

Sistemas Distribuídos

Aula 20

Roteiro

- Tipos de falhas
- Falhas Bizantinas
- Grupos redundantes
- Acordos bizantinos
- Propriedades
- Algoritmos de Consenso

Tipos de Falhas

- Sistemas falham por razões diferentes
- Classificação dos tipos de falha
 - permite oferecer robustez em diferentes casos

Type of failure	Description
Crash failure	A server halts, but is working correctly until it halts
Omission failure <i>Receive omission</i> <i>Send omission</i>	A server fails to respond to incoming requests A server fails to receive incoming messages A server fails to send messages
Timing failure	A server's response lies outside the specified time interval
Response failure <i>Value failure</i> <i>State transition failure</i>	A server's response is incorrect The value of the response is wrong The server deviates from the correct flow of control
Arbitrary failure	A server may produce arbitrary responses at arbitrary times

Mais fácil
de lidar!

Mais difícil
de lidar!

Falhas Bizantinas

- Outro nome para “*Arbitrary Failures*”
 - nome vem de histórias de traição e corrupção no Império Bizantino (cunhado por Lamport)
- Modelo mais geral de falha, que também inclui comportamento malicioso
 - “tudo pode acontecer”
- Respostas podem ser incorretas, inclusive de forma proposital para induzir falhas
 - adversário que controla algumas partes do sistema deseja induzir o sistema a falhar

Grupos Redundantes



- Como lidar com falhas em processos (ou subsistemas)?

Redundância!

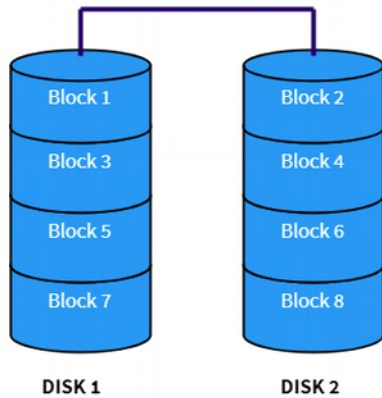
- Ideia: grupo de processos redundantes que executam uma função e se comunicam
 - processos (ou subsistemas) distintos, que podem responder um pelo outro
 - ex. grupo de discos, grupo de SGBDs, etc
- Outros processos interagem com o grupo (sem saber que é um grupo)
 - grupo é transparente para outros processos

Exemplo: RAID

- Redundant Array of Inexpensive Disks (RAID)
 - grupo de discos que se apresentam (para SO) como um único disco
 - controladora RAID de disco esconde o grupo
- Melhorar desempenho
 - via *stripping* dos blocos e acesso simultâneo
- Aumentar confiabilidade
 - via paridade/redundância dos blocos
- Diferentes configurações
 - com vantagens e desvantagens entre desempenho e confiabilidade

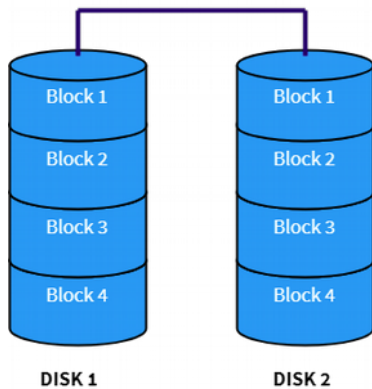
Configurações RAID

RAID 0



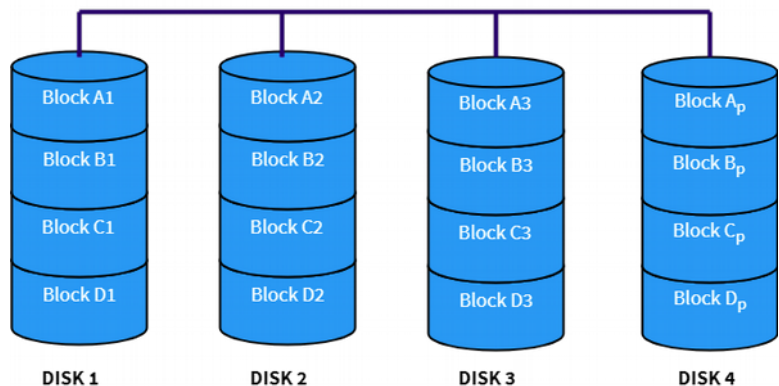
- RAID 0: dois discos para melhorar desempenho, não possui redundância. Capacidade de 2 discos.

