Tolerância a falhas: conceitos e exemplos

Taisy Silva Weber[1]
Programa de Pós-Graduação em Computação - Instituto de Informática - UFRGS
taisy@inf.ufrgs.br

Resumo

Sistemas computacionais totalmente confiáveis e cem por cento disponíveis: utopia ou exigência viável? A confiabilidade e a disponibilidade de equipamentos e serviços de computação podem ser medidos quantitativamente. Várias técnicas de projeto podem ser usadas para aumentar o valor dessas medidas, valor que pode chegar bem próximo a cem por cento. O texto mostra que sistemas totalmente infalíveis são impossíveis, pois falhas são inevitáveis. Mas usuários e desenvolvedores não precisam, e não devem, se conformar com equipamentos e serviços de baixa qualidade, desde que dispostos a arcar com o custo do emprego de técnicas de tolerância a falhas.

Índice

1 Introdução 2
1.1 Problemas com sistemas de computação 3
1.2 Desafios atuais 3
1.3 Falha, erro e defeito 4
1.4 Dependabilidade 6
1.5 Medidas de confiabilidade 6
2 Técnicas para alcançar dependabilidade 8
2.1 Tolerância a falhas 8
2.2 Fases de aplicação das técnicas de tolerância a falhas 8
2.3 Mascaramento de falhas 10
3 Redundância 11
3.1 Redundância de hardware 12
3.1.1 Redundância de hardware passiva 12
3.1.2 Redundância dinâmica 14
3.2 Redundância de software 14
3.2.1 Diversidade 15
3.2.2 Blocos de recuperação 16
4 Aplicações de Sistemas Tolerantes a Falhas 16

[1] Professora orientadora do PPGC, UFRGS, coordenadora do mestrado profissional em Engenharia da Computação, UFRGS, Diretora Administrativa da Sociedade Brasileira de Computação

4.1 Áreas de Aplicação 17
4.2 Sistemas de tempo real 17
4.3 Sistemas digitais de telefonia 18
4.4 Sistemas de transações 18
4.5 Redes de serviço locais 19
4.6 Sistemas seguros 20
4.7 Exemplos por área de aplicação 21
5 Conclusão 22
6 Bibliografia 22

1 Introdução

Computadores e seus programas são conhecidos por automatizarem e acelerarem uma série de tarefas enfadonhas e repetitivas, liberando seus usuários para atividades mais criativas e gratificantes. Na prática, os usuários se vem às voltas com atividades bastante criativas, mas nada gratificantes, de tentar recuperar dados perdidos e enfrentar equipamento fora do ar devido às múltiplas falhas a que os sistemas de computação estão sujeitos.

Falhas são inevitáveis, mas as conseqüências das falhas, ou seja o colapso do sistema, a interrupção no fornecimento do serviço e a perda de dados, podem ser evitadas pelo uso adequado de técnicas viáveis e de fácil compreensão. O conhecimento dessas técnicas habilita o usuário a implementar as mais simples, ou exigir dos fornecedores soluções que as incorporem. Entretanto, as técnicas que toleram falhas tem um certo custo associado. O domínio da área auxilia usuários e desenvolvedores de sistemas a avaliar a relação custo benefício para o seu caso especifico e determinar qual a melhor técnica para seu orçamento.

Sistemas mais robustos em relação a falhas eram, até recentemente, preocupação exclusiva de projetistas de sistemas críticos como aviões, sondas espaciais e controles industriais de tempo real. Com a espantosa popularização de redes fornecendo os mais variados serviços, aumentou a dependência tecnológica de uma grande parcela da população aos serviços oferecidos. Falhas nesses serviços podem ser catastróficas para a segurança da população ou para a imagem e reputação das empresas.

Para não ser o elo fraco de uma corrente, o mais simples dos computadores deve apresentar um mínimo de confiabilidade. Conhecer os problemas, quais soluções existem e qual o custo associado torna-se imprescindível para todos que pretendem continuar usando e expandindo seus negócios fornecendo um serviço computacional de qualidade aos seus clientes. Para desenvolvedores de software, projetistas de hardware e gerentes de rede o domínio das técnicas de tolerância a falhas torna essencial na seleção de tecnologias, na especificação de sistemas e na incorporação de novas funcionalidades aos seus projetos.

O texto visa dar uma visão geral da área, que é bastante ampla. Os leitores interessados podem encontrar mais informações na bibliografia listada no final do texto, especialmente nos livros do Pradhan [Prad96], Siewiorek [SiSw82], Jalote [Jal94] Anderson e

Lee [AnLe81] e Birman [Bir96], que são usados nas disciplinas de Tolerância a Falhas na UFRGS e nos quais, em grande parte, se baseia esse texto.

1.1 Problemas com sistemas de computação

Confiabilidade e disponibilidade são cada vez mais desejáveis em sistemas de computação pois dia a dia aumenta a dependência da sociedade a sistemas automatizados e informatizados. Seja no controle de tráfego terrestre e aéreo ou de usinas de geração de energia, na manutenção de dados sigilosos sobre a vida e a finança de cidadãos e empresas, nas telecomunicação e nas transações comerciais internacionais de todo tipo, computadores atuam ativa e continuamente. É fácil imaginar que defeitos nesses sistemas podem levar a grandes catástrofes.

Desde os primeiros computadores, é notável como os componentes de hardware cresceram em confiabilidade. Entretanto, o software e os procedimentos para projeto estão se tornando cada vez mais complexos e apresentando cada vez mais problemas. Só a confiabilidade dos componentes de hardware não garante mais a qualidade e segurança desejada aos sistemas de computação. Como exemplo recente desses problemas pode ser citada a bem conhecida falha de projeto na unidade de ponto flutuante do Pentium, que prejudicou seu lançamento comercial. Nem todo mundo sabe entretanto que falhas de projeto são comuns no lançamento de qualquer processador e muitos *bugs* em microprocessadores de uso geral sequer ainda foram descobertos. Alguns outros defeitos [Lapr98] valem a pena ser mencionados: na guerra do Golfo em fevereiro de 1991 preocupantes relatos de falhas em mísseis foram noticiados. Em novembro de 1992 houve um colapso no sistema de comunicação do serviço de ambulâncias em Londres. Em junho de 1993, durante dois dias, não foi autorizada nenhuma operação de cartão de crédito em toda a França. Todos esses defeitos foram investigados e suas causas determinadas, mas não se tem garantia que algo semelhante não possa voltar a ocorrer a qualquer momento.

1.2 Desafios atuais

Para tornar populares soluções que nos garantam a confiança que depositamos em sistemas de computação, vários desafios devem ser vencidos:

- Como evitar, detectar e contornar *bugs* no projeto de hardware e software?
- Como gerenciar a altíssima complexidade dos sistemas atuais de computação construídos com dezenas de chips de milhões de transistores e com software de centenas de milhares de linhas de código?
- Como explorar paralelismo para aumentar o desempenho sem comprometer a qualidade dos resultados mesmo em caso de falha de um ou mais componentes do sistema?
- Como aproveitar novas tecnologias mais rápidas, baratas e eficientes (mas ainda não totalmente provadas e testadas) sem saber ainda seu comportamento em situações inesperadas sob falha ou sobrecarga?
- Como aproveitar, para aplicações críticas e para operação em tempo real, o modelo de sistemas distribuídos construídos sobre plataformas não confiáveis de redes, contornando os problemas de perdas de mensagens, particionamento de rede e intrusão de hackers?

- Como desenvolver computadores móveis e sistemas embarcados, garantindo confiabilidade e segurança nesses dispositivos, e assegurando simultaneamente baixo consumo de potência, sem recorrer as técnicas usuais de replicação de componentes que aumentam peso e volume?
- Finalmente, como conciliar alta confiabilidade e alta disponibilidade com as crescentes demandas por alto desempenho?

