Sistemas Distruídos

Terceira lista de Exercícios

Para ler:

1. Slide 10 a 13 aula 17.
2. Aula 19.
3.

Questão 1: Cite e explique dois desafios que precisam ser resolvidos na implementação de RPC.

1. Representação de dados possivelmente diferentes. (Little Endian, Big Endian). A representação de dados diferentes pode acarretar uma codificação diferente em cada maquina, podendo levar a erros na execução ou diferenciacão dos dados esperados.
2. Processos que chama função e que executa funcao estao em maquinas diferentes. Dessa forma, o espaco de enderecamento é diferente, alem do sistema operacional e do hardware.
3. Redes e máquinas podem falhar durante a chamada de função.

Questão 2: Explique todos os passos envolvidos em uma chamada RPC assíncrona.

1. O cliente faz uma chamada de um procedimento remoto e aguarda a resposta do servidor(espera bloqueante).
2. O pedido chega ao servidor, que envia uma mensagem de aceitação de pedido.
3. O cliente recebe a confirmação do servidor e continua o processamento normalmente.
4. O servidor faz o processamento correspondente a solicitação, quando termina, retorna os resultados para o cliente.
5. A execução do cliente é interrompida com a chegada da mensagem do servidor. O cliente uma mensagem de confirmação de recebimento dos resultados.

Questão 3: Qual o problema inerente a sistemas distribuidos que relógios sincronizados ajudam a resolver ? Dê um exemplo do problema.

O problema que a sincronização de relógios ajuda a resolver é sincronização de eventos. Idealmente, a sincronização de relogio ajudaria a descobrir a ordem cronologica dos eventos. Um exemplo bem comum é a edição de um arquivo

por duas pessoas em computadores diferentes. A primeira pessoa edita um arquivo e salva no Dropbox, o seu relogio local está adiantado, dessa forma o Dropbox reconhece como ultima edição e salva as alterações. A segunda pessoa edita depois da primeira, porém sua edição não será salva pois seu relógio local esta atrasado e o Dropbox reconhece que seu evento de edição ocorreu antes da primeira pessoa.

Questão 4: Considere o problema de manter a hora sincronizada entre dois relógios. Cite e explique os dois aspecto fundamentais que dificultam a sincronização.

A primeira grande dificuldade na sincronização de relógio é conhecido como *Clock Drift*. Esse problema está relacionado a oscilação do relógio, nenhum oscilador é fundamentalmente igual ao outro, dessa forma, as frenquências de oscilação são ligeiramente diferentes(mesmo que sejam de mesmo material, fatores externos como temperatura pode alterar a frenquência de oscilação) os pode acarretar em uma diferenciação de tempo. O segundo problema é como ajustar a hora do computador de forma que os todos os computadores envolvidos marquem a mesma hora. É necessário que haja uma sincronização entre eles e que o método usado consiga lidar com atrasos.

Questão 5: Explique como funciona o mecanismo básico (visto em aula) para sincronização de relógios quando um computador deseja usar o relógio de outro como referência. Utilize os horários dos relógios e mostre como o relógio deve ser acertado.

O Algoritmo utilizado para sincronização de relógios para máquinas na mesma rede local consiste na utilização de um servidor para a aplicação do algoritmo.

1. Inicialmente o servidor informa sua hora para máquinas clientes.
2. As máquinas clientes por sua vez, respondem com a diferença com seus relógios.
3. O servidor faz a média dos valores recebidos (remove *outleirs*).
4. Servidor transmite a cada cliente ajuste necessário.
5. Clientes fazem ajustes em seus relógios.

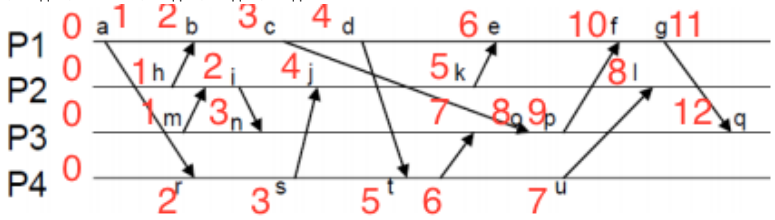
Questão 6: Explique sucintamente como funciona o Network Time Protocol (NTP).

Network Time Protocol é um serviço de sincronização de relógios para a Internet, utilizando UTC como referência. Ele é composto por uma hierarquia de servidores e modelado como cliente/servidor. Funcionamento:

1. O cliente periodicamente solicita hora a três ou mais servidores.
2. O cliente faz uma estimativa de tempo em relação ao tempo de solicitação e o tempo de resposta.

