Distributed Photo Collage

2021/2022

Gonçalo Rodrigues Silva – 103668
Guilherme Costa Antunes – 103600

Computação Distribuída
Prof. Diogo Gomes

deti
universidade de aveiro

Índice

Introdução

No seguimento do projeto de recurso proposto na unidade curricular de Computação Distribuída, o seguinte relatório tem como finalidade apresentar e explicar a solução formulada para a questão colocada.

A proposta consiste numa rede centralizada num broker com vários workers independentes que devem simular computadores antigos pouco fiáveis. O objetivo é utilizar esses workers para colarem e redimensionarem imagens por ordem de data da última modificação e desenvolver esse sistema tal que o mesmo seja tolerante a possíveis falhas não perdendo informação.

Implementação

Mediante as condicionantes colocadas à implementação a solução foi implementada na linguagem python, uma vez que foi a linguagem lecionada nas aulas práticas. Como já referido, o sistema é composto por um broker centralizado e vários workers independentes sendo que a comunicação entre broker e workers ocorre mediante sockets UDP, obrigando assim a maiores cuidados no sentido de evitar perdas de dados e à construção do formato de mensagens usadas para comunicar, ao contrário daquilo que um protocolo TCP ofereceria. O protocolo UDP tem como vantagens ser mais rápido, simples e eficiente no entanto é preciso acautelar a perda de pacotes pois não há retransmissão dos mesmos neste protocolo.

Sendo um dos objetivos do trabalho a distribuição eficiente de trabalho pelos diferentes workers foram também implementados algoritmos no sentido de reduzir o tempo de execução total da solução.

Scripts

- **broker.py**
  Inicializa o broker da rede.

  ```
  $ python3 broker.py -f images_folder -s heigth
  ```

  - **images_folder** nome da pasta com as imagens iniciais
  - **heigth** altura da imagem da colagem

- **worker.py**

Inicializa e introduz um novo broker na rede

```
$ python3 worker -b broker_adress
```

- **broker_adress** endereço do broker

Protocolo

Procedimentos de envio e receção de mensagens

- ### Envio

  Para enviar uma mensagem a mesma sofre marshalling e é precedida pelo marcador "<inicio>" e procedida pelo marcador "<fim>", ambos codificados em código ASCII uma vez que neste código todos os símbolos ocupam 1 byte. Esta característica permite mais facilmente calcular o tamanho da mensagem final a enviar. Tendo sido adotado o tamanho padrão e máximo de uma socket UDP, no caso de 65507 bytes, esta é dividida, se necessário, e enviada por partes, intervaladas por 0.02 segundos entre envios, para o recetor da mesma. Antes de qualquer trecho da mensagem codificada ser enviado é-lhe acrescentado um marcador do tipo "" no início do trecho e "" no fim, variando "i" entre os valores 0, 1, 2 e 3, 1 se for uma mensagem de *return_img_resized*, 2 se for uma mensagem de *return_img_concat*, 3 se for uma mensagem de *im_alive* e 0 se for qualquer outro tipo de mensagem.

- ### Receção

  Quando recebe uma mensagem o programa espera uma do mesmo tamanho que foi definido no envio e após a receção são armazenados a mensagem recebida e o endereço que a enviou.
  São removidos os marcadores do tipo "" e "", sendo o valor de "i" armazenado como *flag*. Caso tenha sido recebida uma mensagem antes do *timeout* definido para o programa em questão é devolvido para o programa principal a mensagem sem os marcadores numéricos, o endereço que a enviou e a flag armazenada. Caso seja atingido o *timeout* são devolvidos valores *None*. No nó que recebe a mensagem devolvida este adiciona as mensagens recebidas mediante o endereço que as envia e a flag que é retornada e quando este reconhece que a mensagem começa pelo marcador "<inicio>" em bytes ASCII e termina pelo marcador "<fim>" em bytes ASCII são removidos os marcadores e a mensagem após ser reconvertida de bytes pode ser tratada pelo nó.

Mensagens

Mensagem	Descrição	Formato
Worker_register	Enviada pelo worker ao ser inicializado	**{"type": "worker_register"}**
Give_work_resize	Enviada pelo broker para atribuir um trabalho de redimensionar a um worker	**{"type": "resize", "args": {"img": img, "heigth": height}}** **img:** imagem, em bytes, a redimensionar **height:** altura a que deve ser redimensionada a imagem
Give_work_concat	Enviada pelo broker para atribuir um trabalho de colagem a um worker	**{"type": "concat", "args": {"img1": img1, "img2": img2}}** **img1 e img2:** imagens, em bytes, a colar numa nova
Return_img_resize	Resposta a um give_work_resize com a respetiva imagem redimensionada	**{"type": "image_resized", "args": {"img": img}}** **img:** imagem, em bytes, após redimensionamento
Return_img_concat	Resposta a um give_work_concat com a nova imagem após colagem	**{"type": "image_concat", "args": {"img": img}}** **img:** imagem, em bytes, após colagem de outras duas
Im_alive	Mensagem enviada periodicamente pelos workers para o broker saber que estes estão online	**{"type": "im_alive"}**
Close_worker	Enviada pelo broker para dar ordem de fecho a um worker	**{"type": "close_worker"}**
Error	Enviada por um worker quando não consegue interpretar a mensagem enviada pelo broker	**{"type": "error"}**

