DANIEL AZEVEDO NICOLAY

Sistemas Distribuidos

PETRÓPOLIS
19 de novembro de 2015

Sumário

1 RPC - Remote Procedure Call

Chamada de procedimento remoto é uma tecnologia de comunicação entre processos que permite uma aplicação chamar um procedimento em outro espaço de endereçamento (Outro PC em outra rede). Também é uma tecnologia popular para a implementação do modelo cliente-servidor de computação distribuida. Ela é iniciada pelo cliente enviando uma mensagem a um servidor remoto solicitando a execução de um procedimento, e o servidor retorna uma mensagem.

1.1 O Modelo

É similar ao modelo de chamadas locais, onde a rotina que invoca o procedimento coloca argumentos em uma área de memória bem conhecida e transfere o controle para o procedimento em execução.

Uma thread é responsavel pelo controle dos processos invocador e servidor. O primeiro, manda uma mensagem para o servidor, contendo os parametros do procedimento e aguarda uma mensagem de resposta que contém o resultado da execução.

Do lado do servidor um processo permanece em espera até a chegada de uma mensagem de invocação, que quando recebida extrai os parâmetros, processando-os e produzindo os resultados que são enviados na resposta. Uma RPC difere das LPC's em alguns pontos:

1- Tratamento de erros: falhas do servidor ou de rede devem ser tratadas.

2- Variáveis globais e efeitos colaterais: Uma vez que o servidor não possui acesso ao espaço de endereços do cliente, argumentos protegidos não podem ser passados como variáveis globais ou retornados.

3- Desempenho: são mais lentas que as LPC's.

4- Autenticação: podem ser transportadas em redes sem segurança, autenticação pode ser necessário.

1.2 Implementação

Para permitir que servidores sejam acessados por diferentes clientes, diversos sistemas padronizados de RPC foram criados, tendo sua maioria utilizando uma linguagem de descrição de interface (IDL) para que essas diferentes plataformas possam chamar esses procedimentods. Através de algumas ferramentas, pode-se gerar interfaces entre cliente e servidor a partir de um arquivo IDL, chamado stubs. Esses stubs são embarcados nas aplicações cliente e servidor.

Em sua codificação, o RPC chama o stub cliente como qualquer outro procedimento local e sua implementação interna do stub cliente é responsavel por iniciar a transmissão para o stub servidor, empacotando a chamada em mensagem. Ao chegar, o stub servidor desempacota a mensagem e invoca o procedimento. Quando a chamada local retorna, o stub servidor é responsavel por iniciar o processo de transmissão para o stub cliente.

Ao invocar um RPC, deve-se ficar atento as plataformas que o cliente e o servidor utilizam, uma vez que podem ser plataformas diferentes, que representam dados diferentes.

1.3 Limitações

A maioria das implementações não suporta P2P ou interação assíncrona entre o cliente e servidor. A comunicação síncrona implica na disponibilidade constante do cliente e do servidor.

2 RMI - Remote Method Invocation

Remote Method Invocation (RMI) é uma interface de programação que permite a execução de chamadas remotas no estilo RPC só que em aplicações Java. É uma abordagem da plataforma Java para prover funcionalidades de uma plataforma de objetos distribuídos, que faz parte do núcleo básico de Java desde a versão JDK 1.1.

Através dessa arquitetura, é possivel que um objeto ativo em uma máquina virtual java possa interagir com objetos de outras maquinas virtuais java, independente de sua localização.

2.1 Seu Funcionamento

Consiste basicamente em dois programas segundo a arquitetura cliente-servidor. O servidor instancia objetos remotos referenciando-o com um nome e fazendo um vinculo com uma porta, onde este objeto espera por clientes que invoquem seus métodos. O cliente referencia remotamente um ou mais métodos de um objeto remoto. A RMI fornece os mecanismos para que a comunicação seja possível, geralmente sendo denomiada como Aplicação Objeto Distribuido.

Aplicações distribuidas precisam executar as seguites ações: localizar objetos remotos; se comunicar com eles; carregar os bytecodes dos objetos.

