistema em projeto, ou seja, do tipo de resposta que se deseja obter do sistema.

3.3.1 Técnicas para sistema livres de falhas

Se o objetivo é o fornecimento de resposta livre de falhas (fail-safe), os mecanismos associados são basicamente implementados por harware. As técnicas associadas a respostas livres de falhas fazem com que as saídas permaneçam ou sejam conduzidas a um estado seguro, quando ocorre uma falha. Estas técnicas

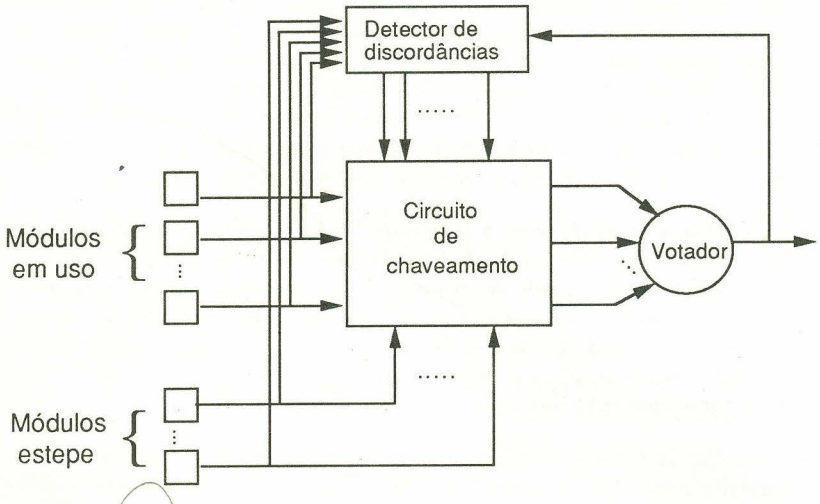

Figura 3: Representação esquemática de redundância híbrida

envolvem basicamente sinais de controle replicados ou sinais adicionais, que impedem a tomada de determinadas ações quando há discordância entre os sinais. Um exemplo simples consiste em um teste embasado no uso de múltiplas assinaturas. Qualquer assinatura inválida ou perdida durante o processamento faz com que o teste indique esta condição (ou não tenha sucesso).

Uma aplicação prática convencional corresponde ao uso desta técnica para a elaboração do esquema de controle de barramento para sistema de multiprocessadores tolerante a falhas. Em caso de falha, os sinais de controle assumem valores divergentes e o acesso ao barramento é impedido [HSL78]. Uma rede simples que cumpre a função recém descrita é mostrada na figura 5. Os sinais $C1$ e $C2$ têm valores idênticos enquanto não há falha. Quando ambos são iguais a 1, a saída A assume o valor de a. Quando ambos são iguais a 0, a saída é 0. Se uma falha faz com que eles discordem, a saída é levada para o valor seguro 0 o qual impede o acesso ao barramento.

3.3.2 Técnicas para sistemas seguros a falhas

Se o objetivo é um sistema que atue na modalidade fail-stop, necessita-se da indicação de erro fornecida pelos sistemas seguros a falhas. Neste caso, a ênfase das técnicas está associada à detecção de erros, e os mecanismos são, em sua maior parte, coincidentes com os utilizados na fase de detecção de erros, apresentada na seção 3.2.1. Os exemplos mais expressivos neste enfoque são: códigos

