

Sistemas Distribuídos

Comunicação em Grupo

abril de 2017

Primitiva de comunicação em grupo

- um processo envia uma mensagem para um grupo de processos e todos os destinatários recebem
- garantias variadas: confiabilidade e ordenação

grupos fortemente acoplados:

- replicação de serviços
 - confiabilidade
 - tempo de resposta
- clientes com estado compartilhado
- computação científica
- ...

Primitiva de comunicação em grupo

- um processo envia uma mensagem para um grupo de processos e *todos* os destinatários recebem
- garantias variadas

grupos fracamente acoplados:

- observadores “melhor esforço”
 - monitoramento de ambientes de execução
 - servidores replicados
 - consumo de dados: valores de mercados, etc

Primitiva de comunicação em grupo

- um processo envia uma mensagem para um grupo de processos e *todos* os destinatários recebem
- garantias variadas: publish/subscribe e serviços de eventos

- em redes: conceito de entrega *para todos* e *para alguns*
- conceitos se confundem em sistemas distribuídos

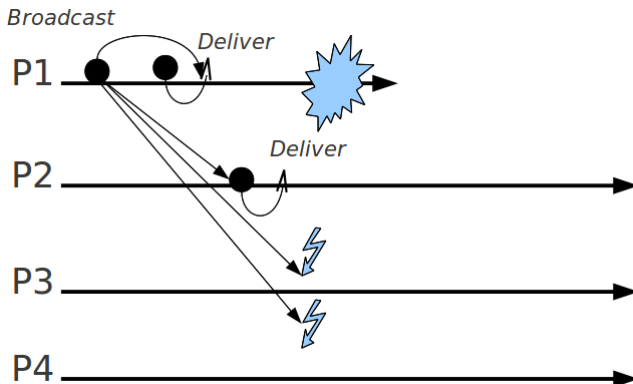
- alguns recebem e outros não
- processos recebem mensagens em ordens diferentes

- confiabilidade
- ordenação

Implementação

- chegada X entrega

Exemplo: envio de msg para um grupo de processos

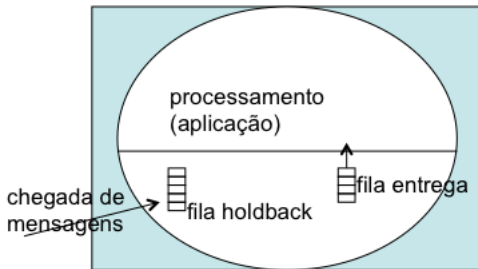


Níveis de garantia de entrega

- 1 *Melhor-esforço (best-effort)*: garantia de entrega entre processos corretos
- 2 *Confiável (reliable)*: garantia *all-or-nothing* mesmo se o emissor falhar
- 3 *Ordem xyz*: garantia de entrega na ordem *xyz*

Garantia de entrega: Implementação

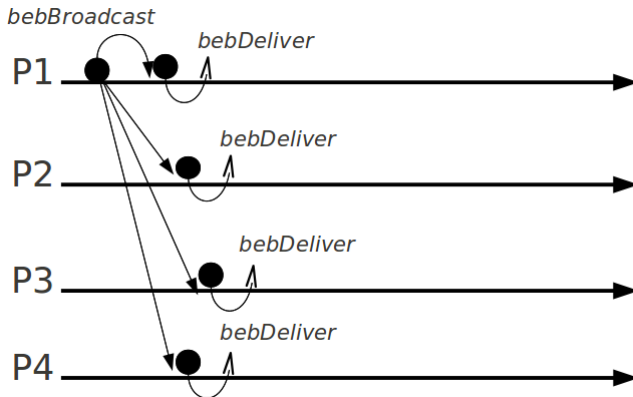
- uso de *holdback queue*
- em geral, mensagens com *identificadores únicos*
 - o que já existe no SO agora em bibliotecas ou middlewares
 - custos de espaço e processamento!



Garantia de entrega: Implementação

- Um processo envia uma msg em um passo de comunicação para todos os processos do sistema, incluindo ele mesmo
- O custo para garantir confiabilidade é apenas do lado do emissor (se ele falhar, nenhuma garantia de entrega é oferecida)

Exemplo de execução de Bcast básico



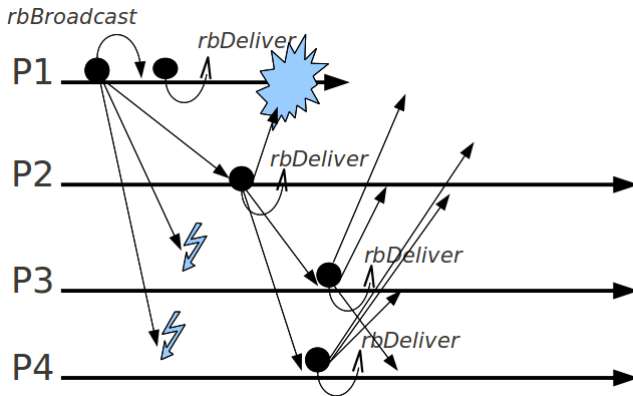
Desempenho: o algoritmo requer um único passo de comunicação e troca N mensagens

- 1 retransmite eternamente (didático mas sem bom desempenho...)
- 2 elimina duplicatas
- 3 entrega mesmo com falhas intermitentes...