2.2 Arquitetura

A arquitetura RMI oferece a transparência de localização através da organização de três camadas entre os objetos cliente e servidor:

A camada de stub/skeleton oferece as interfaces que os objetos da aplicação usam para interagir entre si;

A camada de referência remota é o middleware entre a camada de stub/skeleton e o protocolo de transporte. É nesta camada que são criadas e gerenciadas as referências remotas aos objetos;

A camada do protocolo de transporte oferece o protocolo de dados binários que envia as solicitações aos objetos remotos pela rede.

2.3 Vantagens

Capacidade de baixar o codigo de um objeto, caso a classe dele não seja definida máquina virtual do receptor. Os Comportamentos de um objeto, previamente disponíveis apenas em uma máquina virtual, agora podem ser transmitidos para outra. Permite que o codigo da aplicação seja atualizado dinamicamente sem a necessidade de ser recompliado.

3 Sockets

Um socket é um ponto final de um canal bidirecional de comunicação entre dois programas rodando em rede. Cada socket tem os seguintes endereçõs de ponto final:
1- Endereço local (numero da porta) que se refere ao endereço da porta de comunicação na camada de transporte.
2- Endereço global (nome do host) que se refere ao endereço do computador (host) na rede.

O servidor fica ouvindo o socket aguardando um pedido de conexão do cliente, que por sua vez sabe o host e a porta associada a aplicação. Assim que o servidor aceitar a conexão, este cria um novo Socket e pode ficar esperando novas conexões no socket original, enquanto atende as requisições do cliente.

3.1 Sockets UDP

O Protocolo UDP não estabeleçe conexão, como já conhecemos. O remetente coloca explicitamente endereço IP e Porta do destino, no formato IP:Porta (ex: 192.168.0.1:80). O servidor deve extrair o Endereço IP e a Porta (Socket) do remetente do DataGrama recebido. Os dados recebidos podem ser recebidos fora de ordem ou até mesmo se perderem. Só então os dados são encaminhados para a camada superior, passando da Camada de Transporte (Protocolo UDP) para a Camada de Aplicação (Aplicativo Qualquer).

Estes Datagram Sockets, ou DataGrama (como havia citado anterioemente), por serem UDP, tem a comunicação não orientada a conexão e não é cofiável, pois os dados que são enviados por esse tipo, como já foi explicado anteriormente, podem não chegar ao destino final da forma como foi enviado e sua chegada não é garantida.

3.2 Sockets TCP

Já no Protocolo TCP, o socket cria uma ligação stream bi-direcional entre os endereços 'hostC:portC' e 'hostS:portS', ou seja, entre uma aplicação cliente em execução na máquina 'hostC' consegue controlar também a porta 'portC' e outra aplicação servidora em execução na máquina 'hostS' conseguindo a visualização da porta 'portS' de 'hostS'. Como sabemos também, o Protocolo TCP é orientado à Conexão e portanto, garante que os dados encaminhados de um Cliente para um Servidor, em suas respectivas portas, cheguem ao seu destino.

Esta forma de comunicação com Sockets TCP, fazem uso de Stream Sockets, que representam a forma de comunicação de duas mãos, ou duas vias que são confiáveis e orientado

a conexão, como já falamos anteriormente que é de característica do próprio TCP. Os dados que foram enviados com esse tipo de Socket estarão seguros e irám chegar da mesma forma que foram enviados, pois o protocolo TCP é responsável por "entregar" os dados da mesma forma que foram enviados e sem erros. Ex: os browsers utilizam essa aplicação através do protocolo HTTP, estes normalmente trabalham no socket 80.

3.3 Comunicação Cliente - Servidor

3.3.1 Servidor

Efetua a criação do socket, associando-o a um endereço local onde fica aguardando a solicitação de conexão pelo cliente. Após aceitar a conexão, lê suas requisições, envia reposta e por fim fecha o socket.

3.3.2 Cliente

Efetua a criação do socket, tenta estabelezar conexão com o servidor. Uma vez estabelecida, envia a requisição e aguarda resposta. Por fim fecha o socket.

3.4 API's Sockets

3.4.1 Bind

Função utilizada apenas pelo servidor, uma vez que associa um determinado endereço IP e porta TCP/UDP para o processo servidor. Prove o nr da porta que o servidor espera contato.

