Universidade Estadual de Mato Grosso do Sul
Curso de Ciência da Computação
Disciplina de Algoritmos Paralelos e Distribuídos

Tolerância a Falhas

- Conceitos Básicos
  - Existe uma forte relação entre ser tolerante a falhas e sistemas confiáveis.
  - Confiabilidade abrange uma série de requisitos úteis para sistemas distribuídos, como por exemplo:
    - Disponibilidade;
    - Confiabilidade;
    - Segurança;
    - Capacidade de Manutenção;

Disponibilidade

- Propriedade de um sistema estar pronto para ser usado imediatamente;
- Probabilidade do sistema estar funcionando corretamente em qualquer momento determinado e estar disponível para executar suas funções em nome de seus usuários;
- Alta disponibilidade → Mais provável de estar funcionando em dado instante de tempo.

Confiabilidade

- Propriedade de um sistema poder funcionar continuamente sem falha.
- É definida em termos de um intervalo de tempo em vez de um instante de tempo.
- Alta confiabilidade → Mais provável de continuar a funcionar sem interrupção durante um período de tempo relativamente longo.
- Se um sistema fica fora do ar por um milisegundo a cada hora, terá uma disponibilidade de mais de 99,99%, mas sua confiabilidade ainda será baixa.

Segurança

- Se um sistema deixar de funcionar corretamente durante um certo tempo, nada de catastrófico ocorrerá.
  - Exemplo: Sistemas de controle de processos usados em usinas de energia nuclear.

Capaciadade de Manutenção

- Facilidade com que um sistema que falhou possa ser consertado.

- Sistemas de alta capacidade de manutenção também podem mostrar alto grau de disponibilidade, em especial se as falhas puderem ser detectadas e reparadas automaticamente.

Mais Conceitos

- Defeito: É quando o sistema não pode executar o que foi especificado.
- Erro: Estado de um sistema causado por uma falha.
  - Meio de transmissão errado ou ruim pode danificar pacotes;
  - Alguns erros de transmissão podem ser causados por más transmissões atmosféricas, principalmente em redes sem fio.

Tipos de Falhas

- Transiente:
  - Ocorre uma vez e depois desaparece.
  - Se a operação for repetida, a falha não acontecerá novamente.
- Intermitente:
  - Ocorre e desaparece por "sua própria vontade".
  - Exemplo: Conector com problemas.
- Permanente:
  - Continua a existir até que o componente seja substituído.

Modelos de Falhas

- Diferentes tipos de falhas:

Tipo de falha	Descrição
Falha por queda	O servidor pára de funcionar, mas estava funcionando corretamente até parar.
Falha por omissão *Omissão de recebimento* *Omissão de envio*	O servidor não consegue responder a requisições que chegam O servidor não consegue receber mensagens que chegam O servidor não consegue enviar mensagens
Falha de temporização	A resposta do servidor se encontra fora do intervalo de tempo
Falha de resposta *Falha de valor* *Falha de transição de estado*	A resposta do servidor está incorreta O valor da resposta está errado O servidor se desvia do fluxo de controle correto
Falha arbitrária	Um servidor pode produzir respostas arbitrárias em momentos arbitrários

Mascaramento de Falha por Redundância

- Técnicas para mascarar falhas:
  - Redundância de Informação
    - Bits extras são adicionados para permitir recuperação de bits deteriorados.
  - Redundância de Tempo
    - Uma ação é realizada e, então, se for preciso, ela é executada novamente.
  - Redundância Física
    - Componentes físicos replicados são usados.

Estratégias de Tolerância a Falhas

- Resiliência de Processos:
  - Replicação de processos em grupos;
  - Grupos simples ou hierárquicos;
- Comunicação confiável cliente-servidor:
  - Falhas de comunicação;
  - Canal de comunicação pode exibir falhas por queda, por omissão, entre outros.
  - Tcp, Rpc, etc.
- Comunicação confiável de grupo:
  - Como implementar entrega confiável de mensagens a todos os processos?
- Comprometimento distribuído:
  - Envolve a realização de uma operação por cada membro de um grupo de processos ou por absolutamente nenhum.
    - Exemplo: Entrega de mensagens.

Resiliência de Processos

COMUNICAÇÃO CONFIÁVEL CLIENTE-SERVIDOR

Resiliência de Processos

- Replicar processos em grupos para proteção contra falhas de processos.
- Como fazer?
  - Questões de projeto.
    - Grupos simples versus grupos hierárquicos.
    - Associação a um grupo.
  - Mascaramento de falhas e replicação.
  - Acordos em sistemas com falha
  - Detecção de falha

Resiliência de Processos

- Processos idênticos são organizados em grupos.
  - Quando uma mensagem é enviada a um grupo, todos os membros a recebem.
  - Se um processo do grupo falhar, espera-se que um outro se encarregue do tratamento da mensagem.
- Grupos de processos são dinâmicos
  - Grupos podem ser criados, processos podem se mover entre grupos…
- Exemplo: Um processo pode enviar um mensagem a um grupo de servidores sem precisar saber quem eles são ou quantos existem.

Resiliência de Processos
Estrutura dos Grupos

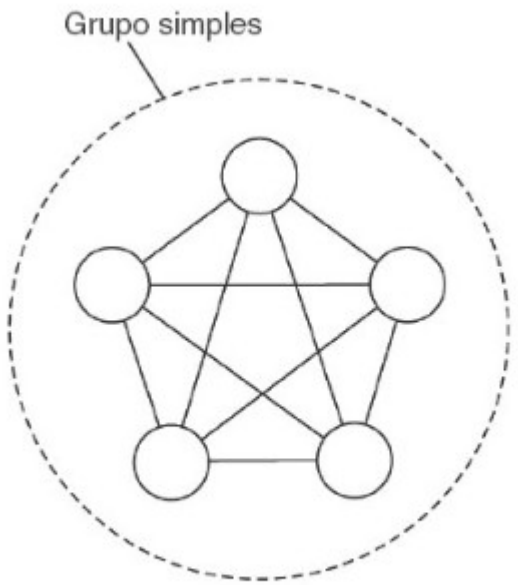
- Grupo Simples:
  - Todos os processos são iguais;
  - Decisões são tomadas coletivamente;
  - Vantagens:
    - Não tem ponto de falha único;
    - Mesmo que um processo caia, o grupo continua a oferecer o serviço.
  - Desvantagem:
    - Tomada de decisão pode ser complicada, às vezes com necessidade de votação.

