MC714

Sistemas Distribuídos

1° semestre, 2017

Sincronização

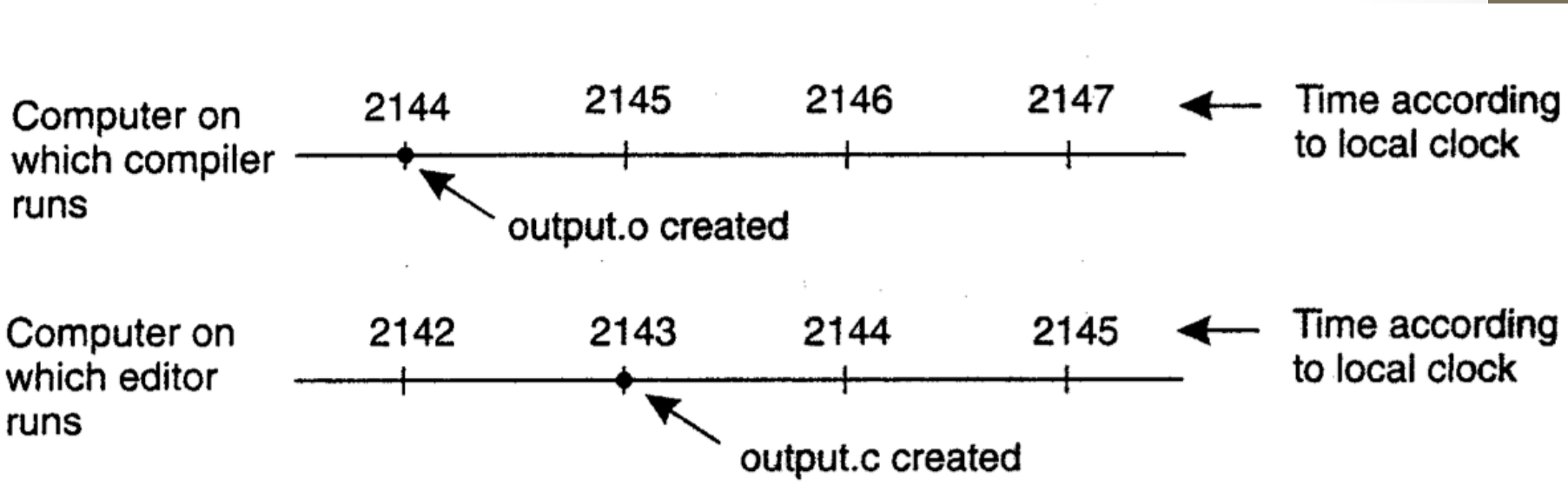
Sincronização

- Evitar acesso concorrente a recursos
- Concordar com ordem que eventos ocorreram
  - Qual mensagem foi enviada primeiro?
- Sincronização baseada em tempo real
- Sincronização relativa
- Pode necessitar de coordenador: eleição de líder

Sincronização de relógios

Sincronização de relógios

- Ex.: Make



- É possível sincronizar todos os relógios em um sistema distribuído?

Sincronização de relógios

- É possível sincronizar todos os relógios em um sistema distribuído?

Relógios físicos

Relógios físicos

- “Temporizador”
  - Número de ciclos de relógio após uma data inicial fixa no sistema.
- Sistema com N computadores
  - Defasagem de relógio

Medição do tempo

- Problema: tempo solar e tempo de relógios atômicos divergem.
- Sol: dia solar de 24h
  - 1s = 1/86.400 de um dia solar
- Relógio atômico: transições por segundo de átomo de césio 133.
  - 1s = 9.192.631.770 transições.
- Hoje, 86.400 segundos TAI (tempo atômico internacional) equivalem 3ms a menos que um dia solar médio.

Medição do tempo

- Solução: adicionar segundos à hora do relógio quando diferença > 800ms
  - Bureau International de l'Heure - BIH
- Sistema de medição baseado em segundos TAI constantes → Hora (ou tempo) coordenada universal (UTC)

- UTC: NIST broadcast por rádio WWV (precisão prática +- 10ms) e por satélite (0,5ms)

Sistema de posicionamento global

- Global positioning system – GPS
- Sistema distribuído de determinação de posição geográfica baseado em satélites e lançado em 1978.
- 29 (31) satélites a ~20.000km de altura
- Cada satélite tem até 4 relógios atômicos calibrados periodicamente
- Satélite transmite sua posição em broadcast com marcas de tempo
- Receptor na Terra calcula sua posição

Sistema de posicionamento global

- Leva um certo tempo para que os dados sobre posição de um satélite cheguem ao receptor.
- Relógio do receptor não está em sincronia com o do satélite.

