1. TOLERÂNCIA A FALHAS EM SISTEMAS DISTRIBUÍDOS

Na busca de sistemas mais confiáveis, alguns meios foram desenvolvidos para oferecer mais confiança aos sistemas, entre eles está a tolerância a falhas. Tendo em mente que falhas são inevitáveis, procura-se atribuir aos sistemas a capacidade de tolerar a ocorrência de falhas, apresentando funcionamento desejado ou pré-definido, evitando assim danos ao usuário.

1.1 - Conceitos Básicos

Na área de tolerância a falhas, os termos falha, erro e defeito (ou falta) apresentam diferentes significados.

Uma falha é uma condição física anômala, causada por erros de projeto, problemas de fabricação ou distúrbios externos (ANTÔNIO, 199?), como por exemplo: uma flutuação na fonte de alimentação. Erro é a manifestação de uma falha no sistema, causando disparidade nas respostas apresentadas que diferem do valor previsto (ANTÔNIO, 199?) , como por exemplo: devido à falha citada anteriormente, um terminal de um circuito integrado deveria receber o bit 0, mas recebeu o bit 1, e isto causaria um resultado errado. Não necessariamente as falhas presentes no sistema resultarão em erros, pois poderão existir dispositivos que corrijam a falha. Defeito ou falta corresponde a um erro não tratado (ANTÔNIO, 199?) .

Para o melhor entendimento, esses conceitos podem ser representados utilizando-se o Modelo de Três Universos (WEBER, 2000b, Sld 5) (SANTOS & CAVALCANTE, 2000, Cap 2. p.10) (Figura 1). O primeiro é o Universo Físico, que compreende os dispositivos semicondutores, elementos mecânicos, fontes de energia, ou qualquer outra entidade física. Uma falha ocorre nesse universo. O Universo da Informação compreende os dados manipulados pelo sistema, e é onde um erro pode ocorrer, em virtude da existência de alguma falha no Universo Físico. O último universo é o Externo ou do Usuário. É neste onde o usuário final percebe que o sistema apresentou comportamento indesejado e, portanto, possui um defeito.

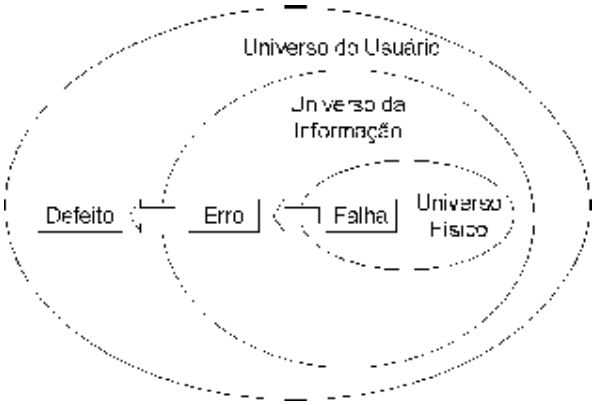

Figura 1 - Modelo dos Três Universos

Como podemos perceber na figura acima, uma falha pode acarretar em um erro e um erro pode acarretar em um defeito (TEIXEIRA & MERCER, 2000, cap. 9). Mas, segundo (WEBER, 2000b, Sld 6), existe entre uma falha e um erro, e entre um erro e um defeito uma fase chamada latência. Esta latência pode ser:

- Latência de falha: É o período de tempo entre a ocorrência da falha até a manifestação do erro devido àquela falha;

- **Latência de erro**: É o período de tempo entre a ocorrência do erro até a manifestação do defeito devido àquele erro.

1.1.1 - Classificação de falhas

Segundo (WEBER, 2000b, Sld 7) (SANTOS & CAVALCANTE, 2000, Cap 2. p.10), as falhas podem ser classificadas quanto à sua origem em:

- **Físicas**: As falhas físicas são causadas por fenômenos naturais como desgaste do material, problemas de interconexão ou quaisquer outros que afetem a estrutura mecânica ou eletrônica do sistema. As falhas físicas podem ainda ser subclassificadas quanto à duração em:

Permanentes – uma vez que se manifestam sempre o farão.

