Replicação

Tolerância a faltas

- Duas técnicas principais
  - Recuperação “para trás”
  - Recuperação “para a frente”

Recuperação “para trás”

- Guardar o estado algures, periodicamente ou em momentos relevantes para a aplicação
  - Usando um algoritmo para salvaguarda distribuída (aula 3)
- Quando um ou mais processos falham, estes são relançados o mais rapidamente possivel
- O novos processos lêem o último estado guardado e recomeçam a execução a partir desse estado
- Tipicamente designa-se esta solução por “checkpoint-recovery”

Recuperação para a frente

- São mantidas várias cópias (réplicas) do processo
- Quando uma réplica é alterada as restantes réplica devem também ser actualizadas
- Se uma réplica falhas os clientes podem usar outra réplica

Recapitulando…

- Na recuperação para trás:
  - Os estados guardados pelos vários **processos que falharam** devem estar mutuamente coerentes

- Na recuperação para a frente
  - Os estado das **réplicas que sobrevivem** devem estar mútuamente coerentes

Recuperação para a frente: Qual a estrutura de dados replicada?

- Registo
  - Vimos atrás (algoritmo ABD)
- Espaço de tuplos
  - Vimos atrás (algoritmo Xu-Liskov)
- Qualquer estrutura de dados, oferecendo uma qualquer interface remota
  - Exige algoritmos de replicação genéricos
  - Não são tão otimizados como os anteriores, pois não podem recorrer a otimizações específicas de uma estrutura de dados
  - Tema desta aula

Tentemos construir algoritmos de replicação genéricos *(por recuperação para a frente)*

- Duas variantes principais
  - Primário-secundário
  - Replicação de máquina de estados

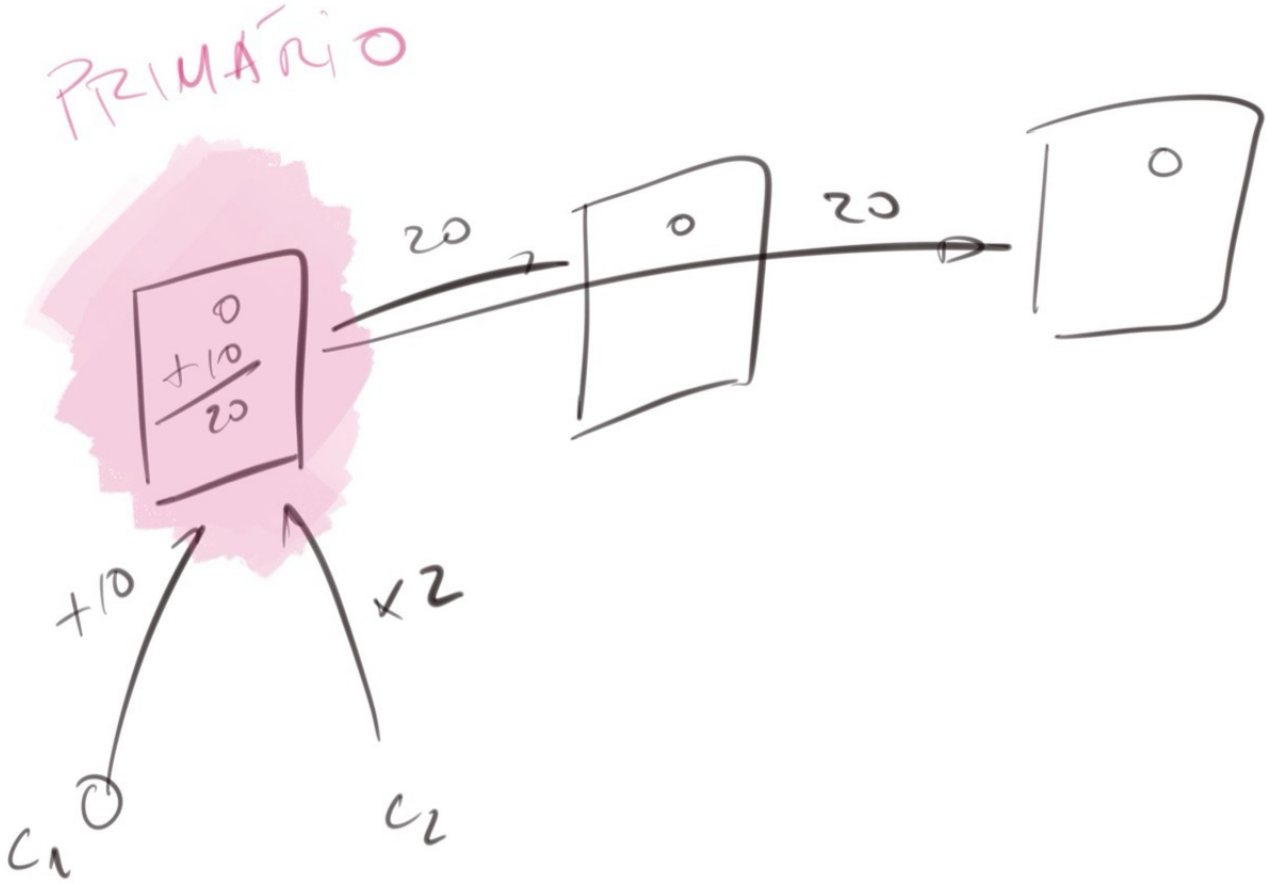
- Objetivo: assegurar **linearizabilidade** (coerência forte)

