Sistemas Distribuídos

Projecto:

Sistema de Mercado Distribuído para Hash Matching Colaborativo

Rui S. Moreira & Christophe Soares

UFP - FCT

Março 2020

1. Sistema de Mercado Distribuído para Hash Matching Colaborativo

1.1. Introdução

Este projecto tem dois objectivos principais: i) enriquecer os conhecimentos e a familiaridade dos alunos em relação aos vários aspectos e requisitos vulgarmente associados a projectos de sistemas distribuídos; ii) melhorar o entendimento e a prática dos alunos em relação à especificação e desenvolvimento de sistemas distribuídos.

Neste projecto irá utilizar-se o conceito de *hashing*, no qual uma função de *hash* recebe tipicamente uma sequência de caracteres de comprimento variável e implementa um algoritmo que mapeia essa sequência de entrada num código (*digest*) de comprimento fixo. Os códigos de saída da função designam-se tipicamente por *hash codes* ou simplesmente *hashes*. NB: no projecto proposto iremos utilizar apenas inputs de comprimento fixo e previamente conhecido.

Figure 1: algoritmo ou função de *hashing*.

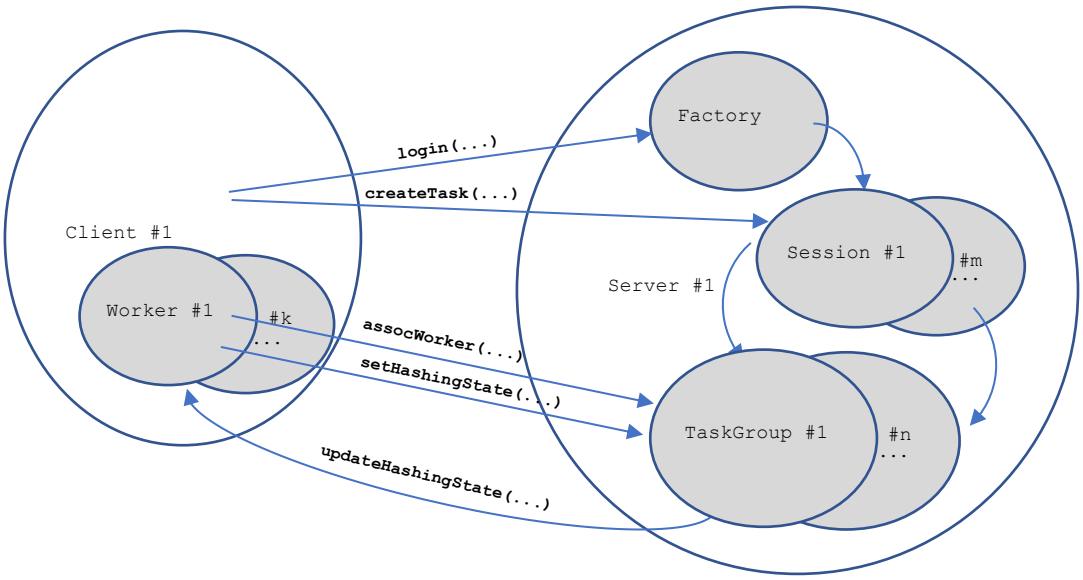
Os algoritmos de *hashing* têm várias aplicações como, por exemplo, a verificação de integridade de software descarregado da Internet e a codificação de *passwords* em sistemas de autenticação. Para que estes algoritmos sejam eficazes devem possuir algumas propriedades, nomeadamente:

- Serem determinísticos: a mesma mensagem processada pela mesma função *hash* deve produzir sempre o mesmo código *hash*;
- Serem não reversíveis: ser computacionalmente difícil/impraticável gerar uma mensagem a partir de seu *hash*;

- Possuírem entropia elevada: qualquer pequena alteração numa mensagem deve produzir um *hash* muito diferente;
- Serem resistentes a colisões: duas mensagens diferentes não devem produzir o mesmo *hash*.

1.2. Descrição e Objectivos do Projecto

Neste projecto propõe-se o desenvolvimento de um sistema de mercado distribuído que permita a vários clientes/utilizadores registarem-se no serviço. Posteriormente os utilizadores podem iniciar sessões de trabalho (*session*), através das quais podem listar e criar grupos de tarefas partilhadas (*task groups*). Cada *task group* serve para que o seu criador/dono submeta uma tarefa de *hash matching*. Para o fazer deve carregar no sistema um conjunto de créditos suficiente para a execução da tarefa. Os outros utilizadores podem associar-se a uma tarefa de um *task group*, participando na sua execução, através de um ou mais *workers*, obtendo com isso créditos. Os *worker* ganham créditos associados à mineração de cada sequência de caracteres do *task group* (i.e., à semelhança do que acontece na mineração de um *blockchain*).



Cada *task group* permitirá então dividir por vários trabalhadores (*workers*), de forma colaborativa, a tarefa de testar a correspondência (*match*) entre dois conjuntos de dados:

i) um conjunto exaustivo de várias sequências de caracteres (combinação de letras e dígitos) com um determinado tamanho fixo. Este conjunto de sequências de caracteres pode ser enumerado explicitamente ou, em alternativa, indicado através de uma gama de valores entre os quais se devem considerar todas as combinações possíveis;

ii) um conjunto pequeno de códigos de *hash* pré-definidos, ou seja, uma lista de *hash codes* com os quais se pretende encontrar possíveis correspondências (*match*) com as sequências de caracteres do primeiro conjunto.