- Relógios não estão perfeitamente sincronizados
  - Correção de 38 microssegundos por dia devido à relatividade (gravidade +45; dilatação do tempo -7)
  - Não leva em conta segundos extras UTC
- Velocidade de propagação não é constante
- Terra não é esfera perfeita

Algoritmos de sincronização de relógios

- Se uma máquina tiver receptor WWV, meta é manter todas as outras máquinas sincronizadas com ela.
- Se nenhuma tem receptor WWV, cada uma monitora seu próprio horário e objetivo é manter todas as máquinas com relógio o mais próximo possível.
- Cada máquina tem temporizador que provoca interrupção H vezes por segundo.
  - Manipulador de interrupção soma 1 a um relógio de software
  - Relógio mantém número de tics que ocorreram desde um instante pré-determinado.

Algoritmos de sincronização de relógios - Modelo

- C = valor do relógio
- Para UTC = t , $C_p(t)$ é valor do relógio da máquina *p*.
  - Ideal: $C_p(t)$ = t para todo p e t → $C'_p(t) = dC/dt$ seria 1
  - $C'_p(t)$ é a frequência do relógio de *p* no tempo t
  - $C'_p(t)$ - 1 é a defasagem do relógio de *p*
    - magnitude da diferença entre relógio de *p* e o relógio perfeito
  - $C_p(t)$ – t é o deslocamento em relação a uma hora específica

Algoritmos de sincronização de relógios - Modelo

- Temporizadores reais têm imprecisão
  - H = 60 → 216.000 ciclos por hora
  - Erro relativo em chips temporizadores: 10^{-5} → 215.998 a 216.002 ciclos por hora
- Se existir uma constante ρ tal que $1 - \rho \leq \frac{dC}{dt} \leq 1 + \rho$

então o temporizador está funcionando dentro da especificação.

- ρ: taxa máxima de deriva → especificado pelo fabricante
- Fig 93.

Algoritmos de sincronização de relógios - Modelo

- Defasagem Δt após sincronização: até $2\rho\Delta t$ se derivarem em direções opostas.
- Para garantir defasagem máxima δ, relógios devem ser sincronizados no mínimo a cada $\frac{\delta}{2\rho}$ segundos.

- Para esse modelo, diferença nos algoritmos de sincronização de relógios está na maneira como essa sincronização periódica é feita.

Algoritmos de sincronização de relógios

- Abordagem 1: clientes consultam servidor de tempo que possui hora precisa.
- Atrasos nas mensagens farão com que hora fique desatualizada.
- Fig. 94
- A envia req. a B com marca de tempo T1
- B marca T2 (sua hora), tempo que recebeu
- B envia T3 e T2.
- A marca T4. desloc = $\frac{(T2 - T1) + (T3 - T4)}{2}$

Algoritmos de sincronização de relógios

- Se relógio de A estiver adiantado, A precisaria atrasar seu relógio → consequências ruins.

- Pode ser feito de forma gradual, fazendo relógio andar mais devagar (ou mais rápido).

Protocolo de tempo de rede (NTP)

- Ajustado entre pares de servidores.
- B também consulta A para saber sua hora.
- Usa estimativa anterior de deslocamento, e atraso:

$$\delta = \frac{(T2 - T1) + (T4 - T3)}{2}$$

- Obtém 8 pares (deslocamento,atraso).
- Valor mínimo de atraso e respectivo deslocamento são adotados.

