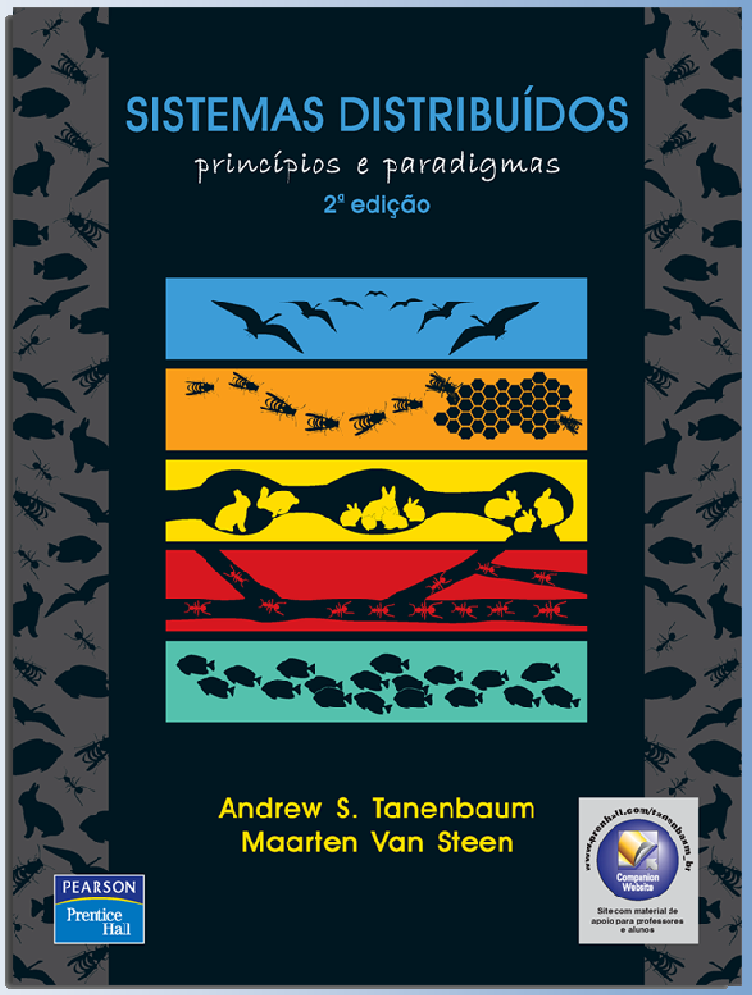
Tolerância a falha



capítulo 8

Tolerância a Falha

1. Introdução à tolerância a falha
2. Resiliência **de** processo
3. Comunicação confiável cliente-servidor
4. Comunicação confiável de grupo
5. Comprometimento distribuído
6. Recuperação

Introdução à tolerância a falhas

- Sistemas de uma máquina (não-distribuídos) – uma falha é quase sempre total;
- Sistemas distribuídos – pode ocorrer uma falha parcial, quando um componente do sistema falha.
  - Objetivo: recuperar automaticamente de falhas parciais sem afetar seriamente o desempenho global.

Introdução à tolerância a falha
Conceitos Básicos

- Tolerância a falhas está fortemente relacionada a sistemas confiáveis. Confiabilidade é um termo que abrange uma série de requisitos:
  - **Disponibilidade** – O sistema está pronto para ser usado imediatamente?
  - **Confiabilidade** – O sistema funciona continuamente sem falhas?
  - **Segurança** – Se o sistema deixar de funcionar corretamente por um certo tempo acontecerá algo "catastrófico"?
  - **Capacidade de Manutenção** – Um sistema que falhou pode ser consertado facilmente?

Introdução à tolerância a falha
Conceitos Básicos

- **Defeito** – se um sistema não pode cumprir suas promessas, apresenta defeito. Ex.: Não consegue garantir as consistências prometidas.
- **Erro** – parte do estado de um sistema que pode levar a uma falha. Ex.: Pacotes danificados.
- **Falha** – é a causa de um erro. Ex.: Um meio de transmissão errado ou ruim pode danificar pacotes.

Introdução à tolerância a falha
Conceitos Básicos

- **Tolerância a falhas**: um sistema pode prover seus serviços mesmo na presença de falhas.
- Tipos de falhas:
  - **Falha transiente** – ocorre uma vez e desaparece.
  - **Falha intermitente** – ocorre, para por um período indeterminado, reaparece, e assim por diante.
  - **Falha permanente** – continua a existir até que o componente faltoso seja substituído.

Introdução à tolerância a falha
Modelos de Falha

- Para melhor identificar as falhas, foram desenvolvidos diversos esquemas de classificação, entre eles, o quadro abaixo:

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

Introdução à tolerância a falha
Mascaramento de falha por redundância

- Para o sistema ser tolerante a falhas, as ocorrências destas devem ser ocultas de outros processos e dos usuários.
- A técnica fundamental para mascarar falhas é usar redundância:
  - **Redundância de informação** – bits extras para recuperação de pacotes (Hamming);
  - **Redundância de tempo** – executar novamente uma ação, se for preciso.
  - **Redundância física** – adicionar equipamentos ou processos extras para possibilitar tolerância a perda ou mal funcionamento de alguns componentes.

Introdução à tolerância a falha
Mascaramento de falha por redundância

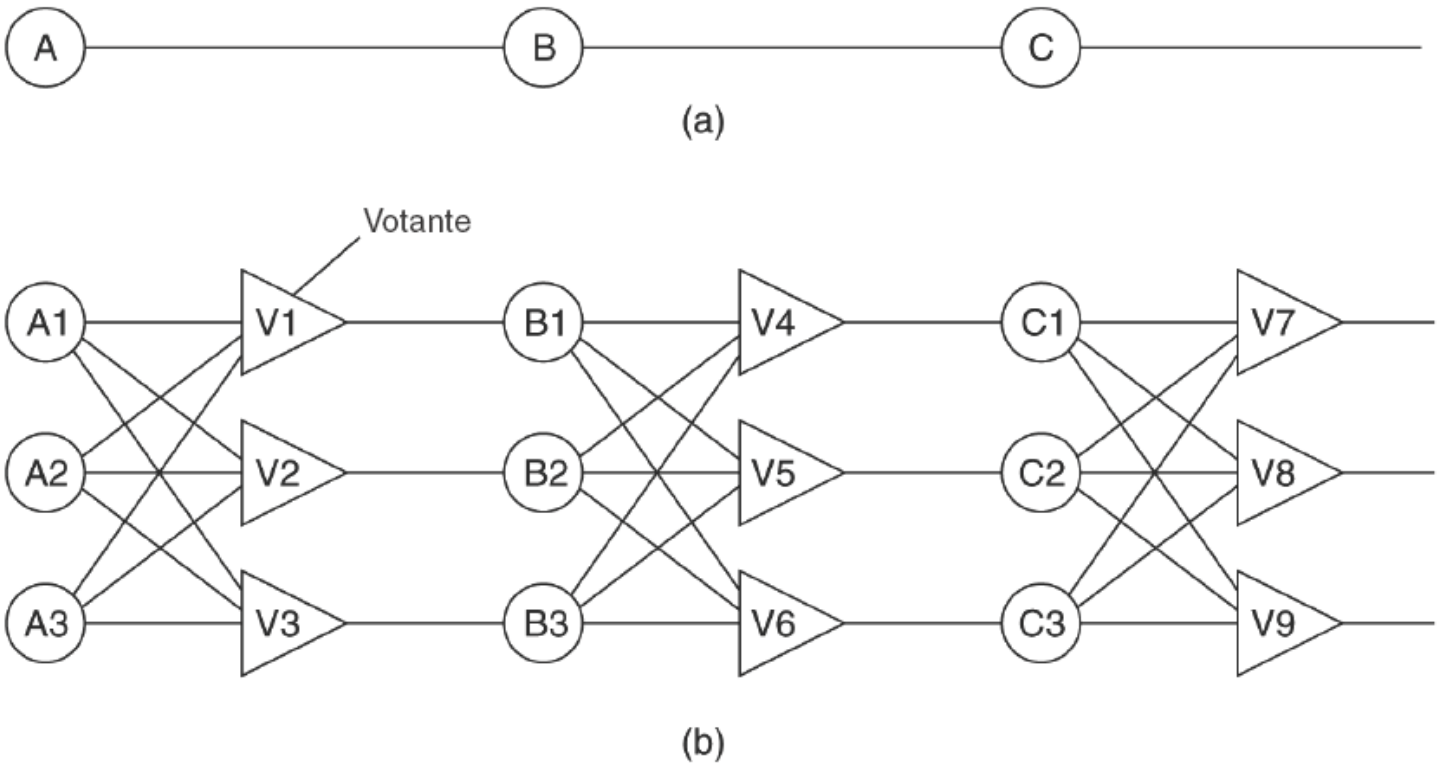


Figura 8.1 Redundância modular tripla.

