Estudo dirigido sobre um serviço de nomeação baseado em tabela hash distribuída (Protocolo Chord)

1. Quais são as motivações principais para o protocolo proposto?

A criação de um protocolo Peer-to-Peer (P2P) que seja eficiente na busca por um nó alvo. O protocolo Chord provê apenas uma operação de busca, dada uma chave ele retorna o nó em questão que esteja armazenando algum dado ou serviço. O Chord ultrapassa as principais barreiras de um protocolo P2P:

- o protocolo é escalável, isto é, o custo da comunicação e manutenção da rede cresce de forma logarítmica com o número de nós.
- é balanceado, ou seja, cada nó guarda informações sobre apenas uma parcela de nós na rede, não sobrecarregando-a.
- é descentralizado, uma vez que todos os nós estão em um mesmo nível de importância e hierarquia.
- alta viabilidade, pois não possui nenhuma necessidade de manutenção, uma vez que suas tabelas de busca são atualizadas automaticamente com a entrada e saída de um nó.

2. Em quais contextos o protocolo pode ser usado?

São apresentados pelo autor algumas aplicações que nas quais o protocolo provê um bom funcionamento: *Cooperative mirroring*, *Time-shared storage, Distributed indexes e Large-scale combinatorial search.*

Cooperative mirroring consiste em múltiplos provedores de conteúdos guardar e prover tais conteúdos uns para os outros, de modo que a carga fica distribuída entre os nós (foco em **balanceamento**), não havendo picos de uso isolados.

Time-shared storage consiste em manter disponíveis um conjunto de dados utilizando (foco em **disponibilidade/viabilidade**) nós com conexão intermitente. Cada nó é responsável por guardar uma parcela de um conjunto de dados importantes para uma determinada rede, mas não necessariamente para esse nó. A ideia é que, caso um nó necessite utilizar determinado serviço, mas seu próprio servidor não esteja disponível, um outro nó estará hospedando-o.

Distributed indexes é usado para suportar buscas do tipo *Gnutella ou Napster* (que buscam dados por palavras-chave em uma rede distribuída).

Large-scale combinatorial search pode ser usada para quebra de código, onde as chaves do protocolo Chord são soluções candidatas para o problema

e utilizando Chord é possível mapear quais máquinas são responsáveis para testar tais soluções.

3. Como funciona a operação de busca (lookup)?

São apresentados dois algoritmos de lookup: *Simple e Scalable*. Primeiramente, é importante frisar que cada nó guarda uma chave, de modo que a chave K é guardada por um nó cujo rótulo N seja o menor possível acima ou igual a K.

O algoritmo *Simple* consiste em cada nó guardar apenas a informação de quem é seu sucessor direto da "direita" (assumindo uma topologia lógica circular crescente com orientação no sentido horário*)* e uma busca nesse modelo seria feita passando a key buscada por todos os nós até encontrar o nó alvo. A busca se inicia no primeiro nó (menor rótulo N da topologia) e vai de sucessor em sucessor até encontrar um nó N cujo valor seja acima ou igual a K (este nó, portanto, armazenará a chave K), retornando o valor.

O algoritmo *Scalable* é menos ingênuo e exige que cada nó guarde mais informações além de unicamente quem é seu sucessor, a fim de reduzir os custos da busca. Considere, inicialmente, que a topologia é formada por nós com rótulos de *m* bits, portanto, variando em um intervalo de [0, $2^m - 1$]. Dessa forma, cada nó manterá a chamada *finger table*, que é uma tabela que possui *m* entradas. A i-ésima entrada do nó N é preenchida por:

- $K = N + 2^{i-1}$, caso K seja um rótulo existente na topologia
- tratando $K = N + 2^{i-1}$ como uma chave, isto é, encontrando-se o nó que armazenaria uma chave com valor K

Seja *N* o nó realizando a busca, *K* o nó buscado, N.finger a *finger table* de *N* e *N.finger[i]* a *hashkey* da *i*-ésima entrada na N.finger. A busca realizada pelo nó N por um nó K começa verificando se o *id* de K está entre o nó *N* e seu sucessor imediato, se sim é retornado seu sucessor (literalmente *N.finger[1]*). Senão, é feita uma busca na tabela começando pela última entrada até seu o início (loop de m até 2). Quando achar o nó *N'* com *id'* que imediatamente precede o *id* buscado é então chamado de forma recursiva essa busca na tabela para o nó *N'.*

4. Como funcionam as operações de inserção e remoção de nós?

Inserção:

Um novo nó N ao entrar na rede Chord deve utilizar a função de *create* para criar uma nova rede Chord ou a função de *join* em algum nó K conhecido para que K possa procurar o sucessor de N. Uma vez que N entrou na topologia, seu sucessor S foi identificado e N se identificou como o

predecessor de S. Porém, o antigo predecessor de S ainda possui S como seu sucessor. Sendo assim, cada nó T da rede executa periodicamente a função de *stabilize* para conhecer novos nós, perguntando (ao seu sucessor) se o predecessor de seu sucessor é ele mesmo (T). Caso não seja, significa que um novo nó N entrou ali no meio, portanto, ele será seu novo sucessor. Periodicamente cada nó T da rede executa *fix_fingers* para garantir que suas *finger tables* estão corretas, dessa forma os novos nós podem preencher suas tabelas e "velhos" nós podem atualizar as suas. Outras funções que também são executadas periodicamente são *check_predecessor e notify*; a primeira verifica se o predecessor de um nó T qualquer falhou/saiu da rede atribuindo a essa variável nulo e a segunda atualiza o predecessor de T com seu novo predecessor.

Remoção/falha:

Basicamente cada nó N da rede deve sempre conhecer seu sucessor S para que a operação de busca funcione de forma correta. Se S falha/sai da rede, N deve ser capaz de determinar seu novo sucessor S'; para isso N mantém uma lista com os seus r próximos sucessores. Sendo p a probabilidade de um nó sair/falhar, a probabilidade de todos os sucessores de N caírem ao mesmo tempo é p^r , ou seja, decai exponencialmente com o tamanho de r. Uma remoção voluntária de um nó pode simplesmente ser tratada como uma falha, porém, uma abordagem mais direta pode ser utilizada para otimizar o Chord. Primeiramente, o nó que saiu torna transfere suas chaves para seu sucessor. Em seguida, notifica seu sucessor de quem é seu novo predecessor e vice-versa.

5. Aponte uma vantagem do trabalho em comparação com trabalhos relacionados.

Além das vantagens, inerentes à implementação do Chord, apresentadas na questão 1 (escalabilidade, balanceamento, descentralização e viabilidade), uma possível vantagem do Chord em relação ao protocolo DNS é que não é necessário ter um servidor que resolve nomes em endereços IP, tampouco manutenção, uma vez que as redes Chord são atualizadas automaticamente pelo próprio funcionamento do algoritmo.

6. Aponte uma crítica ao trabalho que merece discussão.

O autor destaca que muitas premissas assumidas e conclusões chegadas ao longo do algoritmo levam em consideração que o anel de Chord está estável. Porém, também afirma que é muito improvável, em um cenário real, que a estabilidade seja atingida por conta da perturbação sofrida com a entrada e saída de nós.