Comunicação em Sistemas Distribuídos

A comunicação exerce um papel crucial em um sistema distribuído. Estudar sistemas distribuídos sem estudar comunicação não faz nenhum sentido.

Como sabemos, em um sistema distribuído, várias máquinas dividem a tarefa de executar um processo e, por isso saber como elas conversam entre si é um fator tão importante. Este post inicia os estudos de como é feita a comunicação entre cada um dos componentes de um sistema distribuído. Esta comunicação abrange desde os protocolos de rede utilizados na troca de informações entre as máquinas quanto as quatro maneiras de comunicação mais utilizadas nos sistemas de hoje, chamada de procedimento remota (RPC- Remote Procedure Call), middleware orientado a mensagens (MOM Message Oriented Middleware), streaming e finalmente comunicação do tipo multicasting será descrita.

Protocolos de rede

Quando duas máquinas querem conversar elas não compartilham a memória, ao invés disto elas enviam e recebem mensagens de comunicação.

O primeiro passo para se estabelecer uma comunicação entre duas máquinas é a definição do protocolo que será utilizado na comunicação entre elas. Os protocolos são análogos as línguas que usamos para falar, portanto duas máquinas conversando com protocolos diferentes é a mesma coisa que colocarmos um Árabe para falar com um Chinês, como eles não falam a mesma língua a conversa não vai fluir.

No caso das pessoas basta que elas saibam uma língua em comum para que elas consigam se comunicar de forma eficiente, já no caso das máquinas elas precisam definir qual protocolo será utilizado.

Um protocolo define desde a parte física da comunicação, definindo atributos, como a quantidade de volts que serão utilizadas para determinar o valor do 1 e o valor do 0, no caso de uma comunicação binária, quanto saber qual é a última informação de uma mensagem, se ela perdeu alguma informação entre a transmissão e a recepção. Este tipo de controle é muito importante para manter o fluxo de comunicação de maneira correta e válido. Outros tipos de informações muito importantes a serem definidos nos protocolos são, as informações de qual será o tamanho das variáveis numéricas, textuais e outros tipos de dados. Todas estas informações são necessárias para estabelecer a comunicação entre duas máquinas.

O padrão mais conhecido de comunicação entre duas máquinas é o modelo OSI, que foi criado em 1995. Este modelo divide a comunicação entre máquinas em 7 camadas, que serão mostradas a seguir:

1. Física: onde são definidos quais dispositivos serão usados como modem, Bluetooth, USB, DSL, Rede Ethernet, etc
2. Enlace ou dados: camada utilizada para corrigir os possíveis erros de comunicação existentes entre as partes, controla o fluxo e estabelece comunicação entre sistemas diretamente conectados, como Ethernet, 802.11 WiFi, etc.

3. Rede: Fornece meios funcionais para conexões entre redes e realiza o roteamento das funções,podendo realizar uma fragmentação dos dados, como IP(IPV4, IPV6), IPSec.
4. Transporte: Responsável pelo transporte dos dados recebidos pela camada de sessão para a camada de Rede. Dentre as funções desta camada temos o controle de fluxo, a ordenação dos pacotes e a correção de erros. Garante a entrega das mensagens na sequencia correta, sem perder ou duplicar pacotes. Esta camada separa as camadas de nível físico (1 a 3) das camadas do nível de aplicação (5 a 7), e ela é utilizada pelo TCP, UDP, RTP, SCTP, etc
5. Sessão: Determina como será a transmissão dos dados para que duas máquinas diferentes se comuniquem. Marca os dados, no intuito de possibilitar a correção deles no futuro para caso haja falha na transmissão dos pacotes, a transmissão será reiniciada de onde parou, usado por exemplo, pelo NetBIOS, etc
6. Apresentação: Responsável por realizar traduções, como uma Tabela ASCII para outro padrão, pela compactação dos dados e pode ser usada com alguma criptografia, como exemplos temos o XDR, TLS, etc
7. Aplicação: São os programas que usamos para se comunicar na rede, é a interface entre o usuário e a máquina, como exemplos temos o BitTorrent, Telnet, SSH, DNS, Ping, HTTP, FTP, SMTP, etc.

Antes de definir como os computadores se comunicam iremos abordar alguns pontos importantes para o entendimento de como os parâmetros são passados para os métodos em uma linguagem.

