



Nomes desempenham um papel muito importante em todos os sistemas de computadores. Eles são usados para compartilhar recursos, identificar entidades de maneira única, fazer referência a localizações e outras funções. Uma questão importante na nomeação é que um nome pode ser resolvido para a entidade à qual se refere. Portanto, a resolução de nomes permite que um processo acesse a entidade nomeada. Para resolver nomes é necessário implementar um sistema de nomeação. As diferenças entre nomeação em sistemas distribuídos e sistemas não distribuídos se encontram no modo como são implantados os sistemas de nomeação.

Em um sistema distribuído, a implementação de um sistema de nomeação costuma ser, ela mesma, distribuída por várias máquinas. O modo como essa distribuição é feita desempenha um papel fundamental na eficiência e escalabilidade do sistema de nomeação. Neste capítulo, vamos nos concentrar em três modos diferentes e importantes pelos quais nomes são usados em sistemas distribuídos.

Em primeiro lugar, após discutirmos algumas questões gerais referentes à nomeação, vamos examinar mais de perto a organização e implementação de nomes amigáveis para seres humanos. Exemplos típicos desses nomes são os nomes de sistemas de arquivo e os utilizados na World Wide Web. Construir sistemas de nomeação de âmbito mundial e escaláveis é uma preocupação primária para esses tipos de nomes.

Em segundo lugar, nomes são usados para localizar entidades de um modo que independe de sua localização no instante em questão. Ocorre que sistemas de nomeação com nomes amigáveis a seres humanos não são particularmente adequados para suportar esse tipo de monitoração de entidades. A maioria dos nomes nem ao menos sugere a localização da entidade. São necessárias organizações alternativas, como as que são usadas para a telefonia móvel — na qual os nomes são identificadores independentes de localização — e as utilizadas para tabelas de hash distribuídas.

Por fim, os seres humanos muitas vezes preferem descrever entidades por meio de várias características, o que resulta em uma situação em que precisamos resolver uma descrição por intermédio de atributos para uma enti-

dade que corresponda àquela descrição. Esse tipo de resolução de nomes é notoriamente difícil e daremos atenção especial a ele.

5.1 Nomes, Identificadores e Endereços

Vamos começar examinando mais de perto o que é realmente um nome. Um nome em um sistema distribuído é uma cadeia de bits ou caracteres usada para referenciar uma entidade. Uma entidade em um sistema distribuído pode ser praticamente qualquer coisa. Entre os exemplos típicos estão recursos como hospedeiros, impressoras, discos e arquivos. Outros exemplos bem conhecidos de entidades que costumam ser nomeadas explicitamente são processos, usuários, caixas postais, grupos de discussão, páginas Web, janelas gráficas, mensagens, conexões de rede e assim por diante.

Entidades são ativas. Por exemplo, um recurso como uma impressora oferece uma interface que contém operações para imprimir um documento, requisitar o estado de um serviço de impressão e coisas semelhantes. Além do mais, uma entidade como uma conexão de rede pode fornecer operações para enviar e receber dados, ajustar parâmetros de qualidade de serviço, requisitar estado e assim por diante.

Para agir sobre uma entidade, é necessário acessá-la e, para isso, precisamos de um **ponto de acesso**. Um ponto de acesso é mais um outro tipo de entidade, embora especial, em um sistema distribuído. O nome de um ponto de acesso é denominado **endereço**. O endereço de um ponto de acesso de uma entidade também é denominado simplesmente endereço daquela entidade.

Uma entidade pode oferecer mais de um ponto de acesso. Para exemplificar: um telefone pode ser visto como um ponto de acesso de uma pessoa, ao passo que o número do telefone corresponde a um endereço. Na verdade, hoje em dia muitas pessoas têm diversos números de telefone, e cada número corresponde a um ponto em que elas podem ser alcançadas. Em um sistema distribuído, um exemplo típico de um ponto de acesso é um hos-

pedeiro que executa em um servidor específico, cujo endereço é formado pela combinação de, por exemplo, um endereço IP e um número de porta, isto é, o endereço de nível de transporte do servidor.

Uma entidade pode mudar seus pontos de acesso ao longo do tempo. Por exemplo, quando um computador móvel muda para uma outra localização, muitas vezes recebe um endereço IP diferente daquele que tinha antes. Da mesma maneira, quando uma pessoa se muda para uma outra cidade ou país, muitas vezes é preciso mudar também os números de seus telefones. De modo semelhante, mudar de emprego ou de provedor de serviços de Internet significa mudar seu endereço de e-mail.

Portanto, um endereço é apenas um tipo especial de nome: ele se refere a um ponto de acesso de uma entidade. Como um ponto de acesso está fortemente associado com uma entidade, pode parecer conveniente usar o endereço de um ponto de acesso como um nome comum para a entidade associada. No entanto, isso raramente é feito porque tal nomeação em geral é muito inflexível e frequentemente não é amigável para seres humanos.

Por exemplo, não é incomum reorganizar periodicamente um sistema distribuído, de modo que, agora, um servidor específico está executando em um hospedeiro diferente daquele em que executava antes. A antiga máquina na qual o servidor costumava executar pode ser designada a um servidor completamente diferente. Em outras palavras, uma entidade pode mudar facilmente um ponto de acesso, ou um ponto de acesso pode ser designado a uma entidade diferente. Se for usado um endereço para referenciar uma entidade, teremos uma referência inválida no momento em que o ponto de acesso mudar ou for designado a uma outra entidade. Portanto, é muito melhor deixar que um serviço seja conhecido por um nome separado, independente do endereço do servidor associado.

Da mesma maneira, se uma entidade oferecer mais do que um ponto de acesso, não fica claro qual endereço usar como referência. Por exemplo, muitas organizações distribuem seus serviços Web por diversos servidores. Se usássemos os endereços desses servidores como uma referência para o serviço Web, não ficaria óbvio qual endereço deveria ser escolhido como o melhor. Mais uma vez, uma solução muito melhor é ter um único nome para o serviço Web, independente dos diferentes servidores Web.

Esses exemplos ilustram que um nome para uma entidade, que seja independente dos endereços dessa entidade, freqüentemente é muito mais fácil e flexível de usar. Tal nome é denominado **independente de localização**.

Além de endereços, há outros tipos de nomes que merecem tratamento especial, como nomes que são usados para identificar exclusivamente uma entidade. Um **identificador** verdadeiro é um nome que tem as seguintes propriedades (Wieringa e De Jonge, 1995):

1. Um identificador referencia, no máximo, uma entidade

2. Cada entidade é referenciada por, no máximo, um identificador
3. Um identificador sempre referencia a mesma entidade, isto é, nunca é reutilizado.

Usando identificadores, fica muito mais fácil referenciar uma entidade sem nenhuma ambigüidade. Por exemplo, suponha que cada um de dois processos referencie uma entidade por meio de um identificador. Para verificar se os processos estão referenciando a mesma entidade, basta testar se os dois identificadores são iguais. Esse teste não seria suficiente se os dois processos estivessem usando nomes normais, não exclusivos e não identificados. O nome ‘John Smith’ não pode ser considerado como uma referência exclusiva a uma única pessoa.

Da mesma maneira, se um endereço puder ser designado novamente a uma entidade diferente, não podemos usar um endereço como identificador. Considere a utilização de números de telefone que sejam razoavelmente estáveis, no sentido de que um número de telefone referencia durante algum tempo a mesma pessoa ou organização. Contudo, usar um número de telefone como identificador não funcionará, porque ele pode ser designado novamente a outra pessoa ao longo do tempo. Em decorrência, a nova padaria de Bob pode receber telefonemas feitos para o antiquário de Alice por um longo tempo. Nesse caso, teria sido melhor usar um identificador verdadeiro para Alice em vez do número de seu telefone.

Endereços e identificadores são dois tipos importantes de nomes, e cada um deles é usado para finalidades muito diferentes. Em muitos sistemas de computadores, endereços e identificadores são representados sob uma forma que só pode ser lida por uma máquina, isto é, sob a forma de cadeias de bits. Por exemplo, um endereço de Ethernet é, em essência, uma cadeia aleatória de 48 bits. Do mesmo modo, endereços de memória costumam ser representados como cadeias de 32 bits ou 64 bits.

Um outro tipo importante de nome é aquele construído para ser utilizado por seres humanos, também denominado **nome amigável a seres humanos**. Ao contrário de endereços e identificadores, um nome amigável a seres humanos em geral é representado por uma cadeia de caracteres. Esses nomes aparecem sob variadas formas. Por exemplo, os nomes de arquivos em sistemas Unix são cadeias de caracteres cujo comprimento pode chegar a 255 caracteres, sendo definidos inteiramente pelo usuário. De modo semelhante, nomes DNS são representados como cadeias de caracteres relativamente simples, nas quais letras maiúsculas ou minúsculas são indistintas.

Ter nomes, identificadores e endereços nos conduz ao tema central deste capítulo: como resolvemos nomes e identificadores para endereços? Antes de analisar várias soluções, é importante entender que muitas vezes há uma estreita relação entre resolução de nomes em sistemas distribuídos e roteamento de mensagens. Em prin-

cípio, um sistema de nomeação mantém uma vinculação **nome–endereço** que, em sua forma mais simples, é apenas uma tabela de pares (*nome, endereço*). Contudo, em sistemas distribuídos que abrangem redes de grande porte e para os quais há muitos recursos a nomear, uma tabela centralizada não vai funcionar.

Ao contrário, o que costuma acontecer é que um nome é decomposto em várias partes como *ftp.cs.vu.nl*, e a resolução do nome é realizada por meio de consulta recursiva dessas partes. Por exemplo, um cliente que precisa saber o endereço do servidor FTP cujo nome é *ftp.cs.vu.nl* em primeiro lugar resolveria *nl* para achar o servidor *NS(nl)* responsável por nomes que terminam com *nl*; em seguida, o resto do nome é passado para o servidor *NS(nl)*. Em seguida esse servidor pode resolver o nome *vu* indicando o servidor *NS(vu.nl)* responsável por nomes que terminam com *vu.nl* e que pode manipular a parte restante do nome, *ftp.cs*. A certa altura, isso resulta no roteamento da requisição de resolução de nome como:

NS(.) → NS(nl) → NS(vu.nl) → endereço de ftp.cs.vu.nl

onde *NS(.)* mostra o servidor que pode retornar o endereço de *NS(nl)*, também conhecido como **servidor-raiz**. *NS(vu.nl)* retornará o endereço real do servidor FTP. É interessante observar que as fronteiras entre resolução de nomes e roteamento de mensagens começam a ficar indistintas.

Nas próximas seções vamos considerar três classes diferentes de sistemas de nomeação. Em primeiro lugar, vamos estudar como identificadores podem ser resolvidos para endereços. Nesse caso, também veremos um exemplo em que a resolução de nomes é realmente indistinguível do roteamento de mensagens. Depois disso, consideraremos nomes amigáveis a seres humanos e nomes descritivos, ou seja, entidades que são descritas por um conjunto de nomes.

5.2 Nomeação Simples

Já explicamos que identificadores são convenientes para representar entidades com exclusividade. Em muitos casos, identificadores são simplesmente cadeias aleatórias de bits às quais nos referimos, por conveniência, como nomes não estruturados, ou simples. Uma propriedade importante de tal nome é que ele não contém sequer uma informação sobre como localizar o ponto de acesso de sua entidade associada. Na discussão a seguir, estudaremos como nomes simples podem ser resolvidos, ou, o que é equivalente, como podemos localizar uma entidade quando temos somente seu identificador.

5.2.1 Soluções simples

Em primeiro lugar, considere duas soluções simples para localizar uma entidade. Ambas são aplicáveis somente a redes locais. Apesar disso, elas costumam funcionar bem nesse ambiente, o que torna sua simplicidade particularmente atraente.

Broadcasting e multicasting

Considere um sistema distribuído construído em cima de uma rede de computadores que ofereça recursos eficientes de broadcasting. Normalmente esses recursos são oferecidos por redes locais nas quais todas as máquinas estão conectadas a um único cabo ou a seu equivalente lógico. Além disso, redes locais sem fio caem nessa categoria.

Localizar uma entidade nesse ambiente é simples: uma mensagem que contém o identificador da entidade é enviada por broadcast a cada máquina da rede e cada uma delas deve verificar se tem essa entidade. Somente as máquinas que podem oferecer um ponto de acesso para a entidade enviam uma mensagem de resposta que contém o endereço daquele ponto de acesso.

Esse princípio é utilizado no **protocolo de resolução de endereços** (Address Resolution Protocol — ARP) da Internet para achar o endereço de enlace de uma máquina quando é dado apenas um endereço IP (Plummer, 1982). Em essência, uma máquina faz uma transmissão broadcast de um pacote na rede local perguntando quem é o dono de determinado endereço IP. Quando a mensagem chega a uma máquina, o receptor verifica se deve ouvir o endereço IP requisitado. Caso positivo, envia um pacote de resposta que contém, por exemplo, seu endereço Ethernet.

Broadcasting se torna ineficiente quando a rede cresce. Não somente a largura de banda de rede é desperdiçada por mensagens de requisição, mas também, o que é mais sério, um número muito grande de hospedeiros pode ser interrompidos por requisições às quais não pode responder. Uma possível solução é passar para multicasting, pelo qual somente um grupo restrito de hospedeiros recebe a requisição. Por exemplo, redes Ethernet suportam multicasting de nível de enlace diretamente em hardware.

Multicasting também pode ser usado para localizar entidades em redes ponto-a-ponto. Por exemplo, a Internet suporta multicasting de nível de rede permitindo que hospedeiros se juntem a grupos multicast específicos. Esses grupos são identificados por um endereço multicast. Quando um hospedeiro envia uma mensagem a um endereço multicast, a camada de rede fornece um serviço de melhor esforço para entregar aquela mensagem a todos os membros do grupo. Implementações eficientes para multicasting na Internet são discutidas em Deering e Cheriton (1990) e em Deering et al. (1996).

Um endereço multicast pode ser usado como serviço geral de localização para várias entidades. Por exemplo,

considere uma organização na qual cada empregado tenha seu próprio computador móvel. Quando um desses computadores se conecta com a rede disponível no local, recebe dinamicamente um endereço IP a ele designado. Além disso, ele se junta a um grupo multicast específico. Quando um processo quer localizar o computador A, envia uma requisição ‘onde está A?’ ao grupo multicast. Se A estiver conectado, ele responde com seu endereço IP corrente.

Um outro modo de usar um endereço multicast é associá-lo com uma entidade replicada e usar multicasting para localizar a réplica *mais próxima*. Quando é enviada uma requisição para o endereço multicast, cada réplica responde com seu endereço IP (normal) corrente. Um modo grosseiro de selecionar a réplica mais próxima é escolher aquela cuja resposta chegar antes. Discutiremos outros modos em capítulos posteriores. Ocorre que, em geral, selecionar a réplica mais próxima não é assim tão fácil.

Ponteiros repassadores

Uma outra abordagem popular para a localização de entidades móveis é utilizar ponteiros repassadores (Fowler, 1985). O princípio é simples: quando uma entidade se move de *A* para *B*, deixa para trás, em *A*, uma referência a sua nova localização em *B*. A principal vantagem dessa abordagem é sua simplicidade: tão logo uma entidade seja localizada, por exemplo, usando um serviço tradicional de nomeação, um cliente pode consultar o endereço corrente da entidade percorrendo uma cadeia de ponteiros repassadores.

Também há algumas desvantagens importantes. Em primeiro lugar, se não forem tomadas providências especiais, uma cadeia para uma entidade de alta mobilidade pode se tornar tão longa que localizar aquela entidade tenha um custo proibitivo. Em segundo lugar, todas as localizações intermediárias em uma cadeia terão de manter sua parte da cadeia de ponteiros repassadores pelo tempo que for necessário. Uma terceira e relacionada desvantagem é a vulnerabilidade de enlaces rompidos. Tão logo qualquer ponteiro repassador seja perdido (por qualquer razão), a entidade não pode mais ser alcançada. Portanto, uma ques-

tão importante é manter cadeias relativamente curtas e garantir que os ponteiros repassadores sejam robustos.

Para entender melhor como ponteiros repassadores funcionam, considere sua utilização em relação a objetos remotos: objetos que podem ser acessados por meio de uma chamada de procedimento remoto. Segundo a abordagem em **cadeias SSP (Stub Scion Pairs)** (Shapiro et al., 1992), cada ponteiro repassador é implementado como um par (*apêndice (stub) de cliente, apêndice de servidor*), conforme mostra a Figura 5.1. (Observamos que, na terminologia original de Shapiro, um apêndice de servidor era denominado *scion*; como lidava com pares (*stubscion*), tal fato veio justificar o nome da técnica.) Um apêndice de servidor contém ou uma referência local ao objeto propriamente dito ou uma referência local a um apêndice de cliente remoto para aquele objeto.

Sempre que um objeto se move do espaço de endereço *A* para *B*, deixa para trás, em seu lugar, um apêndice de cliente em *A* e instala um apêndice de servidor que referencia aquele apêndice em *B*. Um aspecto interessante dessa abordagem é que a migração é completamente transparente para um cliente. A única coisa que o cliente vê de um objeto é um apêndice de cliente. O modo como esse apêndice de cliente repassa suas invocações, e para qual localização, ficam ocultos do cliente. Além disso, note que essa utilização de ponteiros repassadores não é como consultar um endereço. Em vez disso, uma requisição de cliente é repassada ao longo da cadeia até o objeto propriamente dito.

Para tomar um atalho em uma cadeia de pares (*apêndice de cliente, apêndice de servidor*), uma invocação de objeto transporta a identificação do apêndice de cliente de onde essa invocação foi iniciada. Uma identificação do apêndice de cliente consiste no endereço de nível de transporte do cliente, combinado com um número gerado no local para identificar aquele apêndice. Quando a invocação chega ao objeto em sua localização corrente, uma resposta é enviada de volta ao apêndice de cliente onde a invocação foi iniciada (muitas vezes sem voltar pela cadeia). A localização corrente pega uma carona nessa

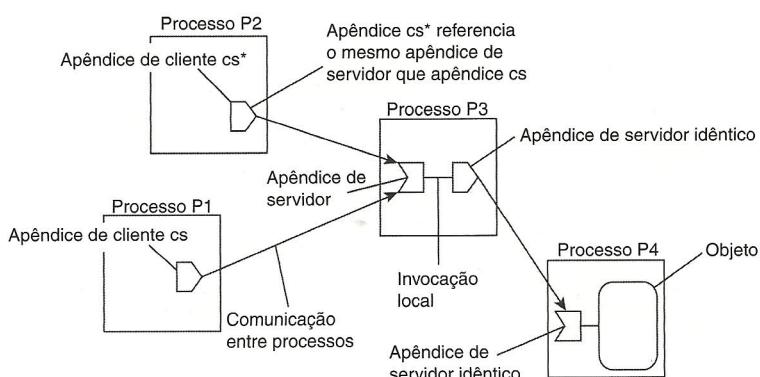


Figura 5.1 Princípio de ponteiros repassadores que utilizam pares (*apêndice de cliente, apêndice de servidor*).

resposta e o apêndice de cliente ajusta seu apêndice de servidor correspondente para a localização corrente do objeto. Esse princípio é mostrado na *Figura 5.2*.