3.4.2 Listen

Indica ao sistema para colocar o socket em podo espera para aguardar conexões de clientes.

3.4.3 Accept

Cria um novo socket apartir do estabelecimento de uma conexão para iniciar a comunicação.

3.4.4 Read & Write

Lê o conteúdo do buffer associado ao socket e escreve os dados em um buffer associado ao socket.

4 DHT - Distributed Hash Table

São uma classe de sistemas distribuídos descentralizados que provém um serviço de busca parecida com a de uma tabela hash: pares (key, value) são armazenados e qualquer nó pode recuperar o valor associado para cada chave. A responsabilidade por manter o mapa de chaves até os valores é distribuida entre os nós de modo que uma mudança no

grupo de nós cause uma interrupção mínima. Isso permite que o DHT funcione para uma grande escala de nós com esses chegando, saindo e falhando continuamente.

4.1 Estrutura

Existe um keyspace abstrato, que corresponde a um conjunto de strings de 128 ou 160 bits. Um esquema de partição do keyspace divide a posse deste entre os nós participantes.Uma rede sobreposta conecta os nós, permitindo que qualquer um possa encontrar o dono de uma certa chave. Para armazenar um arquivo no DHT, faz-se o hash do nome do arquivo, produzindo uma chave k de 128 ou 160 bits. Uma mensagem put(k,data) é enviada de nó em nó pela rede sobreposta até chegar ao nó ao qual foi delegada a posse da chave k na partição do keyspace. Esse nó guarda a chave e os dados. Para buscar o arquivo, basta fazer o hash do nome do arquivo para gerar a chave k e a mensagem get(k). Essa mensagem também é enviada de nó em nó até chegar naquele que possui k, que responderá enviando os dados.

Os DHTs usam um variante de hashing consistente (no qual uma pequena alteração no nome não altera significativamente o hash) que utiliza uma função f(k1, k2) que define uma distância abstrata entre k1 e k2, que não tem a ver com a distância geográfica nem com o tempo de latência. Cada nó recebe um identificador (ID). Se um nó recebe um ID x, ele recebe todas as chaves km para as quais x é o ID mais próximo (por exemplo, o Chord, um dos protocolos DHT mais usados, usa um círculo no sentido horário), que é calculado por f(km,x).

A escolha de quem possuirá k varia, mas todos tem um princípio básico: será escolhido um nó que tenha k em seu ID ou que tenha ligação com um nó que tenha o ID mais parecido com k. è utilizado, então, um algoritmo para encontrar esse nó. Esse tipo de roteamento é chamado key-based routing.

4.2 Protocolos e implementações

4.2.1 Tapestry

Tapestry é uma Tabela Hash Distribuída (DHT - Distributed Hash Table) que fornece localização de objeto, roteamento, e infraestrutura de multicast descentralizados para aplicações distribuídas. Ele é composto de uma rede de sobreposição par-a-par que oferece um roteamento para recursos próximos de forma eficiente, escalável, auto-reparável e ciente de localização.

4.2.2 Chord

O Chord é normalmente utilizado para armazenar pares contendo uma chave e seu valor associado em nós distribuídos pela rede. Posteriormente, o serviço de busca da rede permite que, dada uma chave, seja determinado o nó responsável pela mesma.

O espaço de identificadores é formado por um anel conectado, onde cada identificador possui m bits (tipicamente 160 bits). Tanto os nós quanto os dados a serem armazenados são mapeados através de uma função hash consistente (Karger et al, 1997, citado por Pinheiro, 2006) para pontos desse espaço de identificadores. O mapeamento dos nós

é feito aplicando-se a função hash ao seu endereço IP. Cada chave k é armazenada no nó cujo identificador é igual, ou segue k, no espaço de identificadores. Para permitir a localização das chaves seria necessário apenas que cada nó mantivesse um apontador para o nó sucessor e o predecessor no anel.