Stub

Stub em ciência da computação é usado para definir um esboço de um método e eles são muito úteis para portabilidade, sistemas distribuídos e teste de software.

Um método stub contém algumas partes de código que realizam uma abstração de uma camada de software. Eles se comportam como um proxy para objetos remotos. Tendo como exemplo um método de soma, localizado em um servidor, um stub deste método seria uma interface local, com apenas a mesma assinatura deste método e ao ser chamado ele chamaria o método remoto. Atuaria como um proxy.

Call-by-name e Call-by-value

Quando usamos parâmetros em métodos dentro de um programa que estamos construindo, eles podem ser interpretados de maneiras diferentes, que são definidos por *call-by-name*, e *call-by-value*.

Um parâmetro *Call by value* tem o seu valor copiado para o parêmetro do método, ou seja, quando o método alterar aquela variável nada acontecerá com o valor da variável original.

Já em um parâmetro do tipo *Call by name* isso não acontece, pois ela é passada como um ponteiro para o método, e o valor utilizado dentro daquele método referencia a mesma área da memória que a variável original, com isso qualquer modificação realizada no método também será realizada na variável original.

Na linguagem C parâmetros do tipo int, byte são *value* e parâmetros do tipo array são *name*. O mesmo vale para Java e C#, assim como as outras linguagens que são executadas em seus ambientes.

Esta diferença entre as variáveis recebidas pelos métodos também acontece, nas chamadas remotas a métodos, conforme será mostrado a seguir.

RPC

RPC é um tipo de chamada remota a um método, e estas chamadas quase sempre se comportam de maneira análoga ao procedimento realizado localmente na máquina, quando estamos tratando de parâmetros dos métodos, mesmo para os parâmetros *Call-by-name* caso este valor seja alterado remotamente, a variável local será alterada também. Como exemplo de RPC podemos ter a leitura de um arquivo, a execução de um método, a autenticação, etc.

As RPCs são utilizadas quando queremos rodar um método, ou um procedimento em uma máquina remota.

No caso desta execução ser em um método de leitura de dados, por exemplo uma leitura de um arquivo do sistema de armazenamento, quando este tipo de operação é realizada em um sistema local, o computador chama um procedimento de leitura de dados do Sistema Operacional, ou seja, existe uma interface entre o método de leitura da linguagem utilizada e a chamada a leitura do Sistema Operacional. Para fazer uma chamada de leitura de dados de maneira remota, uma chamada análoga é realizada, só que desta vez faremos uma requisição de leitura para o stub da máquina remota.

Por padrão as RPCs são síncronas, ou seja, ao chamar um método seu sistema fica travado, esperando que a resposta deste método chegue, para poder continuar a execução.

Chamadas síncronas são utilizadas quando a resposta daquele método é necessária para a continuidade da execução do método corrente, por exemplo, em um método de login, a validação da senha pode ser uma chamada síncrona, já que ela é necessária para validar se o usuário pode ou não prosseguir.

RPC Assíncrono

Em uma execução de um RPC comum o cliente ficará travado, esperando pelo resultado da chamada ao método para poder continuar a sua execução. Ao fazer uma chamada assíncrona isso não acontece. Esta implementação de RPC provê ferramentas para facilitar a utilização dela de maneira assíncrona.

Um exemplo de chamada assíncrona é o carregamento de dados após um login, muitas vezes o sistema já carrega dados do usuário, que não são necessários para o login, como endereço, telefones, etc, assincronamente, com isso é possível logar em um site, e ir para a home, sem ter que esperar o resultado da busca destas informações no banco de dados.

DCE ou Distributed Computing Environment

O DCE é uma implementação de chamadas remotas a procedimentos, no nível do sistema operacional. Esta implementação provê uma maneira de realizarmos chamadas remotas as interfaces do sistema operacional, e isso é feito de maneira que elas pareçam ser realizadas realizadas localmente, pelo menos pro programador há esta impressão. Este tipo de implementaçã é uma abstração do sistema operacional para a rede.

Ele foi inicialmente desenvolvido para o Unix por um consórcio chamado de Apollo Computer, mas depois uma implementação para o Windows foi realizada.

A implementação do DCE provê um framework para o desenvolvimento de aplicativos cliente/servidor, que inclui RPC, um serviço de nomeação de diretórios, autenticação, um serviço de relógio e um sistema de arquivos distribuído.