3. O cliente faz a media entre as três horas estimadas anteriormente.
4. **A hora não é setada diretamente**, ele utiliza a média das horas para atualizar a taxa de progressão do seu relógio de forma que sempre haja uma progressão da hora e nunca "volte no tempo".

Questão 7: Considere a figura abaixo com quatro processos e alguns eventos: Assumindo que os valores de relógio lógico inicialmente são zero:

1. (a e r), (h e b).
2. (a e m), (h e m).
3. b||k, b||n, b→u, k||n, k||u, n||u
4. 

5. ~
6. ~
7. ~
8. ~

Questão 8: Considere a figura acima com três processos e alguns eventos indicados: Utilize o algoritmo para ordenação total de eventos (globally ordered multicast) para definir a ordem em que os eventos indicados serão processados. Mostre o progresso do algoritmo, indicando como suas filas locais mudam com as mensagens e eventos.

Definição dos passos do algoritmo :

1. L(a) = 1.1, L(b) = 1.3 .
2. Inicio da fila em P1:[a], P2:[] e P3:[b].
3. Depois de a: P1:[a], P2:[a] e P3:[a,b].
4. Depois de b: P1:[a,b], P2:[a,b] e P3:[a,b].
5. Depois de c: P1:[a,b,c], P2:[a,b,c] e P3:[a,b,c].
6. Depois de e: P1:[a,b,c,e], P2:[a,b,c,e] e P3:[a,b,c,e].
7. Depois de d: P1:[a,b,c,e,d], P2:[a,b,c,e,d] e P3:[a,b,c,e,d].

Questão 9: Cite e explique uma vantagem e uma desvantagem do algoritmo de exclusão mútua centralizado.

O algoritmo centralizado de exclusão mútua, tem como principio um coordenador que será responsável por coordenar acesso a região crítica, utilizando mensagens para a comunicação e fila para armazenar os pedidos dos processos. Temos também os processos que fazem a solicitação para entrada na região critica e quando saem, avisam ao coordenador. Dessa maneira temos como :

- Vantagem: O número de mensagens para cada pedido de acesso é pequeno(apenas 3: *request*, *grant* e *realease*). Dessa maneira, torna o processo mais eficiente.
- Desvantagem: Ponto único de falha, visto que se o coordenador falhar, não haverá mais coordenação para acesso a região crítica.

Questão 10: No algoritmo de exclusão mútua em anel, qual a vantagem de um nó conhecer seus dois próximos vizinhos no anel, ao invés de apenas o próximo? Seria vantajoso conhecer mais de dois vizinhos?

A vantagem de um nó conhecer seus dois próximos vizinhos no anel é que se um de seus vizinho falhar ou desconectar, ainda é possível conectar-se no anel a partir do outro vizinho. Conhecer mais dois vizinhos seria vantajoso, visto que se mais de dois nós falharem, ainda é possivel reconectar.

Questão 11: Em um sistema transacional, o que é ACID? Explique também seu significado.

Em sistemas transacionais, ACID é um conjunto de propriedades que o sistema deve ter para garantir a corretude e facilitar o seu uso. Cada letra representa uma propriedade:

1. A: ***Atomicity***. A transação completa por inteiro ou é abortada por inteiro. Em caso de aborto, o estado global não é modificado.
2. C: ***Consistency***. Cada transação preserva propriedades do estado global.
3. I: ***Isolation***. Transação executa como se fosse a única no sistema ao ler/escrever dados.
4. D: ***Durability***. Ao concluir uma transação, sistema passa a um novo estado global que permanece, independente de eventos externos.

Questão 12: Considere um sistema bancário transacional e a seguinte implementação da função que transfere da conta c1 para a conta c2 o valor v. Explique o que pode acontecer com esta implementação. Como você corrigiria a implementação?

```
transferencia(c1, c2, v) {
  acquire(c1)
  se (retirada(c1,v) >= 0)
    acquire(c2)
```

```
    deposito(c2,v)
    release(c1)
    release(c2)
    retorna 0
  release(c1)
  retorna -1
}
```

A implementação da função `tranferencia` é capaz de gerar um *deadlock*. Se estiver ocorrendo duas tranferencias simultaneas : c1 quer tranferir para c2 e c2 quer transferir para c1. Quando a primeira realizar o `acquire(c1)` , a segunda transferência realizará o `acquire(c2)`. Dessa maneira, cada um terá um `aquire` e nenhuma das duas transações será capaz de prosseguir. Para corrigir a função `transferencia` podemos fazer da seguinte forma :

```
transferencia(c1, c2, v) {
  acquire(min(c1,c2))
  acquire(max(c1,c2))
  se (retirada(c1,v) >= 0)
    deposito(c2,v)
    release(c1)
    release(c2)
    retorna 0
  release(c1)
  release(c2)
  retorna -1
}
```

Questão 13: Para que serve a técnica de *Two Phase Locking* (2PL)? Explique sucintamente como a mesma funciona.

