**Tema 12 - Tolerância à falhas em sistemas distribuídos**

Um sistema tolerante a falhas é aquele que mantém a continuidade dos serviços apropriadamente, mesmo diante de falhas em componentes de hardware ou software. Neste tipo de sistema, os defeitos não são perceptíveis aos usuários, pois há mecanismos que detectam, ocultam ou corrigem esses defeitos antes que eles possam comprometer a funcionalidade do sistema, evitando que atinjam o ponto onde as falhas se tornam aparentes. A tolerância a falhas é uma característica de design que permite que um sistema seja resistente a falhas, utilizando um conjunto de técnicas específicas para identificar, mascarar e lidar com falhas. Essas técnicas garantem que o sistema continue operacional mesmo quando ocorrem problemas.

Nesse contexto, uma falha é a ocorrência de um estado não especificado no hardware ou software que pode levar a um erro. Um erro é a manifestação dessa falha dentro do sistema, caracterizada por uma discrepância entre o estado lógico do sistema e o estado esperado. Um defeito é como esse erro se manifesta em termos de comportamento, ou seja, o sistema começa a operar de uma forma que desvia do comportamento esperado. Esses conceitos são fundamentais para o desenvolvimento e operação de sistemas tolerantes a falhas, visando minimizar impactos negativos na experiência do usuário e na integridade do sistema.

Em sistemas distribuídos, uma característica distintiva é a possibilidade de falhas parciais, onde falhas em componentes individuais podem não afetar o sistema como um todo. Isso contrasta com sistemas não distribuídos, onde uma falha geralmente resulta na paralisação completa do sistema. A capacidade de um sistema distribuído de se recuperar automaticamente de falhas parciais e continuar operando de maneira aceitável durante reparos é um aspecto crucial do design desses sistemas, destacando a importância da tolerância a falhas.

A tolerância a falhas está intrinsecamente ligada à confiabilidade do sistema. Um sistema distribuído confiável deve ser capaz de operar continuamente sem falhas, enquanto sistemas altamente disponíveis estão prontos para uso a qualquer momento. Segurança em sistemas distribuídos significa que falhas temporárias não resultam em consequências catastróficas, uma consideração vital para sistemas de controle de processos críticos, como usinas nucleares. A manutenção, por sua vez, refere-se à facilidade com que um sistema pode ser reparado após uma falha.

Falhas são definidas pela incapacidade de um sistema de cumprir com suas funções prometidas, enquanto erros são estados do sistema que podem levar a falhas. Os erros podem ser causados por vários fatores, incluindo meios de transmissão inadequados ou condições ambientais adversas, como mau tempo em redes sem fio. Identificar e corrigir as causas dos erros é fundamental para garantir a confiabilidade e manutenção do sistema.

Falhas são geralmente classificadas como transitórias, intermitentes ou permanentes. Falhas transitórias ocorrem uma vez e depois desaparecem, como quando um pássaro interrompe brevemente o sinal de um transmissor de micro-ondas, causando perda de bits em uma rede. Se a transmissão for refeita, provavelmente funcionará na segunda tentativa. Já as falhas intermitentes aparecem e desaparecem espontaneamente, como um contato solto em um conector, sendo notoriamente difíceis de diagnosticar devido à sua natureza variável. Por fim, falhas permanentes persistem até que o componente defeituoso seja substituído, exemplos incluem chips queimados, bugs de software e falhas de cabeça de disco.

Um sistema que falha não está fornecendo adequadamente os serviços para os quais foi projetado. Em sistemas distribuídos, que são coleções de servidores que se comunicam entre si e com seus clientes, o não fornecimento adequado de serviços pode significar que servidores ou canais de comunicação, ou ambos, não estão funcionando como deveriam. Uma falha em um servidor pode não ser a causa do problema se esse servidor depender de outros para fornecer seus serviços, então a causa do erro pode estar em outro lugar. Relações de dependência são comuns em sistemas distribuídos; por exemplo, um disco rígido defeituoso pode afetar a performance de um servidor de arquivos projetado para oferecer um sistema de arquivos altamente disponível, afetando potencialmente o funcionamento de uma base de dados distribuída se apenas parte dos dados estiver acessível.