As implementações de WebServices, Java e a própria internet carregam em si muitos dos conceitos elaborados pelo DCE. Uma das implementações atuais de DCE é o sistema de ODBC da Microsoft, utilizado para comunicação com banco de dados.

O DCE é dividido em células onde cada uma delas contem um Serviço de Segurança, um Diretório, que será o repositório de Controle de acesso ao sistema e um Sistema de relógio distribuído.

Socket, stream, mpi e message queue

1. Comunicação orientada por mensagens

Remote procedure calls e remote object invocations são chamadas que promovem a transparência nas chamadas e com isso acabam escondendo um pouco da comunicação existente para a realização destas chamadas.

Ao fazer uma RPC não conseguimos detectar se o sistema, onde faremos a chamada está rodando, com isso o desenvolvimento de sistemas de comunicação alternativos foram necessários. Além disto, a natureza síncrona de uma chamada RPC algumas vezes precisa ser substituída por outra implementação.

Aqui serão apresentados os sistemas de mensagem que são uma alternativa ao RPC nos sistemas distribuídos.

2. Sockets

Conceitualmente um socket é um ponto de conexão onde uma aplicação pode escrever dados que serão transmitidos pela rede, ou ler dados que chegam pela rede. O socket é uma camada de abstração utilizada pelo sistema operacional e por um protocolo de transmissão de dados específico, para abrir um canal de comunicação entre duas máquinas.

Um servidor socket, quando se inicializa, reserva recursos da máquina para mandar e receber mensagens em um protocolo específico, isso se dá por reservar uma porta local para a transmissão dos dados.

A implementação de socket possui uma lista de comandos primitivos que serão utilizados para estabelecer a comunicação e transmitir dados, estes comandos são listados a seguir:

1. **Socket** Cria um endpoint de comunicação
2. **Bind** Anexa um endereço local em um socket
3. **Listen** Escuta um endereço para aceitar conexões
4. **Accept** Bloqueia o chamador até que uma requisição seja feita
5. **Connect** Tenta estabelecer uma conexão
6. **Send** Manda dados pela conexão
7. **Receive** Recebe dados pela conexão
8. **Close** Fecha a conexão

Ao inicializar um processo socket no servidor, ele irá utilizar o comando de bind de uma porta, ou seja, ele irá atribuir uma porta ao serviço de socket. Esta porta ficará esperando requisições de um cliente para transmitir dados. Um cliente socket conectado ao servidor, pode executar chamadas de leitura e escrita de dados nele, e estas mensagens podem ser tanto síncronas quanto assíncronas.

Um ponto importante a ser observado na comunicação que utiliza um socket é que o servidor deve manter aberta a porta de conexão enquanto ele estiver rodando para que com isso, seja possível transmitir dados entre duas máquinas. Este ponto é uma das desvantagens do socket já que os recursos das máquinas dos servidores devem ficar alocados enquanto uma comunicação é esperada.

3. Interface de envio de mensagens MPI

Conforme as máquinas foram evoluindo e com o aparecimento de processadores com multiplos cores, o que gerou a necessidade de se criar softwares que podiam processar requisições em paralelo. Surgiu uma interface de envio de mensagens, e com isso, inicialmente os fabricantes de grandes máquinas desenvolveram sistemas próprios de comunicação, adaptada e otimizada para o seu tipo de hardware.

Estas mensagens eram incompatíveis entre elas e isso gerou um problema de transcrição para os programadores.

Caso fosse necessário realizar uma comunicação entre dois sistemas de fabricantes diferentes, uma tradução de mensagens deveria ser implementada pelos programadores, o que se mostrou uma das desvantagens destes sistemas.

Desta incompatibilidade surgiu um padrão de troca de mensagens que fosse independente de plataforma e hardware, este sistema foi chamado de MPI (Message Passing Interface).

A MPI assume que falhas de rede e problemas sérios com as máquinas são fatais e não precisam de uma recuperação automática, ela assume também que a comunicação é feita por um grupo conhecido de processos, no qual cada grupo possui um identificador, assim como cada processo de um grupo também possui um identificador, o que identifica unicamente o emissor e o destino de uma mensagem.

Tanto operações transientes síncronas e assíncronas são suportadas pela MPI, tudo dependerá de quais comandos primitivos serão executados durante a troca de mensagens. Mensagens transientes são aquelas mensagens que não são armazenadas.