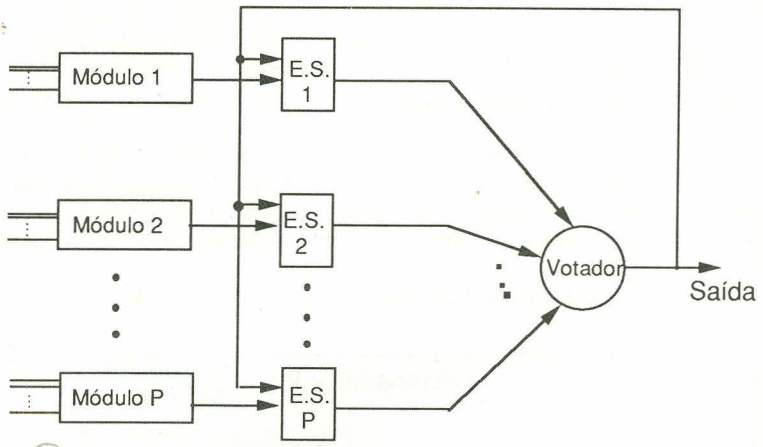

Figura 4: Representação esquemática de redundância auto-eliminadora

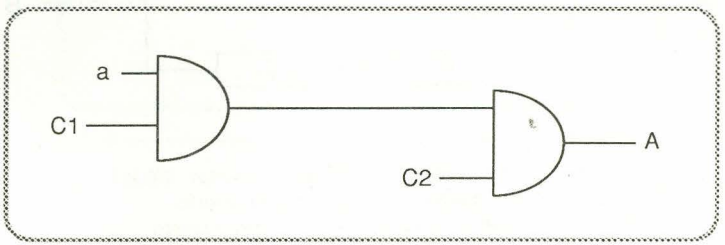

Figura 5: Exemplo de rede livre de falhas

de detecção de erros e duplicação, usando técnicas de redundância de hardware; teste de razoabilidade, empregando redundância de software; repetição de programa, realizado com redundância temporal; e técnicas combinadas de hardware e software como proteção de memória, processo duplicado e o uso de processador cão-de-guarda (watchdog).

3.3.3 Técnicas para mascaramento de falhas

Por outro lado, se o sistema é projetado para não permitir o aparecimento dos efeitos das falhas para o meio externo (ou usuário), é preciso optar por técnicas de mascaramento de falhas. Deste modo, o sistema continua a fornecer a resposta correta, mesmo diante da ocorrência de falhas, sendo corrigidos os erros potenciais antes da sua manifestação.

Estas técnicas são apresentadas a seguir, subdivididas segundo sua forma de implementação, por hardware ou software.

Redundância de hardware

Quadruplicação de componentes:

Esta opção é usada para dispositivos analógicos ou digitais; substitui cada componente por uma rede com outros quatro [Crev56]. A ocorrência de falha em um dos componentes é corrigida automaticamente pelo padrão de interconexão. Este padrão de interconexão baseia-se em probabilidades de ocorrência dos diversos tipos de falhas.

Esta técnica é ilustrada pela figura 6.

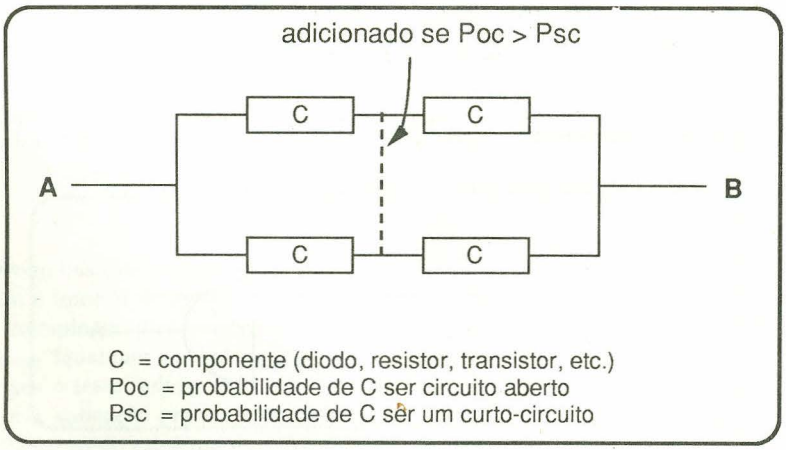

Figura 6: Exemplo da técnica de quadruplicação de componentes

Quadruplicação lógica:

Baseia-se no mesmo princípio da técnica anterior, sendo aplicável a redes de portas lógicas [Tryo62]. Cada porta lógica é substituída por uma rede de quatro portas, de tal forma a manter inalterada a função do circuito, e assegurar que uma única falha em uma das portas seja corrigida automaticamente pelo padrão de interconexão.

Redundância modular tripla:

É a técnica mais geral de mascaramento de hardware. Encontra-se referências a esta técnica em [VonN56] e em [LyVa62]; posteriormente, tornou-se uma técnica

de larga aplicação. Cada módulo é substituído por outros três idênticos ao primeiro, cujas saídas passam por uma "rede de votação" antes de serem transferidas às saídas. Qualquer falha afetando um único módulo é mascarada.