Falhas por queda ocorrem quando um servidor para de funcionar prematuramente, mas estava operando corretamente até parar. Após parar, o servidor não dá mais sinais de funcionamento, com a única solução sendo reiniciar. Falhas de omissão acontecem quando um servidor não responde a uma solicitação. Isso pode ser devido a uma falha de omissão de recebimento, onde o servidor nunca recebeu a solicitação, ou uma falha de omissão de envio, onde o servidor falha ao enviar uma resposta após processar um serviço. Essa falha pode ocorrer, por exemplo, se o buffer de envio transborda. Outras falhas de omissão não relacionadas à comunicação podem ser causadas por erros de software, como loops infinitos ou má gestão de memória, fazendo com que o servidor "trave".

Falhas de temporização ocorrem quando a resposta está fora de um intervalo de tempo real especificado. Um servidor que responde muito tarde, por exemplo, gera uma falha de desempenho. Falhas de resposta, onde a resposta do servidor é simplesmente incorreta, podem ser subdivididas em falhas de valor, onde o servidor fornece uma resposta errada, e falhas de transição de estado, onde o servidor reage de forma inesperada a uma solicitação. Falhas arbitrárias ou bizantinas são o tipo mais grave de falhas, onde o servidor pode produzir saídas que jamais deveria produzir, às vezes até de maneira maliciosa em colaboração com outros servidores para fornecer respostas intencionalmente erradas.

Para que um sistema seja tolerante a falhas, sua melhor estratégia é tentar ocultar a ocorrência de falhas dos outros processos. A técnica chave para mascarar falhas é usar redundância, que pode ser de três tipos: de informação, temporal e física. A redundância de informação envolve adicionar bits extras aos dados transmitidos para recuperar bits corrompidos, como o uso de códigos de Hamming para corrigir ruídos na linha de transmissão. A redundância temporal envolve repetir uma ação se necessário, como no caso de transações que, se abortadas, podem ser refeitas sem danos. Isso é útil especialmente para falhas transitórias ou intermitentes. A redundância física adiciona equipamentos ou processos extras ao sistema, permitindo que ele tolere a perda ou mau funcionamento de alguns componentes, podendo ser implementada tanto em hardware quanto em software, como adicionar processos extras ao sistema para manter sua funcionalidade mesmo com a falha de alguns deles.

Outra estratégia envolve a replicação de processos, que podem ser organizados de maneira hierárquica ou plana. Na replicação baseada em primário, um processo primário coordena todas as operações e pode ser substituído por um backup em caso de falha. Na replicação de escrita, os processos são organizados de forma plana, sem um único ponto de falha, mas com coordenação distribuída necessária para a tomada de decisões.

Um exemplo de design que utiliza redundância física é a Redundância Modular Tripla (TMR), na qual cada dispositivo é triplicado e seguido por um circuito votante que compara as entradas e produz uma saída com base na maioria. Esse design permite que o sistema tolere falhas de componentes sem afetar o resultado final, pois as falhas são mascaradas pelos votantes, garantindo que as saídas permaneçam corretas mesmo na presença de componentes defeituosos. O TMR também pode ser aplicado de maneira recursiva para aumentar a confiabilidade de componentes específicos, como chips. Embora nem todos os sistemas distribuídos tolerantes a falhas utilizem TMR, a técnica é amplamente aplicável e ilustra claramente como um sistema pode ser tolerante a falhas, diferenciando-se de sistemas cujos componentes individuais são confiáveis, mas cuja organização não tolera falhas.

A principal abordagem para tolerar processos com falhas é organizar vários processos idênticos em um grupo. A propriedade fundamental dos grupos é que, quando uma mensagem é enviada ao grupo, todos os membros recebem-na. Assim, se um processo falhar, outro pode assumir sua função. Os grupos de processos são dinâmicos; novos grupos podem ser criados e antigos desfeitos, e um processo pode entrar ou sair de um grupo durante a operação do sistema. Um processo pode ser membro de vários grupos simultaneamente, necessitando de mecanismos para gerenciar grupos e membros.

Grupos podem ter estruturas internas diferentes. Em alguns, todos os processos são iguais, como na Figura 1(a), e as decisões são tomadas coletivamente. Em outros, existe uma hierarquia, como um processo coordenador e outros operários, como na Figura 1(b). Nesse modelo, solicitações de trabalho são enviadas ao coordenador, que escolhe o operário mais adequado para realizá-las.