Temporárias – não-permanentes, podendo ser intermitentes, normalmente causadas pelo processo de degradação do componente e que fatalmente se tornarão permanentes com o tempo; ou transitórias, geralmente relacionadas à interferência de sistemas externos.

- **Humanas**: As falhas humanas são aquelas introduzidas no sistema pela ação do homem. Podem ser subdivididas em:

Falhas de Projeto – são introduzidas na fase de projeto do sistema.

Falhas de Interação – ocorrem durante a interação dos usuários com o sistema. Considera-se que as falhas nunca são introduzidas pelo usuário. Este apenas causaria a manifestação de uma falha já existente no sistema, originadas por erros de projeto.

1.1.2 - Aplicações que demandam tolerância a falhas

Segundo (WEBER, 2000b, Sld 28) (SANTOS & CAVALCANTE. Cap 2. p.11), os sistemas que devem apresentar características de tolerância a falhas podem ser categorizados em quatro áreas de aplicações:

- **Longa Vida**: São aplicações que foram projetadas para estar em operação por um grande período. É considerado grande, um período que ultrapasse dez anos. Exemplos de aplicações de Longa Vida são os satélites e sondas espaciais.

- **Computação Crítica**: É talvez a categoria de aplicação onde a tolerância a falhas é mais aplicada. Compreendem sistemas que, se apresentarem funcionamento inadequado, podem levar a conseqüências catastróficas, seja pondo em risco vidas humanas, seja causando altos danos materiais. Exemplos clássicos de aplicações de Computação Crítica são os sistemas de controle de tráfego aéreo, sistemas de mísseis teleguiados e de controle de indústrias químicas.

- **Adiamento de Manutenção**: São aplicações cuja manutenção é extremamente cara, inconveniente ou difícil de executar. Para sistemas como esses se deseja que a manutenção seja feita periodicamente e enquanto isso, o sistema por si só consiga evitar e tratar as falhas que ocorram durante sua execução. Um exemplo de aplicação desse tipo é o sistema de estações de comutação telefônicas.

- **Alta Disponibilidade**: São aplicações cuja disponibilidade é um fator crítico. Exemplos clássicos são os terminais de caixa eletrônicos dos bancos.

1.1.3 - Fases do processo de tolerância a falhas

Segundo (WEBERSLD, 2000b, Sld 26) (SANTOS & CAVALCANTE, 2000, Cap 2. p.11), há várias fases para o processo de tolerância a falhas nos sistemas. As fases serão mais bem detalhadas a seguir:

- **Detecção de Erros**: A primeira fase do processo de tolerância a falhas é claramente a detecção de erros no sistema. Todo o mecanismo de tolerância a falhas empregado no sistema dependerá da eficiência do seu módulo de detecção de erros.

O módulo de detecção de erros deve observar o funcionamento do sistema sendo capaz de perceber desvios de comportamento a partir da especificação inicial.

Existem algumas propriedades que um mecanismo ideal para detecção de erros deve apresentar:

Independência – o módulo de detecção de erros não deve ser influenciado pela estrutura interna do sistema, pois os erros existentes no sistema poderiam afetar também este módulo.

Completo – deve detectar todos os erros possíveis.

Correto – não pode considerar um comportamento esperado como um comportamento anômalo, ou seja, sempre que um erro for detectado, pode-se ter certeza da existência de falhas no sistema.

- **Confinamento de Erros**: Após a detecção do erro, deve-se identificar o módulo ou componente falho do sistema. Com as passagens de informações entre os módulos e componentes, os erros podem propagar-se pelo sistema. Assim, todo o fluxo de informação originado do módulo ou componente falho deve ser observado e as conseqüências das ações devem ser delimitadas, determinando as partes do sistema que foram corrompidos pela manifestação da falha. Este é o objetivo desta fase.

- **Recuperação de Erros**: Detectado o erro e identificada sua extensão pelo sistema, as alterações de estado devem ser removidas para levar o sistema a um estado aceitável evitando o mau funcionamento do sistema futuramente.

Nesta fase, o sistema deve restabelecer um estado livre de erros após uma falha. Existem duas abordagens para a recuperação de erros:

Por avanço – se a natureza dos erros pode ser completamente avaliada, então se pode remover estes erros do estado do processo e habilitá-lo a prosseguir.

