Um roteiro para exploração dos conceitos básicos de tolerância a falhas

Taisy Silva Weber[1]
Instituto de Informática – UFRGS
Curso de Especialização em Redes e Sistemas Distribuídos
taisy@inf.ufrgs.br

Resumo

Em um ambiente distribuído suportado por infra-estrutura de rede de computadores, supõem-se que o sistema computacional opere apropriadamente, sem interrupção no seu serviço e sem perda de dados ou mensagens. No mundo ideal, sistemas computacionais são totalmente confiáveis e cem por cento disponíveis. No mundo real, confiabilidade e disponibilidade absolutas estão muito longe de serem alcançadas. A confiabilidade e a disponibilidade de equipamentos e serviços de computação não são conceitos abstratos e absolutos, mas são atributos de um sistema que podem ser medidos quantitativamente. Várias técnicas de projeto podem ser usadas para aumentar o valor dessas medidas, que podem chegar próximas a cem por cento. Mesmo assim, sistemas totalmente infalíveis são impossíveis, pois falhas são inevitáveis. Mas usuários e desenvolvedores não devem se conformar com equipamentos e serviços de baixa qualidade, desde que estejam dispostos a arcar com o custo do emprego de técnicas de tolerância a falhas.

Esse texto conduz o leitor a um visão geral da área de tolerância a falhas visando motivá-lo para aprofundamentos e pesquisas posteriores. São explorados tanto aspectos teóricos como exemplos práticos. O texto não visa substituir um bom livro texto. Na bibliografia recomendada no final do texto, referências a tais livros podem ser encontradas.

Palavras-chave: tolerância a falhas, alta disponibilidade, confiabilidade, medidas, arquiteturas de alta disponibilidade, clusters, sistemas distribuídos, consenso, comunicação confiável, replicação, recuperação, injeção de falhas.

---

[1] Professora orientadora do PPGC, UFRGS, coordenadora da especialização em Redes e Sistemas Distribuídos, UFRGS, Coordenadora da comissão de Extensão, do Instituto de Informática, Diretora Administrativa e de Finanças da Sociedade Brasileira de Computação

Índice

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, administradores de sistemas e usuários se vêm às voltas com atividades bastante criativas, mas nada gratificantes, de tentar recuperar dados perdidos e de enfrentar equipamento fora do ar devido às múltiplas falhas a que 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 administrador de sistemas a implementar as mais simples, ou exigir dos fornecedores e desenvolvedores de sistemas soluções que as incorporem.

Entretanto, as técnicas que toleram falhas tem um alto custo associado. Pode ser a simples necessidade de *backup* dos dados, que consome espaço de armazenamento e tempo para realizar a cópia, ou a redundância de equipamentos e espelhamento de discos, que consome recursos de hardware sem contribuir para o aumento do desempenho. O domínio da área de tolerância a falhas auxilia administradores e desenvolvedores de sistemas a avaliar a relação custo benefício para o seu caso específico e determinar qual a melhor técnica para seu orçamento.

Sistemas mais robustos em relação a falhas eram, até recentemente, preocupação apenas de projetistas de sistemas críticos, como aviões, sondas espaciais e controles industriais de tempo real, e em certo grau também de projetistas de *mainframes* com exigências de alta disponibilidade. 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 conectado a uma rede deve apresentar um mínimo de confiabilidade.

Conhecer os problemas potencialmente provocados por falhas no sistema, as soluções que existem para evitar falhas ou recuperar o sistema após a sua ocorrência, assim como o custo associado a essas soluções, torna-se imprescindível a todos que pretendem continuar usando computadores, desenvolvendo sistemas ou fornecendo um serviço computacional de qualidade aos seus clientes. Para desenvolvedores de software, projetistas de hardware e administradores de rede, o domínio das técnicas de tolerância a falhas torna-se essencial na seleção de tecnologias, na especificação de sistemas e na incorporação de novas funcionalidades aos seus projetos.

1.1 Mercado para produtos tolerantes a falhas

Existe um mercado mundial para tolerância a falhas que envolve grande soma de recursos financeiros. Esse mercado engloba não apenas operações críticas de tempo real (como transportes, aviônica, controle de processos em tempo real, comunicações), mas

também operações comerciais de missão crítica (como as suportados por sistemas de transações e sistemas distribuídos).

Empresas que dominavam o mercado mundial para aplicações comerciais tolerantes a falhas até a década de 80, Tandem e Stratus, produziam *mainframes* de altíssimo custo para organizações bancárias e financeiras. A partir da década de 90, essas empresas, e também SUN, Digital, IBM, Novell e Compac (que incorporou a Tandem) além de várias outras, começaram a lançar soluções de alta disponibilidade para servidores de rede e clusters, geralmente de alto custo (como por exemplo a série de servidores SUN Enterprise). Uma família de microprocessadores muito popular, Intel 80x86, incorpora desde o i486 uma gama de recursos para tolerância a falhas, que, se bem utilizados, poderiam aumentar consideravelmente a confiabilidade dos sistemas produzidos com esses microprocessadores. Infelizmente, por razões associadas a custos e também principalmente pela carência de uma cultura em confiabilidade, os recursos desses microprocessadores não são plenamente aproveitados.

Com a popularização de aplicações na Internet é prevista uma grande demanda por equipamentos de alta disponibilidade e software e serviços que tolerem em maior ou menor grau a inevitável ocorrência de falhas que assola sistemas computacionais.

1.2 Sobre o texto

O texto visa dar uma visão geral da área, que é bastante ampla. Foi escrito especialmente para cursos de especialização e não visa cobrir em profundidade cada um dos tópicos tratados.

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. Outras fontes de referência importante são os anais de eventos da Sociedade Brasileira de Computação específicos da área, como o SCTF, Simpósio Brasileiro de Tolerância a Falhas, e do WTF, Workshop de Testes e Tolerância a Falhas. Além deles, também nos simpósios SBRC e SBAC podem ser encontrados alguns artigos que tratam de assuntos relacionados a Tolerância a Falhas. Mais informações sobre esses eventos podem ser obtidas no site da SBC (www.sbc.org.br). Na arena internacional, o melhor evento é o DSN, Dependable Systems and Networks, que substituiu o FTCS desde 2000. Os artigos completos desses dois últimos eventos podem ser obtidos em DVD [IEEE01].

1.3 Defeitos em 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. Dos primeiros computadores a válvula, que queimavam e sobreaqueciam rotineiramente, eram extremamente sensíveis à umidade dos nossos trópicos e se soltavam dos soquetes a qualquer trepidação, até a robustez dos notebooks modernos, um acelerado caminho tecnológico foi percorrido.

Entretanto, o software e os procedimentos de 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 foram ainda descobertos.

Alguns defeitos relatados na literatura [Lapr98] valem a pena ser mencionados: na guerra do Golfo em fevereiro de 1991 foram noticiados vários relatos de falhas em mísseis. 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. Várias missões da Nasa a Marte terminaram em fracasso total ou parcial. 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.

1.4 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 no 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 a 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 definitiva.

1.5 Tolerância a falhas ou dependabilidade?

O termo **tolerância a falhas** foi cunhado por Avizienis em 1967. Desde então tem sido amplamente utilizado pela comunidade acadêmica para designar toda a área de pesquisa ocupada com o comportamento de sistemas computacionais sujeitos a ocorrência de falhas, sem ter entretanto logrado sucesso como designação popular. Na indústria o termo nunca teve boa aceitação, sendo que desenvolvedores de sistemas de controle preferem usar o termo **sistemas redundantes** para seus equipamentos. Na comercialização de sistemas computacionais como mainframes e servidores de rede, o termo usual é **alta disponibilidade,** designando a principal qualidade desses sistemas.

Sistemas redundantes e sistemas de alta disponibilidade apresentam técnicas comuns mas alcançam resultados diferentes, uns visam alta confiabilidade e outros continuidade de serviço. Para englobar essas qualidades embaixo de um único chapéu, freqüentemente aparece o termo **segurança de funcionamento**. Com a popularidade do termo segurança computacional, relacionado aos aspectos de segurança contra intrusos e mal-intencionados e que engloba criptografia, autenticação e vários tipos de proteção de sistemas, o termo segurança de funcionamento relacionado a tolerância a falhas caiu em desuso.

O próprio termo tolerância a falhas como designação de área sofre várias críticas, não apenas no Brasil, mas também internacionalmente. A maior crítica é a possibilidade de entender o termo como uma propriedade absoluta. Nessa visão distorcida, um sistema tolerante a falhas toleraria toda e qualquer falha em qualquer situação, o que realmente é uma promessa irrealizável e pode conduzir a falsas expectativas entre usuários.

Aos poucos o termo **dependabilidade** vem substituindo tolerância a falhas no meio acadêmico. Em 2000, o Fault Tolerant Computing Symposium, FTCS, foi rebatizado Dependable Systems and Networks. Em 2003, o SCTF vai passar a se chamar LADC, Latin America Dependable Computing.

Será o fim de tolerância a falhas? Entre nós, por enquanto, ainda não. Dependabilidade é um termo que soa estranho aos nossos ouvidos e não conseguimos encontrar ainda um adjetivo que se ajuste ao termo. Mais detalhes sobre dependabilidade serão apresentados no item 2.2.

2 Conceitos clássicos

Sendo a computação uma tecnologia recente, muitos termos e conceitos não estão ainda consolidados e nem são amplamente aceitos. Vários grupos usam os mesmos termos para conceitos distintos ou então termos diferentes para designar a mesma propriedade ou conceito.

Muitos autores nacionais ou estrangeiros tem se ocupado da nomenclatura e conceitos básicos da área. No SCTF e no WTF, painéis e discussões tem sido conduzidos tentando uma nomenclatura comum no território nacional e até mesmo uma tradução homogênea dos termos usados. Como o grupo de pesquisadores está em expansão, a cada trabalho de um novo autor novos termos conflitantes são introduzidos.

Não tenho a pretensão de estabelecer meus termos e minhas traduções como padrão da área. A minha intenção é que o leitor entenda os conceitos relacionados aos termos e consiga identificar esses conceitos em contextos diferentes, com termos diferentes. Os conceitos e termos 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].

2.1 Falha, erro e defeito

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 um defeito. Finalmente define-se **falha** ou falta (*fault*) como a causa física ou algorítmica do erro.

Falhas são inevitáveis. Componentes físicos envelhecem e sofrem com interferências externas, sejam 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.

Defeitos são evitáveis usando técnicas de tolerância a falhas.

Alguns autores nacionais traduzem as palavras inglesas *failure* como falha e *fault* como falta. Para ser coerente com essa última tradução a área deveria se chamar tolerância a faltas, pois *failures* não podem ser toleradas.

2.1.1 O modelo de 3 universos

Na Figura 1 é mostrada uma simplificação, sugerida por Barry W. Johnson [Prad96], e também 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.

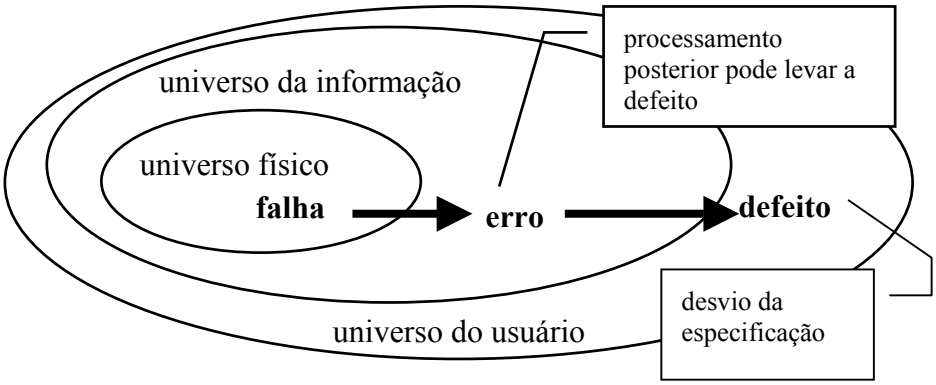

Figura 1: Modelo de 3 universos: falha, erro e defeito

Por exemplo: 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).

2.1.2 Latência

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

Baseando-se no modelo de 3 universos, o tempo total desde a ocorrência da falha até o aparecimento do defeito é a soma da latência de falhas e da latência de erro.

2.1.3 Classificação de falhas

Existem várias classificações para falhas na literatura ([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, para definir uma falha considera-se ainda:

- Natureza: falha de hardware, falha de software, de projeto, de operação, ...
- Duração ou persistência: permanente ou temporária (intermitente ou transitória)

- Extensão: local a um módulo, global
- Valor: determinado ou indeterminado no tempo

Vem crescendo a ocorrência daquelas falhas provocadas por interação humana maliciosa, ou seja, por aquelas ações que visam propositadamente provocar danos aos sistemas. Essas falhas e suas conseqüências são tratados por técnicas de segurança computacional (*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 tolerantes a falhas		**Tolerantes a falhas**			
MTTF: 6 a 12 semanas Indisponibilidade após defeito: 1 a 4 horas		MTTF: 21 anos (Tandem)		Disponibilidade média: 98%	
Defeitos:		**Defeitos:**		**Defeitos:**	
hardware	50%	software	65%	projeto	60%
software	25%	operações	10%	operações	24%
comunicação/ambiente	15%	hardware	8%	físicos	16%
operações	10%	ambiente	7%		

Tabela 1 - Causas usuais de defeitos em sistemas de computação
(MTTF - *Mean time to failure*)

2.2 Dependabilidade

O objetivo de tolerância a falhas é alcançar dependabilidade. 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.

Tolerância a falhas e dependabilidade não são propriedades de um sistema a que se possa atribuir diretamente valores numéricos. Mas os todos atributos da dependabilidade correspondem a medidas numéricas.

Principais atributos de dependabilidade [Prad96] são confiabilidade, disponibilidade, segurança de funcionamento (*safety*), segurança (*security*), mantenabilidade, testabilidade e comprometimento do desempenho (*performability).* Um resumo dos principais atributos é mostrado na Tabela 2.

Atributo	Significado
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– Resumo dos atributos de dependabilidade

2.2.1 Confiabilidade

A confiabilidade R(t) é a capacidade de atender a especificação, dentro de condições definidas, durante certo período de funcionamento e condicionado a estar operacional no início do período.