Todos esses desafios ainda permanecem sem uma solução ideal.

1.3 Falha, erro e defeito

Vários autores tem se ocupado da nomenclatura e conceitos básicos da área. Os conceitos apresentados aqui são os usados por grande parte da comunidade nacional [Web90] e foram derivados dos trabalhos de Laprie [Lapr85] e Anderson e Lee [AnLe81].

Estamos interessados no sucesso de determinado sistema de computação no atendimento da sua especificação. Um defeito (*failure*) é definido como um desvio da especificação. Defeitos não podem ser tolerados, mas deve ser evitado que o sistema apresente defeito. Define-se que um sistema está em estado errôneo ou em erro se o processamento posterior a partir desse estado pode levar a defeito. Finalmente define-se falha (ou falta) como a causa física ou algorítmica do erro.

Na figura 1 é mostrada uma simplificação, adotada nesse texto, para os conceitos de falha, erro e defeito. Falhas estão associadas ao universo físico, erros ao universo da informação e defeitos ao universo do usuário. Assim um chip de memória, que apresenta uma falha do tipo grudado-em-zero *(stuck-at-zero)* em um de seus bits (falha no universo físico), pode provocar uma interpretação errada da informação armazenada em uma estrutura de dados (erro no universo da informação) e como resultado o sistema pode negar autorização de embarque para todos os passageiros de um vôo (defeito no universo do usuário). É interessante observar que uma falha não necessariamente leva a um erro (aquela porção da memória pode nunca ser usada) e um erro não necessariamente conduz a um defeito (no exemplo, a informação de vôo lotado poderia eventualmente ser obtida a partir de outros dados redundantes da estrutura).

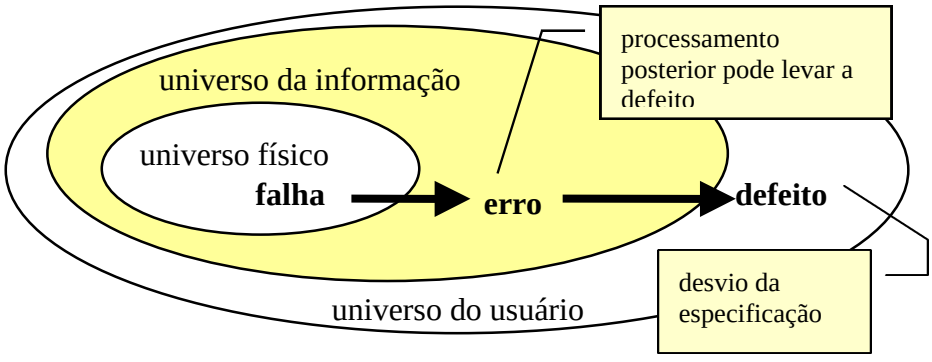


Figura 1: Modelo de 3 universos: falha, erro e defeito

Define-se latência de falha como o período de tempo desde a ocorrência da falha até a manifestação do erro devido aquela falha e latência de erro como o período de tempo desde a ocorrência do erro até a manifestação do defeito devido aquele erro.

Falhas são inevitáveis. Componentes físicos envelhecem e sofrem com interferências externas, seja ambientais ou humanas. O software, e também os projetos de software e hardware, são vítimas de sua alta complexidade e da fragilidade humana em trabalhar com grande volume de detalhes ou com deficiências de especificação.

Existem várias classificações para falhas na literatura técnica ([AnLe81], [Lapr85], [Web90], [Prad96]). As falhas aparecem geralmente classificadas em falhas físicas, aquelas de que padecem os componentes, e falhas humanas. Falhas humanas compreendem falhas de projeto e falhas de interação.

As principais causas de falhas são problemas de especificação, problemas de implementação, componentes defeituosos, imperfeições de manufatura, fadiga dos componentes físicos além de distúrbios externos como radiação, interferência eletromagnética, variações ambientais (temperatura, pressão, umidade) e também problemas de operação.

Além da causa da falha, para definir uma falha considera-se ainda:

- Natureza: falha de hardware, falha de software, de projeto, de operação, ...
- Duração ou persistência: permanente ou temporária
- Extensão: local a um módulo, global
- Valor: determinado ou indeterminado no tempo

Vem crescendo em freqüência aquelas falhas provocadas por interação humana maliciosa, ou seja, por aquelas ações que visam propositadamente provocar dados aos sistemas. Essas falhas e suas conseqüências são tratados por técnicas de *security* e não por técnicas de tolerância a falhas. Deve ser considerado, entretanto, que um sistema tolerante a falhas deve ser também seguro a intrusões e ações maliciosas.

Falhas de software e também de projeto são consideradas atualmente o mais grave problema em computação crítica. Sistemas críticos, tradicionalmente, são construídos de forma a suportar falhas físicas. Assim é compreensível que falhas não tratadas, e não previstas, sejam as que mais aborreçam, pois possuem uma grande potencial de comprometer a confiabilidade e disponibilidade do sistema. Um exame de estatísticas disponíveis [Lapr98] confirma essas considerações (tabela 1).

Sistemas tradicionais				**Redes cliente-servidor** (não tolerantes a falhas)	
Não tolerante a falhas		**Tolerante a falhas**			
Mean time to failure: 6 a 12 semanas Indisponibilidade após defeito: 1 a 4 h		Mean time to failure: 21 anos (Tandem)		Disponibilidade média: 98%	
Defeitos:		**Defeitos:**		**Defeitos:**	
hardware	50%	software	65%	projeto	60%
software	25%	operações	10%	operações	24%
comunicações / ambiente	15%	hardware	8%	físicos	16%
operações	10%	ambiente	7%		

Tabela 1 - Causas usuais de defeitos em sistemas de computação

1.4 Dependabilidade

O objetivo de tolerância a falhas é alcançar dependabilidade (tabela 2). O termo dependabilidade é uma tradução literal do termo inglês *dependability*, que indica a qualidade do serviço fornecido por um dado sistema e a confiança depositada no serviço fornecido. Principais medidas de dependabilidade [Prad96] são confiabilidade, disponibilidade, segurança de funcionamento (*safety*), segurança (*security*), mantenabilidade, testabilidade e comprometimento do desempenho (*performability).*

Dependabilidade *(dependability)*	qualidade do serviço fornecido por um dado sistema
Confiabilidade *(reliability)*	capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período
Disponibilidade (*availability*)	probabilidade do sistema estar operacional num instante de tempo determinado; alternância de períodos de funcionamento e reparo
Segurança *(safety)*	probabilidade do sistema ou estar operacional e executar sua função corretamente ou descontinuar suas funções de forma a não provocar dano a outros sistema ou pessoas que dele dependam
Segurança (*security*)	proteção contra falhas maliciosas, visando privacidade, autenticidade, integridade e irrepudiabilidade dos dados

Tabela 2 - Atributos de dependabilidade

Confiabilidade é a medida mais usada em sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis ou em sistemas em que reparo é impossível. Exemplos são aviação e exploração espacial. Confiabilidade é uma medida probabilística, pois falhas são um fenômeno aleatório. Confiabilidade não pode ser confundida com disponibilidade. Um sistema pode ser altamente disponível mesmo apresentando períodos de inoperabilidade, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço.

Outras qualidades importantes de um sistema são: comprometimento do desempenho (*performability),* mantenabilidade e testabilidade. *Performability* está relacionada à queda de desempenho provocado por falhas, onde o sistema continua a operar, mas degradado em desempenho. Mantenabilidade significa a facilidade de realizar a manutenção do sistema, ou seja, a probabilidade que um sistema com defeitos seja restaurado a um estado operacional dentro de um período determinado. Restauração envolve a localização do problema, o reparo físico e a colocação em operação. Finalmente testabilidade é a capacidade de testar certos atributos internos ao sistema ou facilidade de realizar certos testes. Quanto maior a testabilidade, melhor a mantenabilidade, por conseqüência menor o tempo de indisponibilidade do sistema devido a reparos.

