

MCTA025-13 - SISTEMAS DISTRIBUÍDOS

RELÓGIOS LÓGICOS

Emilio Franceschini

30 de julho de 2018

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC



- Estes slides foram preparados para o curso de **Sistemas Distribuídos na UFABC**.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Estes slides foram adaptados daqueles originalmente preparados (e gentilmente cedidos) pelo professor **Daniel Cordeiro, da EACH-USP** que por sua vez foram baseados naqueles disponibilizados online pelos autores do livro “Distributed Systems”, 3ª Edição em:
<https://www.distributed-systems.net>.

NUM RELÓGIO É QUATRO E VINTE,
NO OUTRO É QUATRO E MEIA,
É QUE DE UM RELÓGIO PRO OUTRO
AS HORAS VAREIA!

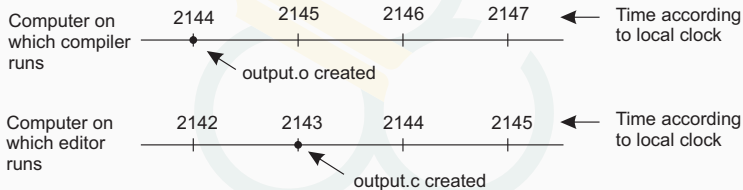
ADONIRAN BARBOSA

- relógios físicos
- relógios lógicos
- relógios vetoriais (Próxima Aula)

SINCRONIZAÇÃO DE RELÓGIOS

- Em sistemas centralizados a definição do horário não é ambígua
- Em sistemas distribuídos isto não é verdade
 - Mesmo em sistemas multi-processador pode não ser verdade

Exemplo de problema: Make



O QUE É UM SEGUNDO?

- **Dia solar** - Tempo entre trânsitos solares
- **Segundo solar** - Dia solar / 86400 ($3600 * 24$)

Problema

- Em 1940 estabeleceu-se que o dia solar não é constante.
- Há 300 milhões de anos o ano tinha 400 dias.

O problema foi resolvido pegando a duração média do dia por um período muito longo.

- Em 1948, com a invenção do relógio atômico passou a ser possível medir o tempo com muito mais precisão
- Os físicos então “tomaram” a definição do segundo dos astrônomos e definiram 1 segundo como o tempo que um átomo de césio 133 leva para fazer exatamente 9.192.631.770 transições
 - Este era o valor exato do segundo solar na época da definição
 - Deste então o segundo solar e O segundo divergiram em mais de 3 milissegundos

Na metade do século passado foram instituídos o **Bureau International de l'Heure (BIH)** e o **Temps Atomique International (TAI)**.

- Todos os países poderiam, então, seguir o tempo seguindo o TAI

Problema

- Como há uma defasagem entre o tempo solar e o tempo atômico em alguns milhares de anos o nascer do sol iria nascer ao meio dia
- Para corrigir essa defasagem estabeleceu-se o **segundo bissexto**
 - Introduzido sempre que a defasagem ultrapassar 800 milissegundos

Diferença entre UT1 e UTC

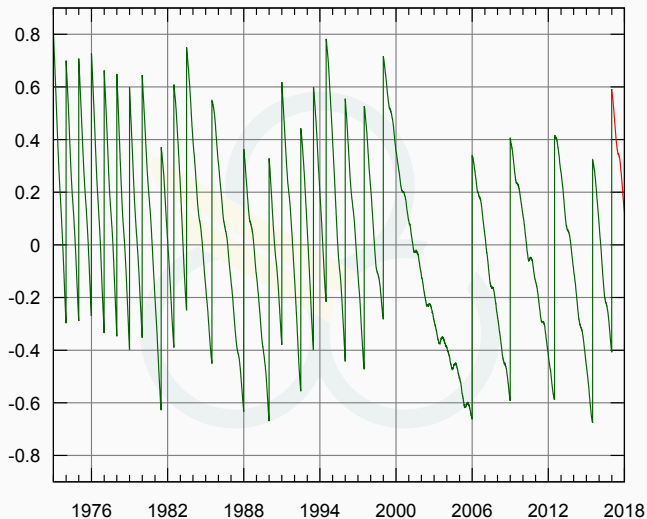


Figura: Wikipedia

Problema

Algumas vezes precisamos saber a hora exata e não apenas uma ordenação de eventos.

Universal Coordinated Time (UTC):

- baseado no número de transições por segundo do átomo de césio 133 (bastante preciso)
- atualmente, o tempo é medido como a média de cerca de 50 relógios de césio espalhados pelo mundo
- introduz um *segundo bissexto* de tempos em tempos para compensar o fato de que os dias estão se tornando maiores

Nota:

O valor do UTC é enviado via *broadcast* por satélite e por ondas curtas de rádio. Satélites tem um acurácia de ± 0.5 ms.

