

ACH 2147 — DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO DISTRIBUÍDOS

NOMES

Daniel Cordeiro

9 e 11 de maio de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

- Nomes, identificadores e endereços
- Resolução de nomes
- Implementação de um espaço de nomes

Essencialmente

Nomes são usados para denotar entidades em um sistema distribuído. Para realizar operações em uma entidade, é preciso ter acesso a ela usando um **ponto de acesso**. Pontos de acessos são entidades que são identificadas por um **endereço**.

Nomes “puros”

são nomes que não tem significado próprio; são apenas strings aleatórias. Nomes puros podem ser usados apenas para comparação.

Identificador

Um nome que possui as seguintes propriedades:

1. Cada identificador se refere a, no máximo, uma entidade
2. Cada entidade é referenciada por, no máximo, um identificador
3. Um identificador sempre se refere à mesma entidade (reutilização de identificadores é proibida)

Problema

Dado um nome **não estruturado** (ex: um identificador), como localizar seu **ponto de acesso**?

- Solução simples (broadcasting)
- Abordagens baseadas em um local pré-determinado (*home-based*)
- Tabelas de hash distribuídas (P2P estruturado)
- Serviço hierárquico de nomes

Broadcasting

Solução: fazer o *broadcast* do ID, requisitando que a entidade devolva seu endereço

- Não escala para além de redes locais
- Requer que todos os processos escutem e processem os pedidos de localização

Address Resolution Protocol (ARP)

Para encontrar o endereço MAC associado a um endereço IP, faz o *broadcast* da consulta “quem tem esse endereço IP?”

Ponteiros de redirecionamento

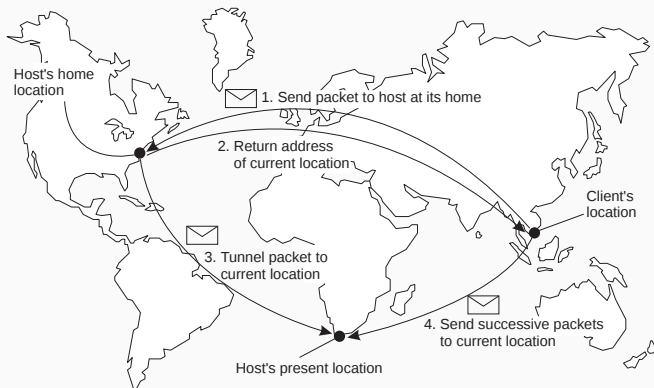
Solução: quando uma entidade se mover, ela deixa para trás um ponteiro para sua nova localização

- Dereferenciamento pode ser automático e invisível para o cliente, basta seguir a sequência de ponteiros
- Atualize a referência do ponteiro quando o local atual for encontrado
- Problemas de escalabilidade geográfica (que podem requerer um mecanismo separado para redução da sequência)
 - Sequências longas não são tolerantes à falhas
 - Maior latência de rede devido ao processo de dereferenciamento

Faça com que um local fixo (sua “casa”) sempre saiba onde a entidade está:

- O endereço pré-determinado é registrado em um serviço de nomes
- O endereço mantém um registro do **endereço externo** da entidade
- O cliente primeiro contacta o endereço pré-determinado e depois continua para seu endereço externo

Princípio do endereçamento IP móvel:



Abordagem em dois níveis

Mantenha um registro das entidades que foram “visitadas”:

- Verifique o endereço local primeiro
- Se falhar, só então vá para o endereço pré-determinado

Problemas

- O endereço pré-determinado deve existir enquanto a entidade existir
- O endereço é **fixo** e pode se tornar desnecessário se a entidade se mover permanentemente
- Escalabilidade geográfica ruim (a entidade pode estar do lado do cliente)

Pergunta: como resolver o problema da mudança permanente?
(talvez com outro nível de endereçamento (DNS))

Chord

Considere que os nós estejam organizados em forma de **anel lógico**

- A cada nó é atribuído um identificador aleatório de m -bits
- A cada entidade é atribuído uma única **chave** de m -bits
- Entidades com chave k estão sob a jurisdição do nó com o menor $id \geq k$ (seu sucessor)

Solução ruim

Faça com que cada nó mantenha o registro de seus vizinhos e faça uma busca linear ao longo do anel

Notação

Chamamos de nó p o nó cujo identificador é p .