4. Fila de Mensagens

Um sistema de fila prove um protocolo de comunicação por troca de mensagens onde quem envia e quem recebe não precisam tratar a mensagem ao mesmo tempo. A mensagem é armazenada em uma fila até que um recebedor peça para processá-la.

As mensagens são repassadas entre servidores de comunicação até que elas, eventualmente, serão entregues ao seu destino final, mesmo quando o seu destino não estiver funcionando quando a mensagem for entregue.

Existem leitores de mensagens, que são responsáveis por receber, enfileirar e processar as mensagens recebidas, e os escritores os quais enviam mensagens aos receptores. Na maioria das vezes eles estão diretamente conectados, o que faz com que as mensagens sejam enviadas diretamente.

Os leitores possuem uma fila de mensagens a serem processadas, que serão processadas uma a uma, obedecendo o algorítimo de ordenamento da fila (fifo, filo, etc)

Neste tipo de processamento quem envia a mensagem tem a garantia que a sua mensagem será colocada na fila de processamento do receptor, mas não há garantias de quando isso vai acontecer, nem se esta mensagem um dia será lida, pois isto é totalmente determinado pelo comportamento do leitor da mensagem.

As mensagens poderão carregar qualquer tipo de dado, o ponto mais importante das mensagens é o endereço de entrega. A maioria dos sistemas de fila habilitam a instalação de uma função de callback, a qual será executada quando a mensagem for colocada na fila. A função de callback serve para avisar a quem enviou a mensagem, de que a sua mensagem foi executada, com sucesso ou não.

Um sistema de fila é persistente já que as mensagens são armazenadas e podem ser recuperadas em caso de falha da máquina.

Existem várias possibilidades de acontecer uma troca de mensagens, já que quem recebe e quem envia não precisam estar necessariamente rodando quando uma mensagem será processada. Tanto um, quanto o outro podem estar desligados quando a mensagem for enviada ou recebida, mas isso não altera o resultado final do processamento da mensagem.

Um sistema de fila típico, possui um administrador de filas, ou broker, que é configurado pelo administrador do sistema. Ele será responsável por armazenar e endereçar as mensagens aos receptores corretos, assim como administrar o tamanho da fila e das mensagens, que são limitados, alem de um software que irá retirar e processar as mensagens que foram colocadas na fila, e um software que enviará mensagens para a fila.

O administrador da fila irá armazenar as mensagens recebidas até que algum recebedor peça a ele aquela mensagem, ou que essa mensagem caduque, ou seja, estoure um tempo

determinado pelao configuração daquela fila. O processamento da mensagem irá acontecer assim que a mensagem é transferida ao recebedor.

Um sistema de fila pode possuir mais de um administrador de fila, e eles podem estar tanto em máquinas diferentes, quanto em redes ou localidades diferentes.

Este tipo de sistema também é usado como backpressure, ou seja, ele pode atenuar a chegada de uma grande quantidade de mensagens em um período muito curto, e evitar que o sistema de processamento caia.

5 Comunicação por Stream

Stream pode ser definido como fluxo de dados em um sistema de computador. Até agora só estudamos sistemas onde mandávamos um pacote de dados, uma mensagem, e pronto, mas e quando precisamos enviar uma grande quantidade de dados? Ou mesmo e quando devemos transmitir algo que está acontecendo neste momento, como transmissões de voz, vídeo, etc.

Nas outras formas de transmissões estudadas até agora, a sequência em que os dados eram enviados, ou recebidos não importavam tanto, já que eles poderiam ser reordenados antes de serem processados. Quando o stream é usado, isso não acontece, pois durante a transmissão de um show, não dá para misturar a sequência de sons, eles tem que ser transmitidos na mesma sequência.

Uma outra maneira de se utilizar o stream é durante a leitura dos arquivos de dados que temos armazenados em um hd. Estas operações de leitura, escrita ou atualização, utilizam um stream, visto que, ele também deverá ser aberto e gravado de forma sequencial.

Na transmissão de dados multimídia, por exemplo um vídeo que contém tanto o som quanto as imagens e estes dois fluxos de dados devem ser sincronizados para que o vídeo seja mostrado corretamente para o usuário.