Muitos relógios usam a frequência da rede elétrica (60Hz no Brasil) para manter a hora. Quando há segundos bissextos as companhias mantêm a frequência em 61 Hz por 60 segundos para adiantá-los.

SINCRONIZAÇÃO DE RELÓGIOS

Precisão

O objetivo é tentar fazer com que o desvio **entre dois relógios em quaisquer duas máquinas** fique dentro de um limite especificado, conhecido como a **precisão** π :

$$\forall t, \forall p, q : |C_p(t) - C_q(t)| \leq \pi$$

onde $C_p(t)$ é o horário do relógio **computado** para a máquina p no **horário UTC** t .

Acurácia

No caso da **acurácia**, queremos manter o relógio limitado a um valor α :

$$\forall t, \forall p : |C_p(t) - t| \leq \alpha$$

Sincronização

Sincronização interna: manter a **precisão** dos relógios

Sincronização externa: manter a **acurácia** dos relógios

Especificação dos relógios

- Todo relógio tem especificado sua taxa máxima de desvio do relógio ρ .
- $F(t)$: frequência do oscilador do relógio do hardware no tempo t
- F : frequência (constante) do relógio ideal:

$$\forall t : (1 - \rho) \leq \frac{F(t)}{F} \leq (1 + \rho)$$

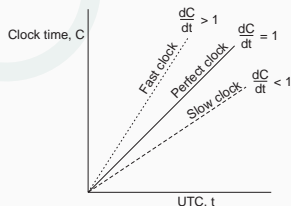
Observação

Interrupções de hardware acoplam um relógio de software a um relógio de hardware, que também tem sua taxa de desvio:

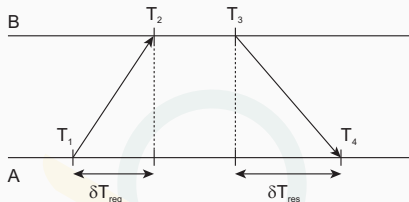
$$C_p(t) = \frac{1}{F} \int_0^t F(t) dt \Rightarrow \frac{dC_p(t)}{dt} = \frac{F(t)}{F}$$

$$\Rightarrow \forall t : 1 - \rho \leq \frac{dC_p(t)}{dt} \leq 1 + \rho$$

Relógios rápidos, perfeitos e lentos



Recuperação do horário atual de um servidor



Cálculo da diferença relativa θ e o atraso δ

Assumindo que: $\delta T_{req} = T_2 - T_1 \approx T_4 - T_3 = \delta T_{res}$

$$\theta = T_3 + ((T_2 - T_1) + (T_4 - T_3))/2 - T_4 = ((T_2 - T_1) + (T_3 - T_4))/2$$

$$\delta = ((T_4 - T_1) - (T_3 - T_2))/2$$

Network Time Protocol

Colete oito pares (θ, δ) e escolha os θ cujos atrasos δ são minimais.

Sincronização externa

Cada máquina pede a um *servidor de hora* a hora certa pelo menos uma vez a cada $\delta/(2\rho)$ (Network Time Protocol)

OK, mas...

you still need a way to measure the *round trip delay*, including the handling of the interruption and the processing of the messages.

Sincronização interna

Permita o servidor de hora sonde todas as máquinas periodicamente, calcule uma média e informe cada máquina como ela deve ajustar o seu horário **relativo ao seu horário atual**.

Nota:

Você provavelmente terá todas as máquinas em sincronia. Você nem precisa propagar o horário UTC.

É fundamental

saber que atrasar o relógio **nunca** é permitido. Você deve fazer ajustes suaves.

RELÓGIO QUE ATRASA NÃO ADIANTA.

DESCONHECIDO

RELÓGIOS LÓGICOS

O que importa na maior parte dos sistemas distribuídos não é fazer com que todos os processos concordem exatamente com o horário, mas sim fazer com que eles concordem com **a ordem em que os eventos ocorreram**. Ou seja, precisamos de uma noção de ordem entre os eventos.