Ideia:

- cada nó p mantém um **finger table** $FT_p[]$ com no máximo m entradas

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

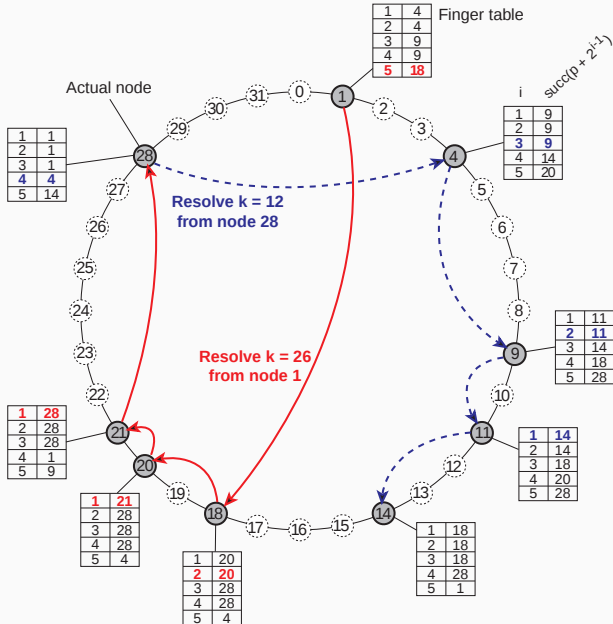
$FT_p[i]$ aponta para o primeiro nó que sucede p com distância de pelo menos 2^{i-1} posições

- Para procurar por uma chave k , o nó p encaminha o pedido para o nó com índice j tal que:

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

- Se $p < k < FT_p[1]$, a requisição é encaminhada para $FT_p[1]$

DHTS: FINGER TABLES



```

class ChordNode:
    def finger(self, i):
        succ = (self.nodeID + pow(2, i-1)) % self.MAXPROC      # succ( $p+2^{(i-1)}$ )
        lwbi = self.nodeSet.index(self.nodeID)                 # self in nodeset
        upbi = (lwbi + 1) % len(self.nodeSet)                   # next neighbor
        for k in range(len(self.nodeSet)):                      # process segments
            if self.inbetween(succ, self.nodeSet[lwbi]+1, self.nodeSet[upbi]+1):
                return self.nodeSet[upbi]                       # found successor
        (lwbi, upbi) = (upbi, (upbi+1) % len(self.nodeSet))    # next segment

    def recomputeFingerTable(self):
        self.FT[0] = self.nodeSet[self.nodeSet.index(self.nodeID)-1] # Pred.
        self.FT[1:] = [self.finger(i) for i in range(1, self.nBits+1)] # Succ.

    def localSuccNode(self, key):
        if self.inbetween(key, self.FT[0]+1, self.nodeID+1): # in (FT[0], self]
            return self.nodeID                                # responsible node
        elif self.inbetween(key, self.nodeID+1, self.FT[1]): # in (self, FT[1]]
            return self.FT[1]                                # succ. responsible
        for i in range(1, self.nBits+1):                      # rest of FT
            if self.inbetween(key, self.FT[i], self.FT[(i+1) % self.nBits]):
                return self.FT[i]                             # in [FT[i], FT[i+1]]

```

Problema:

A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $\text{succ}(k + 1)$ podem estar muito longes um do outro.

Soluções

Problema:

A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $\text{succ}(k + 1)$ podem estar muito longes um do outro.

Soluções

- **Atribuição ciente da rede:** ao atribuir um ID ao nó, assegure-se de que nós próximos no espaço de endereçamento estão próximos na rede real. **Pode ser muito complicado.**

Problema:

A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $\text{succ}(k + 1)$ podem estar muito longes um do outro.