RAID 1



- RAID 1: dois discos (um é cópia do outro), melhora desempenho de leitura, oferece redundância. Capacidade de 1 disco

RAID 4

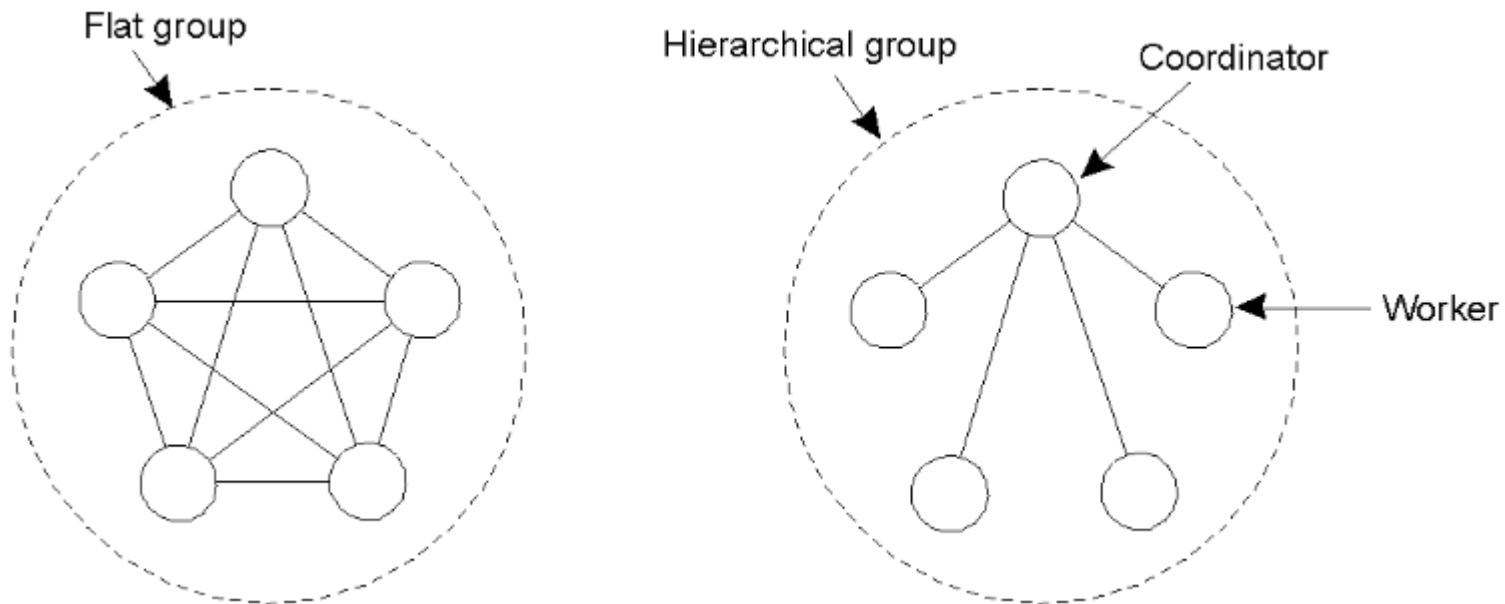


- RAID 4: 4 discos (bloco de paridade), melhora desempenho de leitura e escrita, oferece redundância. Capacidade de 3 discos.

Organização de Grupos



- Como organizar grupos de processos redundantes?



- **Com hierarquia:** coordenador lidera e gerencia o grupo, faz interface com o exterior
- **Sem hierarquia:** papéis são simétricos, gerência mais complicada, não possui ponto único de falha
- **RAID:** hierárquico (controladora é o coordenador)

Falha em Grupos Redundantes

- Quantos processos podem falhar e grupo continuar operacional?

Depende do tipo de falha!

- e do tipo de grupo
- Assumir grupos simétricos com k processos redundantes, sem hierarquia
- *Crash failure*: basta ter 1 processo operacional
 - sistema tolera até $k - 1$ falhas no grupo
- *Value failure*: processos em falha geram valores errados. Usar valor da maioria como resposta
 - sistema tolera até $k/2 - 1$ falhas no grupo

Consenso / Acordo

- Processos em sistemas distribuídos precisam de consenso (acordo)
 - ex. definição do coordenador do grupo, definição de uma transação (commit/abort), etc
- Problema relativamente fácil, ignorando falhas
 - ou considerando alguns tipos de falha
 - ex. 2PC tolera alguns tipos de *Crash Failures*, *Omission Failures* e *Timing Failures*
- Considere falhas bizantinas (processos induzindo falhas de forma arbitrária)
- Processos que não estão em falha conseguem chegar a um consenso?