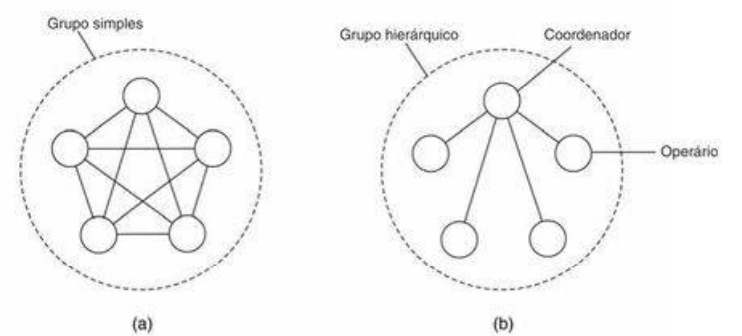


Figura 1: (a) Comunicação em um grupo simples, (b) Comunicação em um grupo hierárquico simples

A comunicação em grupo requer métodos para criar e deletar grupos, além de permitir a entrada e saída de processos. Uma abordagem é usar um servidor de grupo que mantém um banco de dados completo dos grupos e seus membros. Este método é direto e eficiente, mas tem a desvantagem de ser um ponto único de falha. Se o servidor de grupo falhar, a gestão do grupo cessa. A abordagem oposta é gerenciar a adesão ao grupo de forma distribuída, usando multicast confiável para anunciar a entrada no grupo e uma mensagem de despedida para sair. Deve-se garantir que a entrada e a saída de membros sejam sincronizadas com as mensagens enviadas ao grupo. Finalmente, se muitas máquinas falharem e o grupo não puder mais funcionar, é necessário um protocolo para reconstruir o grupo, mesmo que várias tentativas ocorram simultaneamente.

Os grupos de processos são essenciais para construir sistemas tolerantes a falhas, permitindo mascarar processos defeituosos através da replicação e organização em grupos. Essa replicação pode ser implementada de duas maneiras principais: protocolos baseados em primário ou escrita replicada. No modelo baseado em primário, um processo primário coordena todas as operações de escrita e pode ser substituído por um backup em caso de falha. Na escrita replicada, a replicação ativa e os protocolos baseados em quorum organizam os processos em um grupo plano, aumentando a tolerância a falhas sem um ponto único de falha, embora exija coordenação distribuída.

A quantidade de replicação necessária depende do nível de tolerância a falhas desejado. Um sistema é considerado k tolerante a falhas se puder sobreviver à falha de k componentes e ainda assim atender às suas especificações. Se os componentes falharem silenciosamente, ter k + 1 processos é suficiente para proporcionar k tolerância a falhas. Por outro lado, se os processos exibirem falhas bizantinas, emitindo respostas errôneas ou aleatórias, são necessários no mínimo 2k + 1 processos para alcançar k tolerância a falhas.

Além disso, alcançar um consenso em grupos de processos é crucial, especialmente em situações como eleição de um coordenador, decisões de commit de transações ou sincronização. Isso se torna complexo quando os processos não são perfeitos, pois diferentes suposições sobre o sistema subjacente exigem diferentes soluções. Os sistemas podem ser sincronizados, onde os processos operam em um modo de passo de bloqueio, ou assíncronos, onde não há garantias de ordem ou tempo de entrega de mensagens.

Finalmente, algoritmos de acordo distribuído buscam que todos os processos não defeituosos cheguem a um consenso sobre algum problema em um número finito de passos. Este problema é complicado pela natureza das suposições feitas sobre o sistema. Em muitos sistemas práticos, que assumem comportamento assíncrono e comunicação unicast com atrasos de comunicação ilimitados, é necessário usar entrega de mensagens ordenada e confiável, como a fornecida pelo TCP. A questão do acordo bizantino, que aborda a necessidade de consenso entre várias partes mesmo na presença de traidores, exemplifica a complexidade do acordo distribuído.

Além da replicação, é crucial que haja mecanismos eficazes para detectar falhas. A detecção de falhas geralmente envolve o envio ativo de mensagens de "está vivo?" entre processos ou a espera passiva por comunicações, sendo a primeira abordagem mais comum na prática. Esses mecanismos muitas vezes dependem de temporizadores para identificar falhas, o que pode levar a falsos positivos devido à natureza não confiável das redes.