Primário-secundário (um esboço)

- Um processo é eleito como primário
  - Se o primário falha é eleito novo primário
- Os clientes enviam pedidos ao primário
- Para cada pedido recebido, primário:
  - Executa o pedido
  - Propaga o estado para os secundários e aguarda confirmação de todos
  - Responde ao cliente
  - (E processa o próximo pedido que esteja na fila de pedidos)

partes sombreadas: veremos depois como as concretizar

Primário-secundário (um esboço)

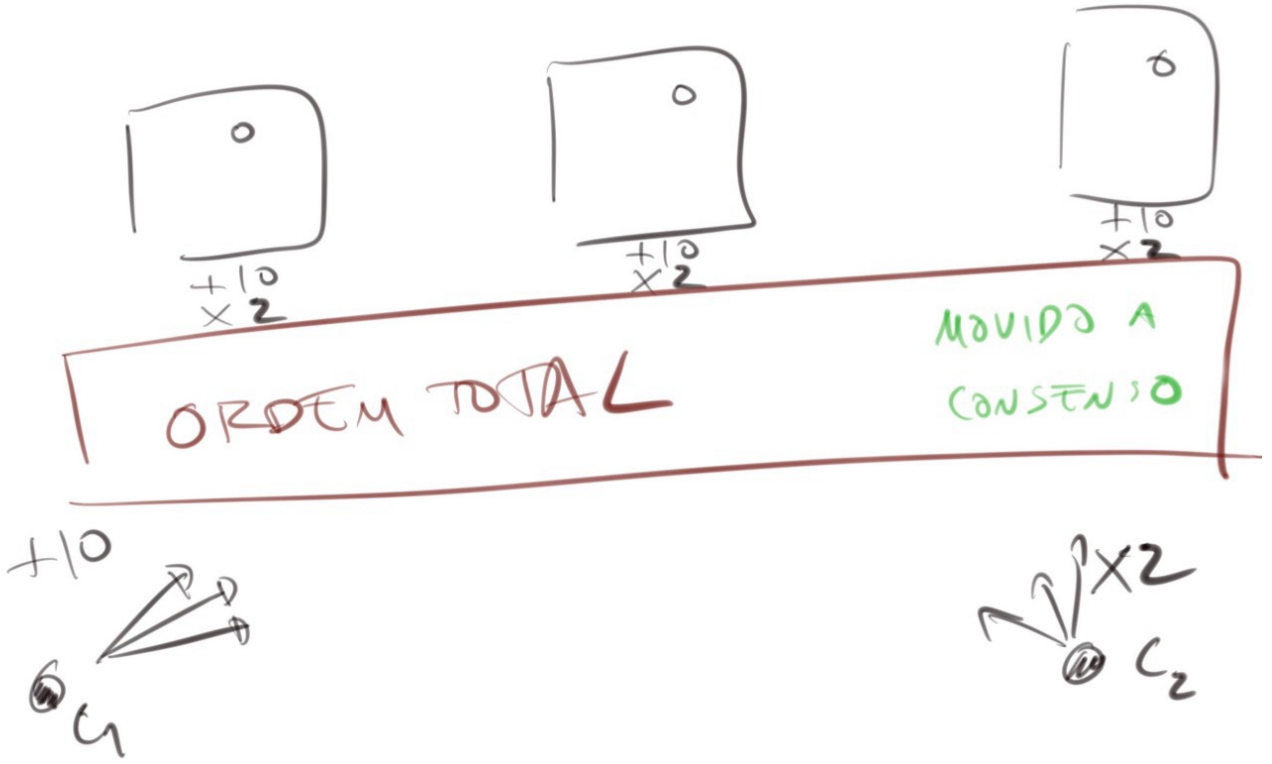


Replicação de máquina de estados (um esboço)

- Os clientes enviam os pedidos para todas as réplicas
- Todos os pedidos são ordenados por ordem total
- Todas as replicas processam os mesmos pedidos, pela mesma ordem
  - Assume-se que operações são determinísticas => replicas ficarão idênticas

partes sombreadas: veremos depois como as concretizar

Replicação de máquina de estados (um esboço)



Primário secundário vs RME

- Primário-secundário
  - Suporta operações não deterministas (o líder decide o resultado)
  - Se o líder produzir um valor errado, este valor é propagado para as réplicas

- Replicação máquina de estados
  - Se uma réplica produzir um valor errado não afecta as outras réplicas
  - A operações necessitam de ser deterministas

Abstrações para a construção de sistemas replicados

Abstracções para a construção de sistemas replicados

- Canais Perfeitos
- Difusão Fiável Regular
- Difusão Fiável Uniforme
- Difusão Atómica
- Sincronia na Vista

Canais Perfeitos

- Informalmente:
  - Garante a entrega de mensagens, ponto a ponto. de forma ordenada, no caso em que tanto o emissor como o destinatário não falham