A definição acima implica algumas condições essenciais, muitas vezes esquecidas:

- especificação: sem uma especificação do sistema, não é possível determinar se o sistema está operando conforme esperado ou não, quando mais formal e completa a especificação, mais fácil estabelecer essa condição. Não é possível estabelecer se um sistema sem especificação é confiável ou não.
- condições definidas: as condições de funcionamento do sistema devem ser bem definidas. Um exemplo simples são as condições ambientais de temperatura e umidade. Outro exemplo são os dados ou estímulos de entrada que o sistema deve processar.
- período de funcionamento: o tempo de missão deve ser conhecido. O tempo de missão de uma viagem espacial é diferente do tempo de missão de um vôo comercial doméstico. Um sistema pode ser altamente confiável para 12 horas de operação e depois necessitar de um longo período de repouso e reparo.
- estado operacional no início do período: não é possível falar em confiabilidade de sistemas que já partem operando com defeitos.

Confiabilidade é a medida mais usada em sistemas críticos, ou seja nos seguintes tipos de sistemas:

- sistemas em que mesmo curtos períodos de operação incorreta são inaceitáveis,
- sistemas em que reparo é impossível.

Exemplos já mencionados de sistemas confiáveis são aviação e exploração espacial.

Confiabilidade é uma medida de probabilidade, pois a ocorrência de falhas é um fenômeno aleatório. Confiabilidade não pode ser confundida com disponibilidade. Um sistema pode ser de alta confiabilidade e de baixa disponibilidade. Um exemplo seria um avião que precisa de reparos e manutenção nos intervalos de vôo.

2.2.2 Disponibilidade

Assim como a confiabilidade, a disponibilidade é uma medida de probabilidade. Disponibilidade é a probabilidade do sistema estar operacional num instante de tempo determinado.

Disponibilidade é o atributo mais usado em sistemas de missão crítica. Sistemas de consulta de base de dados on-line, servidores de rede, servidores de páginas web, são alguns exemplos de sistemas onde alta disponibilidade é requerida.

Disponibilidade não pode ser confundida com confiabilidade. Um sistema pode ser altamente disponível mesmo apresentando períodos de inoperabilidade, quando está sendo reparado, desde que esses períodos sejam curtos e não comprometam a qualidade do serviço (Figura 2). Disponibilidade está muito relacionada com o tempo de reparo do sistema. Diminuir o tempo de reparo resulta em um aumento de disponibilidade.

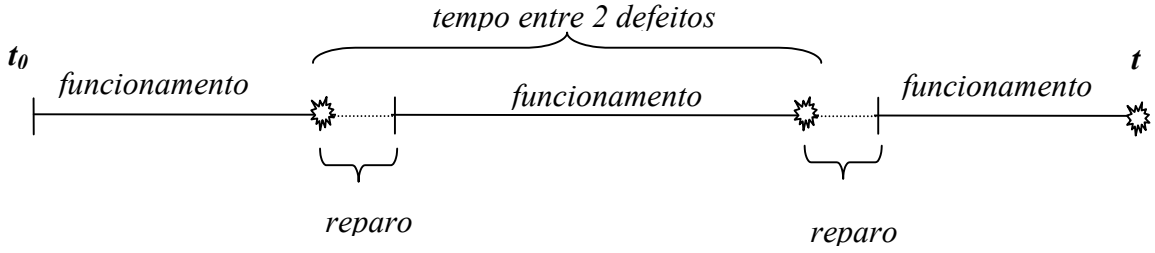

Figura 2: Alternância de períodos de funcionamento e reparo

Apesar de disponibilidade e confiabilidade representarem atributos e corresponderem a medidas diferentes, usuários no geral gostariam de ter sistemas com as duas características. Disponibilidade e confiabilidade não são excludentes, mas as técnicas para implementar uma e outra podem ser bem diferentes.

2.2.3 Segurança de funcionamento

Segurança (*safety*) é a probabilidade do sistema ou estar operacional, e executar sua função corretamente, ou descontinuar suas funções de forma a não provocar dano a outros sistemas ou pessoas que dele dependam.

Segurança é a medida da capacidade do sistema de se comportar de forma livre de falhas (*fail-safe*). Um exemplo seria um sistema de transporte ferroviário onde os controles de um trem providenciam sua desaceleração e parada automática quando não mais conseguirem garantir o seu funcionamento correto. Em um sistema *fail-safe,* ou a saída é correta ou o sistema é levado a um estado seguro.

2.2.4 Outros atributos

Outras atributos importantes de um sistema são: comprometimento do desempenho (*performability),* mantenabilidade e testabilidade. Todas essas medidas são igualmente representadas por uma probabilidade.

Comprometimento do desempenho (*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, e por conseqüência menor o tempo que o sistema não estará disponível devido a reparos.

2.3 Número de noves

Com o interesse crescente do mercado de servidores por sistemas de alta disponibilidade, como servidores de redes e servidores web, uma medida de disponibilidade está se tornando popular. É a medida do número de noves na expressão de percentagem de tempo de disponibilidade. Assim um sistema de cinco noves possui disponibilidade de 99,999%. Um sistema de cinco noves (com 99,999% de disponibilidade) oferece um serviços contínuos, exceto aproximadamente 5 minutos por ano.

Da mesma forma, para representar probabilidades muito próximas de 1 na medida de confiabilidade, com um grande número de 9 após a vírgula, é usual usar a notação $0{,}9i$. Assim por exemplo 0,9999999 é representado por $0{,}9_7$. Esses números, e até maiores, são típicos de sistemas de controle na aviação, representando a probabilidade do sistema operar corretamente durante o tempo de missão, ou seja, durante o tempo de vôo.

2.4 Medidas relacionadas a tempo médio de funcionamento

As medidas para avaliação de dependabilidade mais usadas na prática são: taxa de defeitos, MTTF, MTTR, MTBF. Todas essas medidas estão relacionadas a confiabilidade R(t). A Tabela 3 mostra uma definição informal dessas medidas.

Os fabricantes deveriam fornecer medidas de dependabilidade para os seus produtos, tanto para os componentes eletrônicos, como para os sistemas de computação mais complexos. Tais medidas são determinadas pelo fabricante estatisticamente, observando o comportamento dos componentes e dispositivos fabricados.

Medida	**Significado**
Taxa de defeitos - *failure rate, hazard function, hazard rate*	número esperado de defeitos em um dado período de tempo; é 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 defeitos do sistema

Tabela 3- Medidas de confiabilidade

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

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

- mortalidade infantil: componentes fracos e mal fabricados
- vida útil: taxa de defeitos constante
- envelhecimento: taxa de defeitos 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 defeitos 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.

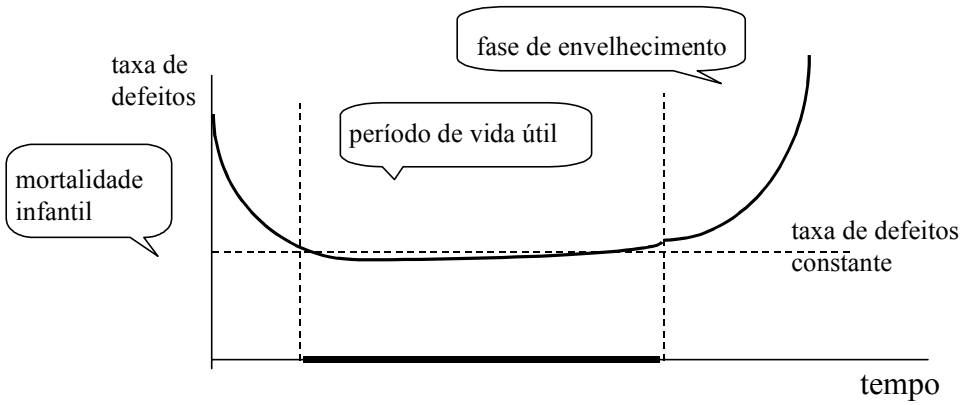

Figura 3: Curva da banheira

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.

3 Técnicas para alcançar dependabilidade

Para desenvolver um sistema com os atributos de dependabilidade desejados, um conjunto de métodos e técnicas deve ser empregado. Esses métodos e técnicas são classificados conforme a Tabela 4.

Técnica	**Função**
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 4 - Técnicas para alcançar dependabilidade

Na Tabela 4 pode ser observado que o chapéu "dependabilidade" abriga várias outras técnicas, além de tolerância a falhas. Todas tem em comum o fato de se preocuparem, de uma forma ou de outra, com a ocorrência inevitável de falhas. Vários autores consideram que a intrusão maliciosa em um sistema computacional seria um tipo de falha e, portanto, as técnicas de segurança computacional (*security*) poderiam ser abrigadas no mesmo chapéu. Essa é a razão de vários eventos acadêmicos na área de dependabilidade aceitarem também trabalhos de segurança.

3.1 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. A 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 termo "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 e mais recentemente a sistemas de missão crítica.

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.

3.2 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, e será discutida em detalhes no item 3.3.

Fases	**Mecanismos**
detecção de erros	duplicação e comparação 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 5 - 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. Eu prefiro 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 nas fases de detecção de erros e de localização de falhas, como uma técnica isolada conduzida periodicamente para diminuir a latência.

3.2.1 A primeira fase: detecção de erro

A primeira fase de Anderson 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 5. 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.

Um exemplo de mecanismo de detecção é o esquema de duplicação e comparação (Figura 4). Duas peças idênticas de hardware realizam a mesma computação sobre os mesmos dados de entrada e comparam os resultados na saída. Havendo desacordo é assinalado erro.

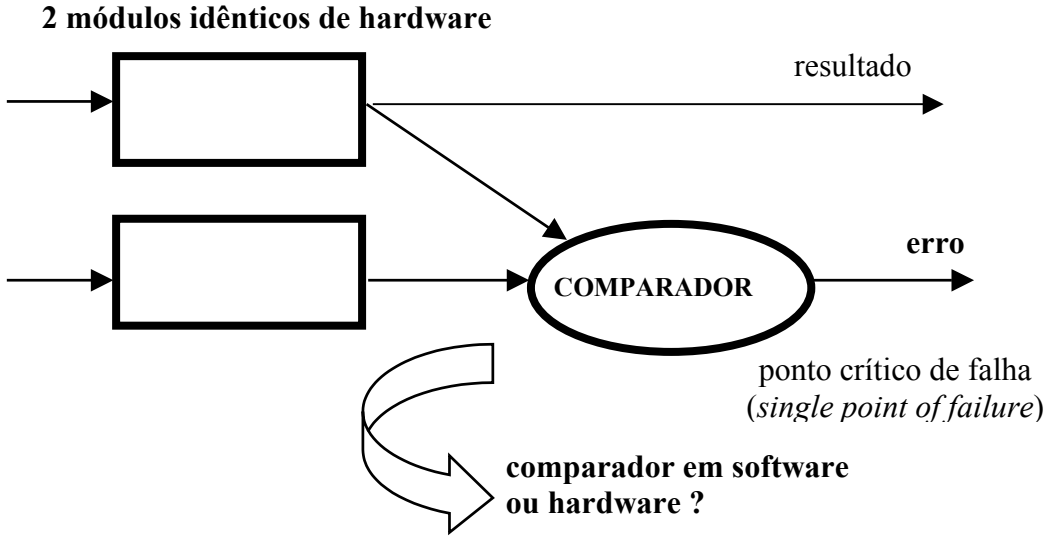

Figura 4: Duplicação e comparação

3.2.2 A segunda fase: confinamento

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.

3.2.3 A terceira fase: recuperação

A recuperação de erros ocorre após a detecção e envolve a troca do estado atual incorreto para um estado livre de falhas.

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 recovery	condução a estado anterior	alto custo, mas de aplicação genérica	pontos de recuperação (*checkpoints*), pistas de auditoria, ...

Tabela 6 - Técnicas de recuperação

A recuperação pode ser de duas formas (Tabela 6): técnicas de recuperação por retorno (*backward error recovery)* e técnicas de recuperação por avanço (*forward error recovery)*. Os dois tipos podem ser visualizados na Figura 5.

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 outros processos, que receberam suas mensagens, 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.

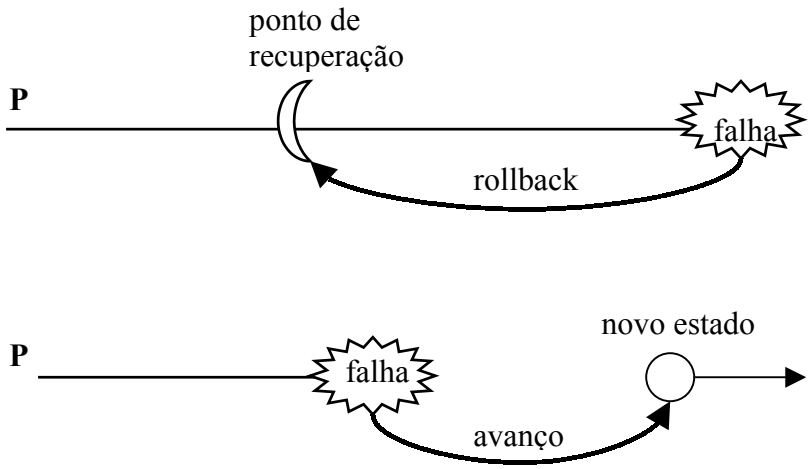

Figura 5: Recuperação por retorno e por avanço

3.2.4 A quarta fase: tratamento

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 e rápida (aplicada sobre um módulo ou subsistema) e localização fina, mais demorada, onde o componente falho é determinado. Para os dois tipos de localização é usado diagnóstico.

O diagnóstico é um teste com comparação dos resultados gerados com os resultados previstos. Pode ser conduzido no sistema de forma manual ou automática:

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

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
- **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.

3.3 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 7 lista mecanismos usuais para implementação de mascaramento de falhas. Alguns desses mecanismos serão vistos em detalhes no item 4, que trata de redundância.

Mecanismo	**Aplicado a:**
replicação de componentes	qualquer componente de hardware
ECC (código de correção de erros)	informação transmitida ou armazenada
diversidade, programação n-versões	especificação, projetos, programas
blocos de recuperação	software

Tabela 7 - Mecanismos para mascarar falhas

4 Redundância

Redundância é a palavra mágica em tolerância a falhas. Redundância para aumento de 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.