A RELAÇÃO “ACONTECEU-ANTES”

A relação “aconteceu-antes” (*happened-before*)

- se a e b são dois eventos de um mesmo processo e a ocorreu antes de b , então $a \rightarrow b$



A relação “aconteceu-antes” (*happened-before*)

- se a e b são dois eventos de um mesmo processo e a ocorreu antes de b , então $a \rightarrow b$
- se a for o evento de envio de uma mensagem e b for o evento de recebimento desta mesma mensagem, então $a \rightarrow b$



A RELAÇÃO “ACONTECEU-ANTES”

A relação “aconteceu-antes” (*happened-before*)

- se a e b são dois eventos de um mesmo processo e a ocorreu antes de b , então $a \rightarrow b$
- se a for o evento de envio de uma mensagem e b for o evento de recebimento desta mesma mensagem, então $a \rightarrow b$
- se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$

A RELAÇÃO “ACONTECEU-ANTES”

A relação “aconteceu-antes” (*happened-before*)

- se a e b são dois eventos de um mesmo processo e a ocorreu antes de b , então $a \rightarrow b$
- se a for o evento de envio de uma mensagem e b for o evento de recebimento desta mesma mensagem, então $a \rightarrow b$
- se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$

A RELAÇÃO “ACONTECEU-ANTES”

A relação “aconteceu-antes” (*happened-before*)

- se a e b são dois eventos de um mesmo processo e a ocorreu antes de b , então $a \rightarrow b$
- se a for o evento de envio de uma mensagem e b for o evento de recebimento desta mesma mensagem, então $a \rightarrow b$
- se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$

Eventos concorrentes

Se dois eventos x e y ocorrem em processos distintos e esses processos nunca interagem (mesmo que indiretamente) então nem $x \rightarrow y$ nem $y \rightarrow x$ são verdade. São eventos **concorrentes**. Quando se diz que dois eventos são concorrentes na verdade quer dizer que *nada pode (ou precisa) ser dito sobre a sua ordem*.

A RELAÇÃO “aconteceu-antes”

A relação “aconteceu-antes” (*happened-before*)

- se a e b são dois eventos de um mesmo processo e a ocorreu antes de b , então $a \rightarrow b$
- se a for o evento de envio de uma mensagem e b for o evento de recebimento desta mesma mensagem, então $a \rightarrow b$
- se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$

Eventos concorrentes

Se dois eventos x e y ocorrem em processos distintos e esses processos nunca interagem (mesmo que indiretamente) então nem $x \rightarrow y$ nem $y \rightarrow x$ são verdade. São eventos **concorrentes**. Quando se diz que dois eventos são concorrentes na verdade quer dizer que *nada pode (ou precisa) ser dito sobre a sua ordem*.

Nota:

Isso introduz uma noção de **ordem parcial dos eventos** em um sistema com processos executando concorrentemente.

Problema

Como fazemos para manter uma visão global do comportamento do sistema que seja consistente com a relação aconteceu-antes?



Problema

Como fazemos para manter uma visão global do comportamento do sistema que seja consistente com a relação aconteceu-antes?

Solução

Associar um *timestamp* $C(e)$ a cada evento e tal que:

- P1 se a e b são dois eventos no mesmo processo e $a \rightarrow b$, então é obrigatório que $C(a) < C(b)$
- P2 se a corresponder ao envio de uma mensagem m e b ao recebimento desta mensagem, então também é válido que $C(a) < C(b)$

Problema

Como fazemos para manter uma visão global do comportamento do sistema que seja consistente com a relação aconteceu-antes?

Solução

Associar um *timestamp* $C(e)$ a cada evento e tal que:

- P1 se a e b são dois eventos no mesmo processo e $a \rightarrow b$, então é obrigatório que $C(a) < C(b)$
- P2 se a corresponder ao envio de uma mensagem m e b ao recebimento desta mensagem, então também é válido que $C(a) < C(b)$

Outro problema

Como associar um *timestamp* a um evento quando não há um relógio global? Solução: manter um conjunto de relógios lógicos **consistentes**, um para cada processo

Solução

Cada processo P_i mantém um contador C_i **local** e o ajusta de acordo com as seguintes regras:

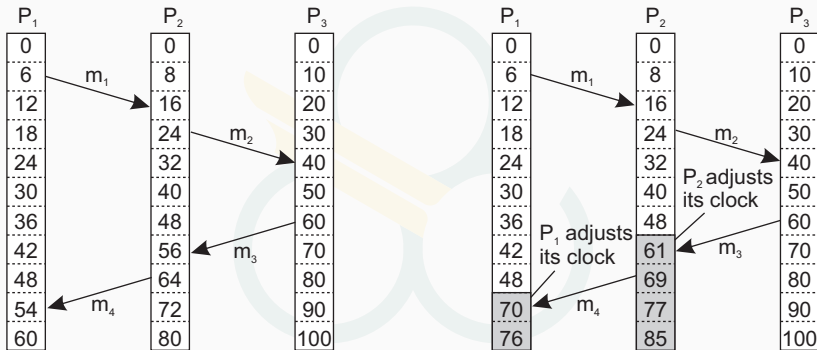
1. para quaisquer dois **eventos sucessivos** que ocorrer em P_i , C_i é incrementado em 1
2. toda vez que uma mensagem m for **enviada** por um processo P_i , a mensagem deve receber um *timestamp* $ts(m) = C_i$
3. sempre que uma mensagem m for **recebida** por um processo P_j , P_j ajustará seu contador local C_j para **$\max\{C_j, ts(m)\}$** e executará o passo 1 antes de repassar m para a aplicação