- Como fazer:
  - Retransmitir uma mensagem até que a receção desta seja confirmada pelo destinatário
  - Usar ids de mensagens e não entregar uma mensagem antes daquelas com id inferior já terem sido entregues

Difusão Fiável

- Informalmente:
  - Permite enviar uma mensagem em difusão com a garantia que todos os destinatários recebem a mensagem ou nenhum recebe
- Duas variantes
  - Regular
  - Uniforme

Difusão Fiável Regular

- Seja *m* uma mensagem enviada para um grupo de processos *{p1, p2, ..., pN}* por um membro desse grupo.
- *Validade:* se um processo correto *pi* envia *m* então to mais cedo ou mais tarde *pi* entrega *m*
- *Não-duplicação:* nenhuma mensagem m é entregue mais do que uma vez
- *Não-criação:* se uma mensagem *m* é entregue então *m* foi enviada por um processo correto
- Acordo: se um processo correto entrega *m* então todos os processos corretos entregam *m*

Difusão Fiável Regular: algoritmo

- O emissor envia a mensagem usando canais perfeitos para todos os membros do grupo
- Quando um membro do grupo recebe a mensagem, entrega-a à aplicação e reenvia-a para todos os membros do grupo

Difusão Fiável Uniforme

- Seja *m* uma mensagem enviada para um grupo de processos *{p1, p2, ..., pN}* por um membro desse grupo.
- *Validade:* se um processo correto *pi* envia *m* então to mais cedo ou mais tarde *pi* entrega *m*
- *Não-duplicação:* nenhuma mensagem m é entregue mais do que uma vez
- *Não-criação:* se uma mensagem *m* é entregue então *m* foi enviada por um processo correto
- Acordo: se um processo ~~correcto~~ entrega *m* então todos os processos corretos entregam *m*