Tolerância a Falha

1. Introdução à tolerância a falha
2. Resiliência **de** processo
3. Comunicação confiável cliente-servidor
4. Comunicação confiável de grupo
5. Comprometimento distribuído
6. Recuperação

Resiliência de Processo

- **Resiliência** – elasticidade, capacidade de recuperação.
- Para conseguir realmente tolerância a falha em sistemas distribuídos, o primeiro enfoque é dado na proteção contra falhas dos processos, conseguida com replicação de processos em grupos.

Resiliência de Processo
Questões de Projeto de Grupos

- A abordagem fundamental para tolerar um processo faltoso é organizar vários processos idênticos em um grupo.
  - Quando uma mensagem é enviada a um grupo, todos membros do grupo a recebem;
  - Se um processo falhar, espera-se que algum outro se encarregue da mensagem em seu lugar.
  - Grupos podem ser dinâmicos;
  - A finalidade de introduzir grupos é permitir que processos tratem conjuntos de processos como uma única abstração.

Resiliência de Processo
Grupos simples vs. Grupos hierárquicos

- Grupos simples: Todos processos são iguais dentro de um grupo; Ninguém manda, todas decisões são coletivas.
- Grupos hierárquicos: Possui coordenador(es) e operários.

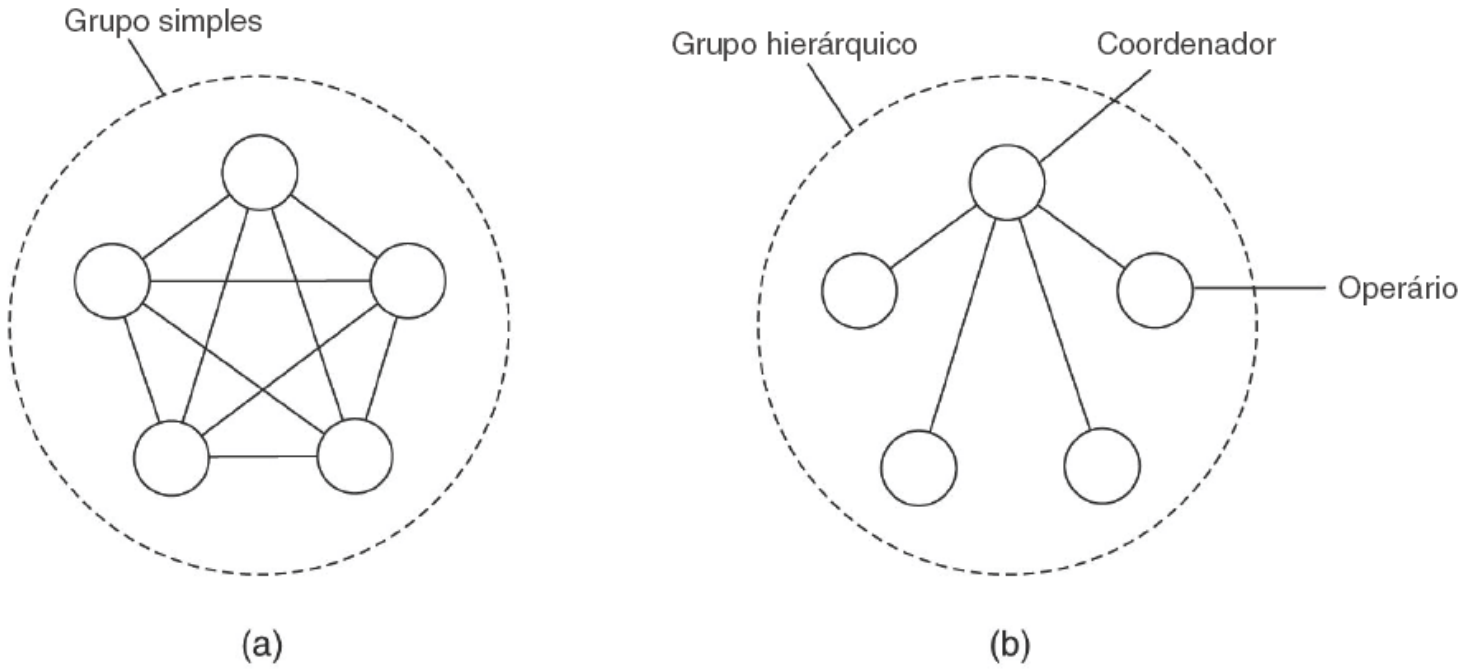


Figura 8.2 (a) Comunicação em um grupo simples. (b) Comunicação em um grupo hierárquico simples.

Resiliência de Processo
Associação a um grupo

- Como criar e eliminar grupos, assim como permitir a entrada e saída de processos em um grupo?
  - **Servidor de grupo** recebe todas as requisições e mantém banco de dados completo sobre todos grupos e seus membros;
  - Gerenciar associação de grupo de modo distribuído
- O que fazer se várias máquinas caírem, provocando a parada de funcionamento?
  - Protocolo para reconstruir grupo; Este deve ser capaz de controlar a nova criação do grupo e resistir a vários processos tentando recriá-lo ao mesmo tempo.

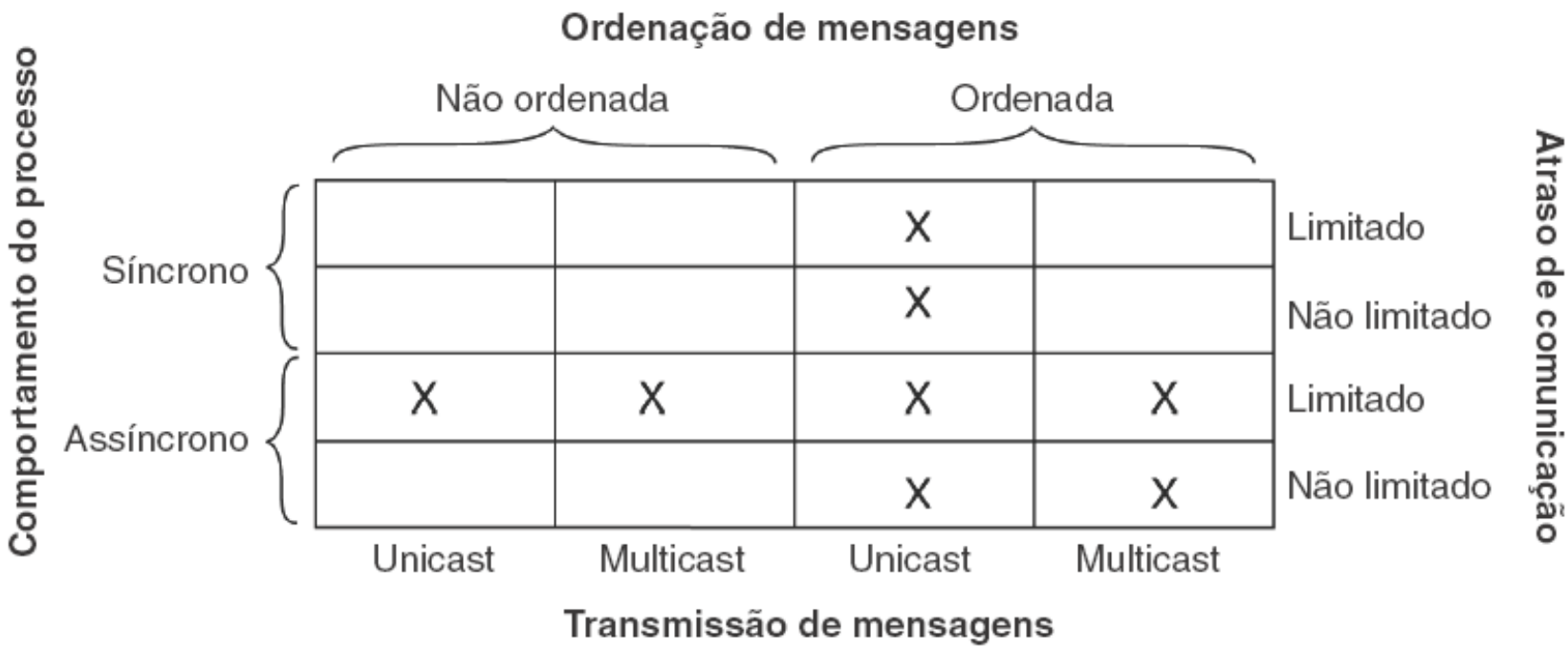
Resiliência de Processo
Mascaramento de falha e replicação