A técnica 2PL é um mencanismo para controle de concorrência. Usada para garantir atomicidade, consistência e etc. Ele permite varias leituras simultaneas e a escrita nao ocorre conconrrencia, ou seja, quando a escrita está sendo feita, não a leitura e nem escrita simultânea.(**2PL funciona para sistemas com estado global centralizado**). Funcionamento:

- Utiliza dois tipos de *lock* para cada objeto(dado) : *read lock* e *write lock*.
- O *read lock* permite outro *read lock*, poré não permite *write lock*.
- O *write lock* não permite nenhum outro *lock*.
- É composto por duas fases, para cada transação:
  1. Fase 1(*expanding*) : *locks* são adquiridos, nenhum é liberado.
  2. Fase 2(*shrinking*) : *locks* são liberados, nenhum é adquirido.
- Existem duas variações de 2PL:
  1. *Strict Two Phase Locking* : fase 2 libera todos os *writes locks* apenas no final da transação.

2. *Strong Strict Two Phase Locking* : fase 2 libera os *read* e *write locks* apenas ao final da transação.

Questão 14: Para que serve o protocolo de *Two Phase Commit* (2PC)? Explique sucintamente como o mesmo funciona.

O *Two Phase Commit* é um protocolo para *commit* atômico distribuído, ou seja, é um protocolo usado para coordenação de alteração de dados em um sistema com estado global distribuido. Funcionamento:

- O Processo que inicia a transação age como coordenador.
- Duas Fases:
    1. Preparar e votar: processos localmente decidem se podem efetuar a transação.
        1. O coordernador envia partes diferentes da transação para diferentes processos.
        2. Cada processo se prepara para executar a sua parte da transação.
        3. Cada processo responde ao coordenador se tudo está certo para a execução ou se houve algum problema, caso haja, a transação é abortada.
    2. Executar: processos mudam o estado local.
        1. Se todas as respostas dos procesos forem positivas, envia a mensagem *commit*, caso ao contrario, envia uma mensagem *abort*.
        2. Ao receber o *commit*, o processo efetua a subtransação, libera os *locks* e responde OK.
        3. Ao receber o *Abort*, o processo aborta a transação, libera os *locks* e responde OK.

Questão 15: O protocolo Two Phase Commit (2PC) evita deadlocks em sistemas transacionais distribuídos? Explique sua resposta. Em caso negativo, como podemos lidar com deadlocks.

O 2PC não evita os *deadlocks*, pois a a execução distribuída das transações pode causar uma dependência cíclica no *locks*. A forma utilizada para lidar com esse tipo de *deadlock* é a utilização de de *timeouts*. Os processos aguardam um tempo para a espera do *lock* ou das mensagens do coordenador, caso nenhum dos casos ocorra, a transação é abortada. Após abortar, o coordenador aguarda um tempo e tenta novamente executar a transação.

Questão 16: Considere o diagrama de transição de estados do protocolo Two Phase Commit (2PC):

1. **Explique o que acontece quando um processo participante falha no estado INIT. Como o protocolo recupera desta falha?**

- Caso um processo participante falhe no estado INIT, envia para o coordenador uma mensagem *abort* para que a transação não seja efetivada.

2. **Explique o que acontece quando um processo participante falha no estado READY. Como o protocolo recupera desta falha?**
   - Caso um processo participante galhe no estado READY, quando ele se recuperar, irá trocar mensagens com outros processos participantes para saber se foi para o estado ABORT ou COMMIt e executará o mesmo.
3. **Explique o que acontece quando o coordenador falha no estado WAIT. Como o protocolo recupera desta falha?**
   - Caso o coordenador falhe no estado WAIT, o processo em READY vai contactar outro processo para saber se a ação foi de COMMIT ou ABORT e irá repetir o mesmo. Caso todos os processos participantes estejam em READY, nenhuma decisão pode ser tomada, logo, todos aguardam C se recuperar.

Questão 17: Explique os conflitos read-write e write-write que surgem quando temos sistemas distribuídos com dados replicados.