Protocolo de tempo de rede (NTP)

- B também poderia ajustar seu relógio em relação a A
- Se B é mais preciso, não deveria ajustar
- NTP divide servidores em estratos
  - Estrato 1: relógio de referência (WWV, relógio atômico)
  - A contacta B → A só ajusta seu relógio se estrato de A é maior
  - Após sincronização, A torna-se um nível mais alto que B
    - Se estrato de B é k, A torna-se k+1

Algoritmo de Berkeley

- NTP: servidor é passivo
- Unix/Berkeley: servidor é ativo → pergunta hora das máquinas de tempos em tempos
- Média das repostas e envia para as máquinas ajustarem seus relógios
- Adequado para sistemas sem um receptor WWV.
- Relógio do daemon ajustado manualmente de tempos em tempos.
- Fig. 95

Sincronização em redes sem fio

- Redes sem fio, em particular redes sensores: recursos restritos.
- Algoritmos diferentes para sincronização de relógios.
- Uma abordagem: sincronização em broadcast de referência – RBS.
- Não adota como premissa único nó com valor de hora real.
- Visa sincronização interna (não visa UTC).
- Permite somente que receptores sincronizem.

Sincronização em redes sem fio

- Remetente transmite mensagens de referência em broadcast.
- Obs.: tempo de propagação é aproximadamente constante para qualquer nó (sem múltiplos saltos).
- Atraso desconsidera tempo de preparação da mensagem e tempo gasto no adaptador de rede
- Protocolos como NTP adicionam marca de tempo antes de ser passada para a interface de rede
  - Redes sem fio: fatores não-determinísticos como contenção → variabilidade
  - Eliminados em RBS → usa tempo de entrega no receptor

Sincronização em redes sem fio

- Nó transmite mensagem de referência *m.*
- Cada nó *p* registra hora $T_{p,m}$ em que recebeu m.
  - Obtida do relógio local de *p*
- Dois nós *p* e *q* trocam seus respectivos horários de entrega para estimar deslocamento relativo

$$Deslocamento[p,q] = \frac{\sum_{k=1}^{M}(T_{p,k} - T_{q,k})}{M}$$

- onde M é o número de mensagem de referência enviadas

Sincronização em redes sem fio

- Alternativa: regressão linear

$$Deslocamento[p,q](t) = \alpha t + \beta$$

- α e β calculados pelos pares ($T_{p,k}$, $T_{q,k}$)

Relógios lógicos

Relógios lógicos

- Relógios físicos: sincronização relacionada à hora real
  - Flexibilização: não precisa “bater” com a hora real, mas podem concordar com uma hora corrente
- Relógio lógico: o que importa é a ordem de ocorrência dos eventos
- Lamport: sincronização possível, mas não precisa ser absoluta.
  - Processos que não interagem não precisam sincronizar
- Ex.: make → concordar que objeto está desatualizado

Relógios lógicos de Lamport

- Relação “acontece antes”
- a → b indica que **a** acontece antes de **b**
  - Todos os processos concordam que primeiro ocorre **a** e depois **b**.

1. Se **a** e **b** são eventos do mesmo processo, e **a** ocorre antes de **b,** então a → b é verdadeira.
2. Se **a** é o evento de uma mensagem sendo enviada por um processo, e **b** é o evento da mensagem sendo recebida por outro processo, então a → b é verdadeira (pois atraso é > 0).

Relógios lógicos de Lamport

- Transitiva: a→b, b→c implica a→c

- Se dois eventos x e y acontecem em processos diferentes que não trocam mensagens (nem indiretamente), x→y não é verdadeira, nem y→x.
  - x e y: eventos concorrentes – nada pode, e nem precisa, ser dito sobre quando ou qual aconteceu antes.

- Queremos: cada evento **a** tenha um valor de tempo C(a) com o qual todos os processos concordam.

Relógios lógicos de Lamport

- a → b, então C(a) < C(b)
- Em (1) e (2)...

- Tempo de relógio C deve correr para frente
  - Correções por adição de valor, nunca subtração

- Algoritmos de Lamport: designar tempos para eventos.

Relógios lógicos de Lamport

- Considere P1, P2, P3 em máquinas diferentes
- Cada um tem seu próprio relógio
  - P1: 6 pulsos, P2: 8 pulsos, P3: 10 pulsos
  - Relógios a taxas constantes (mas diferentes por causa de diferenças nos cristais).

- Fig. 96

Relógios lógicos de Lamport

- T6: P1 envia m1 a P2; P2 recebe em T16.
- P2 conclui que m1 levou 10 pulsos para chegar.
- m2 de P2 a P3 levou 16 pulsos (na visão de P3).
- m3: sai de P3 em 60 e chega em P2 em 56.
- m4: sai de P2 em 64 e chega em 54.