- Para construir sistemas tolerantes a falhas, podemos replicar processos e organizá-los em um grupo para substituir um único processo (vulnerável) por um grupo (tolerante a falha). Feito de duas maneiras:
  - Protocolos baseados em primários: Um grupo de processos é organizado de modo **hierárquico** no qual um servidor primário coordena todas operações de escrita. Os servidores de backup executam um algoritmo de eleição caso o primário caia.
  - Protocolos de escrita replicada: conjunto idêntico em um grupo simples.

Resiliência de Processo
Acordo em sistemas com falha

- Organizar processos replicados em um grupo ajuda a aumentar a tolerância a falha. Existem várias situações:
  - Sistemas síncronos vs. assíncronos;
  - Atraso de comunicação limitado ou não;
  - Entrega de mensagens ordenada ou não;
  - Transmissão de mensagens unicast ou multicast

Resiliência de Processo
Acordo em sistemas com falha

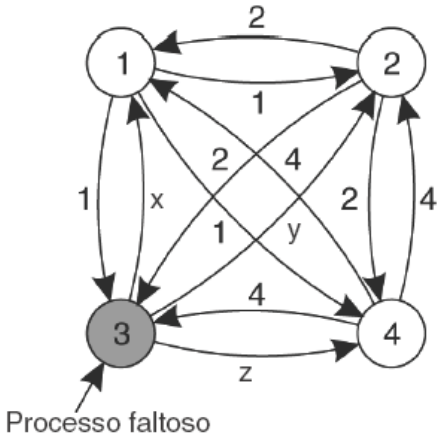


Comportamento do processo	Ordenação de mensagens: Não ordenada – Unicast	Não ordenada – Multicast	Ordenada – Unicast	Ordenada – Multicast	Atraso de comunicação
Síncrono			X		Limitado
Síncrono			X		Não limitado
Assíncrono	X	X	X	X	Limitado
Assíncrono			X	X	Não limitado

Figura 8.3 Circunstâncias sob as quais se pode chegar a um acordo distribuído.

Resiliência de Processo
Acordo em sistemas com falha

- Esse problema, estudado pela primeira vez por Lamport (1982), é também conhecido como **problema do acordo bizantino**, consideramos problemas assíncronos, mensagens unicast, ordenação preservada e atraso limitado.



(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)

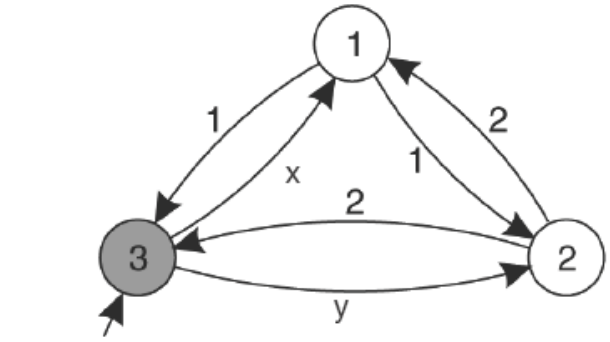
1 Obteve	2 Obteve	4 Obteve
(1, 2, *y*, 4)	(1, 2, *x*, 4)	(1, 2, *x*, 4)
(*a*, *b*, *c*, *d*)	(*e*, *f*, *g*, *h*)	(1, 2, *y*, 4)
(1, 2, *z*, 4)	(1, 2, *z*, 4)	(*i*, *j*, *k*, *l*)

(c)

Figura 8.4 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.

Resiliência de Processo
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 (ou seja, mínimo mais que 2/3).



(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)

Figura 8.5 Igual à Figura 8.4, exceto que, agora, há dois processos corretos e um processo faltoso.

Resiliência de Processo
Detecção de falha

- Detecção de falhas pode ser resumida da seguinte forma: Dado um grupo de processos, membros não faltosos devem ser capazes de decidir quem ainda é um membro e quem não é, ou seja: detectar quando um membro falhou.
  - Ativamente, através de **pings**;
    - Em redes não confiáveis, declarar que um processo falhou porque não retornou uma resposta a uma mensagem ping pode ser incorreto;
    - Esgotamento de temporização é tosco (uma mensagem perdida é pouco para garantir falha).
  - Passivamente, através de recebimento de atividades via *gossip*.

Resiliência de Processo
Detecção de falha

- Distinção de falha de rede ou falha de nó é uma situação ideal:
  - Implementável estabelecendo que apenas um nó identificando a falha é insuficiente. Se apenas um nó a detecta, é uma falha de rede, sendo que se todos detectam é que realmente o nó caiu.

Tolerância a Falha

1. Introdução à tolerância a falha
2. Resiliência **de** processo
3. Comunicação confiável cliente-servidor
4. Comunicação confiável de grupo
5. Comprometimento distribuído
6. Recuperação

Comunicação Confiável Cliente-Servidor
Comunicação ponto-a-ponto

- Um canal de comunicação pode apresentar falhas por queda, por omissão, de temporização ou arbitrárias.
  - O foco da construção de canais de comunicação confiável é em ocultar as falhas por **queda** e **omissão**.
- Em sistemas distribuídos, a comunicação confiável ponto-a-ponto é estabelecida pelo uso de um protocolo de transporte confiável, como TCP, que mascara falhas por omissão, mas não mascara falhas por queda.
  - Mascarável permitindo o SD reestabelecer a conexão através do reenvio de um pedido de conexão.
  - Espera-se que o outro lado ainda, ou novamente, será capaz de responder.
- A forma mais adotada para adquirir esta transparência é através do uso de RPC.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Chamadas de procedimento remoto (RPC) têm como objetivo ocultar comunicação, fazendo com que chamadas de procedimentos remotos pareçam exatamente como locais.
  - Se não existem erros, o RPC desempenha bem seu papel. No entanto, os erros nem sempre são fáceis de se mascarar.
- São distintas 5 falhas a serem mascaradas em RPC:
  1. O 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.
  - Cada uma dessas apresenta problemas diferentes e requer soluções diferentes.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Cliente não pode localizar servidor:
  - O cliente pode não localizar um servidor adequado, por exemplo, na atualização de apêndices, o apêndice de cliente não é atualizado.