1.5 Medidas de confiabilidade

As medidas de confiabilidade mais usadas na prática são: taxa de defeitos, MTTF, MTTR, MTBF. A tabela 3 mostra uma definição informal dessas medidas. Os fabricantes devem fornecer essas medidas para os seus produtos, tanto para os componentes eletrônicos, como para os sistemas de computação mais complexos. Essas medidas são determinadas pelo fabricante estatisticamente, observando o comportamento dos componentes e dispositivos fabricados.

Taxa de defeitos - failure rate, hazard function, hazard rate	número esperado de defeitos em um dado período de tempo, assumido um valor constante durante o tempo de vida útil do componente.
MTTF - mean time to failure	tempo esperado até a primeira ocorrência de defeito
MTTR - mean time to repair	tempo médio para reparo do sistema
MTBF - mean time between failure	tempo médio entre as falhas do sistema

Tabela 3 - Medidas de confiabilidade

A taxa de defeitos de um componente é dada por falhas por unidade de tempo e varia com o tempo de vida do componente.

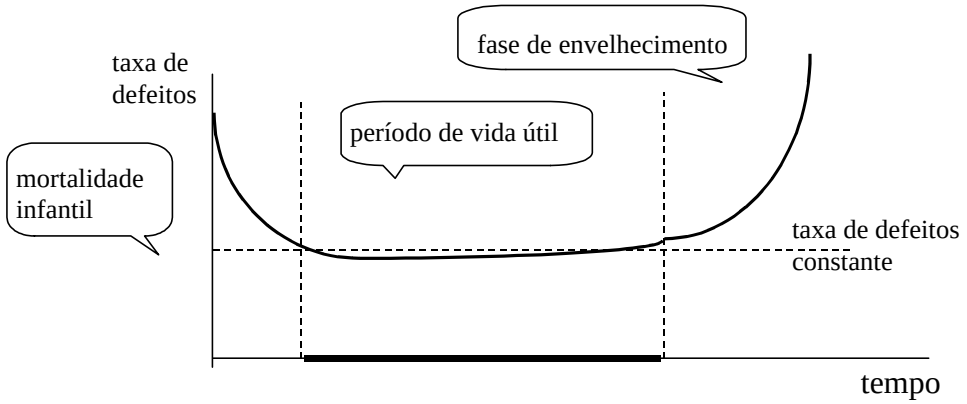


Figura 2 - Curva da banheira

Uma representação usual para a taxa de defeitos de componentes de hardware é dada pela curva da banheira (figura 2). Na figura 2 podem se distinguir 3 fases:

- mortalidade infantil: componentes fracos e mal fabricados
- vida útil: taxa de falhas (defeitos) constante
- envelhecimento: taxa de falhas crescente

Os componentes de hardware só apresentam taxa de defeitos constante durante um período de tempo chamado de vida útil, que segue uma fase com taxa de falhas decrescente chamada de mortalidade infantil. Para acelerar a fase de mortalidade infantil, os fabricantes recorrem a técnicas de *burn-in*, onde é efetuada a remoção de componentes fracos pela colocação dos componentes em operação acelerada antes de colocá-los no mercado ou no produto final.

É questionável se a curva da banheira pode ser aplicada também para componentes de software. Pode ser observado, no entanto, que os componentes de software também apresentam uma fase de mortalidade infantil ou taxa de erros alta no início da fase de testes, que decresce rapidamente até a entrada em operação do software. A partir desse momento, o software apresenta um taxa de erros constante até que, eventualmente, precise sofrer alguma alteração ou sua plataforma de hardware se torne obsoleta. Nesse momento, a taxa de erros volta a crescer. Intencionalmente mencionamos taxa de erros para software e não defeitos, porque erro é o termo usualmente empregado quando se trata de programas incorretos.

2 Técnicas para alcançar dependabilidade

No desenvolvimento de um sistema com os atributos de dependabilidade desejados, um conjunto de métodos e técnicas devem ser empregadas. Esses métodos e técnicas são classificados conforme a tabela 3.

prevenção de falhas	impede a ocorrência ou introdução de falhas. Envolve a seleção de metodologias de projeto e de tecnologias adequadas para os seus componentes
tolerância a falhas	fornece o serviço esperado mesmo na presença de falhas. Técnicas comuns: mascaramento de falhas, detecção de falhas, localização, confinamento, recuperação, reconfiguração, tratamento
validação	remoção de falhas, verificação da presença de falhas
previsão de falhas	estimativas sobre presença de falhas e estimativas sobre conseqüências de falhas

Tabela 3 - Técnicas para alcançar dependabilidade

2.1 Tolerância a falhas

Prevenção e remoção de falhas não são suficientes quando o sistema exige alta confiabilidade ou alta disponibilidade. Nesses casos o sistema deve ser construído usando técnicas de tolerância a falhas. Essas técnicas garantem funcionamento correto do sistema mesmo na ocorrência de falhas e são todas baseadas em redundância, exigindo componentes adicionais ou algoritmos especiais. Tolerância a falhas não dispensa as técnicas de prevenção e remoção. Sistemas construídos com componentes frágeis e técnicas inadequadas de projeto não conseguem ser confiáveis pela simples aplicação de tolerância a falhas.

O conceito de tolerância a falhas foi apresentado originalmente por Avizienis em 1967 [Aviz98]. Entretanto estratégias para construção de sistemas mais confiáveis já eram usadas desde a construção dos primeiros computadores [VonN56]. Apesar de envolver técnicas e estratégias tão antigas, a tolerância a falhas ainda não é uma preocupação rotineira de projetistas e usuários, ficando sua aplicação quase sempre restrita a sistemas críticos.

As técnicas de tolerância a falhas são de duas classes disjuntas:

- mascaramento ou
- detecção, localização e reconfiguração

Na primeira classe, mascaramento, falhas não se manifestam como erros, pois são mascaradas na origem. A primeira classe geralmente emprega mais redundância que a segunda e, por não envolver os tempos gastos para as tarefas de detecção, localização e reconfiguração, é a preferida para sistemas de tempo real críticos.

2.2 Fases de aplicação das técnicas de tolerância a falhas

Vários autores apresentaram suas próprias classificações para as técnicas de tolerância a falhas. A mais comum é a classificação em 4 fases de aplicação [AnLe81]: detecção, confinamento, recuperação e tratamento. Essas fases excluem mascaramento de falhas, que é uma técnica a parte, não usada em complemento às demais.

Fases	**Mecanismos**
detecção de erros	replicação testes de limites de tempo cão de guarda (watchdog timers) testes reversos codificação: paridade, códigos de detecção de erros, Hamming, ... teste de razoabilidade, de limites e de compatibilidades testes estruturais e de consistência diagnóstico
confinamento e avaliação	ações atômicas operações primitivas auto encapsuladas isolamento de processos regras do tipo tudo que não é permitido é proibido hierarquia de processos controle de recursos
recuperação de erros	técnicas de recuperação por retorno (*backward error recovery)* técnicas de recuperação por avanço (*forward error recovery)*
tratamento da falha	diagnóstico reparo

Tabela 4 - Quatro fases de Anderson e Lee

As fases envolvem o conceito de uma seqüência complementar de atividades, que devem ser executadas após a ocorrência de uma ou mais falhas. Alguns autores consideram as fases extras de diagnóstico e mascaramento. Nós preferimos considerar o mascaramento de falhas como uma classe a parte, não complementar às fases citadas. Diagnóstico, por outro lado pode ser usado tanto como um mecanismo usado nas fases de detecção de erros e de localização de falhas, como uma técnica isolada conduzida periodicamente para diminuir a latência.