5 Bit torrent x Napster x Emule x Kazaa

5.1 Bit Torrent

O compartilhamento de arquivos utilizando a tecnologia Torrent começa com um usuário da Internet chamado de "seeder" (ou, em português, semeador) envia um arquivo de extensão .torrent a um site para que outras pessoas possam achar e baixar o arquivo. O arquivo é dividido em partes menores de 256 kilobytes assim o arquivo a ser baixado pode ser obtido por meio de vários "seeders" que possuem um pedaço distribuído em ordem aleatória.

Ao fim do download os pedaços do arquivo são organizados automaticamente e os usuários que estavam realizando o download podem virar um "seeder" deste que autorize o upload para outros usuários chamados de "leecher" (ou sanguessugas no português). Entretando, as ligações entre os semeadores e o sanguessugas são feitas com o auxílio de um servidor chamado de Trackers. Este servidor não possui nenhum arquivo para ser compartilhado, tem a função de estabelecer a comunicação entre os usuários para isso ele utiliza as DHT (Distributed Hash Table).

5.2 Napster

O computador se conecta ao servidor central do Napster e, na rede de computadores, há vários servidores que processam os pedidos de usuários e respondem esses pedidos quando necessários. Os navegadores da Web permitem que os usuários se conectem a servidores e visualizem as respostas dos pedidos ou seja os pedidos processadas pelos servidores. Os grandes sites com muito tráfego podem ter que comprar e manter milhares de máquinas para apoiar todas as solicitações dos usuários.

Inaugurou o conceito de peer-to-peer de compartilhamento de arquivos. As pessoas armazenavam os arquivos que queria compartilhar em seus discos rígidos e compartilhado diretamente com outras pessoas. O sistema de buscas dependia do servidor central. Cada máquina do usuário tornou-se um mini-servidor.

O Napster procura por arquivos usando um protocolo cliente/servidor. Se um usuário deseja certo arquivo, ele faz um requerimento ao servidor central do Napster ao qual ele está conectado. O servidor verifica em seu diretório se existe um arquivo que corresponda ao requerimento feito pelo usuário. O servidor então envia ao usuário todos os arquivos encontrados (se existirem) juntamente com o endereço IP, nome do usuário, tamanho do arquivo e taxa de transferência de cada um. Depois o usuário escolhe o arquivo desejado da lista correspondente ao requerimento e tenta estabelecer uma comunicação direta com o local onde se localiza o arquivo escolhido. O usuário tenta fazer isto enviando uma mensagem ao outro computador cliente juntamente com seu próprio endereço IP e o nome do arquivo desejado. Se a conexão é estabelecida então o computador cliente

inicia a transferência do arquivo para o usuário. Este protocolo utiliza o protocolo de comunicação TCP/IP. O servidor mantém uma lista com os endereços de todos os nós conectados a ele, e então faz a busca por arquivos . O Napster permite transferência de arquivos que sejam independentes do servidor central. Esta transferência ocorre entre computadores cliente.

5.3 Emule

A idéia do projeto eMule surgiu no dia 13 de maio de 2002 quando um garoto chamado Merkur se reuniu com outros desenvolvedores para criar um aplicativo que utilizasse a rede eDonkey de maneira mais eficiente e mais agradável para o usuário. O eDonkey já era um sistema de distribuição de arquivos ponto-a-ponto popular mas que necessitava de alguns ajustes para ser um sucesso. Foi por essa necessidade que surgiu o eMule.

O eMule é um aplicativo ponto-a-ponto (peer-to-peer) de compartilhamento de arquivos baseado no protocolo eDonkey. Ele também trabalha com a rede eDonkey2000, a qual possui cerca de 11,5 a 12,5 milhões de usuários conectados simultaneamente compartilhando cerca de 700 a 900 milhões de arquivos. É um programa de código aberto lançado sobre a GNU General Public License e roda no sistema operacional Windows possuindo uma versão para Linux denominada aMule. Além de utilizar a rede eDonkey, o eMule tem uma rede ainda experimental própria denominada Kademlia que adota uma política descentralizada - diferente da rede eDonkey.

A rede do eMule possui centenas de servidores e milhões de clientes. Estes devem se conectar ao servidor para ter acesso à rede. Tal conexão se estabelecerá até que o cliente saia do sistema. Os servidores realizam um serviço de indexação centralizado (assim como era no Napster) e não se comunicam com outros servidores.