Existem vários exemplos de sistemas de stream disponíveis na web, como o youtube, o netflix, deezer, spotify, globoplay, além de transmissões de dados de eventos ao vivo, como jogos, rádios, etc. Vejam que, alguns destes exemplos implementam um sistema que diminui a qualidade do vídeo, ou mesmo do som, de acordo com a velocidade de conexão do usuário. Conforme a velocidade de conexão aumenta, a qualidade do vídeo e do som melhora, conforme ela diminui, a qualidade do vídeo e do som piora, isto é muito útil em casos de conexões com velocidades variáveis como a internet dos celulares, já que ao se movimentar com um aparelho celular, nem sempre é possível manter a mesma velocidade da sua conexão.

Multicast

Multicast é utilizado quando há a necessidade de difundir uma informação para vários pontos diferentes, dentro de uma mesma rede. Várias técnicas foram utilizadas para difundir as informações, incluindo espalhamento por pontos, técnicas de infecção e protocolo de fofoca.

A ideia principal no multicast feito ao nível de aplicativos é fazer com que os nós se reconheçam dentro de uma mesma rede, e com isso consigam disseminar informações.

Existem duas maneiras de se construir uma rede para multicast, na primeira delas os nós se organizarão em uma árvore, o que significa que só existirá um caminho entre qualquer par de nós. Já na outra maneira os nós são organizados em forma de malha, onde cada ponto tem múltiplas conexões com outros nós. A principal diferença entre estes dois tipos de rede é que na segunda, em forma de malha, uma falha de um nó não afeta muito a performance da rede, já que a rede não precisará se reorganizar imediatamente, ou seja ela é mais robusta. Já na primeira, em forma de árvore, cada falha de um nó, irá impedir um caminho entre dois pontos, obrigando a rede a se reorganizar.

Técnicas de infecção

As técnicas de infecção foram algorítmos criados a partir de estudos do comportamento dos vírus, de como eles fazem para se espalhar nos organismos vivos. Estes algorítmos tentam reproduzir no software o mesmo comportamento dos vírus, só que aqui este comportamento será utilizado para disseminar informações.

Um dos principais modelos de propagação é o de anti-entropia e neste modelo um nó **I** pega um outro nó **D** randomicamente e troca informações com este nó **D**. Neste modelo existem três formas de propagação da informação.

1. **I** somente envia informações para o **D**
2. **I** somente recebe informações do **D**
3. **I** recebe e envia informações com o **D**

No primeiro caso, as informações não serão espalhadas rapidamente, já que a informação só será espalhada pelos nós que estão contaminados (por contaminados entenda atualizados), com isso no começo poucas atualizações serão repassadas. Ao mesmo tempo, quando muitos nós estão atualizados, as chances de eles selecionarem um nó ainda não atualizado é relativamente pequena, o que faz com que demore para completar a disseminação para todos os nós.

O segundo caso trabalha muito bem quando muitos nós estão contaminados, já que as chances de se conectar a um nó não contaminado é grande.

Dentre os modelos de propagação apresentados, o mais rápido será o terceiro, onde as informações vão para os dois nós, o que faz com que ela se propague mais rapidamente.

Uma variância deste método é o chamado de método de fofoca, que funciona exatamente da mesma forma como quando conversamos sobre alguma novidade. Imagine que você gostaria de contar algo novo aos seus amigos, no inicio você liga para uma pessoa e conta a novidade.

Assim como você esta pessoa também ficará empolgada em espalhar esta atividade, até que ela tente contar a novidade para uma pessoa que já saiba o fato, isso fará com que o disseminador (leia-se fofoqueiro) desanime de contar a novidade para outras pessoas, já que isso não será mais novidade.

No modelo de fofoca, os nós vão espalhando a informação até que encontrem alguém que já foi atualizado, a partir dai este nó para de disseminar a informação. Este método é uma maneira muito boa de se espalhar as informações, porém com ela não é possível garantir que a informação será espalhada para todos os nós.

A grande vantagem dos algorítmos de epidemia é a sua escalabilidade, já que o número de sincronizações entre os processos é bem pequena.

Removendo dados

Remover dados utilizando os algoritmos de epidemia é complicado, esta complicação se dá pois após remover os dados de um nó, ele ficará vazio, e eventualmente ele irá receber uma atualização que recarregará os dados nele.

A dica para remover dados usando um algorítmo de epidemia é gravar a remoção como sendo uma atualização de dados, com isso os dados daquele nó não serão atualizados novamente.

O grande problema desta solução é que ela é muito custosa em relação ao espaço ocupado pelas informações, uma vez que os dados apagados, não foram retirados do sistema.