Em cada *task group* pretende-se aplicar exaustivamente um algoritmo de *hashing* a cada sequência de caracteres do 1º conjunto, produzindo assim os respectivos *hash codes*. Para cada um dos códigos *hash* calculados, pretende-se verificar a correspondência (*match*) com os *hash codes* enumerados no 2º conjunto. Para isso, cada *worker* deverá aplicar exaustivamente um algoritmo de *hashing* pré-seleccionado na criação do *task group* (e.g. SHA-512, PBKDF2, BCrypt, SCrypt, ...) a cada sequência de caracteres do 1ª conjunto, e verificar se existe *matching* com algum dos códigos de *hash* presentes no 2ª conjunto.

Dado que o 1º conjunto de caracteres pode ser demasiado grande, para ser analisado num único computador, pretende-se que o *task group* divida o conjunto de sequências de caracteres em vários sub-conjuntos mais pequenos. Posteriormente, deverá entregar a cada *worker*, associado ao *task group*, um dos sub-conjuntos, juntamente com o 2º conjunto de códigos *hash*. Cada *worker* receberá, portanto, um sub-conjunto de sequências de caracteres às quais aplicará o algoritmo de *hashing* pré-seleccionado; e comparará o código de *hash* resultante com o conjunto de códigos de *hash* a descobrir; o *worker* ganhará créditos por cada sub-conjunto analisado e também por cada correspondência (*match*) de *hash code*.

Aquando da criação do *task group*, o utilizador deverá carregar os créditos necessários à mineração da tarefa escolhida (parcial ou total) e especificar um conjunto de informação, que pode ser passada através de parâmetros na consola, através de um ficheiro ou de uma interface gráfica simples:

i) o algoritmo de *hashing* a utilizar;

ii) as sequências de caracteres a analisar, de acordo com a respectiva estratégia de definição/divisão dos sub-conjuntos (e.g., um ficheiro global, uma gama de valores, etc.), conforme descrito abaixo;

iii) o tamanho de cada sub-conjunto de sequências de caracteres a extrair do conjunto global anterior;

iv) o conjunto de *hash codes* para o qual se pretende identificar possíveis *matches* com os *hash codes* extraídos de cada sub-conjunto.

1.3. Descrição dos Requisitos Concretos do Projecto

Dada a descrição genérica do problema de correspondência entre um conjunto vasto e exaustivo de sequências de caracteres e um conjunto menor de *hashes* a encontrar, pretende-se de seguida detalhar mais pormenorizadamente os requisitos concretos deste projecto:

R1. O sistema deverá aceitar o registo de vários utilizadores/clientes, que devem indicar o *username* e a *password*. Posteriormente, os clientes podem efectuar *login* para obterem uma sessão de trabalho, na qual poderão efectuar operações básicas de gestão de *task groups* (e.g. *list task groups*, *create task group, pause task group, delete task group*, etc.).

R2. Cada *task group* corresponderá a uma "reunião/sala" à qual vários *workers* (de diferentes utilizadores) podem ser associados, para colaborar na tarefa de *matching* de códigos *hash* definida para o mesmo grupo. Cada utilizador poderá associar um ou mais *workers* ao mesmo *task group*. A associação de um *worker* permite que lhe seja atribuído sub-conjunto de sequências de caracteres para analisar. Os sub-conjuntos distribuídos pelos *workers* deverão ser exclusivos, para evitar *overlaping* e duplicação de esforços.

R3. A distribuição de tarefas pelos vários *workers* deverão seguir algumas estratégias, em particular na definição dos sub-conjuntos a serem analisados.

- Estratégia 1: cada *task group* deverá usar um ficheiro[1] com todas as sequências de caracteres comumente utilizadas; neste caso o sistema deverá definir para cada *worker* a *linha x* correspondente ao início de cada sub-conjunto e o *delta n* de linhas a analisar (e.g., *subset 1: line 1000, delta 5000*);
- Estratégia 2: como a primeira solução pode não gerar uma distribuição homogênea pelos vários *workers*, uma vez que nem todas as linhas podem corresponder aos critérios de pesquisa (e.g. tamanho das sequências de caracteres) deverão explorar-se heurísticas que tornem mais equitativa a divisão dos sub-conjuntos, e.g. cada *worker* poderá identificar e fazer *bidding* por um sub-conjunto de linhas no ficheiro que correspondam aos critérios de selecção definidos; o bidding deverá ser coordenado para evitar sobreposições entre sub-conjuntos;
- Estratégia 3: para cada *task group*, o sistema deverá gerar *ranges* para as combinações de sequências de caracteres de cada sub-conjunto. A especificação dos *ranges* deverá incluir o alfabeto (conjunto de caracteres a combinar) e o tamanho das sequências de caracteres a gerar. Cada *worker* receberá um *range* para análise e deverá gerar todas as combinações de sequências de caracteres possíveis desse *range*, para analisar.