  ☺O erro pode ativar uma exceção para permitir seu tratamento e emitir uma resposta;

  ☹Nem todas linguagens suportam exceções;

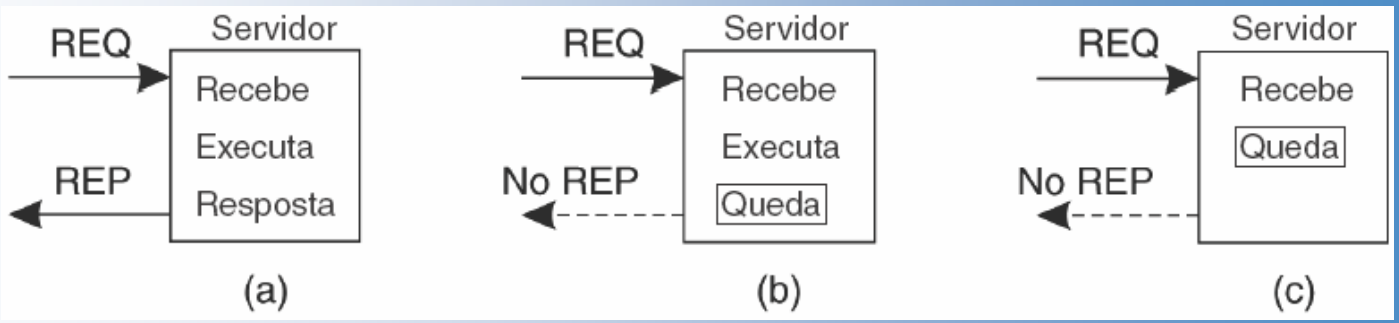
  ☹A transparência é destruída: O erro "não foi possível encontrar servidor" é incomum em sistemas monoprocessador.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Mensagens de requisição perdidas:
  - A mensagem não é entregue ou a resposta não chega.
  - ☺ É o erro mais fácil de se tratar: basta fazer um temporizador ao enviar a requisição. Se o temporizador expirar antes do recebimento da resposta de reconhecimento, a mensagem é reenviada.
  - ☹ Se um certo número de mensagens de requisição for perdido, o cliente pode chegar à falsa conclusão de que o servidor está fora do ar;
  - ☹ Novamente a transparência é destruída: O erro “não foi possível encontrar servidor”.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Quedas de Servidor – falhas como em (b) e (c):



☺ (b) corrigido informando a falha ao cliente (exceção)

☺ (c) corrigido por retransmissão da requisição

☹ O temporizador do cliente não possui discernimento entre os 2 erros.

1. Semântica ao menos uma vez;
2. Semântica no máximo uma vez;
3. Não garantir nada;

☹ Infelizmente não é possível fazer uma semântica exatamente uma vez, que seria a desejada.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Quedas de Servidor – situações possíveis na ocorrência de uma queda do servidor.
  Ex.: O cliente tenta imprimir um documento

Cliente	Servidor					
	Estratégia M→P			Estratégia P→M		
Estratégia de reemissão	*MPC*	*MC(P)*	*C(MP)*	*PMC*	*PC(M)*	*C(PM)*
Sempre	DUP	OK	OK	DUP	DUP	OK
Nunca	OK	ZERO	ZERO	OK	OK	ZERO
Somente quando recebe ACK	DUP	OK	ZERO	DUP	OK	ZERO
Somente quando não recebe ACK	OK	ZERO	OK	OK	DUP	OK

OK = Texto é impresso uma vez
DUP = Texto é impresso duas vezes
ZERO = Texto não é impresso

Figura 8.7 Diferentes combinações de estratégias de cliente e servidor na presença de quedas de servidor.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Mensagens de resposta perdidas:
  - A mensagem é entregue, mas a resposta não.
  - ☺ É feito como nas mensagens de requisição perdidas: é feito um temporizador ajustado pelo SO do cliente. Se o temporizador expirar antes do recebimento da resposta de reconhecimento, a mensagem é reenviada.
  - ☹ O que torna mais difícil é que o cliente não sabe com certeza o motivo da ausência de resposta: a requisição ou a resposta se perderam, ou é lentidão do servidor?
  - ☹ Pedidos **idempotentes** podem ser executados mais de uma vez sem problemas, mas nem todos o são;
    - ☺ Deve-se estruturar todas as requisições de modo idempotente, fazendo o cliente designar o número de seqüência a cada requisição. Um mesmo número indica um mesmo pedido, e não um novo, e a retransmissão será ignorada.
    - ☺ Outra opção é um bit no cabeçalho para distinguir mensagens iniciais de retransmissões.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Quedas de cliente:
  - O que acontece se um cliente enviar uma requisição a um servidor para executar algum trabalho e cair antes de o servidor responder? Identificar o processo como **órfão**.
  - ☹ Processos órfãos podem interferir com a operação normal do sistema. O mínimo seria desperdício de CPU, mas também pode travar arquivos e amarrar recursos valiosos. Além disso, se o RPC cliente for reiniciado e executar novamente a RPC, mas a resposta voltar logo em seguida, pode haver confusão.

Comunicação Confiável Cliente-Servidor
Semântica da RPC na presença de falhas

- Quedas de cliente:
  - ☺ O que fazer com órfãos?
    1. Antes de enviar uma mensagem RPC, um apêndice de cliente faz uma entrada de registro informando o que está prestes a fazer, mantendo o registro em disco ou outro meio que sobreviva a quedas. Ao voltar, verifica-se o registro e é feito o **extermínio de órfão**.
       - ☹ Vários acessos a disco deterioram o desempenho.
       - ☹ Órfãos podem gerar novas RPCs, gerando netos órfãos, etc.
    2. Através da **reencarnação**, o tempo é dividido em épocas seqüenciais numeradas em seqüência. Ao ser reiniciado, o cliente envia broadcast declarando início de nova época. TODAS as computações remotas em nome dele são removidas, e caso algum órfão sobreviva, pela época é facilmente detectado.
    3. A **reencarnação gentil** apenas remove as computações se não puder localizar os seus proprietários.
    4. Por **expiração**, é dado um tempo padrão T para um trabalho ser feito. Caso não dê tempo, deve solicitar novo quantum. Caso não encontre o cliente, não há a quem solicitar o quantum, e o processo é removido.
  - ☹ Todos estes métodos são grosseiros e podem gerar várias situações indesejadas como: se um órfão é exterminado e o mesmo mantinha travas em um ou mais arquivos ou registros de bancos de dados, estas podem permanecer para sempre.
  - ☹ Isso é pior ainda se o RPC ainda utiliza filas remotas para iniciar outros processos futuramente, o que significa que traços dos órfãos permanecerão intactos.

Tolerância a Falha

1. Introdução à tolerância a falha
2. Resiliência **de** processo
3. Comunicação confiável cliente-servidor
4. Comunicação confiável de grupo
5. Comprometimento distribuído
6. Recuperação

Comunicação Confiável de Grupo
Esquemas básicos de multicast confiável

- Tem como objetivo garantir entrega a todos os membros em um grupo de processos.
  - Multicast através de unicasts confiáveis requer que cada processo estabeleça conexão ponto-a-ponto com os que se quer trabalhar, e exige n*(n-1)/2 conexões e n-1 mensagens iguais trafegando na rede, sendo inviável para grande número de processos!
- Sendo multicast confiável enviar uma mensagem a todos os membros de um grupo, deve-se definir:
  - Um processo que acabou de entrar em um grupo deve receber uma mensagem enviada antes de sua entrada?
  - E um processo que acabou de sair de um grupo deve receber uma mensagem enviada antes de sua saída?
- Distinguir entre comunicação confiável que:
  - Considera presença de processos faltosos;
  - Considera que processos funcionam corretamente.