Há um compromisso entre enviar a resposta diretamente ao apêndice de cliente iniciador ou ao longo do caminho inverso de ponteiros repassadores. No primeiro caso, a comunicação é mais rápida porque pode haver um número menor de processos pelos quais a resposta precisa passar. Por outro lado, somente o apêndice de cliente iniciador pode ser ajustado, ao passo que enviar a resposta ao longo do caminho inverso permite o ajuste de todos os apêndices intermediários.

Quando um apêndice de servidor não é mais referenciado por nenhum cliente, ele pode ser removido. Isso, por si só, está fortemente relacionado com coleta distribuída de lixo, um problema que, em geral, está longe de ser trivial e que não discutiremos aqui. O leitor interessado pode consultar Abdullahi e Ringwood (1998), Plainfosse e Shapiro (1995) e Veiga e Ferreira (2005).

Agora, suponha que o processo P_1 na Figura 5.1 passe sua referência para o objeto ao processo P_2 . A passagem de referência é feita por meio da instalação de uma cópia p' do apêndice de cliente p no espaço de endereço do processo P_2 . O apêndice de cliente p' referencia o mesmo apêndice de servidor que p , de modo que o mecanismo de repasse de invocação funciona do mesmo modo que antes.

Surgem problemas quando um processo em uma cadeia de pares (*apêndice de cliente*, *apêndice de servidor*) cai ou se torna inalcançável por qualquer outra razão. Há diversas soluções possíveis. Uma possibilidade, seguida por Emerald (Jul et al., 1988) e pelo sistema LII (Black e Artsy, 1990), é deixar que a máquina em que um objeto foi criado (denominada **localização nativa** do objeto) sempre mantenha uma referência a sua localização corrente. Essa referência é armazenada e mantida de modo tolerante a falha. Quando uma cadeia é rompida, pergunta-se à localização nativa do objeto onde ele está agora. Para permitir que a localização nativa de um objeto mude, pode ser usado um serviço tradicional de nomeação para registrar a localização nativa corrente. Essas abordagens baseadas na localização nativa serão discutidas a seguir.

5.2.2 Abordagens baseadas na localização nativa

A utilização de broadcasting e ponteiros repassadores impõe problemas de escalabilidade. É difícil implementar com eficiência broadcasting ou multicasting em redes de grande escala, ao passo que longas cadeias de ponteiros repassadores introduzem problemas de desempenho e são suscetíveis a enlaces interrompidos.

Uma abordagem popular para suportar entidades móveis em rede de grande escala é introduzir uma **localização nativa**, que monitora a localização corrente de uma entidade. Técnicas especiais podem ser aplicadas para proteção contra falhas de rede ou de processo. Na prática, a localização nativa costuma ser escolhida como o lugar em que a entidade foi criada.

A abordagem da localização nativa é usada como mecanismo de emergência para localização de serviços baseada em ponteiros repassadores, que já discutimos. Um outro exemplo em que a abordagem da localização nativa é seguida é em Mobile IP (Johnson et al., 2004), que examinamos brevemente no Capítulo 3. Cada hospedeiro móvel usa um endereço IP fixo. Toda a comunicação para aquele endereço IP é inicialmente dirigida ao **agente nativo** do hóspede móvel. Esse agente nativo está situado na rede local correspondente ao endereço de rede contido no endereço IP do hospedeiro móvel. No caso do IPv6, ele é realizado como um componente da camada de rede. Sempre que um hospedeiro móvel passar para uma outra rede, ele requisita um endereço temporário que possa usar para comunicação. Esse **endereço externo (care-of address — COA)** é registrado no agente nativo.

Quando o agente nativo recebe um pacote para o hospedeiro móvel, ele consulta a localização corrente do hospedeiro. Se este estiver na rede local corrente, o pacote é simplesmente repassado. Caso contrário, o pacote é enviado por um túnel até a localização corrente do hospedeiro, isto é, envelopado como dados em um pacote IP e enviado para o endereço COA. Ao mesmo tempo, o remetente do pacote é informado da localização corrente do hospedeiro. Esse princípio é mostrado na *Figura 5.3*. Note

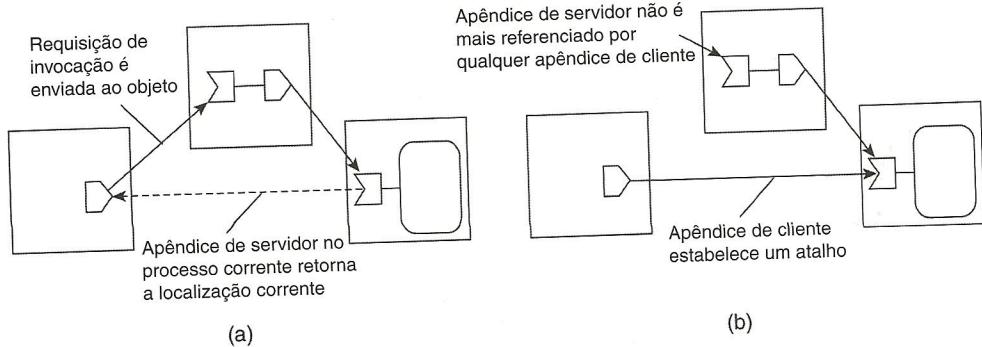


Figura 5.2 Redirecionamento de ponteiro repassador pelo armazenamento de um atalho no apêndice de cliente.

que o endereço IP é efetivamente utilizado como identificador pelo hospedeiro móvel.

A Figura 5.3 também ilustra uma outra desvantagem de abordagens baseadas em localização nativa para redes de grande escala. Para se comunicar com uma entidade móvel, em primeiro lugar um cliente tem de contatar a localização nativa, que pode estar em um lugar completamente diferente de onde está a própria entidade. O resultado é um aumento na latência de comunicação.

Uma desvantagem da abordagem baseada na localização nativa é a utilização de uma localização nativa fixa. Por um lado, é preciso assegurar que a localização nativa sempre exista. Caso contrário, será impossível contatar a entidade. Os problemas se agravam quando uma entidade que está em determinado lugar há muito tempo decide se mudar permanentemente para uma parte da rede completamente diferente de onde está sua localização nativa. Nesse caso, seria melhor se a localização nativa pudesse se mudar junto com o hospedeiro.

Uma solução para esse problema é registrar a localização nativa em um serviço tradicional de nomeação e deixar que um cliente consulte, em primeiro lugar, a localização nativa. Como podemos adotar a premissa de que a localização nativa é relativamente estável, essa localização pode ser efetivamente mantida em cache após ter sido consultada.

5.2.3 Tabelas de hash distribuídas [DHT]

Agora vamos examinar mais de perto recentes desenvolvimentos sobre como resolver um identificador para o endereço da entidade associada. Já mencionamos tabelas de hash distribuídas várias vezes, mas adiamos a discussão sobre o modo como elas realmente funcionam. Nesta seção, corrigimos essa situação considerando, em

primeiro lugar, o sistema Chord como um sistema baseado em DHT fácil de explicar. Em sua forma mais simples, sistemas baseados em DHT não consideram proximidade de rede. Essa negligência pode resultar facilmente em problemas de desempenho. Discutimos também soluções para sistemas de equipamentos de rede.

Mecanismo geral

Existem vários sistemas baseados em DHT, e uma breve visão geral desses sistemas é dada em Balakrishnan et al. (2003). O sistema Chord (Stoica et al., 2003) é representativo de muitos deles, embora apresente importantes diferenças sutis que influenciam a complexidade de sua manutenção e de seus protocolos de consulta. Como explicamos brevemente no Capítulo 2, o Chord usa um espaço de identificadores de m bits para designar identificadores escolhidos aleatoriamente a nós, bem como chaves a entidades específicas. Estas podem ser praticamente qualquer coisa: arquivos, processos etc. O número m de bits é usualmente 128 ou 160, dependendo da função de hash utilizada. Uma entidade com chave k cai sob a jurisdição do nó que tenha o menor identificador $id \geq k$. Esse nó é denominado *sucessor* de k e denotado por $succ(k)$.

A questão principal em sistemas baseados em DHT é resolver com eficiência uma chave k para o endereço de $succ(k)$. Uma abordagem óbvia não escalável é deixar que cada nó p monitore o sucessor $succ(p+1)$, bem como seu predecessor, $pred(p)$. Nesse caso, sempre que um nó p recebe uma requisição para resolver a chave k , ele simplesmente repassará a requisição para um de seus dois vizinhos — qualquer um que seja adequado —, a menos que $pred(p) < k \leq p$, caso em que o nó p deve retornar seu próprio endereço para o processo que iniciou a resolução da chave k .

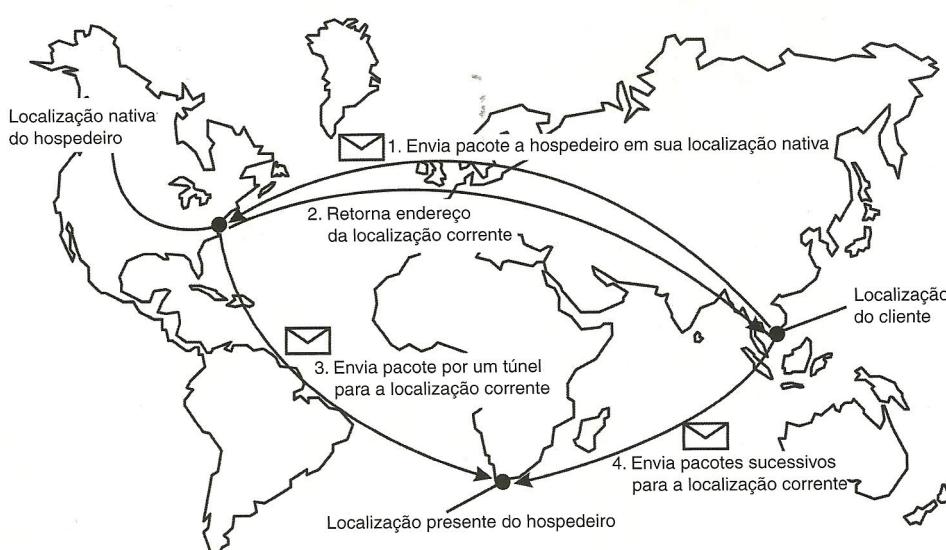


Figura 5.3 Princípio do Mobile IP.

Em vez dessa abordagem linear para a consulta de chaves, cada nó Chord mantém uma **tabela de derivação** (**finger table**) de, no máximo, m entradas. Denotando a tabela de derivação do nó p por FT_p , então:

$$FT_p[i] = \text{succ}(p + 2^{i-1})$$

Em linguagem corrente, a i -ésima entrada aponta para o primeiro nó que sucede p por, no mínimo, 2^{i-1} . Note que essas referências são, na verdade, atalhos para nós existentes no espaço de identificadores, onde a distância do atalho em relação ao nó p aumenta exponencialmente à medida que o índice na tabela de derivação cresce. Portanto, para consultar uma chave k , o nó p repassará imediatamente a requisição ao nó q com índice j na tabela de derivação de p' , onde:

$$q = FT_p[j] \leq k \leq FT_p[j+1]$$

(Por clareza, ignoramos a aritmética modular.)

Para ilustrar essa consulta, considere a resolução de $k = 26$ a partir do nó 1, como mostra a Figura 5.4. Em primeiro lugar, o nó 1 consultará $k = 26$ em sua tabela de derivação para verificar se esse valor é maior do que $FT_1[5]$, o que significa que a requisição será repassada para o nó 18 = $FT_1[5]$. Por sua vez, o nó 18 selecionará o nó 20, porque $FT_{18}[2] < k \leq FT_{18}[3]$.

Por fim, a requisição é repassada do nó 20 para o nó 21, e deste para o nó 28, que é responsável por $k = 26$. Nesse ponto, o endereço do nó 28 é retornado para o nó 1, e a chave foi resolvida. Por razões semelhantes, quando o nó 28 é requisitado a resolver a chave $k = 12$, uma requisição será retornada, como mostra a linha pontilhada da Figura 5.4. Pode-se mostrar que uma consulta geralmente exigirá $O(\log(N))$ etapas, sendo N o número de nós no sistema.

Em grandes sistemas distribuídos, pode-se esperar que o conjunto de nós participantes mude o tempo todo. Não basta considerar apenas os nós que se juntam e saem voluntariamente; é preciso também levar em conta os nós que falham (e, assim, deixam efetivamente o sistema) e mais tarde se recuperam novamente (quando então se juntam à rede mais uma vez).

Juntar-se a um sistema baseado em DHT como o Chord é relativamente simples. Suponha que o nó p queira se juntar. Ele simplesmente contata um nó arbitrário no sistema existente e requisita uma consulta para $\text{succ}(p+1)$. Tão logo esse nó tenha sido identificado, o próprio p pode inserir a si próprio no anel. Da mesma maneira, sair também pode ser simples. Note que os nós ainda monitoram seu predecessor.

É óbvio que a complexidade vem da necessidade de manter as tabelas de derivação atualizadas. O mais importante é que, para todo nó q , $FT_q[1]$ esteja correta, porque essa entrada se refere ao próximo nó do anel, isto é, ao sucessor de $q+1$. Para atingir esse objetivo, cada nó q executa periodicamente um procedimento simples

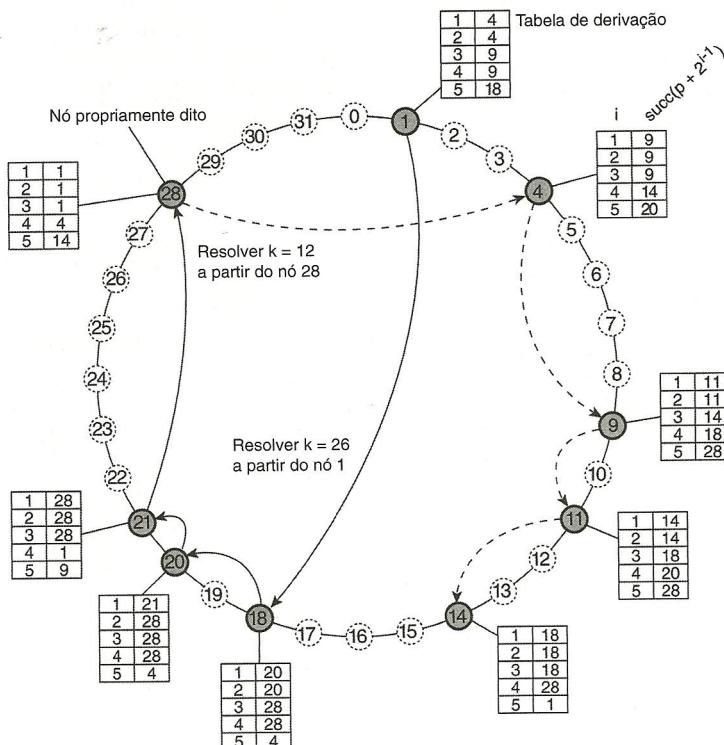


Figura 5.4 Resolução da chave 26 a partir do nó 1 e da chave 12 a partir do nó 28 em um sistema Chord.

que contata $\text{succ}(q+1)$ e requisita que ele retorne $\text{pred}(\text{succ}(q+1))$. Se $q = \text{pred}(\text{succ}(q+1))$, então q sabe que suas informações são consistentes com as de seu sucessor. Ao contrário, se o sucessor de q tiver atualizado seu predecessor, então, aparentemente, um novo nó p entrou no sistema, com $q < p \leq \text{succ}(q+1)$, de modo que q ajustará $FT_q[1]$ para p . Nesse ponto, ele também verificará se p registrou q como seu predecessor. Caso não tenha registrado, é preciso um outro ajuste de $FT_q[1]$.

De modo semelhante, para atualizar uma tabela de derivação, o nó q precisa simplesmente achar o sucessor para $k = q + 2^{i-1}$ para cada entrada i . Mais uma vez, isso pode ser feito pela emissão de uma requisição para resolver $\text{succ}(k)$. Em Chord, tais requisições são emitidas periodicamente por meio de um processo residente.

Da mesma forma, cada nó q verificará periodicamente se seu predecessor está vivo. Se o predecessor tiver falhado, a única coisa que q pode fazer é registrar o fato ajustando $\text{pred}(q)$ para ‘desconhecido’. Por outro lado, quando o nó q estiver atualizando seu enlace para o próximo nó no anel e descobrir que o predecessor de $\text{succ}(q+1)$ foi ajustado para ‘desconhecido’, ele simplesmente avisará $\text{succ}(q+1)$, que suspeita que ele é o predecessor. De modo geral, esses procedimentos simples garantem que um sistema Chord seja, normalmente, consistente, talvez com exceção de alguns nós. Os detalhes podem ser encontrados em Stoica et al. (2003).

Exploração de proximidade na rede

Um dos problemas potenciais de sistemas como o Chord é que as requisições podem ser roteadas erraticamente pela Internet. Por exemplo, suponha que o nó 1 da Figura 5.4 esteja localizado em Amsterdã, Holanda; o nó 18, em San Diego, Califórnia; o nó 20 novamente em Amsterdã; e o nó 21 em San Diego. O resultado da resolução da chave 26 incorrerá em três transferências de mensagens de longo alcance que, argumenta-se, poderiam ser reduzidas no máximo a uma. Para minimizar essas anomalias, o projeto de um sistema baseado em DHT requer que se leve em conta a rede subjacente.

Castro et al. (2002b) distinguem três modos diferentes de fazer com que um sistema baseado em DHT fique ciente da rede subjacente. No caso de **identificadores de nós designados com base na topologia**, a idéia é designar identificadores de modo tal que dois nós próximos tenham identificadores que também estejam próximos um do outro. Não é difícil imaginar que essa abordagem pode impor sérios problemas no caso de sistemas relativamente simples como o Chord. Sob circunstâncias em que identificadores de nós são amostrados com base em um espaço unidimensional, mapear um anel lógico para a Internet está longe de ser trivial. Além do mais, esse mapeamento pode expor, com facilidade, falhas correlacionadas: nós que estão na mesma rede corporativa terão identificadores dentro de um intervalo relativamente

pequeno. Quando essa rede ficar inalcançável, de repente teremos uma lacuna no que, quanto ao mais, seria uma distribuição uniforme de identificadores.

Com **roteamento por proximidade**, os nós mantêm uma lista de alternativas para repassar uma requisição. Por exemplo, em vez de ter só um único sucessor, cada nó em Chord poderia perfeitamente bem monitorar r sucessores. Na verdade, essa redundância pode ser aplicada para toda entrada em uma tabela de derivações. Para o nó p , $FT_p[i]$ aponta para o primeiro nó na faixa $[p+2^{i-1}, p+2^i-1]$. Não há nenhuma razão por que p não possa monitorar r nós naquela faixa: se for necessário, cada um deles pode ser usado para rotear uma requisição de consulta para uma chave $k > p+2^i-1$. Nesse caso, quando um nó está escolhendo um outro nó para repassar uma requisição de consulta, ele pode optar por um dos r sucessores que estão mais próximos dele mesmo, mas também satisfaz a restrição de que o identificador do nó escolhido deva ser menor do que o da chave requisitada. Uma vantagem adicional de ter vários sucessores para cada entrada de tabela é que falhas de nós não precisam resultar imediatamente na falha de consultas porque várias rotas podem ser exploradas.