A primeira fase de Anderson e Lee é a de detecção de um erro. Uma falha primeiro se manifesta como um erro, para então ser detectada por alguns dos mecanismos listados na tabela 4. Antes da sua manifestação como erro, a falha está latente e não pode ser detectada, eventualmente a falha pode permanecer no sistema durante toda a sua vida útil sem nunca se manifestar, ou seja, sem nunca levar o sistema a um estado errôneo.

Devido a latência da falha, após a ocorrência da falha até o erro ser detectado, pode ter ocorrido espalhamento de dados inválidos. O confinamento estabelece limites para a propagação do dano, mas depende de decisões de projeto; os sistemas por sua natureza não provêm confinamento. Durante o projeto devem ser previstas e implementadas restrições ao fluxo de informações para evitar fluxos acidentais e estabelecer interfaces de verificação para detecção de erros.

A recuperação de erros ocorre após a detecção e envolve a troca do estado atual incorreto para um estado livre de falhas. Pode ser de duas formas (tabela 5): técnicas de recuperação por retorno (*backward error recovery)* e técnicas de recuperação por avanço (*forward error recovery*.

Técnica	**Definição**	**Características**	**Implementação**
forward error recovery	condução a novo estado ainda não ocorrido desde a última manifestação de erro	eficiente, mas danos devem ser previstos acuradamente	específica a cada sistema
backward error	condução a estado anterior	alto custo, mas de aplicação genérica	pontos de verificação (checkpoints), pistas de

recovery			auditoria (audit trails), ...

Tabela 5 - Técnicas de recuperação

A recuperação é simples para um sistema com um único processo, mas é complexa em processamento distribuído [Jans97]. A recuperação nesses sistemas, usualmente de retorno pode provocar efeito dominó. Ao desfazer a computação, um processo deixa algumas mensagens órfãs na rede. Processos que receberam e incorporaram essas mensagens devem por sua vez desfazer também a computação realizada, provocando que, eventualmente, outros processos, que receberam suas mensagens, agora órfãs, também tenham que desfazer suas computações. O efeito pode atingir todos os processos de um sistema e provocar o retorno ao início do processamento. Uma solução para esse problema é impor restrições a comunicação entre os processos.

Técnicas de recuperação por retorno não são adequadas a sistemas de tempo real. Nesses sistemas deve ser usada recuperação por avanço.

A última fase, tratamento de falhas, consiste em:

- localizar a origem do erro (falha),
- localizar a falha de forma precisa,
- reparar a falha,
- recuperar o restante do sistema.

Nessa fase geralmente é considerada a hipótese de falha única, ou seja, uma única falha ocorrendo a cada vez. A localização da falha é realizada em duas etapas: localização grosseira (módulo ou subsistema) e rápida e localização fina, mais demorada, onde o componente falho é determinado. Para os dois tipos diagnóstico é usado. O diagnóstico é um teste com comparação dos resultados gerados com os resultados previstos. Pode ser conduzido no sistema de forma manual ou automática:

- manual - executado por um operador local ou remoto,
- automático - executado pelos os componentes livres de falha.

Após a localização, a falha é reparada através da remoção do componente danificado. Também o reparo pode ser manual ou automático. O reparo automático pode envolver degradação gradual, ou seja, uma reconfiguração para operação com menor número de componentes, ou a substituição imediata por outro componente disponível no sistema. Substituição automática é usada em sistemas com longo período de missão sem possibilidade de reparo manual, como por exemplo em sondas espaciais e satélites.

2.3 Mascaramento de falhas

O mascaramento de falhas garante resposta correta mesmo na presença de falhas. A falha não se manifesta como erro, o sistema não entra em estado errôneo e, portanto, erros não precisam ser detectados, confinados e recuperados. Entretanto, em caso de falhas permanentes, a localização e o reparo da falha ainda são necessários. A tabela 6 mostra mecanismos usuais para implementação de mascaramento de falhas. Alguns desses mecanismos serão vistos em detalhes na próxima seção.

mascaramento de falhas	redundância de hardware (replicação de componentes) codificação: ECC (código de correção de erros) diversidade, programação diversitária (n-versões) blocos de recuperação

Tabela 6 - Mecanismos para mascarar falhas

3 Redundância

Redundância é a palavra mágica em tolerância a falhas. Redundância para aumentar confiabilidade é quase tão antiga como a história dos computadores ([Crev56], [VonN56]). Todas as técnicas de tolerância a falhas envolvem alguma forma de redundância. Redundância está tão intimamente relacionada a tolerância a falhas que na indústria nacional o termo usado para designar um sistema tolerante a falhas é sistema redundante. Redundância para tolerância a falhas pode aparecer de várias formas:

- Redundância de hardware
- Redundância de software
- Redundância de informação
- Redundância de tempo

As técnicas de redundância de hardware e de software são apresentadas nas próximas seções. Redundância de informação é provida por códigos de correção de erros, como ECC. ECC (*error correction code*) está sendo exaustivamente usado em memórias e em transferência entre memórias e processadores. Como a codificação da informação implica no aumento do número de bits, e os bits adicionais não aumentam a capacidade de representação de dados do código, é fácil perceber a razão da codificação também ser considerada uma forma de redundância.

Redundância temporal repete a computação no tempo. Evita custo de hardware adicional, aumentando o tempo necessário para realizar uma computação. É usada em sistemas onde tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

- detecção de falhas transientes: repetindo a computação, resultados diferentes são uma forte indicação de uma falha transitória. Essa estratégia não é adequada para falhas permanentes, porque os resultados repetidos nesse caso serão iguais.
- detecção de falhas permanentes: repete-se a computação com dados codificados e decodifica-se o resultado antes da comparação com o resultado anterior. Essa estratégia provoca a manifestação de falhas permanentes. Por exemplo, considere um barramento com uma linha grudada em zero. Nas duas computações, uma transferência com dado normal; e a segunda com dado invertido, por exemplo, a linha vai transferir sempre zero.

Todas essas formas de redundância, de hardware, de software, temporal de informação, tem algum impacto no sistema, seja no custo, no desempenho, na área (tamanho, peso), ou na potência consumida. Assim, apesar de ser a solução "mágica" para tolerância a falhas, o uso de redundância em qualquer projeto deve ser bem ponderada.

Redundância tanto serve para detecção de falhas, como para mascaramento de falhas. O grau de redundância em cada caso é diferente. Para mascarar falhas são necessários

mais componentes do que para detectar falhas. Por exemplo, para detectar falhas em um microprocessador é necessário outro microprocessador idêntico, sincronizado ao primeiro, além de um comparador de sinais na saída de ambos. Qualquer diferença na comparação indica que o par de microprocessadores está em desacordo, e que portanto um dos dois está falho. Entretanto está falha não pode ser mascarada. O resultado da comparação não indica quais as saídas são as corretas.

3.1 Redundância de hardware

Redundância de hardware está baseada na replicação de componentes.