A maior vantagem da técnica é a flexibilidade na escolha do módulo que compõe a unidade de multiplicação. Ao longo do texto é empregada a abreviatura TMR, do inglês: Triple Modular Redundancy, como referência a esta técnica.

Uma variação desta, consiste no uso de um número de módulos maior do que três, sendo que o votador avalia a função correspondente à resposta fornecida pela maioria dos blocos envolvidos, ou seja, para $N = 2n+1$ módulos, o votador verifica a igualdade entre $(n+1)/N$.

Esta técnica é ilustrada na figura 7.

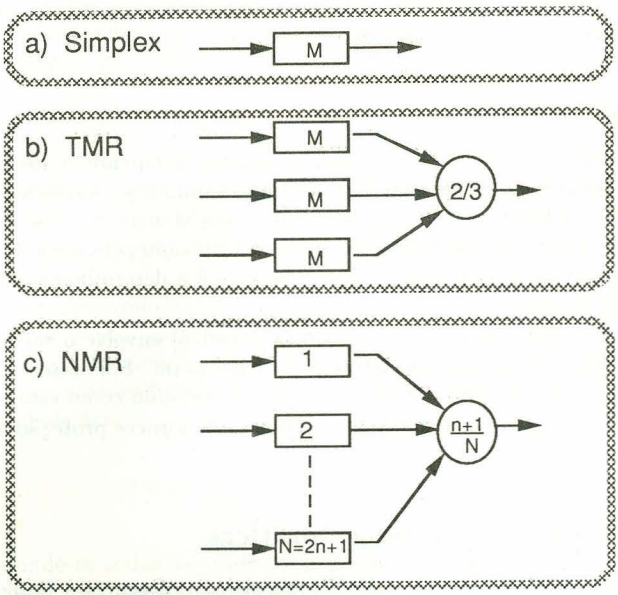

Figura 7: Redundância modular tripla

Códigos de correção de erros:

São a forma mais comumente empregada de mascaramento por hardware. Alguns bits são adicionados à informação armazenada ou transmitida. Quaisquer padrões de bits afetados por falha, cuja alteração esteja dentro da capacidade do código usado, podem ser corrigidos, de tal forma que somente a informação correta é fornecida às saídas.

Este método é usado predominantemente em memórias e dispositivos de acesso serial como discos. A recuperação da informação baseia-se no uso de um circuito o qual corrige os erros de uma determinada classe, antes da transmissão. A técnica é portanto efetiva enquanto este circuito corretor não apresenta falhas ([Hsia70] e [Chie73]).

Redundância de software

Programas em n-versões ou programação diversitária:

São executadas diversas versões de um programa sobre o mesmo conjunto de dados; obtém-se as saídas pela votação entre os resultados dos programas individuais [ChAv78]. As diversas versões do programa devem ser escritas de forma independente, a partir de uma dada especificação, a fim de garantir sua efetividade na detecção de falhas de projeto.

Bloco de recuperação:

Requer diversas versões de um programa escritas independentemente, mas só executa os programas-extra quando é detectado um erro. Ao término de cada segmento, há um teste de aceitação que verifica a existência de erros: em caso positivo, um programa alternativo é executado e verificado pelo teste de aceitação. Uma das dificuldades relacionadas a esta técnica é a determinação de testes de aceitação adequados.

A técnica padrão proposta por Randall [Rand75] envolve o salvamento periódico do estado do programa chamado de checkpoint. Sob detecção de falha, o programa é repetido com o último checkpoint servindo como estado inicial.

É efetiva somente para falhas temporárias e não fornece proteção para falhas no projeto de software.

3.4 Algumas considerações práticas

A seguir, são comentados alguns aspectos práticos referentes à implementação de técnicas de tolerância a falhas.

Inicialmente são estudadas algumas conseqüências do uso de redundância modular sobre o nível de confiabilidade do sistema. A idéia é mostrar que a adição de módulos redundantes não é necessariamente benéfica, exigindo verificar sua repercussão sobre os demais parâmetros de avaliação do sistema.

No outro ítem, é relatado um experimento prático envolvendo diversidade de programação, o qual foi realizado nos últimos quatro anos (de 1987 a 1990) na disciplina de Sistemas Tolerantes a Falhas, ministrada no Curso de Pós-Graduação em Ciência da Computação da UFRGS. O contato prático com a técnica suscita alguns comentários interessantes.

