Resumo P2 - IPPD

link:
https://sdpisutic.files.wordpress.com/2017/08/sistemas-distribuc3ados-conceitos-e-projeto-2013.pdf

Sistema distribuído foi definido como aquele no qual os componentes de hardware ou software localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens.

Modelos de Sistema:

- **modelos físicos**: são a maneira mais explícita de descrever um sistema; a composição de **hardware** de um sistema, em termos dos **computadores** (e **outros equipamentos, incluindo os móveis**) e suas **redes** de interconexão. Abstraem os detalhes específicos do computador e das tecnologias de rede empregadas
- **modelos de arquitetura**: descrevem um sistema em termos das **tarefas computacionais e de comunicação** realizadas por seus elementos computacionais – os computadores individuais ou seus agregados (clusters) suportados pelas interconexões de rede apropriadas.
- **modelos fundamentais**: examinar os aspectos individuais de um sistema distribuído:
    - modelos de interação, que consideram a estrutura e a ordenação da comunicação entre os elementos do sistema;
    - modelos de falha, que consideram as maneiras pelas quais um sistema pode deixar de funcionar corretamente;
    - modelos de segurança, que consideram como o sistema está protegido contra tentativas de interferência em seu funcionamento correto ou de roubo de seus dados;

Modelos de arquitetura:

É a estrutura em termos de componentes especificados separadamente e suas inter-relações. As maiores preocupações são tornar o sistema confiável, gerenciável, adaptável e rentável.

- **Cliente-servidor:** os processos clientes interagem com processos servidores, localizados possivelmente em distintos computadores hospedeiros, para acessar os recursos compartilhados que estes gerenciam. Os servidores podem, por sua vez, ser clientes de outros servidores(ex: serviço DNS, que mapeia nomes de domínio Internet a endereços de rede (IP)).

- **Peer-to-peer:** os processos envolvidos em uma tarefa desempenham funções semelhantes, interagindo cooperativamente como pares, sem diferença entre processos clientes e servidores, nem entre os computadores em que são executados. Em termos práticos, todos os processos participantes executam o mesmo programa e oferecem o mesmo conjunto de interfaces uns para os outros. Vantagem: os recursos disponíveis para executar o serviço aumentam com o número de usuários.

O problema da alocação dos serviços:

Mapeamento dos serviços sobre os recursos físicos. Deve considerar as características da aplicação: Padrão de comunicação, Confiabilidade, qualidade de comunicação, carga do nó.

- Estratégias básicas:
  - **Múltiplos servidores**: Organizar acesso de clientes a múltiplos servidores. Os serviços podem ser replicados entre os servidores ou cada um fazer algumas tarefas.
  - **Caching:** realizar um armazenamento de objetos de dados recentemente usados em um local mais próximo dos clientes, do que a origem real dos objetos em si;
  - **Código móvel:** o usuário, executando um navegador, seleciona um link que aponta para um applet, cujo código é armazenado em um servidor Web; o código é carregado no navegador. Vantagem de executar um código localmente é que ele pode dar uma boa resposta interativa, pois não sofre os atrasos nem a variação da largura de banda associada à comunicação na rede.
  - **Agentes móveis**: um agente móvel é um programa em execução (inclui código e dados) que passa de um computador para outro em um ambiente de rede, realizando uma tarefa em nome de alguém, como uma coleta de informações, e finalmente retornando com os resultados obtidos a esse alguém.

Padrões Arquiteturais:

Os padrões arquitetônicos baseiam-se nos elementos de arquitetura mais primitivos,

- **Camadas lógicas(múltiplas camadas):** um sistema complexo é particionado em várias camadas, com cada uma utilizando os serviços oferecidos pela camada lógica inferior.

Aplicação
Middleware
S.O.
Hardware(nós e rede)
Plataforma

- **camada física(ou tiered):** as camadas físicas representam uma técnica para organizar a funcionalidade de uma camada lógica e colocar essa funcionalidade nos servidores apropriados e, como uma consideração secundária, nos nós físicos.
- **Thin Client:** camada de software que suporta uma interface baseada em janelas que é local para o usuário, enquanto executa programas aplicativos ou, mais geralmente, acessa serviços em um computador remoto.
- **Brokerage:** oferece uma abstração para acesso a um conjunto de serviços complexos fornecidos por dois ou mais servidores.
- **Proxy:** um proxy é criado no espaço local para representar o objeto remoto. Esse proxy oferece exatamente a mesma interface do objeto remoto.
- **Reflexão:** reflete sobre a estrutura do sistema e/ou reflete para a localização desejada, promovendo dinamicidade na estrutura.