Resiliência de Processos
Estrutura dos Grupos

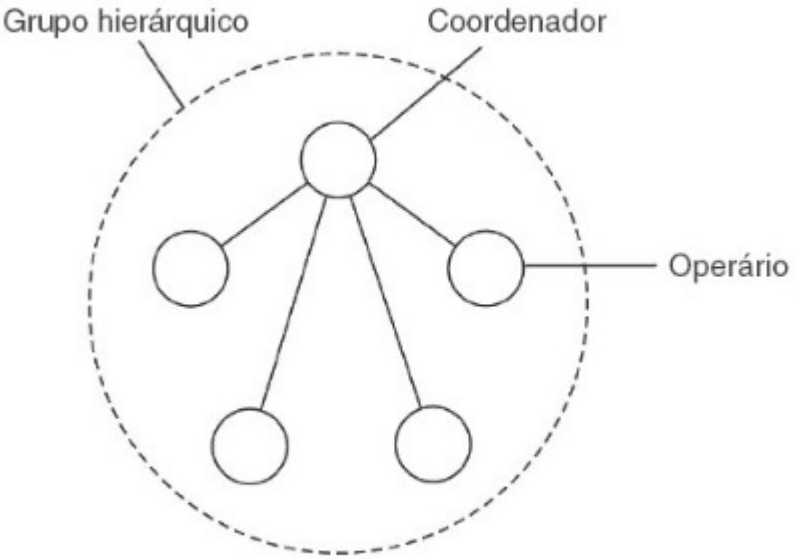
- Grupo Hierárquico:
  - Existe um processo coordenador, os demais são denominados operários;
  - Sempre que uma requisição é gerada, esta é enviada ao coordenador.
  - O coordenador decide qual é o operário mais adequado para executá-la.
  - Vantagens: Decisões são centralizadas.
  - Desvantagens: Caso o coordenador falhe, o serviço falhará.

Resiliência de Processos
Estrutura dos Grupos

- a) Comunicação em um grupo simples;
- b) Comunicação em um grupo hierárquico simples;



(a)



(b)

Resiliência de Processos
Associação a Grupos

- Como criar e eliminar grupos?

- Como permitir que processos se juntem e saiam de grupos?

Resiliência de Processos
Associação a Grupos

- Servidor de grupos
  - Todas as requisições são enviadas a este servidor;
  - Mantém um banco de dados completo de todos os grupos e seus associados;
  - Método direto e eficiente;
- Se o servidor de grupo cair, o gerenciamento deixa de existir.
  - Os grupo deverão ser reconstituídos do zero.

Resiliência de Processos
Associação a Grupos

- Gerenciamento Distribuído
  - Se existe multicast confiável, um processo pode enviar uma mensagem a todos os membros do grupo anunciando que deseja se juntar ao grupo.
  - Para sair de um grupo, o processo deveria mandar uma mensagem.
  - Tanto entrar como sair de um grupo deve ser síncrono com as mensagens enviadas e recebidas.

Mascaramento de Falhas e Replicação

- Terminologia:
  - Quando um grupo de processos deseja mascarar **k** falhas simultaneamente, este é classificado como um grupo **k-tolerante** a falhas (k-fault tolerante)

- A questão é: o quão grande k precisa ser?

Mascaramento de Falhas e Replicação

1) Falhas silenciosas

Se **k falhas** pararem sem propagar informações erradas, basta ter **k+1** processos.

2) Se os processos exibirem falhas e continuarem a enviar respostas erradas às requisições, será preciso um mínimo de 2k+1 processadores para conseguir k-tolerância.

Acordo em Sistemas com Falha

- Em muitos casos, um grupo de processos deve chegar ao acordo: eleger um coordenador, decidir a validação de uma transação.

- Problema: Caso em que os processos não podem ser considerados perfeitos.

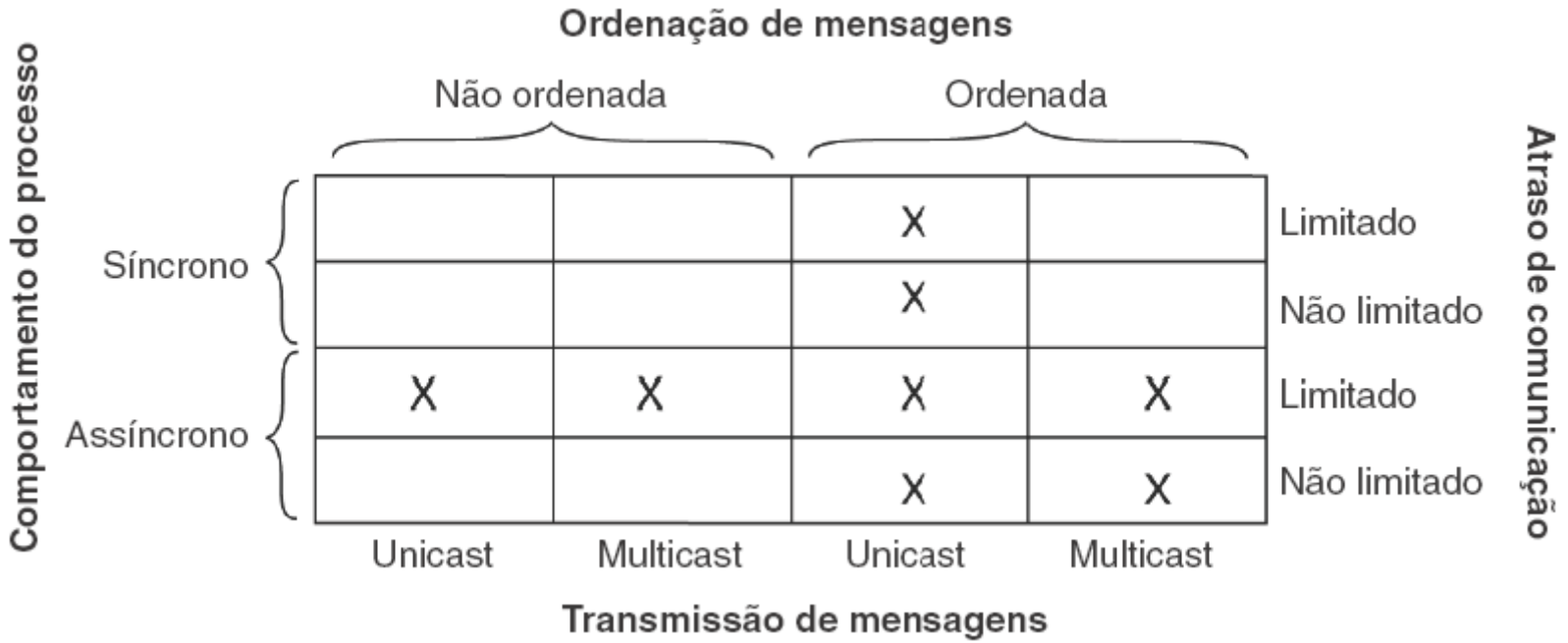
Acordo em Sistemas com Falhas

- Objetivo: todos os processos que não apresentam falhas devem chegar a um consenso sobre alguma questão dentro de um número finito de etapas.

