Laboratórios de Informática III

Diogo Gonçalves(a81860), Pedro Gomes(a82418), Pedro Lima(a80785)

May 6, 2018

Abstract

No âmbito da unidade curricular de LI3, foi proposto um projeto que envolve conceber uma estrutura de dados capaz de suportar todos os metadados do *Stack Overflow* e executar 11 *querys* sobre os mesmos, da forma mais eficiente possível. Ao longo deste relatório, serão abordadas as ideias fundamentais na definição da estrutura, os dificuldades encontradas e as soluções implementadas.

1 Introdução

O *Stack Overflow* é uma das maiores plataformas de suporte à comunidade informática, onde qualquer utilizador pode ver as questões colocadas e respondidas anteriormente ou colocar ele próprio a sua questão, de modo a obter respostas objetivas ao problema apresentado.

A ideia subjacente ao projeto de Laboratórios de Informática III (LI3) proposto é criar uma estrutura de dados capaz de não só suportar os metadados desta plataforma, como efetuar 11 *querys* diferentes sobre os mesmos, tendo estes de ser acedidos e retornados o mais eficientemente possível, de modo a reduzir o tempo de execução.

Neste sentido, o trabalhado realizado foi implementado na linguagem C que, sendo uma linguagem de baixo nível, permite um maior controlo de todo o programa e um maior grau de eficiência.

Este relatório está organizado da seguinte forma: na Secção 2 será descrita a estrutura de dados principal (TAD_comunity); na Secção 3 serão explicadas as *querys* individualmente; na Secção 4 serão explicadas as decisões tomadas no trabalho numa perspetiva de otimização do desempenho do programa; na Secção 5 serão expostas as conclusões do trabalho realizado.

2 Estrutura de dados principal

Neste projeto foram elaboradas múltiplas estruturas, de modo a guardar da forma mais acessível a informação. No entanto, existe uma estrutura principal, a TAD_comunity, que agrupa as sub estruturas principais: três tabelas de *hash*; um *array*; uma lista ligada de *arrays* de listas ligadas. A Figura 1 ilustra a estrutura principal definida e, a título de exemplo, a estrutura de suporte aos *posts* por dia e a estrutura que armazena informação por ano.

Em mais detalhe:

- as tabelas de *hash* são usadas para: (i) os *posts*, onde o *id* do *post* é a *key*, que permite encontrar um POST (estrutura que guarda toda a informação necessária relativa a um *post*); (ii) os *users*, onde o *id* do *user* é a *key*, que permite encontrar um USERS (estrutura que guarda toda a informação relativa a um *user*); (iii) as *tags*, onde o nome da *tag* é a *key*, que permite encontrar uma TAG (estrutura que guarda toda a informação relativa a uma *tag*);
- o *array* suporta os *users* organizados pelo seu número de *posts* publicados;
- a lista ligada é uma forma muito eficiente de guardar os *posts* por tempo, onde, sabendo uma data, se encontram todos os *posts* desse dia guardados cronologicamente até ao milisegundo.

As *hash* foram implementadas usando o código do *glib*, sendo assim extremamente eficientes na procura de um elemento. A lista ligada foi implementada manualmente, sendo também muito eficiente uma vez que a componente de lista ligada é relativa aos anos, logo nunca são muitos elementos, sendo o dia atingido através do *array* (com uma funçao de *hash* de $\theta(1)$).

Na Secção 4 debatem-se as principais questões de desempenho que levaram à escolha destas estruturas de dados. Como posteriormente mencionado, destaca-se o elevado desempenho de resposta às distintas *querys*, na ordem dos milisegundos.

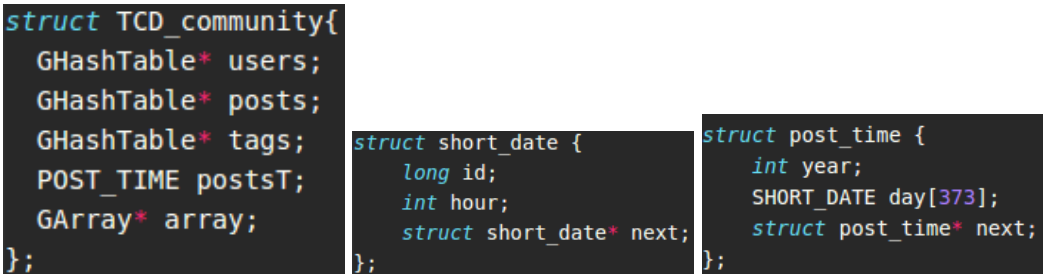

Figure 1: Estrutura principal; *Posts* do dia; Estrutura para cada ano

3 Querys

Nesta secção serão explicadas todas as *querys* individualmente, discutindo-se a estrutura implementada e o seu grau de eficiência.