Observações:

- a propriedade **P1** é satisfeita por (1); propriedade **P2** por (2) e (3)
- ainda assim pode acontecer de dois eventos ocorrerem ao mesmo tempo. **Desempate usando os IDs dos processos.**

RELÓGIO LÓGICO DE LAMPORT – EXEMPLO

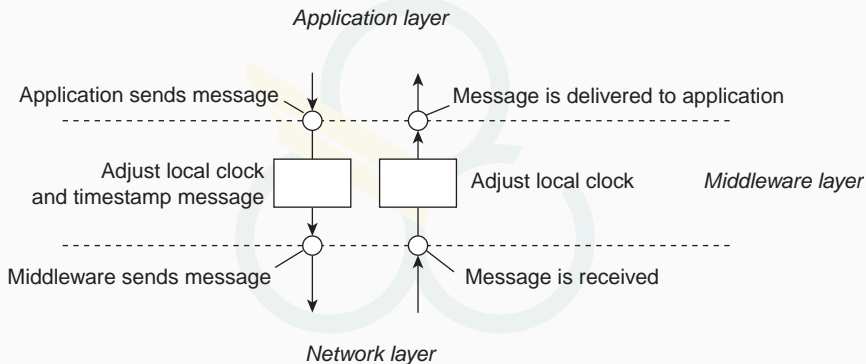
Considere três processos com **contadores de eventos** funcionando a velocidades diferentes.



RELÓGIO LÓGICO DE LAMPORT – EXEMPLO

Nota

Os ajustes ocorrem na camada do *middleware*

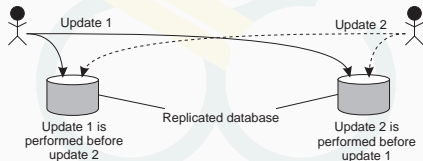


EXEMPLO: MULTICAST COM ORDEM TOTAL

Problema

Alguma vezes precisamos garantir que atualizações concorrentes em um banco de dados replicado sejam vistos por todos como se tivessem ocorrido na mesma ordem.

- P_1 adiciona R\$ 100 a uma conta (valor inicial: R\$ 1000)
- P_2 incrementa a conta em 1%
- Há duas réplicas



Resultado

Na ausência de sincronização correta,
réplica #1 \leftarrow R\$ 1111, enquanto que na réplica #2 \leftarrow R\$ 1110.

EXEMPLO: MULTICAST COM ORDEM TOTAL

Solução

- processo P_i envia uma **mensagem com timestamp** m_i para todos os outros. A mensagem é colocada em sua fila local $queue_i$.
- toda mensagem que chegar em P_j é colocada na fila $queue_j$ **priorizada pelo seu timestamp** e **confirmada** (*acknowledged*) por todos os outros processos

P_j repassa a mensagem m_i para a sua aplicação somente se:

- (1) m_i estiver na cabeça da fila $queue_j$
- (2) para todo processo P_k , existe uma mensagem m_k na $queue_j$ com um *timestamp* maior.

Nota

Assumimos que a comunicação é **confiável** e que a **ordem FIFO** (entre as mensagens enviadas por um nó, e não na fila) é respeitada.

O ALGORITMO DE MULTICAST FUNCIONA?

Observe que:

- se uma mensagem m ficar pronta em um servidor S , m foi recebido por todos os outros servidores (que enviaram ACKs dizendo que m foi recebido)
- se n é uma mensagem originada no mesmo lugar que m e for enviada antes de m , então todos receberão n antes de m e n ficará no topo da fila antes de m
- se n for originada em outro lugar, é um pouco mais complicado. Pode ser que m e n cheguem em ordem diferente nos servidores, mas é certa de que antes de tirar um deles da fila, ele terá que receber os ACKs de todos os outros servidores, o que permitirá comparar os valores dos relógios e entregar para as mensagens na ordem total dos relógios