Comunicação Confiável de Grupo
Esquemas básicos de multicast confiável

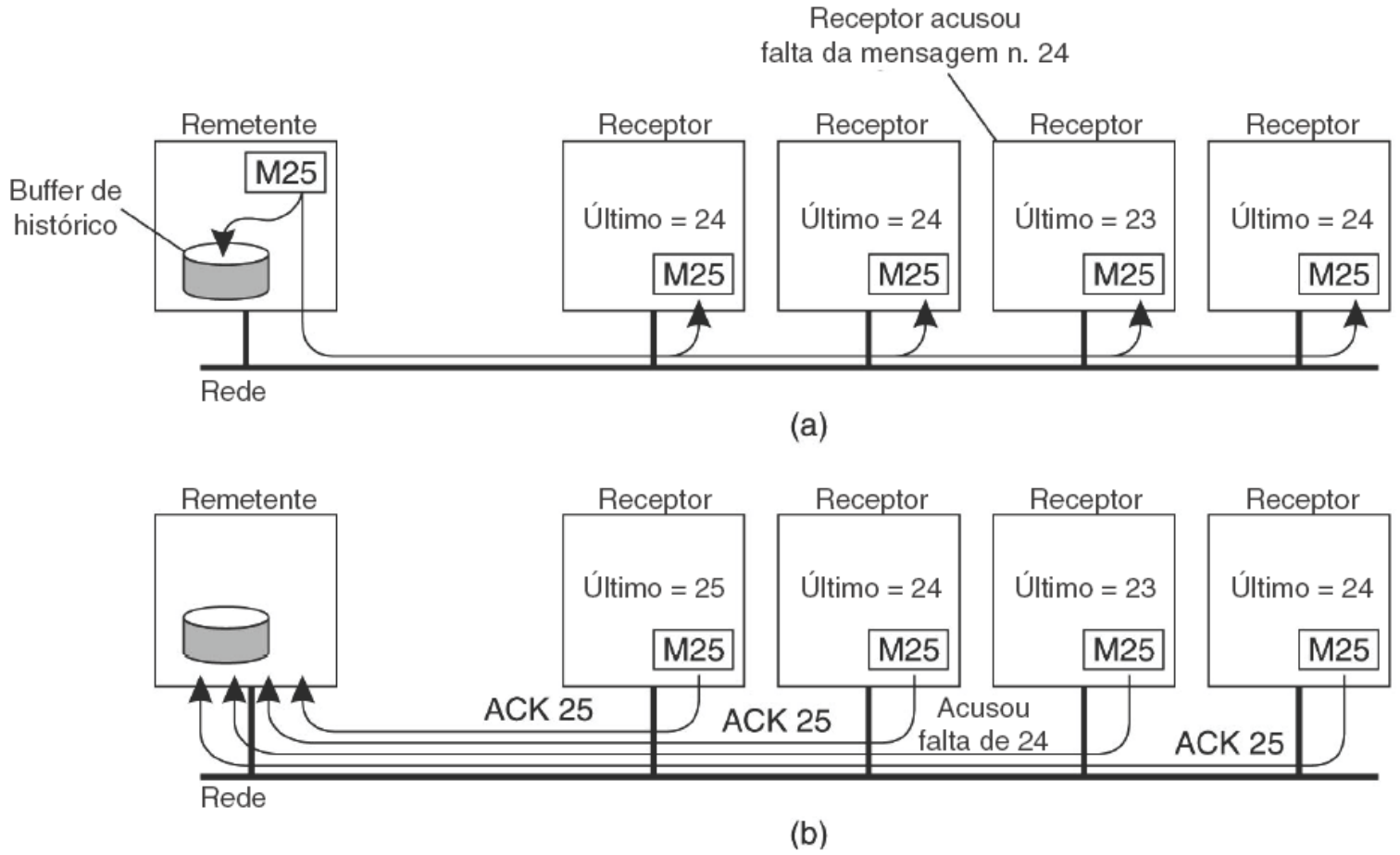


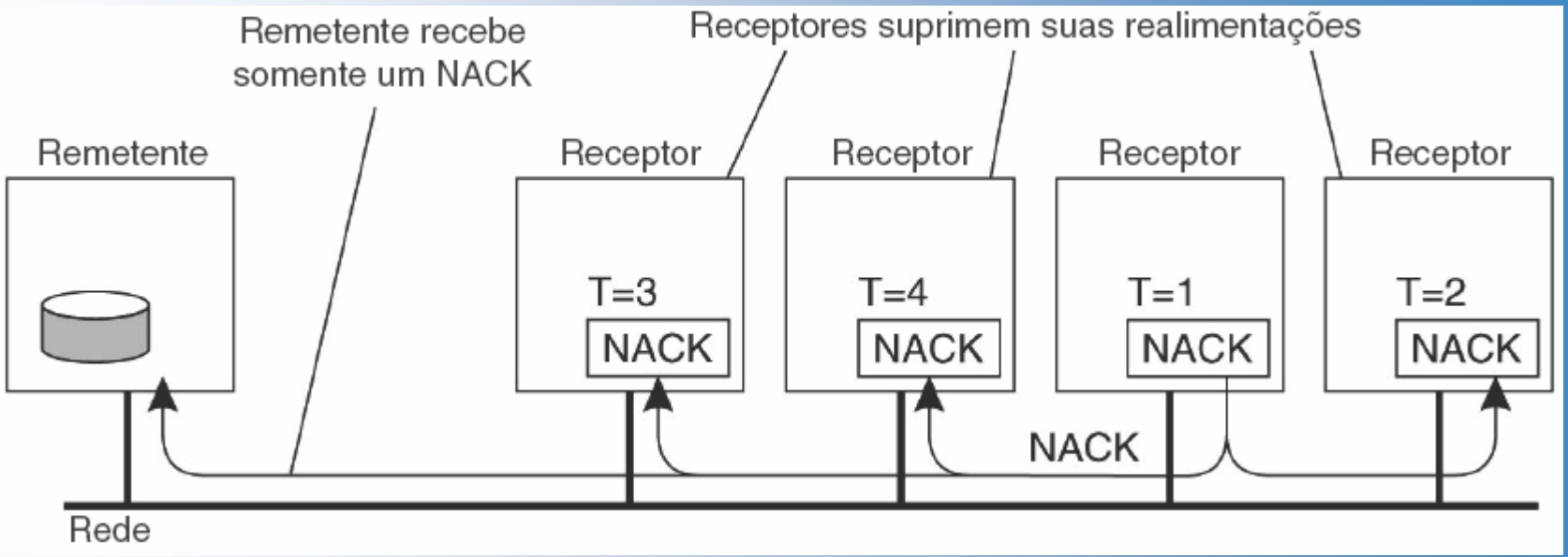
Figura 8.8 Uma solução simples para multicast confiável quando todos os receptores são conhecidos; a premissa é que nenhum falhe. (a) Transmissão de mensagem. (b) Realimentação de relatório.

Comunicação Confiável de Grupo
Escalabilidade em multicast confiável

- Um problema apresentado é a **implosão de retorno**, que pode lotar o remetente com mensagens de confirmação de recebimento.
  - Solução: Não confirmar recebimento, mas apenas devolver respostas quando detectar faltas.
    - Gera novo problema: Quando remover mensagem enviada do buffer?
      - Quando fica cheio, mas pode resultar em uma retransmissão não ser honrada!
    - Implosões de retorno ainda podem acontecer!

CCG - Escalabilidade em multicast confiável
Controle de realimentação não hierárquico

- Uso da supressão de retorno (protocolo SRM), que envia negativas em multicast após tempo aleatório.
  - Caso outro receptor recebe NACK, suprime seu envio de NACK.
  - Para aumentar escalabilidade, pode-se implementar:
    - novo grupo a processos que perdem mesmos pacotes;
    - um receptor ajudar na realimentação local.



CCG - Escalabilidade em multicast confiável
Controle de realimentação hierárquico