3.1 Query 1

Nesta *query*, de modo a obter o *output* desejado, o título do *post* e o *username* do criador, faz-se um *lookup* na *hash* com o *id* do *post*, obtendo-se assim toda a informação deste. Se for uma resposta, repete-se o processo para o *id* da pergunta original. Caso seja a pergunta do *lookup*, obtém-se diretamente o título e, seguidamente, faz-se um *lookup* na *hash* de *users* com o *id* do criador da pergunta, obtendo-se assim o seu *username*.

3.2 Query 2

Nesta *query*, de modo a obter o *output* desejado, o top N de utilizadores com maior número de *posts*, ao fazer o *load* dos *posts* para a sua estrutura, faz-se de imediato a leitura do *user* que o criou e incrementa-se um na sua variável que conta o número de *posts*. Na ativação da *query* faz-se uma passagem desses valores para um *array*, faz-se *sort* ao *array* e retorna-se os N primeiros elementos.

3.3 Query 3

Nesta *query*, de modo a obter o *output* desejado, o número de perguntas e respostas num certo intervalo de tempo, percorre-se a estrutura dos *posts* organizada pelo tempo. Inicializa-se na data final e percorre-se até a data inicial, uma vez que o *output* é necessário ser em cronologia inversa, e verifica-se se o *post* é uma pergunta ou resposta, incrementando a varíavel do seu tipo. No fim, retorna-se os dois valores incrementados ao longo da fusão.

3.4 Query 4

Nesta *query*, de modo a obter o *output* desejado, todas as perguntas que contêm uma determinada *tag* num certo intervalo de tempo, percorre-se a estrutura dos *posts* organizada por tempo e, através do *id*, verifica-se na estrutura principal dos *posts* se esse contem a *tag*, guardando as que verificam tal condição.

3.5 Query 5

Nesta *query*, de modo a obter o *output* desejado, a bio de um utilizador e os seus últimos 10 *posts*, percorre-se a estrutura dos *posts* organizada por tempo e através do *id* verifica-se na estrutura principal dos *posts* se o criador é o *id* do input. Como a estrutura é organizada com a cronologia inversa, as 10 primeiras ocorrências são as desejadas.

3.6 Query 6

Nesta *query*, de modo a obter o *output* desejado, as N respostas com mais votos num determinado intervalo de tempo, percorre-se a estrutura dos *posts* organizada por tempo, e sempre que se

encontra uma resposta, insere-se numa lista ligada com o *id* e o *score*. Inserindo as respostas que se encontram de forma ordenada ao longo do intervalo de tempo, no fim do intervalo de tempo, retornam-se os N primeiros.

3.7 Query 7

Nesta *query*, de modo a obter o *output* desejado, as N perguntas com mais respostas num determinado intervalo de tempo, percorre-se a estrutura dos *posts* organizada por tempo, e sempre que se encontra uma resposta é adicionada uma entrada numa *hash table*, ou caso já exista a entrada, é incrementado um contador dentro da mesma. De seguida, as entradas são organizadas por ordem descrescente em função do número de respostas.

3.8 Query 8

Nesta *query*, de modo a obter o *output* desejado, todas as perguntas que contêm uma determinada palavra no título, percorre-se a estrutura dos *posts* organizada por tempo e, através do *id*, verifica-se na estrutura principal se tem título. Caso tenha, testa-se se a palavra do input faz parte, guardando as que verificam tal condição.

3.9 Query 9

Nesta *query*, de modo a obter o output desejado, as últimas N perguntas que dois utilizadores interagiram, percorre-se a estrutura dos *posts* organizada por tempo e sempre que se encontra uma interação por parte de um utilizador, guarda-se numa hash table local uma identificação desse post. Posteriormente se houver uma interação por parte do outro utilizador nesse mesmo post, é completada a parcela na *hash table* e adiciona-se o *id* dessa pergunta à lista de retorno.

3.10 Query 10

Nesta *query*, de modo a obter o output desejado, a melhor resposta para uma certa pergunta, logo que se faz o load do ficheiro, sempre que é encontrada um post resposta, é calculado o seu score e guarda-se no psot pergunta. Ao longo das N respostas que se encontram, vai se comparando se é melhor que a que já estava guardade e se tal condição se verificar, atualiza-se a melhor resposta.