Acordos com Falhas Bizantinas

- **Problema:** processos conseguem chegar a um consenso em tempo finito na presença falhas bizantinas?
 - processos operacionais não sabem quais estão em falha bizantina (obviamente)
 - inicialmente estudado por Lamport et al. Em 82
- Características do sistema distribuído para estudo
- Comportamento dos processos: síncrono (todos processos “andam” juntos), ou assíncrono
- Ordenação das mensagens: FIFO (mensagens entregues em ordem), ou não
- Tipo de transmissão de rede: multicast (um para todos), ou unicast (um para um)
- Atraso das mensagens: limitado (mensagens chegam dentro de um tempo T), ou não

Acordos com Falhas Bizantinas

- Tabela indica casos onde é possível chegar a um consenso (com algum algoritmo, a saber)
 - com multicast e FIFO, é sempre possível
 - com síncrono e atraso limitado, é sempre possível
 - impossível em metade dos casos!

Process behavior		Message ordering				Communication delay
		Unordered		Ordered		
		Unicast	Multicast	Unicast	Multicast	
Synchronous	{	X	X	X	X	Bounded
				X	X	Unbounded
Asynchronous	{				X	Bounded
					X	Unbounded
		Unicast	Multicast	Unicast	Multicast	
		Message transmission				

Não é possível

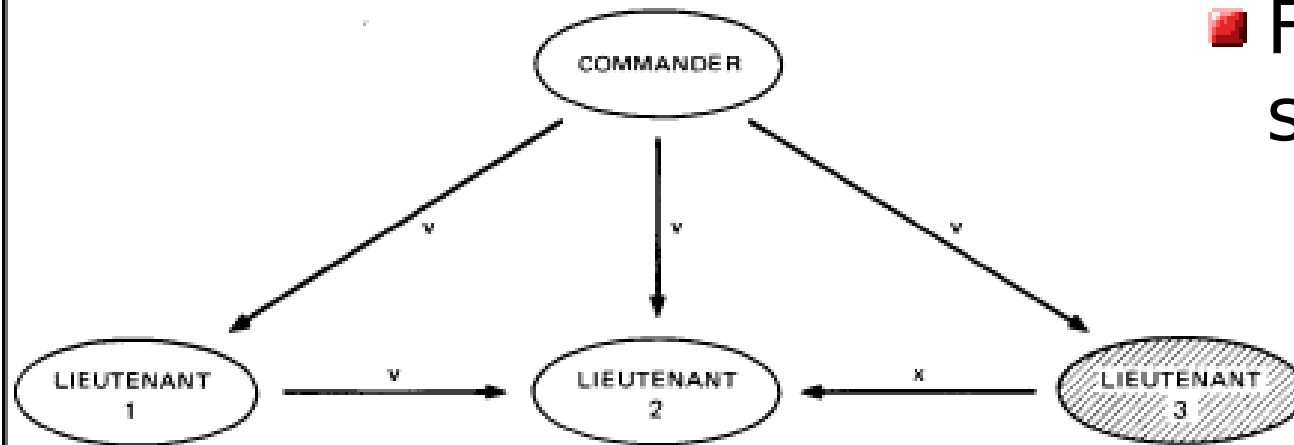
Não é possível
chegar a um
consenso!

Protocolo de Acordo Bizantino

- Assumir processos síncronos, comunicação unicast, com FIFO e atraso limitado
- **Problema:** N processos, um deles envia um valor v aos outros
 - todos os operacionais devem concordar no valor v , ou concordar que desconhecem o valor
 - processo que envia v pode estar em falha bizantina
- Algoritmo proposto por Lamport et al. em 82
 - 1) Cada processo envia a todos os outros o valor que recebeu
 - 2) Ao receber de todos, verifica se há uma maioria (este é o valor de consenso) ou que não consenso