- 1 se transmissor não falhar, garantia de entrega a todos
- 2 se transmissor falhar:
 - processos podem “discordar” sobre entrega de mensagem
 - broadcast de melhor esforço pode não ter transmitido para todos ou falha pode ter ocorrido antes de terem sido feitas as retransmissões necessárias

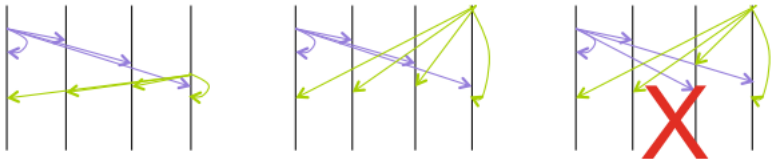
- todos os processos devem receber (tratar) o mesmo conjunto de mensagens
 - noção de *acordo*
- solução para modelo *fail-stop*

Exemplo de execução de Bcast confiável regular



- As entregas anteriores não garantem ordem de entrega das mensagens enviadas por processos diferentes (**mensagens de um mesmo processo devem ser entregues na ordem em que foram difundidas**)
- Algumas aplicações precisam de garantias de ordem de entrega.
 - ordem total
 - ordem causal

Ordem Total



- Não importa a ordem de entrega, mas deve ser a mesma em todos os processos do grupo.

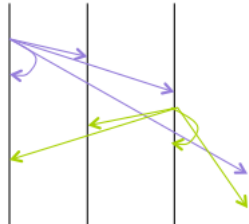
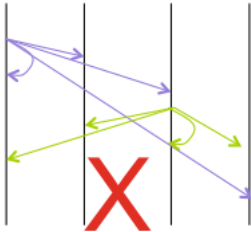
Ordem total com sequenciador

Ordem total com votação de ordem

Ordem Causal

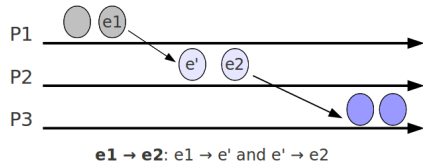
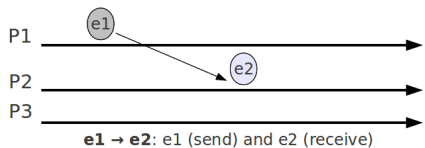
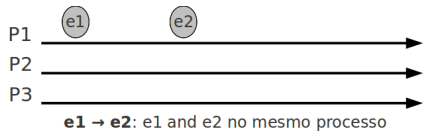
possível relação de causalidade

- uma mensagem pode ter sido consequencia de outra ainda não vista



Causalidade em eventos distribuídos...

- $a \rightarrow b$: o evento a ocorre antes do evento b se
 - a ocorre antes de b em um mesmo processo P
 - $a = \text{send}(m)$ no processo P e $b = \text{receive}(m)$ no processo Q
 - existe c tal que a precede c e c precede b
- $a \parallel b$: os eventos a e b são concorrentes



Relógio Lógico (Lamport)

- conceito de relógio lógico modela ordenação (parcial) de eventos
- cada evento associado a um valor $r(e_i)$ de forma que $e_i \prec e_j \Rightarrow r(e_i) < r(e_j)$

- cada processo P mantém um contador, inicialmente $RL_P = 0$
- a cada evento e , P associa um valor $RL_P(e)$ e incrementa seu contador
- se evento e é envio de mensagem, P acrescenta campo *timestamp* com valor $RL_P(e)$
- ao receber mensagem M com timestamp TS , processo Q faz $RL_Q = \max(RL_Q, TS) + 1$

- uso de relógio lógico e timestamps: funciona se tempo de entrega de mensagens tem limite máximo

- uso de mensagens “posteriores” para validar mensagens com timestamp TS

Vetor de timestamps: idéia básica

- Dados N processos, usa-se um vetor de N elementos (ao invés de um valor escalar)
- A cada evento “e”, associa-se um **vetor de timestamps** cujo i -ésimo elemento indica quantos mensagens do processo i já foram vistas