3.4.1 Confiabilidade x TMR

A opção pelo uso de estruturas redundantes em um sistema, para obter maior segurança de funcionamento, tem implicações sobre os parâmetros de confiabilidade do mesmo. Com o intuito de tornar estes conceitos mais facilmente compreensíveis, serão comparados os dados referentes a situações simplex e com redundância TMR. Maiores detalhes sobre os cálculos de confiabilidade associados ao caso tratado podem ser obtidos em referências como [SiSw82] e [SoMa87].

A confiabilidade $R(t)$ é medida através da probabilidade de que um sistema funcione adequadamente durante o período de tempo compreendido no intervalo $[0,t]$, conforme [OsNi80]. Numericamente e de acordo com o modelo exponencial, a confiabilidade é expressa por:

$$R(t) = e^{-\lambda t}$$

onde λ corresponde à taxa de falhas do módulo.

O tempo médio entre falhas, $MTBF$, é dado por:

$$MTBF = \int_0^\infty R(t)dt.$$

Para uma estrutura não redundante (composta por um módulo) tem-se:

$$R_N(t) = e^{-\lambda t} \qquad \text{e} \qquad MTBF = 1/\lambda.$$

Se for usada redundância modular tripla (com voto majoritário), pode-se adotar por base a configuração r–em–n do modelo estrutural, que representa um sistema com n unidades cujo funcionamento correto é assegurado se ao menos r das n unidades funcionam de forma adequada. Neste caso, a expressão da confiabilidade fica:

$$R(t) = \sum_{k=r}^{n} \binom{n}{k} e^{-k\lambda t}(1 - e^{-\lambda t}),$$

considerando-se todas as unidades com taxas de falha iguais a λ.

Para a estrutura 2–em–3, obtém-se

$$R_R(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$

e

$$MTBF = 3/2\lambda - 2/3\lambda = 5/6\lambda.$$

Comparando-se os $MTBF$ das duas estruturas, redundante e não redundante, verifica-se que a estrutura não redundante tem $MTBF$ maior do que sua correspondente implementada com TMR. Uma explicação rápida para o caso pode embasar-se no fato de que o uso de redundância aumenta o número de

componentes do sistema, e conseqüentemente a probabilidade de ocorrência de falhas.

Na verdade, como a probabilidade de ocorrência de falhas não varia linearmente com o tempo, a representação das curvas de $R(t)$ através de um gráfico (ver figura 8) permite verificar que a opção pelo uso de TMR tem maior confiabilidade para missões de curta duração, ou seja, quando:

$$\text{duração da missão} \leq (\ln 0,5)/\lambda = 0,693 MTBF.$$

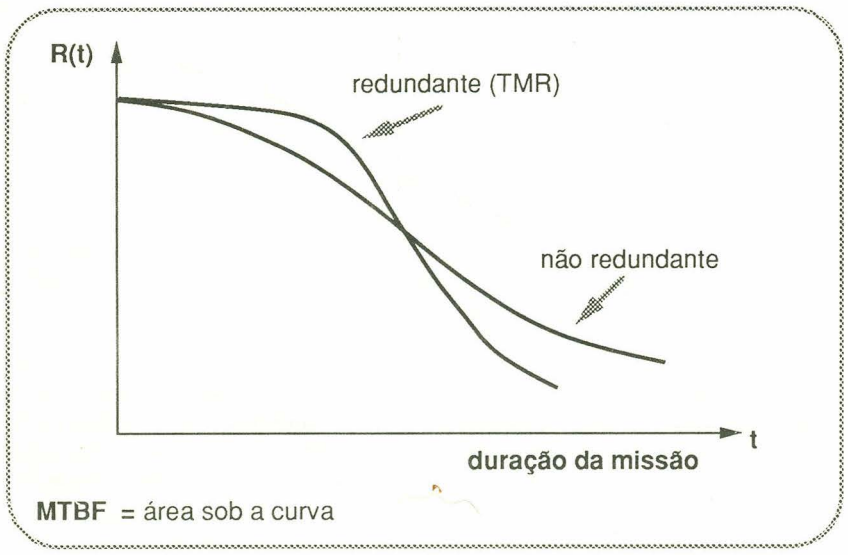

Figura 8: Curvas de confiabilidade para o exemplo em estudo

Para a execução dos cálculos acima, considerou-se o votador como não sujeito a falhas, ou seja, com confiabilidade igual a 1. Em um cálculo real, o votador seria associado em série ao conjunto de módulos redundantes, e com uma implementação de alta confiabilidade, o que se refletiria em uma baixa taxa de falhas. E a fórmula corrigida para o cálculo da estrutura TMR com votador, para esta nova situação será dada pelo produto da confiabilidade da estrutura redundante (R_R) pela confiabilidade do votador (R_V):