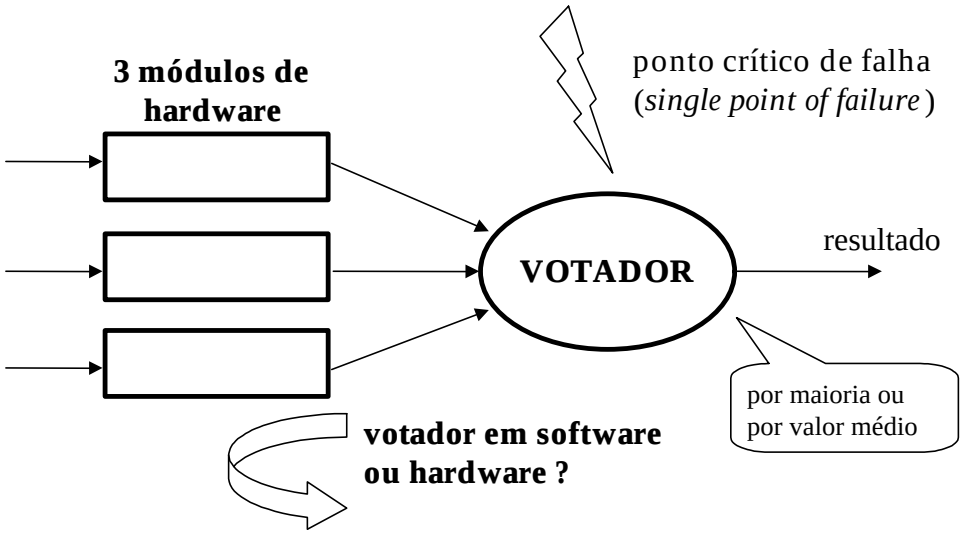
redundância de hardware	**técnicas**	**vantagens**
redundância **passiva** ou **estática**	mascaramento de falhas	não requer ação do sistema, não indica falha
redundância **ativa** ou **dinâmica**	detecção, localização e recuperação	substituição de módulos, usada em aplicações de longa vida
redundância **híbrida**	combinação de ativa e passiva	usada para garantir mascaramento e longa vida; geralmente de alto custo

Tabela 7 - Redundância de hardware

3.1.1 Redundância de hardware passiva

Na redundância de hardware passiva os elementos redundantes são usados para mascarar falhas. Todos os elementos executam a mesma tarefa e o resultado e determinado por votação. Exemplos são TMR (*triple modular redundancy)* e NMR.

TMR, redundância modular tripla, é uma das técnicas mais conhecidas de tolerância a falhas. TMR mascara falha em um componente de hardware triplicando o componente e votando entre as saídas para determinação do resultado (figura 3). A votação pode ser



por maioria (2 em 1) ou votação por seleção do valor médio [LyVa62].

Figura 3 - TMR

O votador realiza uma função simples, fácil de verificar a correção. É interessante observar que o votador não tem a função de determinar qual o módulo de hardware discorda da maioria. Se esse função for necessária no sistema, ela deve ser realizada por um detector de falhas.

Apesar de simples, o votador, por estar em série com os módulos de hardware e ter a responsabilidade de fornecer o resultado, é o ponto crítico de falhas no esquema TMR. Se o votador apresentar baixa confiabilidade, todo o esquema vai ser frágil, tão frágil como o votador.

Soluções para contornar a fragilidade do votador são:

- Construir o votador com componentes de alta confiabilidade.
- Triplicar o votador.
- Realizar a votação por software.

A figura 4 mostra um esquema com votador triplicado. Os 3 resultados gerados podem, nesse caso, ser levados aos próximos 3 módulos de hardware do sistema.

Deve ser observado que a redundância aumenta o número de componentes do sistema. Quando maior o número de componentes, maior possibilidade de falha.

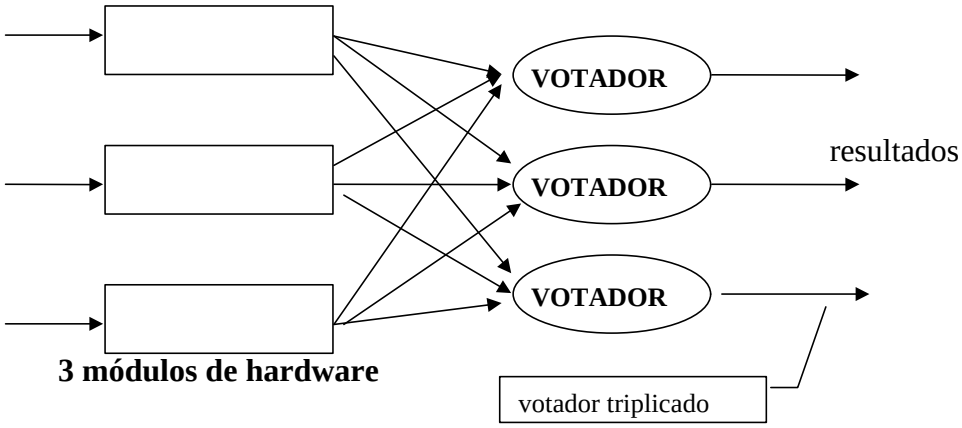


Figura 4 - TMR com votador triplicado

TMR apresenta uma confiabilidade maior que um sistema de um único componente até a ocorrência da primeira falha permanente (figura 5). Depois disso, perde a capacidade de mascarar falhas e vai apresentar uma confiabilidade menor que um sistema de um único componente. Com o tempo, quando aumenta a probabilidade de componentes falharem (ou seja aumenta a taxa de defeitos) TMR apresenta uma confiabilidade pior do que um sistema não redundante. TMR é ideal então para períodos não muito longos de missão. Suporta uma falha permanente apenas e é ideal para falhas temporárias, uma de cada vez.

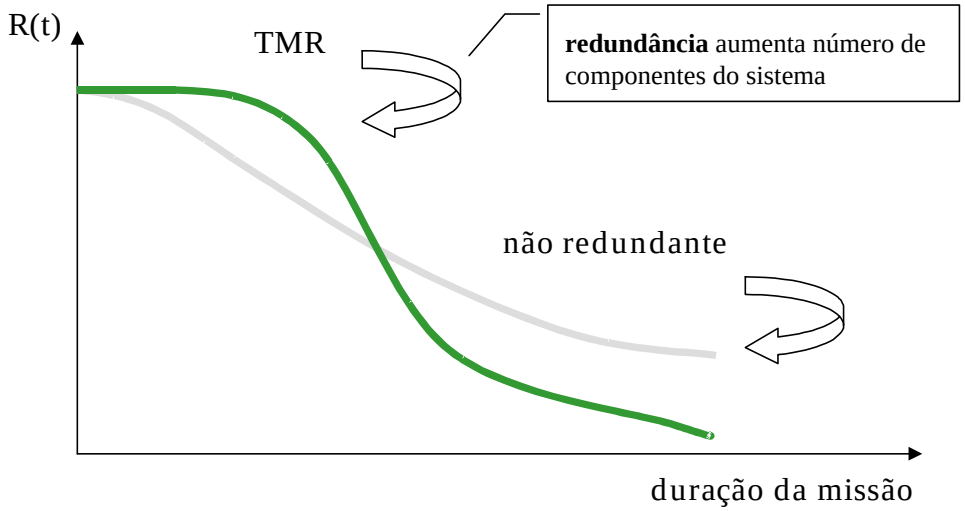


Figura 5 - Confiabilidade de TMR

Redundância modular múltipla, NMR, é a generalização do conceito de TMR. Ou seja, TMR é um caso especial de NMR. O computador de bordo do Space Shuttle é um exemplo de NMR, com n igual a 4 e votação por software.

3.1.2 Redundância dinâmica

Na redundância dinâmica ou ativa, tolerância a falhas é provida pelo emprego das técnicas de detecção, localização e recuperação. A redundância empregada neste caso não provê mascaramento.

Redundância dinâmica é usada em aplicações que suportam permanecer em um estado errôneo durante um curto período de tempo, tempo esse necessário para a detecção do erro e recuperação para um estado livre de falhas. Um exemplo de implementação de redundância dinâmica é através de módulos estepe *(standby sparing)*. Módulos estepe podem ser operados de duas formas conforme tabela 8.

Operação de módulos estepe	**comentários**
alimentados (*hot standby*)	Módulo estepe conectado eletricamente ao sistema. Minimiza a descontinuidade do processamento durante o chaveamento entre os módulos.
não alimentados (*cold standby*)	Módulo estepe só começa a operar quando conectado. Aumenta a vida útil dos módulos estepe.