Por retorno – se não for possível remover todos os erros do estado do processo, então o processo deve ser restaurado para um estado prévio livre de erros. Normalmente, são utilizados *checkpoints* nessa fase.

- **Tratamento de Falhas**: Nas três primeiras fases, o erro é detectado, sua extensão avaliada e removido deixando o sistema livre de erros. Isso pode ser suficiente se a causa do

erro foi uma falha transitória. Se as falhas forem permanentes, então o mesmo erro poderá ocorrer novamente em processamento futuro. O objetivo desta fase, também conhecida como reconfiguração, é identificar o componente falho e removê-lo do sistema para não mais ser utilizado. O componente causador da falha pode ter a granularidade variada, podendo corresponder a um *chip* ou a uma placa inteira, por exemplo.

1.2 – Sistemas Distribuídos

Um Sistema Distribuído é aquele que roda em um conjunto de máquinas sem memória compartilhada, criando uma ilusão para o usuário de que somente existe uma máquina executando o processo por ele requisitado (ANTÔNIO, 199?). Os Sistemas Distribuídos permitem que uma aplicação seja dividida em diferentes partes, que se comunicam através de linhas de comunicação, e cada parte podendo ser processada em um sistema (*hardware* e *software*) independente. Mas, a partir do momento em que estas partes estão distribuídas em diferentes máquinas, que por sua vez possuem dispositivos de armazenamento susceptíveis a falhas, há uma necessidade de utilizar técnicas específicas de tolerância a falhas. Na seção 1.3 veremos uma abordagem sobre as principais técnicas utilizadas para tolerância a falhas.

Como percebemos, os sistemas distribuídos possuem características naturais que induzem ao uso de tolerância a falhas, pois estando várias partes de um sistema distribuído em diferentes computadores de uma rede mantêm uma maior confiabilidade no que diz respeito ao funcionamento contínuo de um processamento, pois pelo menos uma das partes estará processando a requisição do usuário caso haja ocorrência de falhas nos *hosts* (servidores).

1.3 - Técnicas de Tolerância a Falhas em Sistemas Distribuídos

A distribuição de partes de um sistema por diferentes meios de armazenamento leva a um problema intrínseco de sistemas distribuídos que é a inconsistência e a falta de integridade dos dados (WEBER, 2000a, Item 7). Para resolver este problema são utilizadas várias técnicas que garantem que todas as partes sejam consistentes, ou seja, que eles possuam o mesmo estado no momento em que houver uma falha e uma outra parte possa assumir o processamento a partir do ponto em que ocorreu a falha.

1.3.1 - Técnica de Recuperação

Para que um sistema não sofra pane total, ao ser detectada uma falha o sistema deve ser automaticamente reconfigurado, ou seja, um processo que estava sendo executado deve ser realocado para outros caminhos alternativos que mantenham a comunicação contínua. Mas, para que o processo de recuperação não reduza a performance do sistema e seja o mais transparente possível para o usuário, a técnica de recuperação por avanço é bastante utilizada e requer permanentemente um estado consistente entre as partes de um sistema. Os mecanismos utilizados na técnica de recuperação são (LUNG, 2000, Sld 59):

- ***Logging***: O *logging* é o mecanismo pelo qual as requisições feitas pelos usuários a uma parte do sistema que está distribuído em algum meio físico são gravadas em um

arquivo *log*, para que no caso de uma falha, uma outra parte do sistema que assumir o lugar do anterior possa obter o estado corrente e daí continuar o processo normalmente.

- ***Checkpoint***: O *checkpoint* é um determinado ponto no qual o estado atual de uma parte do sistema que está no arquivo *log* é verificado e copiado no arquivo de *log* das demais partes. Isto faz com que todas as partes distribuídas do sistema estejam sempre atualizadas, mantendo-se assim consistentes. Para uma melhor visualização do funcionamento destes dois mecanismos, vejamos a figura 2.