Soluções

- **Atribuição ciente da rede:** ao atribuir um ID ao nó, assegure-se de que nós próximos no espaço de endereçamento estão próximos na rede real. **Pode ser muito complicado.**
- **Roteamento de proximidade:** mantenha mais de um sucessor possível e encaminhe a mensagem para o mais próximo. Ex: no Chord, $FT_p[i]$ aponta para o primeiro nó em $INT = [p + 2^{i-1}, p + 2^i - 1]$. O nó p pode guardar também ponteiros para outros nós em INT .

Problema:

A organização lógica dos nós em uma rede de overlay pode levar à transferência de mensagens de forma errática na Internet: os nós k e $\text{succ}(k + 1)$ podem estar muito longes um do outro.

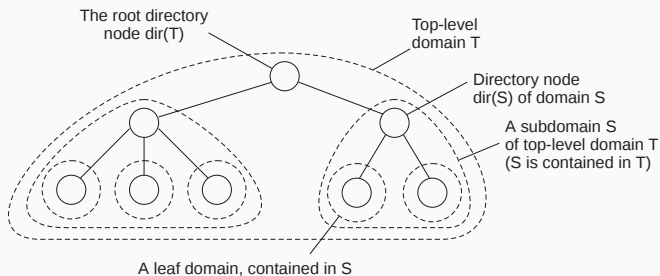
Soluções

- **Atribuição ciente da rede:** ao atribuir um ID ao nó, assegure-se de que nós próximos no espaço de endereçamento estão próximos na rede real. **Pode ser muito complicado.**
- **Roteamento de proximidade:** mantenha mais de um sucessor possível e encaminhe a mensagem para o mais próximo. Ex: no Chord, $FT_p[i]$ aponta para o primeiro nó em $INT = [p + 2^{i-1}, p + 2^i - 1]$. O nó p pode guardar também ponteiros para outros nós em INT .
- **Seleção de vizinho por proximidade:** quando houver uma escolha para determinar quem será seu vizinho, escolha o mais próximo.

SERVIÇOS DE LOCALIZAÇÃO HIERÁRQUICOS (HLS)

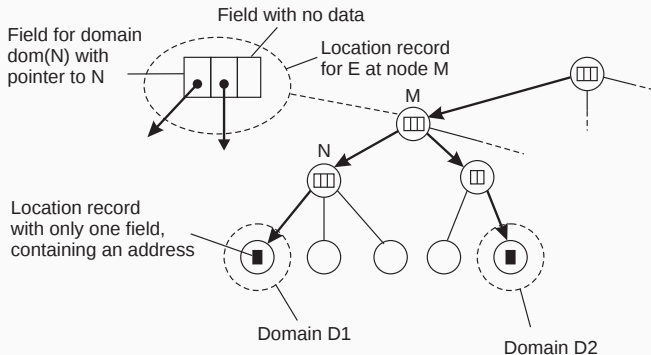
Ideia:

Construir uma árvore de busca em larga escala onde a rede é dividida em domínios hierárquicos. Cada domínio é representado por um diretório de nós separado.



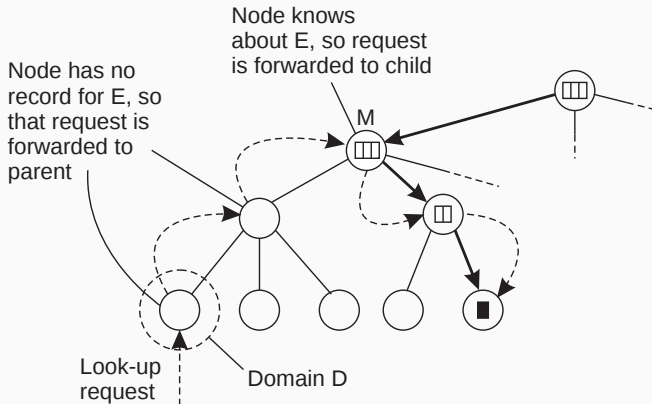
Invariantes

- Endereço da entidade E é armazenado numa folha ou nó intermediário
- Nós intermediários contêm um ponteiro para um filho sse a subárvore cuja raiz é o filho contenha o endereço da entidade
- A raiz conhece todas as entidades

