Distributed Photo Collage

Relatório do Trabalho de Melhoria

Bruno Rocha Moura — 97151

Introdução

O presente relatório pretende descrever os procedimentos utilizados para a resolução do trabalho de melhoria de Computação Distribuída do segundo ano da Licenciatura em Engenharia Informática pela Universidade de Aveiro.

Como motivação, nos foi proposta a implementação de um sistema distribuído que deve receber uma lista de imagens, redimensioná-las a uma altura comum para então uni-las horizontalmente tal que o resultado seja uma única imagem em tira.

A arquitetura do sistema deve ser baseada no algoritmo Map-Reduce, onde um processo *broker* delega tarefas à processos *worker* de forma que o trabalho computacional é distribuído entre os diferentes processos.

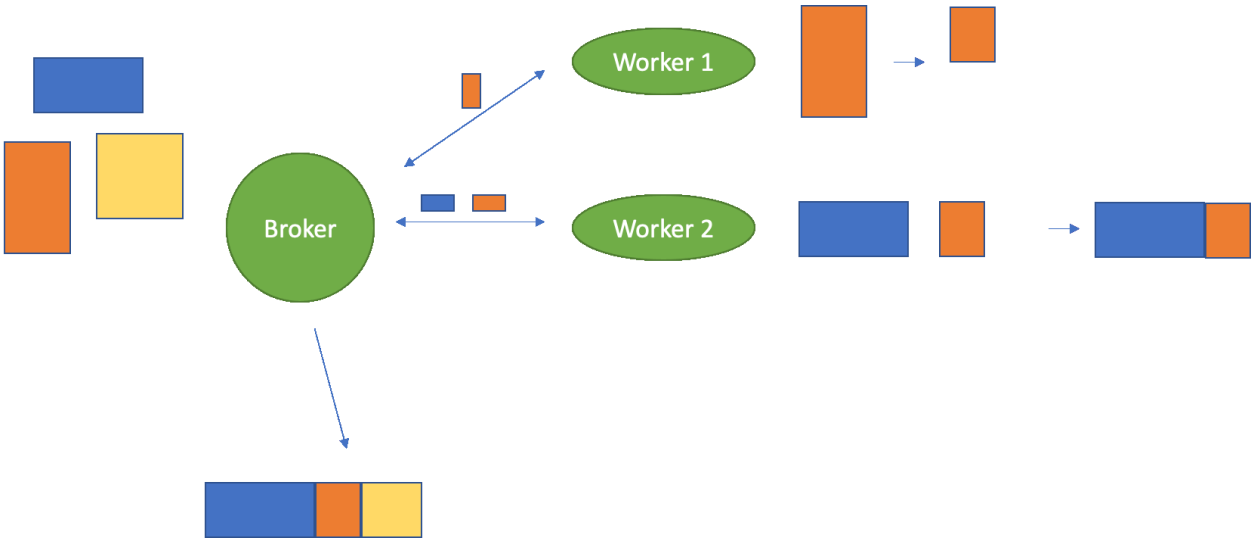

Fig1. - Ilustração fornecida pelo guião de motivação

A motivação também requer que o sistema seja desenhado com a possibilidade em mente que os computadores onde os processos *worker* são executados são lentos e pouco fiáveis. Em função disso, o sistema deve ter uma alta tolerância e recuperação de falhas.

Por último, mais uma restrição é que a comunicação entre os processos deve ser feita por mensagens seguindo um protocolo à ser implementado por nós. Essas mensagens devem ser trocadas via sockets UDP.

Problemas Iniciais

A maioria das complicações no desenvolvimento do sistema foram decorreram protocolo de comunicação UDP. Por ser um protocolo *Fire-and-Forget*, não há garantias que mensagens enviadas chegarão no seu destino ou em que ordem.