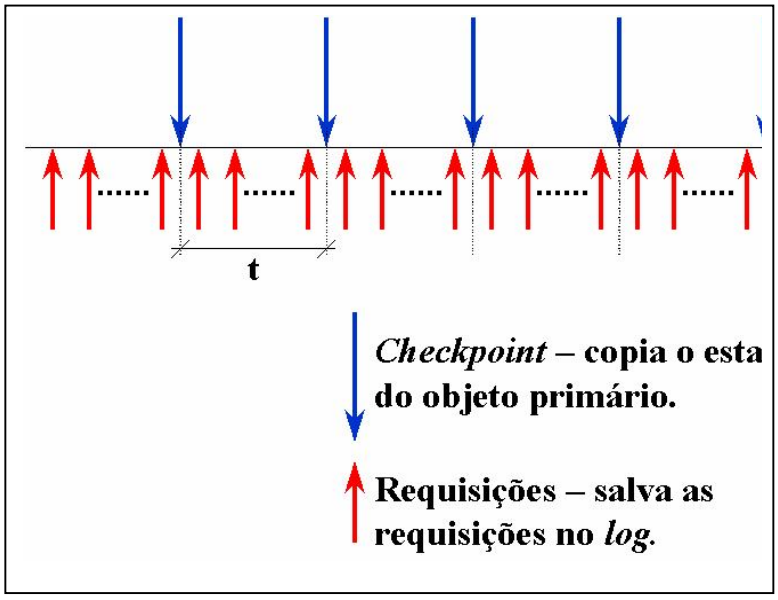

Figura 2 - *Checkpoint* e *Logging*

Como vemos na figura 2, o *checkpoint* é realizado a cada tempo **t,** que pode ser definido pelo usuário ou fixado na implementação do sistema. Durante este intervalo de tempo **t**, várias requisições de usuários são realizadas e gravadas no arquivo de *log*. Um exemplo da utilização da técnica de recuperação é uma transação bancária onde um cliente requisita a

transferência de um determinado valor de uma conta X para uma conta Y. Nesta transação ocorrem duas operações: uma de débito da conta X, e uma de crédito na conta Y. Suponhamos que no intervalo de tempo entre o débito e o crédito ocorra uma falha e só tenha sido realizado o débito da conta X, então haveria aí uma inconsistência de dados, pois o valor debitado da conta X estaria perdido. Mas, com o uso do *checkpoint* e *logging* isto é resolvido da seguinte maneira: suponhamos que temos partes do sistema que realiza esta operação de transferência em vários meios físicos confiáveis, como por exemplo: em dois servidores de aplicação (um ativo e um outro como backup). Quando o cliente requisita a operação de transferência, o servidor ativo assume a operação e o mecanismo de *logging* grava esta requisição em um servidor de *log*. Se o intervalo de *checkpoint* for suficientemente curto, esta requisição será repassada para um arquivo de *log* no servidor backup. No momento em que ocorre uma falha como a descrita anteriormente, o cliente é logo redirecionado para o servidor backup e o mecanismo de recuperação (*recovery*) consulta o *log* para verificar qual foi a última operação realizada, e a partir deste ponto continuar a operação, sem causar transtornos para o cliente.

1.3.2 - Técnicas de Replicação

A técnica de replicação consiste em facilitar a criação de réplicas ou cópias de um mesmo objeto em meios físicos diferentes, garantindo assim que, no caso de uma falha em um dos dispositivos, o cliente seja redirecionado de forma transparente para outro que tenha uma réplica do objeto ativa (JATENE, 1999).

As técnicas de replicação podem ser resumidas em dois tipos básicos (JATENE, 1999):

- **Replicação passiva**: Neste tipo de replicação, há apenas um objeto que recebe as requisições dos clientes, este objeto é chamado primário. Os outros objetos são chamados secundários ou *backups*, e constantemente são informados pelo primário sobre o estado atual do mesmo. Para isto, esta replicação utiliza os mecanismos de *checkpoint* e *logging*. Na figura 3 podemos visualizar melhor o funcionamento desta replicação.