$$R = R_R . R_V$$

Em [SiSw82], são apresentados adicionalmente cálculos para situação com mais de um circuito votador, portanto com uma rede redundante de votadores,

além do estudo dos reflexos do uso de redundância sobre a confiabilidade em outros exemplos.

3.4.2 Experiência em diversidade de programação

Diversidade, também chamada programação diversitária, é uma técnica utilizada para obter tolerância a falhas em software ([ChAv78], [AvKe84]). A partir de um problema a ser solucionado são implementadas diversas soluções alternativas, sendo a resposta do sistema determinada por votação.

A aplicação da técnica não leva em conta se erros em programas alternativos apresentam a mesma causa (por exemplo: falsa interpretação de uma especificação ou uma troca de um sinal em uma fórmula). Os erros, para poderem ser detectados, devem necessariamente se manifestar de forma diferente nas diversas alternativas, ou seja, devem ser estatisticamente independentes. Experimentos comprovam que o número de manifestações idênticas (erros que assim não seriam detectados) é significativamente menor que o número total de erros [KLJ85].

Diversidade pode ser utilizada em todas as fases do desenvolvimento de um programa, desde sua especificação até o teste, dependendo do tipo de erro que se deseja detectar (erro de especificação, de projeto, ou de implementação). Essa técnica é chamada de projeto diversitário quando o desenvolvimento do sistema é realizado de forma diversitária e de programação em n-versões quando se restringe à implementação do sistema. Pode ser também usada como técnica de prevenção de falhas. Nesse último caso, várias alternativas de projeto ou de implementação são desenvolvidas para que, na fase de teste, erros eventuais possam ser localizados e corrigidos de uma forma mais simples e efetiva. No final da fase de teste, é então escolhida a alternativa em que se detectou a menor ocorrência de erros, e apenas esta alternativa é integrada ao sistema.

A facilidade no reconhecimento de erros na fase de teste do sistema, a tolerância tanto de falhas intermitentes quanto de permanentes e a atuação potencial também contra erros externos ao sistema (como por exemplo erros do compilador, do sistema operacional e até mesmo falhas de hardware) são vantagens comumente atribuídas a programação diversitária. Entretanto desvantagens da técnica também devem ser citadas, como o aumento dos custos de desenvolvimento e manutenção, a complexidade de sincronização das versões e o problema de determinar a correlação das fontes de erro.

Enquanto pode ser provado formalmente que a redundância de elementos de hardware aumenta a confiabilidade do sistema, tal prova não existe para a diversidade em software. Vários fatores influenciam a eficácia da programação diversitária: as equipes podem trocar algoritmos entre si, os membros das equipes podem por formação tender a adotar os mesmos métodos de desenvolvimento, ou as equipes podem buscar suporte nas mesmas fontes. Qualquer uma dessas correlações imprevisíveis se constitui uma fonte potencial de erros. Assim, a eficácia da programação diversitária deve ser comprovada na prática.

Com esse objetivo, foram promovidos experimentos no Curso de Pós-Graduação em Ciência da Computação, UFRGS, nos anos de 87 a 90. Esses experimentos são baseados em uma experiência similar realizada em 1987 na Universidade de Karlsruhe, Alemanha Ocidental. O resultado do primeiro experimento da UFRGS, realizado com 9 versões, e sua comparação com o experimento alemão foram relatados em [WeWe88].

Em [WeWe89] são analisados os resultados alcançados executando em conjunto as versões dos experimentos de 87 e 88 realizados no CPGCC/UFRGS, somando 21 versões. Ênfase especial é dada a discussão sobre o conjunto mínimo de versões necessárias para garantir a confiabilidade desejada. É apresentado também o ambiente que suporta, em Porto Alegre, a execução do experimento e coleta os resultados de cada versão contra uma bateria de 48 testes disponíveis, simulando paralelismo na execução das versões. O ambiente foi desenvolvido em Modula-2 [Wirt85], e é robusto o suficiente para manter o experimento ativo, independente da interrupção do processamento das versões causadas por toda a sorte de erros durante sua execução. O ambiente é genérico e pode ser usado em outros experimentos de programação diversitária, com um número variável de participantes e versões.