- Conflitos *read-write*:
  - Ess tipo de conflito ocorre quando dois processos querem ler e escrever simultaneamente. O processo 1 faz a leitura enquanto o processo 2 está alterando os dados, dessa forma o dado que foi lido pelo processo 1 não é mais o mesmo, dessa forma, incorreto.
- Conflitos *write-write*:
  - Ocorre quando dois processos tentam escrever simultaneamente. O processo 1 faz a alteração de dados e o processo 2 sobreescreve o dado. É uma situação de condição de corrida, onde o valor final não é possível ser determinado.

Questão 18: Considere as seguintes execuções de instruções em diferentes processos, cada qual com sua memória local (assuma que inicialmente os valores das variáveis são zero). Indique quais casos (execuções) respeitam o modelo de consistência sequencial, indicando uma possível ordenação para as instruções.

1. P2: R(x,0); P1: W(x,1); P2: R(x,1).
2. Não respeita a consistência sequencial. P2 leu inicialmente `x= 1` e depois `x= 0`, sendo que o processo começa com o valor `x= 0`.
3. P1: W(x,1); P3: R(x,1); P2: W(x,2); P3: R(x,2).
4. P2: W(x,2); P3: R(x,2); P1: W(x,1); P3: R(x,1);
5. Não respeita a consistência sequencial. P3 e P4 fazem a leitura de contraria, dessa forma, não é possivel determinar uma ordem para os eventos, logo a consistência sequencial não é respeitada.

6. P4; P1; P3; P2;
7. Não respeita a ordem de execução.

Questão 19: Em se tratando de sistemas tolerante a falhas, qual é a diferença entre disponibilidade (availability) e confiabilidade (reliability)? Dê um exemplo.

- Confiabilidade (*reliability*): Diz respeito ao tempo operacional continuamente até que ocorra falha no sistema. Exemplo: o tempo em que a lampada leva pra queimar.
- Disponibilidade (*availability*): Diz respeito a fração de tempo em que o sistema está operacional. Exemplo: A fração de tempo em que a lampada está acesa.

Questão 20: Considere um componente com MTTF = 2.5 ano e MTTR = 32 horas. Considere o uso de componentes redundantes para projetar um sistema cujo componente tem disponibilidade de 99.99%. Assumindo que falhas deste componente são independentes no sistema, determine o número de componentes redundantes necessários.

A desejada = 99,99%. A = MTTF/(MTTF + MTTR) = 99,85%$ A = 1 - (1 - p)ˆk. 0,9999 = 1 - (1 - 0,9985)$^{k.\ (0,0014)}$k = 0,0001. k = 2.

Questão 21: Considere a organização de componentes redundantes TMR (Triple Modular Redundancy).Explique o que ocorre nos seguintes casos:

1. **Exatamente um componente e um votador falha em cada linha.**
   - Depende. Se a falha ocorrer na mesma coluna, ou seja, no mesmo tipo de componente, o sistema irá falhar. Porém, se for um falha em cada tipo de componente, o sistema funcionará sem problemas.
2. **Dois votadores falham na mesma coluna.**
   - Se dois vetores falharem na mesma coluna, o sistema não funcionará. Se dois votadores falharem, os proximos componentes receberão nenhuma entrada, logo não haverá resoltado. Como são a maioria, os proximos votadores irão ficar com a resposta vazia e o sistema não funcionará mais.

Questão 22: Explique por que falhas bizantinas são mais difíceis de lidar do que falhas que travam (crash failures).

Falhas bizantinas são falhas em que o comportamento do processo gera um resultado errado, podendo ser de forma proposital. *Crash failures* são tipos de falhas relacionados ao desligamento/desconexão do processo, sem produzir

resultados errados. Dessa forma as falhas bizantinas são mais difíceis de serem tratas pois não é tão simples descobrir se o resultado gerado é errado ou não. O comportamento malicioso aumenta essa dificuldade, visto que pode-se produzir resultados inesperados.

Questão 23: Considere o protocolo de acordo bizantino com três participantes sendo um deles operando em falha bizantina. Mostre que o protocolo falha, ou seja, que os processos que não estão em falha não chegam a um consenso sobre o identificador do outro

- Três processos : P1, P2 e B(bizantino).
- O processo P1 envia 1 para todos.
- O processo P2 envia 2 para todos.
- O processo B envia x para P1 e y para P2.
- Os vetores dos Processos:
  - P1: (1,2,x)
  - P2: (1,2,y)
  - B : (1,2,3)
- Os processos não bizantinos não conseguem chegar em um acordo em relação ao identificador do b, visto que, não existe uma maioria para que seja possível eleger um indentificador.