A detecção de falhas pode também ocorrer como um subproduto da troca regular de informações, como no método de disseminação de informações baseado em fofocas. Neste método, processos periodicamente divulgam sua disponibilidade, e essa informação é espalhada através da rede, permitindo que cada processo determine localmente se outro falhou.

Outro aspecto importante é a capacidade do subsistema de detecção de falhas de distinguir entre falhas de rede e falhas de nós. Uma abordagem é solicitar a outros nós que verifiquem a acessibilidade de um nó presumivelmente falho. Além disso, quando uma falha é detectada, é vital decidir como os processos não defeituosos serão informados. Um exemplo de abordagem para monitoramento de falhas em rede ampla é o uso de uma árvore de expansão para monitorar as falhas dos membros, onde falhas de resposta resultam na rápida propagação de uma notificação de falha do grupo. Esta estratégia reduz a influência de falhas de conexão ao depender de conexões TCP ponto a ponto entre membros do grupo.

A detecção de falhas é fundamental para a tolerância a falhas em sistemas distribuídos. Essencialmente, os membros não falhos de um grupo devem ser capazes de identificar quais processos falharam. Existem dois mecanismos principais para detectar falhas de processo: o envio ativo de mensagens e a espera passiva por mensagens de outros processos. Na prática, a abordagem mais comum é o envio ativo de pings. A detecção de falhas geralmente utiliza um mecanismo de timeout para verificar se um processo falhou, mas isso pode levar a falsos positivos devido à natureza não confiável das redes.

A detecção de falhas também pode ser realizada através da disseminação de informações baseada em fofocas, onde os processos periodicamente informam aos vizinhos sobre sua disponibilidade. Esse método ajuda a distinguir falhas de rede de falhas de nós, solicitando a verificação de outros vizinhos em caso de timeout em uma mensagem ping.

Uma abordagem prática é o sistema FUSE, onde os membros do grupo criam uma árvore de abrangência usada para monitorar falhas dos membros. Se um vizinho não responder a um ping, o nó que enviou o ping assume imediatamente que houve uma falha e para de responder a pings de outros nós, promovendo rapidamente uma notificação de falha no grupo. Este sistema minimiza os problemas de falhas de link por depender de conexões TCP ponto a ponto entre os membros do grupo.

Na construção de sistemas de comunicação tolerantes a falhas em sistemas distribuídos, cinco tipos principais de falhas podem ocorrer, cada uma exigindo abordagens específicas para mitigação:

1. **Incapacidade do Cliente de Localizar o Servidor:** Este problema ocorre quando um cliente não consegue encontrar um servidor adequado, o que pode ser causado por todos os servidores estarem inativos ou mudanças nas interfaces do servidor que tornam o cliente desatualizado. Uma solução comum é lançar uma exceção que informe ao cliente sobre a falha na localização do servidor.
2. **Mensagens de Solicitação Perdidas:** Isso acontece quando uma mensagem do cliente para o servidor é perdida durante a transmissão. Normalmente, um mecanismo de temporização é usado para reenviar a mensagem se uma confirmação não for recebida em um tempo esperado. Isso ajuda a garantir que a mensagem eventualmente chegue ao servidor, a menos que ocorra uma falha mais grave.
3. **Falha do Servidor após Receber uma Solicitação:** Se um servidor falhar após receber uma solicitação mas antes de responder, é necessário que o sistema identifique essa falha e informe ao cliente, ou retransmita a solicitação se o servidor falhar antes de processar a solicitação.
4. **Mensagens de Resposta Perdidas:** Semelhante às solicitações perdidas, as respostas do servidor ao cliente também podem se perder. A solução envolve retransmissões e a utilização de temporizadores para garantir que o cliente reenvie sua solicitação se uma resposta não for recebida, considerando que a operação possa ser idempotente ou que medidas adicionais, como sequências de números de solicitação, sejam usadas para evitar operações duplicadas.
5. **Falha do Cliente após Enviar uma Solicitação:** Se um cliente falhar após enviar uma solicitação, pode resultar em uma computação órfã no servidor, que pode continuar a processar uma operação sem que haja uma parte interessada para receber o resultado. Estratégias para lidar com orfandades incluem exterminação de órfãos, reincarnação (onde computações órfãs são terminadas ao detectar o reinício do cliente), e expiração, que limita o tempo de vida de uma RPC.