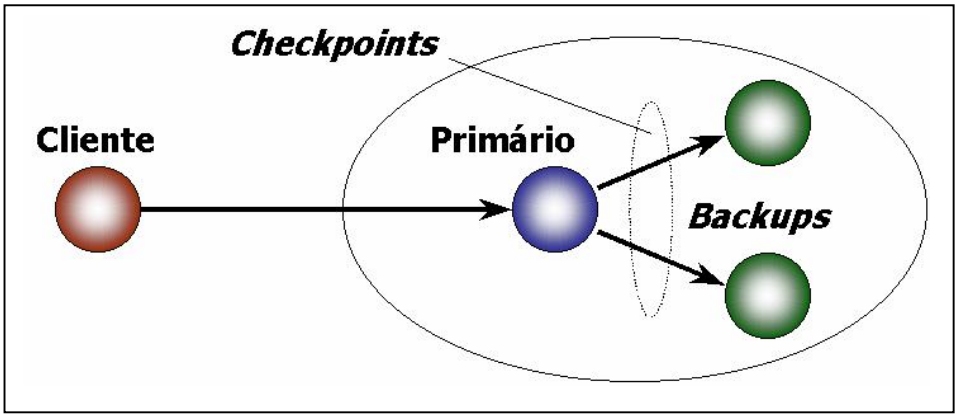

Figura 3 - Replicação Passiva

Como podemos perceber na figura 3, somente o objeto primário recebe as requisições, e através do mecanismo de *checkpoint* estas requisições são salvas nos arquivos de *log* dos objetos *backups* ou secundários. No caso de uma falha do objeto primário, um dos dois objetos secundários assume o lugar do primário, continuando assim o processamento.

- **Replicacão ativa**: Neste tipo de replicação não são utilizadas as técnicas de recuperação, pois todas as réplicas são primárias, ou seja, todas elas recebem a requisição do

cliente , processam e devolvem os resultados de forma simultânea. Neste caso, a replicação deve obedecer às propriedades de atomicidade (todas as réplicas recebem as requisições num mesmo instante) e de ordenação (todas as réplicas recebem as requisições na mesma ordem). Na figura 4, podemos perceber melhor o funcionamento da replicação ativa.

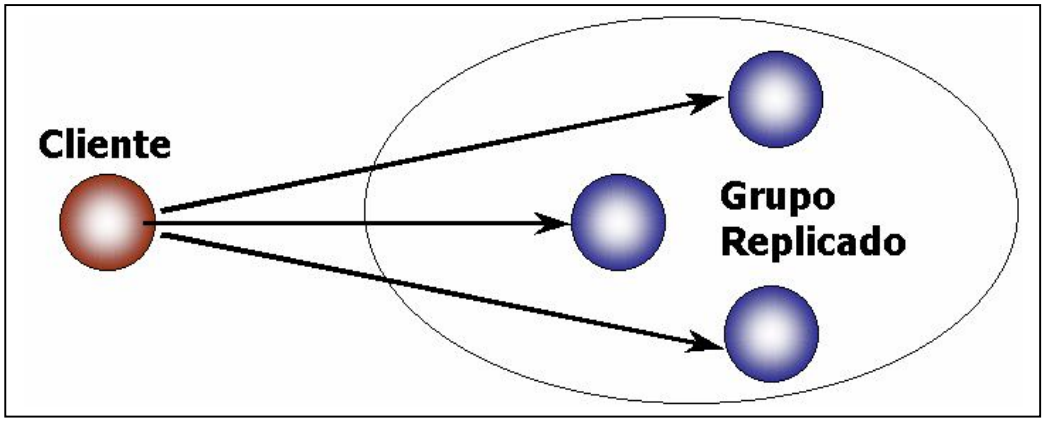

Figura 4 - Replicação Ativa

Podemos perceber que este tipo de replicação pode gerar inconsistência dos dados no momento em que todas as réplicas enviam os resultados dos seus processamentos para o cliente, pois pode ocorrer de uma resposta chegar antes da outra, causando assim múltiplos resultados para o cliente. Para resolver este problema, é comum utilizar uma variação deste método chamada replicação ativa com votação, no qual ocorre uma votação e somente uma das respostas retorna para o cliente. Esta técnica será detalhada no capítulo 3.

Podemos perceber também que neste tipo de replicação há uma melhora de performance no funcionamento dos sistema, pois não teríamos mecanismos de recuperação que causaria um certo atraso no processamento, principalmente quando se trata de sistemas de tempo real como sistema clínicos, onde uma demora de milissegundos pode ser fatal. Mas,

como podemos perceber, este tipo de replicação apesar de ser mais confiável, causa um tráfego de informações razoável na rede, pois haveria uma difusão da requisições para todas as máquinas da rede (broadcast) e as mesmas responderiam ao mesmo tempo. Para resolver este problema, foi criada a técnica de gerenciamento de grupos.