- Premissas diferentes sobre o sistema requerem soluções diferentes.

Acordo em Sistemas com Falha

- Circunstânicias sob as quais se pode chegar a um acordo distribuído.
- Mais comum: assíncrono, unicast e atrasos não limitados.

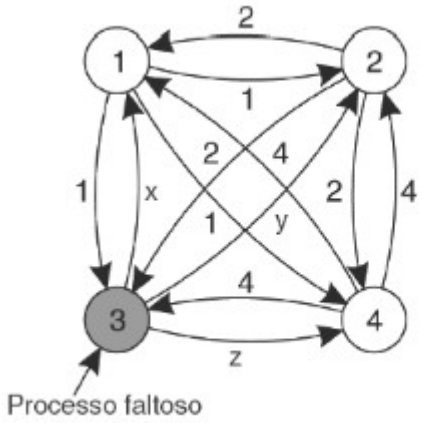


Acordo em Sistemas com Falha

- Segundo Lamport:
  - Premissa: Processos síncronos, mensagens unicast, ordenação preservada e o atraso de comunicação limitada.
  - Sistema: *N* processos e cada processo *i* fornece um valor *V_i* aos demais.
  - Objetivo: Cada processo deve construir um vetor V de comprimento *N* tal que, se o processo *i* não for faltoso, *V[i]* = *V_i*. Caso contrário, *V[i]* é indefinido.

Acordo em Sistemas com Falha

- Problema do acordo bizantino para três processos não faltosos e um faltoso.
  - a) Cada processo envia seu valor aos outros;
  - b) Vetores que cada processo monta com base em a);
  - c) Vetores que cada processo recebe na etapa 3.



(a)

1	Obteve (1, 2, *x*, 4)
2	Obteve (1, 2, *y*, 4)
3	Obteve (1, 2, 3, 4)
4	Obteve (1, 2, *z*, 4)

(b)

1 Obteve	2 Obteve	4 Obteve
(1, 2, *y*, 4)	(1, 2, *x*, 4)	(1, 2, *x*, 4)
(*a*, *b*, *c*, *d*)	(*e*, *f*, *g*, *h*)	(1, 2, *y*, 4)
(1, 2, *z*, 4)	(1, 2, *z*, 4)	(*i*, *j*, *k*, *l*)

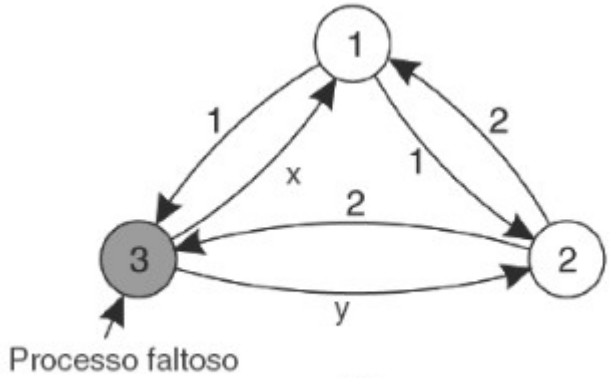
(c)

Acordo em Sistemas com Falha

- Em um sistema com *k* processos faltosos, pode-se conseguir um acordo somente se estiverem presentes *2k+1* processos funcionando corretamente, para um total de *3k+1*.
- Um acordo somente é possível se mais do que dois terços dos processos estiverem funcionando adequadamente.

Acordo em Sistemas com Falha

- Dois processos corretos e um processo faltoso.



(a)

1 Obteve (1, 2, *x*)
2 Obteve (1, 2, *y*)
3 Obteve (1, 2, 3)

(b)

1 Obteve	2 Obteve
(1, 2, *y*)	(1, 2, *x*)
(*a*, *b*, *c*)	(*d*, *e*, *f*)

(c)

Acordo em Sistemas com Falha

- Chegar a um acordo é muito complicado.
- Fischer:
  - Provaram que em um sistema distribuído no qual não se pode garantir que as mensagens sejam entregues dentro de um tempo conhecido e finito, nenhum acordo é possível, mesmo que só um processo seja faltoso.
- Como saber se um processo caiu ou se está lento?

Detecção de Falhas

- Para ter sistemas tolerantes a falhas, devemos primeiramente detectar a falha propriamente dita.
  - Processos enviam ativamente uns aos outros mensagens keep alive.
  - Ou esperam passivamente pela entrada de mensagens de processos diferentes.
  - Abordagem mais utilizada: ping.

Detecção de Falhas

- Essência: Falhas são detectadas através de mecanismos de timeouts.
  - Definir timeouts de maneira eficiente é muito difícil e depende da aplicação.
  - É difícil distinguir falhas dos processos das falhas da rede.
- Possível Solução:
  - Considerar possíveis notificações de falhas entre os membros do sistema.
    - Gossiping
    - Árvores onde os processos fingem que falharam.

Comunicação Confiável Cliente Servidor

- Falhas de comunicação?
  - Canais de comunicação podem exibir falhas por queda, por omissão, de temporização e arbitrárias.
  - Na construção de canais de comunicação: evitar falhas por queda e omissão.
  - Falhas arbitrárias podem ocorrer sob a forma de mensagens duplicadas.

Comunicação Confiável Cliente Servidor e Ponto a Ponto

- RPC confiável:
  1) Cliente não consegue localizar o servidor;
  2) A mensagem de requisição do cliente para o servidor se perde;
  3) O servidor cai após receber uma requisição;
  4) A mensagem de resposta do servidor para o cliente se perde;
  5) O cliente cai após enviar uma requisição.

Comunicação Confiável Cliente Servidor e Ponto a Ponto

- A mensagem de requisição do cliente para o servidor se perde.
  - Esta é a situação mais simples. O controle deve ser feito no servidor.
    - Se a mensagem foi realmente perdida, o servidor não conseguirá perceber a diferença entre a retransmissão e a mensagem original.
    - Se a requisição foi perdida.
      - Permitir que o servidor consiga detectar que está lidando com retransmissão.

Comunicação Confiável Cliente Servidor e Ponto a Ponto

- A mensagem de resposta do servidor para o cliente se perde.
  - Situação mais complicada.
    - Cliente não sabe com certeza o motivo da falta de resposta: a requisição ou resposta se perdeu ou é o servidor que está lento?
    - Se a informação for do tipo idempotente (que pode ser executada diversar

Comunicação Confiável Cliente Servidor e Ponto a Ponto

- A mensagem de resposta do servidor para o cliente se perde.
  - Cliente gera um número de sequência a cada requisição.
  - Se o servidor monitorar o número de sequência mais recentemente recebido de cada cliente que o está usando, poderá distinguir entre uma requisição original e uma retransmissão e poderá se recursar a executar qualquer requisição uma segunda vez.