Difusão Fiável Uniforme: algoritmo

- O emissor envia a mensagem usando canais perfeitos para todos os membros do grupo
- Quando um membro do grupo recebe a mensagem, reenvia-a para todos os membros do grupo
- Quando um membro receber uma mensagem *m* de *f* membros distintos, entrega a mensagem m à aplicação
- Em que *f* é o número de processos que pode falhar.

$$f < N/2$$

Difusão Atómica

- Informalmente:
  - Fiável: se uma réplica recebe o pedido, todas as réplicas recebem o pedido
  - Ordem total: todas as réplicas recebem os pedidos pela mesma ordem

Dois algorimos para o caso sem falhas

- Ordem total baseada em sequenciador
- Ordem total baseada em acordo colectivo

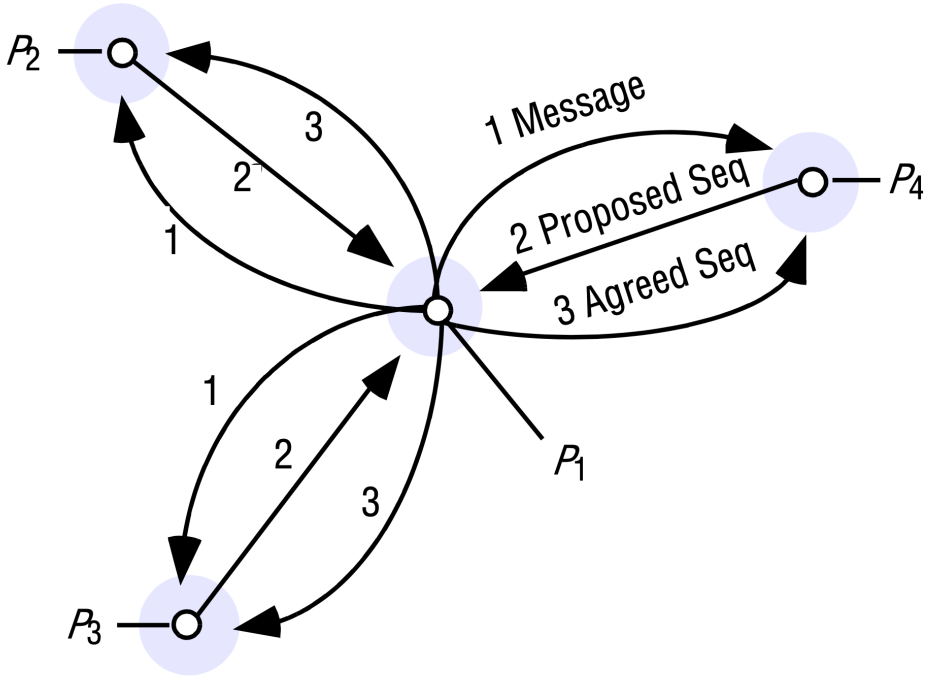
Ordem total baseada em sequenciador

- As mensagems são enviadas para todas as réplicas usando um algoritmo de difusão fiável
- Uma das réplica é eleita líder, decide qual a ordem pela qual as mensagens devem ser processadas, e envia esta informação para as réplicas restantes

Ordem total baseada em sequenciador

- Quando existe uma falha, por exemplo do líder, as réplicas podem ser confrontadas com um estado incoerente:
  - Mensagens de dados que não foram ordenadas pelo líder
  - Mensagens do líder referentes a mensagens que ainda não chegaram
  - Para além disto, réplicas distintas podem ter persectivas diferentes de qual o estado do sistema
- É necessário executar um algoritmo de recuperação complexo, baseado em consenso.