Tabela 8 - Exemplo de redundância dinâmica

3.2 Redundância de software

A simples replicação de componentes idênticos é uma estratégia inútil em software. Componentes idênticos de software vão apresentar erros idênticos. Assim não basta copiar um programa e executá-lo em paralelo ou executar o mesmo programa duas vezes. Exemplos de outras formas de redundância em software:

- diversidade (ou programação n-versões)

- blocos de recuperação
- verificação de consistência

É interessante lembrar que se o software foi projetado corretamente desde o início, então não são necessárias técnicas de tolerância a falhas para software. Infelizmente estamos longe de poder garantir, na prática, programas corretos.

3.2.1 Diversidade

Diversidade, também chamada programação diversitária, é uma técnica de redundância usada para obter tolerância a falhas em software ([ChAv78], [AvKe84], [WeWe89]). A partir de um problema a ser solucionado são implementadas diversas soluções alternativas, sendo a resposta do sistema determinada por votação (figura 6).

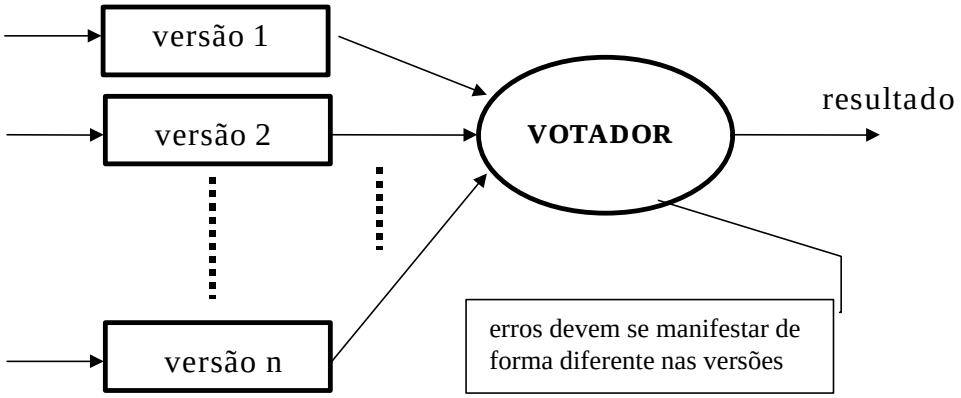


Figura 6 - Programação n-versões

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 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.

Um exemplo de diversidade é o sistema de computadores de bordo do Space Shutle. Quatro computadores idênticos são usados em NMR. Um quinto computador, diverso em hardware e em software dos outros quatro, pode substituir os demais em caso de colapso do esquema NMR [Prad96].

3.2.2 Blocos de recuperação

Semelhante a programação a n-versões, mas nessa técnica programas secundários só serão necessários na detecção de um erro no programa primário. Essa estratégia envolve um teste de aceitação (figura 7). Programas são executados e testados um a um até que o primeiro passe no teste de aceitação. A estratégia de blocos de recuperação tolera n-1 falhas, no caso de falhas independentes nas n versões.

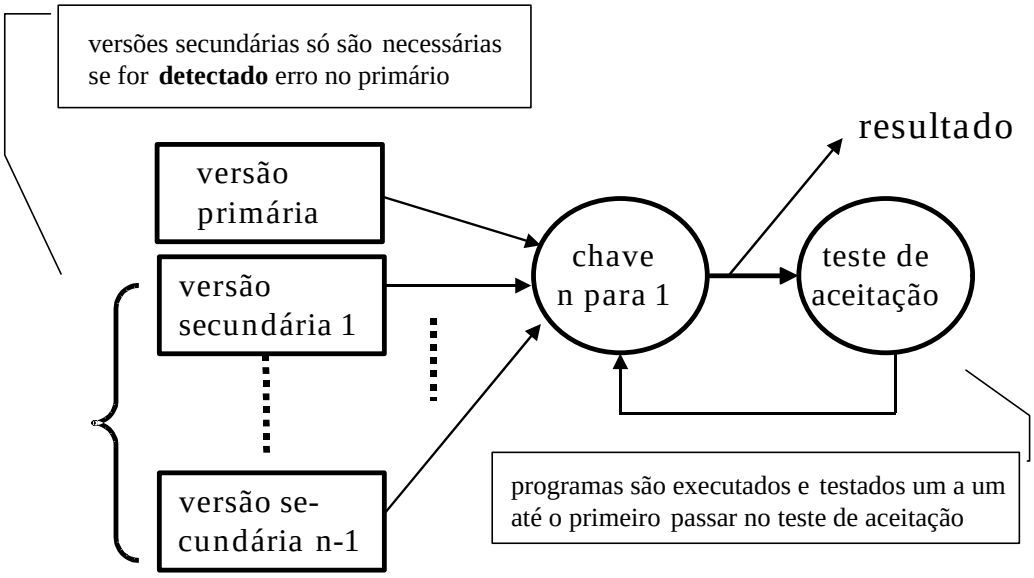


Figura 7 - blocos de recuperação

4 Aplicações de Sistemas Tolerantes a Falhas

As técnicas apresentadas nas seções anteriores podem sugerir a possibilidade da construção de um sistema perfeitamente confiável e permanentemente disponível. Infelizmente, estas técnicas apenas aumentam a dependabilidade de um sistema, não fornecendo nenhuma garantia de que todas as falhas possíveis possam ser toleradas. A

redundância inerente à tolerância a falhas aumenta consideravelmente o custo de um sistema de computação. Só esse acréscimo de custo já implica no estabelecimento de fronteiras claras ao emprego indiscriminado de técnicas de tolerância a falhas.

Para a escolha adequada de um sistema de computação tolerante a falhas, as características especiais da aplicação e as suas exigências quanto a confiabilidade e disponibilidade devem ser conhecidas em detalhe. Só obedecendo criteriosamente às exigências essenciais de cada aplicação torna-se possível contornar o custo associado ao emprego de técnicas de tolerância a falhas.

4.1 Áreas de Aplicação

As áreas tradicionais onde são empregados sistemas tolerantes a falhas são:

- aplicações críticas de sistemas de tempo real como medicina, controle de processos e transportes aéreos
- aplicações seguras de tempo real como transportes urbanos;
- aplicações em sistemas de tempo real de longo período de duração sem manutenção, como em viagens espaciais, satélites e sondas;
- aplicações técnicas como telefonia
- aplicações comerciais como sistemas de transação e redes locais.

Essas áreas não abrangem todo o universo de aplicações onde tolerância a falhas pode ser empregada com vantagens para o usuário de sistemas de computação. Exigências quanto a disponibilidade e confiabilidade são encontradas em qualquer área. Usuários, que inicialmente se mostram satisfeitos em contar apenas com a simples automação de serviços, logo passam a desejar que esses serviços sejam prestados corretamente e sem interrupções. Sistemas tolerantes a falhas são caros e portanto empregados apenas naquelas situações em que a sua não utilização acarretaria prejuízos irrecuperáveis.

No mercado brasileiro as áreas tradicionais de aplicação de tolerância a falhas são telefonia e pesquisas espaciais. A Telebrás e o INPE são exemplos de instituições que vêm desenvolvendo trabalhos de pesquisa no sentido de gerar tecnologia nacional em tolerância a falhas para as áreas de telefonia e pesquisas espaciais respectivamente.

4.2 Sistemas de tempo real

Sistemas de computação de tempo real são aplicados no controle, supervisão e automação (controle de processos industriais, transportes, medicina) e em comunicação. Condições de para aplicação desses sistemas são:

- Disponibilidade de apenas um curto intervalo de tempo para reconhecimento de erros, de forma a não prejudicar o processamento em tempo real;
- Impossibilidade de emprego de recuperação por retorno, uma vez que eventos passados não são reproduzíveis em sistemas de tempo real;
- Exigência de redundância estática para garantir a continuidade do processamento em tempo real em caso de falha de um componente;

- Comportamento livre de falhas (*fail-safe*), ou seja, em caso de ocorrência de uma falha não mascarável, o sistema deve ir imediatamente para um estado seguro.