Por fim, na **seleção de vizinho por proximidade**, a idéia é otimizar tabelas de roteamento de maneira tal que o nó mais próximo seja selecionado como vizinho. Essa seleção só funciona quando há mais nós entre os quais escolher. Em Chord, isso normalmente não se aplica. Contudo, em outros protocolos, como Pastry (Rowstron e Druschel, 2001), quando um nó se junta ao grupo, recebe informações sobre a sobreposição corrente de vários outros nós. Essa informação é usada pelo novo nó para construir uma tabela de roteamento. É óbvio que, quando há nós alternativos entre os quais escolher, a seleção de vizinho por proximidade permitirá ao nó que está se juntando ao grupo escolher o melhor deles.

Note que pode não ser muito fácil separar roteamento por proximidade e seleção de vizinho por proximidade. Na verdade, quando o Chord é modificado para incluir r sucessores para cada entrada de tabela de derivações, a seleção de vizinho por proximidade recorre à identificação dos r vizinhos mais próximos, o que chega muito perto do roteamento por proximidade, como acabamos de explicar (Dabek et al., 2004b).

Por fim, observamos também que se pode fazer uma distinção entre **consultas iterativas** e **consultas recursivas**. No primeiro caso, um nó ao qual é requisitada uma consulta de chave retornará ao processo requisitante o endereço de rede do próximo nó encontrado. Portanto, o processo requisitará que o próximo nó avance uma etapa na resolução da chave. Uma alternativa, e, em essência, o modo como o explicamos até aqui, é deixar que um nó repasse uma requisição de consulta para o próximo nó. Ambas as abordagens têm suas vantagens e desvantagens, que estudaremos mais adiante neste capítulo.

5.2.4 Abordagens hierárquicas

Nesta seção, em primeiro lugar discutiremos uma abordagem geral de um esquema de localização hierárquica e depois apresentaremos várias otimizações. A abordagem que apresentaremos é baseada no serviço de localização Globe, descrito em detalhes em Ballintijn (2003). Uma visão geral pode ser encontrada em Van Steen et al. (1998). O Globe é um serviço de localização de uso geral representativo de muitos serviços de localização hierárquica propostos para o que denominamos Sistemas de Comunicação Pessoal, dos quais Pitoura e Samaras (2001) dão uma visão geral.

Em um esquema hierárquico, uma rede é dividida em um conjunto de **domínios**. Há um único domínio de nível mais alto que abrange toda a extensão da rede. Cada domínio pode ser subdividido em vários domínios menores. Um domínio de nível mais baixo é denominado **domínio-folha** e normalmente corresponde a uma rede local quando se trata de redes de computadores ou a uma célula em uma rede de telefonia móvel.

Cada domínio D tem um nó de diretório associado, $dir(D)$, que monitora as entidades nesse domínio. Isso resulta em uma árvore de nós de diretório. O nó de diretório do domínio de nível mais alto, denominado **nó (de diretório) raiz**, sabe quais são todas as entidades. Essa

organização geral de uma rede em domínios e nós de diretório é ilustrada na Figura 5.5.

Para monitorar o paradeiro de uma entidade, cada entidade que está localizada em um domínio D no momento considerado é representada por um **registro de localização** no nó de diretório $dir(D)$. Um registro de localização para a entidade E no nó de diretório N para um domínio-folha D contém o endereço corrente dessa entidade naquele domínio. Em comparação, o nó de diretório N' para o próximo domínio de nível mais alto, D' , que contém D , terá um registro de localização para E que contém somente um ponteiro para N . Da mesma maneira, o nó-pai de N' armazenará um registro de localização para E que contém somente um ponteiro para N' . Em decorrência, o nó-raiz terá um registro de localização para cada entidade, e cada registro de localização armazenará um ponteiro para o nó de diretório do próximo subdomínio de nível mais baixo onde a entidade associada àquele registro vai estar localizada no momento em questão.

Uma entidade pode ter vários endereços, por exemplo, se ela for replicada. Se uma entidade tem um endereço no domínio-folha D_1 e D_2 , respectivamente, o nó de diretório do menor domínio que contém ambos, D_1 e D_2 , terá dois ponteiros, um para cada subdomínio que contém um endereço. Isso resulta na organização geral da árvore, como mostra a Figura 5.6.

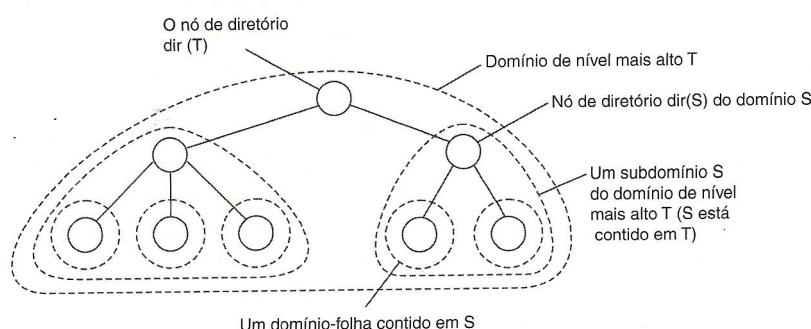


Figura 5.5 Organização hierárquica de um serviço de localização em domínios, cada um com um nó de diretório associado.

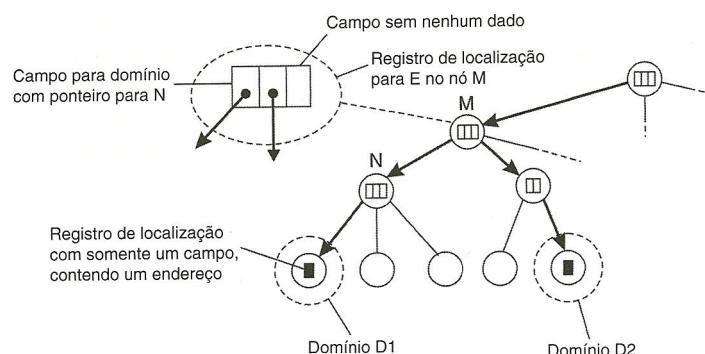


Figura 5.6 Exemplo de armazenamento de informação de uma entidade que tem dois endereços em domínios-folha diferentes.

Agora vamos considerar como ocorre uma operação de consulta em tal serviço de localização hierárquico. Como mostra a Figura 5.7, um cliente que deseja localizar uma entidade E emite uma requisição de consulta ao nó de diretório do domínio-folha D no qual o cliente reside. Se o nó de diretório não armazenar um registro de localização para a entidade, ela não está localizada em D naquele momento. Por consequência, o nó repassa a requisição para seu pai. Note que o nó-pai representa um domínio maior do que o de seu filho. Se o pai também não tiver nenhum registro de localização para E , a requisição de consulta é repassada para o próximo nível mais alto e assim por diante.

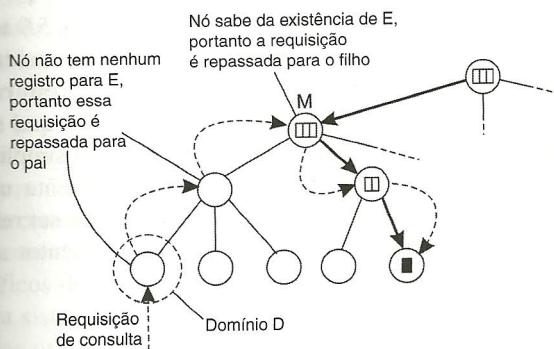


Figura 5.7 Consulta de uma localização em um serviço de localização organizado por hierarquia.

Tão logo a requisição chegue ao nó de diretório M , que armazena um registro de localização para a entidade E , sabemos que E está em algum lugar no domínio $dom(M)$ representado pelo nó M . Podemos ver, na Figura 5.7, que M armazena um registro de localização que contém um ponteiro para um de seus subdomínios. Então, a requisição de consulta é repassada para o nó de diretório daquele subdomínio que, por sua vez, a repassa para baixo pela árvore, até que a requisição finalmente alcance um nó-folha. O registro de localização armazenado no nó-folha conterá o endereço de E naquele domínio-folha.

Então, esse endereço pode ser retornado para o cliente que requisitou inicialmente a consulta.

Uma observação importante em relação a serviços de localização hierárquica é que a operação de consulta explora localidade. Em princípio, a entidade é procurada dentro de um anel que cresce gradativamente e está centrado no cliente requisitante. A área de busca é expandida toda vez que a requisição de consulta é repassada para o próximo diretório de nível mais alto. Na pior das hipóteses, a busca continua até a requisição chegar ao nó-raiz. Como o nó-raiz tem um registro de localização para cada entidade, a requisição pode ser simplesmente repassada para baixo, ao longo de um caminho de ponteiros, até um dos nós-folha.

Operações de atualização exploram localidade de modo semelhante, como mostra a Figura 5.8. Considere uma entidade E que criou uma réplica no domínio-folha D na qual ela precisa inserir seu endereço. A inserção é iniciada no nó-folha $dir(D)$ de D que imediatamente repassa a requisição de inserção a seu pai. O pai também repassará a requisição de inserção, até que ela chegue a um nó de diretório M que já armazena um registro de localização para E .

Portanto, o nó M armazenará um ponteiro no registro de localização para E , que referencia o nó-filho de onde a requisição de inserção foi repassada. Nesse ponto, o nó-filho cria um registro de localização para E , que contém um ponteiro para o próximo nó de nível mais baixo de onde veio a requisição. Esse processo continua até chegar ao nó-folha de onde foi iniciada a inserção. Por fim, o nó-folha cria um registro com o endereço da entidade no domínio-folha associado.

Inserir um endereço como acabamos de descrever resulta na instalação da cadeia de ponteiros de cima para baixo, começando no nó de diretório de nível mais alto que tem o registro de localização para a entidade E . Uma alternativa é criar um registro de localização antes de passar a requisição de inserção para o nó-pai. Em outras palavras, a cadeia de ponteiros é construída de baixo para

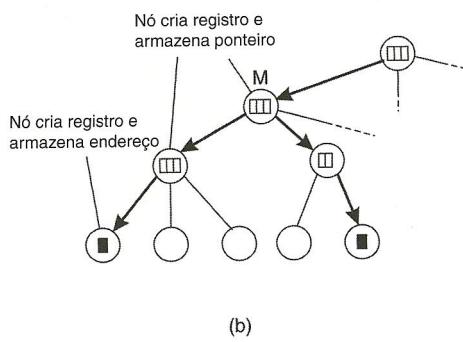
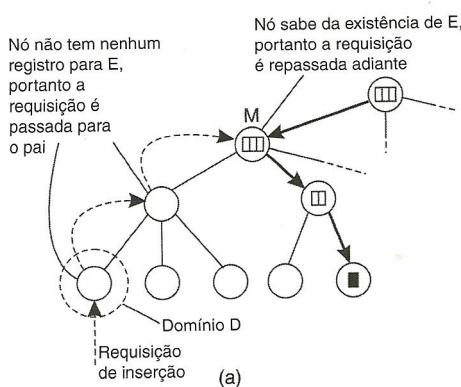


Figura 5.8 (a) Requisição de inserção é repassada para o primeiro nó, que sabe da existência da entidade E .
(b) É criada uma cadeia de ponteiros repassadores até o nó-folha.

cima. A vantagem dessa última é que um endereço se torna disponível para consultas logo que seja possível. Em consequência, se um nó-pai estiver temporariamente inalcançável, o endereço ainda pode ser consultado dentro do domínio representado pelo nó corrente.

Uma operação de remoção é análoga a uma operação de inserção. Quando um endereço para a entidade E no domínio-folha D precisa ser removido, requisita-se ao nó de diretório $dir(D)$ que remova aquele endereço de seu registro de localização para E . Se esse registro de localização ficar vazio, isto é, se não contiver nenhum outro endereço para E em D , ele pode ser removido. Nesse caso, o nó-pai do $dir(D)$ quer remover seu ponteiro para $dir(D)$. Se, no entanto, o registro de localização para E no pai também ficar vazio, esse registro deve ser removido também, e o próximo nó de diretório de nível mais alto deve ser informado. Novamente, esse processo continua até um ponteiro ser removido de um registro de localização que permanece não vazio dali em diante, ou até a raiz ser alcançada.

5.3 Nomeação Estruturada

Nomes simples são bons para máquinas mas, em geral, não são muito convenientes para a utilização de seres humanos. Como alternativa, sistemas de nomeação comumente suportam nomes estruturados, que são compostos por nomes simples, passíveis de leitura pelas pessoas. A nomeação de arquivos, bem como a nomeação de hospedeiros na Internet, segue essa abordagem. Nesta seção, vamos nos concentrar em nomes estruturados e no modo como esses nomes são resolvidos para endereços.

5.3.1 Espaços de nomes

Normalmente, nomes são organizados no que denominamos **espaço de nomes**. Espaços de nomes para nomes estruturados podem ser representados como um gráfico dirigido e rotulado com dois tipos de nós. Um **nó-folha** representa uma entidade nomeada e tem a propriedade de não ter ramos de saída. Um nó-folha geralmente armazena informações sobre a entidade que está representando — por exemplo, seu endereço — de modo que um

cliente possa acessá-las. Como alternativa, ele pode armazenar o estado daquela entidade, como no caso de sistemas de arquivo no qual um nó-folha realmente contém o arquivo completo que está representando. Mais adiante voltaremos ao conteúdo de nós.

Ao contrário de um nó-folha, um **nó de diretório** tem vários ramos de saída, cada um rotulado com um nome, como mostra a Figura 5.9. Cada nó em um gráfico de nomeação é considerado como mais uma outra entidade em um sistema distribuído e, em particular, tem um identificador associado. Um nó de diretório armazena uma tabela na qual um ramo de saída é representado por um par (*rótulo do ramo*, *identificador do nó*). Essa tabela é denominada **tabela de diretório**.

O gráfico de nomeação mostrado na Figura 5.9 tem um nó, a saber, n_0 , que tem somente ramos de saída e nenhum ramo de entrada. Tal nó é denominado (**nó**) **raiz** do gráfico de nomeação. Embora seja possível que um gráfico de nomeação tenha vários nós-raiz, por simplicidade, muitos sistemas de nomeação têm somente um. Cada caminho em um gráfico de nomeação pode ser referenciado pela sequência de rótulos correspondentes aos ramos naquele caminho, como

$N:<label-1, label-2, \dots, label-n>$

onde N se refere ao primeiro nó no caminho. Tal sequência é denominada **nome de caminho**. Se o primeiro nó no nome de caminho for a raiz do gráfico de nomeação, ele é denominado **nome de caminho absoluto**. Caso contrário, é chamado **nome de caminho relativo**.

É importante perceber que nomes são sempre organizados em um espaço de nomes. Por consequência, um nome é sempre definido em relação a apenas um nó de diretório. Nesse sentido, o termo ‘nome absoluto’ é um pouco enganador. Da mesma maneira, a diferença entre nomes globais e locais muitas vezes pode ser confusa. Um **nome global** é um nome que denota a mesma entidade, sem importar onde ele é usado em um sistema. Em outras palavras, um nome global é sempre interpretado em relação ao mesmo nó de diretório. Ao contrário, um **nome local** é um nome cuja interpretação depende de onde aquele nome está sendo usado. Em outras palavras,

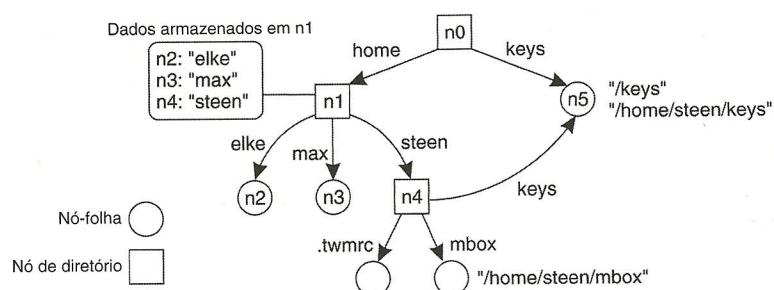


Figura 5.9 Gráfico de nomeação geral com um único nó-raiz.

um nome local é, em essência, um nome relativo cujo diretório no qual ele está contido é (implicitamente) conhecido. Voltaremos a essas questões mais adiante quando discutirmos resolução de nomes.

Essa descrição de um gráfico de nomeação se aproxima da que é implementada em muitos sistemas de arquivo. Contudo, em vez de escrever a sequência de rótulos de caminhos para representar um nome de caminho, os nomes de caminho em sistemas de arquivo em geral são representados como uma única cadeia na qual os rótulos são separados por um caractere separador especial, como uma barra (/). Esse caractere também é usado para indicar se um nome de caminho é absoluto. Por exemplo, na Figura 5.9, em vez de usar $n_0:<\text{home}, \text{steen}, \text{mbox}>$, isto é, o nome de caminho propriamente dito, na prática é comum usar sua representação em cadeia, /home/steen/mbox.

Observe também que, quando há diversos caminhos que levam ao mesmo nó, esse nó pode ser representado por diferentes nomes de caminho. Por exemplo, o nó n_5 na Figura 5.9 pode ser referenciado por /home/steen/keys, bem como por /keys. A representação em cadeia de nomes de caminho pode ser igualmente bem aplicada a outros gráficos de nomeação que não sejam os usados somente para sistemas de arquivo. Em Plan 9 (Pike et al., 1995), todos os recursos, como processos, hospedeiros, dispositivos de E/S e interfaces de rede, são nomeados do mesmo modo que arquivos tradicionais. Essa abordagem é análoga à implementação de um único gráfico de nomeação para todos os recursos em um sistema distribuído.

Há muitas maneiras diferentes de organizar um espaço de nomes. Como mencionamos, a maioria dos espaços de nomes tem apenas um único nó-raiz. Em muitos casos, um espaço de nomes também é estritamente hierárquico no sentido de que o gráfico de nomeação é organizado como uma árvore. Isso significa que cada nó, exceto a raiz, tem exatamente um ramo de entrada; a raiz não tem nenhuma. Em decorrência, cada nó também tem exatamente um nome de caminho (absoluto) associado.

O gráfico de nomeação mostrado na Figura 5.9 é um exemplo de *gráfico acíclico dirigido*. Nessa organização, um nó pode ter mais do que um ramo de entrada, mas não é permitido que o gráfico tenha um ciclo. Também há espaços de nomes que não têm essa restrição.

A fim de trazer esse assunto para um terreno mais concreto, considere o modo como são nomeados os arquivos em um sistema tradicional de arquivos Unix. Em um

gráfico de nomeação para Unix, um nó de diretório representa um diretório de arquivos, ao passo que um nó-folha representa um arquivo. Há um único diretório-raiz, representado no gráfico de nomeação pelo nó-raiz. A implementação do gráfico de nomeação é uma parte integrante da implementação completa do sistema de arquivos. A implementação consiste em uma série de blocos contíguos de um disco lógico, geralmente divididos em um bloco de inicialização, um superbloco, uma série de nós de índice (denominados inodes) e blocos de dados de arquivo. Veja também Crowley (1997), Silberschatz et al. (2005) e Tanenbaum e Woodhull (2006). Essa organização é mostrada na Figura 5.10.

O bloco de inicialização é um bloco especial de dados e instruções que é carregado automaticamente na memória principal quando o sistema é inicializado. O bloco de inicialização é usado para carregar o sistema operacional na memória principal.

O superbloco contém informações sobre todo o sistema de arquivo, como seu tamanho, quais blocos de disco ainda não estão alocados, quais inodes ainda não foram usados e assim por diante. Inodes são referenciados por um número de índice, começando pelo número zero, que é reservado para o inode que representa o diretório-raiz.