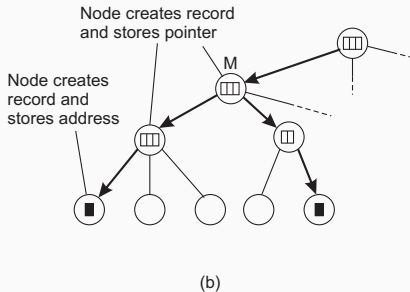
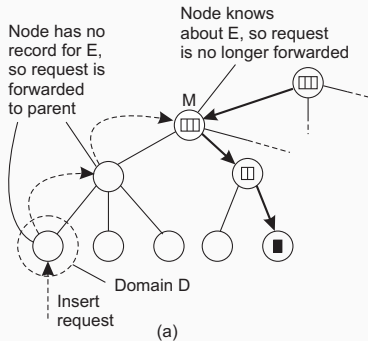


Princípios básicos:

- Comece a busca pelo nó folha local
- Se o nó souber sobre E , seguir o ponteiro pra baixo, senão continue a subir
- Continue subindo até encontrar a raiz



HLS: INSERÇÃO

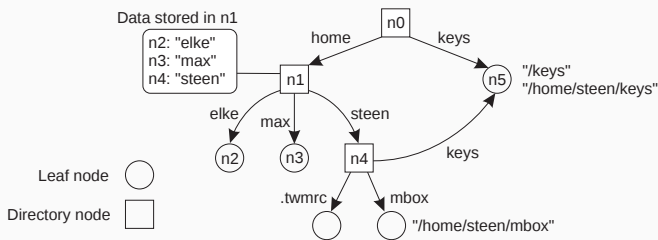


- (a) um pedido de inserção é encaminhado ao primeiro nó que conhece a entidade E
- (b) uma cadeia de ponteiros de redirecionamento é criada até o nó folha

ESPAÇOS DE NOMES ESTRUTURADOS

Ideia

Um grafo no qual um **nó folha** representa uma entidade que tem um nome. Um **diretório de nós** é uma entidade que se refere a outros nós.



Nota:

Um diretório contém uma tabela de pares (*identificador do nó*, *rótulo da aresta*)

Observação

É fácil guardar em um nó vários tipos de atributos para descrever propriedades da entidade que o nó representa:

- tipo da entidade
- identificador da entidade
- endereço da localização da entidade
- *nicknames*
- ...

Observação

É fácil guardar em um nó vários tipos de atributos para descrever propriedades da entidade que o nó representa:

- tipo da entidade
- identificador da entidade
- endereço da localização da entidade
- *nicknames*
- ...

Nota:

Nós de diretórios também podem ter atributos, além de guardar a tabela com os pares (identificador, rótulo)

Problema:

Para resolver um nome precisamos de um diretório. Como encontrar esse nó inicialmente?

Mecanismo de closure

- `each.usp.br`: inicia em um servidor DNS
- `/home/danielc/Maildir`: inicia em um servidor de arquivos NFS local (busca potencialmente recursiva)
- `00551126480149`: disca um número de telefone
- `200.144.248.41`: rota para um servidor da USP

Hard link

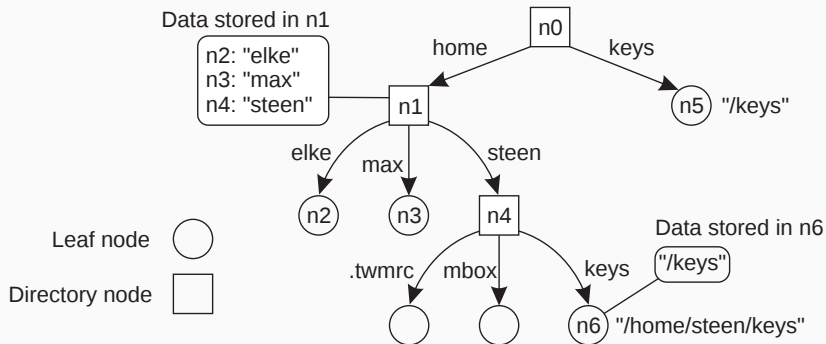
O que descrevemos até agora foi um **caminho**: um nome que é resolvido ao seguir um caminho específico em um grafo de nomes, indo de um nó para outro.