Ao iniciar o aplicativo eMule, o cliente tentará se conectar a diversos servidores em paralelo, mas estabelecerá a conexão TCP apenas com um. O eMule não permite que um cliente se conecte a vários servidores ao mesmo tempo ou mude de servidor dinamicamente sem a intervenção do usuário. No aperto de mãos (ou handshake) entre o cliente e o servidor, este último envia uma identificação de quatro bytes ao cliente que é denominada Client ID. Esta ID será válida enquanto durar a conexão, a qual somente será encerrada caso o cliente se desconecte daquele servidor, feche o eMule ou perca o acesso à internet.

cliente poderá receber uma ID alta (High ID) ou baixa (Low ID). Ele terá uma ID baixa se ele não aceitar receber conexões, o que ocorrerá caso ele esteja usando algum tipo de firewall não configurado adequadamente para o eMule, estiver conectado através de uma NAT, por um servidor proxy ou se o servidor estiver muito ocupado. Do contrário, ele receberá uma ID alta. Ter uma ID baixa significa ter um acesso restrito a rede e ser possivelmente rejeitado por servidores, que dão preferência a clientes com ID alta. Esta última, garante ao usuário total proveito da rede e será a mesma quando ele se conectar a outros servidores.

5.4 Kazaa

Utiliza uma arquitetura semi-centralizada, e é uma rede subjacente ao Morpheus e o Grokster. Possui um protocolo proprietário que é usado numa rede auto- organizável que aloca um maior número de tarefas a pares que possuem uma largura de banda maior. Os usuários fazem os pedidos e não ficam esperando respostas. Baixa taxa de transferência. A maioria dos pedisos são para objetos pequenos, em que sua popularidade dura pouco. Seu O sucesso de uma rede P2P como o Kazaa depende imensamente da introdução de novos objetos e entrada de novos usuários.

6 Web Services

É utilizado na integração de sistemas e na comunicação entre aplicações diferentes pois permite interagir entre aquelas que já existem e que foram desenvolvidas em diferentes plataformas, indiferente do sistema operacional. São componentes que permitem as aplicações enviar e receber dados em formato XML. Cada aplicação envia uma mensagem que é traduzida para uma linguagem universal onde todas as aplicações a entendem. Seu objetivo é a comunicação entre aplicações através da internet, onde é realizada com untuito de facilitar a integração de aplicações. Muitos consideram que corrigem um grande problema: a falta de integração entre os sistemas. Os web services permitem que a integração seja de maneira reutilizável e padronizada, onde tenta organizar um cenário cercado por uma vasta variedade de diferentes aplicativos em diversas linguagens.

6.1 Papéis dos Web Services

Serviços podem assumir diferentes papéis quando envolvidos em diversos cenários de interação. Dependendo do contexto pelo qual é visualizado, assim como o estado da tarefa rodando no momento, o mesmo web service pode trocar de papéis ou ser designado para múltiplos papéis simultâneos

6.1.1 Provedor de serviços

Disponibiliza uma interface pública onde contém a descrição dos serviços disponíveis por ele.

6.2 Solicitante de serviços

Solicita um web service por um serviço em específico.

6.2.1 Intermediário

É assumido pelo web service quando recebe a mensagem de um solicitante e o encaminha para outro provedor de serviços. Nesse caso, para preservar a integridade da mensagem, os seus dados não devem ser alterados.

6.2.2 Remetente inicial

Responsável por iniciar a transmissão da mensagem. Existe esse termo para diferenciar dos intermediarios, uma vez que vários encaminhamentos podem existir.

6.2.3 Receptor final

É o último a receber a mensagem. Representa o destino final de uma mensagem e pode ser considerado provedor de serviço.

6.3 XML - Extensible Markup Language

É a base dos web services pois é onde contém toda a mensagem. O XML fornece a descrição, o armazenamento, o formato da transmissão para trocar os dados através dos Web Services e também para criar tecnologias Web Services para a troca dos dados. Os Web Services decodificam as várias partes de XML para interagir com as várias aplicações

6.4 SOA

É um conceito de arquitetura que permite criar, padronizar, documentar serviços únicos e interoperáveis que possam ser reutilizados por aplicações diferentes, sem a necessidade de se dsenvolver a aplicação novamente. Também possibilita o fornecimento de uma nova geração de aplicações dinâmicas. Que é um serviço composto de vários outros pequenos serviços onde todos buscam uma resposta pelo serviço solicitado pelo usuário.