Cada inode contém informações sobre o lugar no disco em que podem ser encontrados os dados de seu arquivo associado. Além disso, um inode contém informações sobre seu proprietário, quando foi criado e quando ocorreu a última modificação, proteção e coisas semelhantes. Em consequência, dado o número de índice de um inode, é possível acessar seu arquivo associado. Cada diretório também é implementado como um arquivo. Isso também acontece com o diretório-raiz, que contém um mapeamento entre nomes de arquivo e números de índices de inodes. Assim, podemos perceber que o número de índice de um inode corresponde a um identificador de nó no gráfico de nomeação.

5.3.2 Resolução de nomes

Espaços de nomes oferecem um mecanismo conveniente para armazenar e recuperar informações sobre entidades por meio de nomes. De modo mais geral, dado um nome de caminho, deve ser possível consultar qualquer informação armazenada no nó referenciado por aquele nome. O processo de busca de um nome é denominado **resolução de nomes**.

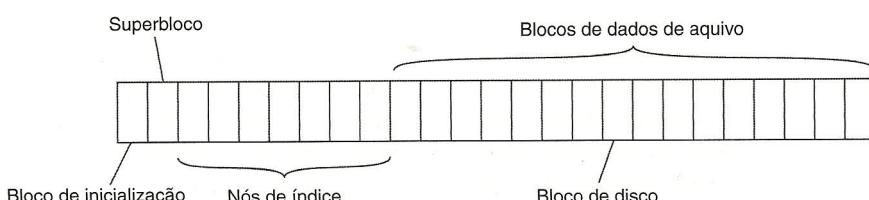


Figura 5.10 Organização geral da implementação do sistema de arquivos Unix em um disco lógico de blocos contíguos de disco.

Para explicar como funciona a resolução de nomes, vamos considerar um nome de caminho tal como $N:<label_1,label_2,\dots,label_n>$. A resolução desse nome começa no nó N do gráfico de nomeação, onde o nome $label_1$ é consultado na tabela de diretório e de onde retorna o identificador do nó ao qual $label_1$ se refere. Então a resolução continua no nó identificado, pela consulta ao nome $label_2$ em sua tabela de diretório e assim por diante. Tendo como premissa que o caminho nomeado realmente existe, a resolução pára no último nó referenciado por $label_n$, pelo retorno do conteúdo daquele nó.

Uma consulta de nome retorna o identificador de um nó do lugar em que o processo de resolução de nomes continua. Em particular, é necessário acessar a tabela de diretório do nó identificado. Considere novamente um gráfico de nomeação para um sistema de arquivos Unix. Como mencionamos, um identificador de nó é implementado como o número de índice de um inode. Acessar uma tabela de diretório significa que o primeiro inode tem de ser lido para descobrir em que lugar do disco os dados propriamente ditos estão armazenados e, então, na sequência, ler os blocos de dados que contêm a tabela de diretório.

Mecanismo de fechamento

A resolução de nomes só pode ocorrer se soubermos como e onde começar. Em nosso exemplo, o nó de início foi dado e admitimos que tínhamos acesso a sua tabela de diretório. Saber como e onde iniciar uma resolução de nomes é geralmente denominado **mecanismo de fechamento**. Em essência, um mecanismo de fechamento trata da seleção do nó inicial em um espaço de nomes a partir do qual a resolução de nomes deve começar (Radia, 1989). O que faz com que às vezes seja difícil entender mecanismos de fechamento é que eles são, necessariamente, em parte implícitos e podem ser muito diferentes quando comparados uns com os outros.

Por exemplo, resolução de nomes no gráfico de nomeação para um sistema de arquivos Unix utiliza o fato de que o inode do diretório-raiz é o primeiro inode no disco lógico que representa o sistema de arquivos. Sua posição propriamente dita é calculada de acordo com os valores presentes em outros campos do superbloco, junto com informações sobre a organização interna do superbloco codificadas no próprio sistema operacional.

Para esclarecer esse ponto, considere a representação em cadeia de um nome como `/home/steen/mbox`. Para resolver esse nome, é necessário já ter acesso à tabela de diretório do nó-raiz do gráfico de nomeação adequado. Por ser um nó-raiz, o próprio nó não pode ter sido procurado, a menos que tenha sido implementado como um nó diferente em um outro gráfico de nomeação, digamos, G . Mas, nesse caso, teria sido necessário já ter acesso ao nó-raiz de G . Em consequência, resolver um nome de arquivo requer que já tenha sido implementado algum mecanismo pelo qual o processo de resolução possa começar.

Um exemplo completamente diferente é a utilização da cadeia ‘0031204430784’. Muitos não saberiam o que fazer com esses números, a menos que lhes digam que essa seqüência é um número de telefone. Essa informação é suficiente para iniciar o processo de resolução, em particular, para discar o número. Na seqüência, o sistema telefônico faz o resto.

Como último exemplo, considere a utilização de nomes globais e locais em sistemas distribuídos. Um exemplo típico de um nome local é uma variável de ambiente. Por exemplo, em sistemas Unix, a variável *HOME* é usada para se referenciar o diretório nativo de um usuário. Cada usuário tem sua própria cópia dessa variável, que é inicializada para o nome global, válido no âmbito do sistema, que corresponde ao diretório nativo do usuário. O mecanismo de fechamento associado com variáveis ambientais assegura que o nome da variável seja resolvido adequadamente, fazendo uma consulta em uma tabela específica do usuário.

Ligação e montagem

Estreitamente relacionada à resolução de nomes está a utilização de **apelidos (aliases)**. Um apelido é um outro nome para a mesma entidade. Uma variável ambiental é um exemplo de um apelido. Em termos de gráficos de nomeação, há basicamente dois modos diferentes de implementar um apelido. A primeira abordagem é simplesmente permitir que vários nomes de caminhos absolutos referenciem o mesmo nó em um gráfico de nomeação. Essa abordagem está ilustrada na Figura 5.9, na qual o nó n_5 pode ser referenciado por dois nomes de caminho diferentes. Em terminologia Unix, ambos os nomes de caminho, `/keys` e `/home/steen/keys`, na Figura 5.9, são denominados **ponteiros estritos** para o nó n_5 .

A segunda abordagem é representar uma entidade por um nó-folha, digamos, N , porém, em vez de armazenar o endereço ou estado daquela entidade, o nó armazena um nome de caminho absoluto. Ao resolver pela primeira vez um nome de caminho absoluto que leva a N , a resolução de nomes retornará o nome de caminho armazenado em N ; nesse ponto ela pode continuar com a resolução do novo nome de caminho. Esse princípio corresponde à utilização de **ponteiros simbólicos** em sistemas de arquivos Unix e é ilustrado na Figura 5.11. Nesse exemplo, o nome de caminho `/home/steen/keys`, que referencia um nó que contém o nome de caminho absoluto, `/keys`, é uma ligação simbólica para o nó n_5 .

A resolução de nomes, como a descrevemos até aqui, ocorre completamente dentro de um único espaço de nomes. Contudo, a resolução de nomes também pode ser usada para fundir diferentes espaços de nomes de maneira transparente. Em primeiro lugar, vamos considerar um sistema de arquivos montado. Em termos de nosso modelo de nomeação, um sistema de arquivos montado corresponde a deixar que um nó de diretório armazene o identificador

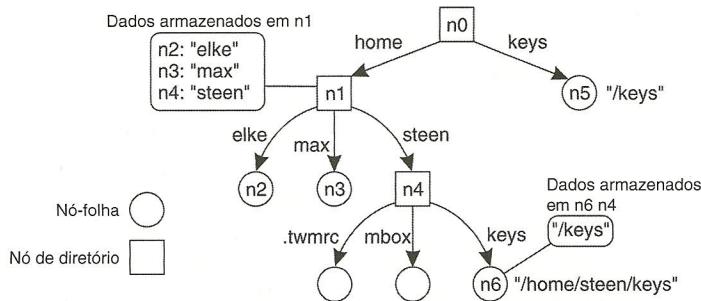


Figura 5.11 Conceito de um ponteiro simbólico explicado em um gráfico de nomeação.

um nó de diretório de um espaço de nomes *diferente*, ao qual nos referimos como espaço de nomes externo. O nó de diretório que armazena o identificador de nó é denominado **ponto de montar**. De acordo com isso, o nó de diretório no espaço de nomes externo é denominado **ponto de montagem**. Normalmente, o ponto de montagem é a (o *nó*) raiz de um espaço de nomes. Durante a resolução de nomes, o ponto de montagem é consultado e a resolução prossegue acessando sua tabela de diretório.

O princípio da montagem pode ser generalizado também para outros espaços de nomes. Em particular, o que precisamos é um nó de diretório que aja como um ponto de montar e armazene todas as informações necessárias para identificar e acessar o ponto de montagem no espaço de nomes externo. Essa abordagem é adotada em muitos sistemas distribuídos de arquivos.

Considere um conjunto de espaços de nomes que é distribuído por máquinas diferentes. Em particular, cada espaço de nomes é implementado por um servidor diferente, cada um possivelmente executando em uma máquina separada. Consequentemente, se quisermos montar um espaço de nomes externo NS_2 em um espaço de nomes NS_1 , talvez seja necessária a comunicação por uma rede com o servidor de NS_2 , porque esse servidor pode estar executando em uma máquina diferente da do servidor para NS_1 . Montar um espaço de nomes externo em um sistema distribuído requer, no mínimo, as seguintes informações:

1. O nome de um protocolo de acesso
2. O nome do servidor
3. O nome do ponto de montagem no espaço de nomes externo

Note que cada um desses nomes precisa ser resolvido. O nome de um protocolo de acesso precisa ser resolvido na implementação de um protocolo pelo qual pode ocorrer a comunicação com o servidor do espaço de nomes externo. O nome do servidor precisa ser resolvido em um endereço no qual esse servidor possa ser alcançado. Como a última parte na resolução de nomes, o nome do ponto de montagem precisa ser resolvido em um identificador de nó no espaço de nomes externo.

Em sistemas não distribuídos, pode ser que nenhum dos três pontos seja realmente necessário. Por exemplo, em Unix, não há nenhum protocolo de acesso e nenhum servidor. Além disso, o nome do ponto de montagem não é necessário porque ele é, apenas, o diretório-raiz do espaço de nomes externo.

O nome do ponto de montagem deve ser resolvido pelo servidor do espaço de nomes externo. Contudo, precisamos também de espaços de nomes e implementações para o protocolo de acesso e o nome do servidor. Uma possibilidade é representar os três nomes apresentados antes como um URL.

Para exemplificar mais concretamente, considere uma situação em que um usuário que está usando um laptop quer acessar arquivos que estejam armazenados em um servidor remoto de arquivos. A máquina cliente e o servidor de arquivos são ambos configurados com o **Sistema de Arquivos de Rede (Network File System – NFS)** da Sun, que discutiremos com detalhes no Capítulo 11. O NFS é um sistema distribuído de arquivo que vem com um protocolo que descreve com precisão como um cliente pode acessar um arquivo em um servidor (remoto) de arquivos NFS. Em particular, para permitir que o NFS funcione em toda a extensão da Internet, um cliente pode especificar exatamente qual arquivo ele quer acessar por meio de um URL do NFS: por exemplo, *nfs://flits.cs.vu.nl//home/steen*. Esse URL nomeia um arquivo — que, por acaso, é um diretório — denominado */home/steen* em um servidor de arquivos NFS *flits.cs.vu.nl*, que pode ser acessado por um cliente por meio do protocolo NFS (Shepler et al., 2003).

O nome *nfs* é um nome bem conhecido no sentido de que existe um acordo de âmbito mundial sobre como interpretar esse nome. Dado que estamos lidando com um URL, o nome *nfs* será resolvido em uma implementação do protocolo NFS. O nome do servidor é resolvido no seu endereço usando DNS, que será discutido em uma seção mais adiante. Como dissemos, */home/steen* é resolvido pelo servidor do espaço de nomes externo.

A organização de um sistema de arquivos na máquina cliente é parcialmente mostrada na Figura 5.12. O diretório-raiz tem uma quantidade de entradas definida pelo

usuário, incluindo um subdiretório denominado */remote*. A tarefa desse subdiretório é incluir pontos de montagem para espaços de nomes externos, como o diretório nativo de um usuário na Universidade de Vrije. Com essa finalidade, um nó de diretório denominado */remote/vu* é usado para armazenar o URL *nfs://flits.cs.vu.nl//home/steen*.

Agora, considere o nome */remote/vu/mbox*. Esse nome é resolvido começando no diretório-raiz na máquina cliente e continua até que o nó */remote/vu* seja alcançado. Então, o processo de resolução de nomes continua, retornando o URL *nfs://flits.cs.vu.nl//home/steen*, que, por sua vez, leva a máquina cliente a contatar o servidor de arquivo *flits.cs.vu.nl* por meio do protocolo NFS e, na seqüência, a acessar o diretório */home/steen*. Portanto, a resolução de nomes pode continuar pela leitura do arquivo denominado *mbox* naquele diretório; depois disso, o processo de resolução pára.

Sistemas distribuídos que permitem a montagem de sistemas de arquivo remoto, como acabamos de descrever, permitem que uma máquina cliente execute, por exemplo, os seguintes comandos:

```
cd /remote/vu  
ls -l
```

que, na seqüência, apresenta a lista de arquivos no diretório */home/steen* no servidor de arquivos remoto. O bom de tudo isso é que o usuário é pouparado dos detalhes do acesso propriamente dito ao servidor remoto. O ideal é que seja notada apenas uma certa perda de desempenho em comparação com o acesso de arquivos disponíveis no local. Na verdade, para o cliente, parece que o espaço de nomes enraizado na máquina local e o enraizado em */home/steen* na máquina remota formam um único espaço de nomes.

5.3.3 Implementação de um espaço de nomes

Um espaço de nomes é o centro de um serviço de nomeação, isto é, um serviço que permite que usuários e processos adicionem, removam e consultem nomes. Um serviço de nomeação é implementado por servidores de nomes. Se um sistema distribuído estiver restrito a uma rede local, muitas vezes é viável implementar um serviço de nomeação por meio de um único servidor de nomes. Contudo, em sistemas distribuídos de grande escala com muitas entidades possivelmente dispersas por uma grande área geográfica, é necessário distribuir a implementação de um espaço de nomes por vários servidores de nomes.

Distribuição de espaços de nomes

Espaços de nomes para um sistema distribuído de grande escala, possivelmente de âmbito mundial, costumam ser organizados em hierarquia. Como antes, considere que tal espaço de nomes tenha apenas um único nó-raiz. Para implementar efetivamente esse espaço de nomes, é conveniente reparti-lo em camadas lógicas. Cheriton e Mann (1989) distinguem as três camadas seguintes.

A **camada global** é formada por nós do nível mais alto, isto é, o nó-raiz e outros nós de diretório logicamente próximos ao raiz, ou seja, seus filhos. Os nós na camada global costumam ser caracterizados por sua estabilidade, no sentido de que as tabelas de diretório raramente mudam. Esses nós podem representar organizações, ou grupos de organizações, cujos nomes estão armazenados no espaço de nomes.

A **camada administrativa** é formada por nós de diretório que, juntos, são gerenciados por uma única organização. Um aspecto característico dos nós de diretório na camada administrativa é que eles representam grupos de entidades que pertencem à mesma organização ou unidade

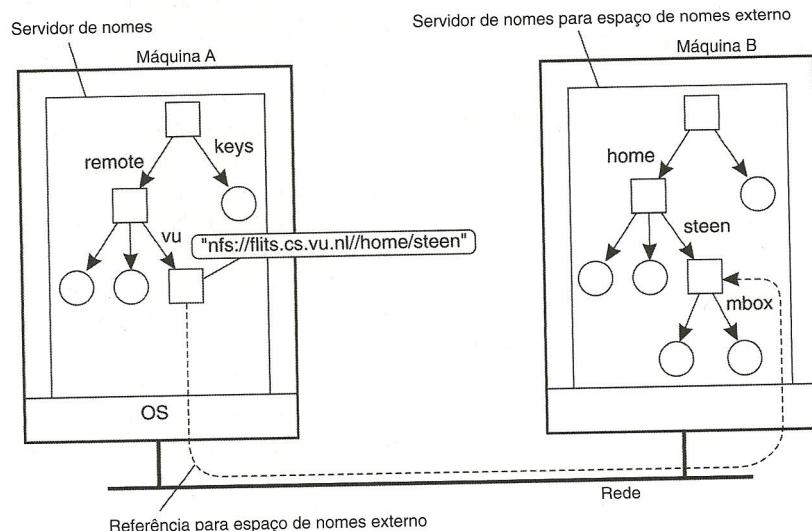


Figura 5.12 Montagem de espaços de nomes remotos por meio de um protocolo de acesso específico.

administrativa. Por exemplo, pode haver um nó de diretório para cada departamento em uma organização, ou um nó de diretório com base no qual todos os hospedeiros podem ser encontrados. Um outro nó de diretório pode ser usado como ponto de partida para nomear todos os usuários e assim por diante. Os nós na camada administrativa são relativamente estáveis, embora, de modo geral, as mudanças ocorram com maior freqüência do que nos nós da camada global.

Por fim, a **camada gerencial** consiste em nós cujo comportamento típico é a mudança periódica. Por exemplo, nós que representam hospedeiros na rede local pertencem a essa camada. Pela mesma razão, a camada inclui nós que representam arquivos compartilhados como os de bibliotecas ou binários. Uma outra classe importante de nós inclui os que representam diretórios e arquivos definidos por usuários. Ao contrário das camadas global e administrativa, os nós na camada gerencial são mantidos não somente por administradores de sistemas, mas também por usuários individuais de um sistema distribuído.

Exemplificando concretamente, a Figura 5.13 mostra um exemplo da repartição de parte do espaço de nomes DNS, incluindo os nomes de arquivos que estão dentro de uma organização e que podem ser acessados pela Internet, por exemplo, páginas Web e arquivos transferíveis. O espaço de nomes é dividido em partes que não se sobrepõem, denominadas **zonas** em DNS (Mockapetris, 1987). Uma zona é uma parte do espaço de nomes que é implementada por um servidor de nomes separado. Algumas dessas zonas são ilustradas na Figura 5.13.

No que se refere à disponibilidade e ao desempenho, os servidores de nomes em cada camada têm de cumprir requisitos diferentes. Alta disponibilidade é especialmente crítica para servidores de nomes na camada global. Se um servidor de nomes falhar, uma grande porção do espa-

ço de nomes será inalcançável porque a resolução de nomes não pode passar do servidor que falhou.

O desempenho é um pouco mais sutil. Devido à baixa taxa de mudança de nós na camada global, os resultados de operações de consulta em geral permanecem válidos por um longo tempo. Como consequência, esses resultados podem ser efetivamente mantidos em cache — isto é, armazenados no local — pelos clientes. Da próxima vez que a mesma operação de consulta for executada, os resultados podem ser retirados da cache do cliente, em vez de deixar que o servidor de nomes retorne os resultados. O efeito disso é que os servidores de nomes na camada global não têm de responder rapidamente a uma requisição de consulta isolada. Por outro lado, a vazão pode ser importante, em especial em sistemas de grande escala com milhões de usuários.