A resiliência de processos por meio da replicação é fundamental, assim como os serviços de multicast confiáveis, que garantem a entrega de mensagens a todos os membros de um grupo de processos. Apesar disso, o multicasting confiável apresenta desafios significativos. Embora a maioria das camadas de transporte ofereça canais ponto a ponto confiáveis, raramente fornecem comunicação confiável para um conjunto de processos. Uma solução comum é permitir que cada processo estabeleça uma conexão ponto a ponto com cada processo com o qual deseja se comunicar. Esta abordagem pode não ser eficiente, desperdiçando largura de banda da rede, mas se o número de processos for pequeno, essa solução simples pode ser eficaz.

Multicasting confiável significa que uma mensagem enviada a um grupo de processos deve ser entregue a cada membro do grupo. Desafios surgem se um processo se juntar ao grupo durante a comunicação ou se o processo remetente falhar. Nestes casos, deve-se garantir que todos os membros não falhos do grupo recebam a mensagem, e é necessário um consenso sobre a composição do grupo antes da entrega da mensagem, além de restrições de ordenação.

Se assumirmos que os processos não falham e que não há alterações no grupo durante a comunicação, o multicasting confiável significa simplesmente que toda mensagem deve ser entregue a cada membro atual do grupo. A implementação desta forma de multicasting é relativamente fácil, contanto que o número de receptores seja limitado.

Considere um sistema de comunicação que oferece apenas multicasting não confiável, onde uma mensagem multicast pode ser perdida e entregue apenas a alguns dos receptores pretendidos. Uma solução simples envolve o processo remetente atribuir um número de sequência a cada mensagem e armazená-la localmente em um buffer de histórico. O remetente mantém a mensagem no buffer até receber um reconhecimento de cada receptor. Se um receptor detectar que está faltando uma mensagem, ele pode enviar um reconhecimento negativo solicitando a retransmissão. O remetente também pode optar por retransmitir automaticamente a mensagem se não receber todos os reconhecimentos dentro de um determinado tempo. As retransmissões podem ser feitas ponto a ponto ou por meio de um único multicast para todos os processos.

O esquema de multicast confiável enfrenta desafios significativos, especialmente ao lidar com um grande número de receptores. Quando muitos receptores estão envolvidos, o remetente pode ficar sobrecarregado com um grande volume de mensagens de feedback, um fenômeno conhecido como implosão de feedback. Além disso, há a consideração adicional de que os receptores podem estar distribuídos em uma ampla rede geográfica. Uma solução proposta é evitar que os receptores enviem confirmações de recebimento de mensagens. Em vez disso, eles só enviariam mensagens de feedback negativo para informar ao remetente sobre a falta de uma mensagem. Essa abordagem de apenas reconhecimentos negativos tende a escalar melhor, mas não garante completamente a ausência de implosões de feedback. Outro desafio é que o remetente pode precisar manter indefinidamente as mensagens em seu buffer de histórico, pois nunca pode ter certeza de que uma mensagem foi entregue corretamente a todos os receptores. Na prática, as mensagens são eventualmente removidas do buffer para evitar o transbordamento, correndo o risco de não poder atender a pedidos futuros de retransmissão. Existem várias propostas para tornar o multicast confiável mais escalável, e uma comparação entre diferentes esquemas pode ser encontrada na literatura especializada.

A chave para soluções escaláveis em multicasting confiável é reduzir o número de mensagens de feedback retornadas ao remetente. Um modelo popular utilizado em várias aplicações de grande área é a supressão de feedback. Esse esquema, que fundamenta o protocolo Scalable Reliable Multicasting (SRM) desenvolvido por Floyd et al. (1997), funciona da seguinte forma: no SRM, os receptores nunca confirmam o recebimento bem-sucedido de uma mensagem, mas apenas relatam quando uma mensagem está faltando. A detecção de perda de mensagem é responsabilidade da aplicação, e apenas reconhecimentos negativos são enviados como feedback. Quando um receptor percebe que uma mensagem foi perdida, ele transmite seu feedback para o resto do grupo. Isso permite que outro membro do grupo suprima seu próprio feedback. Se vários receptores não receberem a mensagem m, cada um enviará um reconhecimento negativo ao remetente S, para que m seja retransmitida. No entanto, assumindo que as retransmissões sejam sempre enviadas ao grupo inteiro, é suficiente que apenas uma solicitação de retransmissão chegue a S.