Acordo colectivo



Algoritmo explicado nas páginas 655-656 do livro

Algumas notas sobre este algoritmo

- Quando há falhas sofre dos mesmos problemas e requer uma solução semelhante à referida anteriormente para a ordem total baseada num sequenciador
- Este algoritmo funciona mesmo que cada mensagem seja enviada para um conjunto diferente de nós
  - O que acontece se aplicar este algoritmo ao algoritmo de exclusão mútua do Maekawa?

Sincronia na Vista

- Informalmente:
  - Permite mudar a filiação de um grupo de processos de uma forma que facilita a tolerância a faltas

Sincronia na Vista

- Vista: conjunto de processos que pertence ao grupo
  - Novos membros podem ser adicionados dinamicamente
  - Um processo pode sair do grupo voluntariamente ou ser expulso case falhe
- O sistema evolui através de uma sequência totalmente ordenada de vistas
- Exemplo:
  - *V1* = {p1, p2, p3}
  - *V2* = {p1, p2, p3, p4}
  - *V3* = {p1, p2, p3, p4, p5}
  - *V4* = {p2, p3, p4, p5}

Sincronia na Vista

- Um processo considera-se correcto numa vista Vi se faz parte da vista Vi e faz parte da vista Vi+1
  - Por oposição, um processo que faz parte da vista Vi e que não faz parte da vista Vi+1, pode ter falhado durante a vista Vi

- Uma aplicação à qual já foi entregue a vista Vi mas à qual ainda não foi entregue a vista Vi+1 diz-se que “está na vista Vi”

Sincronia na Vista

- Uma aplicação que usa o modelo de sincronia na vista recebe vistas e mensagens
- Se uma aplicação envia uma mensagem m quando está na vista *Vi*, a messagem *m* diz-se que foi enviada na vista *Vi*
- Se uma mensagem *m* é entregue à aplicação depois da vista *Vi* ser entregue e antes da vista *Vi+1* ser entregue, a mensagem *m* diz-se que foi entregue na vista *Vi*
- Uma mensagem *m* enviada na vista *Vi* é entregue na vista *Vi*

Sincronia na Vista

- Difusão fiável *síncrona na vista*:
  - Se um processo correto *p* na vista *Vi* envia uma mensagem *m* na vista *Vi*, então *m* é entregue a *p* na vista *Vi*
  - Se um processo entrega uma mensagem *m* na vista *Vi,* todos os processos correctos da vista *Vi* entregam *m* na vista *Vi*

- Corolário:
  - Dois processos que entregam a vista *Vi* e a vista *Vi+1* entregam exactamente o mesmo conjunto de mensagens na vista *Vi*

Mudança de vista

- Para mudar a vista é necessário obrigar as aplicações interromper temporariamente a transmissão de mensagens de forma a que o conjunto de mensagens a entregar na vista seja finito
- Para além disso, é necessário executar um algoritmo de coordenação para garantir que todos os processos correctos chegam a acordo sobre:
  - Qual a composição da próxima vista
  - Qual o conjunto de mensagens a entregar antes de mudar a vista

Não estudaremos estes algoritmos em SD

Usar estas abstrações para concretizar os algoritmos que esboçámos hoje

Primário-secundário (agora concretizado)

- Réplicas usam **sincronia na vista** para lidar com falha do primário:
  - Quando o primário *p*, algum tempo depois será entregue nova vista sem *p*
  - Quando nova vista é entregue e o anterior primário não consta nela, os restantes processos elegem o novo primário
    - Pode ser o primeiro membro da nova vista
    - Ou podemos usar outro algoritmo de eleição de líder
  - O novo primário anuncia o seu endereço num serviço de nomes (para que os clientes o descubram)
- Primário usa **difusão fiável uniforme *síncrona na vista*** para propagar novos estados aos secundários
  - Só reponde ao cliente quando a uniformidade está garantida

Replicação de máquina de estados (agora concretizado)

- Processos usam **sincronia na vista**
- Clientes usam **difusão atómica** *síncrona na vista* para enviar pedidos para todas as réplicas