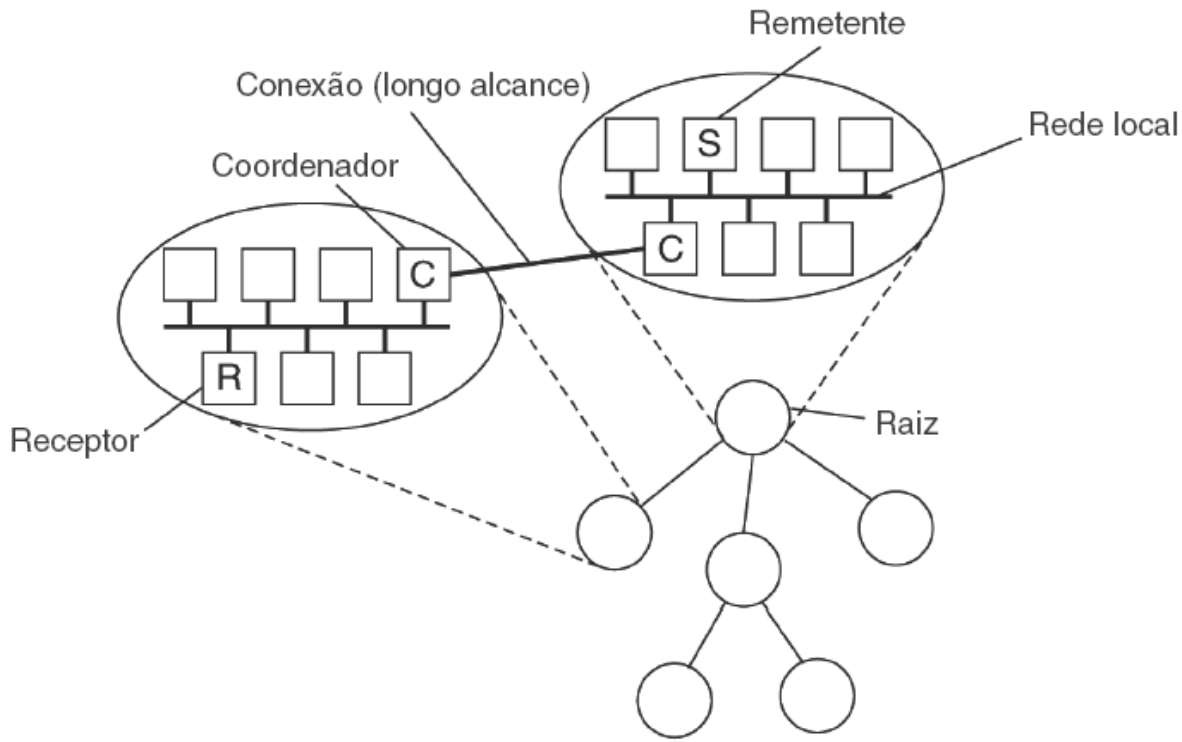


Figura 8.10 Essência do multicast confiável hierárquico. Cada coordenador local repassa a mensagem a seus filhos e mais tarde manipula requisições de retransmissão.

Comunicação Confiável de Grupo
Multicast atômico

- Para multicast confiável na presença de falhas, deseja-se que a mensagem seja entregue a todos os processos ou nenhum deles, sendo, ainda, entregue na mesma ordem em todos os processos.
  - Útil para garantir consistência em réplicas de bancos de dados baseados em protocolos de replicação ativa.
    - A operação do multicast atômico pode ser realizada se as réplicas concordarem que uma réplica que caiu não pertence mais ao grupo.
    - Uma réplica que se recupera é forçada a se juntar ao grupo mais uma vez.
      - Nenhuma atualização é enviada enquanto não se registra efetivamente ao grupo;
      - O registro só é aceito se a réplica se atualizar com o grupo.

Comunicação Confiável de Grupo
Multicast atômico – Sincronia virtual

- Uma mensagem recebida é colocada em buffer local da camada de comunicação até que ela possa ser entregue à aplicação.
- É criada uma **visão de grupo**, e mensagens só podem ser entregues se todos processos mantêm a mesma visão de grupo.

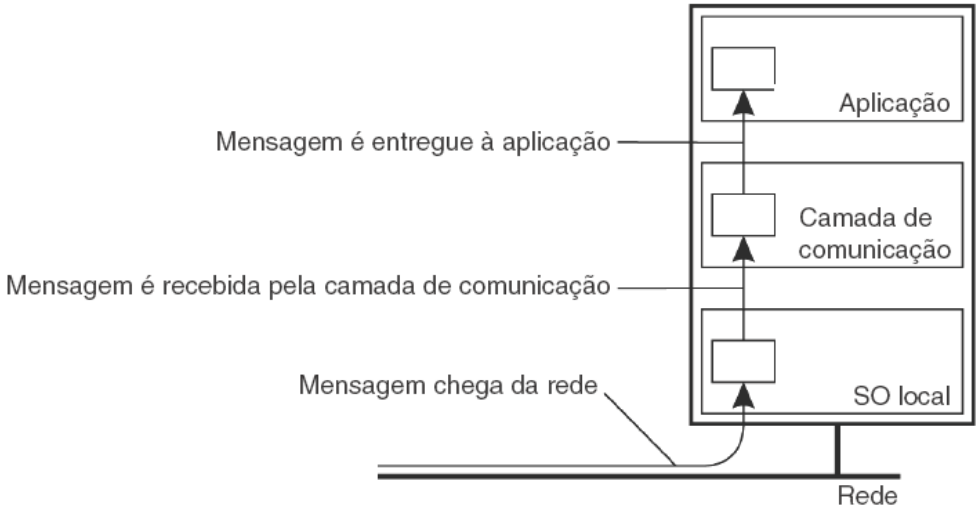
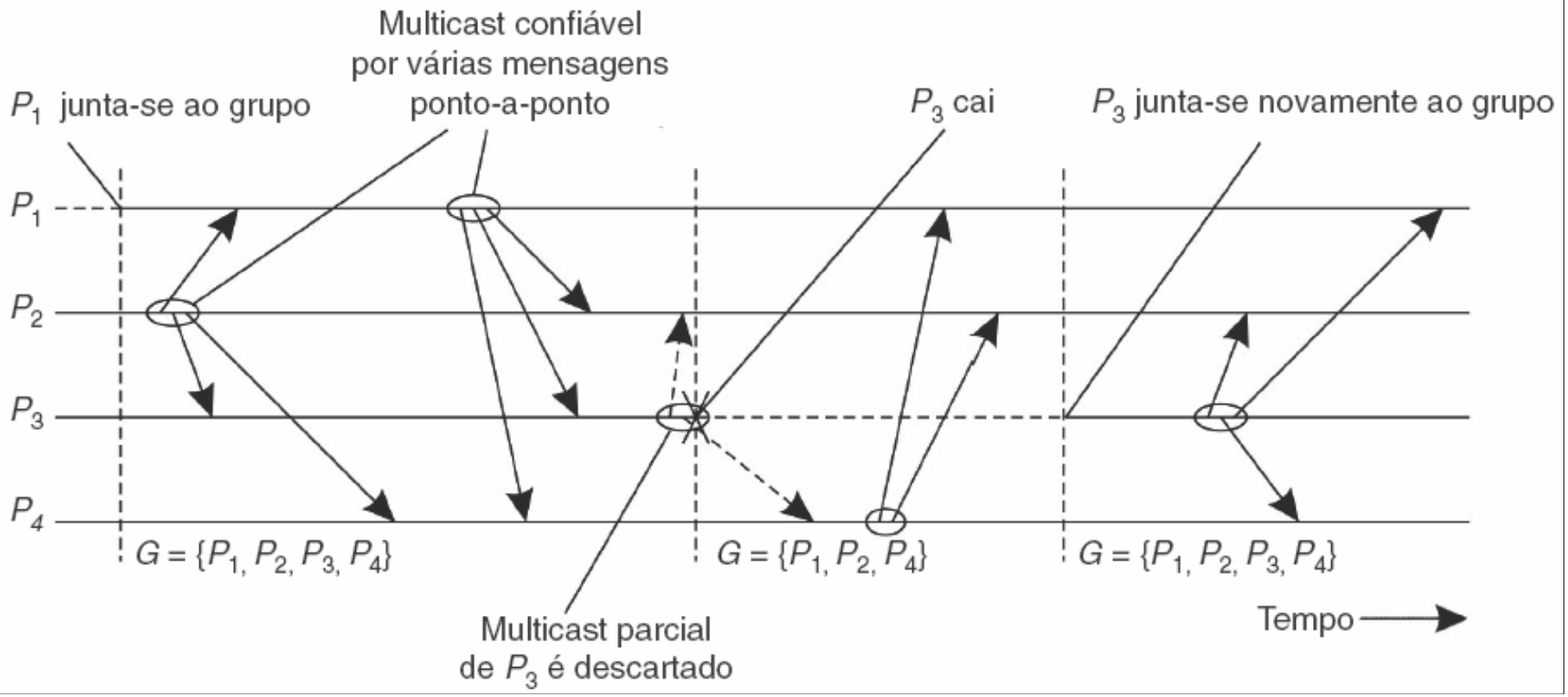


Figura 8.11 Organização lógica de um sistema distribuído para distinguir entre recebimento de mensagem e entrega de mensagem.

Comunicação Confiável de Grupo
Multicast atômico – Sincronia virtual

- Uma **mudança de visão** ocorre quando um processo cai ou quando entra no grupo um novo processo.
- A mudança de visão implica no cancelamento de mensagens que ainda se encontram em trânsito.
- Se o remetente cair durante o multicast, a mensagem pode ser entregue a todos os processos restantes ou ser ignorada por cada um deles (multicast **virtualmente síncrono**).