Como exemplos de sistemas de tempo real tolerantes a falhas podemos citar os sistemas FTMP [HSL78] e SIFT [Wens78], que foram especificados pela NASA para serem usados como computadores de bordo, e o sistema August System: Series 300 [Wens83] derivado do computador SIFT. O August 300, ao contrário de SIFT, não é um computador de bordo, mas uma máquina para controle de processos em tempo real. O sistema de computação de bordo do Space Shuttle é outro exemplo de sistema tempo real tolerante a falhas inspirado no SIFT.

4.3 Sistemas digitais de telefonia

Sistemas eletrônicos para telefonia, devido a rigorosas exigências quanto à disponibilidade, tradicionalmente empregam técnicas de tolerância a falhas. Sistemas de telefonia devem apresentar alta disponibilidade. É especificado um tempo máximo em falha não maior que 2 horas em 30 anos. Junto ao uso de componentes de alta qualidade e longa vida, requisitos para a aplicação nessa área são:

- Detecção e localização automática de erros tanto de software como de hardware;
- Tratamento automático de erros por reconfiguração do sistema;
- Isolamento e substituição de componentes faltosos durante o período de operação normal do sistema.

A principal técnica de tolerância a falhas, presente nos processadores em sistemas telefônicos, tem sido duplicação de componentes. Duplicação é usada para substituição de componentes com falhas permanentes e também para detecção de erros, onde então duas unidades executam de forma sincronizada e um comparador verifica os resultados. Além disso é também usada uma grande variedade de técnicas de tolerância a falhas, como códigos de detecção e correção de erros, memória duplicada, temporizadores para detecção de time-out, componentes estepe, auto-teste, canal de manutenção, ...

Os mais antigos exemplos são os computadores ESS da AT&T [Toy78], no mercado desde 1965. Inicialmente o sistema ESS era formado por simples controladores de tempo real dedicados; atualmente são complexos sistemas de propósito geral combinando hardware e software sofisticado. Todos computadores ESS são, entretanto, construídos usando a técnica de duplicação-comparação.

Outros exemplos de computadores para telefonia são os sistemas EDS e o SSP113, da Siemens. O SSP113 é uma máquina multiprocessadora construída com microprocessadores de 32 bits e, como os demais exemplos citados, emprega exaustivamente duplicação de elementos.

4.4 Sistemas de transações

Um número significativo de aplicações comerciais são realizados sob forma de processamento de transações. Esse processamento implica na existência de uma base de dados comum usada interativa e concorrentemente por um grande número de usuários através de terminais de acesso. Exemplos são bancos de dados para aplicações

financeiras, bancarias e de bolsa de valores, e para reserva internacional de vôos. Requisitos indispensáveis para aplicações nessa área são:

- Garantia de integridade e consistência de dados na base de dados;
- Alta disponibilidade para o processamento contínuo de transações.

Adicionalmente aos requisitos citados, são características desejáveis:

- Tratamento automático de erros no processamento de transações e de erros de hardware, sem interrupção do funcionamento normal;
- Reconfiguração tanto de hardware como de software, sem interrupção do processamento normal.

Integridade e consistência de dados são os requisitos mais importantes. Disponibilidade assim como tratamento de erros e reconfiguração sem interrupção do funcionamento podem até ser sacrificados, se for para garantir a correção na base de dados.

A maioria dos sistemas comerciais de transações operam baseados no modelo *fail-stop*. Em caso de erro, o sistema interrompe o processamento evitando assim a propagação do erro com conseqüente dano à base de dados.

A partir da introdução dos sistemas NonStop [Katz78] no final da década de 70, a firma Tandem Computers (fundada em 1976) foi durante muito tempo líder do mercado de sistemas tolerantes a falha comerciais. Forte concorrente para a Tandem foi a Stratus Computers [Hend83], fundada em 1980, com o sistema Continuous Processing. Modelos desse sistema foram também comercializados por outros fornecedores, com os nomes de Olivetti CPS32 e IBM/88. Tandem e Stratus são também dois bons exemplos da aplicação de mecanismos de tolerância a falhas implementados em software (Tandem) e em hardware (Stratus). Stratus, por implementar tolerância a falhas em hardware e assim tornar os mecanismos transparentes às aplicações, acabou por suplantar a Tandem como líder no mercado.

Outros exemplos de computadores comerciais de transações de alta disponibilidade são: VAX 8600, IBM 3090, VAXft 3000 (1990), Teradata (computadores baseados na família Intel 80x86) e Sequoia (baseados na família Motorola 68000).

4.5 Redes de serviço locais

Redes locais são formadas por estações de trabalho e por um ou mais servidores de rede que se comunicam por trocas de mensagens. O servidor é responsável por serviços de suporte e controle da rede local como armazenamento de dados e arquivos, gerenciamento de documentos, impressão de documentos e conexão a outras redes.

Redes locais se tornaram, nos últimos tempos, a plataforma de computação mais popular em corporações, centros de pesquisa e universidades, substituindo com vantagens mainframes e terminais. É notável sua aceitação também e também em automação industrial.

Requisitos quanto a tolerância a falhas para aplicações nessa área são semelhantes aos requisitos para sistemas de transação:

- Garantia de integridade e redundância de dados;
- Alta disponibilidade do servidor para prover continuidade de serviços na rede;

- Tratamento automático de erros de hardware e no serviço da rede;
- Reconfiguração da rede em caso de erro e estabelecimento de uma nova configuração com a entrada de outras estações (clientes e servidores), sem interrupção do processamento normal.

Com o aumento de popularidade das redes e com o crescimento do comércio eletrônico o mercado para servidores de redes tolerantes a falhas aumentou consideravelmente. Praticamente todos os fabricantes de computadores oferecem servidores de redes tolerantes a falhas, entre eles se destacam Sun Microsystems (com a série Enterprise), Compac (com tecnologia Tandem), e Stratus.

Os servidores comerciais são baseados em redundância de componentes de hardware, troca de módulos sem necessidade de desligar o sistema, espelhamento de disco ou RAID e sistemas operacionais convencionais como UNIX.

Um exemplo é linha Sun Enterprise X500, com altos índices de disponibilidade e confiabilidade [http://www.sun.com].

Algumas características gerais presentes nos servidores Sun Enterprise são mostradas na tabela 8.

Reconfiguração automática	possibilita ao sistema reiniciar imediatamente após um defeito, desabilitando automaticamente o componente falho. Envolve testes nos vários componentes do sistema a cada vez que o equipamento é ligado ou quando é gerada uma interrupção externa.
Fontes de energia e ventilação redundantes	defeitos nesses componentes não interrompem a operação do sistema
ECC para garantir integridade dos dados	ECC (*error correction code*) usado em todos os caminhos de dados. Adicionalmente, paridade é usada nos endereços e linhas de controle do barramento e também nas caches interna e externa dos processadores.
Monitoramento ambiental	para proteger o sistema contra defeitos causados por temperaturas extremas, pela falta de fluxo de ar através do sistema ou devido a flutuações de energia.
Ferramentas de monitoramento avançadas	vários registradores e contadores que são usados para monitorar a atividade do sistema e fornecer mecanismos de cão de guarda (*watchdog*) em hardware. Os registradores são acessados por software especial, como o Solstice SyMON, para extração de dados relativos a performance e manutenção.
Ferramentas de diagnóstico *online*	*Sun Validation Test Suite* (SunVTS) realiza testes e registra os resultados obtidos. Os testes podem ser realizados tanto em componentes (processadores, memória, I/O,…), como no sistema como um todo
Troca de componentes com o sistema em operação	habilidade em substituir componentes sem necessidade de desligar o sistema (*hot swap*). Módulos de energia/ventilação, placas de CPU/memória e placas de I/O podem ser instalados e removidos com sistema em operação norma;.