Definição da relação $<$

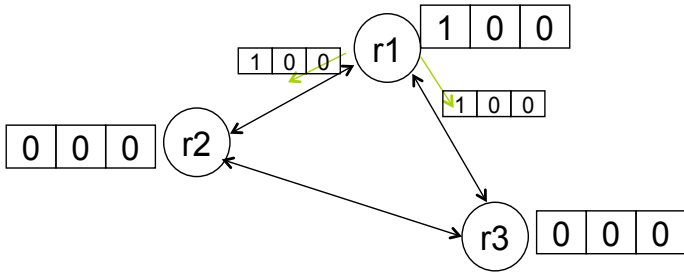
- Sejam a e b dois eventos quaisquer
- $VT(a) < VT(b)$:
 - 1 $\forall i, VT(a)[i] \leq VT(b)[i]$
 - 2 $\exists j, VT(a)[j] < VT(b)[j]$
- Note que $<$ é uma *ordem parcial*, existem eventos que não podem ser comparados por serem concorrentes
 - Ex., $x = [0, 0, 1, 0]$ e $y = [1, 0, 0, 0]$, pois $VT(x) \neq VT(y)$ e não vale $VT(x) < VT(y)$ nem $VT(y) < VT(x)$

Funcionamento

- Cada processo mantém seu VT que começa com 0 em todas as posições
- Para cada evento “e” local e de envio de msg, incrementa o componente do processo local no VT e associa VT_p ao evento “e”: $VT_p[p] = VT_p[p] + 1$ e $VT(e) = VT_p$
- Quando Q recebe uma mensagem, atualiza cada campo do seu VT:
 - $VT_q[i] = VT_q[i] + 1, i = q$
 - $VT_q[i] = \max(VT_q[i], VT_m[i]), i \neq q$

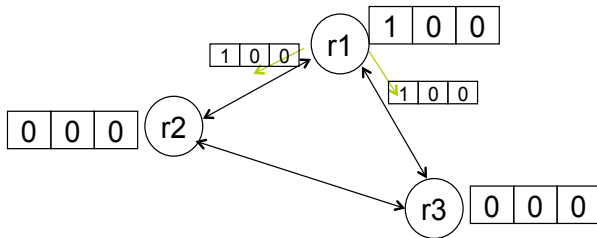
Esta última parte garante que tudo que acontecer depois em P_q passa a ser causalmente relacionado com tudo o que aconteceu antes em P_i

Exemplo de uso de VT para ordenação de eventos

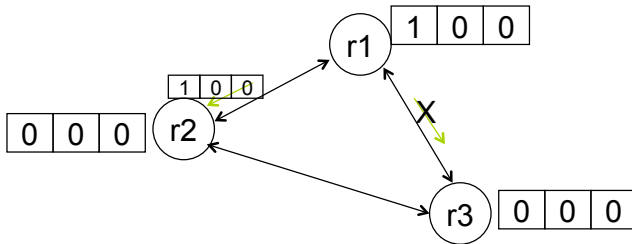


- é necessário não apenas estabelecer ordem mas garantir que foram vistas todas as mensagens que precedem causalmente determinada mensagem!

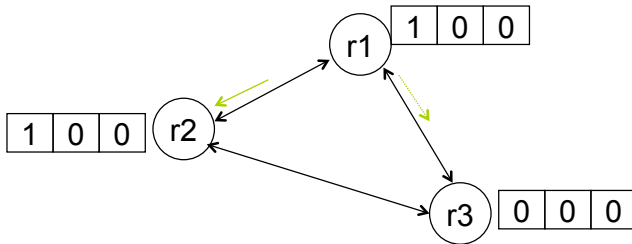
Exemplo de entrega em ordem causal



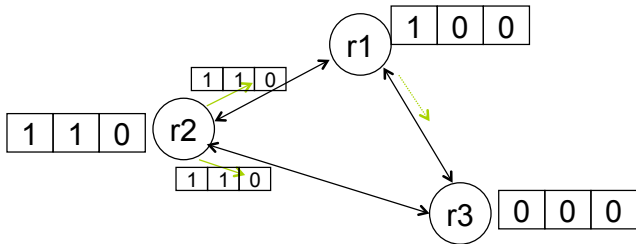
Exemplo de entrega em ordem causal



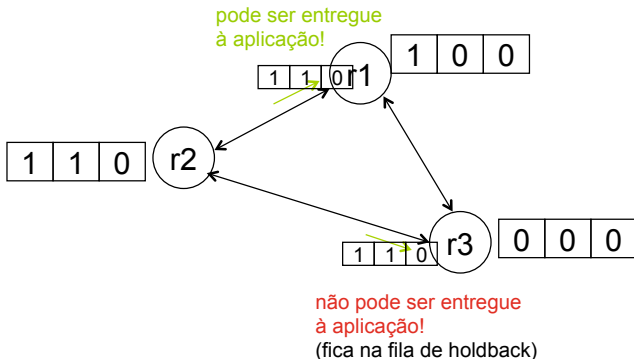
Exemplo de entrega em ordem causal



Exemplo de entrega em ordem causal



Exemplo de entrega em ordem causal



- a entrega deve ser postergada até as mensagens anteriores serem vistas
- ack's ou reenvios sob demanda