Resumo cap 14

Sincronização de relógios físicos:

Método de Cristian: Sincroniza o relógio utilizando um servidor com um relógio atômico.

Cliente manda mensagem pedindo a hora-> Servidor recebe a mensagem e envia a hora como mensagem-> Cliente recebe a mensagem e atualiza seu relógio utilizando:

Relógio = Tempo do relógio na mensagem + Tempo de transmissão das mensagens

É um algoritmo probabilístico.

Berkeley: Baseado em uma arquitetura **mestre escravos**
1- o mestre de tempos em tempos pede as horas para os escravos
2- o mestre faz a **média desses relógios**;
3- o mestre envia a média menos o valor original do escravo;
4 - o escravo atualiza seu relogio.

Para o calculo da média são utilizados apenas as horas dentro de um desvio padrao.

NTP: O **NTP (Network Time Protocol)** [Mills 1995] define uma arquitetura para um serviço de **tempo** e um protocolo para distribuir informações de tempo pela **Internet**.

É UMA ARVORE DE SINCRONIZAÇÃO, ONDE A RAIZ É O MAIS SINCRONIZADO.

NTP **são sincronizados entre si de três maneiras: multicast, chamada de procedimento e modo simétrico**.

Periodicamente, um ou mais servidores de tempo enviam em **multicast** a informação de tempo para servidores que estão em execução em outros computadores conectados na rede local, os quais **configuram seus relógios pressupondo um pequeno atraso**. Esse modo pode obter apenas uma precisão relativamente baixa, mas considerada suficiente para muitos propósitos.

O modo de **chamada de procedimento** é **semelhante** ao funcionamento do **algoritmo de Cristian**, descrito anteriormente. Nesse modo, um **servidor aceita requisições de outros computadores, os quais ele processa respondendo com seu carimbo de tempo** (leitura corrente do relógio). Esse modo é conveniente quando é exigida uma precisão melhor do que a que pode ser obtida com o multicast – ou quando multicast não é suportado por hardware.

Dois servidores operando no **modo simétrico** trocam mensagens com informações de temporização. Esses dados são armazenados como parte de uma associação entre os servidores que são mantidos para **melhorar a precisão de sua sincronização com o passar do tempo.**

Sincronização de relógios logicos:

Lamport: contador de software que aumenta a contagem monotonicamente e cujo valor não precisa ter nenhum relacionamento em particular com qualquer relógio físico.

Cada processo pi mantém seu próprio **relógio lógico, Li** , que utiliza para aplicar os conhecidos carimbos de tempo de Lamport nos eventos. Denotamos o carimbo de tempo do evento e em pi como Li (e) e com L(e), o carimbo de tempo do evento e no processo em que ela ocorreu

Para capturar a relação acontece antes →, os processos atualizam seus relógios lógicos e transmitem os valores de seus relógios lógicos em mensagens, como segue:
RL1: Li é incrementado antes da ocorrência de um evento no processo pi : Li := Li + 1
RL2:
(a) Quando um processo pi envia uma mensagem m, m leva "de carona" (piggybacking) o valor t = Li .
(b) Na recepção (m, t), um processo pj calcula Lj : max(Lj , t) e, então, aplica RL1 antes de registrar o carimbo de tempo do evento receive(m).

Os **relógios vetoriais** resolvem esse problema fazendo com que cada processo mantenha um array de contadores, sendo cada posição do array, um contador para cada processo no sistema.

Resumo cap 15

- Algoritmos onde um conjunto de processos coordene suas ações ou concorde com um ou mais valores.
- O motivo para se evitar relacionamentos mestre-escravo fixos é que, frequentemente, exigimos que nossos sistemas continuem funcionando corretamente, mesmo que ocorram falhas; portanto, precisamos evitar pontos únicos de falha, como os mestres fixos

Exclusão Mútua distribuída: um conjunto de processos compartilha um recurso, ou uma coleção de recursos, então, frequentemente, a exclusão mútua é exigida

para evitar interferência e garantir a consistência ao acessar esses recursos. Baseada unicamente na passagem de mensagens.