Isso gera problemas quanto à atribuição de tarefas. Por exemplo, a situação onde um *broker* atribui uma tarefa a um *worker*, mas a mensagem se perdide no caminho. Na ausência de algum mecanismo de controle, o *broker* não atribuíria essa tarefa para nenhum outro *worker* (pois ele acredita que já tem alguém realizando-a) e o *worker* não só não a realizaria (pois não recebeu a mensagem) como também não receberia nenhuma outra (o *broker* não delega tarefas para *worker*s com tarefas pendentes). Dessa forma, a tarefa jamais seria concluída e o *worker* permaneceria ociosos dali em diante.

O problema se agrava em combinação com outra restrição do UDP — o seu limite relativamente baixo para tamanho de pacotes (65.507 bytes). As imagens comunicadas entre os processos podem facilmente exceder esse limite, principalmente durante os estágios finais onde as imagens são unidas. Se a comunicação fosse feita via TCP, uma solução fácil seria quebrar a mensagem em pacotes menores e enviá-los sequencialmente para serem posteriormente reconstruídas pelo recetor. Infelizmente, entretanto, essa solução não se aplica ao nosso contexto devida à falta de garantia quanto a ordem e receção de pacotes.

Questões de *networking* à parte, houve também um desafio quanto a encontrar uma estrutura de dados para representar as imagens. Essa estrutura deve conter as imagens originais assim como as dos processos intermediários de redimensionamento e união, de forma que é sempre possível perceber as suas relações entre si e quais delas são passíveis de serem unidas.

Soluções

Para o problema da atribuição de tarefas, foi implementado um mecanismo de controle. Após o *broker* atribuir uma tarefa a um *worker*, o primeiro espera uma mensagem de confirmação de receção do segundo. Caso essa confirmação ocorra, a tarefa é atribuída oficialmente ao *worker*, que certamente irá realizà-la exceto caso este morra (mais sobre isso mais adiente). Entretanto, se não houver uma mensagem de confirmação dentro de uma quantidade configurável de tempo, o *broker* assume que o *worker* não recebeu-la e tenta atribuí-la de novo a um *worker* (não necessariamente ao mesmo).

Uma alternativa para esta solução poderia ser o estabelecimento de um tempo limite para a realização de tarefas até se assumir que a mensagem foi perdida. Mas esta seria ineficiente, pois para qualquer dada operação o tempo limite seria ou longo demais (tempo perdido até atribuir a tarefa novamente) ou não suficiente, fazendo com que tarefas particularmente complicadas não pudessem ser realizadas por processos correndo em computadores mais lentos. O tempo limite também teria que ser configurado para cada lista de imagens e altura da imagem final.

Esse mecanismo de controle acabou por ser o maior *bottleneck* do sistema, mas este pode ser reduzido se o tempo limite para confirmação for diminuído o máximo possível.

Para contornar a limitação para o tamanho de pacotes, foi implementado um mecanismo para o pedido, envio e receção de imagens. O conteúdo de cada imagem do sistema é dividido em fragmentos, cada um dos quais tem tamanho inferior a um limite estabelecido. Quando um processo quer anunciar a existência de uma imagem, este envia uma mensagem de anúncio que contém o identificador da imagem e a sua quantidade de fragmentos. O processo recetor então envia mensagens de pedido de um fragmento em específico, correspondidas por mensagens de resposta até o processo recetor ter todos os fragmentos necessários para a reconstrução da imagem. Como é possível que pacotes se percam, pode demorar mais de uma tentativa para receber todos os fragmentos.