Comunicação Confiável de Grupo
Ordenação de Mensagens

- São distinguidas quatro ordenações diferentes:
  - Multicasts não ordenados;

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

  - Multicasts ordenados 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	

  - Multicasts ordenados por causalidade
  - Multicasts totalmente ordenados*

Comunicação Confiável de Grupo
Ordenação de Mensagens

- A entrega totalmente ordenada é aplicada em adição às três primeiras, e significa que, independente da ordem aplicada, a ordem de entrega deve ser a mesma em todos os membros do grupo.
  - Multicasts que apresentam essa propriedade são denominados **multicasts atômicos**.

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

Tabela 8.2 Seis versões diferentes de multicast confiável virtualmente síncrono.

Tolerância a Falha

1. Introdução à tolerância a falha
2. Resiliência **de** processo
3. Comunicação confiável cliente-servidor
4. Comunicação confiável de grupo
5. Comprometimento distribuído
6. Recuperação

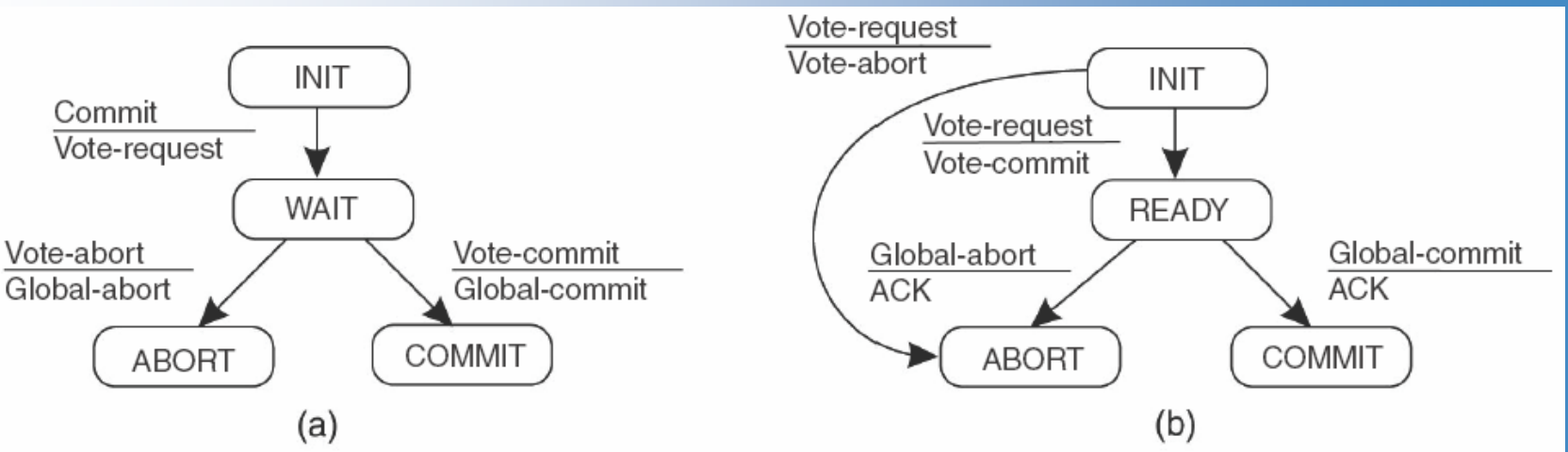
Comprometimento Distribuído

- É um problema que envolve a realização de uma operação por cada membro de um grupo de processos ou por absolutamente nenhum.
- Costuma ser estabelecido por um coordenador.
  1. Uma fase;
  2. Duas fases;
  3. Três fases;

Comprometimento Distribuído
Comprometimento de duas fases

1. Coordenador envia VOTE_REQUEST a todos;
2. Participante responde com VOTE_COMMIT se preparado para comprometer localmente, ou VOTE_ABORT se não estiver preparado;
3. Coordenador envia multicast GLOBAL_COMMIT se todos concordarem, ou GLOBAL_ABORT se um ou mais recusarem;
4. Cada participante espera: caso receba GLOBAL_COMMIT, compromete localmente a transação, caso receba GLOBAL_ABORT, aborta localmente a transação.

Comprometimento Distribuído
Comprometimento de duas fases



- Em caso de falha do coordenador, os participantes podem tomar decisões baseadas no estado dos demais participantes:

Estado de *Q*	Ação por *P*
COMMIT	Fazer transição para COMMIT
ABORT	Fazer transição para ABORT
INIT	Fazer transição para ABORT
READY	Contatar um outro participante

Comprometimento Distribuído
Comprometimento de três fases

- Há (raras) situações em que os participantes continuam bloqueados até que o coordenador se recupere. Uma das soluções é o uso do Comprometimento de três fases, que se baseia no de duas fases, porém adiciona que os estados devem satisfazer a duas regras:
  1. Não há nenhum estado único a partir do qual seja possível fazer uma transição diretamente para um estado COMMIT ou para um estado ABORT.
  2. Não há nenhum estado no qual não seja possível tomar uma decisão final e a partir do qual possa ser feita uma transição para um estado COMMIT.

Comprometimento Distribuído
Comprometimento de três fases

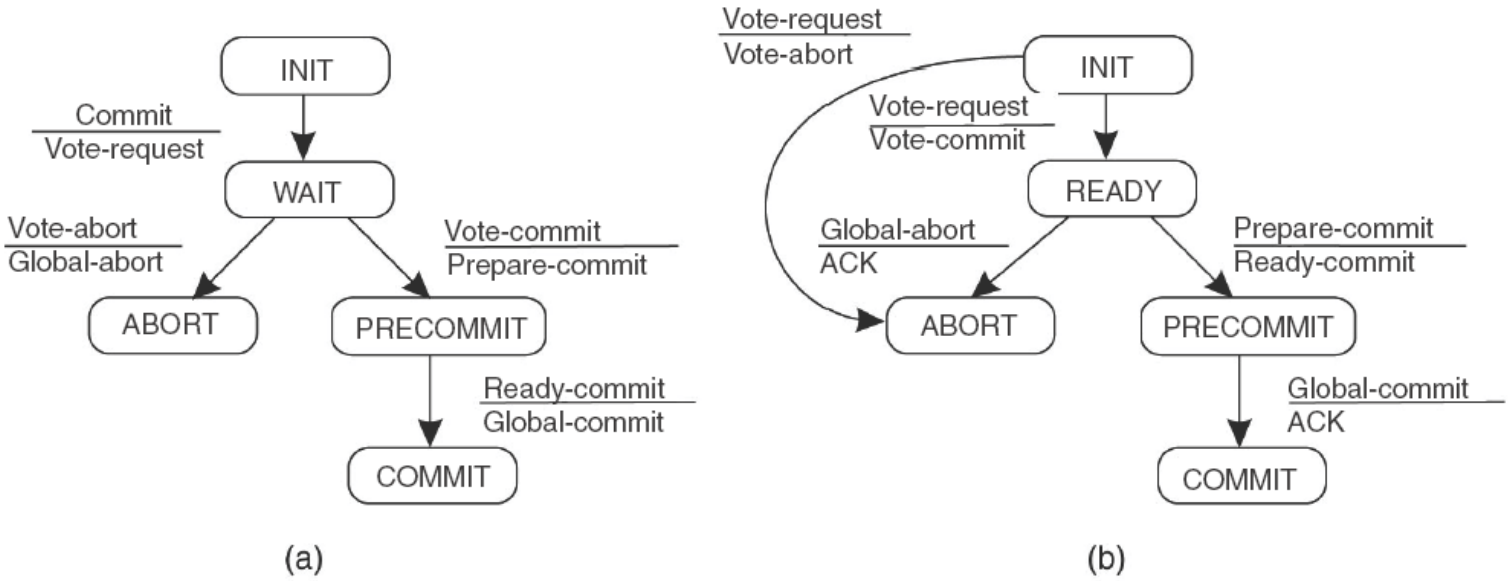


Figura 8.19 (a) Máquina de estado finito para o coordenador em 3PC. (b) Máquina de estado finito para um participante.