Os requisitos de disponibilidade e desempenho para servidores de nomes na camada global podem ser cumpridos pela replicação de servidores, combinada com cache no lado do cliente. Como discutiremos no Capítulo 7, atualizações nessa camada em geral não têm de surtir efeito imediato, o que facilita muito a manutenção da consistência das réplicas.

A disponibilidade de um servidor de nomes na camada administrativa é de importância primordial para clientes na mesma organização que o servidor de nomes. Se o servidor de nomes falhar, muitos recursos dentro da organização tornam-se inalcançáveis porque não podem ser consultados. Por outro lado, para usuários de fora da organização, pode ser menos importante que os recursos de uma organização fiquem temporariamente inalcançáveis.

Quanto ao desempenho, servidores de nomes na camada administrativa têm características semelhantes aos da camada global. Como mudanças em nós não ocorrem com tanta freqüência, manter resultados de consulta

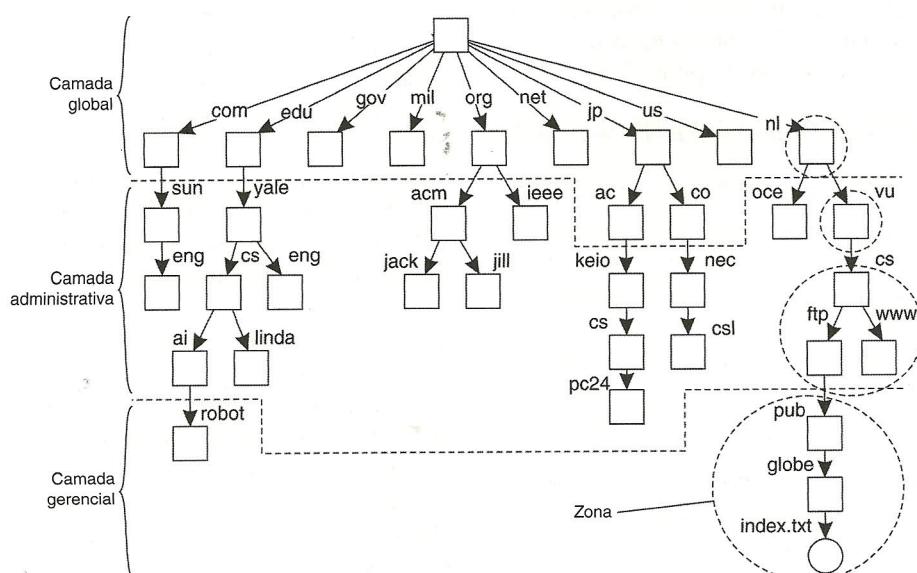


Figura 5.13 Exemplo de repartição do espaço de nomes DNS, incluindo arquivos acessíveis pela Internet, em três camadas.

em cache pode ser muito eficiente, o que torna o desempenho menos crítico. Contudo, ao contrário da camada global, a camada administrativa deve providenciar que os resultados de consultas sejam retornados dentro de alguns milissegundos, seja diretamente a partir do servidor, seja a partir da cache local do cliente. Da mesma maneira, as atualizações devem ser processadas, em geral, com mais rapidez do que as da camada global. Por exemplo, é inaceitável que uma conta de um novo usuário leve horas para se tornar efetiva.

Esses requisitos muitas vezes podem ser cumpridos usando máquinas de alto desempenho para rodar servidores de nomes. Além disso, deve ser aplicada cache do lado do cliente, combinada com replicação, para aprimorar a disponibilidade global.

Os requisitos de disponibilidade para servidores de nomes no nível gerencial são, de modo geral, menos exigentes. Em particular, muitas vezes é suficiente usar uma única máquina (dedicada) para rodar servidores de nomes, correndo o risco de indisponibilidade temporária. Todavia, o desempenho é crucial. Usuários esperam que as operações ocorram imediatamente. Como as atualizações ocorrem periodicamente, manter cache do lado do cliente costuma ser menos efetivo, a não ser que sejam tomadas providências especiais, que discutiremos no Capítulo 7.

Uma comparação entre servidores de nomes em diferentes camadas é mostrada na *Tabela 5.1*. Em sistemas distribuídos, servidores de nomes nas camadas global e administrativa são os mais difíceis de implementar. As dificuldades são causadas pela replicação e manutenção de cache necessárias para disponibilidade e desempenho, mas que também introduzem problemas de consistência. Alguns dos problemas são agravados pelo fato de que caches e réplicas são espalhadas por toda a extensão da rede de longa distância, o que introduz longos atrasos de comunicação e, por conseguinte, torna a sincronização ainda mais difícil. Replicação e manutenção de cache serão discutidas extensivamente no Capítulo 7.

Item	Global	Administrativa	Gerencial
Escala geográfica da rede	Mundial	Organização	Departamento
Número total de nós	Poucos	Muitos	Grandes quantidades
Capacidade de resposta a consultas	Segundos	Milissegundos	Imediata
Propagação de atualizações	Lerda	Imediata	Imediata
Quantidade de réplicas	Muitas	Nenhuma ou poucas	Nenhuma
É aplicada cache do lado do cliente?	Sim	Sim	Às vezes

Tabela 5.1 Comparação entre servidores de nomes para implementar nós de espaço de nomes de grande escala repartidos em uma camada global, uma camada administrativa e uma camada gerencial.

Implementação de resolução de nomes

A distribuição de um espaço de nomes por vários servidores de nomes afeta a implementação da resolução de nomes. Para explicar a implementação de resolução de nomes em serviços de nomeação de grande escala, consideraremos, por enquanto, que os servidores de nomes não são replicados e que não são usadas caches no lado do cliente. Cada cliente tem acesso a um **resolvedor de nomes** local, que é responsável por assegurar que o processo de resolução de nomes seja executado. Com referência à Figura 5.13, suponha que o nome de caminho (absoluto)

root:<nl, vu, cs, ftp, pub, globe, index.html>

deva ser resolvido. Usando uma notação URL, esse nome de caminho corresponderia a *ftp://ftp.cs.vu.nl/pub/globe/index.html*. Agora, há dois modos de implementar resolução de nomes.

Em **resolução iterativa de nomes**, um resolvedor de nomes entrega o nome completo ao servidor-raiz de nomes. Adotamos como premissa que o endereço em que o servidor-raiz pode ser contatado é bem conhecido. O servidor-raiz resolverá o nome de caminho até onde puder e retornará o resultado ao cliente. Em nosso exemplo, o servidor-raiz pode resolver somente o rótulo *nl*, para o qual ele retornará o endereço do servidor de nomes associado.

Nesse ponto, o cliente passa o restante do nome de caminho, isto é, *nl:<vu, cs, ftp, pub, globe, index.html>*, para esse servidor de nomes. Ele pode resolver somente o rótulo *vu* e retorna o endereço do servidor de nomes associado, junto com o restante do nome de caminho *vu:<cs, ftp, pub, globe, index.html>*.

Em seguida, o resolvedor de nomes do cliente entrará em contato com o próximo servidor de nomes, que responde resolvendo o rótulo *cs* e, na sequência, também *ftp*, retornando o endereço do servidor FTP junto com o nome de caminho *ftp:<pub, globe, index.html>*. Sendo assim, o cliente contata o servidor FTP e requisita que ele resolva a última parte do nome de caminho original. Na sequência, o servidor FTP resolverá os rótulos *pub, globe* e *index.html*, e transferirá o arquivo requisitado (nesse caso usando FTP). Esse processo de resolução iterativa de nomes é mostrado na Figura 5.14. (A notação *#<cs>* é usada para indicar o endereço do servidor responsável pela manipulação do nó referenciado por *<cs>*.)

Na prática, a última etapa, ou seja, contatar o servidor FTP e requisitar que ele transfira o arquivo cujo nome de caminho é *ftp:<pub, globe, index.html>*, é realizada em separado pelo processo cliente. Em outras palavras, o cliente normalmente entregaria somente o nome de caminho *root:<nl, vu, cs, ftp>* ao resolvedor de nomes, do qual esperaria o endereço de onde ele poderia contatar o servidor FTP, como também mostra a Figura 5.14.

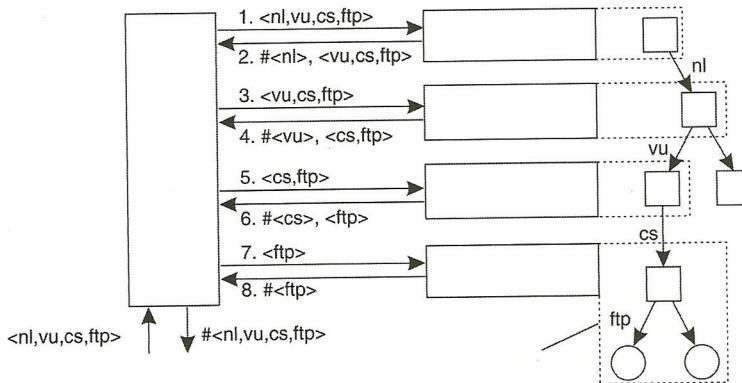


Figura 5.14 Princípio da resolução iterativa de nomes.

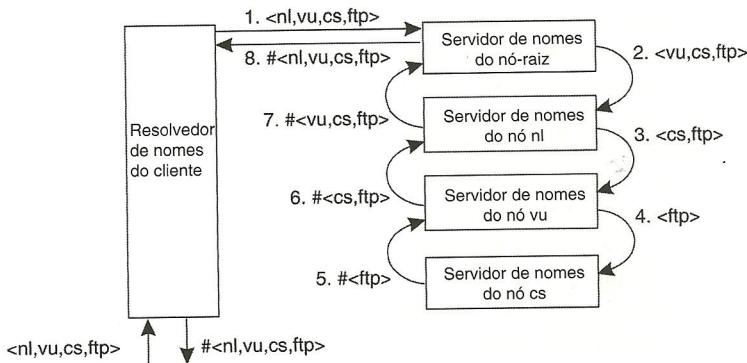


Figura 5.15 Princípio da resolução recursiva de nomes.

Uma alternativa para a resolução iterativa de nomes é usar recursão durante a resolução de nomes. Com **resolução recursiva de nomes**, em vez de retornar cada resultado intermediário de volta ao resolvedor de nomes do cliente, um servidor de nomes passa o resultado para o próximo servidor de nomes que encontrar. Portanto, por exemplo, quando o servidor-raiz de nomes encontrar o endereço do servidor de nomes que implementa o nó denominado *nl*, ele requisita que o servidor de nomes resolva o nome de caminho *nl:<vu, cs, ftp, pub, globe, index.html>*. Esse próximo servidor, que também utiliza resolução recursiva de nomes, resolverá o caminho completo e, a certa altura, retornará o arquivo *index.html* ao servidor-raiz que, por sua vez, passará esse arquivo para o resolvedor de nomes do cliente.

A Figura 5.15 mostra a resolução recursiva de nomes. Como na resolução iterativa de nomes, a última etapa da resolução — contatar o servidor FTP e solicitar que ele transfira o arquivo indicado — geralmente é realizada como um processo separado pelo cliente.

A principal desvantagem da resolução recursiva de nomes é que ela impõe uma exigência de desempenho mais alta a cada servidor de nomes. Basicamente, a tarefa de um servidor de nomes é manipular a resolução com-

pleta de um nome de caminho, embora possa fazer isso em cooperação com outros servidores de nomes. Essa carga adicional em geral é tão alta que os servidores de nomes na camada global de um espaço de nomes suportam somente resolução iterativa de nomes.

A resolução recursiva de nomes tem duas vantagens importantes. A primeira é que manter resultados em cache é mais efetivo em comparação com a resolução iterativa de nomes. A segunda é que os custos de comunicação podem ser reduzidos. Para explicar essas vantagens, suponha que o resolvedor de nomes de um cliente aceitará nomes de caminho que referenciem somente nós na camada global ou na camada administrativa do espaço de nomes. Para resolver a parte de um nome de caminho que corresponde a nós na camada gerencial, um cliente contatará separadamente o servidor de nomes retornado por seu resolvedor de nomes, como acabamos de discutir.

A resolução recursiva de nomes permite que cada servidor de nomes aprenda gradativamente o endereço de cada servidor de nomes responsável pela implementação de nós de nível mais baixo. O resultado é que a manutenção de cache pode ser usada efetivamente para aprimorar o desempenho. Por exemplo, quando o servi-

dor-raiz é requisitado para resolver o nome de caminho $root:<nl, vu, cs, ftp>$, a certa altura ele obterá o endereço do servidor de nomes que implementa o nó referenciado por esse nome de caminho. Para chegar a esse ponto, o servidor de nomes para o nó nl tem de consultar o endereço do servidor de nomes para o nó vu , ao passo que este tem de consultar o endereço do servidor de nomes que manipula o nó cs .

Como mudanças em nós na camada global e na camada administrativa não ocorrem com freqüência, o servidor-raiz de nomes pode efetivamente manter em cache o endereço retornado. Além do mais, como o endereço também é retornado por recursão ao servidor de nomes responsável por implementar o nó vu e ao servidor de nomes que implementa o nó nl , ele também pode muito bem ser mantido em cache nesses servidores.

Da mesma maneira, os resultados de consultas intermediárias de nomes também podem ser retornados e mantidos em cache. Por exemplo, o servidor para o nó nl terá de consultar o endereço do servidor de nomes vu . Esse endereço pode ser retornado para o servidor-raiz quando o servidor nl retornar o resultado da consulta de nomes original. A Tabela 5.2 mostra uma visão completa do processo de resolução e dos resultados que podem ser mantidos em cache por cada servidor de nomes.

O principal benefício dessa abordagem é que, a certa altura, as operações de consulta podem ser manipuladas com bastante eficiência. Por exemplo, suponha que, mais tarde, um outro cliente requisite a resolução do nome de caminho $root:<nl, vu, cs, flits>$. Esse nome é passado para o raiz, que imediatamente o repassa para o servidor de nomes do nó cs , e requisita que ele resolva o restante do nome de caminho $cs:<flits>$.

Com resolução iterativa de nomes, a manutenção de cache fica necessariamente restrita ao resolvedor de nomes do cliente. Como resultado, se um cliente A requisitar a resolução de um nome e, mais tarde, um outro cliente B requisitar que esse mesmo nome seja resolvido,

a resolução de nomes terá de passar pelos mesmos servidores de nomes pelos quais passou a do cliente A . Como solução de compromisso, muitas organizações usam um servidor de nomes intermediário, local, que é compartilhado por todos os clientes. Esse servidor de nomes local manipula todas as requisições de nomeação e coloca os resultados em cache. O servidor intermediário também é conveniente do ponto de vista de gerenciamento. Por exemplo, somente ele precisa saber onde o servidor-raiz de nomes está localizado; outras máquinas não precisam dessa informação.

A segunda vantagem da resolução recursiva de nomes é que muitas vezes ela é mais barata no que diz respeito à comunicação. Considere, mais uma vez, a resolução do nome de caminho $root:<nl, vu, cs, ftp>$ e suponha que o cliente esteja localizado em San Francisco. Adotando como premissa que o cliente conhece o endereço do servidor para o nó nl , com resolução recursiva de nomes a comunicação segue a rota desde o hospedeiro do cliente em San Francisco até o servidor nl na Holanda, representado por $R1$ na Figura 5.16. A partir desse ponto, é preciso comunicação subsequente entre o servidor nl e o servidor de nomes da Universidade de Vrije no campus universitário em Amsterdã, Holanda. Essa comunicação é representada por $R2$. Por fim, é preciso comunicação entre o servidor vu e o servidor de nomes no Departamento de Ciência da Computação, representado por $R3$. A rota para a resposta é a mesma, mas na direção contrária. Claro que os custos de comunicação são ditados pela troca de mensagens entre o hospedeiro do cliente e o servidor nl .

Em comparação, com resolução iterativa de nomes, o hospedeiro do cliente tem de se comunicar em separado com o servidor nl , o servidor vu e o servidor cs ; o total dessas operações pode ser aproximadamente três vezes da resolução recursiva de nomes. As setas rotuladas $I1$, $I2$ e $I3$ na Figura 5.16 mostram o caminho da comunicação para resolução iterativa de nomes.

Servidor para o nó	Deve resolver	Consulta	Passa para filho	Recebe e mantém em cache	Retorna ao requisitante
cs	<ftp>	#<ftp>	—	—	#<ftp>
vu	<cs,ftp>	#<cs>	<ftp>	#<ftp>	#<cs> #<cs, ftp>
nl	<vu,cs,ftp>	#<vu>	<cs,ftp>	#<cs> #<cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>
root	<nl,vu,cs,ftp>	#<nl>	<vu,cs,ftp>	#<vu> #<vu,cs> #<vu,cs,ftp>	#<nl> #<nl,vu> #<nl,vu,cs> #<nl,vu,cs,ftp>

Tabela 5.2 Resolução recursiva de nomes de $<nl, vu, cs, ftp>$. Servidores de nomes mantêm resultados intermediários em cache para consultas subsequentes.

5.3.4 Exemplo: Sistema de Nomes de Domínio

Um dos maiores serviços distribuídos de nomeação em uso hoje é o Sistema de Nomes de Domínio (Domain Name System — DNS) da Internet. O DNS é usado primordialmente para consultar endereços IP de hospedeiros e servidores de correio. Nas páginas seguintes, vamos nos concentrar na organização do espaço de nomes DNS e nas informações armazenadas em seus nós. Além disso, examinaremos mais de perto a implementação propriamente dita do DNS. Mais informações podem ser encontradas em Mockapetris (1987) e em Albitz e Liu (2001). Uma avaliação recente do DNS, em particular no que se refere à sua adequação às necessidades da Internet de hoje, pode ser encontrada em Levien (2005). Por esse relatório, podemos chegar à conclusão um tanto surpreendente de que, mesmo após mais de 30 anos, não há nenhuma indicação de que o DNS precise ser substituído. Poderíamos argumentar que a principal causa se encontra no profundo entendimento do projetista de como manter as coisas simples. A prática em outros campos de sistemas distribuídos indica que não há muitos que tenham esse mesmo dom.

Espaço de nomes DNS

O espaço de nomes DNS é organizado em hierarquia como uma árvore com raiz. Um rótulo é uma cadeia composta por caracteres alfanuméricos na qual a utilização de minúsculas ou maiúsculas é indiferente. Um rótulo tem um comprimento máximo de 63 caracteres; o comprimento de um nome de caminho completo está restrito a 255 caracteres. A representação em cadeia de um nome de caminho consiste em uma listagem de seus rótulos, começando com a da extrema direita e separando os rótulos por um ponto ('.'). A raiz é representada por um ponto. Assim, por exemplo, o nome de caminho *root:<nl, vu, cs, flits>* é representado pela cadeia *flits.cs.vu.nl.*, que inclui o ponto da extrema direita para indicar o nó-raiz. Em geral omitimos esse ponto por questão de facilidade de leitura.

Como cada nó no espaço de nomes DNS tem exatamente um ramo de entrada (com a exceção do nó-raiz, que não tem nenhum ramo de entrada), o rótulo anexado

ao ramo de entrada de um nó também é usado como o nome para aquele nó. Uma subárvore é denominada **domínio**; um nome de caminho até seu nó-raiz é denominado **nome de domínio**. Note que, exatamente como um nome de caminho, um nome de domínio pode ser absoluto ou relativo.