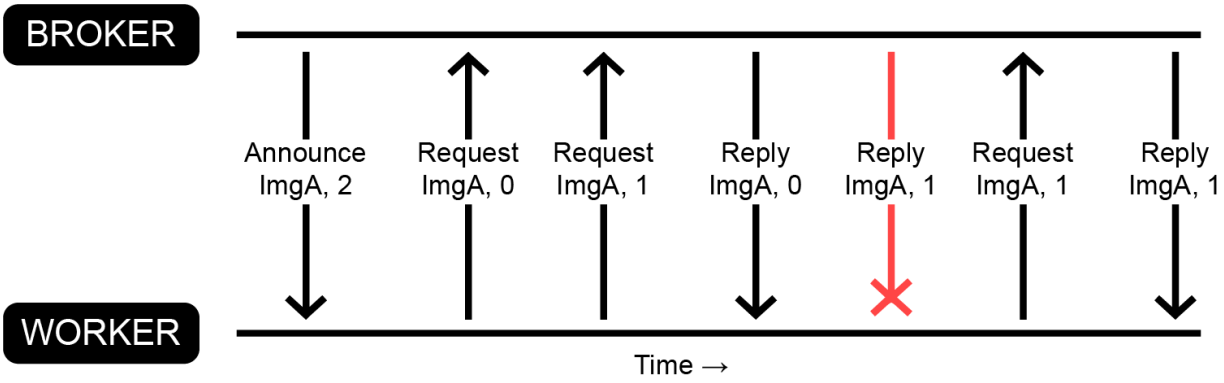

Fig2. - Ilustração do processo de resolução de fragmentos

A estrutura de dados para armazenamento das imagens é uma árvore binária, onde as folhas são as imagens iniciais, inseridas primeiro. Para cada passo intermediário no processo do sistema, novas imagens são inseridas na árvore. O processo de união ocorre entre as imagens mais superficiais vizinhas. O processo acaba quando a imagem raiz é criada.

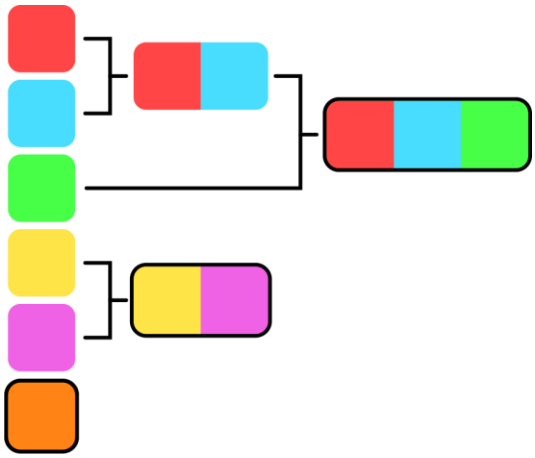

Fig3. - Ilustração da árvore de imagens durante o processo. Imagens elegíveis para união tem bordas preta.

Para evitar que a distribuição de tarefas entre *workers* seja desigual demais, qualquer dada tarefa é delegada ao *worker* com menor número de tarefas concluídas entre todos os disponíveis naquele momento, i.e. sem tarefas pendentes.

Protocolo

O protocolo de comunicação implementado contém as seguintes mensagens:

Hello

Como UDP é um protocolo *connectionless*, deve existir uma mensagem cuja função é anunciar ao *broker* que um *worker* se juntou ao sistema e está pronto para trabalhar. Após a receção da mensagem pelo *broker*, este vai associar o *worker* em questão a um identificador único. Daí, o *broker* retorna a mensagem para o *worker*, mas desta vez contendo o seu identificador. O único propósito do identificador do lado do *worker* é imprimir no ecrã, por questões de *debugging*.

Keepalive

Gestão de falhas faz parte do cerne da problemática desse projeto, uma vez que é preciso assumir que os *worker*s vão estar a correr em computadores pouco fiáveis. Para isto, é essencial que exista algum mecanismo para identificar quando um *worker* vai a baixo, para que nenhuma tarefa seja delegada-lhe. Mensagens de *keepalive* são enviadas periodicamente entre o *broker* e os seus *workers* para assegurar que estes ainda se mantém responsivos. Caso um *worker* não responda a um número configurável de mensagens *keepalive* em sequência, é assumido que alguma falha aconteceu e ele é tratado como morto. É claro, depois deste se recuperar da falha e enviar uma mensagem de *hello*, este voltará a participar da rede. Tarefas de *workers* mortos são atribuídas à um novo *worker*.