Comunicação Confiável de Grupo

- A camada de transporte oferece comunicação ponto-a-ponto confiável (tcp).
  - Raro oferecer comunicação confiável a um conjunto de processos.
  - Possível solução: Estabelecer comunicações ponto-a-ponto entre os processos.
  - Desperdício de largura de banda da rede.

Solução Básica de Multcast Confiável

- Cenário 1
  - Processos não falham e não se juntam ao grupo nem saem dele enquanto a comunicação está em curso.
  - Neste caso, multicast confiável significa que toda mensagem deve ser entregue a cada membro do grupo no momento em questão.

Solução Básica de Multcast Confiável

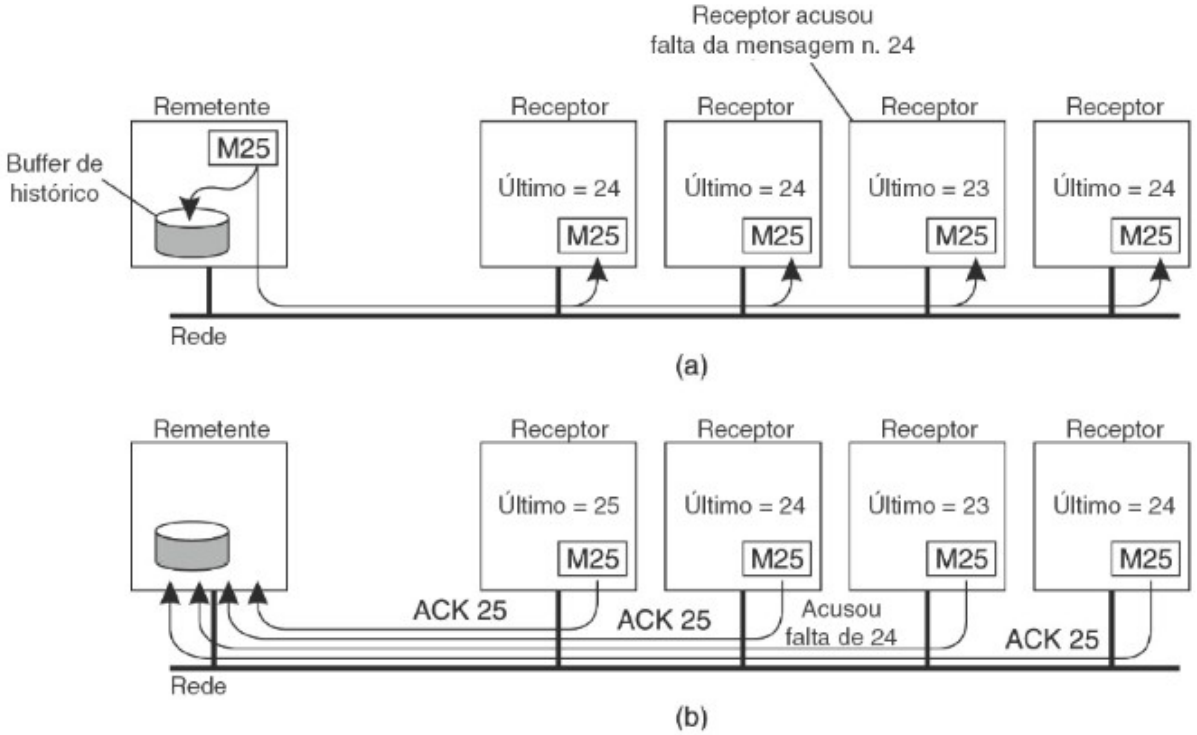
- Consideremos o caso em que um único remetente queira enviar uma mensagem multicast a vários receptores.
- Lembrando:
  - Rede de comunicação não confiável – mensagem multicast pode se perder em algum ponto do caminho e ser entregue a alguns, mas não a todos os receptores.

Solução Básica de Multcast Confiável

- Solução 1
  - Processo remetente designa um número de sequência a cada mensagem multicast.
  - Mensagens são recebidas na ordem.
  - Cada mensagem multicast é armazenada no remente em um buffer.
  - A mensagem é mantida até que receptores confirmem o recebimento.
  - Retransmissão: Reconhecimento negativo ou timeout.

Solução Básica de Multcast Confiável

- Solução simples para multicast confiável quando todos os receptores são conhecidos.
  - A premissa é que nenhum falhe.
  - a) Transmissão de mensagem.
  - b) Realimentação de relatório.



Solução Básica de Multicast Confiável

- Questões em aberto
  - Como reduzir o número de mensagens retornadas ao remetente?
  - Como fazer a retransmissão?
    - Ponto a ponto ou multicast?
  - Como fica a escalabilidade?
- Se houver *N* receptores, o remetente deve estar preparado para aceitar no mínimo *N* reconhecimentos.

Solução para Escalabilidade em Multicast Confiável

- Solução 2
  - Receptores enviam mensagens de retorno somente para informar ao remetente a falta de uma mensagem.
  - Retornar somente reconhecimentos negativos melhora a escalabilidade, mas não garante que implosões de retorno nunca acontecerão.
  - Problema:Remetente será forçado a manter uma mensagem em seu buffer de histórico para "sempre".

Soluções para Escalabilidade em Multicast Confiável

- Solução 3:
  - Objetivo: Reduzir mensagens de retorno.
  - Utiliza o protocolo de multicast confiável escalável (SRM).
  - Somente reconhecimentos negativos são devolvidos como realimentação.
  - Ao reconhecer que está faltando uma mensagem, o receptor envia o pedido da mensagem perdida, usando multicast, ao resto do grupo.