Tolerância a Falha

1. Introdução à tolerância a falha
2. Resiliência **de** processo
3. Comunicação confiável cliente-servidor
4. Comunicação confiável de grupo
5. Comprometimento distribuído
6. Recuperação

Recuperação

- Ocorrida uma falha, é necessário não apenas identificá-la, mas recuperar-se da mesma e voltar para um estado correto.
- Essencialmente existem 2 maneiras de se recuperar:
  - Recuperação retroativa – volta para um estado anterior à falha;
  - Recuperação para a frente – tenta levar o sistema para um novo estado correto para que possa continuar a executar.

Recuperação
Garantia de estado estável

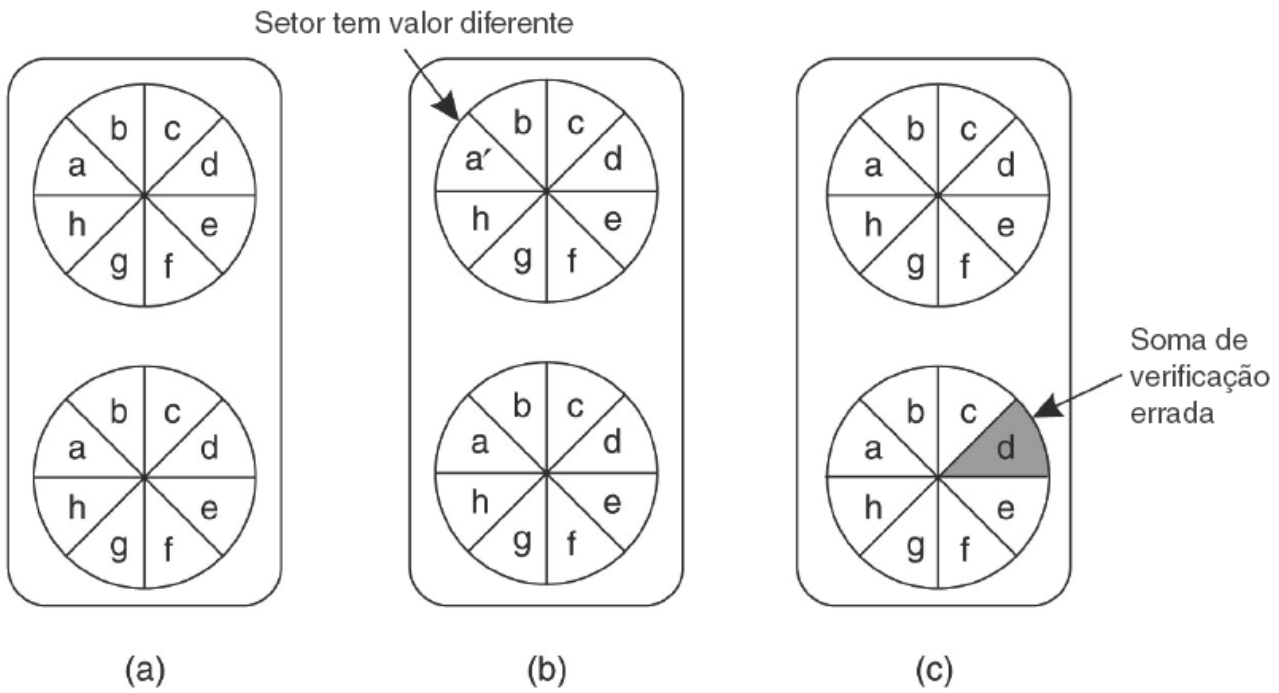


Figura 8.20 (a) Armazenamento estável. (b) Queda após a unidade 1 ter sido atualizada. (c) Local com erro.

Recuperação
Pontos de verificação

- Independentes:

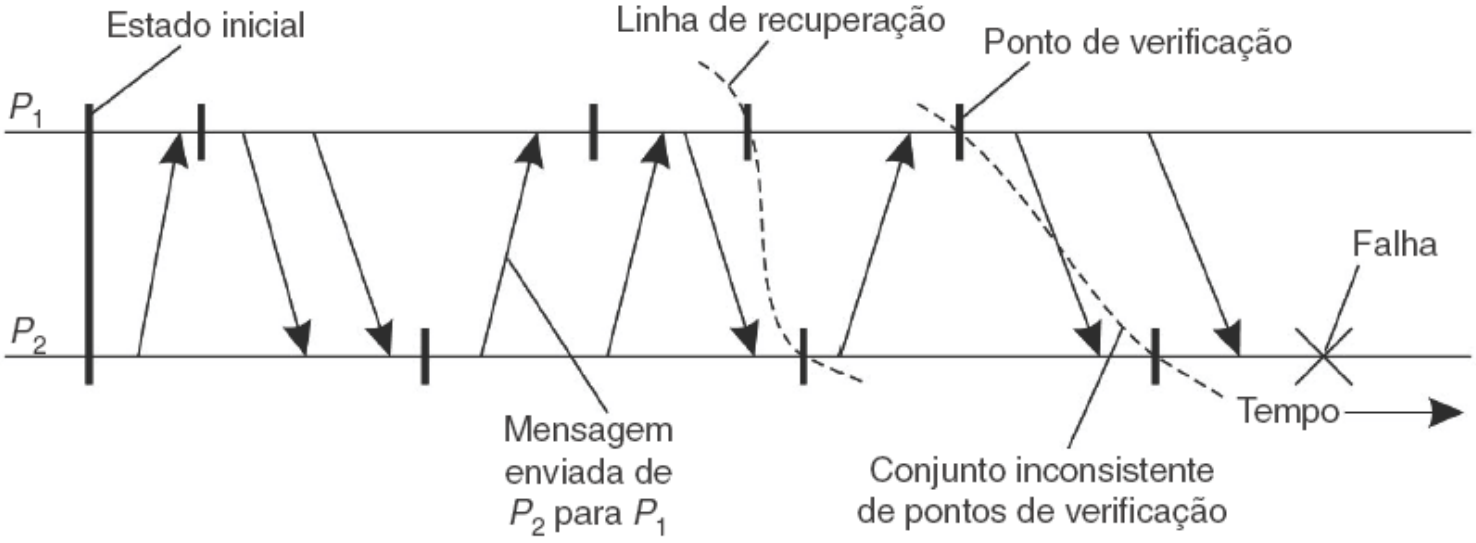


Figura 8.21 Linha de recuperação.

- Coordenados: usam, por exemplo, bloqueio em duas fases

Recuperação - Caracterização de esquemas de registro de mensagens

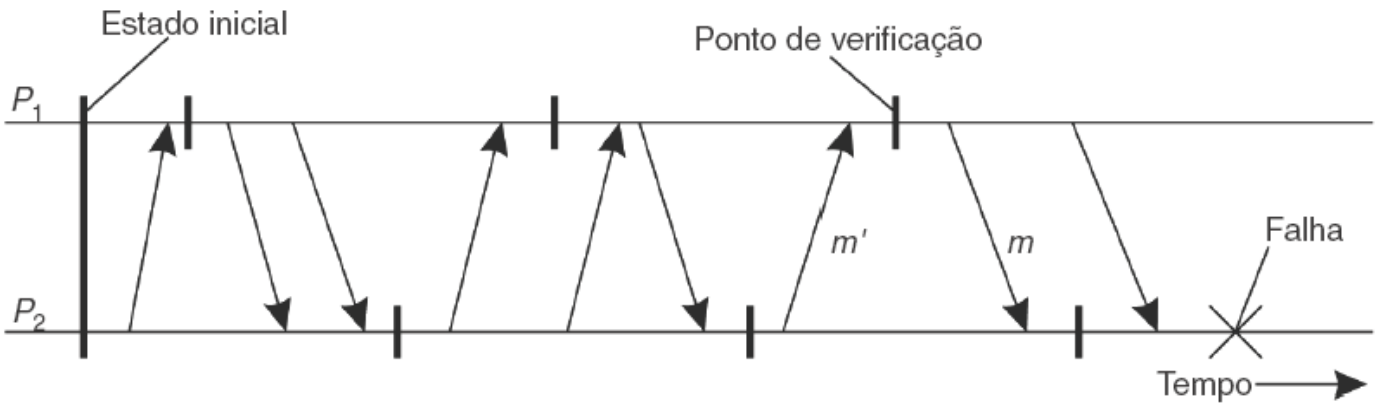


Figura 8.22 Efeito dominó.