- Violam “acontece antes”.
  - Como fazer para “desviolar”?

Relógios lógicos de Lamport

- Receptor adianta relógio para ficar em uma unidade a mais do tempo marcado como envio da mensagem.
- m3 chega em 61; m4 chega em 70.
- Fig. 97

Relógios lógicos de Lamport

- Relógio implementado na camada de middleware
- Fig. 98

- Processo mantém um contador local C_i.

1. Antes de executar um evento (enviar msg, entregar msg a uma aplicação, etc), P_i executa $C_i \leftarrow C_i + 1$
2. Quando P_i envia msg *m* a P_j : ts(*m*) $\leftarrow C_i$
3. Ao receber msg *m*, P_j faz $C_j \leftarrow \max\{C_j, ts(m)\}$, executa (1) e entrega mensagem para aplicação.

Relógios lógicos de Lamport

- pode ser desejável que eventos nunca ocorram exatamente ao mesmo tempo
  - Pode-se anexar número do processo ao tempo
  - Tempo 40 em P_i : 40.i

- Resultado: designar tempo C(a) ← C_i(a) ao evento **a** do processo P_i é uma implementação distribuída do valor do tempo global, como desejado.

Ex: Multicast totalmente ordenado

- Banco de dados replicado.
- 2 cópias, mais próxima responde.
- Custo de resposta mais rápida: cada atualização deve ser executada em cada réplica.
  - Mais: atualizações devem ser feitas na mesma ordem nas réplicas.
  - Ex.: Saldo de 1000; adição de juros de 1% em um banco de dados de depósito de 100 em outro.
  - Apesar de ordem fazer diferença no resultado, não faz na consistência: todas as cópias devem ser iguais.
  - Solução: multicast totalmente ordenado

Ex: Multicast totalmente ordenado

- Operação onde todas as mensagens são entregues na mesma ordem a cada receptor.
- Grupo de processos enviam mensagens multicast uns aos outros.
- Mensagens transportam marca de tempo lógico.
- Mensagem enviada também é enviada ao próprio remetente.
- Suposição: mensagens do mesmo remetente são recebidas na ordem que foram enviadas e mensagens não são perdidas.

Ex: Multicast totalmente ordenado

- Processo recebe mensagem: coloca em uma fila local ordenada por marcas de tempo.
- Receptor envia ack em multicast para mensagem recebida.
  - Marca de tempo da mensagem de ack sempre maior que da mensagem original devido ao ajuste de relógios por Lamport
- Resultado: todos os processos terão a mesma cópia da fila local (se nada for removido).

Ex: Multicast totalmente ordenado

- Entrega de msg à aplicação somente quando a msg no início da fila tiver sido reconhecida por todos.
- Filas iguais → mensagens entregues na mesma ordem → multicast totalmente ordenado

- Importante para consistência de réplicas → replicação de estado de máquina

Relógios vetoriais

- Relógios lógicos de Lamport: todos os eventos são totalmente ordenados
  - Se evento **a** aconteceu antes do evento **b**, C(a) < C(b).
- Nada se pode dizer sobre a relação entre dois eventos, **a** e **b**, pela comparação de seus valores de tempo C(a) e C(b).
  - C(a) < C(b) não implica necessariamente que **a** realmente aconteceu antes de **b**.

Relógios vetoriais

- Fig. 99
- $T_{snd}(m_i) < T_{rcv}(m_i)$
- $T_{rcv}(m_i) < T_{snd}(m_j)$?
  - $T_{rcv}(m_1) < T_{snd}(m_3)$
  - $T_{rcv}(m_1) < T_{snd}(m_2)$
  - Envio de m2 não tem nenhuma relação com recebimento de m1 → Não captura **causalidade.**

- Pode ser capturada por relógios vetoriais.

Relógios vetoriais

- Relógio vetorial VC(a) para um evento **a**:
  - Se VC(a) < VC(b) para algum evento **b**, sabe-se que o evento **a** precede por causalidade o evento **b.**
- Relógios vetoriais
  - Cada processo P_i mantém um vetor VC_i tal que:
  1. $VC_i[i]$ é o número de eventos que ocorreram em Pi até o instante em questão. $VC_i[i]$ é o relógio lógico local do processo P_i.
  2. Se $VC_i[j] = k$, então P_i sabe que **k** eventos ocorreram em P_j. Portanto, P_i conhece o tempo local em P_j.