O algoritmo do servidor central • Para entrar em uma seção crítica, um processo envia uma mensagem de requisição para o servidor e espera uma resposta. Conceitualmente, a resposta constitui uma ficha (token) significando permissão para entrar na seção crítica. Se nenhum outro processo tiver a ficha no momento da requisição, então o servidor responderá imediatamente, concedendo a ficha. Se a ficha estiver de posse de outro processo, então o servidor não responderá, mas enfileirará a requisição. Na saída da seção crítica, uma mensagem é enviada para o servidor, devolvendo a ficha a ele.

Um algoritmo usando multicast e relógios lógicos(Ricart e Agrawala) - Os processos p1, p2,..., pN apresentam identificadores numéricos. As mensagens que solicitam entrada são da forma **<T,pi>** onde T é o carimbo de tempo do remetente e pi é o identificador do remetente. Cada processo registra seu estado de estar fora da seção crítica (RELEASED), querendo entrar (WANTED) ou estar na seção crítica (HELD).

- caso 1: Se um processo solicita entrada, e o estado dos outros processos é RELEASED, então todos responderão imediatamente a requisição e o solicitante obterá a entrada.
- caso 2: Se algum processo estiver no estado HELD, então esse processo não responderá as requisições até que tenha terminado com a seção crítica; portanto, o solicitante não poderá entrar nesse meio tempo.
- caso 3: Se dois ou mais processos solicitam a entrada ao mesmo tempo, a requisição do processo que apresentar o carimbo de tempo mais baixo será o primeiro a coletar N – 1 respostas, garantindo a próxima entrada.
- caso 4: Se as requisições apresentarem carimbo de tempo de Lamport iguais, serão ordenados de acordo com os identificadores correspondentes dos processos.

A obtenção da entrada exige 2(N – 1) mensagens nesse algoritmo: N – 1 para difundir a requisição por multicast, seguido de N – 1 respostas.

Algoritmo de votação de Maekawa Os processos só precisam obter permissão de subconjuntos de seus pares para entrar, desde que os subconjuntos usados por quaisquer dois processos se sobreponham. EXPLICAÇÃO MELHOR NAS RESPOSTAS ABAIXO.

A utilização de largura de banda do algoritmo é de 2(SQRT(N)) mensagens por entrada na seção crítica e de SQRT(N) mensagens por saída (supondo que não exista nenhum recurso de hardware para multicast). O total de 3(SQRT(N)) é superior às 2(N − 1) mensagens exigidas pelo algoritmo de Ricart e Agrawala, se N > 4.

Perguntas de prova antiga:

1 - Em que situação um deadlock pode acontecer no algoritmo Ricart e Agrawala e como solucionar?

One of the problems in this algorithm is failure of a node. In such a situation a process may starve forever. This problem can be solved by detecting failure of nodes after some timeout. https://en.wikipedia.org/wiki/Ricart%E2%80%93Agrawala_algorithm

2 - Em que situação um deadlock pode acontecer no algoritmo Maekawa e como solucionar? https://en.wikipedia.org/wiki/Maekawa%27s_algorithm

Infelizmente, o algoritmo é propenso a impasses. Considere três processos p_1, p_2 e p_3 com $V_1 = \{p_1, p_2\}$, $V_2 = \{p_2, p_3\}$ e $V_3 = \{p_3, p_1\}$. Se os três processos solicitam a entrada na seção crítica, então é possível que p_1 responda para si mesmo e detenha p_2, que p_2 responda para si mesmo e detenha p_3 e que p_3 responda para si mesmo e detenha p_1. Cada processo recebeu uma de duas respostas e nenhum pode prosseguir.

O algoritmo pode ser adaptado [Sanders 1987] de modo que se torne livre de impasses. No protocolo adaptado, os processos enfileiram as requisições pendentes na ordem acontece antes, de modo que o requisito EM3 também é satisfeito.

Para resolver pode ser utilizados os timestamps para decidir a prioridade.