A aplicação de redundância para implementar técnicas de tolerância a falhas pode aparecer de várias formas:

- redundância de informação
- redundância temporal
- redundância de hardware
- redundância de software

Essas técnicas de redundância são apresentadas em mais detalhes nas próximas seções.

Todas essas formas de redundância, de hardware, de software, temporal e 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 pode servir 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, muitas vezes é usado outro microprocessador idêntico, sincronizado ao primeiro, além de um comparador de sinais na saída de ambos (duplicação e comparação). Qualquer diferença na comparação indica que o par de microprocessadores está em desacordo, e que portanto um dos dois está danificado (ou sofreu uma falha temporária). Entretanto está falha não pode ser mascarada. O resultado da comparação não indica quais as saídas são as corretas.

4.1 Redundância de informação

Na redundância de informação, bits ou sinais extras são armazenados ou transmitidos junto ao dado, sem que contenham qualquer informação útil. Esses bits ou sinais extras servem apenas para detecção de erros ou mascaramento de falhas.

Exemplos clássicos são códigos de paridade, onde para cada n bits são armazenados $n+1$ bits. O bit extra indica apenas se o número de bits com valor 1 nos restantes n bits é par (ou ímpar, dependendo do tipo de paridade usada). Bits de paridade servem para detecção de falhas simples, ou seja aquelas falhas que afetam apenas um bit de uma palavra. Outros exemplos de códigos usados para detecção de erros são *checksums*, códigos de duplicação, códigos cíclicos.

Mascaramento usando redundância de informação é provida por códigos de correção de erros, como ECC (*error correction code*). Códigos de correção de erros estão sendo exaustivamente usados em memórias e em transferência entre memórias e processadores. Exemplos de ECC são os códigos de Hamming, uma combinação de bits de paridade que permite tanto detecção como correção.

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.

4.2 Redundância temporal

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 o tempo não é crítico, ou onde o processador trabalha com ociosidade. Aplicações usuais da redundância temporal:

- detecção de falhas transitórias: 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 sempre 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, a linha vai transferir sempre zero.

4.3 Redundância de hardware

Redundância de hardware (Tabela 8) está baseada na replicação de componentes físicos.

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 8 - Redundância de hardware

Os 3 tipos básicos de redundância de hardware são apresentados nos próximos itens.

4.3.1 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 é determinado por votação. Exemplos são TMR (*triple modular redundancy)* e NMR (redundância modular com *n* módulos).

TMR, redundância modular tripla, é uma das técnicas mais conhecidas de tolerância a falhas. TMR mascara falhas em um componente de hardware triplicando o componente e votando entre as saídas para determinação do resultado (Figura 6). A votação pode ser por maioria (2 em 1) ou votação por seleção do valor médio [LyVa62].

O votador realiza uma função simples, cuja correção é fácil de verificar. É interessante observar que o votador não tem a função de determinar qual o módulo de hardware discorda da maioria. Se essa 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.

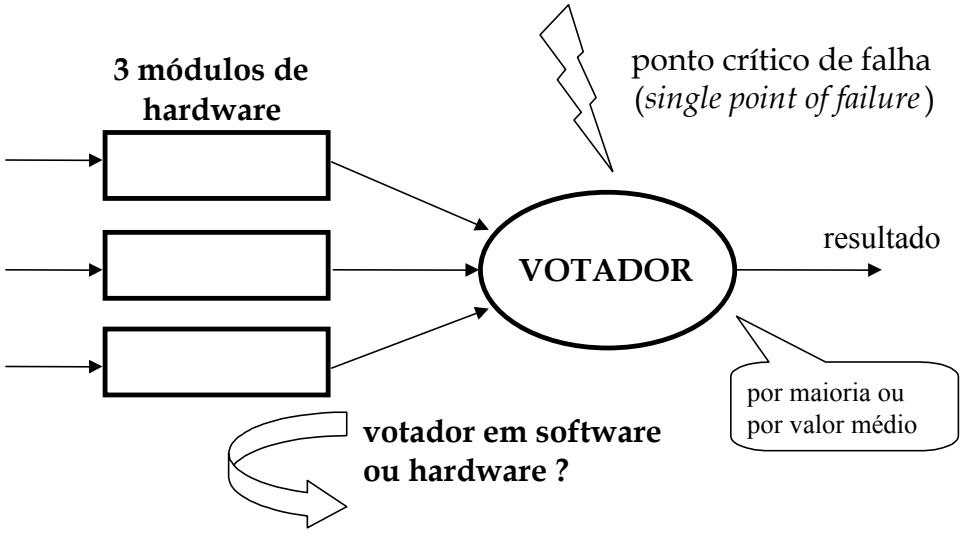

Figura 6: TMR

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 7 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 a possibilidade de falha.

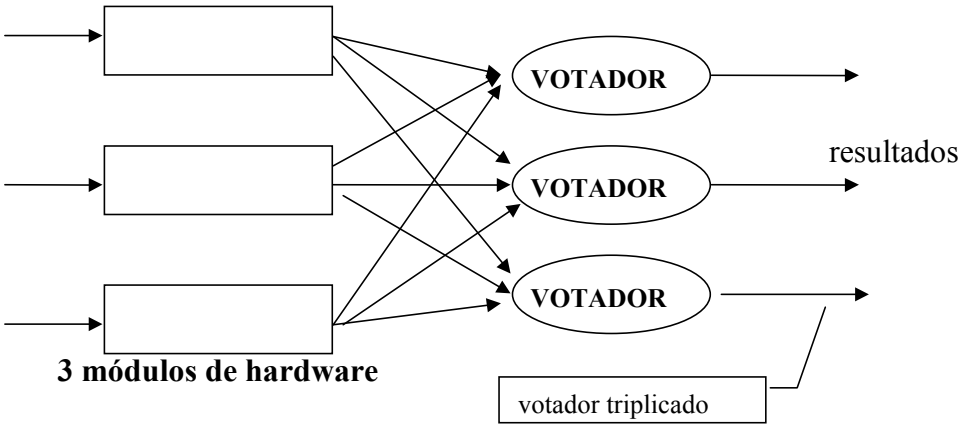

Figura 7: TMR com votador triplicado

TMR apresenta uma confiabilidade maior que um sistema de um único componente até a ocorrência da primeira falha permanente (Figura 8). 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.

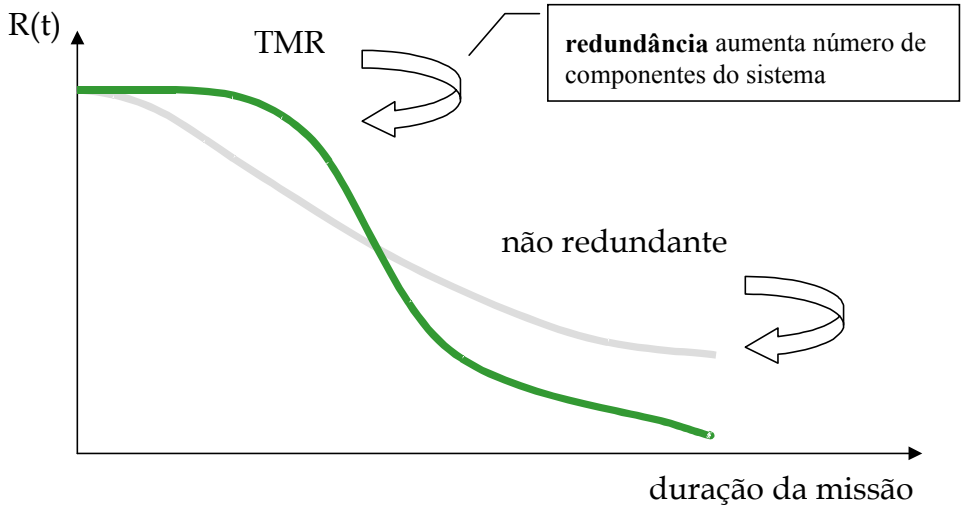

Figura 8: Confiabilidade de TMR

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.

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.

4.3.2 Redundância dinâmica

Na redundância dinâmica ou ativa, tolerância a falhas é provida pelo emprego das técnicas de detecção, localização e recuperação (Figura 9). A redundância empregada neste caso não provê mascaramento.

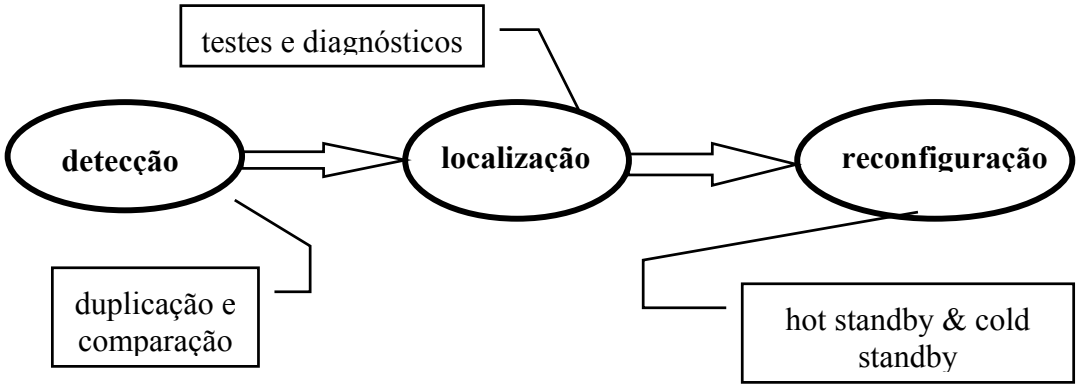

Figura 9: Redundância dinâmica ou ativa

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. Geralmente é preferível defeitos temporários do que suportar o custo de grande quantidade de redundância necessária para o mascaramento de falhas.

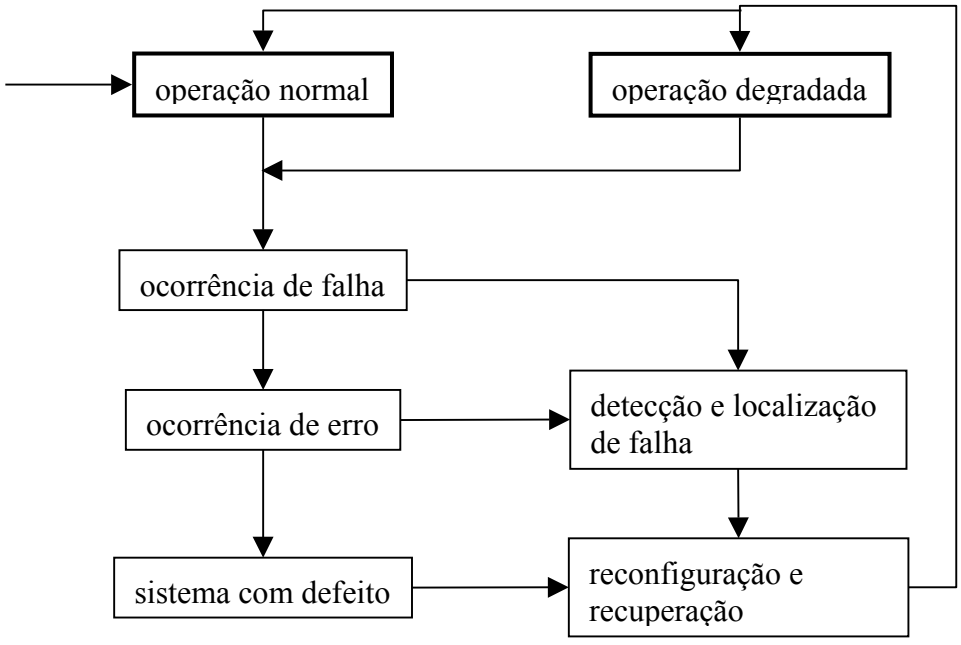

Figura 10: Estados de um sistema com redundância dinâmica

Os estados de um sistema com redundância dinâmica podem ser representado conforme a Figura 10.

Um exemplo de implementação de redundância dinâmica é através de módulos estepe *(standby sparing)*. Módulos estepe podem ser operados de duas formas conforme Tabela 9.

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 9 - Exemplo de redundância dinâmica

4.4 Redundância de software

A simples replicação de componentes idênticos é uma estratégia de detecção e mascaramento de erros 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 em tempos diferentes. Erros de programas idênticos vão se apresentar, com grande probabilidade, de forma idêntica para os mesmos dados de entrada.

Existem outras formas de aplicar redundância em software para detecção e mascaramento, que não envolver cópias idênticas de software. Exemplos dessas outras formas de redundância em software são:

- 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.

4.4.1 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 11).

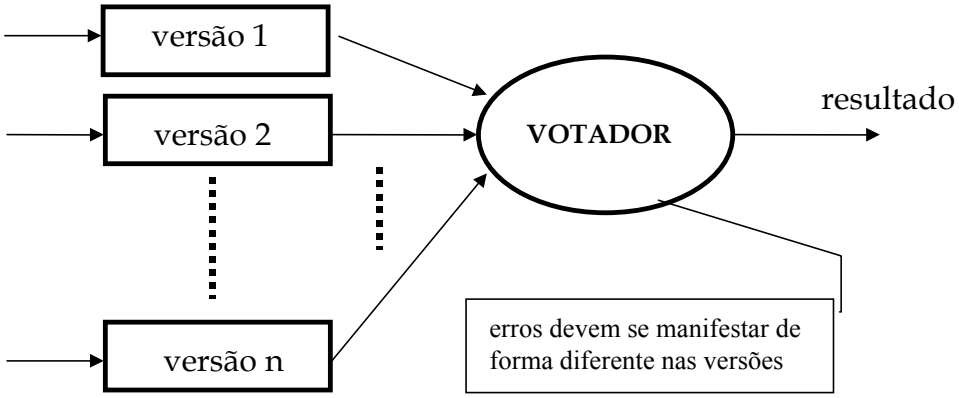

Figura 11: Programação n-versões

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.

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 usando diversidade de metodologias e equipes, e de programação em n-versões, quando se restringe à implementação.

Diversidade 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].

4.4.2 Blocos de recuperação

É 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 12). 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.