Task Confirm

Como descrito anteriormente, deve haver uma confirmação por parte do *worker* que a mensagem de atribuição de tarefa foi recebida. Essa mensagem não possui nenhum parâmetro, uma vez que é possível fazer a resolução do identificador do *worker* e da sua tarefa a partir do endereço do remetente da mensagem. Para além da confirmação que a atribuição de uma tarefa foi bem-sucedida, outro uso para esta mensagem é para que um *worker* tenha a confirmação que o resultado da sua tarefa foi recebido pelo *broker*. Após o *worker* concluir a sua tarefa, ele envia a sua mensagem de conclusão de tarefa em períodos regulares até receber uma mensagem de confirmação do *broker*.

Resize Request

Uma das três mensagens de anúncio de imagem, acompanhada de parâmetros pertinentes à operação de redimensionamento. Quando um *broker* atribuí um *worker* a uma tarefa de redimensionamento, essa mensagem é enviada. Ela contém o identificador único da imagem em questão, bem como a sua quantidade de fragmentos e a altura desejada do redimensionamento. O *worker* recetor precisa então reconstruir a imagem a partir do identificador e do número de fragmentos para poder então operar sobre ela.

Merge Request

Outra mensagem de anúncio de imagem. Quando um *broker* atribuí um *worker* a uma tarefa de união, essa mensagem é enviada. Ela contém os identificadores de ambas as imagens, assim como os seus números de fragmentos. A reconstrução das imagens é feita da mesma maneira ilustrada anteriormente.

Operation Reply

A terceira e última mensagem de anúncio de imagem. Quando um *worker* termina com a sua tarefa, essa mensagem é enviada ao *broker*. Ela contém o identificador da nova imagem assim como a sua quantidade de fragmentos. A reconstrução das imagens por parte do *broker* é feita da mesma maneira.

Fragment Request

Após receber uma mensagem de anúncio de imagem, o processo precisa pedir pelos fragmentos que a constituem. Para isso, ele envia esta mensagem, composta do identificador da imagem e o índice do fragmento em questão, para cada fragmento que não recebeu ainda. Essas mensagens são enviadas continuamente enquanto houverem fragmentos faltando.

Fragment Reply

Quando um processo recebe uma mensagem de pedido de um fragmento, é responsabilidade deste enviá-lo para o processo requerente. O fragmento da imagem é enviada numa mensagem de resposta contendo o identificador da imagem, o índice do fragmento em questão, assim como o conteúdo do fragmento — uma fração da imagem codificada em base64. A reconstrução da imagem pelo processo recetor é trivial depois de todos os fragmentos serem recebidos, já que estes são enumerados com os seus índices.

Done

Uma simples mensagem enviada para os *workers* quando o *broker* é desligado, informando-os que podem se desligar também.

Resultados

Para avaliar a performance do sistema em função do número de imagens e de *workers* trabalhando em simultâneo, foi realizado um simples experimento:

.	1 Worker	4 Workers
50 Imagens	00:00:11	00:00:05
100 Imagens	00:00:22	00:00:08
500 Imagens	00:02:25	00:00:45
1000 Imagens	00:06:50	00:02:01

Fig4. - Performance do sistema.

Todos os experimentos foram feitos com as imagens disponibilizadas pelo professor e escolhidas arbitráriamente. Todas as corridas foram feitas no mesmo computador e com parâmetros de execução idênticos (mesma altura final, tempo máximo para confirmação de tarefas, sem simulação de falhas, etc.)

Nota-se que o fator que mais afeta o tempo de execução é o tempo máximo para aceitação de uma tarefa. Portanto, deve-se se escolher o menor tempo de aceitação possível que a configuração da rede permite para assegurar a melhor performance.