O conteúdo de um nó é formado por um conjunto de **registros de recursos**. Há tipos diferentes de registros de recursos. Os principais são mostrados na *Tabela 5.3*.

Um nó no espaço de nomes DNS freqüentemente representará várias entidades ao mesmo tempo. Por exemplo, um nome de domínio como *vu.nl* é usado para representar um domínio e uma zona. Nesse caso, o domínio é implementado por meio de diversas zonas não sobrepostas.

Um registro de recurso de *início de autoridade* (Start of Authority — SOA) contém informações como o endereço de e-mail do administrador de sistemas responsável pela zona representada, o nome do hospedeiro em que os dados sobre a zona podem ser buscados e assim por diante.

Entidade associada	Descrição
Zona	Contém informações sobre a zona representada
Hospedeiro	Contém um endereço IP do hospedeiro que esse nó representa
Domínio	Refere-se a um servidor de correio para manipular o correio endereçado a esse nó
Domínio	Refere-se a um servidor que manipula um serviço específico
Zona	Refere-se a um servidor de nomes que implementa a zona representada
Nó	Ponteiro simbólico para o nome primário do nó representado
Hospedeiro	Contém o nome canônico de um hospedeiro
Hospedeiro	Mantém informações sobre o hospedeiro que esse nó representa
Qualquer tipo	Contém qualquer informação específica de entidade considerada útil

Tabela 5.3 Tipos mais importantes de registros de recursos que formam o conteúdo de nós no espaço de nomes DNS.

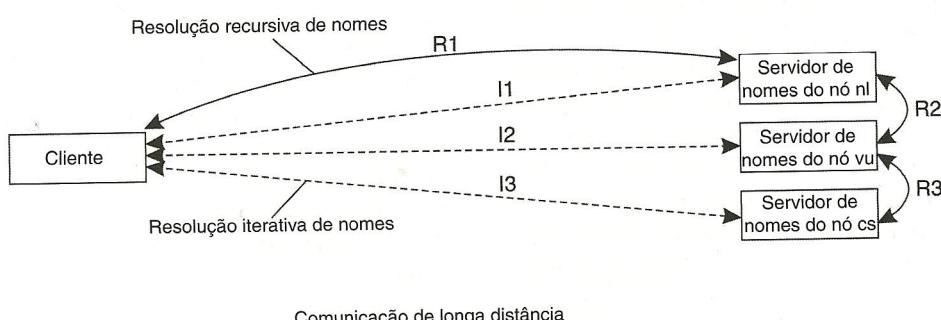


Figura 5.16 Comparação entre resolução recursiva e iterativa de nomes no que diz respeito aos custos de comunicação.

Um registro *A* (endereço) representa um hospedeiro particular na Internet. O registro *A* contém um endereço IP para esse hospedeiro para permitir a comunicação com ele. Se um hospedeiro tem vários endereços IP, como acontece com máquinas que se ligam a várias redes, o nó conterá um registro *A* para cada endereço.

Um outro tipo de registro é o *MX* (troca de correio), que é como um ponteiro simbólico para um nó que representa um servidor de correio. Por exemplo, o nó que representa o domínio *cs.vu.nl* tem um registro *MX* que contém o nome *zephyr.cs.vu.nl*, que se refere a um servidor de correio. Esse servidor manipulará todo o correio endereçado aos usuários no domínio *cs.vu.nl*. Pode haver vários registros *MX* armazenados em um nó.

Registros *SRV*, relacionados com registros *MX*, contêm o nome de um servidor para um serviço específico. Registros *SRV* são definidos em Gulbrandsen et al. (2000). O serviço em si é identificado por meio de um nome, junto ao nome de um protocolo. Por exemplo, o servidor Web no domínio *cs.vu.nl* poderia ser nomeado por meio de um registro *SRV* como *_http._tcp.cs.vu.nl*. Esse registro referenciaria portanto o nome propriamente dito do servidor (que é *soling.cs.vu.nl*). Uma vantagem importante dos registros *SRV* é que os clientes não precisam mais saber o nome DNS do hospedeiro que oferece um serviço específico. Em vez disso, somente nomes de serviços precisam ser padronizados; depois, o hospedeiro fornecedor pode ser consultado.

Nós que representam uma zona contêm um ou mais registros *NS* (servidores de nomes). Um registro *NS*, assim como os registros *MX*, contém o nome de um servidor de nomes que implementa a zona representada pelo nó. Em princípio, cada nó no espaço de nomes pode armazenar um registro *NS* que referencia o servidor de nomes que o implementa. Contudo, como discutiremos mais adiante, a implementação do espaço de nomes DNS é tal que somente nós que representam zonas precisam armazenar registros *NS*.

O DNS distingue apelidos daquilo que são denominados **nomes canônicos**. Cada hospedeiro deve ter um nome canônico, ou nome primário. Um apelido é implementado por meio do nó que armazena um registro *CNAME* que contém o nome canônico de um hospedeiro. Assim, o nome do nó que armazena tal registro é um ponteiro simbólico, como mostra a Figura 5.11.

O DNS mantém um mapeamento inverso de endereços IP para nomes de hospedeiros por meio de registros *PTR* (ponteiros). Para acomodar as consultas de nomes de hospedeiros quando é dado somente um endereço IP, o DNS mantém um domínio denominado *in-addr.arpa*, que contém nós que representam hospedeiros da Internet e que são nomeados pelo endereço IP do hospedeiro representado. Por exemplo, o hospedeiro *www.cs.vu.nl* tem endereço IP 130.37.20.20. O DNS cria um nó denominado

do 20.37.130.in-addr.arpa, que é usado para armazenar o nome canônico daquele hospedeiro (que, por acaso, é *soling.cs.vu.nl*) em um registro *PTR*.

Os dois últimos tipos de registro são os registros *HINFO* e os registros *TXT*. Um registro *HINFO* (informações de hospedeiro) é usado para armazenar informações adicionais sobre um hospedeiro, como seu tipo de máquina e sistema operacional. De modo semelhante, registros *TXT* são usados para qualquer outro tipo de dados que um usuário achar útil armazenar sobre a entidade representada pelo nó.

Implementação do DNS

Em essência, o espaço de nomes DNS pode ser dividido em uma camada global e em uma camada administrativa, como mostra a Figura 5.13. A camada gerencial, que em geral é formada por sistemas locais de arquivo, não é parte formal do DNS e, portanto, também não é gerenciada por ele.

Cada zona é implementada por um servidor de nomes, que praticamente sempre é replicado por questão de disponibilidade. Atualizações para uma zona normalmente são manipuladas pelo servidor primário de nomes. As atualizações ocorrem pela modificação do banco de dados DNS local do servidor primário. Servidores secundários de nomes não acessam o banco de dados diretamente mas, em vez disso, requisitam ao servidor primário que transfira seu conteúdo. Essa operação é denominada **transferência de zona** na terminologia do DNS.

Um banco de dados DNS é implementado como um (pequeno) conjunto de arquivos, dos quais o mais importante contém todos os registros de recursos para *todos* os nós em determinada zona. Essa abordagem permite que os nós sejam simplesmente identificados por meio de seus nomes de domínio e, por isso, a noção de um identificador de nó se reduz a um índice (implícito) para um arquivo.

Para entender melhor essas questões de implementação, a Tabela 5.4 mostra uma pequena parte do arquivo que contém a maioria das informações para o domínio *cs.vu.nl* (o arquivo foi editado, para simplificar). O arquivo mostra o conteúdo de vários nós que fazem parte do domínio *cs.vu.nl*, no qual cada nó é identificado por meio de seu nome de domínio.

O nó *cs.vu.nl* representa o domínio, bem como a zona. Seu registro de recurso *SOA* contém informações específicas sobre a validade desse arquivo, o que não tem grande importância para nós. Há quatro servidores de nomes para essa zona, referenciados por seus nomes canônicos de hospedeiros nos registros *NS*. O registro *TXT* é usado para dar algumas informações adicionais sobre essa zona, mas não pode ser processado automaticamente por qualquer servidor de nomes. Além do mais, há um único servidor de correio que pode manipular cor-

Nome	Tipo de registro	Valor do registro
cs.vu.nl.	SOA	star.cs.vu.nl. hostmaster.cs.vu.nl. 2005092900 7200 3600 2419200 3600
cs.vu.nl.	TXT	"Vrije Universiteit - Math. & Comp. Sc."
cs.vu.nl.	MX	1 mail.few.vu.nl.
cs.vu.nl.	NS	ns.vu.nl.
cs.vu.nl.	NS	top.cs.vu.nl.
cs.vu.nl.	NS	solo.cs.vu.nl.
cs.vu.nl.	NS	star.cs.vu.nl.
star.cs.vu.nl.	A	130.37.24.6
star.cs.vu.nl.	A	192.31.231.42
star.cs.vu.nl.	MX	1 star.cs.vu.nl.
star.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
star.cs.vu.nl.	HINFO	"Sun" "Unix"
zephyr.cs.vu.nl.	A	130.37.20.10
zephyr.cs.vu.nl.	MX	1 zephyr.cs.vu.nl.
zephyr.cs.vu.nl.	MX	2 tornado.cs.vu.nl.
zephyr.cs.vu.nl.	HINFO	"Sun" "Unix"
ftp.cs.vu.nl.	CNAME	soling.cs.vu.nl.
www.cs.vu.nl.	CNAME	soling.cs.vu.nl.
soling.cs.vu.nl.	A	130.37.20.20
soling.cs.vu.nl.	MX	1 soling.cs.vu.nl.
soling.cs.vu.nl.	MX	666 zephyr.cs.vu.nl.
soling.cs.vu.nl.	HINFO	"Sun" "Unix"
vucs-das1.cs.vu.nl.	PTR	0.198.37.130.in-addr.arpa.
vucs-das1.cs.vu.nl.	A	130.37.198.0
inkt.cs.vu.nl.	HINFO	"OCE" "Proprietary"
inkt.cs.vu.nl.	A	192.168.4.3
pen.cs.vu.nl.	HINFO	"OCE" "Proprietary"
pen.cs.vu.nl.	A	192.168.4.2
localhost.cs.vu.nl.	A	127.0.0.1

Tabela 5.4 Excerto do banco de dados DNS para a zona cs.vu.nl.

reio endereçado aos usuários nesse domínio. O número que precede o nome de um servidor de correio especifica uma prioridade de seleção. Um servidor que remete correio sempre deve tentar contatar em primeiro lugar o servidor de correio que tenha o número mais baixo.

O hospedeiro *star.cs.vu.nl* opera como um servidor de nomes para essa zona. Servidores de nomes são críticos para qualquer serviço de nomeação. O diferencial desse servidor de nomes é que ele possui duas interfaces de rede separadas, para robustez adicional, cada uma representada por um registro de recurso A separado. Desse modo, os efeitos da interrupção de uma ligação com a rede seriam, até certo ponto, amenizados, porque o servidor continuaria acessível.

As quatro linhas seguintes (para *zephyr.cs.vu.nl*) dão as informações necessárias sobre um dos servidores de correio do departamento. Note que esse servidor de correio é também apoiado por um outro servidor de correio, cujo caminho é *tornado.cs.vu.nl*.

As seis linhas seguintes mostram uma configuração típica na qual o servidor Web do departamento, bem como

o servidor FTP do departamento, é implementado por uma única máquina, denominada *soling.cs.vu.nl*. Executar ambos os servidores na mesma máquina (e, em essência, usar essa máquina só para serviços de Internet e nada mais) facilita o gerenciamento do sistema. Por exemplo, ambos os servidores terão a mesma visão do sistema de arquivos e, por questão de eficiência, parte do sistema de arquivos pode ser implementada em *soling.cs.vu.nl*. Essa abordagem muitas vezes é aplicada no caso de serviços WWW e FTP.

As duas linhas seguintes mostram informações sobre um dos mais antigos clusters de servidores do departamento. Nesse caso, elas nos informam que o endereço 130.37.198.0 está associado com o nome de hospedeiro *vucs-das1.cs.vu.nl*.

As quatro linhas seguintes mostram informações sobre duas impressoras importantes conectadas à rede local. Note que os endereços na faixa 192.168.0.0 a 192.168.255.255 são privados: eles só podem ser acessados de dentro da rede local e não estão acessíveis a um hospedeiro arbitrário da Internet.

Nome	Tipo de registro	Valor do registro
vu.nl.	NS	solo.cs.vu.nl.
vu.nl.	NS	star.cs.vu.nl.
vu.nl.	NS	ns.vu.nl.
vu.nl.	NS	top.cs.vu.nl.
vu.nl.	A	130.37.129.4
cs.vu.nl.	A	130.37.20.4
:cs.vu.nl.	A	130.37.20.5
:cs.vu.nl.	A	130.37.24.6
:cs.vu.nl	A	192.31.231.42

Tabela 5.5 Parte da descrição para o domínio vu.nl que contém o domínio cs.vu.nl.

Como o domínio cs.vu.nl é implementado como uma única zona, a Tabela 5.4 não inclui referências a outras zonas. O modo de referenciar nós de um subdomínio implementados em uma zona diferente é mostrado na Tabela 5.5. Para fazer isso é preciso especificar um servidor de nomes para o subdomínio simplesmente dando seu nome de domínio e endereço IP. Ao resolver um nome para um nó que se encontra no domínio cs.vu.nl, a certa altura a resolução de nomes prosseguirá pela leitura do banco de dados DNS armazenado pelo servidor de nomes para o domínio cs.vu.nl.

Implementações de DNS descentralizadas

A implementação de DNS que descrevemos até aqui é a padrão. Ela segue uma hierarquia de servidores com 13 servidores-raiz bem conhecidos e termina em milhões de servidores nas folhas. Uma observação importante é que nós de níveis mais altos recebem quantidade muito maior de requisições do que os nós de nível mais baixo. O único modo de impedir que sejam enviadas requisições a esses nós e que, portanto, eles sejam afogados é manter caches de vinculações nome–endereço desses níveis mais altos.

Esses problemas de escalabilidade podem ser completamente evitados com soluções totalmente descentralizadas. Em particular, podemos calcular o hash de um nome DNS e, na sequência, tomar esse hash como um valor de chave a ser consultado em uma tabela de hash distribuída ou em um serviço de localização hierárquica com um nó-raiz totalmente particionado. A desvantagem óbvia dessa abordagem é que perdemos a estrutura do nome original. Essa perda pode impedir implementações eficientes, por exemplo, para achar todos os filhos em um domínio específico.

Por outro lado, há muitas vantagens em mapear DNS para uma implementação baseada em DHT, em particular, sua escalabilidade. Como argumentaram Walfish et al. (2004), quando há necessidade de muitos nomes, usar identificadores como um modo livre de semântica de acessar dados permitirá que sistemas diferentes usem um único sistema de nomeação. A razão é simples: a essa altura já entendemos bem como um conjunto enorme de nomes (simples) pode ser suportado com eficiência. O que preci-

sa ser feito é manter o mapeamento de informações identificador-nome, no qual, nesse caso, um nome pode vir do espaço DNS, pode ser um URL e assim por diante.

A utilização de identificadores pode se tornar mais fácil ao permitir que usuários ou organizações usem um espaço de nomes local e estrito. Esse espaço é completamente análogo a manter um conjunto privado de variáveis de ambiente em um computador.

O mapeamento de DNS para sistemas peer-to-peer baseados em DHT foi explorado em CoDoNS (Ramasubramanian e Sirer, 2004a). Eles utilizaram um sistema baseado em DHT no qual os prefixos de chaves são usados para rotear para um nó. Como explicação, considere o caso em que cada dígito de um identificador é retirado do conjunto $\{0, \dots, b-1\}$, onde b é o número-base. Por exemplo, em Chord, $b = 2$. Supondo que $b = 4$, então considere um nó cujo identificador seja 3210. No sistema daqueles autores, adota-se como premissa que esse nó mantém uma tabela de roteamento de nós que tem os seguintes identificadores:

- n_0 : um nó cujo identificador tem prefixo 0
- n_1 : um nó cujo identificador tem prefixo 1
- n_2 : um nó cujo identificador tem prefixo 2
- n_{30} : um nó cujo identificador tem prefixo 30
- n_{31} : um nó cujo identificador tem prefixo 31
- n_{33} : um nó cujo identificador tem prefixo 33
- n_{320} : um nó cujo identificador tem prefixo 320
- n_{322} : um nó cujo identificador tem prefixo 322
- n_{323} : um nó cujo identificador tem prefixo 323

O nó 3210 é responsável por manipular chaves que tenham prefixo 321. Se esse nó receber uma requisição de consulta para a chave 3123, ele a repassará para o nó n_{31} que, por sua vez, verificará se precisa repassá-la para um nó cujo identificador tenha prefixo 312. (Devemos notar que cada nó mantém duas outras listas que ele pode usar para rotear, caso perca uma entrada em sua tabela de roteamento.) Detalhes dessa abordagem podem ser encontrados para Pastry em Rowstron e Druschel (2001) e para Tapestry em Zhao et al. (2004).

Voltando ao CoDoNS, um nó responsável pela chave k armazena os registros de recursos DNS associados com o nome de domínio cujo hash é k . A parte interessante, entretanto, é que o CoDoNS tenta minimizar o número de saltos ao rotear uma requisição, replicando os registros de recursos. A principal estratégia é simples: o nó 3210 replicará seu conteúdo para nós que tenham prefixo 321. Essa replicação reduzirá de um salto cada caminho de roteamento que termina no nó 3210. Claro que essa replicação pode ser aplicada novamente a todos os nós que tenham prefixo 32 e assim por diante.

Quando um registro DNS é replicado para todos os nós que tenham i prefixos idênticos, diz-se que foi replicado no nível i . Observe que um registro replicado no nível i (geral-

mente) requer i etapas de consulta para ser encontrado. Contudo, há um compromisso entre o nível de replicação e a utilização de recursos de rede e de nós. O que o CoDoNS faz é replicar até o ponto em que a latência agregada de consulta resultante seja menor do que uma dada constante C .

Mais especificamente, pense um pouco sobre a distribuição de freqüência das consultas. Imagine que as requisições de consulta sejam classificadas pelo número de vezes que uma chave específica é requisitada e que a chave mais requisitada ocupe a primeira posição. A distribuição das consultas é denominada **tipo Zipf** se a freqüência do *enésimo* item classificado for proporcional a $\frac{1}{i^\alpha}$, com α próximo de 1. George Zipf foi um especialista em lingüística de Harvard que descobriu essa distribuição enquanto estudava as freqüências de utilização de palavras em uma língua natural. Contudo, ocorre que ela também se aplica, entre muitas outras coisas, a população de cidades, dimensões de terremotos, distribuições de alta renda, receitas de empresas e, talvez, de modo não surpreendente, a requisições DNS (Jung et al., 2002).

Agora, se x_i é a fração dos registros mais populares que deverão ser replicados no nível i , Ramasubramanian e Sirer (2004b) mostram que x_i pode ser expresso pela seguinte fórmula (para nossa finalidade, só é importante saber que essa fórmula existe; em breve veremos como usá-la):

$$x_i = \left[\frac{d(\log N - C)}{1 + d + \dots + d^{\log N - 1}} \right]^{\frac{1}{(1-\alpha)}} \text{ com } d = b^{(1-\alpha)/\alpha}$$

onde N é o número de nós na rede e α é o parâmetro na distribuição Zipf.