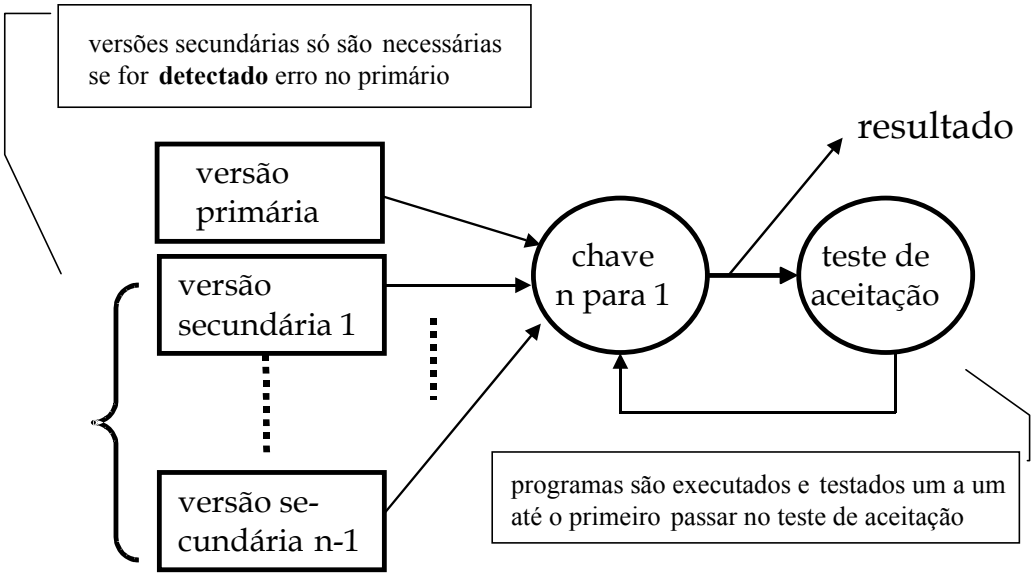

Figura 12: blocos de recuperação

5 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.

5.1 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 e telecomunicações;
- aplicações comerciais de alta disponibilidade como sistemas de transação e servidores de redes.

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.

5.2 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 que não possa ser mascarada, 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. O sistema de computação de bordo do Space Shuttle é um exemplo de sistema tempo real tolerante a falhas inspirado no projeto do SIFT.

5.3 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 (duplicação e comparação), 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 - Electronic Switching System [Toy78], da AT&T 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 e comparação.

Outros exemplos de computadores para telefonia são os sistemas Siemens e, como os demais exemplos citados, empregam exaustivamente duplicação de elementos.

5.4 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 em máquinas clientes. Exemplos são bancos de dados para aplicações financeiras, bancárias, 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 e sinaliza o erro, 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). Tandem implementava tolerância a falhas com processos primários e processos backups, com supervisão de seu sistema operacional proprietário. Stratus, por implementar tolerância a falhas em hardware para detecção (duplicação e comparação) e assim tornar os mecanismos transparentes às aplicações, acabou por suplantar a Tandem como líder no mercado.

Outros exemplos antigos 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).

Tandem, Stratus, Sequoia e outros fornecedores especializados em computadores tolerantes a falhas acabaram saindo do mercado, se associaram a outras empresas de maior porte e partir para desenvolver soluções para nichos específicos de mercado. As

arquiteturas comuns na década de 80 e início de 90 passaram a figurar como exemplos interessantes de soluções de alta disponibilidade.

5.5 Servidores de redes

Redes são formadas por estações de trabalho clientes e por servidores de rede. todas as máquinas que formam uma rede se comunicam por trocas de mensagens. Em uma rede local, o servidor é responsável por serviços de suporte e controle da rede como armazenamento e distribuição de dados e arquivos, gerenciamento e impressão de documentos, gerenciamento de usuários, e também conexão a outras redes, além de várias outras funções.

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 seus inúmeros terminais. É notável sua aceitação também em automação industrial. É fácil perceber que a queda de um servidor pode comprometer a produtividade de toda uma empresa, principalmente se esse servidor for o único disponível ou se estiver executando serviços vitais.

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.

Característica	Descrição
Reconfiguração automática	o sistema reinicia imediatamente após um defeito, desabilitando automaticamente o componente danificado. Envolve testes de componentes a cada vez que o equipamento é ligado ou quando é gerada uma interrupção externa.
Fontes de energia e ventilação redundantes	defeito em um desses componentes não interrompe a operação do sistema.
ECC para garantir integridade dos dados	ECC usado em todos os caminhos de dados. Adicionalmente, paridade é usada nos endereços e linhas de controle do barramento e também nas caches.
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	registradores e contadores são usados para monitorar a atividade do sistema e fornecer cão de guarda (*watchdog*) em hardware. Os registradores são acessados por software para obter dados de desempenho e para manutenção.
Ferramentas de diagnóstico *online*	realizam testes e registram os resultados obtidos. Os testes são realizados em componentes (processadores, memória, I/O,…), e 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 normal.

Tabela 10- Características da série Sun Enterprise

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), HP, Dell 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 10.

5.6 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.

6 Arquiteturas de Sistemas Tolerantes a Falhas

É interessante que um sistema de computação seja suprido das técnicas de tolerância a falhas adequadas para garantir a confiabilidade desejada, sem que as aplicações tomem conhecimento das técnicas empregadas. A tolerância a falhas de um sistema de computação deveria idealmente estar suportada pelos níveis de hardware e software de abstração menor do que o nível ocupado pelas aplicações. Naturalmente essa característica não é suprida por todos sistemas. Geralmente um especialista deve se ocupar do planejamento de certas tarefas complementares, como por exemplo estabelecimento de pontos de recuperação ou elaboração de rotinas diversitárias.

Um nível adequado para suprir tolerância a falhas é a arquitetura do sistema de computação. A arquitetura representa os componentes de hardware de um sistema (como processadores, memórias, controladores, interfaces) e suas interconexões (como barramentos e linhas seriais e paralelas de comunicação).

Recursos de tolerância a falhas implementados na arquitetura para detecção de erros, diagnóstico, recuperação e reconfiguração são mais eficazes do que os implementados exclusivamente nos níveis de aplicação e de sistema operacional, sem o suporte dos níveis inferiores. A seguir são mostrados, como ilustração, exemplos de arquiteturas para sistemas tolerantes a falhas.

6.1 Tolerância a falhas em microprocessadores

Microprocessadores formam a base de computadores pessoais, estações de trabalho e servidores de rede. Como foram inicialmente desenvolvidos aplicações não críticas, os microprocessadores mais populares recentemente começaram a apresentar alguns mecanismos intrínsecos para o suporte de técnicas de tolerância a falhas.

É grande e crescente o número de equipamentos nas áreas de controle de processos industriais, controle de tráfego e instrumentação que usam microprocessadores convencionais devido principalmente ao baixo custo, alto desempenho e capacidade de processamento que esses componentes apresentam. Esses equipamentos e suas aplicações exigem um alto grau de dependabilidade. Naturalmente, desenvolvendo hardware adicional como votadores e comparadores, microprocessadores convencionais, mesmo sem recursos intrínsecos para suporte a tolerância a falhas, podem ser aproveitados para construir sistemas com falhas mascaradas ou com detecção de falhas por duplicação e comparação.

Uma melhor solução em termos de custo pode ser alcançada, entretanto, se suporte para detecção e recuperação for suprida diretamente pelo microprocessador, sem necessidade de hardware adicional. Um exemplo clássico de microprocessador com essa característica é o antigo e já descontinuado iAPX432 da Intel [John84]. O suporte a tolerância a falhas implementado por esse processador não está relacionado às suas características específicas (como conjunto de instruções, modos de endereçamento e registradores internos) e pode ser implementado em qualquer sistema digital integrado com apenas um pequeno acréscimo na área de silício ocupada. Atualmente todos

microprocessadores Pentium apresentam recursos derivados dessa primeira experiência da Intel.

6.1.1 Mestre e verificador

O para mestre-verificador implementa o esquema de detecção de falhas por duplicação e comparação, com o circuito comparador interno ao chip. Basicamente um microprocessador pode ser configurado como mestre ou verificador (Figura 13). Um mestre pode operar sozinho ou ligado a um verificador. Um verificador sempre deve estar ligado a um mestre.

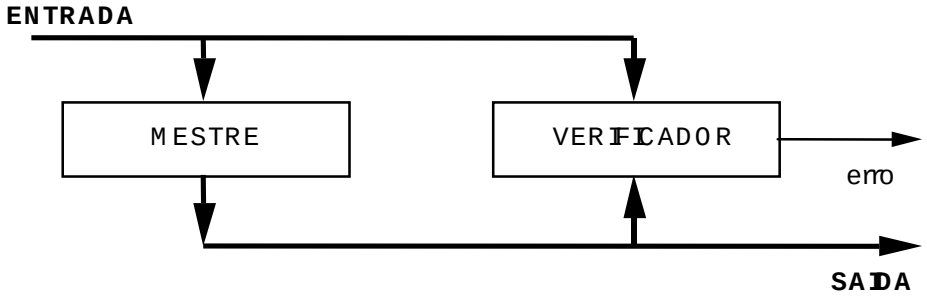

Figura 13 - Configuração mestre-verificador

Um chip configurado como verificador tem seus pinos de saída revertidos para entrada. Os pinos revertidos recebem sinais diretamente das saídas correspondentes do chip mestre. As entradas originais dos dois chips, mestre e verificador, estão ligadas em curto circuito. Internamente ao verificador, os sinais gerados como saída são comparados aos sinais recebidos pelos pinos revertidos. Ocorrendo qualquer discrepância na comparação, o verificador sinaliza erro.

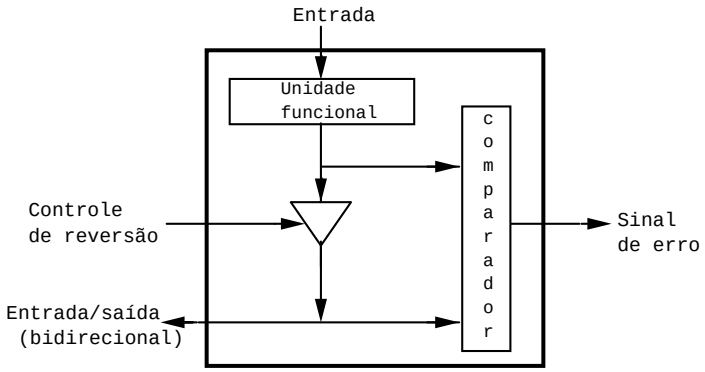

Figura 14: Unidade básica para redundância em microprocessadores

O par mestre-verificador permite facilmente detecção de erro. Com apenas essa redundância simples, o tratamento do erro não é possível por hardware. Usando uma maior redundância, por exemplo quatro chips, ligados dois a dois, tanto detecção quando reconfiguração são possíveis. Essa arquitetura quadruplicada apresenta um par mestre-verificador primário e outro par estepe. Os dois pares são ativos, mas apenas o

par primário fornece resultados ao sistema. Quando o verificador do par ativo detecta um erro, chaveia-se para o par estepe, que passa a partir desse momento a operar sozinho. Redundância quádrupla degrada, nesse caso, para duplex, mas o funcionamento do sistema é garantido sem queda de desempenho.

A redundância usada aqui é independente da função específica realizada pelo chip e pode ser implementada em qualquer circuito digital. O chip resultante tem sua área em silício aumentada em função do comparador, das chaves bidirecionais para todos os pinos de saída, do sinal de controle adicional necessário para configurar o chip como mestre ou verificador e do sinal de erro gerado no comparador (Figura 14). Todos os circuitos construídos usando essa técnica devem ser usados aos pares para detecção de erros.

6.1.2 Tolerância a falhas nos processadores Pentium

No microprocessador Intel 486 foi usada verificação de paridade para as transferências internas de dados entre caches e unidades de execução. Já no Pentium, adicionalmente à paridade nas caches, a TLB e a memória de microcódigo também são verificadas quando à paridade. No Pentium foram introduzidos recursos adicionais para verificação de exceções suportada por hardware (*machine check exception*). Também o esquema de mestre-verificador (usado inicialmente no iapx432) com dois chips voltou a ser adotado pela Intel a partir do Pentium.

O Pentium Pro mantém todas as técnicas do Pentium e adicionalmente:

- paridade nos bytes de dados substituída por 8 bits de ECC
- 2 bits de paridade para barramento de endereço associado a técnicas de *retry*
- bits de paridade para sinais de controle

No Pentium Pro verificação de exceções é conduzida pela MCA (*machine check architecture)* que adiciona 3 registradores de controle e 5 bancos de 4 registradores de erro aos recursos do microprocessador.

6.2 Tolerância a falhas em sistemas de grande porte

Sistemas de grande porte para aplicações universais (*mainframes*) apresentam arquitetura composta de várias unidades processadoras de alto desempenho, uma memória comum de grande capacidade, além de canais que permitem a ligação a uma grande quantidade e variedade de periféricos. Apesar das aplicações de *mainframes* não serem geralmente críticas, os requisitos de disponibilidade impostos a esses sistemas tem crescido fortemente nos últimos anos.

Os vários processadores de um *mainframe*, por serem de alto desempenho e velocidade, são também de alto custo, o que impede a aplicação de redundância pura e simples (ou seja, a mera replicação de componentes) como técnica para aumentar a disponibilidade.

A solução mais comum, usada pelos fabricantes para aumentar a disponibilidade, é incluir um processador auxiliar com funções de console e manutenção. Esse processador

de manutenção [Liu84], é de pequeno porte e autônomo, operando independentemente do *mainframe*, mas ligado diretamente a ele para poder supervisioná-lo.

Um processador de manutenção deve ter capacidade de autoteste e ser construído com componentes confiáveis. Não deve interferir no processamento normal do computador. A existência desse processador não dispensa o uso de outras técnicas de tolerância a falhas, como códigos de correção e detecção de erros. Tais códigos permitem recuperar erros transitórios de alta incidência a baixo custo, sem necessidade de intervenção do processador de manutenção.

Tolerância a falhas é suprida pelas funções de supervisão, diagnóstico e recuperação. Supervisionando continuamente o sistema, situações de erro podem ser imediatamente detectadas. Na detecção de um erro, o processador de manutenção atua da seguinte maneira:

- foi detectado um erro transitório: o processo em erro é interrompido e, tão rápido quanto possível, é recuperado para um estado livre de erros. O sistema operacional recebe então indicação de reiniciar o processo recuperado;
- foi detectado um erro permanente: é localizado o componente danificado e procura-se uma configuração que garanta a operação normal. Pode acontecer:
  - reconfiguração é possível, mesmo com desempenho degradado. Por exemplo, se um processador falhar, o processo é alocado para outro processador;
  - reconfiguração não é possível; o processador de manutenção diagnostica a falha, facilitando o posterior reparo do sistema.