Tabela 8 - Características da série Sun Enterprise

4.6 Sistemas seguros

Por sistemas seguros são designados os sistemas em que segurança é mais importante que disponibilidade. Exemplos típicos são sistemas de transportes urbanos, que devem

apresentar um comportamento *fail-safe*. Em caso de erro o sistema deve ir imediatamente para um estado seguro. É relativamente fácil parar trens para evitar colisões. O estado parado é um estado seguro para transporte urbano.

Requisitos quanto a tolerância a falhas para aplicações nessa área são aproximadamente semelhantes aos requisitos para sistemas de tempo real, uma vez que o controle desse tipo de sistema também ocorre em tempo real:

- Existência de um estado seguro e facilidade de alcançá-lo em caso de erro;
- Rapidez no reconhecimento de erros;
- Redundância para mascaramento e para reconhecimento de erros.

Para aplicações em transportes urbanos como bondes, trens subterrâneos e de superfície, utilizados largamente na Europa, foi desenvolvida na Alemanha uma família de microprocessadores, SIMIS [Goer89], que facilita a implementação de sistemas seguros. Todos os componentes são construídos para operar de forma duplicada. Em caso de discordância de qualquer saída, qualquer processamento ou distribuição de sinal posterior é bloqueado, e os sinais de saída do sistema são colocados em zero. O sistema é projetado para, quando ocorrer o bloqueio dos componentes, ir imediatamente para um estado seguro.

4.7 Exemplos por área de aplicação

A tabela 9 resume as áreas de aplicação apresentadas anteriormente citando alguns exemplos por área. Essa lista de aplicações e exemplos não é exaustiva. Cresce dia a dia o número de aplicações de sistemas de computação onde disponibilidade e confiabilidade são exigidas em alto grau. Os usuários de sistemas de computação estão se tornando mais exigentes quanto a segurança e se mostram um pouco mais dispostos a enfrentar os custos adicionais das técnicas de tolerância a falhas.

Área de Aplicação	**Exemplos**
Tempo real	FTMP, SIFT, August 300, Space Shuttle
Telefonia	AT&T ESS, Siemens EDS, Ericson AXE, Siemens SSP113
Sistemas de transações	Tandem, Stratus (Olivetti CPS32 e IBM/88), VAXft, Teradata e Sequoia
Servidores de rede	Sun Enterprise, Stratus, Compac, Cluster NT
Sistemas seguros	SIMIS

Tabela 9 - Exemplos por área de aplicação

Convém ressaltar que mesmo para as áreas onde se dispõe de sistemas tolerantes a falhas, esses não se apresentam prontos para a imediata utilização. O desenvolvedor de software ou o usuário especializado desses sistemas deve sempre prover alguns recursos complementares para garantir tolerância a falhas na sua aplicação. Além disso, os sistemas comerciais geralmente só garantem tolerância a falhas isoladas de hardware. Mecanismos contra falhas múltiplas e mesmo falhas de software devem ser supridas pelo desenvolvedor.

O desenvolvedor deve, portanto, reconhecer exigências quanto a confiabilidade e disponibilidade de uma determinada aplicação, saber escolher o sistema de menor custo que supra essas exigências e ter condições de desenvolver os mecanismos

complementares de tolerância a falhas para atingir a confiabilidade desejada. Naturalmente esse é um desenvolvedor especialista, conhecedor de técnicas de tolerância a falhas e sua utilização eficiente.

5 Conclusão

O texto apresentou os conceitos básicos de tolerância a falhas e mostrou algumas áreas de aplicação de computadores tolerantes a falhas. Praticamente todos os exemplos citados toleram erros provocados por falhas de hardware. É entretanto fácil de imaginar que com a utilização de componentes cada vez mais confiáveis e sistemas de software cada vez mais complexos, erros que ocorram em sistemas de computação sejam devidos predominantemente a falhas de software. Essas falhas tanto podem estar localizadas no sistema operacional, nos programas aplicativos ou nos compiladores e interpretadores dos programas aplicativos.

Falhas em software podem ser contornadas por técnicas de tolerância a falhas específicas, como diversidade, ou por técnicas que evitam erros como verificação formal. É impossível prever qual dessas técnicas prevalecerá no futuro. É interessante observar que em muitos sistemas detecção de erros provocados por falhas de hardware, mascaramento, recuperação e reconfiguração são comandados por software. Nesses sistemas é essencial que esse software seja seguro, preferencialmente verificado quanto a correção.

Profissionais de computação devem encarar seriamente os problemas ocasionados por falhas não tratadas nos sistemas informatizados. Tolerância a falhas compreende muitas das técnicas que permitem aumentar a qualidade de sistemas de computação. Apesar da tolerância a falhas não garantir comportamento correto na presença de todo e qualquer tipo de falha e apesar das técnicas empregadas envolverem algum grau de redundância e, portanto, tornarem os sistemas maiores e mais caros, ela permite alcançar a confiabilidade e a disponibilidade desejadas para os sistemas computadorizados. Vários desafios ainda devem ser vencidos, tolerância a falhas não é uma área de pesquisa completamente dominada. Apesar de antiga é uma área onde muita aplicação, avaliação e popularização se fazem necessárias.

6 Bibliografia

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

[Aviz98] AVIZIENIS, A. Infraestructure-based design of fault-tolerant systems. In: Proceedings of the IFIP International Workshop on Dependable Computing and its Applications. DCIA 98, Johannesburg, South Africa, January 12-14, 1998. p. 51-69.

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

[Bir96] BIRMAN, K. *Building secure and reliable network applications*, 1996

[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.

[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.

[Goer89] GÖRKE, W. *Fehlertolerante Rechensysteme*. München, Oldenburg Verlag, 1989.

[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.

[Jal94] JALOTE, P. *Fault tolerance in distributed systems*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[Jans97] JANSCH-PORTO, I. E. S; WEBER, T. S. Recuperação em Sistemas Distribuídos. In: XVI Jornada de Atualização em Informática, XVII Congresso da SBC, Brasília, 2-8 agosto de 1997. anais. pp 263-310

[John84] JOHNSON, D. The Intel 432: a VLSI architecture for fault-tolerant computer systems. *Computer*, New York, 17(8):40-48, Aug. 1984.

[Katz78] KATZMAN, J. A. A fault-tolerant computing system. In: Hawaii International Conference of System Sciences, 1978, *Proceedings*. p. 85-102.

[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.

[Lapr98] LAPRIE, J. C.; Dependability: von concepts to limits. In: Proceedings of the IFIP International Workshop on Dependable Computing and its Applications. DCIA 98, Johannesburg, South Africa, January 12-14, 1998. p. 108-126.

[Liu84] LIU, T. S. The role of a maintenance processor for a general-purpose computer system. *IEEE Transations on Computers*. New York, c-33(6):507-517. June 1984.

[LyVa62] LYONS, R.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.

[Mul93] MULLENDE, S. *Distributed systems*. Addison-Wesley, ACM Press, New York, 1993.

[Prad96] PRADHAN, D. K., *Fault-Tolerant System Design*. Prentice Hall, New Jersey, 1996.

[RSNG82] REILLY, J; SUTTON, A.; NASSER, R.; GRISCOM, R. Processor controller for the IBM 3081. *IBM Journal of Research and Development*, 26(1):22-29. Jan. 1982.

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

[Toy78] TOY, W. N. Fault-tolerant design of local ESS processors.*Proceedings of the IEEE,* New York, 66(10):1126-1145, Oct. 1978.

[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.

[Web90] Weber, T.; Jansch-Pôrto, I.; Weber, R. *Fundamentos de tolerância a falhas*. Vitória: SBC/UFES, 1990. (apostila preparada para o IX JAI - Jornada de Atualização em Informática, no X Congresso da Sociedade Brasileira de Computação).

[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.

[Wens83] WENSLEY, J. H. Industrial-control system does things in threes for safety. *Electronics*, New York, 56(2):98-102. Jan. 1983.

[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.