Relatório T2 - Sistemas Avançados de Banco de Dados

Guilherme Carbonari Boneti
PPGInf
UFPR
guilherme.boneti@ufpr.br

I. Objetivo

Este relatório descreve a implementação do trabalho prático **T2 - Distributed Hash Tables** da disciplina de Desempenho de SGBD. O objetivo deste trabalho é desenvolver uma Tabela Hash Distribuída (DHT) com capacidade de roteamento e armazenamento de chaves em um ambiente distribuído. A DHT implementada é uma versão centralizada simplificada baseada na estrutura da DHT Chord.

II. O Problema

O problema proposto envolve o gerenciamento dinâmico de nós em um anel distribuído, onde cada nó se conecta aos demais através de uma *Finger Table*. A implementação deve permitir a inclusão e exclusão de nós, além de suportar a inserção de chaves na DHT. A inserção deve ser iniciada por um nó arbitrário, e o algoritmo é responsável por localizar o nó adequado para armazenar cada chave em sua *Hash Table*. As chaves são armazenadas no nó de menor valor que ainda seja maior que a chave.

A *Finger Table* de cada nó tem tamanho $\log_2$ do maior identificador da rede, com entradas indexadas por 2^m. Cada entrada aponta para nós específicos no anel, segundo a fórmula $(N + 2^{(k-1)}) \mod 2^m$.

III. Solução

A solução foi desenvolvida em C, organizada em três arquivos principais, com um *Makefile* que facilita a compilação. As funções implementadas seguem as especificações, garantindo a operação correta dos algoritmos de inclusão e exclusão de nós, *lookup* e inserção de chaves.

- **hash.[c,h]**: Contém a implementação da tabela hash e as funções principais para manipulação de nós, *lookup* e inserção de chaves.
- **utils.[c,h]**: Inclui funções auxiliares para leitura de entradas e formatação da saída.
- **myth.c**: Arquivo principal, que utiliza as funções definidas para executar as operações conforme o caso de teste fornecido.

O *Makefile* foi configurado para compilar todos os arquivos e gerar o executável **mydht**, conforme especificado no enunciado. A compilação é realizada com o comando `make`, sem a necessidade de parâmetros adicionais.

IV. Implementação

A. Inicialização das Estruturas

- `inicializaFingerTable` aloca a *Finger Table* com tamanho máximo (`MAX_SIZE`), definindo todas as posições inicialmente como vazias e atribuindo índices usando potências de 2.
- `inicializaHashTable` cria uma tabela hash com tamanho `HASH_SIZE`, inicializando todas as posições como vazias.
- `inicializaNodes` cria um vetor de nós e cada nó é inicializado com sua própria tabela hash.
- `inicializaRing` aloca o anel, associando-o ao vetor de nós inicializado.

B. Gerenciamento de Nós no Anel

- `entrada` insere um novo nó no anel de forma ordenada e redistribui as chaves entre os nós vizinhos, chamando a função `updateKeys`.
- `saida` remove um nó do anel e transfere suas chaves para o nó sucessor, utilizando a função `transferKeys`.

C. Lookup

A função `lookup` realiza a busca de uma chave no anel utilizando a *Finger Table* do nó:

- Usando a *Finger Table* do nó, podemos encontrar o nó mais próximo do valor da chave.
- O *lookup* é então repassado para esse nó, que verifica se ele armazena a chave requisitada.
- Esse método permite dar saltos eficientes no anel e chegar ao nó correto rapidamente.

1) Detecção de Ciclos: Durante o lookup, é necessário garantir que o processo de roteamento não entre em um ciclo. A detecção de ciclos é implementada da seguinte maneira:

- O vetor `lookup_nodes` é usado para armazenar o caminho dos nós acessados durante a operação de busca.
- Para cada nó acessado, verifica-se se ele já está presente no vetor. Caso esteja, um ciclo foi detectado, e o processo de busca é imediatamente interrompido.

D. Distribuição de Chaves

Para cada operação de inclusão e exclusão de nós, as chaves precisam ser redistribuídas:

- `inclusao` insere uma nova chave em um nó no anel, atribuindo-a ao primeiro nó que tem um identificador maior ou igual ao valor da chave.
- `transferKeys` transfere as chaves de um nó removido para seu nó sucessor, garantindo que os dados permaneçam acessíveis.
- `updateKeys` redistribui as chaves de um nó sucessor ao incluir um novo nó no anel, deslocando chaves conforme necessário para balancear a carga entre os nós.

V. Resultados

Os testes executados com inserções e remoções de chaves demonstraram que o algoritmo da DHT é eficaz no gerenciamento de armazenamento distribuído de chaves. A saída inclui os resultados das operações de *lookup* e as *Finger Tables* dos nós que participaram das operações.

VI. Conclusão

A estrutura implementada permite o funcionamento adequado de uma Tabela Hash Distribuída. O código foi devidamente comentado para facilitar a compreensão e a análise, e as funcionalidades foram organizadas em arquivos separados, promovendo a modularidade do sistema.