O processador de manutenção é ligado a uma central de manutenção através de rede. Assim ele pode ser suprido de programas de diagnóstico sofisticados e atualizados quando necessário. Pode também avisar imediatamente o pessoal de manutenção da necessidade de trocar placas ou componentes específicos.

Desde que seja garantida alta confiabilidade para o processador de manutenção, ele representa uma solução eficaz e de baixo custo para tornar computadores de grande porte mais disponíveis, diminuindo o tempo de reparo do sistema. Um exemplo de sua aplicação pode ser encontrado nos computadores IBM de grande porte [RSNG82] e em quase todos os computadores de grande porte atuais.

6.3 Computadores de bordo

Na aviação exigências quanto a confiabilidade são extremamente altas, uma vez que vidas humanas estão em jogo em situações onde é impossível interrupção do sistema para reparos. Computadores de bordo tem por função controlar ativamente a aeronave em uma situação cuja normalidade é caracterizada por instabilidade. Correções para manter estabilidade de vôo devem ser feitas continuamente, no instante preciso em que são necessárias (ou seja, em tempo real) e com tempo de atuação curto.

Para computadores de bordo é exigida uma taxa de falhas da ordem de 10^{-9} falhas por hora para um vôo de 10 horas. Além disso deve ser considerado:

- reparo é possível apenas durante os intervalos de vôo, e não mais freqüentemente do que a cada centena de horas de vôo;

- não é admissível qualquer tipo de interrupção no funcionamento do sistema.

Essas exigências extremas quanto à confiabilidade só podem ser alcançadas através da aplicação em larga escala de tolerância a falhas. Como exemplo podem ser citados dois computadores de bordo desenvolvidos na década de 70 sob encomenda da NASA: FTMP (*Fault Tolerant Multi-Processor*) e SIFT (*Software Implemented Fault Tolerance*). Os dois processadores foram projetados tendo por base a mesma especificação.

Tanto FTMP [HSL78] como SIFT [Wens78] são baseados em redundância modular tripla (TMR). Entretanto, nos dois sistemas o votador é implementado de forma diversa. No FTMP o votador é um elemento de hardware, todos os processadores são sincronizados e o relógio central é tolerante a falhas. No SIFT votação é realizada por software, os processadores são assíncronos, sem relógio central, de tal forma que também o sincronismo para o fornecimento de resultados para votação deve ser garantida por software.

Os dois sistemas, FTMP e SIFT, apresentam alta confiabilidade para aplicações em tempo real. FTMP apresenta entretanto um esquema de votação mais eficiente e mais rápido, pois é realizado em hardware. No FTMP tolerância a falhas não é visível a partir da aplicação, ao contrário do que ocorre no sistema SIFT onde a votação é realizada por software.

Baseado no SIFT foi desenvolvido o computador de bordo para o Space Shuttle [Prad96], o que de certa forma atesta que a flexibilidade obtida pela votação em software supera em vantagem a velocidade obtida pela votação em hardware.

6.4 Sistemas Comerciais Tolerantes a Falhas

São muito citados na literatura dois exemplos de computadores de grande porte especialmente desenvolvidos para aplicações comerciais tolerantes a falhas: Tandem [Katz78] e Stratus [Hend83]. Esses dois sistemas foram os mais populares para aplicações em sistemas comerciais de transações [Serl84] durante a década de 80 até meados da década de 90. Tandem e Stratus são também dois bons exemplos de mecanismos de tolerância a falhas implementados em software (Tandem) e em hardware (Stratus).

Atualmente a área de computadores tolerantes a falhas de grande porte é praticamente inexpressiva em termos de novidades e negócios. Todos os grandes fabricantes comercializam soluções ditas de alta disponibilidade ou mesmo tolerantes a falhas. Tandem, incorporada a Compac, e a Stratus continuam aplicando suas técnicas proprietárias na área de servidores de redes.

6.4.1 Tandem

Um sistema é composto vários módulos computacionais interligados por um barramento duplicado. Cada módulo é formado por um processador, uma memória local, um canal de entrada e saída e fonte de alimentação. O sistema dispõe ainda de uma série de

controladores de dispositivos de entrada e saída. Os controladores podem aparecer duplicados. Cada um deles está conectado a dois canais de E/S.

Complementando redundância em hardware, o sistema possui também redundância dinâmica em software. O sistema operacional é formado por um kernel e um grande número de processos, em especial processos de supervisão para cada um dos processadores. Tanto para processos do sistema como para processos do usuário, o sistema operacional permite a criação de pares. Um par é formado por um processo primário ativo e um processo substituto passivo. O processo primário envia pontos de recuperação ao processo substituto.

Diagnóstico de erros se processa da seguinte forma:

- erros em um módulo são detectados em outros módulos processadores;
- a cada segundo, o processo supervisor de um módulo envia sinal de vida a todos os outros módulos no sistema;
- a cada dois segundos, o processo supervisor verifica se recebeu sinal de vida de cada um dos outros módulos. Se faltar um sinal, o processo entende que o módulo correspondente falhou.

Além desse controle mútuo, para cada operação de entrada e saída é realizado controle de time-out. Em caso de falha, o processo de entrada e saída substituto entra em operação.

Um vez diagnosticada a falha em um módulo, todos os processos substitutos relacionados aos processos primários que estavam sendo executados no módulo são rolados para o último ponto de recuperação (recuperação por retorno) e ativados, tornando-se então processos primários. O sistema é reconfigurado em função dos novos processos primários. Tão logo o módulo faltoso seja reparado, os novos processos primários criam seus processos substitutos nesse módulo. Em caso de falha de um canal de entrada e saída, o processo substituto correspondente é rolado e ativado, enquanto o processo primário é desativado passando a ser substituto do primeiro.

6.4.2 Stratus Continuous Processing

Um sistema típico pode ser composto de 1 a 32 módulos, interconectados através de uma rede local. Os elementos em um módulo são interligados por um barramento interno. Os módulos do sistema Stratus não estão disponíveis para redundância dinâmica.

Cada módulo é composto de dois grupos idênticos de componentes de hardware e um circuito lógico para comparação dos resultados de todas as operações que são realizadas em paralelo (duplicação e comparação). Todos os módulos podem aparecer por sua vez duplicados. Essa duplicação entretanto é transparente ao usuário e às aplicações.

Tanto o sistema operacional como os programas de aplicação apresentam apenas alguns recursos óbvios de tolerância a falhas, uma vez que o sistema foi especificado para prover tolerância a falhas por hardware.

Cada módulo do sistema compara os resultados fornecidos pelos elementos duplicados. Quando a comparação indica erro, nenhum resultado é fornecido como saída, o módulo

é desconectado do sistema, sendo então enviado um sinal de erro a um programa de manutenção. Esse programa providencia testes no módulo para determinar se a falha é permanente ou transitória. Em ambos os casos, é registrado o problema e indicado o erro em um terminal de supervisão. Se o módulo faltoso aparecia duplicado no sistema, sua falha permanece invisível à aplicação, pois a unidade redundante garante a continuidade do processamento.

Caso o programa de manutenção tenha detectado uma falha transitória, o módulo que sofreu a falha é ressincronizado com a unidade redundante correspondente e entra imediatamente em operação. Caso seja uma falha permanente, o módulo é substituído manualmente sem interromper o processamento normal.

7 Clusters de alta disponibilidade

O termo cluster é aplicado a uma vasta gama de configurações de computadores com múltiplos processadores. Em comum, as várias configurações de clusters compreendem um número variável de nodos de computação convencionais (de 2 nodos a poucos milhares), alguns dispositivos de armazenamento compartilhados e interconexões de alta velocidade. Clusters lembram os supercomputadores paralelos dos anos 80 e 90 e geralmente são associados a alto desempenho. Entretanto, na prática, cluster são mais similares a sistemas distribuídos [BIR96] e são usados tanto para alcançar alto desempenho quanto alta disponibilidade, ou ambos.

Clusters permitem processamento paralelo: dois ou mais nodos trabalham cooperativamente, compartilhando recursos de processamento e de entrada e saída. Uma aplicação suportada por um cluster executa seus processos concorrentemente em vários nodos, aumentando assim seu desempenho. Nodos em um cluster podem acessar e compartilhar os mesmos dispositivos de armazenamento, substituindo uns aos outros em caso de falha de hardware ou software. A capacidade de nodos operacionais substituírem nodos defeituosos torna a arquitetura cluster interessante para aplicações que exigem alta disponibilidade. Arquiteturas cluster fornecem ainda: (a) um alto grau de escalabilidade, (b) a habilidade de gerenciar múltiplos servidores como se fossem um só e (c) a habilidade de equilibrar a carga de trabalho mais efetivamente fazendo um melhor uso do hardware disponível.

Arquiteturas cluster são ideais para o fornecimento de serviços altamente disponíveis. Em clusters de alta disponibilidade (HA-clusters), alguns nodos permanecem prontos para a qualquer momento assumir a carga de trabalho de nodos que apresentarem defeito, sem contribuir para o aumento de desempenho. Esses clusters também facilitam as tarefas de manutenção. Um nodo pode ser retirado, desligado ou substituído sem interrupção de serviço.

HA-clusters geralmente não compartilham a carga de processamento como os clusters de alto desempenho, nem distribuem tráfego como clusters que balanceiam carga.

Em clusters rodando serviços altamente disponíveis, além da base de dados replicada, o sistema operacional automaticamente migra os serviços de um nodo defeituoso para um nodo operacional e, automaticamente, reinicia as aplicações.

7.1 Compartilhamento de recursos de armazenamento

Clusters são semelhantes a sistemas distribuídos e como tal não possuem memória compartilhada. Compartilhamento de dados pode ser realizado simplesmente através de troca de mensagens ou por armazenamento de arquivos em um disco comum. Em função do compartilhamento de disco, clusters podem ser descritos como:

- sistemas de disco compartilhado: necessitam de um gerenciador bloqueio para organizar as requisições de acesso simultâneo a arquivos. Um arquivo sendo escrito por um nodo do cluster não pode ser aberto para escrita em outro nodo.

- sistemas que não compartilham armazenamento: cada nodo no cluster é efetivamente independente, toda a interação é por troca de mensagens.

Existem sistemas, algumas vezes classificados como cluster, que apresentam memória compartilhada através de hardware especial. Nesse caso, os nodos usam um barramento rápido de memória compartilhada, o que dá a cada nodo acesso ao espaço de memória de todo o sistema. Entretanto, esses nodos precisam estar muito próximos fisicamente uns das outros, enquanto que nodos em sistemas que não compartilham memória podem estar geograficamente distantes uns dos outros.

Sistemas sem armazenamento compartilhado são os mais populares. Nesse caso, os nodos são servidores efetivamente independentes, unidos por uma rede de alta velocidade e coordenados pelo software de cluster.

Uma barreira para a efetiva independência entre os nodos é a necessidade de conectar os servidores a altas velocidades. Uma Fast Ethernet pode ser suficiente para recuperação de falhas. Mas se suporte a alto desempenho e longas distâncias é desejado, então uma largura de banda muito maior e não contenciosa deve ser usada. Entre as soluções, um exemplo é o ServerNet da Tandem, um tipo de backbone de alta velocidade desenvolvida pela Microsoft, pela Intel e pela Compaq (agora proprietária da Tandem)

7.2 Exemplos de cluster de alta disponibilidade

O primeiro cluster de sucesso foi o sistema VAXcluster da Digital. Era formado por nodos VAX fisicamente conectados a um dispositivo central em uma topologia estrela. No sistema operacional, cada nodo tinha sua própria identidade, *drives* e periféricos [SUW99].

A generalização da arquitetura cluster para outros fornecedores de sistemas UNIX foi dificultada porque no UNIX original não existia um maneira automática para criar informação distribuída de usuários, nem para compartilhar o sistema de arquivos através da rede. Hoje em dia, a arquitetura cluster se tornou comum, principalmente porque muitas companhias acharam melhor investir em clusters a sofrer perdas devido a falhas no sistema.

Um bom exemplo de HA-clusters comerciais são os clusters formados por nodos SUN Enterprise (http://www.sun.com), que possuem caminhos redundantes entre todos os nodos, entre todos os subsistemas de disco e para todas as redes externas. Nenhum ponto único de falha – de hardware, software ou rede – afeta a continuidade de serviço no cluster. Um software de gerenciamento de falha detecta falhas e gerencia a recuperação automaticamente. Em clusters SUN que rodam Oracle Parallel Server, uma réplica da base de dados é logicamente presente em cada nodo, e a base de dados permanece disponível enquanto pelo menos um nodo permanecer ativo.

Outro exemplo é o pacote para servidores Linux, denominado Piranha (http://www.sources.redhat.com/piranha), que permite a servidores proverem serviços de alta disponibilidade sem hardware especial. O cluster é totalmente baseado em software, com os servidores interagindo através de uma rede de alta velocidade. Piranha tanto pode ser configurado para alta disponibilidade como para balanceamento de carga.

7.2.1 Cluster SUN Enterprise

O Sun Enterprise Cluster é baseado em um conjunto comum de serviços do sistema e interconexões entre servidores Sun Enterprise. É um sistema de cluster assimétrico, os nodos no cluster não precisam ser idênticos. É mais adequado, entretanto, manter o hardware dos diferentes nodod com poder de processamento equilibrado. Assim, o sistema não sofrerá uma grande queda no desempenho caso o nodo principal falhe.

O número máximo de nodos em um Sun Enterprise Cluster é 256. Entretanto o limite prático realmente depende dos tipos de servidores e suas conexões. Cada cluster é formado de acordo com uma topologia de servidores e conectividade de discos. Dependendo da topologia, do número de conexões entre os servidores, conexões entre os servidores e os discos e conexões públicas, o custo pode variar significativamente. Em geral, quanto mais conexões, maior a confiabilidade, a disponibilidade e também o custo. Entretanto, enquanto a capacidade de processamento do cluster aumenta com cada nodo adicional, eventualmente o desempenho pode saturar ou até diminuir.