Execução do sistema

O sistema é iniciado aquando da atribuição de uma pasta de ficheiros ao broker e da altura da tira final

Ordenamento das imagens

```
for img in os.listdir(folder):
    shutil.copy(folder+"/"+img, self.temp_folder+os.path.splitext(img)[0]+".png")
    images.append(img)

images=sorted(images, key=lambda x: os.stat(folder+"/"+x).st_mtime)
images=[os.path.splitext(img)[0]+".png" for img in images]

for i in range(0,len(images),2):
    self.combs.append((images[i], images[i+1]))
    if i+2==len(images)-1:
        self.combs.append((images[i+2], None))
        break

nl=0
l=len(self.combs)
while nl+1!=l:
    for i in range(nl,l,2):
        self.combs.append((self.combs[i][0], self.combs[i+1][0]))
        if i+2==l-1:
            self.combs.append((self.combs[i+2][0], None))
            break
    nl=l
    l=len(self.combs)
for comb in list(self.combs):
    if comb[1] is None:
        self.combs.remove(comb)
```

Para garantir que as imagens são coladas na ordem certa foi desenvolvido um algoritmo que numa primeira instância copia todas as imagens para uma pasta temporária, convertendo-as em PNG, no sentido de evitar limitações de tamanho da tira. Os nomes das imagens são armazenados numa lista e estas são agrupadas duas a duas. Numa segunda fase a primeira imagem de cada grupo é agrupada com a primeira do grupo a seguir e assim sucessivamente até só restar um par, verificando-se uma organização em árvore. É utilizada esta estratégia pois quando duas imagens são coladas a imagem final fica com o nome da primeira do conjunto. Este algoritmo permite não só garantir que as imagens são coladas pela ordem certa mas também que vários workers podem colar pares diferentes sem ter de esperar pelo primeiro para colar um segundo de forma sequencial e pouco eficiente.

Registo de workers e atribuição de trabalho

Quando um worker é inicializado a primeira ação que este faz é enviar um pedido de registo para o broker que caso este não tenha nenhum worker registado no mesmo endereço aceita-o.
Sendo que o broker contém uma lista dos workers a quem este deu trabalho, quando a lista de workers a trabalhar tem tamanho diferente da lista de workers registados é sinal de que existem workers disponíveis para receber um trabalho. São percorridas as combinações geradas no ponto anterior e são avaliadas as suas imagens, ou seja, se a imagem pode ser alterada no momento e se esta tem a altura desejada para a tira final. Uma imagem está disponível para ser alterada quando a mesma não está reservada para ser usada numa operação futura, quando a combinação referente à imagem não implica imagens que têm de ser alteradas antes desta combinação ou quando a imagem não está a ser trabalhada atualmente. Em caso positivo de ambas as condições avançamos para a imagem seguinte que em igual caso positivo nos conduz para a colagem das duas

```
def checkImgAvailable(self, comb):
    for i in self.reserved:
        if self.reserved[i][0]==1:
            if self.reserved[i][1][0] in comb:
                return False
        else:
            if comb==(self.reserved[i][1]):
                return False

    idx=self.combs.index(comb)
    for i in comb:
        for c in range(len(self.combs)):
            if idx > c and i in self.combs[c]:
                return False
    for i in comb:
        for w in self.works:
            if i in self.works[w][0]:
                return False
    return True
```

imagens que, à semelhança de caso seja necessário alterar a altura antes da colagem é escolhido o melhor worker disponível ou futuramente disponível.

Escolha do melhor worker (função *getBestWorkerAvailable*)

O algoritmo de escolha do melhor worker tem em conta absolutamente todos os workers registados para trabalhar para o broker. De entre os disponíveis são ordenados por ordem crescente de tempo médio de execução para a operação que se pretende realizar e em caso de ainda não ter trabalhos feitos são tomados como prioritários. De entre os workers atualmente a trabalhar são ordenados pelo mesmo critério numa lista diferente. É selecionado o worker com menor média de tempo de execução e caso este pertença à lista de workers a trabalhar é avaliado se compensa esperar que este worker acabe o trabalho, tendo em conta o tempo médio deste e o tempo em que iniciou o trabalho atual, e caso este seja menor que o tempo médio do worker livre mais rápido é reservado para a operação sobre a qual foi avaliado e adicionado à lista de espera.