Essa fórmula permite tomar decisões conscientes sobre quais registros DNS devem ser replicados. Para exemplificar de maneira mais concreta, considere o caso em que $b = 32$ e $\alpha = 0,9$. Assim, em uma rede com 10.000 nós e 1.000.000 de registros DNS, e tentando conseguir uma média de $C=1$ salto somente quando estivermos fazendo uma consulta, teremos que $x_0 = 0,0000701674$, o que significa que somente os 70 registros DNS mais populares devem ser replicados em todos os lugares. Da mesma maneira, com $x_1 = 0,00330605$, os 3.306 registros mais populares seguintes devem ser replicados no nível 1. Claro que é obrigatório que $x_i < 1$. Nesse exemplo, $x_2 = 0,155769$ e $x_3 > 1$, portanto somente os 155.769 registros mais populares seguintes são replicados; os outros não. Não obstante, na média, um único salto é suficiente para achar um registro DNS requisitado.

estruturados foram projetados, em parte, para oferecer uma maneira de nomear entidades que fosse amigável aos seres humanos, de modo que possam ser convenientemente acessados. Na maioria dos casos, a premissa é de que o nome se refere a uma única entidade. Contudo, independência de localização e ser amigável a seres humanos não são os únicos critérios para nomeação de entidades. Em particular, à medida que há cada vez mais informações disponíveis, torna-se mais importante procurar entidades com certa eficiência. Essa abordagem requer que um usuário possa fornecer uma simples descrição do que ele está procurando.

Há muitos modos de fornecer descrições, mas um que é muito usado em sistemas distribuídos é descrever uma entidade em termos de pares (*atributo, valor*), em geral denominada **nomeação baseada em atributos**. Nessa abordagem, adota-se como premissa que uma entidade tem um conjunto associado de atributos. Cada atributo diz algo sobre essa entidade. Quando um usuário especifica quais valores um determinado atributo deve ter, em essência, ele restringe o conjunto de entidades nas quais está interessado. Cabe ao sistema de nomeação retornar uma ou mais entidades que atendam à descrição do usuário. Nesta seção, vamos examinar mais de perto um sistema de nomeação baseado em atributos.

5.4.1 Serviços de diretório

Sistemas de nomeação baseados em atributos também são conhecidos como **serviços de diretório**, ao passo que sistemas que suportam nomeação estruturada são geralmente denominados **sistemas de nomeação**. Com serviços de diretório, entidades têm um conjunto de atributos associados que podem ser usados para procurá-las. Em muitos casos, a escolha de atributos pode ser relativamente simples. Por exemplo, em um sistema de e-mail, mensagens podem ser rotuladas com atributos para o remetente, o receptor, o assunto e assim por diante. Contudo, mesmo no caso do e-mail, as coisas ficam difíceis quando são necessários outros tipos de descritores, como ilustrado pela dificuldade de desenvolver filtros que permitirão somente a passagem de certas mensagens (com base em seus descritores).

No fundo, isso quer dizer que projetar um conjunto apropriado de atributos não é algo trivial. Na maioria dos casos, o projeto de atributos tem de ser feito manualmente. Ainda que haja consenso quanto ao conjunto de atributos a usar, a prática mostra que o ajuste consistente de valores por um grupo variado de pessoas é um problema por si mesmo, como muitos podem ter percebido ao acessar bancos de dados de música e vídeo na Internet.

Para amenizar alguns desses problemas, foram realizadas pesquisas para unificar os modos como esses recursos podem ser descritos. No contexto de sistemas distribuídos, um desenvolvimento particularmente relevante é a

5.4 Nomeação Baseada em Atributo

De modo geral, nomes simples e nomes estruturados proporcionam um modo exclusivo e independente de localização para referenciar entidades. Ademais, nomes

estrutura de descrição de recurso (resource description framework — RDF). Fundamental para o modelo RDF é que os recursos são descritos como triplas que consistem em um sujeito, um predicado e um objeto. Por exemplo, (*Pessoa, nome, Alice*) descreve um recurso *Pessoa* cujo *nome* é *Alice*. Em RDF, cada sujeito, predicado ou objeto pode ser ele mesmo um recurso. Isso significa que *Alice* pode ser implementado como referência a um arquivo que, na seqüência, pode ser recuperado. No caso de um predicado, tal recurso poderia conter uma descrição textual desse predicado. É claro que recursos associados com sujeitos e objetos poderiam ser qualquer coisa. Referências em RDF são, em essência, URLs.

Se as descrições de recursos forem armazenadas, torna-se possível consultar aquele armazenamento de modo que seja comum para muitos sistemas de nomeação baseados em atributos. Por exemplo, uma aplicação poderia solicitar a informação associada com uma pessoa chamada Alice. Tal consulta retornaria uma referência ao recurso *pessoa* associado com Alice. Então, na seqüência, esse recurso pode ser buscado pela aplicação. Mais informações sobre RDF podem ser encontradas em Manola e Miller (2004).

Nesse exemplo, as descrições de recursos são armazenadas em uma localização central. Não há nenhuma razão por que os recursos também tenham de residir na mesma localização. Entretanto, não ter as descrições no mesmo lugar pode resultar em sérios problemas de desempenho. Diferentemente de sistemas estruturados de nomeação, consultar valores em um sistema de nomeação baseado em atributos requer, em essência, uma exaustiva busca em todos os descriptores. Quando se considera desempenho, tal busca é menos problemática dentro de um único armazém de dados, mas é preciso aplicar técnicas especiais quando os dados estão distribuídos por muitos computadores, potencialmente dispersos. Na seção seguinte, vamos estudar diferentes abordagens para resolver esse problema em sistemas distribuídos.

5.4.2 Implementações hierárquicas: LDAP

Uma abordagem comum para tratar serviços distribuídos de diretório é combinar nomeação estruturada com

nomeação baseada em atributos. Essa abordagem tem sido amplamente adotada, por exemplo, no serviço Active Directory da Microsoft e em outros sistemas. Muitos desses sistemas usam, ou dependem, do **protocolo leve de acesso a diretório**, referido simplesmente como **LDAP (lightweight directory access protocol)**. O serviço de diretório LDAP foi derivado do serviço de diretório X.500 do modelo OSI. Como muitos serviços OSI, a qualidade de suas implementações associadas atrapalhou uma utilização mais ampla e, para torná-lo funcional, foi preciso fazer simplificações. Informações detalhadas sobre o LDAP podem ser encontradas em Arkills (2003).

Conceitualmente, um serviço de diretório LDAP consiste em vários registros, usualmente conhecidos como entradas de diretório. Uma entrada de diretório é comparável a um registro de recurso em DNS. Cada registro é composto de um conjunto de pares (*atributo, valor*), no qual cada atributo tem um tipo associado. É feita uma distinção entre atributos de valor único e atributos de valores múltiplos. Os últimos representam normalmente vetores e listas. Como exemplo, uma entrada de diretório simples que identifica endereços de rede de alguns servidores gerais da Tabela 5.4 é mostrada na Tabela 5.6.

Em nosso exemplo, usamos uma convenção de nomeação descrita nos padrões LDAP, que se aplica aos cinco primeiros atributos. Os atributos *Organization* e *OrganizationalUnit* descrevem, respectivamente, a organização e o departamento associados com os dados que estão armazenados no registro. Da mesma forma, os atributos *Locality* e *Country* fornecem informações adicionais sobre o lugar em que a entrada está armazenada. O atributo *CommonName* costuma ser usado como um nome (ambíguo) para identificar uma entrada dentro de uma parte limitada do diretório. Por exemplo, o nome ‘Main server’ pode ser suficiente para achar a entrada que usamos como exemplo, dados os valores específicos para os outros quatro atributos: *Country*, *Locality*, *Organization* e *OrganizationalUnit*. Em nosso exemplo, somente o atributo *Mail_Servers* tem múltiplos valores associados a ele. Todos os outros atributos têm apenas um único valor.

O conjunto de todas as entradas de diretório em um serviço de diretório LDAP é denominado **base de informações**.

Atributo	Abreviatura	Valor
Country	C	NL
Locality	L	Amsterdam
Organization	O	Vrije Universiteit
OrganizationalUnit	OU	Comp. Sc
CommonName	CN	Main server
Mail_Servers	—	137.37.20.3, 130.37.24.6, 137.37.20.10
FTP_Server	—	130.37.20.20
WWW_Server	—	130.37.20.20

Tabela 5.6 Exemplo simples de uma entrada de diretório LDAP que utiliza convenções de nomeação LDAP.

magões de diretório (directory information base — DIB). Um aspecto importante de uma DIB é que cada registro é nomeado exclusivamente, de modo que possa ser consultado. Tal nome globalmente exclusivo aparece como uma sequência de atributos de nomeação em cada registro. Cada atributo de nomeação é denominado **nome relativo distinguido** ou, abreviadamente, **RDN (relative distinguished name)**. Em nosso exemplo na Tabela 5.6, os cinco primeiros atributos são todos atributos de nomeação. Usando as abreviaturas convencionais para representar atributos de nomeação em LDAP, como mostra a Tabela 5.6, os atributos *Country*, *Organization* e *OrganizationalUnit* poderiam ser usados para formar o nome globalmente exclusivo análogo ao nome DNS *nl.vu.cs.*

C=NL/O=Vrije Universiteit/OU=Comp. Sc.

Como em DNS, a utilização de nomes globalmente exclusivos pela listagem de RDNs em seqüência resulta em uma hierarquia da coleção de entradas de diretório, que é denominada **árvore de informações de diretório (directory information tree — DIT)**. Em essência, uma DIT forma o gráfico de nomeação de um serviço de diretório LDAP no qual cada nó representa uma entrada de diretório. Além disso, um nó também pode agir como um diretório no sentido tradicional, já que podem existir vários filhos para os quais o nó age como pai. Para explicar, considere o gráfico de nomeação mostrado parcialmente na Figura 5.17(a). (Lembre-se de que rótulos são associados com ramos.)

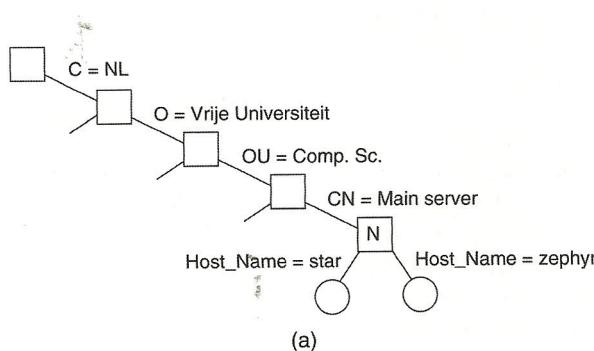
O nó *N* corresponde à entrada de diretório mostrada na Tabela 5.6. Esse nó age, ao mesmo tempo, como um pai para várias outras entradas de diretório que têm atributo adicional de nomeação *Host_Name*, que é usado como um RDN. Por exemplo, tais entradas podem ser usadas para representar hospedeiros, conforme mostrado na Figura 5.17(b).

Portanto, um nó em um gráfico de nomeação LDAP pode representar simultaneamente um diretório no sentido tradicional, como já descrevemos antes, bem como um registro LDAP. Essa distinção é suportada por duas operações de consulta diferentes. A operação *read* é usada para ler um único registro, dado seu nome de caminho na DIT. Ao contrário, a operação *list* é usada para apresentar uma lista dos nomes de todos os ramos de saída de um dado nó na DIT. Cada nome corresponde a um nó-filho do nó dado. Note que a operação *list* não retorna nenhum registro; ela se limita a retornar nomes. Em outras palavras, chamar *read* tendo como entrada o nome

/C=NL/O=Vrije Universiteit/OU=Comp. Sc./CN=Main server

retornará o registro mostrado na Tabela 5.6, ao passo que chamar *list* retornará os nomes *star* e *zephyr* das entradas mostradas na Figura 5.17(b), bem como os nomes de outros hospedeiros que foram registrados de maneira semelhante.

A implementação de um serviço de diretório LDAP ocorre de modo muito parecido com a implementação de um serviço de nomeação como DNS, exceto que LDAP suporta mais operações de consulta, como discutiremos em



(a)

Atributo	Valor
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	star
Host_Address	192.31.231.42

Atributo	Valor
Country	NL
Locality	Amsterdam
Organization	Vrije Universiteit
OrganizationalUnit	Comp. Sc.
CommonName	Main server
Host_Name	zephyr
Host_Address	137.37.20.10

(b)

Figura 5.17 (a) Parte de um árvore de informações de diretório.

(b) Duas entradas de diretório que têm *Host_Name* como RDN.

breve. Quando estamos lidando com um diretório de grande escala, a DIT normalmente é particionada e distribuída por vários servidores, conhecidos como **agentes de serviço de diretório (directory service agents — DSA)**. Portanto, cada porção de uma DIT particionada corresponde a uma zona em DNS. Da mesma maneira, cada DSA se comporta de modo muito parecido com o de um servidor de nomes normal, exceto que ele implementa vários serviços típicos de diretório, como operações avançadas de busca.

Clientes são representados pelo que denominamos **agentes de usuário de diretório (directory user agents)** ou, simplesmente, **DUA**. Um DUA é semelhante a um resolvedor de nomes em um serviço estruturado de nomeação. Um DUA troca informações com um DSA de acordo com um protocolo de acesso padronizado.

O que faz uma implementação LDAP diferente de uma implementação DNS são os recursos de busca por meio de uma DIB. Em particular, são fornecidos mecanismos para procurar uma entrada de diretório dado um conjunto de critérios que os atributos das entradas procuradas devem atender. Por exemplo, suponha que queiramos fazer uma lista de todos os servidores principais na Universidade de Vrije. Usando a notação definida em Howes (1997), essa lista pode ser retornada usando uma operação de busca tal como

```
answer = search("(&(C=NL)(O=Vrije Universiteit)(OU=*)(CN=Main server)")
```

Nesse exemplo, especificamos que o lugar no qual procurar servidores principais é a organização denominada *Vrije Universiteit* no país *NL*, mas que não estamos interessados em determinada unidade organizacional. Contudo, cada resultado retornado deve ter o atributo *CN* igual a *Main server*.

Como já mencionamos, a busca em um serviço de diretório é, em geral, uma operação cara. Por exemplo, achar todos os servidores principais na Universidade de Vrije requer que façamos uma busca em todas as entradas de cada departamento e que combinemos os resultados em uma única resposta. Em outras palavras, em geral precisaremos acessar vários nós-folha de uma DIT para obter uma resposta. Na prática, isso também significa que é preciso acessar vários DSAs. Como exemplo, podemos citar serviços de nomeação que, muitas vezes, podem ser implementados de modo tal que uma operação de consulta requeira acessar somente um único nó-folha.

Toda essa instalação de LDAP pode ser levada um passo mais adiante, permitindo a coexistência de várias árvores, contanto que também estejam ligadas umas às outras. Essa abordagem é seguida no Active Directory da Microsoft, o que resulta em uma *floresta* de domínios LDAP (Allen e Lowe-Norris, 2003). É óbvio que a busca em tal organização pode ser extremamente complexa. Para contornar alguns dos problemas de escalabilidade, o

Active Directory usualmente entende que há um servidor global de índices (denominado catálogo global) que pode ser procurado antes. O índice indicará quais domínios LDAP precisam ser pesquisados ainda mais.

Embora o próprio LDAP já explore a hierarquia por causa da escalabilidade, é comum combinar LDAP com DNS. Por exemplo, toda árvore em LDAP precisa ser acessível na raiz, conhecida no Active Directory como controlador de domínio. A raiz freqüentemente é conhecida sob um nome DNS que, por sua vez, pode ser encontrado por meio de um registro SRV adequado, como já explicamos.

O LDAP representa tipicamente um modo padrão de suportar nomeação baseada em atributos. Recentemente também foram desenvolvidos outros serviços de diretório que seguem essa abordagem mais tradicional, em particular no contexto de computação em grade e serviços Web. Um exemplo específico é a **integração universal de diretório e descoberta (universal directory and discovery integration)**, ou apenas **UDDI**.

Esses serviços consideram uma implementação em que um nó, ou tão-somente alguns nós, coopere para manter um banco de dados distribuído simples. Do ponto de vista tecnológico, isso não é, realmente, uma novidade. Da mesma maneira, também não há nada de realmente novo a dizer quando se trata de introduzir terminologia, como pode-se observar imediatamente ao folhear as centenas de páginas das especificações UDDI (Clement et al., 2004). O esquema fundamental é sempre o mesmo: consigue-se escalabilidade fazendo com que vários desses bancos de dados fiquem acessíveis para aplicações, que então são responsáveis por pesquisar cada banco de dados em separado e agragar os resultados. Não há nada mais a dizer sobre suporte de middleware.

5.4.3 Implementações descentralizadas

Com o advento de sistemas peer-to-peer, os pesquisadores têm procurado soluções para descentralizar sistemas de nomeação baseados em atributos. Nesse caso, a questão fundamental é que pares (*atributo, valor*) precisam ser mapeados com eficiência para que a busca também possa ser realizada com eficiência, isto é, para que se evite uma busca exaustiva por toda a extensão do espaço de atributos. A seguir examinaremos vários modos de estabelecer tal mapeamento.

Mapeamento para tabelas de hash distribuídas

Em primeiro lugar, vamos considerar o caso em que pares (*atributo, valor*) precisam ser suportados por um sistema baseado em DHT. Antes de mais nada, adote como premissa que consultas consistem em uma conjunção de pares, como acontece com LDAP, ou seja, um usuário especifica uma lista de atributos, junto com o valor único que ele quer ver para cada atributo respectivo. A principal vantagem desse tipo de consulta é que não é

preciso suportar nenhuma faixa. Consultas por faixa podem aumentar significativamente a complexidade do mapeamento de pares para uma DHT.

Consultas de um único valor são suportadas no sistema INS/Twine (Balazinska et al., 2002). Considera-se que cada entidade (referida como um recurso) seja descrita por meio de atributos possivelmente organizados em hierarquia, como mostra a Figura 5.18.

Cada uma dessas descrições é traduzida para uma **árvore de valores de atributos (attribute-value tree — AVTree)**, que, então, é usada como a base para uma codificação que mapeia para um sistema baseado em DHT.

A questão principal é transformar as AVTrees em um conjunto de chaves que possa ser consultado em um sistema DHT. Nesse caso, a cada caminho que se origina na raiz é designado um único valor de hash, no qual uma descrição de caminho começa com uma ligação (que representa um atributo) e termina em um nó (valor) ou em uma outra ligação. Tomando a Figura 5.18(b) como nosso exemplo, os seguintes hashes de tais caminhos são considerados:

- h_1 : hash(tipo-livro)
- h_2 : hash(tipo-livro-autor)
- h_3 : hash(tipo-livro-autor-Tolkien)
- h_4 : hash(tipo-livro-título)
- h_5 : hash(tipo-livro-título-LOTR)
- h_6 : hash(gênero-fantasia)

Um nó responsável pelo valor de hash h_i manterá (uma referência para) o recurso propriamente dito. Em nosso exemplo, isso pode resultar em seis nós que armazenam o livro de Tolkien *O Senhor dos Anéis* (Lord of the Rings —

LOTR). Contudo, o benefício dessa redundância é que ela permitirá suportar consultas parciais. Por exemplo, considere uma consulta como ‘Retornar livros escritos por Tolkien’. Essa consulta é traduzida na AVTree mostrada na Figura 5.19 que resulta no cálculo dos seguintes três hashes:

- h_1 : hash(tipo-livro)
- h_2 : hash(tipo-livro-autor)
- h_3 : hash(tipo-livro-autor-Tolkien)

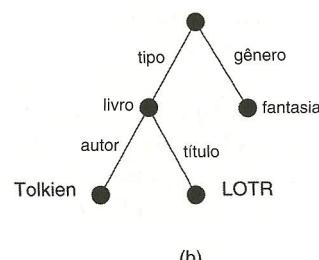
Esses valores serão enviados a nós que armazenam informações sobre livros de Tolkien e, no mínimo, retornarão ‘*O Senhor dos Anéis*’. Note que um hash como h_1 é bastante geral e será gerado com freqüência. Esses tipos de hashes podem ser filtrados para fora do sistema. Além do mais, não é difícil ver que somente os hashes mais específicos precisam ser avaliados. Mais detalhes podem ser encontrados em Balazinska et al. (2002).