1.3.3 - Técnica de gerenciamento de grupos

Uma das dificuldades encontradas nas técnicas de replicação que vimos anteriormente é o tráfego na rede. Mas, outro problema surge quando precisamos tornar transparente para o cliente a localização das réplicas no caso de uma falha, pois teríamos que criar uma referência para cada réplica criada. Para resolver estes problemas, foi criada uma definição de grupos de objetos, onde os objetos são integrados em sub-grupos diferentes, contendo cada grupo uma referência única. Com isto resolveríamos tanto o problema da localização das réplicas pelos clientes, como também o tráfego excessivo na rede, pois nesta técnica já estaria implementado a primitiva de comunicação multicast, onde apenas um determinado grupo de objetos receberiam as requisições, ao contrário da primitiva de comunicação *broadcast*, onde todas os objetos replicados recebem requisições (JATENE, 1999).

Também chamada de *Group Membership* (WEBER, 2000a) (JATENE, 1999) ou associação de grupos, esta técnica visa também facilitar a manipulação de réplicas, onde podemos cadastrar e excluir réplicas de um grupo, além de aumentar a confiabilidade no tratamento de tolerância a falhas, principalmente quando temos redes particionadas, ou seja, sub-redes interconectadas por um roteador, *gateway*, etc. Vejamos na figura 5 como é definida a técnica de gerenciamento de grupo.

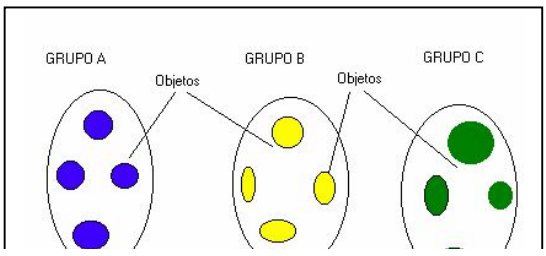

Figura 5 - Grupo de Réplicas de Objetos

Como percebemos na figura 5, cada conjunto de réplicas associadas formam um grupo. Estes grupos podem ser comparados a domínios de tolerância a falhas, ou seja, uma grande rede particionada por sub-redes e interconectadas por hubs, switches ou roteadores. Cada grupo possui uma referência única (JATENE, 1999), ou seja, quando um cliente requisitar uma operação a um dos objetos, a aplicação não irá mais procurar por referências individuais de cada objeto, mas pela referência do grupo como um todo. Os detalhes de como estas referências são obtidas serão vistas no capítulo 3.

1.4 - Validação de Técnicas de Tolerância a Falhas

Vimos nos itens anteriores deste capítulo várias técnicas de tolerância a falhas mais comuns utilizadas em sistemas distribuídos. Mas, quando estamos tratando de sistemas muito complexos e altamente críticos, há uma necessidade de se testar as técnicas de tolerância a falhas, tentando simular situações de falhas mais próximas possíveis da realidade em que serão submetidas. A esses testes damos o nome de injeção de falhas. (WEBER) (LEME , MARTINS & RUBIRA, 199?)

Há três categorias de injeção de falhas, que são mostradas abaixo (WEBER, 2000a) (LEME , MATINS & RUBIRA, 199?):

- Injeção de falhas por *hardware*: Neste tipo de injeção o *hardware* é forçado a erros através de mudanças de valores lógicos no próprio circuito. Exemplo: um determinado pino de um CI (circuito integrado) deve receber o nível lógico 1, e para forçarmos a uma falha, injetamos um nível lógico 0.

- Injeção de falhas por *software*: Neste tipo de injeção são utilizados *softwares* que tentam corromper o código do sistema, modificar características de comunicação, etc.

- Injeção de falhas por simulação: Neste tipo de injeção, há uma simulação de possíveis falhas que possam ocorrer no sistema ainda mesmo em tempo de projeto.

Pela descrição dos tipos de injeção de falhas mais utilizados, podemos perceber que a injeção de falhas por *software* é a mais adequada, pois possui custos mais baixos que a injeção por *hardware* e é o que mais se aproxima das condições de falhas diversas que o sistema em teste irá tratar.