Soluções para Escalabilidade em Multicast Confiável

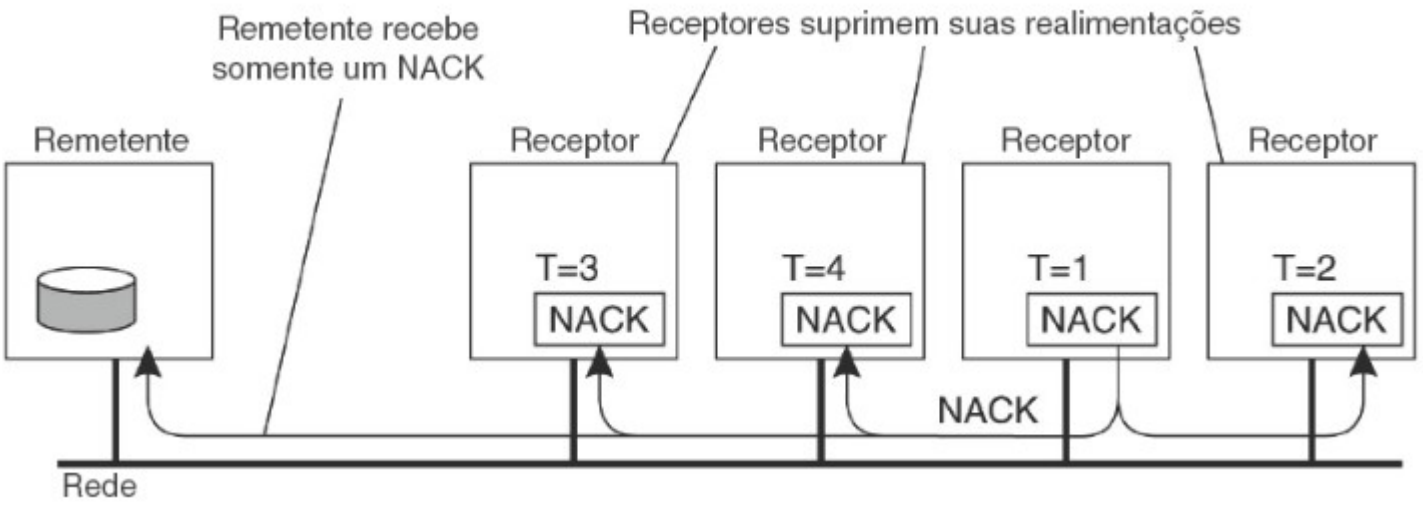
- Solução 3:
  - Permite que um outro membro do grupo suprima seu reconhecimento (realimentação).
  - Suponha que vários receptores tenham percebido a falta da mensagem *m*.
  - Cada um deles precisará retornar um reconhecimento negativo ao remetente *S*, para a retransmissão de *m*.
  - Se considerarmos que as retransmissões são enviadas ao grupo, basta uma única mensagem de requisição para que a retransmissão chegue até S.

Soluções para Escalabilidade em Multicast Confiável

- Solução 3:
  - Um receptor *R* que não recebeu a mensagem *m* escalona uma mensagem de realimentação com certo atraso aleatório.
  - A requisição para retransmissão não é enviada até passar algum tempo aleatório.
  - Se uma requisição de m chegar a *R*, antes do *timeout*, *R* não enviará a mensagem de realimentação negativa.
  - Espera-se que somente uma mensagem de pedido de realimentação de m chegue a *S*.

Soluções para Escalabilidade em Multicast Confiável

- Vários receptores escalonam uma requisição para retransmissão, mas a primeira requisição de retransmissão resulta na supressão de outras.



Soluções para Escalabilidade em Multicast Confiável

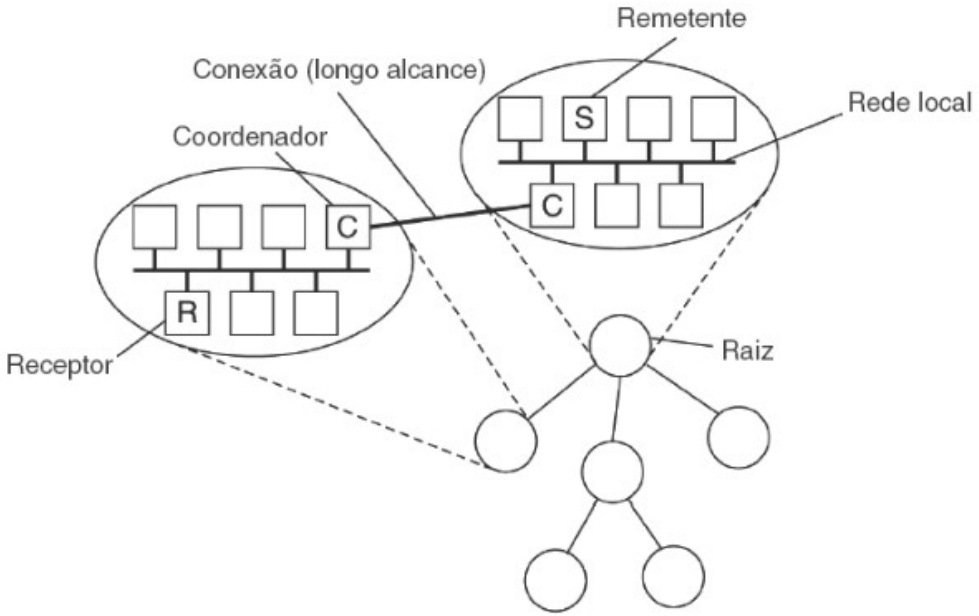
- Solução 3: Problemas com essa solução.
  - Garantir que somente uma requisição de retransmissão chegue ao remetente S (usar temporizadores?)
  - Interrupção dos processos para os quais a mensagem já foi entregue.
    - Separar os processos que não receberam m em um grupo multicast separado (gerenciamento de grupos?).
  - Reunir os processos em grupos que tendem a perder mensagens.

Soluções para Escalabilidade em Multicast Confiável

- Solução 4: Controle de Realimentação hierárquico.
  - Um único remetente.
  - Grupo é dividido em subgrupos, organizados em uma árvore.
  - Subgrupo com remetente: raíz da árvore.
  - Cada subgrupo possui um coordenador, que gerencia os pedidos de retransmissão.
  - O coordenador possui um buffer para armazenar as mensagens e atender pedidos dos membros do seu subgrupo.

Soluções para Escalabilidade
Multicast Confiável

- Essência do multicast confiável hierárquico:
  - Cada coordenador local repassa a mensagem a seus filhos e mais tarde manipula requisições de retransmissão.



Multicast Atômico

- Cenário 2: Os processos podem sofrer falhas.
- Objetivo: Uma mensagem será entregue a todos os processos ou a nenhum deles.
  - Mensagens são entregues na mesma ordem a todos os processos.