Em todos os experimentos realizados, o comportamento do sistema resultante do processamento paralelo e votação de um grande número de versões (entre 9 a 18 em cada ano) mostrou ser mais confiável que cada programa tomado individualmente. Para exemplificar, no experimento executado em maio de 1990 com 18 participantes e com uma bateria de 48 testes, o sistema apresentou resultado correto 47 vezes, enquanto os quatro melhores programas acertaram apenas 46 vezes. Entre os 14 restantes a média de acertos reais esteve em 28,57 em 48 testes, ou seja os programas forneceram em média 59,52% resultados corretos.

Apesar dos resultados positivos alcançados pelos experimentos, uma avaliação definitiva sobre a técnica de diversidade parece no momento prematura. Assim como para os métodos de verificação formal, esse conceito ainda deve ser aprofundado até poder ser aplicado sistematicamente na obtenção de sistemas de alta confiabilidade.

4. Conclusões

Neste artigo foram introduzidas algumas noções básicas, que se constituem em atributos da segurança de funcionamento, e podem ser reunidas em três classes, conforme segue.

- Entraves à segurança de funcionamento: são as dificuldades à manutenção da confiança inicialmente depositada no funcionamento correto do sistema. A causa desta perda de confiança deve-se à ocorrência de circustâncias indesejáveis mas não imprevistas, que são as falhas, erros e defeitos.

- Meios de obtenção da segurança de funcionamento: são as técnicas, procedimentos e ferramentas empregadas para dotar o sistema da qualidade esperada em termos de confiança no serviço fornecido. Estes meios são expressos através das técnicas de prevenção e de tolerância a falhas. A efetividade destas técnicas é validada por procedimentos complementares, referidos como eliminação e previsão de falhas, que utilizam-se de avaliação de probabilidades e observação da ocorrência de falhas, sendo classificados por diversos autores como meios de validação da segurança de funcionamento.

- Medidas de avalição da segurança de funcionamento, tais como confiabilidade e disponibilidade. Estas medidas permitem verificar a qualidade do serviço fornecido, que é atingida por determinado sistema em função do tipo e características da implementação definida, consideradas as condições de ocorrência de falhas as quais ele está submetido.

Estes atributos são resumidos através da taxonomia apresentada na figura 9, que já se constitui em uma apresentação clássica do tema.

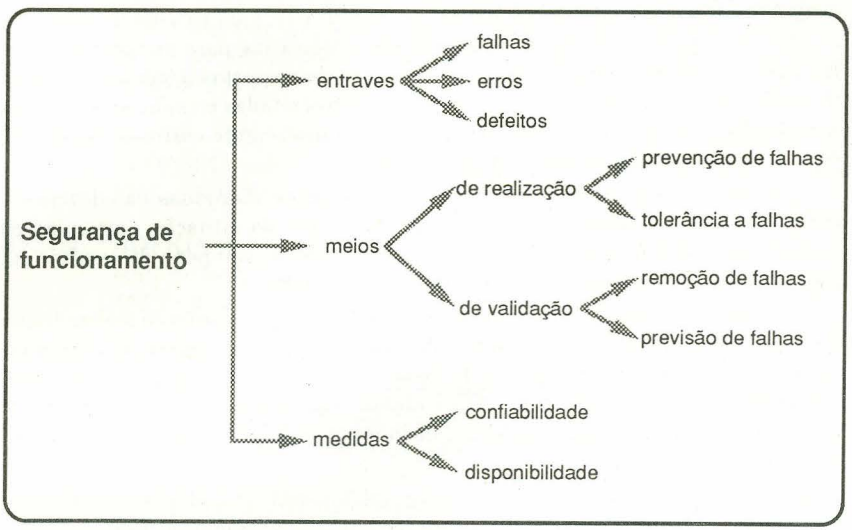

Figura 9: Taxonomia da segurança de funcionamento

A partir destes conceitos, foram detalhados os principais meios através dos quais se obtém segurança de funcionamento , empregando-se tolerância a falhas,

ou seja, o que é referido habitualmente como técnicas de tolerância a falhas. Para tal, foram considerados os seguintes tipos de respostas possíveis:

- saídas sempre corretas, com mascaramento de falhas;
- saídas corretas ou com indicação de erro (seguros a falhas);
- saídas corretas ou em um valor seguro (livre de falhas).