Relógios vetoriais

- Primeira propriedade depende do incremento de $VC_i[i]$ na ocorrência de cada evento no processo P_i.
- Segunda propriedade depende de caronas que os vetores pegam com as mensagens que são enviadas.

Relógios vetoriais

1. Antes de executar um evento (isto é, enviar uma mensagem pela rede, entregar uma mensagem a uma aplicação...), P_i executa $VC_i[i] \leftarrow VC_i[i] + 1$
2. Quando o processo P_i envia uma mensagem **m** a P_j, ele iguala a marca de tempo (vetorial) de **m**, ts(m), à marca de tempo de VC_i, após ter executado a etapa anterior.
3. Ao receber uma mensagem **m** o processo Pj ajusta seu próprio vetor fixando $VC_j[k] \leftarrow \max\{VC_j[k], ts(m)[k]\}$ para cada k; em seguida executa a primeira etapa e entrega a mensagem à aplicação.

Relógios vetoriais

- Se marca de tempo de um evento **a** for ts(a), então ts(a)[i] – 1 é o número de eventos processados em P_i que precedem **a** por causalidade.

- P_j recebe mensagem de P_i com marca de tempo ts(m) → P_j sabe quantos eventos ocorreram em P_i que precedem por causalidade o envio de **m**.

Relógios vetoriais

- P_j também é informado de quantos eventos ocorreram em outros processos antes de P_i enviar a mensagem **m**.
  - Marca de tempo ts(m) informa ao receptor quantos eventos ocorreram em outros processos antes do envio de **m** e dos quais **m** pode depender por causalidade.

Imposição de comunicação causal

- Garantir que uma mensagem seja entregue somente se todas as mensagens que a precederem por causalidade também tenham sido recebidas.
  - Uso de relógios vetoriais + multicast
- Multicast ordenado por causalidade
  - Relação com multicast totalmente ordenado?

Imposição de comunicação causal

- Para duas mensagens não relacionadas:
  - Não importa a ordem que serão entregues às aplicações
  - Podem ser entregues em ordens diferentes para processos diferentes
- Relógios só são ajustados quando enviam e recebem mensagens.
  - Ao enviar mensagem, P_i faz $VC_i[i] \leftarrow VC_i[i] + 1$
  - P_j ao receber mensagem **m** com marca ts(m), ajusta $VC_j[k]$ para $\max\{VC_j[k], ts(m)[k]\}$ para cada k.

Imposição de comunicação causal

- Quando P_j recebe de P_i mensagem **m** com marca de tempo vetorial ts(m), entrega à camada de aplicação será atrasada até que duas condições sejam cumpridas:

1. $ts(m)[i] = VC_j[i] + 1$
   - **m** é a próxima mensagem que P_j está esperando de P_i

2. $ts(m)[k] <= VC_j[k]$ para todo $k != i$
   - P_j viu todas as mensagens que foram vistas por P_i quando este enviou a mensagem **m**.

Imposição de comunicação causal

- Considere P0, P1, P2.
- P0 envia **m** a P1 e P2 no tempo local (1,0,0).
- Após receber **m,** P1 envia **m***, que chega a P2 antes de **m.**
- Entrega de **m*** é atrasada por P2 até que m tenha sido recebida e entregue à camada de aplicação

- Fig. 100

Entrega ordenada de mensagens

- Alguns sistemas de middleware fornecem suporte a multicast totalmente ordenado e multicast ordenado por causalidade.

- Tal suporte deve ser fornecido como parte da camada de comunicação ou aplicações deveriam se encarregar da ordenação?

Entrega ordenada de mensagens

- Middleware não pode dizer o que uma mensagem contém
  - Só pode capturar causalidade potencial.
  - Duas mensagens completamente independentes enviadas pelo mesmo remetente sempre serão marcadas como relacionadas por causalidade pela camada de middleware.
  - Pode resultar em problemas de eficiência
- Nem toda causalidade pode ser capturada
  - Comunicação externa pode implicar causalidade
  - Não capturada pelo middleware.