3.11 Query 11

Nesta *query*, de modo a obter o output desejado, as N tags mais utilizadas pelos N utilizadores com mais reputação, dentro de um certo intervalo de tempo, utilizamos a estrutura que organiza os *posts* por tempo. Percorremos essa estrutura uma vez, para encontrar os N utilizadores com mais reputação nesse intervalo. De seguida percorremos novamente, procurando as perguntas feitas por esses utilizadores, de modo a encontrar as tags que utilizaram.

As tags que vão sendo encontradas são guardadas numa hash, para um acesso mais rápido, uma vez que será necessário incrementar o seu número de aparições. Por fim percorre-se essa hash e organiza-se as N primeiras tags com maior número de utilização.

4 Análise de desempenho

Na selecção das estruturas de dados a usar neste trabalho, tivemos várias preocupações, desde a facilidade/eficiência de acesso à informação, como o número de bytes que iríamos precisar de alocar. Tendo em conta a dimensão de dados que estas têm de guardar, estes aspetos são fundamentais, daí termos tomado as seguintes decisões:

1. Uso de tabelas de *hash* para guardar a informação proveniente do *parser* dos ficheiros. Uma vez que a busca de informação por *id* é algo muito presente no trabalho, esta estrutura permite-nos um acesso muito rápido usando esse parâmetro.

2. Criação de uma estrutura nossa, que organiza os *posts* por tempo, tendo em conta a contínua necessidade de organizar items cronologicamente. Esta estrutura permite-nos percorrer apenas os *posts* necessários de um intervalo de tempo arbitrário, reduzindo significativamente o

tempo de execução das *querys*. De modo a reduzir a repetição de dados (guardados nesta estrutura e na *hash*), guardou-se apenas o *id* do *post* (através do qual se usa a função de *hash* para descobrir a informação completa desse *post*) e a data deste, para se poder organizar os *posts* até ao milisegundo.

Para reduzir ainda mais o despedício de *bytes*, a data é guardada em formato int, representando as horas, minutos e segundos (passando de um total de 23 bytes para 4). O acesso ao ano, é o nodo da maior lista ligada e o mêsdia são o indice do *array*. Esse *array* foi criado com 373 casas, em vez de 365, pois era um acréscimo de apenas 8 casas (quantia irrisória) e permitia um acesso direto à casa através de uma operação aritmética *(mês - 1) * 12 + dia* em vez do uso de uma função para tal.

3. Criação de um *array* com os *id's* dos *users* ordenados pelo maior número de *posts*. Apesar desta estrutura ser necessária apenas para a *query* 2, consideramos mais elegante criá-la no parser, onde o tempo que acresce ao *load* é demasiado pequeno para fazer diferença e assim a execução da *query* torna-se bastante mais rápida. Numa perspetiva do programa, onde as *querys* vão ser executadas mais que uma vez, faz mais sentido por isso no *load* e não na *query*, sendo assim criado apenas uma vez e lido as vezes que a *query* for executada.

Após todos estas decisões, o trabalho ficou com um desempenho que consideramos formidável, sendo os tempos de execução mínimos, na ordem dos milisegundos. Quanto ao espaço alocado, reduzimos também ao que consideramos mínimo.

5 Conclusões

Tendo em perspetiva as decisões tomadas para a concretização do trabalho, realçamos os pontos positivos da estrutura implementada que se refletiram num elevado desempenho de resposta às distintas *querys*.

Durante todo o trabalho tivemos a preocupação de alocar o mínimo espaço possível, evitar desperdícios ou repetições de informações, tendo em memória apenas o vital para uma execução plena e eficiente do programa, tendo inúmeras vezes optado por soluções um pouco mais complexas que permitiam reduzir a quantidade de dados. Note-se, no entanto, que esta complexidade não se traduziu em perda de desempenho, pelo contrário, o mesmo foi cuidadosamente pensado na definição das estruturas de dados, tendo-se obtido um desempenho de resposta da ordem dos milisegundos. Quanto à precisão no aspeto tempo, os *posts* estão organizados até ao milisegundo, sendo assim muito precisos. Na *query* 7 verificamos também que as respostas faziam parte do intervalo, para um resultado mais relevante.

Todo o código desenvolvido neste projeto foi devidamente encapsulado, por uma questão de segurança do código, para que nas inúmero obtenções de informação não houvesse qualquer risco de danificar a informação guardada.

Por fim, achamos necessário realçar o impacto que este trabalho teve nos nossos conhecimentos de estrutura de dados, devido à dimensão dos ficheiros envolvidos, criando uma perspetiva mais realista do funcionamento destas e aprimorando os conceitos previamento aprendidos.