Estas estratégias deverão ser implementadas de forma evolutiva, ou seja, começando por implementar a primeira e só depois passar a estratégias mais complexas/evoluídas. Em qualquer uma das estratégias, os *workers* deverão receber 1 crédito por cada sequência de códigos *hash* testada e 10 créditos por cada *match* encontrado. Os créditos devem ser retirados ao *plafond* que o dono colocou no *task group*, à medida que forem sendo atribuídos aos *workers* dos respectivos utilizadores;

R4. O sistema deverá ainda permitir gerir a propagação das actualizações, ou seja, sempre que for encontrado um *match*, todos os *workers* deverão ser informados para não duplicar trabalho desnecessário; assim que todos os *matches* dos *hash codes* do 2º conjunto forem identificados, todos os *workers* devem ser informados para pararem a sua execução;

[1] URL: https://raw.githubusercontent.com/danielmiessler/SecLists/master/Passwords/darkc0de.txt

R5. O sistema deve ainda garantir a tolerância a falhas, ou seja, se o servidor ou um dos nós falhar, então os restantes nós devem assegurar a continuidade do serviço, dos *task groups* e respectivos sub-conjuntos de sequências de caracteres a analisar.

R6. Deverão ainda ser consideradas questões de segurança tanto na autenticação como na partilha de conteúdos de modo a controlar acessos indevidos aos recursos partilhados. Numa primeira fase pode considerar-se autenticação em *plain text* e numa segunda fase considerar a utilização de um mecanismo de autenticação baseado em *JSON Web Token* (JWT)[2].

De acordo com o descrito, o principal objectivo será o desenvolvimento do serviço de partilha de processamento, de forma distribuída. O sistema deve facilitar a organização e a coordenação remota dos vários grupos de *workers* que acedem de forma concorrencial aos mesmos recursos. Paralelamente devem considerar com os aspectos de tolerância a falhas e as questões de segurança.

Serão valorizados os projectos que tirem partido da utilização de ***threads*** para melhorar a arquitectura e eficiência do código (tanto no servidor como nos clientes), bem como tirem partido das arquitecturas *multicore* existentes actualmente. Deverão também identificar e implementar, sempre que necessário, a respectiva sincronização destas *threads* no acesso a regiões críticas partilhadas.

1.4. Ferramentas Tecnológicas a Utilizar

Há vários padrões de design (***design patters***[3]) que podem ajudar na organização da estrutura e do comportamento da aplicação distribuída solicitada. Por exemplo, o *observer design pattern* é muito utilizado para gerir sistemas com comunicação *publish/subscribe*. Este padrão permite aos clientes subscrever certos assuntos/eventos para posterior notificação. Outro exemplo, o *factory method design pattern*, é também uma solução adequada para criar sessões ou outros tipos de objectos, de acordo com o perfil dos

[2] URL: https://jwt.io/

[3] URL: http://pages.cpsc.ucalgary.ca/~kremer/patterns/; URL: http://www.fluffycat.com/java/patterns.html

utilizadores. Um padrão também relevante será o *visitor design pattern*, que permite o envio/execução remota de operações numa estrutura de dados pré-definida (e.g., manipulação de um sub-conjunto de dados partilhado, árvore de um sistema de ficheiros).

O sistema proposto deve ser implementado recorrendo a tecnologias e ferramentas de middleware oferecendo comunicação síncrona (e.g. RMI) e comunicação assíncrona (e.g. RabbitMQ), podendo eventualmente utilizar-se outras tecnologias e mecanismos de comunicação sempre que se justifiquem (e.g. *sockets UDP*).

Os alunos são convidados a explorar soluções arquitecturais e de *design* que melhor se coadunem à implementação dos algoritmos de gestão, coordenação e sincronização da aplicação distribuída proposta. Devem dar atenção especial às questões de autenticação, sincronização, coordenação, tolerância a falhas e segurança. Sugere-se que utilizem soluções de design e algoritmos semelhantes aos abordados nas aulas.

2. Grupos, Agenda e Relatórios

Os alunos devem organizar-se em grupos de 3 elementos (máx). Devem planear e dividir bem o trabalho de modo a que todos participem na elaboração do mesmo.

Estão previstos dois momentos de entrega que serão agendados na plataforma de *elearning*. Numa primeira fase deverá ser entregue um ficheiro Argo com os diagramas UML de classes bem como os diagramas de sequências de mensagens. Numa segunda fase deverão ser entregues os diagramas UML finais e a implementação efectuada (código) com a respectiva documentação (gerada a partir dos comentários no código).

Deverão adoptar uma arquitectura modular, faseando o trabalho e permitindo uma implementação incremental, solucionando inicialmente os problemas mais fáceis e posteriormente resolvendo as situações mais complexas. Devem focar-se nos aspectos essenciais: arquitecura de distribuição, autenticação, sincronização e propagação de actualizações, tolerância a falhas e segurança.

No final devem entregar num único ficheiro ZIP todo o software desenvolvido (src), a documentação gerada e os respectivos diagramas UML.