Multicast Atômico

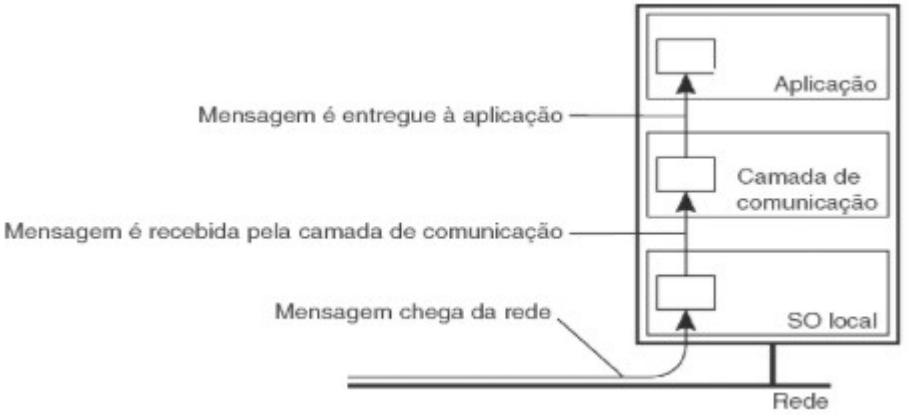
- Exemplo: Banco de Dados Replicado.
  - Banco de dados é construído como um grupo de processos, um processo para cada réplica.
  - Operações de atualizações são enviadas em multicast a todas as réplicas.
- Suponha que durante a execução de uma das atualizações de uma sequência, uma réplica caia.
  - Réplica se recupera: essencial que seja atualizada em relação às outras réplicas.

Multicast Atômico

- Exemplo: Banco de Dados Replicados.
- Se o sistema comporta multicast atômico, então
  - A operação de atualização que foi enviada a todas as réplicas um pouco antes de uma delas cair ou é executada em todas as réplicas não faltosas ou em nenhuma.
  - A atualização é realizada se as réplicas restantes concordarem que a réplica que caiu não pertence mais ao grupo.
  - Após a recuperação, a réplica é validada como sendo do grupo e recebe as atualizações.

Multicast Atômico – Sincronia Virtual

- Implementação
  - O sistema distribuído consiste em uma camada de comunicação que gerencia o recebimento das mensagens em um buffer local, até que possa ser entregue à aplicação.
- Organização lógica de um sistema distribuído para distinguir entre recebimento de mensagem e entrega de mensagem.



Multicast Atômico – Sincronia Virtual

- Implementação
  - Uma mensagem *m* está associada com uma lista de processos aos quais deve ser entregue.
  - Lista de entrega corresponde a uma visão do grupo.
  - Todos os processos possuem a mesma visão, concordando que m deve ser entregue a cada um deles e a nenhum processo.

Multicast Atômico – Sincronia Virtual

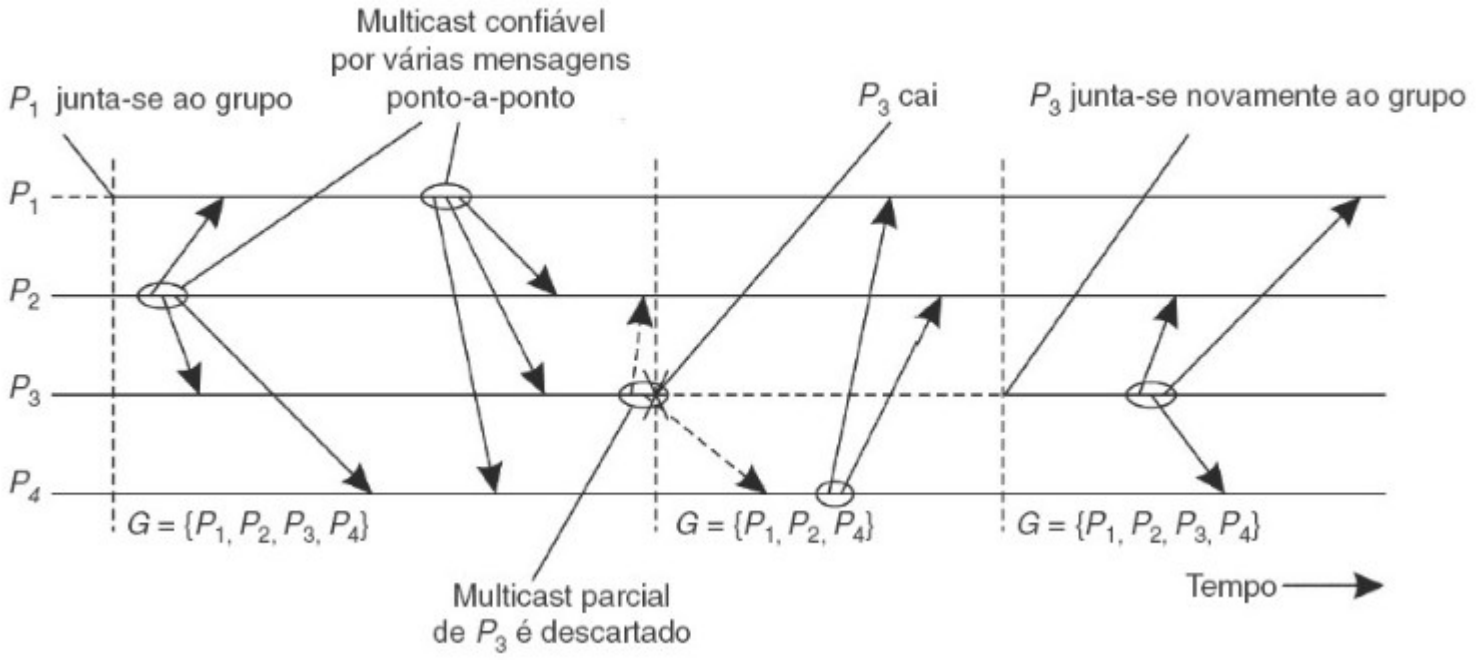
- Suponha que a mensagem m seja enviada em multicast no instante que seu remetente tem visão de grupo G.
- Se um processo entra ou sai do grupo, ocorre uma mudança de visão.
- Duas mensagens: m e a de entrada/saída do novo processo (vc).
- Garantir que todos os processos em G recebam m antes de vc, evitando inconsistência.

Multicast Atômico – Sincronia Virtual

- Multicast confiável garante que uma mensagem enviada em multicast para a visão do grupo G seja entregue a cada processo não faltoso em G.
- Se o remetente cair durante o multicast, a mensagem pode ou ser entregue a todos os processos restantes ou ser ignorada por cada um deles.
- Mensagens multicast ocorrem entre mudanças de visão.

Multicast Atômico – Sincronia Virtual

- Princípio de Multicast Síncrono Virtual



Multicast Atômico – Ordenação de Mensagens

- Tipos de Ordenação de Mensagens
  - Multicasts não ordenados;
  - Multicasts Ordenados em FIFO;
  - Multicasts Ordenados por Causalidade;
  - Multicasts Totalmente Ordenados;

Multicast Atômico – Ordenação de Mensagens

- Multicast Confiável não Ordenado
  - Não são dadas garantias quanto à ordem na qual as mensagens recebidas são entregues aos diferentes processos.
  - Três processos que se comunicam no mesmo grupo. A ordenação de eventos por processo é mostrada ao longo do eixo vertical.