Por essa razão, um receptor R que não recebeu a mensagem m agenda uma mensagem de feedback com um atraso aleatório. A solicitação de retransmissão não é enviada até que algum tempo aleatório tenha passado. Se, nesse ínterim, outra solicitação de retransmissão para m chegar a R, R suprimirá seu próprio feedback, sabendo que m será retransmitida em breve. Idealmente, apenas uma mensagem de feedback chegará a S, que, por sua vez, retransmitirá m.

A supressão de feedback, utilizada em várias aplicações colaborativas na Internet, é escalável, mas apresenta desafios. Exige que os feedbacks sejam agendados com precisão para evitar respostas simultâneas de múltiplos receptores, um processo complexo em redes de grande área. Além disso, o multicast de feedback pode interromper receptores que já receberam a mensagem, forçando-os a processar informações desnecessárias. Uma solução é criar grupos de multicast separados para quem não recebeu a mensagem, embora isso exija uma gestão eficiente do grupo, o que é desafiador. Alternativamente, receptores que frequentemente perdem mensagens podem compartilhar um canal para feedbacks e retransmissões. Adicionalmente, para melhorar a escalabilidade, receptores que receberam a mensagem podem iniciar a retransmissão antes de o pedido chegar ao remetente original. Essas estratégias, detalhadas em trabalhos como os de Liu et al. (1998) e Floyd et al. (1997), permitem uma recuperação local mais eficaz.

A supressão de feedback é uma solução não hierárquica para multicasting confiável, mas a escalabilidade em grupos muito grandes exige uma abordagem hierárquica. Nesse modelo, o grupo de receptores é dividido em subgrupos organizados em uma estrutura de árvore, com o subgrupo do remetente no topo. Cada subgrupo designa um coordenador local responsável por gerenciar solicitações de retransmissão dentro de seu grupo, possuindo um buffer de histórico próprio. Se o coordenador não recebe uma mensagem, solicita a retransmissão ao coordenador do subgrupo pai. O coordenador remove a mensagem do buffer após receber confirmações de todos os membros do seu subgrupo e de subgrupos filhos. Construir essa estrutura de árvore dinamicamente e adaptar roteadores de multicast para funcionar como coordenadores locais são desafios significativos. Assim, soluções de multicasting no nível de aplicação têm se popularizado, embora a construção de esquemas de multicast confiável que escalonem bem em redes de grande área continue sendo um problema complexo, sem uma solução única ideal.

Na presença de falhas de processo, a multicasting confiável em sistemas distribuídos necessita garantir que uma mensagem seja entregue a todos os processos ou a nenhum. Além disso, é essencial que todas as mensagens sejam entregues na mesma ordem a todos os processos, um conceito conhecido como multicast atômico. Essa característica é crucial em aplicações como bancos de dados replicados, onde cada réplica do banco de dados é tratada como um processo separado dentro de um grupo. Operações de atualização são enviadas via multicast a todas as réplicas e executadas localmente. Se uma réplica falhar durante uma atualização, a operação é perdida apenas para aquela réplica, mas continua válida para as outras. Quando a réplica falhada se recupera, é fundamental sincronizá-la com o estado atual das outras réplicas. O multicast atômico garante que uma operação de atualização enviada antes de uma falha seja realizada por todas as réplicas não falhas ou por nenhuma, dependendo do consenso sobre a participação da réplica falhada no grupo. Assim, quando a réplica falhada se recupera, ela deve se reintegrar ao grupo e atualizar seu estado para garantir uma visão consistente do banco de dados entre todas as réplicas.

O multicast confiável em sistemas distribuídos, na presença de falhas de processo, pode ser definido por grupos de processos e mudanças na sua composição. A distinção entre receber e entregar mensagens é crucial. Mensagens são enviadas e recebidas em uma camada de comunicação e ficam armazenadas localmente até serem entregues à aplicação correspondente em uma camada superior.

