Gabriela Lemos Lúcidi Pinhão

Sistemas Distribuídos – Lista 3

1) A abordagem cliente-servidor é aquela é em que o servidor aguarda uma requisição, processa e devolve ao cliente o resultado. Já o cliente envia uma requisição e aguarda a resposta. Nesse modelo, caso algum defeito ocorra no servidor, nenhum ciente poderá obter resposta, sendo um ponto único de falha. Além disso, caso haja muitos clientes fazendo requisições ao servidor, este pode ficar sobrecarregado, apresentando problemas de escalabilidade.
   Já na abordagem par-a-par esses problemas são resolvidos, pois todas as máquinas envolvidas atuam como cliente e servidor, atendendo e fazendo requisições, não tendo um ponto único de falha e, caso muitas máquinas estejam envolvidas fazendo requisições, estarão também envolvidas em atende-las, resolvendo o problema de escalabilidade.
   Considerando o compartilhamento de um arquivo que está num servidor aos seus clientes, utilizando o modelo cliente servidor, o tempo de compartilhamento será maior devido ao fato de que o servidor deve fazer upload de uma cópia para cada cliente. Utilizando o modelo par-a-par, apenas o upload de uma cópia deve ser feito pelo servidor e cada cliente fará download de uma parte do arquivo. Posteriormente, cada cliente fará upload das partes de possui e download das que não possuem a partir de outros clientes, completando o arquivo.
2) Na forma iterativa cada requisição ao servidor devolve como resposta o IP no próximo servidor que deve ser consultado. O próximo fará a mesma coisa até que se obtenha o IP final do nome pesquisado. Na forma recursiva, o servidor consultado faz uma requisição ao próximo, até resolver o nome quando chegar num servidor autoritativo ou “folha”. Esse servidor final enviará para a camada acima o IP do nome resolvido, até chegar ao *root level server*, que retornará a resposta que estava aguardando. A forma recursiva é mais rápida para retornar a resolução do nome, mas também é mais pesada pois cada servidor deve guardar o nome a ser resolvido e esperar a resposta de sua requisição.
3) O único servidor deve ser o próprio cos.ufrj.br, que continuará sendo um servidor autoritativo que conseguirá resolver o IP final de c1 e c2 mas também um *name server*, sabendo resolver o IP do servidor autoritativo lab.cos.ufrj.br que, por sua vez, saberá resolver os IPs finais de suas máquinas.
4) Fazendo o uso de uma CDN, o conteúdo da empresa estará espalhado por vários lugares do mundo, eliminando o ponto único de falha e também trazendo o conteúdo para mais perto de seus consumidores, fazendo a distribuição ser mais rápida.
5) O DNS é usado para resolver o nome de um servidor onde estará armazenado o conteúdo que o cliente quer acessar. Uma estratégia seria primeiro resolver o nome de um servidor onde estará guardado o nome de outro servidor que, por sua vez, guardará o conteúdo requisitado. Isso permite que um usuário possa acessar o mesmo link de qualquer lugar do

mundo, mas tenha o conteúdo fornecido por um servidor mais perto de onde se encontra.