Processo P_1	Processo P_2	Processo P_3
envia m_1	recebe m_1	recebe m_2
envia m_2	recebe m_2	recebe m_1

Multicast Atômico – Ordenação de Mensagens

- Multicast confiável ordenado em FIFO
- Camada de comunicação é forçada a entregar as mensagens que chegam do mesmo processo na mesma ordem em que elas foram enviadas.
- Exemplo: Quatro processos no mesmo grupo com dois remetentes diferentes e uma possível ordem de entrega de mensagens em multicast ordenado em FIFO.

Processo P_1	Processo P_2	Processo P_3	Processo P_4
envia m_1	recebe m_1	recebe m_3	envia m_3
envia m_2	recebe m_3	recebe m_1	envia m_4
	recebe m_2	recebe m_2	
	recebe m_4	recebe m_4	

Multicast Atômico – Ordenação de Mensagens

- Multicast confiável ordenado por causalidade.
  - Entrega de mensagens de modo que o potencial causalidade entre mensagens diferentes seja preservada.
  - Se m1 precede uma outra mensagem m2 por causalidade, independentemente de terem sido enviadas por processos diferentes, camada de aplicação sempre entregará m2 após ter recebido e entregado m1.
  - Utilização de relógios vetoriais.

Multicast Atômico – Ordenação de Mensagens

- Multicast confiável com entrega totalmente ordenada
  - Significa que as mensagens devem ser entregues na mesma ordem a todos os membros do grupo.
  - Entregas podem ser do tipo não ordenada, ordenada em FIFO ou ordenada por causalidade.

Multicast Atômico – Ordenação de Mensagens

- Seis versões diferentes de multicast confiável virtualmente síncrono.

Multicast	Ordenação básica de mensagens	Entrega totalmente ordenada?
Multicast confiável	Nenhuma	Não
Multicast Fifo	Entrega ordenada em Fifo	Não
Multicast por causalidade	Entrega ordenada por causalidade	Não
Multicast atômico	Nenhuma	Sim
Multicast atômico em Fifo	Entrega ordenada em Fifo	Sim
Multicast atômico por causalidade	Entrega ordenada por causalidade	Sim

Comprometimento Distribuído

- Envolve a realização de uma operação por cada membro de um grupo ou por absolutamente nenhum.
- No caso de multicast confiável, a operação é a entrega de uma mensagem.
- Presença de um coordenador que informa aos demais processos se devem ou não realizar a operação em questão.

Recuperação

- Quando ocorre uma falha no sistema é necessário levar o sistema para um estado livre de erro.

- Recuperação de erros é fundamental em tolerância a falhas.

Estratégias para Recuperação

- Recuperação retroativa
  - Retorna o sistema a algum estado que antes estava correto, continuando a execução após a recuperação.
  - Mecanismo de ponto de verificação (estado presente do sistema é registrado).
- Recuperação para Frente
  - Tentativa de levar o sistema para um próximo estado correto.

Estratégias para Recuperação

- Desvantagem da recuperação para a frente
  - É preciso saber de antemão quais erros podem ocorrer.
  - Tendo conhecimento de todos os erros e como levar o sistema para um estado correto, é possível recuperar totalmente o sistema.

Estratégias para Recuperação

- Desvantagem de recuperação retroativa
  - Pontos de validação podem ser caros para serem implementados;
  - Não existem garantias que o erro não acontecerá novamente;
  - Em alguns casos não é possível retroagir a um estado sem erros.
    - Exemplo: comando rm -rf *

Estratégias para Recuperação

- Exemplo: Comunicação confiável
  - Recuperação retroativa:
    - retransmissão de pacote;
  - Recuperação para a frente:
    - recuperação de pacotes a partir de outros pacotes;

Recuperação Retroativa e Registro de Mensagens

- Ponto principal: Combinar pontos de verificação com o registro da sequência de mensagens recebidas.

1) O processo receptor registra uma mensagem antes de entregá-la à aplicação ou então o emisso registra as mensagens antes de enviá-las.

2) Quando um processo cai, o sistema é restaurado para o estado correspondente ao ponto de verificação mais recente e, a partir deste ponto, reproduz as mensagens recebidas.

Recuperação Retroativa e Registro de Mensagens

- Qual a diferença entre utilizar apenas pontos de verificação e pontos de verificação + registros de mensagens?

1) No caso do uso isolado dos pontos de verificação, os processos são restaurados para o ponto antes da falha e o comportamento pode ser diferente após a recuperação. Exemplo: Mensagens podem ser entregues em ordem diferente.

2) No caso do registro de mensagens, o comportamento é reproduzido do mesmo modo entre o ponto de recuperação e o ponto em que ocorreu a falha.

Pontos de Verificação

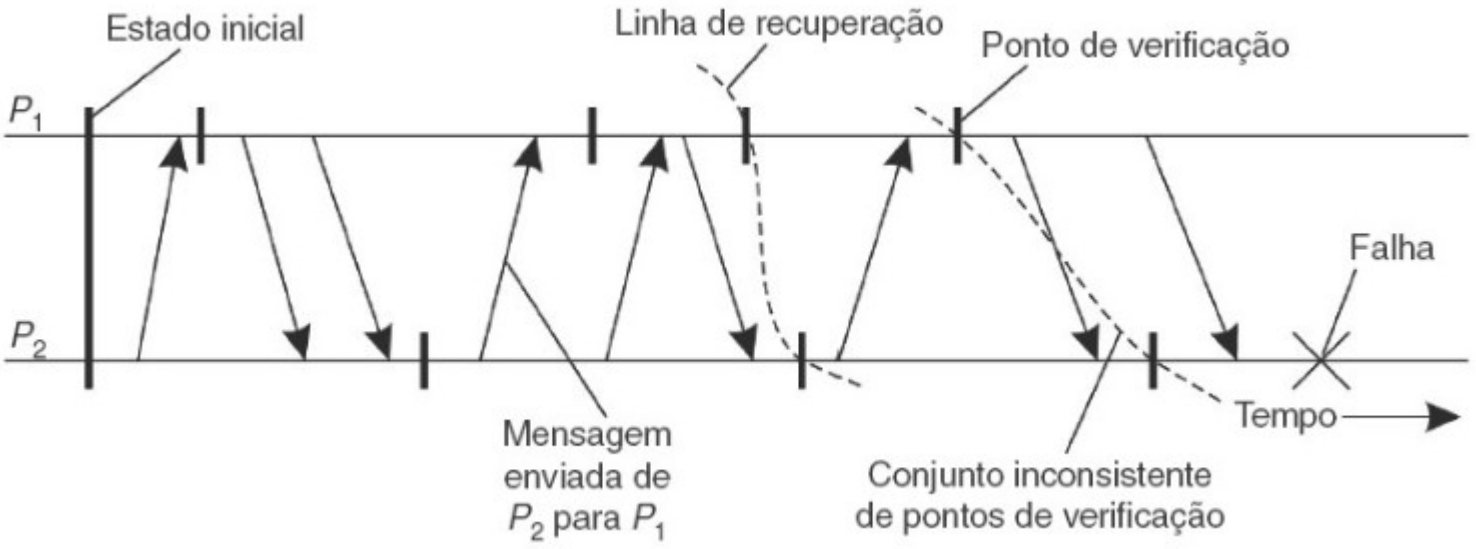
- A recuperação retroativa de erros requer que o sistema salve periodicamente o seu estado em armazenamento estável.
- Em uma fotografia distribuída, se um processo ***P*** tiver registrado o recebimento de uma mensagem, então também deve existir um processo ***Q*** que registrou o envio dessa mensagem.