Soft link

Permite que um nó O mantenha o **nome** de outro nó:

- Primeiro resolva o nome de O (o que leva ao nó O)
- Leia o conteúdo de O , que leva ao **nome**
- A resolução do nome continua com **nome**

LINKS PARA NOMES



Observação:

O nó n5 tem apenas um nome.

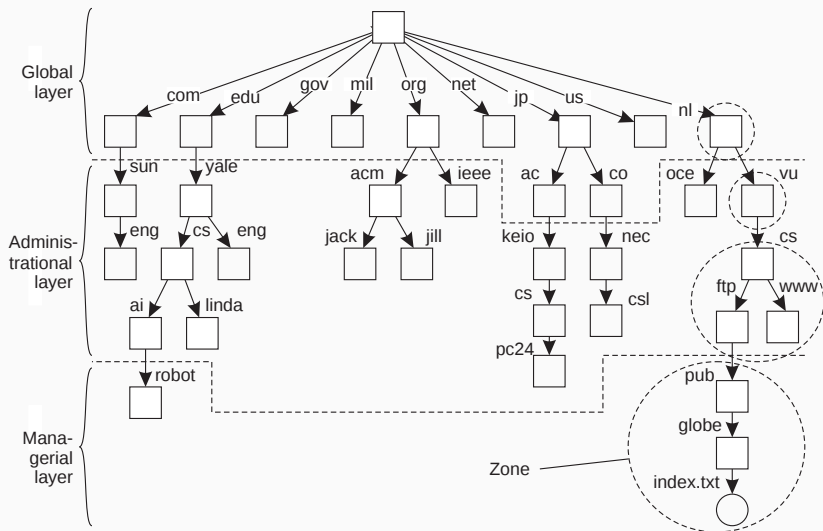
Problema:

Distribuir o processo de resolução de nomes, bem como o gerenciamento do espaço de nomes, em várias máquinas, de forma a distribuir o grafo de nomes.

Três níveis distintos:

1. **Nível global:** consiste de um diretório de nós de alto nível. Sua principal característica é que os nós do diretório devem ser gerenciado em conjunto por diferentes administradores
2. **Nível de administração:** nós de diretório de nível intermediário que podem ser agrupados de modo que cada grupo seja responsabilidade de um administrador diferente
3. **Nível gerencial:** nós de nível inferior que pertencem a um único administrador. O problema principal é mapear os nós de diretório aos servidores de nomes local.

IMPLEMENTAÇÃO DE ESPAÇOS DE NOMES

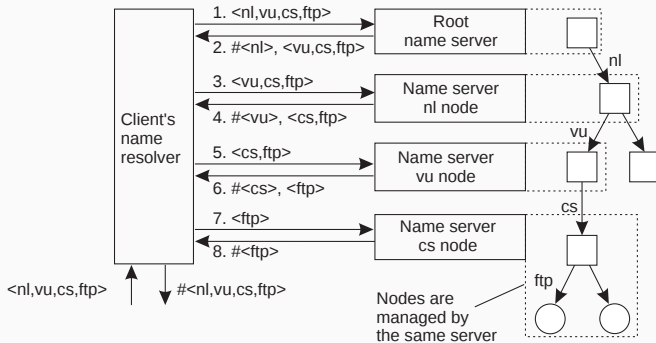


IMPLEMENTAÇÃO DE ESPAÇOS DE NOMES

Item	Global	Administração	Gerencial
1	Mundial	Organização	Departamento
2	Poucos	Muitos	Qdes. enormes
3	Segundos	Milissegundos	Imediato
4	Tardio	Imediato	Imediato
5	Muitos	Nenhum ou poucos	Nenhum
6	Sim	Sim	Às vezes
1: Escala geográfica 2: # Nós 3: Responsividade		4: Propagação de atualizações 5: # Réplicas 6: Cache no lado do cliente?	