Confiabilidade e compartilhamento de carga de trabalho são atingidos através de uma identidade de rede única a todo o cluster. Serviços de rede para cada nodo existem como antes, mas o cluster é visto como uma única entidade, com um único endereço de IP e um único *host name*. Esse endereçamento permite que os usuários se conectem através do nome do cluster e sejam redirecionados para o nodo do cluster com a menor carga de trabalho. Além disso, caso o nodo falhe, um outro membro do cluster toma o seu lugar sem perda de conexão.

Essa visão de sistema único se estende para os dispositivos e processos. Por exemplo, o identificador de processo do UNIX (pid) é único entre todos os membros do cluster, assim como identificadores de dispositivo terminal (o *tty* do UNIX) e nós de estrutura de diretório (o *vnode* do UNIX). Assim, aplicações não necessitam modificações para serem executadas em um cluster. Uma imagem de aplicação de servidor de Web único pode ter processos de sessão HTTP individuais rodando em todos os nodos, distribuindo a carga sempre que possível sem a necessidade de desenvolver uma versão especial do software de servidor de Web compatível com o cluster. O administrador também é capaz de migrar manualmente um processo em execução de um nodo para outro, por qualquer propósito de manutenção, sem qualquer interrupção de serviço.

7.2.2 Microsoft Cluster Server

Clusters da Microsoft foram previstos para ser implementados em duas fases. A primeira delas é o Microsoft Cluster Server (MSCS), que faz parte da Enterprise Edition do Windows NT Server 4.0. O MCS somente proporciona meios básicos de recuperação de falhas no modelo primário-backup para dois nodos. A fase dois está relacionada ao Windows 2000 Advanced Server. Na página http://www.microsoft.com/windows2000/technologies/clustering/default.asp do fabricante podem ser encontrados documentos detalhados sobre cluster com Windows 2000, assim como análises de desempenho, documentação e estudos de casos.

Em sua configuração original, o NT já possui sistemas de arquivo (NTFS e DFS) que podem ser compartilhados através da rede. As maiores limitações da arquitetura cluster NT estão relacionadas ao seu sistema operacional e ao seu modelo de aplicação. Na

maior parte, as aplicações do Windows são projetadas para um único usuário. Além disso, a interface gráfica para essas aplicações as liga fortemente ao sistema operacional. O fato do processamento gráfico sobrecarregar o servidor, escalar o número de usuários exige que se escale proporcionalmente a carga de processamento gráfico no servidor, quando este deveria estar fazendo o processamento de dados e deixando o processamento gráfico para as estações cliente.

O MSCS exige, para armazenamento, SCSI compartilhada que pode ser trocada entre os dois nodos no cluster. Exige também um link dedicado entre os dois nodos. Não existem restrições quanto à tecnologia que pode ser usada: uma conexão Fast Ethernet ou uma tecnologia de interconexão adaptada, como o ServerNet.

Qualquer que seja a tecnologia escolhida, a conexão entre os dois servidores transmite um sinal para que o software de cluster possa monitorar o estado dos sistemas primário e backup. A implementação da Microsoft é um pouco diferente da maioria dos clusters primário/backup pelo fato de que sinais separados podem ser configurados para aplicações particulares. Isso permite que o software recupere falhas de determinados programas, e não de todo o servidor. Ela também usa algo conhecido como *recurso de quorum* (que, na primeira versão do software, é um disco compartilhado) para ajudar a solucionar problemas que podem ocorrer na interconexão. Apenas um sistema de cada vez pode ter a posse desse recurso e assim, em caso de falha na comunicação em uma das conexões, os dois nodos tomam posse do disco de quorum para determinar exatamente qual nodo deve continuar rodando.

O MSCS pode também ser configurado tanto como uma solução ativa/passiva quanto como uma solução ativa/ativa. Assim, o sistema de backup não necessita ser dedicado exclusivamente à tarefa sendo executada no primário, e nem necessita ser idêntico ao primário. O software envolvido é executado como um serviço sobre o sistema operacional Windows NT, permitindo que um servidor virtual e recursos virtuais sejam configurados. Entretanto, existe uma grande diferença na maneira como isso é lidado no MSCS: uma ferramenta gráfica de administração do cluster pode ser executada remotamente de qualquer estação de trabalho NT na rede [PCM99]. O MSCS também permite mover manualmente recursos de um servidor no cluster para outro (para permitir manutenção programada, por exemplo), e configurar o software de recuperação de falhas para atuar automaticamente na reintegração de servidores.

O MSCS suporta apenas TCP/IP, sendo que o software automaticamente move o endereço de IP designado para o servidor de cluster virtual cujo servidor no par de cluster estiver ativo no momento em que um evento de recuperação de falha for detectado [PCM99].

7.2.3 Tandem ServerNet

A Tandem é uma antiga fornecedora no mercado de cluster, com o cluster Himalaya baseado, em RISC, e o software de clusterização NonStop. A Tandem também comercializa o software NonStop na forma de middleware, para uso conjunto com o Unix e o NT sobre plataforma Intel, com particular ênfase no suporte para processamento de transações e aplicações de armazenamento de dados.

A Tandem é responsável por uma das tecnologias mais avançadas em interconectividade, o ServerNet, que se encontra no centro de várias soluções de clusterização (http://www.tandem.com) O ServerNet não apenas permite que os nodos em um cluster se comuniquem entre si, mas também que cada nodo independente interaja com outros sistemas através da rede. O ServerNet pode ser usado para suportar tanto clusters com disco compartilhado (*shared disk*) quanto clusters sem compartilhamento (*shared nothing*), e essa flexibilidade, somada à imensa largura de banda fornecida, permite lidar com alto desempenho, assim como alta disponibilidade convencionais [PCM99].

Comunicação em alta velocidade é alcançada através do uso de uma rede de roteadores interconectados. A interface entre os roteadores e os servidores se dá através de adaptadores PCI plug-in, cada um capaz de suportar até 300Mbps. Isso significa 50Mbps em cada um dos seis links possíveis (efetivamente 100Mbps, pois a transferências são todas bidirecionais). A tecnologia ServerNet também é capaz de acessar informações de nodos distantes, como também "enviar" requisições de informações, e de usar uma técnica chamada *wormhole routing* para facilitar o aumento da taxa de transferência de dados. E, diferente dos sistemas multiprocessadores com memória compartilhada, não existe a necessidade de os nodos no cluster estarem tão próximos uns dos outros, ou unidos usando qualquer tipo de barramento do sistema. Particularmente, os nodos em um sistema ServerNet podem estar a uma distância de até 30m, unidos por cabos de par trançado simples, através dos quais o ServerNet pode proporcionar uma largura de banda completa de 50Gbps ou mais O resultado é uma comunicação entre os nodos muito rápida.

O ServerNet por si próprio apenas proporciona a infra-estrutura de suporte para clustes. São necessários pacotes especiais ou extensões no sistema operacional para permitir tirar o máximo de vantagem das características oferecidas.

Essa tecnologia é suportada por outras companhias, como a Oracle com o Oracle Parallel Server (OPS), o Orion da Novell e o Microsoft Cluster Server. O ServerNet também suporta a especificação Virtual Interface (VI), um padrão para a tecnologia de conexão em clusters desenvolvido pela Tandem, Microsoft e Intel, entre outras,

7.2.4 Clusters Linux

Servidores independentes executando o sistema operacional Linux podem ser agrupados em um cluster sem necessidade de hardware adicional, exceto uma conexão de alta velocidade. Um cluster Linux pode ser baseado puramente em software.

Nem todos os pacotes para cluster suportados por Linux são de software livre e independentes de hardware especial. Por exemplo o Linux Network Evolocity (http://www.linuxnetworx.com) é um produto comercial que compreende hardware e software. O RHHAS (Red Hat High Availability Server) também é um produto comercial cujo núcleo é formado pelo software livre Piranha.

A IBM também vem desenvolvendo produtos para cluster Linux como sistema de arquivos - IBM General Parallel File System (GPFS) for Linux (http://www-1.ibm.com/servers/eserver/clusters/software/gpfs.html), software de gerência de cluster - IBM Cluster System Management (CSM) for Linux (http://www-

1.ibm.com/servers/eserver/clusters/software/) e inclusive hardware especial (http://www-1.ibm.com/servers/eserver/clusters/hardware/1300.html) - o IBM eServer Cluster 1300.

Uma ferramenta desenvolvida no projeto Linux-HA, denominada Heartbeat (http://www.heng.com/alanr/ha/) permite configurar um nodo de backup para qualquer outro nodo em um cluster. O funcionamento é simples e usa uma antiga técnica de tolerância a falhas: o nodo backup monitora continuamente o funcionamento do nodo primário enviando para esse sinais periódicos (*pings* por exemplo); caso o nodo primário falhe, o backup assume o seu lugar. A troca de um nodo por outro se dá pela falsificação de pacotes ARP (Address Resolution Protocol), o nodo backup assume o lugar do principal enganando os demais nodos na rede.

Na Tabela 11 são mostradas algumas outras alternativas de pacotes de software dirigidos a servidores Linux para construção de clusters de alta disponibilidade.

Produto	**Descrição**
SteelEye Lifekepper http://www.steeleue.com	Disponível para servidores Linux, Unix e NT
Piranha http://www.sources.redhat.com/piranha	Fornecido junto a distribuição Red Hat Linux 6.2. Piranha é livre. Provê serviços de "*failover*" (um servidor com defeito é substituído por outro operacional).
TurboCluster EnFusion http://www.turbolinux.com	Para distribuição TurboLinux. Pode integrar Solaris e NT no cluster.
Linux-HA http://www.linux-ha.org	Projeto de desenvolvimento de software de alta disponibilidade. Permite configurar clusters com primários e backups.

Tabela 11 - Exemplos de software para cluster Linux

7.3 Disponibilidade em HA-clusters

A disponibilidade alcançada em clusters é bastante significativa [SUW99]. Por exemplo, 99% de disponibilidade significa que o cluster pode rodar 24 horas por dia, 7 dias por semana, 52 semanas por ano com uma possível *indisponibilidade* de aproximadamente 5 dias, para manutenção planejada ou não. Para muitas empresas, isso é satisfatório. Mas alguns sistemas de missão crítica (como transações bancárias, bolsas de valores, algumas aplicações médicas, guia e monitoramento militar ou controle de tráfego aéreo) exigem um grau mais elevado de disponibilidade. Nesses casos, 99,999% de disponibilidade significa uma *indisponibilidade* de cerca de 5 minutos por ano.

A dependabilidade é mais alta em um cluster do que a alcançada por meio de vários servidores independentes. Uma vez que o cluster é visto como uma única máquina, existe apenas um ambiente para monitorar e gerenciar [SUW99]. Os vários usuários e sistemas de armazenamento estão localizados no mesmo ambiente, tornando mais fácil a localização de problemas e mais rápido o reparo.

Um cluster oferece um potencial para desempenho e disponibilidade aumentados, mas atingir uma mais alta disponibilidade pode ser difícil. Um cluster estará fisicamente localizado em um lugar único, e se ocorrerem acidentes como incêndio ou inundação, um ataque terrorista ou a fonte de energia do prédio for cortada, a disponibilidade do cluster se torna nula. Um sistema distribuído, que replica o estado crítico entre locais distantes, poderia sobreviver a tais acidentes com pouca ou nenhuma interrupção no serviço. Enquanto é seguro afirmar que um cluster oferece opções animadoras para o desenvolvedor de uma aplicação de missão crítica, é também claro que muitos outros aspectos devem ser considerados para se alcançar uma confiabilidade elevada ou uma maior disponibilidade. Nesses caso, outras opções devem ser analisadas, como replicação interna ao nodo para detecção de erros ou mascaramento de falhas.

8 Tolerância a Falhas em Sistemas Distribuídos

Sistemas distribuídos são construídos por vários nodos processadores independentes. Esses sistemas se diferenciam de computadores paralelos pelo acoplamento fraco entre os nodos, ou seja, os elementos de um sistema distribuído não tem acesso a uma memória comum. Toda a interação deve ser feita por troca de mensagens através de canais de comunicação. Nodos de um sistema distribuído também não tem acesso a um relógio global, portanto não possuem uma base de tempo comum para ordenação de eventos. Além disso, sistemas distribuídos são geralmente construídos com elementos não homogêneos e assíncronos.

Sistemas distribuídos não são sinônimo de redes de computadores. Uma rede de computadores pode fornecer a infraestrutura computacional para um sistema distribuído, mas nem toda aplicação de rede é necessariamente distribuída. Por exemplo, uma rede onde cada servidor roda uma aplicação e os demais nodos são clientes dessa aplicação não apresenta os problemas de consistência de dados e sincronização típicos de problemas distribuídos. Por outro lado, um cluster de alto desempenho com centenas e até milhares de nodos pode ser considerado um sistema distribuído, sem necessariamente ser suportado por um rede.

Sistemas distribuídos apresentam um redundância natural, extremamente proveitosa para o emprego de técnicas de tolerância a falhas. Defeito em um nodo processador ou na rede de comunicação não precisa provocar necessariamente a queda de todo o sistema, e o sistema pode ser reconfigurado usando apenas os nodos disponíveis.

A área de tolerância a falhas em sistemas distribuídos é vasta e excitante. Garantir dependabilidade envolve solucionar problemas de consenso, ordenação e atomicidade na troca de mensagens entre grupos de processos, sincronizar relógios quando necessário, implementar réplicas consistentes de objetos, garantir resiliência de dados e processos num ambiente sujeito a quedas de estações tanto clientes como servidoras, particionamento de redes, perda e atrasos de mensagens e, eventualmente, comportamento arbitrário dos componentes do sistema. Leitores interessados no assunto podem encontrar maiores detalhes nos livros do Jalote (Jal94), Birman (Bir96) e Mullende (Mul93).

8.1 Motivação para tolerância a falhas em sistemas distribuídos

Um sistema distribuído deve prover operação continuada, com apenas pequena queda de desempenho, mesmo na presença de qualquer tipo de falha. Apesar de se conhecer um bom conjunto de técnicas de tolerância a falhas, sua aplicação a sistemas distribuídos, principalmente os de tempo-real, ainda é muito recente e seus resultados são muitas vezes insatisfatórios.