O conceito de multicast atômico implica que cada mensagem de multicast está associada unicamente a uma lista de processos destinatários, que compartilham uma visão comum do grupo no momento do envio da mensagem. Se um processo se junta ou deixa o grupo durante o multicast, uma mensagem de alteração de visão é enviada, garantindo que todas as mudanças sejam reconhecidas antes da entrega da mensagem original ou que a mesma não seja entregue.

Apenas em um cenário onde a mudança de membros do grupo resulta da falha do remetente da mensagem é que o envio pode falhar; nesse caso, a mensagem pode ser ignorada por todos. Este formato de multicast confiável, conhecido como sincronia virtual, assegura que mensagens enviadas para uma visão de grupo sejam entregues a todos os processos não falhos, ou ignoradas por todos se o remetente falhar durante o envio.

Este princípio garante que todas as mensagens em trânsito sejam processadas antes de uma mudança na visão do grupo, funcionando como uma barreira sincronizadora. Isso é similar ao uso de variáveis de sincronização em armazenamentos de dados distribuídos. Quando ocorrem mudanças na composição do grupo, todas as comunicações pendentes são completadas antes que novas alterações no grupo sejam efetivadas.

Nesse contexto, a sincronia virtual permite que desenvolvedores de aplicações considerem multicasts como ocorrendo em épocas separadas por mudanças na composição dos grupos. Contudo, diferentes tipos de ordenação de multicasts podem ser aplicados:

1. **Multicasts Desordenados:** Não há garantias sobre a ordem de entrega das mensagens recebidas pelos diferentes processos.
2. **Multicasts Ordenados FIFO (First In, First Out):** Mensagens enviadas pelo mesmo processo devem ser entregues na ordem em que foram enviadas.
3. **Multicasts Ordenados Causalmente:** Mensagens são entregues de forma que a causalidade potencial entre diferentes mensagens seja preservada. Por exemplo, se uma mensagem m1 precede causalmente outra mensagem m2, m1 será sempre entregue antes de m2.
4. **Multicasts Totalmente Ordenados:** Independentemente da ordem de envio (desordenada, FIFO ou causal), as mensagens devem ser entregues na mesma ordem a todos os membros do grupo.

Essas ordens garantem que, mesmo em um sistema distribuído, as mensagens sejam entregues de forma consistente e confiável, respeitando as dependências de ordem necessárias para a correta execução das operações no sistema.

A tolerância a falhas é um aspecto importante dos sistemas distribuídos, onde a capacidade de continuar operando apesar de falhas parciais ou totais de componentes é fundamental para a confiabilidade e disponibilidade do sistema. Um sistema distribuído bem projetado emprega várias estratégias para detectar, mascarar e corrigir falhas, permitindo que o sistema mantenha a operação normal e minimize o impacto nas atividades do usuário. A redundância, seja de informação, tempo ou física, desempenha um papel essencial nesse processo, garantindo que falhas em componentes individuais não comprometam a funcionalidade geral do sistema.

A complexidade de gerenciar e manter a consistência em tais sistemas é agravada pela natureza dinâmica dos mesmos, onde os componentes podem entrar e sair do sistema frequentemente. Portanto, mecanismos robustos para a gestão de grupos e para a comunicação confiável e ordenada entre processos são imprescindíveis. As técnicas de multicasting confiável, como o multicasting atômico e a sincronia virtual, são exemplos de como os sistemas distribuídos lidam com a comunicação entre processos distribuídos geograficamente, garantindo que todas as mensagens sejam entregues de forma consistente e ordenada, apesar das falhas.

Dessa forma, a implementação de sistemas distribuídos que são tanto confiáveis quanto tolerantes a falhas exigem um entendimento profundo das potenciais fontes de erro e uma abordagem meticulosa para o design de comunicação e controle de processos. Ao adotar estas estratégias, os desenvolvedores podem assegurar que os sistemas distribuídos sejam capazes de suportar e se recuperar de falhas, mantendo a integridade e a disponibilidade dos serviços essenciais para os usuários finais.

**Referências**

Livro Sistemas Distribuídos Princípios e Paradigmas - Tanembaum (capítulo 8)