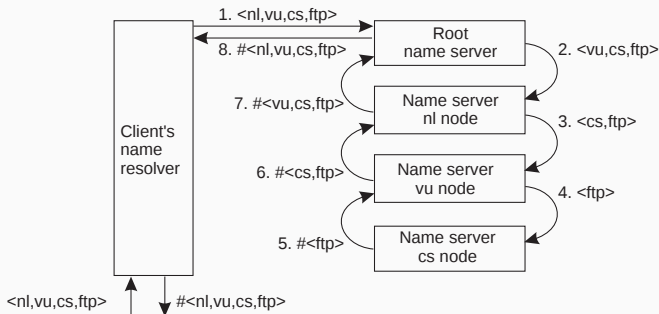
RESOLUÇÃO DE NOMES ITERATIVA

1. `resolve(dir,[name1,...,nameK])` enviado para `Server0` responsável por `dir`
2. `Server0` resolve `resolve(dir,name1) → dir1`, devolve a identificação (endereço) de `Server1`, que contém `dir1`.
3. Cliente envia `resolve(dir1,[name2,...,nameK])` para `Server1`, etc.



RESOLUÇÃO DE NOMES RECURSIVA

1. `resolve(dir,[name1,...,nameK])` enviado a `Server0`, responsável por `dir`
2. `Server0` resolve `resolve(dir,name1) → dir1` e envia `resolve(dir1,[name2,...,nameK])` para `Server1`, que contém `dir1`.
3. `Server0` espera pelo resultado de `Server1` e o devolve para o cliente.



USO DE CACHE NA RESOLUÇÃO RECURSIVA

Servidor para o nó	Deveria resolver	Procura por	Envia para filho	Recebe e and faz cache	Devolve ao solicitante
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>

Escalabilidade de tamanho

Devemos garantir que os servidores possam lidar com um grande número de requisições por unidade de tempo. Servidores de alto nível podem estar com problemas sérios.

Escalabilidade de tamanho

Devemos garantir que os servidores possam lidar com um grande número de requisições por unidade de tempo. Servidores de alto nível podem estar com problemas sérios.

Solução:

Assuma (pelo menos nos níveis global e de administração) que os conteúdos dos nós mudam muito pouco. Assim podemos aplicar replicação extensivamente, mapeando nós a múltiplos servidores e começar a resolução de nomes no servidor mais próximo.

Escalabilidade de tamanho

Devemos garantir que os servidores possam lidar com um grande número de requisições por unidade de tempo. Servidores de alto nível podem estar com problemas sérios.

Solução:

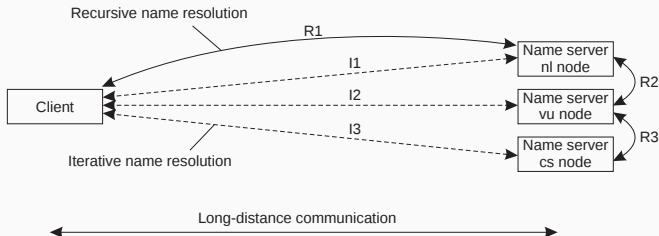
Assuma (pelo menos nos níveis global e de administração) que os conteúdos dos nós mudam muito pouco. Assim podemos aplicar replicação extensivamente, mapeando nós a múltiplos servidores e começar a resolução de nomes no servidor mais próximo.

Observação:

Um atributo importante em muitos nós é o seu **endereço** (onde a entidade pode ser localizada). Replicação faz com que servidores de nomes em larga escala sejam inadequados para localizar entidades móveis.

Escalabilidade geográfica

Precisamos garantir que a resolução de nomes escale mesmo em grandes distâncias geográficas



Porém:

Ao mapear nós em servidores que podem ser localizados em qualquer lugar, nós acabamos introduzindo uma dependência de localização implícita.