6.5 SOAP - Service Oriented Architecture Protocol

O Protocolo de Acesso do Objeto Simples (Simple Object Access Protocol, SOAP) é um protocolo leve proposto ao World Wide Web consortium por um conjunto de projetistas da Microsoft, IBM e outras empresas. O SOAP fornece três dispositivos: um mecarnismo de envelope, que permite a um emissor designar os conteúdos, o receptor pretendido e o status da mensagem, um conjunto de regras de codificação, que especificam como codificar a mensagem para transferência, e um mecanismo RPC independente da linguagem, que permite a uma chamada especificar o procedimento remoto a ser invocado.

6.6 WSDL

É a sigla de Web Services Description Language, padrão baseado em XML para descrever o serviço como no COM, onde ele traz os métodos do Web Service. Funciona como uma espécie de "TypeLibrary"do Web Service, além de ser usado para a validação das chamadas dos métodos. O WSDL (Web Services Description Language) é uma especificação desenvolvida pelo W3C que permite descrever os Web Services segundo um formato XML.

WSDL é extensível para permitir a descrição dos serviços e suas mensagens, independentemente dos formatos de mensagem e dos protocolos de rede que sejam usados. No entanto, é comum usar-se o MIME (Multipurpose Internet Mail Extensions) e o HTtp://SOAP. Descreve os serviços disponibilizados à rede através de uma semântica XML, este providencia a documentação necessária para se chamar um sistema distribuído e o procedimento necessário para que esta comunicação se estabeleça. Enquanto que o SOAP especifica a comunicação entre um cliente e um servidor, o WSDL descreve os serviços oferecidos.

6.7 UDDI

É um protocolo desenvolvido para a organização e registro de Web Services. O UDDI (Universal Description Discovery and Integration) é uma iniciativa em desenvolvimento no âmbito do consórcio industrial UDDI promovido originalmente pela IBM, Microsoft e Arriba, com objectivo de acelerar a interoperabilidade e utilização dos Web Services, pela proposta de um serviço de registro de nomes de organizações e de descrição do serviço. UDDI nada mais é do que um serviço de diretório onde empresas podem registrar (publicar) e buscar (descobrir) por serviços Web (Web Services).

um Rregistro UDDI contém três tipos de informação: informações gerais de cada organização, tais como o nome, endereço e contatos; informações de organizações e serviços por categorias de negócios; informações técnicas sobre os serviços providenciados pelas organizações. O UDDI providencia três funções principais, conhecidas como publicação, descoberta e ligação:

1) publicação: permite que uma organização divulgue o(s) seu(s) serviço(s);

2) descoberta: permite que o cliente do serviço, procure e encontre um determinado serviço;

3) ligação (bind): permite que o cliente do serviço, possa estabelecer a ligação e interagir com o serviço.

6.8 Corba

Talvez o middleware orientado ao objecto mais bem conhecido, o Common Object Request Broker Architecture (CORBA), permite a colocação de objectos num servidor e estende a invocação de método utilizando a mesma abordagem geral da já aqui descrita, Uma diferença no entanto reside em que quando um programa recebe uma referencia a um objecto remoto, é criado um proxy local que corresponde ao objeto. Quando o programa invoca um método no objecto, o controlo passa para o proxy local. O proxy então envia uma mensagem sobre a rede para o servidor, o qual invoca o método especificado e retorna os resultados. Assim, o CORBA faz parecer idêntica a invocação de método para objectos locais e remotos. Para além de ser focado em objectos em lugar de procedimentos, o CORBA difere das tecnologias RPC convencionais pelo facto de ser mais dinamico. Com efeito com RPCs são usadas ferramentas que criam procedimentos "stub" e associam-nos ao programa. Com o CORBA o software cria um proxy em "run-time", sempre que ele é necessário, ou seja quando um método é invocado num objecto remoto para o qual não existe proxy.