Em sistemas não tolerantes a falhas, admite-se a situação de erro na saída, sem qualquer sinalização do evento.

Em seguida, foram explicados os mecanismos usados para intervir sobre o sistema, a fim de garantir segurança de funcionamento. Como visto, é necessário valer-se de técnicas para detectar os erros, antes que as informações que os contém sejam usadas inadvertidamente pelo usuário. Além disto, é preciso proceder às correções internas ao sistema, verificando a extensão dos danos e retornando o sistema a uma situação confiável (recuperação de erros). Como medidas preventivas ao espalhamento de erros, foram vistos mecanismos de confinamento de danos, e de tratamento de falhas, para corrigir fisicamente o elemento falho pertinente ao sistema.

As técnicas de controle de respostas são necessárias para completar a elaboração do sistema, garantindo o fornecimento de respostas adequadas ao tipo de aplicação em projeto. Com este objetivo, foram citadas e explicadas técnicas através das quais pode-se obter sistemas com saídas sempre corretas, em estado seguro ou com indicação da existência de erro.

Considerações práticas mostraram que as técnicas abordadas não devem ser utilizadas exaustivamente, sem uma análise prévia da situação particular de cada sistema, sua futura utilização e objetivos visados, sob pena de resultar em prejuízos sob ponto de vista dos parâmetros desejados.

A abordagem dos experimentos em diversidade de programação realizados na UFRGS visou mostrar os resultados obtidos, a exemplo de outras experiências realizadas no exterior. Do ponto de vista didàtico, a experiência foi extremamente proveitosa, fazendo com que os alunos sugerissem um maior número de experimentos práticos para atingirem melhor sensibilidade com relação aos conceitos estudados.

Agradecimentos

A autora agradece aos Professores Taisy Weber e Raul Weber pelo auxílio na confecção deste artigo, principalmente pelas seções de diversidade de programação e falhas intencionais.

Referências

[AnLe81] ANDERSON, T.; LEE, P. A. *Fault tolerance – principles and practice.* Englewood Cliffs, Prentice-Hall, 1981.

[Aviz78] AVIZIENIS, A. Fault-tolerance, the survival atribute of digital systems. *Proceedings of the IEEE.* New York, 66(10):1109-1125, out. 1978.

[AvKe84] AVIZIENIS, A.; KELLY, J. P. Fault tolerance by design diversity – concepts and experiments. *Computer*, New York, 17(8):67-80, Aug. 1984.

[ChAv78] CHEN, L.; AVIZIENIS, A. N-version programming: a fault tolerance approach to reliability of software operation. In: Annual International Symposium on Fault-Tolerant Computing, 8, 1978. *Proceedings.* New York, IEEE, 1978. p. 3-9.

[Chie73] CHIEN, R.T. Memory error control: beyond parity. *IEEE Spectrum.* New York, 10(7):18-23, jul. 1973.

[Crev56] CREVELING, C.J. Increasing the reliability of electronic equipment by the use of redundant circuits. *Proceedings of the IRE.* New York, 44(4):509-515, abr. 1956.

[Hsia70] HSIAO, M.Y. A class of optimum minimum odd-weight-column SED/DED codes. *IBM Journal of Research and Development.* New York, 14(4):395-401, jul. 1970.

[HSL78] HOPKINS, A. L; SMITH, T. B.; LALA, J. H. FTMP – a highly realible fault-tolerant multiprocessor for aircraft. *Procedings of the IEEE*, New York, 66(10):1221-1239, Oct. 1978.

[KLJ85] KNIGHT, J. C., LEVESON, N. G.; St.JEAN, L. D. A large scale experiment in n-version programming. In: Annual International Symposium on Fault-Tolerant Computing, 15, Ann Arbor, June 19-21, 1985. *Proceedings.* New York, IEEE, 1985. p. 135-139.

[Lapr85] LAPRIE, J-C. Dependable computing and fault-tolerance: concepts and terminology. In: Annual International Symposium on Fault Tolerant Computing, 15. Ann Arbor, jun. 19-21, 1985. *Proceedings.* New York, IEEE, 1985. p. 2-11.