Agora, vamos examinar um outro tipo de consulta, a saber, as que podem conter especificações de faixa para valores de atributos. Por exemplo, alguém que esteja procurando uma casa em geral quer especificar que o preço deve cair dentro de determinada faixa. Novamente, várias soluções foram propostas e nós veremos algumas delas quando discutirmos sistemas publicar/subscrever no Capítulo 13. Aqui, discutiremos uma solução adotada no sistema de descoberta de recurso SWORD (Oppenheimer et al., 2005).

Em SWORD, pares (*atributo, valor*) como fornecidos por uma descrição de recurso são primeiro transformados em uma chave para uma DHT. Note que esses pares sempre contêm um único valor; somente consultas podem conter faixas de valores para atributos. Ao calcu-

```
descrição {
    tipo = livro
    descrição {
        autor = Tolkien
        título = LOTR
    }
    gênero = fantasia
}
```

(a)

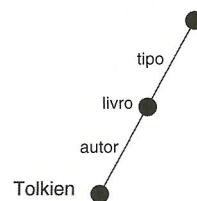


(b)

Figura 5.18 (a) Descrição geral de um recurso. (b) Sua representação como uma AVTree.

```
descrição {
    tipo = livro
    descrição {
        autor = Tolkien
        título = *
    }
    gênero = *
}
```

(a)



(b)

Figura 5.19 (a) Descrição de recurso de uma consulta. (b) Sua representação como uma AVTree.

lar o hash, o nome do atributo e seu valor são mantidos em separado. Em outras palavras, bits específicos na chave resultante identificarão o nome do atributo, enquanto outros identificarão seu valor. Além disso, a chave conterá alguns bits aleatórios para garantir a exclusividade entre todas as chaves que precisam ser geradas.

Desse modo, o espaço de atributos é convenientemente particionado: se n bits forem reservados para codificar nomes de atributos, 2^n grupos diferentes de servidores podem ser usados, um grupo para cada nome de atributo. Da mesma maneira, usando m bits para codificar valores, pode-se aplicar uma repartição adicional por grupo de servidores para armazenar pares específicos (*atributo, valor*). DHTs são usadas somente para distribuir nomes de atributos.

A faixa de valor possível para cada nome de atributo é particionada em subfaixas, e um único servidor é designado para cada subfaixa. Para explicar, considere uma descrição de recurso com dois atributos: a_1 , que toma valores na faixa [1..10], e a_2 , que toma valores na faixa [101...200]. Considere que há dois servidores para a_1 : s_{11} cuida de registrar valores de a_1 em [1..5], e s_{12} , valores na faixa [6..10]. Do mesmo modo, o servidor s_{21} registra valores para a_2 na faixa [101..150], e o servidor s_{22} , valores na faixa [151..200]. Portanto, quando um recurso obtém valores ($a_1 = 7, a_2 = 175$), o servidor s_{12} e o servidor s_{22} terão de ser informados.

A vantagem desse esquema é que consultas por faixa podem ser suportadas com facilidade. Quando é emitida uma consulta para retornar recursos que têm a_2 entre 165 e 189, ela pode ser repassada para o servidor s_{22} , que então pode retornar os recursos que combinam com a faixa de consulta. A desvantagem, entretanto, é que é preciso enviar atualizações a vários servidores. Além do mais, não fica imediatamente clara a qualidade do balançoamento de carga entre os vários servidores.

Em particular, se certas consultas por faixa mostrarem ser muito populares, servidores específicos receberão alta fração de todas as consultas. Bharambe et al. (2004) discutem como esse problema do balanceamento de carga pode ser atacado para sistemas baseados em DHT.

Redes de sobreposição semântica

As implementações descentralizadas de nomeação baseada em atributos já mostraram crescente grau de autonomia dos vários nós. O sistema é menos sensível à entrada e saída de nós em comparação com, por exemplo, sistemas distribuídos baseados em LDAP. Esse grau de autonomia é aumentado quando nós têm descrições de recursos que lá estão para serem descobertos por outros. Em outras palavras, não há nenhum esquema determinístico *a priori* pelo qual os pares (*atributo, valor*) são espalhados por um conjunto de nós.

Não ter tal esquema força os nós a descobrir onde estão os recursos requisitados. Tal descoberta é típica para redes de sobreposição não estruturadas, que já discutimos

no Capítulo 2. Para tornar a busca eficiente, é importante que um nó tenha referências a outros que muito provavelmente responderão a suas consultas.

Se adotarmos como premissa que as consultas originais do nó P estão fortemente relacionadas com os recursos que ele tem, estamos procurando fornecer a P um conjunto de ligações com vizinhos que lhe são *semanticamente próximos*. Lembre-se de que tal lista também é conhecida como **visão parcial**. Proximidade semântica pode ser definida de modos diferentes mas, em essência, ela se resume em monitorar nós que tenham recursos semelhantes. Portanto, os nós e essas ligações formarão o que é conhecido como **rede de sobreposição semântica**.

Uma abordagem comum de redes de sobreposição semântica é considerar que existe algo em comum entre as metainformações mantidas em cada nó. Em outras palavras, os recursos armazenados em cada nó são descritos com a utilização do mesmo conjunto de atributos ou, mais exatamente, o mesmo esquema de dados (Crespo e Garcia-Molina, 2003). Ter tal esquema permitirá definir funções específicas de similaridade entre nós. Sendo assim, cada nó manterá ligações só com os K vizinhos mais semelhantes a ele e consultará esses nós em primeiro lugar quando estiver procurando dados específicos. Note que essa abordagem só faz sentido se pudermos considerar, de modo geral, que uma consulta iniciada em um nó esteja relacionada com o conteúdo armazenado nesse nó.

Infelizmente, considerar coisas em comum em esquemas de dados é, geralmente, errado. Na prática, as metainformações sobre recursos apresentam alto grau de inconsistência entre nós diferentes, e chegar a um consenso sobre como descrever recursos é quase impossível. Por essa razão, normalmente as redes de sobreposição semântica precisarão encontrar modos diferentes para definir similaridade.

Uma abordagem é esquecer totalmente os atributos e considerar somente descritores muito simples como nomes de arquivos. A construção passiva de uma sobreposição pode ser feita com a monitoração de quais nós respondem positivamente a buscas em arquivos. Por exemplo, Sripanidkulchai et al. (2003) primeiro enviam uma consulta aos vizinhos semânticos de um nó; porém, se o arquivo requisitado não estiver ali, é feito um broadcast (limitado). Claro que tal broadcast pode resultar em uma atualização da lista de vizinhos semânticos. Para fins de observação, é interessante perceber que, se um nó requisitar que seus vizinhos semânticos repassem uma consulta para os vizinhos semânticos *deles*, o efeito será mínimo (Handurukande et al., 2004). Esse fenômeno pode ser explicado pelo que é conhecido como **efeito do mundo pequeno**, que, em essência, afirma que os amigos de Alice também são amigos uns dos outros (Watts, 1999).

Uma abordagem mais ativa em relação à construção de uma lista de vizinhos semânticos é proposta por Voulgaris e Van Steen (2005), que usam uma **função pro-**

Proximidade semântica definida nas listas de arquivos FL_P e FL_Q de dois nós P e Q , respectivamente. Essa função simplesmente conta o número de arquivos em comum em FL_P e FL_Q . Portanto, a meta é otimizar a função proximidade, permitindo que um nó mantenha uma lista só com os vizinhos que tenham a maioria dos arquivos em comum com ele.

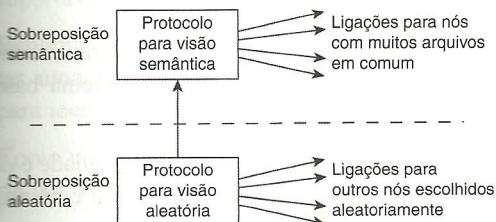


Figura 5.20 Manutenção de uma sobreposição semântica por meio de gossiping.

Com essa finalidade, um esquema de gossiping de duas camadas é oferecido, como mostra a *Figura 5.20*. A camada inferior consiste em um protocolo epidêmico que visa a manter uma visão parcial de nós uniformes selecionados aleatoriamente. Há maneiras diferentes de conseguir isso, como explicamos no Capítulo 2 [veja também Jelasity et al. (2005a)]. A camada superior mantém uma lista de vizinhos semanticamente próximos por meio de gossiping. Para iniciar uma troca, um nó P pode selecionar aleatoriamente um vizinho Q de sua lista corrente, mas o truque é deixar que P envie somente as entradas cuja semântica esteja mais próxima da semântica de Q . Por sua vez, quando P recebe entradas de Q , a certa altura ele manterá uma visão parcial que consiste somente nos nós cuja semântica esteja mais próxima da semântica de Q . Ocorre que as visões parciais mantidas pela camada superior convergirão rapidamente para um ponto ótimo.

Como já deve ter ficado claro a essa altura, redes de sobreposição semântica estão intimamente relacionadas com procura descentralizada. Uma visão geral extensiva da busca em todos os tipos de sistemas peer-to-peer é discutida em Risson e Moors (2006).

5.5 Resumo

Nomes são usados para referenciar entidades. Em essência, há três tipos de nomes. Um endereço é o nome de um ponto de acesso associado a uma entidade, também denominado simplesmente endereço de uma entidade. Um identificador é um outro tipo de nome. Ele tem três propriedades: cada entidade é referenciada por exatamente um identificador, um identificador referencia somente uma entidade e nunca é atribuído a uma outra entidade. Por fim, nomes amigáveis aos seres humanos visam à utilização por seres humanos e, como tal, são representados

por cadeias de caracteres. Dados esses tipos, fazemos uma distinção entre nomeação simples, nomeação estruturada e nomeação baseada em atributos.

Sistemas para nomeação simples precisam, em essência, resolver um identificador para o endereço de sua entidade associada. Essa localização de uma entidade pode ser feita de maneiras diferentes. A primeira abordagem é usar broadcasting ou multicasting. O identificador da entidade é transmitido por broadcast para todo processo no sistema distribuído. O processo que oferece um ponto de acesso para a entidade responde fornecendo um endereço para aquele ponto de acesso. É óbvio que essa abordagem é de limitada escalabilidade.

Uma segunda abordagem é usar ponteiros repassadores. Cada vez que uma entidade mudar para uma outra localização, deixará para trás um ponteiro que informa onde ela estará em seguida. Localizar a entidade requer percorrer o caminho de ponteiros repassadores. Para evitar grandes cadeias de ponteiros, é importante reduzi-las periodicamente.

Uma terceira abordagem é designar uma localização nativa a uma entidade. Cada vez que uma entidade mudar para uma outra localização, ela informa onde está à sua localização nativa. Localizar uma entidade requer primeiro perguntar à localização nativa qual é a localização corrente da entidade.

Uma quarta abordagem é organizar todos os nós em um sistema peer-to-peer estruturado e designar nós sistematicamente a entidades, levando em conta seus respectivos identificadores. Se, na seqüência, planejarmos um algoritmo de roteamento pelo qual as requisições de consulta sejam transmitidas na direção do nó responsável por dada entidade, é possível ter uma resolução de nomes robusta e eficiente.

Uma quinta abordagem é construir uma árvore de busca hierárquica. A rede é dividida em domínios, sem sobreposição. Domínios podem ser agrupados em domínios de nível mais alto (não sobrepostos) e assim por diante. Há um único domínio de nível alto que abrange toda a extensão da rede. Cada domínio em cada nível tem um nó de diretório associado. Se uma entidade estiver localizada em um domínio D , o nó de diretório do próximo domínio de nível mais alto terá um ponteiro para D . Um nó de diretório do nível mais baixo armazena o endereço da entidade. O nó de diretório do nível mais alto conhece todas as entidades.

Nomes estruturados são facilmente organizados em um espaço de nomes. Um espaço de nomes pode ser representado por um gráfico de nomeação no qual um nó representa uma entidade nomeada e o rótulo em um ramo representa o nome pelo qual a entidade é conhecida. Um nó que tenha vários ramos de saída representa um conjunto de entidades e também é conhecido como nó de contexto ou diretório. Gráficos de nomeação de grande escala são freqüentemente organizados como gráficos dirigidos acíclicos com raiz.

Gráficos de nomeação são convenientes para organizar nomes amigáveis aos seres humanos de modo estruturado. Uma entidade pode ser referenciada por um nome de caminho. Resolução de nomes é o processo de percorrer o gráfico de nomeação consultando os componentes de um nome de caminho, um por vez. Um gráfico de nomeação de grande escala é implementado pela distribuição de seus nós por vários servidores de nomes. Ao resolver um nome de caminho percorrendo o gráfico de nomeação, a resolução de nomes continua no próximo servidor de nomes tão logo seja alcançado um nó implementado por aquele servidor.

Mais problemáticos são os esquemas de nomeação baseados em atributos nos quais entidades são descritas por um conjunto de pares (*atributo, valor*). As consultas também são formuladas como tais pares e requerem, em essência, uma busca exaustiva por todos os descritores. Essa busca só é viável quando os descritores são armazenados em um único banco de dados. Contudo, foram inventadas soluções alternativas pelas quais os pares são mapeados para sistemas baseados em DHT, o que, na verdade, resulta em uma distribuição do conjunto de descritores de entidades.

Relacionada com a nomeação baseada em atributos está a substituição gradual da resolução de nomes por técnicas distribuídas de busca. Essa abordagem é seguida em redes de sobreposição semântica nas quais os nós mantêm uma lista de outros nós cujos conteúdos têm semelhança semântica. As listas semânticas permitem que ocorra uma busca eficiente porque em primeiro lugar são consultados os vizinhos imediatos e só depois que essa busca não tiver sucesso será utilizado um broadcast (limitado).

Problemas

1. Dê um exemplo de onde um endereço de uma entidade de *E* precisa ser resolvido para um outro endereço a fim de poder acessar *E*.
2. Você consideraria que um URL como <http://www.acme.org/index.html> é independente de localização? E o endereço <http://www.acme.nl/index.html>?
3. Dê alguns exemplos de identificadores verdadeiros.
4. Um identificador tem permissão de conter informações sobre a entidade que ele referencia?
5. Proponha um esquema para uma implementação eficiente de identificadores globalmente exclusivos.
6. Observe o sistema Chord como mostra a Figura 5.4 e considere que o nó 7 acabou de se juntar à rede. Qual seria sua tabela de derivação? Haveria quaisquer mudanças em outras tabelas de derivação?
7. Considere um sistema Chord baseado em DHT no qual k bits de um espaço de identificadores de m bits foram reservados para designar a superpares. Se os identificadores forem designados aleatoriamente, quantos superpares podemos esperar que um sistema de N nós tenha?
8. Se inserirmos um nó em um sistema Chord, precisaremos atualizar imediatamente todas as tabelas de derivação?
9. Qual é a maior desvantagem de consultas recursivas na resolução de uma chave em um sistema baseado em DHT?
10. Uma forma especial de localizar uma entidade é denominada anycasting, pela qual um serviço é identificado por meio de um endereço IP (veja, por exemplo, RFC 1546). O envio de uma requisição para um endereço anycast retorna uma resposta de um servidor que implementa o serviço identificado por aquele endereço anycast. Faça um esquema da implementação de um serviço anycast baseado no serviço de localização hierárquica descrito na Subseção 5.2.4.
11. Considerando que uma abordagem de duas camadas baseada em uma localização nativa seja uma especialização de um serviço hierárquico de localização, onde está a raiz?
12. Suponha que se saiba que uma entidade móvel específica quase nunca sairá de seu domínio *D* e, se sair, pode se esperar que logo volte. Como essa informação pode ser usada para aumentar a velocidade de operação em um serviço de localização hierárquica?
13. Em um serviço de localização hierárquica com uma profundidade de k , quantos registros de localização precisam ser atualizados, no máximo, quando uma entidade móvel mudar sua localização?
14. Considere uma entidade que muda da localização *A* para *B*, passando por várias localizações intermediárias onde residirá apenas por tempo relativamente curto. Quando chega a *B*, ela se acomoda por um tempo. A mudança de um endereço em um serviço de localização hierárquica ainda pode levar um tempo relativamente longo para ser concluída e, portanto, deve ser evitada quando a entidade estiver visitando uma localização intermediária. Como essa entidade pode ser localizada em uma localização intermediária?
15. O nó-raiz em serviços de localização hierárquica pode se tornar um potencial gargalo. Como esse problema pode ser efetivamente contornado?
16. Dê um exemplo de como poderia funcionar o mecanismo de fechamento para um URL.
17. Explique a diferença entre um ponteiro estreito e um ponteiro flexível em sistemas Unix. Há coisas que podem

- ser feitas com um ponteiro estrito que não podem ser feitas com um ponteiro flexível, ou vice-versa?
- Servidores de nomes de nível alto em DNS, isto é, servidores de nomes que implementam nós no espaço de nomes DNS que está próximo da raiz, em geral não suportam resolução recursiva de nomes. Poderíamos esperar grande aprimoramento de desempenho caso suportassem?
 - Explique como o DNS pode ser usado para implementar uma abordagem baseada em localização nativa para localizar hospedeiros móveis.
 - Como um ponto de montagem é consultado na maioria dos sistemas Unix?
 - Considere um sistema distribuído de arquivo que usa espaços de nomes por usuário. Em outras palavras, cada usuário tem seu próprio espaço privado de nomes. Os nomes desses espaços de nomes podem ser usados para compartilhar recursos entre dois usuários diferentes?
 - Considere o DNS. Para referenciar um nó N em um subdomínio implementado como uma zona diferente

da do domínio corrente, é preciso especificar um servidor de nomes para essa zona. É sempre necessário incluir um registro de recurso para o endereço desse servidor ou às vezes é suficiente dar somente seu nome de domínio?

- 23. Contar arquivos em comum é um modo bastante ingênuo de definir proximidade semântica. Supondo que você queira construir redes de sobreposição semântica baseadas em documentos de texto, que outra função ‘proximidade semântica’ você poderia imaginar?
- 24. (Tarefa de laboratório) Estabeleça seu próprio servidor DNS. Instale BIND em uma máquina Windows ou Unix e a configure para alguns nomes simples. Teste sua configuração usando ferramentas como o Domain Information Groper (DIG). Certifique-se de que o banco de dados DNS inclui registros para servidores de nomes, servidores de correio e servidores padronizados. Note que, se você estiver executando BIND em uma máquina cujo nome de hospedeiro seja *HOSTNAME*, poderá resolver nomes da forma *RESOURCE-NAME.HOSTNAME*.