Pontos de Verificação

- De um modo geral:
  - Cada processo salva seu estado periodicamente em um armazenamento estável disponível localmente.
  - A recuperação após uma falha de processo ou de sistema requer a construção de um estado global consistente com base nesses estados locais.
  - Melhor alternativa é recuperar a fotografia mais recente, denominada linha de recuperação.

Pontos de Verificação

- Exemplo de linha de recuperação:

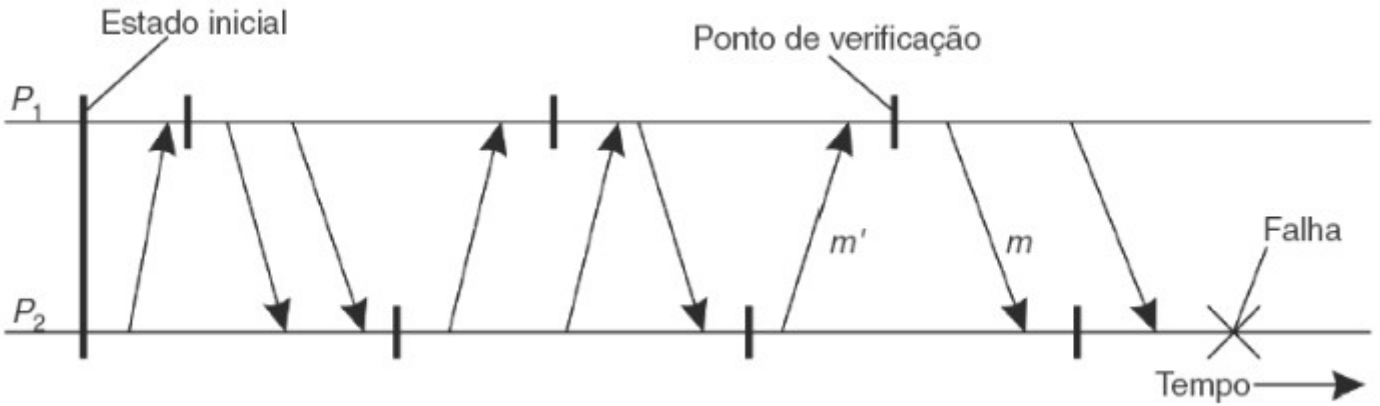


Pontos de Verificação Independentes

- Descobrir uma linha de recuperação requer que cada processo seja revertido a seu estado salvo mais recente.
- Se, em conjunto, os estados locais não formam uma fotografia distribuída, é preciso reverter ainda mais para trás.
- O processo de reversão em cascata pode resultar em efeito dominó.

Pontos de Verificação Independentes

- Efeito dominó:



Pontos de Verificação Coordenados

- Todos os processos sincronizam para escrever, em conjunto, seu estado para armazenamento local.
- O estado salvo é globalmente consistente, evitando as reversões em cascata que levam ao efeito dominó.
- Solução simples:
  - Usar protocolo de bloqueio de duas fases.

Protocolo de Bloqueio de Duas Fases

1) Coordenador envia uma mensagem multicast **checkpoint_request** a todos os processos.
2) Quando o processo recebe esta mensagem, estabelece um ponto de verificação local e enfileira qualquer mensagem e, envia uma mensagem de reconhecimento ao coordenador indicando que estabeleceu o ponto de verificação.
3) Após receber todos os ACKs, o coordenador envia mensagens multicast **checkpoint_done** para desbloquear os processos.

Pontos de Verificação Coordenados

- Estado globalmente consistente:
  - Não existirão mensagens que ultrapassem as linhas de recuperação, ou seja, as mensagens são tratadas entre os pontos de verificação.
- Qualquer mensagem que vier após uma requisição para estabelecer um ponto de verificação não é considerada como parte do ponto de verificação local. Ao mesmo tempo, mensagens que estão saindo são enfileiradas no local até a mensagem **checkpoint_done** ser recebida.

Resumo

- Tolerância a falha é uma questão importante no projeto de sistemas distribuídos.
- É uma característica pela qual um sistema pode mascarar a ocorrência e a recuperação de falhas.
- Há vários tipos:
  - Falha por queda;
  - Omissão;
  - Temporização;
  - Falha de resposta;
  - Arbitrárias.

Resumo

- Redundância é a técnica fundamental necessária para conseguir tolerância a falha.
- No caso de processos, tem-se a formação de grupos.
  - Grupos podem ser simples (todos os processos tomam decisões da mesma maneira) ou hierárquicos (coordenador geral).
- Se os processos falharem, os mesmos deverão chegar a um acordo.
- O algoritmo de Lamport usa a decisão da maioria.

Resumo

- No caso da comunicação confiável multicast, deve-se garantir que todas as mensagens cheguem aos membros do grupo.
- Se não existirem falhas nos processos, e os grupos forem pequenos, pode-se usar os mecanismos de confirmação. Por questões de escalabilidade, pode-se usar NACKs e temporizadores.
- No caso de fornecer multicast atômico, onde todos os membros do grupo devem receber a informação na mesma ordem. A entrega das informações é baseada em visões de grupo.

Resumo

- Recuperação em sistemas tolerantes a falhas é alcançada por pontos de verificação periódicos do estado do sistema.
- A verificação é completamente distribuída, sendo esta uma operação cara, principalmente no caso de pontos de verificação independentes.
- Para melhorar o desempenho, muitos sistemas distribuídos combinam pontos de verificação com registro de mensagens.
- Registrado a comunicação entre processos, torna-se possível reproduzir a execução do sistema após a ocorrência de uma queda.