[LCGP89] LAPRIE, J.C.; COURTOIS, B.; GAUDEL, M.C. e POWELL,D. *Sûreté de fonctionnement des systémes informatiques.* Paris, DUNOD Informatique, 1989.

[LyVa62] LYONS, R.E. e VANDERKULK, W. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development.* New York, 6(3): 200-209, abr. 1962.

[McCl82] McCLUSKEY, E.J. *Fault tolerant systems.* CRC/Computer Systems Laboratory, Stanford University, abr. 1982. (CRC Technical Report n. 82-1).

[OsNi80] Osaki, S. e Nishio, T. Reliability evaluation of some fault-tolerant computer architectures. *Lecture Notes on Computer Science*, 97. Springer-Verlag, Heidelberg, 1980.

[Rand75] RANDALL, B. System structure for software fault tolerance. *IEEE Transactions on Software Engineering.* New York, 1(2):220-232, jun. 1975.

[SiSw82] SIEWIOREK, D. P.; SWARZ, R. S. *The Theory and Practice of Reliable System Design.* Bedford, Digital, 1982.

[SoMa87] SOUZA, J.M. e MARTINI, M.R.B. Avaliação de confiabilidade de sistemas. In: Simpósio em Sistemas de Computadores Tolerantes a Falhas, 2, Campinas, Ago. 24-28, 1987. *Minicurso.* Campinas, UNICAMP, 1987. Capítulo 1. p. 5-44.

[Tryo62] TRYON, J.G. Quadded logic. In: *Redundancy techniques for computing systems*, Wilcox & Mann eds. Spartan Books, Washington, 1962. p. 205-228.

[VonN56] VON NEWMANN, J. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: *Automata Studies*, Shannon & McCarthy eds. Princeton Univ. Press, 1956. p. 43-98.

[Webe89] WEBER, R. F. Vírus de computador. *Revista de Informática Teórica e Aplicada*, Porto Alegre, 1(1):79-118. Out. 1989.

[WeWe88] WEBER, R. F.; WEBER, T. S. Programação diversitária. In: XXI Congresso Nacional de Informática, Rio de Janeiro, 22 a 26 ago.,1988. *Anais.* Rio de Janeiro, SUCESU, 1988. p. 418-425.

[WeWe89] WEBER, R. F.; WEBER, T. S. Um experimento prático em programação diversitária. In: III Simpósio em Sistemas de Computadores Tolerantes a Falhas, SCTF, Rio de Janeiro, 20-22 set. *Anais.* Rio de Janeiro, 1989. p. 271-290.

[WJPW90] WEBER, T. S.; JANSCH-PORTO, I. E. S.; WEBER, R. F.; *Fundamentos de Tolerância a Falhas*, IX JAI – Jornada de Atualização em Informática, X Congresso da Sociedade Brasileira de Computação, Vitória, 22-27 jul. 1990.

[Wens78] WENSLEY, J. H. et al. SIFT: design and analysis of fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, New York, 66(10):1240-1254, Oct. 1978.

[Wirt85] WIRTH, N. *Programming in Modula-2.* 3. ed. Heidelberg, Springer-Verlag, 1985.

Curriculum vitae (Ingrid E. S. Jansch-Pôrto)

Doutora em Engenharia pelo IMAG, Grenoble, França (1985), Mestre em Ciência da Computação pela UFRGS (1982) e Engenheira Eletrônica também pela UFRGS (1979). Professora Adjunto do Instituto de Informática, e professora orientadora do CPGCC, UFRGS. Professora de disciplinas nas áreas de sistemas digitais, teste e sistemas de computação tolerantes a falhas. Pesquisadora e orientadora nas áreas de CAD de sistemas digitais e tolerância a falhas.

Endereço para contato:
Instituto de Informática
Universidade Federal do Rio Grande do Sul
Caixa Postal 1501
90001 – Porto Alegre – RS
Brasil
E-mail: ingrid@sbu.ufrgs.anrs.br

Artigo submetido em outubro 1990. Aceito na forma final em fevereiro 1990. Convertido e formatado para o estilo da Revista por Raul F. Weber a partir do texto fornecido em Wordstar pelo autor. Figuras realizadas por Taisy S. Weber a partir de cópias fornecidas pelo autor.