EXEMPLO: DNS DECENTRALIZADO

Ideia básica

Pegue um nome DNS completo, crie um hash com chave k e use um sistema DHT que permita consulta a chaves. **Desvantagem:** você não pode pedir por todos os nós em um subdomínio (mas pouca gente faz isso)

Informação em um nó:

SOA	Zone	Holds info on the represented zone
A	Host	IP addr. of host this node represents
MX	Domain	Mail server to handle mail for this node
SRV	Domain	Server handling a specific service
NS	Zone	Name server for the represented zone
CNAME	Node	Symbolic link
PTR	Host	Canonical name of a host
HINFO	Host	Info on this host
TXT	Any kind	Any info considered useful

Pastry

Sistema baseado em DHT que funciona com **prefixos** de chaves. Considere um sistema onde as chaves pertençam a um espaço de números de 4 dígitos. Um nó com ID 3210 mantém informações sobre os seguintes nós:

n_k	prefix of ID(n_k)	n_k	prefix of ID(n_k)
n_0	0	n_1	1
n_2	2	n_{30}	30
n_{31}	31	n_{33}	33
n_{320}	320	n_{322}	322
n_{323}	323		

Notas:

O nó 3210 é responsável pelas chaves com prefixo 321. Se ele receber uma requisição pela chave 3012, ela será reencaminhada ao nó n_{30} . Para DNS: um nó responsável pela chave k armazena os registros DNS de nomes cujo valor do hash seja k .

Definição

Replicação no nível i – registro é replicado em todos os nós com prefixo i . **Nota:** # saltos para procurar por um registro no nível i normalmente é i .

Observação:

Seja x_i a fração dos nomes DNS mais populares que deveriam ser replicados no nível i . Então:

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

onde N é o número total de nós, $d = b^{(1-\alpha)/\alpha}$ e $\alpha \approx 1$, assumindo que a “popularidade” siga a distribuição de Zipf: a frequência do n -ésimo item mais popular é proporcional a $1/n^\alpha$

Se quisermos que o número médio de consultas para a resolução de um nome DNS seja $C = 1$, então com $b = 4$, $\alpha = 0,9$, $N = 10.000$ e 1.000.000 registros

61 registros mais populares devem ser replicados no nível 0

284 próximos registros mais populares no nível 1

1323 próximos registros mais populares no nível 2

6177 próximos registros mais populares no nível 3

28826 próximos registros mais populares no nível 4

134505 próximos registros mais populares no nível 5

o resto não deve ser replicado

Observação

Em muitos casos, é mais conveniente nomear e procurar entidades pelos seus **atributos** $\Rightarrow$ serviços tradicionais de diretórios (ex: páginas amarelas).

Observação

Em muitos casos, é mais conveniente nomear e procurar entidades pelos seus **atributos** $\Rightarrow$ serviços tradicionais de diretórios (ex: páginas amarelas).

Problema:

Operações de consulta pode ser muito caras, já que necessitam que os valores dos atributos procurados correspondam aos valores reais das entidades. Em princípio, teríamos que inspecionar todas as entidades.

Observação

Em muitos casos, é mais conveniente nomear e procurar entidades pelos seus **atributos** $\Rightarrow$ serviços tradicionais de diretórios (ex: páginas amarelas).

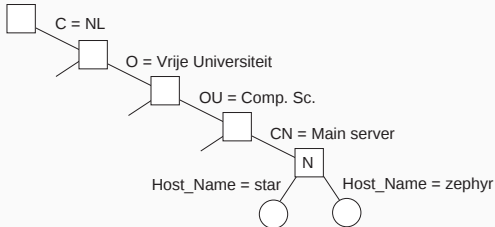
Problema:

Operações de consulta pode ser muito caras, já que necessitam que os valores dos atributos procurados correspondam aos valores reais das entidades. Em princípio, teríamos que inspecionar todas as entidades.

Solução:

Implementar serviços de diretórios básicos (tais como bancos de dados) e combiná-los com os sistemas de nomes estruturados tradicionais.

EXEMPLO: LDAP



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

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

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