Sistemas distribuídos tempo-real diferem dos sistemas convencionais por apresentarem restrições de tempo e tratarem, em muitos casos, com situações críticas. Isto implica que qualquer tipo de falha, inclusive falha temporal, pode causar conseqüências irreparáveis. Tolerância a falhas nesses sistemas é uma exigência óbvia e consolidada. É suportada regularmente pelo hardware do sistema. Com a complexidade aumentada por um

número maior de componentes de software e sofisticadas interações, novas soluções para sistemas distribuídos passíveis de implementação em vários níveis de software e hardware precisam ser desenvolvidas e validadas quanto a correção e temporização.

8.2 Classificação das técnicas de tolerância a falhas em camadas

Jalote [Jal94] apresenta uma interessante classificação em camadas das técnicas e serviços para alcançar dependabilidade em sistemas distribuídos. Essa classificação pode ser representada conforme a Figura 15. Alguns das camadas, blocos básicos, recuperação e resiliência de dados serão mencionadas brevemente nesse texto.

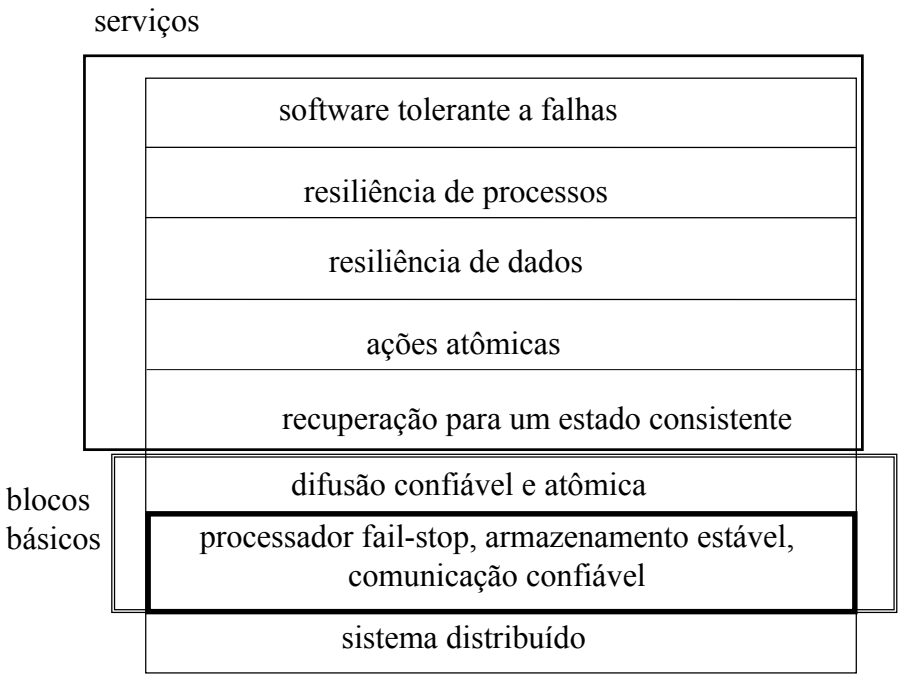

Figura 15: Classificação em camadas segundo Jalote

8.3 Modelos de sistemas distribuídos

Sistemas distribuídos não possuem memória compartilhada nem relógio global. Toda interação entre processos deve ser realizada por troca de mensagens.

Uma melhor análise das características de um sistema distribuído pode ser conduzida considerando um modelo físico e um modelo lógico. No **modelo físico** os componentes são a rede de comunicação e os nodos (processador, relógio local, memória local volátil, armazenamento não volátil, interface de rede, software). No **modelo lógico** a aplicação é vista como distribuída, a rede é considerada completamente conectada e os canais entregam mensagens na ordem que foram enviadas, mas não existe ordenação total de mensagens, apenas ordenação parcial.

Os sistemas distribuídos são classificados como síncronos ou assíncronos, dependendo se existe um limite de tempo finito e conhecido para troca de mensagens. É usual também o conceito de *time-out* associado aos sistemas síncronos.

A ordenação de eventos relaciona a dificuldade de determinar relações temporais devido à inexistência de um relógio global, ou seja, determinar a ordenação temporal de eventos que ocorrem em nodos diferentes, medidos por relógios diferentes. Relógios lógicos (introduzidos por Lamport em 1978) são um meio de assinalar um número a um evento. Não possuem nenhuma relação com o tempo físico. Podem ser implementados através de *timestamps*. O relógio lógico pode ser usado para ordenação total de eventos e é suficiente para a maior parte das aplicações que não exigem respostas críticas quanto ao tempo.

8.4 Falhas em sistemas distribuídos

Algumas características básicas devem ser consideradas para alcançar dependabilidade em sistemas distribuídos. Dependabilidade pode ser necessária, em maior ou menor grau, dependendo da aplicação. Mas todos os sistemas distribuídos devem suprir um mínimo de confiabilidade para permitir operação continuada do sistema mesmo com queda de um ou mais de seus nodos e canais.

Modelos clássicos de falhas para sistemas distribuídos são os de Cristian - falhas de crash, omissão, temporização, resposta e arbitrária - (Figura 16) e Schneider, que estende o modelo de Cristian. - fail-stop, crash, omissão de envio, omissão de recepção, temporização, resposta e arbitrária. Os dois modelos refletem falhas que afetam as trocas de mensagem entre os nodos.

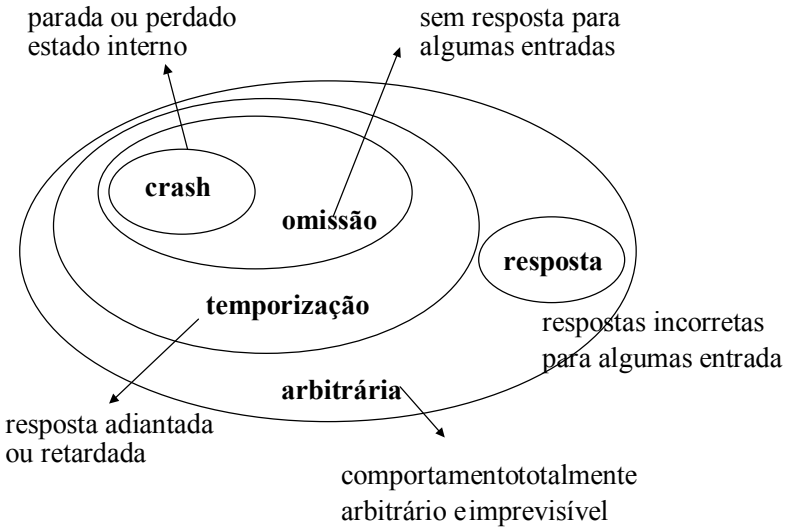

Figura 16: Modelo de falhas em sistemas distribuídos

8.5 Processadores Fail-stop e armazenamento estável

O sistema deve preservar seu estado global mesmo na presença de falhas. Para tanto é necessário que seus nodos possuam armazenamento estável. O armazenamento estável preserva as informações armazenadas mesmo na ocorrência de falhas.

Deve ser lembrado, entretanto, que falhas podem ocorrer inclusive no armazenamento secundário, seja disco magnético ou ótico. Assim o armazenamento secundário não pode ser considerado armazenamento estável. Infelizmente armazenamento estável ideal é apenas uma abstração teórica, mas algumas soluções práticas como RAID ou espelhamento de memória permitem chegar próxima a essa abstração.

O melhor comportamento sob falha para os nodos de um sistema distribuído é simplesmente parar toda e qualquer operação na presença de uma falha interna irrecuperável, assim os demais nodos podem detectar seu estado pela ausência de mensagens. Armazenamento estável e componentes *fail-stop* são dois conceitos importantes aqui.

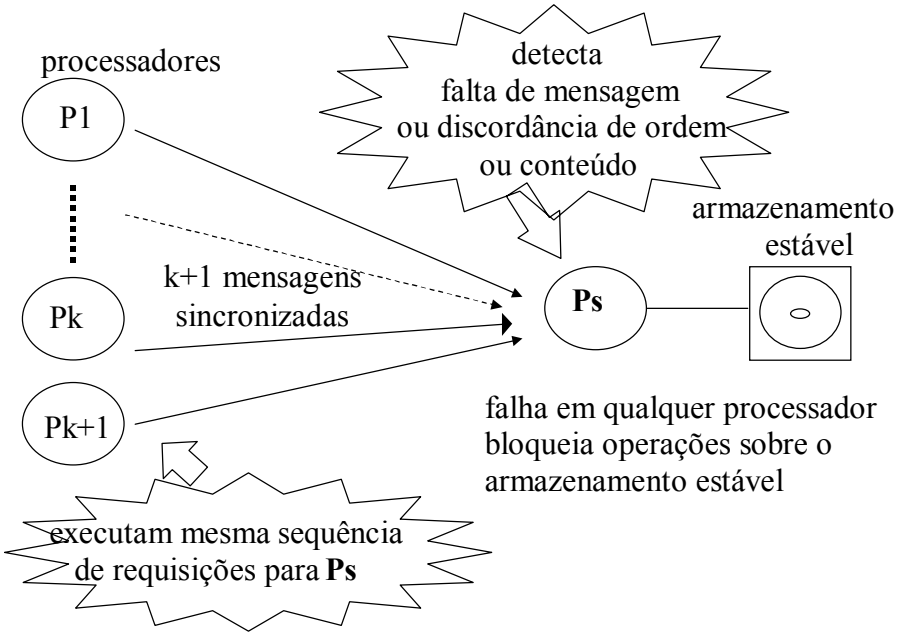

Figura 17: Processador k+1 fail-stop

Assim como armazenamento estável, processadores fail-stop são apenas uma abstração. Processadores fail-stop, apesar de serem assumidos por grande parte das soluções práticas de tolerância a falhas, são existem. Uma aproximação são processadores k+1 fail-stop como mostrado na Figura 17. Esses processadores toleram k falhas.

8.6 Consenso

Em várias situações o sistema distribuído deve alcançar um consenso, ou seja, todos os componentes perfeitos devem contar com os mesmos dados sobre os quais devem aplicar um mesmo algoritmo de decisão. É necessário resolver o problema de consenso

entre todos os nodos mesmo na presença de falhas arbitrárias. A compreensão do problema é essencial para entender os protocolos que envolvem troca de mensagens.

O procedimento clássico para alcançar consenso sob falhas arbitrárias em sistemas síncronos é conhecido como o algoritmo dos generais bizantinos e foi apresentado por Lamport em 1982. O algoritmo garante consenso por troca de mensagens para qualquer tipo de falha quando o número total de nodos for maior ou igual a 3 vezes mais 1 o número de nodos faltosos (chamados de traidores por Lamport).

Lamport mostrou também que consenso é impossível em sistemas assíncronos com falhas arbitrárias. Posteriormente Fisher provou que consenso é impossível em sistemas assíncronos considerando qualquer modelo de falhas.

Infelizmente, mesmo em sistemas síncronos, o número de mensagens trocadas entre os nodos é insuportável em um sistema distribuído real, pois provoca uma exagerada queda de desempenho. Para evitar essa queda, o algoritmo de consenso deve adequar-se a um modelo mais restritivo de falhas e a outras características específicas da rede. Uma série de algoritmos têm sido publicados com esse objetivo desde a apresentação do algoritmo de Lamport. Para resolver o problema da impossibilidade de consenso em sistemas assíncronos foram propostos protocolos probabilísticos, que permitem alcançar consenso com certa probabilidade diferente de 1.

8.7 Protocolos de difusão confiável

Uma técnica de tolerância a falhas necessária em sistemas distribuídos é a difusão (*broadcast*) confiável. Nesta técnica, um processador dissemina um valor para os demais processadores no sistema. Mesmo na presença de falhas, essa difusão deve ser confiável, ou seja a mensagem enviada deve ser recebida por todos os nodos operacionais. Além disso, pode ser necessário preencher algumas outras exigências, como por exemplo atomicidade e ordenação (parcial, total, causal) de mensagens. Como na maior parte dos sistemas a comunicação é ponto a ponto (um para um), a difusão (um para todos) envolve uma série de transmissões de mensagens e torna-se assim extremamente suscetível a falha de nodos ou de *links* de comunicação. Vários protocolos de difusão confiável foram desenvolvidos para suportar determinadas classes de falhas e atendendo uma ou mais exigências específicas (por exemplo os protocolos clássicos de Chang, Cristian e Birman [Jal94], [Bir96]).

Diversos procedimentos de tolerância a falhas baseiam-se em difusão confiável. Sempre que votação e consenso são necessários, por exemplo, difusão confiável pode ser aplicada. Exemplos são sincronização de relógios, replicação de dados, diagnóstico de sistemas.

Em um sistema distribuído são usados tanto protocolos *broadcast*, onde a difusão ocorre para todos os nodos do sistema, como *multicast*, onde a difusão se restringe a um grupo de nodos do sistema. Os protocolos devem tratar os problemas envolvidos na difusão de mensagens em ambientes sujeitos a falhas e a particionamento de redes, inclusive os problemas que afetem o nodo transmissor durante a execução do protocolo. Protocolos para *multicast* devem tratar, além disso, o problema de *membership,* ou seja, a determinação de quais nodos a cada momento pertencem ao grupo de difusão.

8.8 Recuperação em sistemas distribuídos

Para que o sistema não sofra pane devido a falhas em seus componentes, erros devem ser detectados o mais rapidamente possível, o ndo faltoso deve ser identificado através de diagnóstico apropriado e finalmente isolado através de reconfiguração do sistema. Essa reconfiguração se faz realocando processos e escolhendo caminhos alternativos de comunicação entre os processos. Assim, para aumentar a dependabilidade de sistemas distribuídos, detecção de erros, diagnóstico e reconfiguração são técnicas essenciais, que devem ser incorporadas de forma transparente à aplicação e ao seu ambiente de suporte.

Dependendo do tipo de sistema, da especificação de sua resposta no tempo e dos custos de implementação, falhas podem ser mascaradas ou erros podem ser recuperados. Mascaramento é mais efetivo em sistemas de tempo real, podendo também ser usado em sistemas convencionais, mas o custo em hardware é maior. Nesse caso falhas não se propagam e o sistema continua seu processamento sem queda de desempenho. As técnicas de recuperação por retorno não usam tanta redundância como as técnicas de mascaramento, mas pressupõem um estado global anterior seguro armazenado de alguma forma no sistema e a possibilidade de *rollback* para esse estado, o que inviabiliza a sua aplicação a uma vasta gama de sistemas de tempo real usados em controle de processos contínuos. As alternativas à recuperação por retorno são, em geral, de difícil projeto e implementação e extremamente vinculadas a casos bem modelados.