3 - Pergunta sobre o método Christian de sincronização de relogios.

14.3.2 Método de Cristian para sincronização de relógios

Cristian [1989] sugeriu o uso de um servidor de tempo, conectado a um dispositivo que recebe sinais de uma fonte UTC, para sincronizar computadores externamente. Ao receber uma requisição, o processo servidor S fornece o tempo, de acordo com seu relógio,

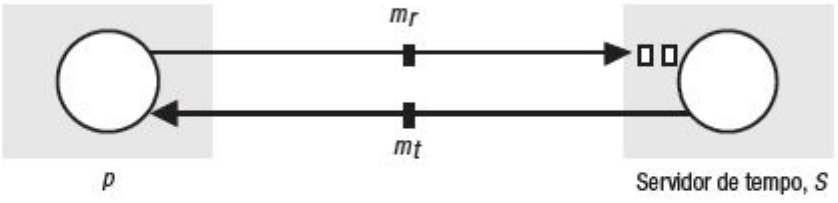


Figura 14.2 Sincronização de relógio usando um servidor de tempo.

como mostrado Figura 14.2. Cristian observou que, embora não haja um limite superior para os atrasos na transmissão de mensagens em um sistema assíncrono, frequentemente os tempos de viagem de ida e volta para mensagens trocadas entre pares de processos são razoavelmente curtos – uma pequena fração de segundo. Ele descreve o algoritmo como *probabilístico*: o método só obtém sincronização se os tempos de viagem de ida e volta observados entre cliente e servidor forem suficientemente curtos, comparados com a precisão exigida.

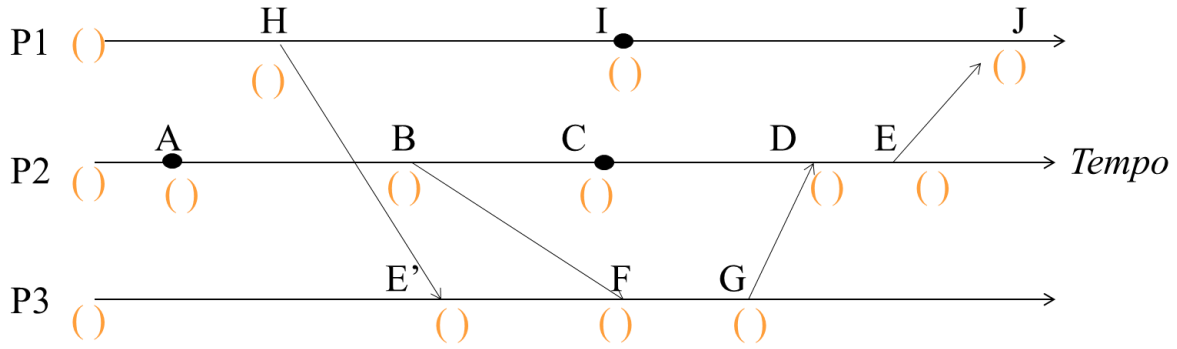
4 - Determinar os tempos lógicos e os eventos concorrentes usando Vector timestamps e Lamport timestamps.

1. A process increments its counter before each event in that process;
2. When a process sends a message, it includes its counter value with the message;
3. On receiving a message, the counter of the recipient is updated, if necessary, to the greater of its current counter and the timestamp in the received message. The counter is then incremented by 1 before the message is considered received.

5 - Pergunta sobre tolerância a falhas bizantinas
Se o número de traídores for **menor** que **1 / 3**, o que foi decidido pelos 2 / 3 é assumido como verdade (N >= 3f + 1). A comunicação deve ser **síncrona**. **Assinatura digital** pode resolver o problema de traídores, utilizando a criptografia de chave pública (igual a RSA). **Não** é bom para **erros** mas é **bom** para **segurança**. Bitcoin resolve esse problema com blockchain (vá que ele queira exemplo).
Fischer et al. [1985] provaram que **nenhum algoritmo pode garantir um consenso em um sistema assíncrono**, mesmo com a falha por colapso de um único processo. Para **contornar** existem três técnicas: **mascaramento de falhas**, **detectores de falha** e usando a **aleatoriedade.**