6) Escolhendo-se aleatoriamente pode ser que não consiga transmitir blocos para nenhum *peer* dos 50 com alta taxa de upload, fazendo com que este nunca seja parte do top 4 dos outros *peers*. Podemos exemplificar um *peer* do Japão recebendo 50 *peers* do Brasil para estabelecer conexão. Ele vai sempre ter taxa de upload baixa quando comparado aos outros *peers*.
7) É possível. Primeiramente, podemos pensar que não, pois todos os outros *peers* estarão utilizando a abordagem tier-by-tier na qual eles farão upload de blocos apenas para os 4 *peers* que lhe oferecerem taxa de download mais alta, o que nunca será o caso do *peer* modificado pelo colega da ECI. Porém, eles também estarão utilizando a abordagem *optimitically unchoke*, em que a cada 30 segundos escolhe um peer aleatório e começa a enviar blocos. Dessa forma, sendo escolhido aleatoriamente por vários peers, o peer em questão pode acabar completando os blocos do arquivo. Isso vai demorar para acontecer, mas pode acontecer, principalmente se o arquivo for pequeno.
8) Primeiramente precisa informar aos *peers* antecessores e sucessores de sua existência e depois armazenar a parte da tabela hash que pertencia ao seu sucessor e que agora lhe pertence.
9) O *cashing* vai fazer com que o peer armazene o resultado da tabela hash que está em outro peer, não precisando fazer a consulta. Como consequência, o resultado é encontrado mais rapidamente.
10) Um dos problemas é lidar com diferentes representações de dados, como, por exemplo *big endian* e *little endian*. Se a máquina mandando a informação utilizar uma e a recebendo utilizar outra, a comunicação será falha. Esse problema é resolvido com o uso de *stubs*, que fará o *marshalling*, convertendo a representação para uma que independe da estrutura, uma padrão, que todas as máquinas fazendo uso de um *stub* poderá assimilar.
    Outro problema é como comunicar à outra máquina uma estrutura de dados já que o ponteiro para a estrutura de uma máquina não fará sentido para a outra. A solução é usar o *stub* para fazer a cópia dessa estrutura quando um ponteiro for referenciado, porém o problema persiste quando a estrutura contiver ponteiros, que, por sua vez, podem conter outros ponteiros. Em algumas linguagens, não é possível para o stub fazer cópia de toda essa hierarquia.
11) O retorno deve acontecer por interrupção em forma de um sinal, que quando recebido, é tratado por uma *signal handler.* Muito provavelmente deverá ser feito o uso de mecanismos de sincronização, como semáforos, para sincronizar as interrupções.
12) O problema é que duas máquinas envolvidas no sistema podem executar funções que dependem do tempo e, se não estiverem sincronizadas, podem alterar o resultado final desejado. Por exemplo, dois alunos editando um arquivo no Dropbox. Supondo que o aluno 1 editou o arquivo e depois desse o aluno 2 também editou, mas seu relógio estava atrasado.

Dessa forma, as alterações do aluno 2 não irão aparecer, pois o Dropbox vai considerar que a versão do aluno 1 é a mais atualizada.

13) A máquina 1 deseja sincronizar seu relógio com a máquina 2. Nesse processo, existem quatro tempos envolvidos. T1 é o instante que a máquina 1 envia o pedido para a máquina 2. T2 é o instante em que a máquina 2 recebe o pedido. T3 é o instante que a máquina 2 envia a sua hora para a máquina 1. T4 é o instante em que a máquina 1 recebe a resposta. A máquina 1 deve ajustar sua hora para T3 + d, sendo d o intervalo de tempo que a máquina 2 demorou para enviar a resposta à máquina 1. A máquina 1 consegue saber a diferença T4 – T1 pelo seu próprio contador. A diferença T3 – T2 a máquina 2 sabe pelo seu próprio contador e envia à maquina 1 junto da resposta da sua hora T3. Desta forma, a máquina 1 faz os seguintes cálculos para achar o intervalo d: d = [(T4-T1) – (T3-T2)] / 2. Depois, ajusta seu relógio para T3 + d.
    Considerando T1 = 1:00; T2=2:02; T3=2:03; T4=1:04, então d=0:015 e a hora da máquina 1 será 2:045.
14) A ajuste direto nos relógios das máquinas acarretaria a variação brusca de horário e também a possibilidade de o relógio “voltar no tempo”. O que acontece é que o há a criação de um relógio virtual com taxa de progressão controlada pelo sistema operacional cuja referencia de tics é a do RTC (Ex: 10 tics do RPC representam 1 tic do relógio virtual). Conforme a sincronização com outros relógios acontece, o SO aumenta ou diminui essa taxa, ajustando a hora do relógio virtual. É esse relógio que as aplicações da máquina vão utilizar.