A recuperação por retorno, que é simples em um sistema com um único processo, pode se tornar bastante complexa em um sistema distribuído [Jans97]. A recuperação nesses sistemas envolve o retorno não apenas dos processos no nodo falho, mas de todos os processos envolvidos direta ou indiretamente na comunicação com os primeiros, e pode provocar efeito dominó.

Efeito dominó significa o retorno sucessivo, e em cascata, de todos os processos do sistema ao início da computação, ou próximo ao início, desfazendo grande quantidade de processamento.

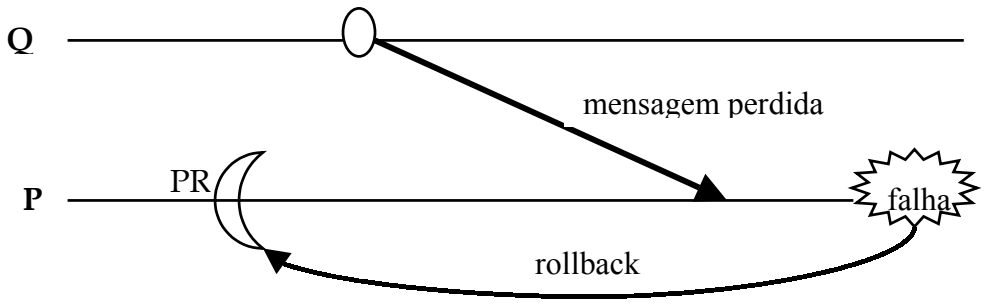

Figura 18: Mensagem perdida

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, eventualmente, que 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.

Vários algoritmos tem sido propostos para estabelecimentos de pontos de recuperação que correspondam a um estado global seguro. Vários algoritmos também sugerem mecanismos para evitar o efeito dominó, para restringir o número de pontos de recuperação que necessitam ser armazenados ou impor restrições a comunicação entre os processos visando evitar o aparecimento de mensagens perdidas (Figura 18) ou órfãs (Figura 19). Jansch-Pôrto e Weber apresentam um resumo dos algoritmos clássicos [Jans97].

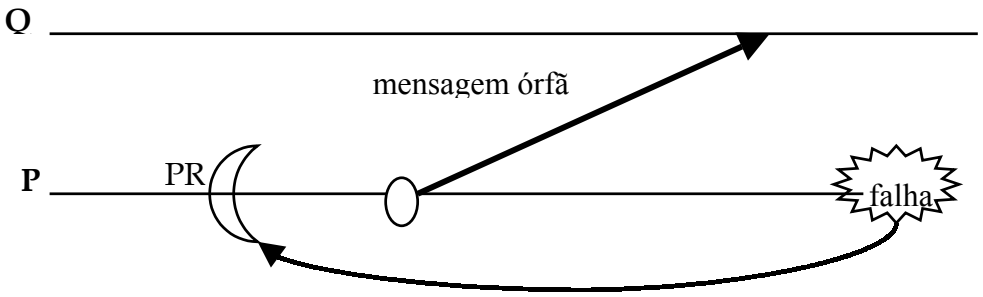

Figura 19: Mensagem orfã

Após a localização da falha por procedimentos de diagnóstico, o sistema distribuído pode ser reconfigurado. A reconfiguração envolve a determinação de uma nova configuração para a rede, o isolamento dos nodos faltosos, redistribuição dos recursos restantes para as aplicações, realocação dos processos aos nodos e reinicialização ou recuperação do sistema. Reconfiguração também é necessária quando um novo nodo é integrado à rede.

Para a determinação de uma nova configuração para a rede é necessário consenso, todos os novos nodos devem reconhecer a mesma configuração (ou seja quais são os nodos perfeitos que restaram e qual a topologia de sua interconexão). Após a migração dos processos e sua recuperação, o processamento pode ser reiniciado. Todas essas operações devem ser realizadas no menor intervalo de tempo possível e considerando, que mesmo nesse pequeno intervalo, novos nodos podem falhar na rede. Deve-se considerar também, no caso de isolamento de nodos, a queda de desempenho gradativa a qual o sistema está sujeito a cada nova reconfiguração. Essa queda de desempenho pode inviabilizar algumas aplicações. A reconfiguração está também relacionada ao gerenciamento e manipulação de grupos de processos no momento da falha.

8.9 Gerenciamento e manipulação de grupos

O tratamento de comunicação de grupo envolve aspectos referentes a forma com que os processos de um grupo são manipulados no sistema distribuído. Isto é conhecido como *group membership*, ou associação de grupos. Este tópico está relacionado ao

gerenciamento e a coordenação dos processos de um determinado grupo do sistema, tanto no momento da falha como no estágio de reconfiguração do grupo.

Gerenciamento e manipulação de grupos relacionam-se aos aspectos de formação e coordenação de grupos em sistemas sujeitos a ocorrência de falhas.

Diversos algoritmos têm sido propostos para resolver o problema de associação de grupos em sistemas distribuídos, sendo uma área promissora para pesquisas, principalmente se associada a aspectos de tempo real. Ao contrário dos tópicos anteriores que são cobertos por Jalote [JAL94], associação de grupos é tratada por Birman [BIR96].

8.10 Replicação de dados

Os recursos físicos usados para armazenar arquivos são, por sua própria natureza, suscetíveis a falhas. Uma ampla gama de falhas pode atingir os meios de armazenamento, quer sejam falhas no próprio meio, nos dispositivos de controle, nos computadores que gerenciam estes dispositivos, e, no caso de redes, nos *links* de comunicação. O uso de grandes redes institucionais e servidores corporativos, com dezenas ou centenas de usuários, tende a agravar este problema. Nas redes, aumenta o número de pessoas ou atividades que dependem dos mesmos dispositivos de armazenamento e servidores de arquivos, criando condições para que aconteçam perdas simultâneas, ou em cascata, relacionadas a falhas em algum dos servidores. As conseqüências de falhas podem ser catastróficas.

Devido a necessidade de confiabilidade no armazenamento de dados, diversas técnicas específicas tem sido desenvolvidas para tornar este recurso mais seguro, sem que uma solução ao mesmo tempo robusta e administrativamente conveniente tenha sido assumida como definitiva. O recurso principal continua a ser o *"backup"* em fita, que possui o grave inconveniente de ser um método *"off-line"*, e de guardar sempre dados algo atrasados, além de implicar em um longo tempo de recuperação.

O aumento de velocidade das redes modernas e sua popularização permitem a adoção de técnicas de replicação de arquivos e a distribuição de réplicas por vários servidores. O algoritmo de replicação é o centro de todo sistema de replicação de dados distribuído, e determina de forma decisiva as decisões a serem tomadas no tratamento de falhas e na recuperação.

Um algoritmo de replicação de arquivos deve resolver principalmente o problema da *manutenção da consistência das cópias sob escritas concorrentes*. Em um sistema multitarefa, diversos processos executam simultaneamente, e podem compartilhar o acesso aos arquivos, inclusive para escrita. Este recurso permite que arquivos sejam usados para troca dinâmica de informações entre programas e usuários. Ainda que leituras concorrentes não gerem nenhum problema, escritas concorrentes podem gerar inconsistências. A questão fundamental está em garantir que todas as alterações geradas sejam aplicadas em todas as cópias na mesma ordem, além da necessidade de que tenham a mesma efetividade. Não importa garantir que esta ordem seja a mesma ordem física em que as requisições foram geradas, nem que as imagens que cada cliente tenha do arquivo sejam as mesmas a qualquer instante, apenas que todas as cópias executem a mesma seqüência de alterações.

Esta ordenação, denominada na literatura como *serialização como cópia única*, não pode ser garantida de forma simples. Os algoritmos desenvolvidos para isto, denominados *algoritmos de controle de réplicas*, devem também ser capazes de operar quando da ocorrência de falhas, quando o número de nodos se reduz, ou sob outras condições, como particionamento da rede, em função do modelo de falhas adotado.

Os algoritmos de controle de réplicas podem ser divididos em três grandes grupos: algoritmos baseados em votação; algoritmos baseados em cópia primária; algoritmos baseados em cópias ativas. Os dois últimos podem ser mais facilmente implementados usando *multicast* confiável e atômico.

9 Validação de técnicas de tolerância a falhas

Um grande problema na área de tolerância a falhas é saber se a técnica implementada resulta realmente em aumento de confiabilidade. Como na maior parte dos sistemas as taxas de falhas são baixas e as falhas acontecem de forma aleatória e incontrolável, o problema se resume em avaliar se a técnica empregada está realmente tolerando as falhas para as quais foi planejada, sem necessidade de esperar meses ou anos para que as falhas realmente aconteçam (por exemplo em aviônica, que é um sistema crítico, a taxa de taxas esperada é uma em cada um pouco mais de milhão de anos). Uma solução para esse problema é a injeção de falhas.

Injeção de falhas é relativamente popular em sistemas isolados. É recente, entretanto, o desenvolvimento de ferramentas de injeção de falhas para sistemas distribuídos, e pouca experiência foi acumulada no tema. Uma breve introdução a essa técnica é particularmente útil ao desenvolvedores de técnicas de tolerância a falhas em ambiente distribuído.

O principal objetivo da injeção de falhas é a validação. A injeção de falhas pode ser vista como um procedimento para o teste da eficácia de técnicas de tolerância a falhas. Através da introdução controlada de falhas, o comportamento da técnica, sob falhas, pode ser avaliado (ou seja, pode ser determinado se a técnica permite ao sistema tolerar ou não as falhas injetadas e qual o custo relacionado a cobertura de falhas alcançada).

A injeção de falhas tem uma importância vital em sistemas críticos de tempo real, sujeitos a graves perdas e danos caso técnicas de tolerância a falhas não consigam garantir a confiabilidade especificada. Também em sistemas de transações, onde falhas podem provocar inconsistências (como em operações bancárias) não é permitido esperar que falhas reais durante a operação normal do sistema venham a determinar que as técnicas empregadas não foram bem concebidas ou implementadas. Outros exemplos são em sistemas distribuídos e sistemas paralelos. Nesses sistemas, o complexidade dos subsistemas de conexão e comunicação facilitam a propagação de falhas e dificultam sua detecção. Sem controle sobre o tipo e origem das falhas torna-se extremamente difícil validar a implementação de tolerância a falhas nesses sistemas.

Injeção de falhas pode ser aplicada ao sistema simulado ou a sistemas físicos operando em seu ambiente natural. Por exemplo: injeção de falhas a nível de pinos e através de distúrbios externos. As técnicas de injeção de falhas a nível de pinos e através de distúrbios externos tem a vantagem de injetar falhas diretamente no hardware, mas podem danificar o componente sob teste e requerem o uso de hardware especial dedicado. Adicionalmente as ferramentas de injeção são específicas a um determinado sistema.

Ferramentas de injeção de falhas implementadas por software não requerem hardware especial para serem aplicadas. Possuem ainda como vantagens baixo custo, baixa complexidade e baixo esforço de desenvolvimento. São facilmente adaptáveis a novas classes de falhas, e não apresentam nenhum problema com interferências físicas. Contudo, a injeção de falhas implementada por software apresenta alguns problemas. A capacidade para modelar certos tipos de falhas, tais como aquelas afetando as unidades de controle de um processador, ainda não foi totalmente desenvolvida. Além disso, a

execução do software responsável pela injeção de falhas afeta as características de temporização do sistema, o que prejudica a execução de funções críticas no tempo.

Mesmo considerando essas desvantagens, injeção de falhas implementada por software tem despertado o maior interesse de pesquisadores e desenvolvedores. A maioria das ferramentas de injeção de falhas mais recentes enfocam sistemas distribuídos, relacionados com falhas de processador, memória e comunicação. Estas ferramentas (como por exemplo FIAT, SFI, DOCTOR, FINE, PFI) injetam falhas por software.

O capítulo 5 do livro do Pradhan [Prad96] escrito por Iyer e Tang traz um ótima introdução a injeção de falhas. Um artigo de Hsue [Hsu97] é um resumo interessante sobre o tema.

No Brasil pesquisa em injeção de falhas tem sido conduzidas principalmente pelos grupos de pesquisa das professoras Eliane Martins, na Unicamp, e Taisy Weber, na UFRGS. Nos anais do SCTF e WTF podem ser encontrados relatos sobre esses trabalhos.

10 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 software cada vez mais complexo, 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.

O texto apresentou ainda uma rápida visão sobre os problemas de falhas em sistemas distribuídos e suas possíveis soluções. Esse texto não visa substituir um livro texto na área. Vários deles são recomendados ao longo da leitura. Um maior aprofundamento visando pesquisas acadêmicas podem ser obtidas nos anais dos eventos da área, tanto internacionais como o DSN, como os nacionais SCTF e WTF.

Os exemplos de sistemas tolerantes a falhas citados no texto não representam uma lista 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 e se mostram um pouco mais dispostos a enfrentar os custos adicionais das técnicas de tolerância a falhas.

Convém ressaltar que mesmo para as áreas onde se dispõe de sistemas tolerantes a falhas, esses nem sempre se apresentam prontos para a imediata utilização. O desenvolvedor de software, ou o usuário especializado desses sistemas, deve muitas vezes prover alguns recursos complementares para garantir a confiabilidade ou a disponibilidade desejada para a 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 são raramente disponíveis devido ao elevado custo associado.

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.

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.

11 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

[BUY99] BUYYA, Rajkumar. *High Performance Cluster Computing.* Volume 1. Upper Saddle River, Prentice-Hall, 1999.

[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.

[Hsu97] HSUEH, M. et. al. Fault Injection Techniques and Tools. IEEE Computer, v. 30, n. 4, Apr. 1997.

[IEEE01] IEEE Computer Society. FTCS/DSN 1971-2001 Compendium.

[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.

[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.

[PCM99] PC Magazine UK Guide to Clustering. Disponível por www em http://www.zdnet.co.uk/pcmag/ (outubro de 1999)

[PRA96] PRADHAN, Dhiraj. *Fault-Tolerant Computer Design*. Englewood Cliffs, Prentice-Hall, 1996.

[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.

[SUW99] Sun World. Disponível por www em http://www.sunworld.com (agosto de 1999)

[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.

[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.