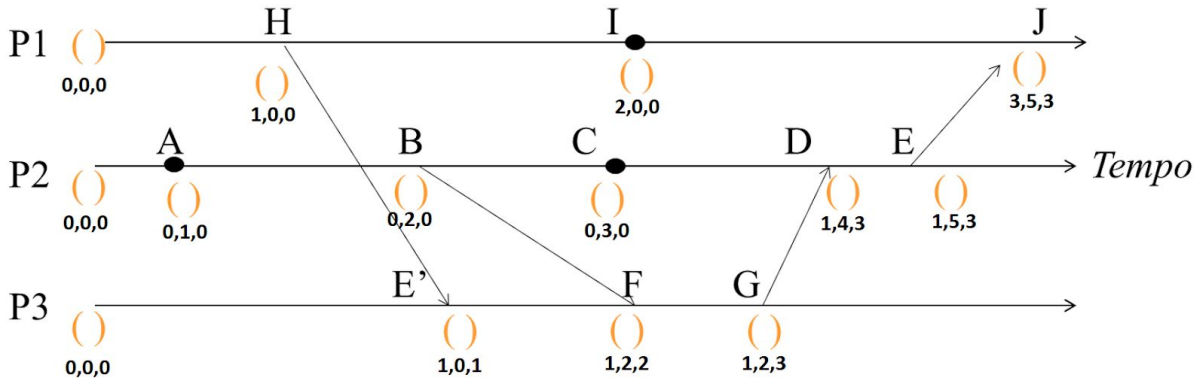
Questoes prova 2:

1 - Dado os 3 processos do desenho coloque entre os parênteses os tempos de Lamport e Vetoriais. Diga quais são os eventos concorrentes e justifique.



Lamport:
A - 1
H - 1
E' - 2
B - 2

F - 3
C - 3
I - 2
G - 4
D - 5
E - 6
J - 7



Concorrente é quando o seu elemento é maior e a do outro é menor.
I || E' = (2,0,0) || (1,0,1) **2 > 1** e (2,0,0) || (1,0,1) **0 < 1**
Vetoriais:
A - (0,1,0)
H - (1,0,0)
E' - (1,0,1)
B - (0,2,0)
F - (1,2,2)
C - (0,3,0)
I - (2,0,0)
G - (1,2,3)
D - (1,4,3)
E - (1,5,3)
J - (3,5,3)

Concorrência:
H || A, C || I, I || G (nao sei se ta correto)

2 - No algoritmo do Valentão qual o número máximo de mensagens numa eleição? Justifique.

O algoritmo valentão exige O(N^2) mensagens, no pior caso – isto é, quando o processo com o menor identificador detecta primeiro a falha do coordenador.

3 - Explique o multicast confiável e justifique o confiável.

O multicast confiável é aquele que satisfaz as seguintes propriedades:

Integridade: um processo correto p entrega uma mensagem m no máximo uma vez. Além disso, p ∈ group(m) e m foi fornecida para uma operação multicast por sender(m). (Assim como acontece com a comunicação de um para um, as mensagens sempre podem ser diferenciadas por um número de sequência relativo aos seus remetentes.)

Validade: se um processo correto executa um multicast da mensagem m, então, ele distribuirá m.

Acordo: se um processo correto entrega a mensagem m, então todos os outros processos corretos em group(m) distribuirão m.

4 - Explique o que é o problema dos generais bizantinos e quais suas aplicações.

O problema dos generais bizantinos difere do consenso porque um processo distinto fornece um valor com o qual os outros devem concordar, em vez de cada um deles propor um valor. Os requisitos são:

Término: cada processo correto acaba por configurar sua variável de decisão.

Acordo: o valor de decisão de todos os processos corretos é o mesmo: se pi e pj estão corretos e entraram no estado decidido, então di = dj (i, j = 1, 2,..., N).

Integridade: se o comandante está correto, então todos os processos corretos decidem pelo valor proposto pelo comandante.

Note que, para o problema dos generais bizantinos, integridade implica em acordo quando o comandante está correto; mas o comandante não precisa estar correto. Aplica-se quando ao fato de os processos concordarem com um valor após um, ou mais, dos processos terem proposto qual deve ser esse valor.