Exemplo

■ $N=4$, $m=1$ – Caso L3 é bizantino

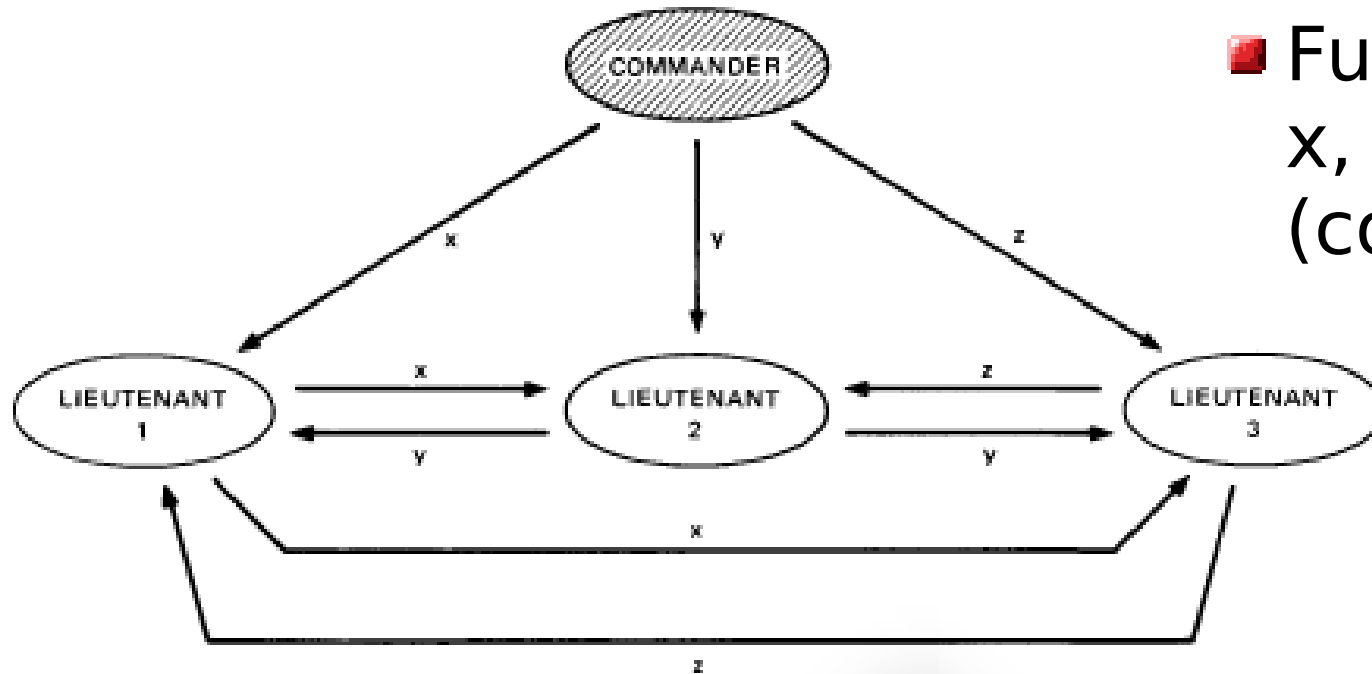


■ Funciona se L3 e L1 são bizantinos? Não!

- C envia v a todos,
- L1, L2 envia v para todos os outros
- L3 envia x para todos os outros
- L2 (L1) recebe v de C e L1 (L2) e x de L3, logo assume que v é o consenso (OK)

Exemplo

■ $N=4$, $m=1$ – Caso C é bizantino



■ Funciona se C enviar x, x, y ? Sim (consenso é x)

■ C envia valores diferentes

■ L1, L2, L3 enviam respectivos valores

■ Nenhum valor tem maioria em nenhum processo, não há consenso (OK)

Algoritmo Geral

- Algoritmo anterior funciona para $m=1$ (um processo em falha bizantina dentre 4 processos)
 - um processo inicial envia v e “sai do jogo”
- Para $m>1$ podemos usar um algoritmo recursivo
- Recursivo no número de processos em falha bizantina
 - Ideia: eliminar um processo bizantino por rodada
 - usa regra majoritária em cada rodada
- Algoritmo mais elaborado (fora de escopo)
 - disponível em livro-texto mais avançado

Propriedades de Acordo Bizantino

- Quando acordo pode ser obtido?
 - assumir unicast, FIFO, atraso limitado
- Se k estão em falha bizantina, precisamos de $2k+1$ operacional
 - de $3k+1$ processos apenas k podem estar em falha bizantina
 - resultado fundamental, conhecido como regra do 2/3
 - maior do que $1/2$ necessário para maioria simples (o caso de *value failure*)
- Não existe algoritmo que consiga chegar a um acordo caso contrário

Algoritmo de Consenso

- Chegar a consenso/acordo (ex. valor de uma variável) em um sistema distribuído mediante falhas
- Propriedades
- **Término:** eventualmente todo processo operacional termina o algoritmo
- **Integridade:** se todos os processos operacionais propõem valor v , todos operacionais decidem por v
- **Acordo:** todos processos operacionais chegam ao mesmo valor
- Paxos: conjunto de algoritmos para problema de consenso (para diferentes condições)
 - muitas implementações, base para outros sistemas

Algoritmo de Consenso

- Problema antigo, difícil e importante na Computação
 - ainda estudado, na teoria e na prática
- Na moda por conta de *blockchains*
 - sistema distribuído precisa de consenso na operação das transações (ex. compra/venda no bitcoin)
 - evitam o problema do “double spending”
 - baseados em “proof-of-work” (bitcoin)

Assunto para ser explorado!