Processamento da informação recebida dos workers

Como já apresentado na parte do Protocolo os workers podem enviar vários tipos de mensagens para o broker. Quando a mensagem é do tipo "im_alive" caso o worker esteja numa lista de workers que se quer verificar que estão online este é removido da mesma, assim como acontecerá se for recebido qualquer pacote vindo desse worker. Se a mensagem for do tipo "error" é sinal de que a última mensagem enviada para o worker foi mal recebida e que é necessário repetir o processo de envio. Caso a mensagem seja de "worker_register" ocorre o já referido registo do worker no broker caso ainda não esteja resgistado. No caso de a mensagem ser do tipo "image_resized" ou "image_concat" a imagem contida na mensagem é convertida novamente para o formato PNG e substitui a com o mesmo nome. Caso se trate de uma colagem a imagem na segunda posição da combinação colada é eliminada da pasta assim como a combinação é retirada da lista de combinações a realizar. Após qualquer uma destas duas operações, caso haja reservas para o worker que acabou de fazer a entrega de imagens essas reservas são executadas dando novamente trabalho ao worker, caso contrário este será apenas removido da lista de workers a trabalhar. Sempre que uma imagem é entregue é atualizada a lista de estatísticas do worker que a entregou convenientemente.

Conversão de imagens

Quando um worker precisa de enviar o resultado da sua operação para o broker ou o broker precisa de enviar imagens para o worker tratar é necessário convertê-las a bytes e para as receber igualmente.

```python
def encodeImg(self, img):
    bytesimg = io.BytesIO()
    img.save(bytesimg, format="png")
    return bytesimg.getvalue()
```

```python
def decodeImg(self, img):
    out = io.BytesIO()
    out.write(img)
    return out
```

Tratamento da imagem final

Quando a última combinação é colada e ficamos apenas com uma imagem na pasta temporária essa imagem é o nosso resultado final e é transferido para a pasta atual de execução e apresentada antes do término do sistema. Quando obtemos a imagem final o broker dá ordem para encerrar todos os workers ligados pela mensagem de "close_worker". Após a apresentação da imagem são impressas todas as estatísticas referentes aos workers que trabalharam para aquele broker e finalmente o broker é encerrado.

Resiliência a falhas

As falhas identificadas foram a desconexão inesperada dos workers, elevados tempos de resposta a trabalhos dados aos workers e erros na leitura das mensagens recebidas do broker. Para as duas primeiras está definido um *TIMEOUT*, que deve ser ajustado mediante a quantidade de workers que se prevê que vão trabalhar para o broker e o tamanho das imagens que vão ser transferidas entre nós. Quando esse timeout é atingido por um worker este é adicionado a uma lista de workers que se não responderem serão considerados como desconectados. Caso passe outro valor de timeout e o broker não tenha recebido nenhum pacote desse worker em risco o worker é considerado como desligado, os seus trabalhos são cancelados e as estatísticas deste passam para um histórico.

No entanto também é possível que o worker continue a dar sinal de vida, mas que simplesmente não envie o trabalho para que foi delegado. Neste caso, quando o tempo a que foi delegado o trabalho tiver sido há mais de 2 vezes o valor do timeout a operação é cancelada e atribuída novamente ao melhor worker disponível.

No caso de o worker identificar um erro na mensagem que foi enviada pelo broker este avisa imediatamente o broker e o broker procede ao reenvio da mesma mensagem.

Simulação de computadores velhos

Como pedido no enunciado foram adicionados time.sleeps antes dos workers enviarem o resultado dos seus trabalhos. De maneira a tornar o atraso regular e consistente quando o worker é criado é gerado um número inteiro aleatório entre 4 e 10 segundos. Quando é dado time.sleep antes do envio de resultados é adicionada uma variação aleatória à constante anteriormente gerada entre -1 e 1 segundo para garantir que o valor de delay do worker varie um pouco e sempre em torno do mesmo valor.

Também foi adicionado um algoritmo que a cada tempo mínimo de 5 segundos gera um valor aleatório entre 0 e 1 e caso esse valor seja maior que o quociente entre a subtração do tempo decorrido desde a última execução deste algoritmo em segundos a 100 e 100 então o worker é fechado inesperadamente simulando assim a “morte” inesperada desse nó. Existe uma macro definida no início do script worker.py que será multiplicada ao valor temporal que é subtraído a 100 ou seja, se essa macro estiver a 0 a chance de o worker se desligar sozinho é nula, se estiver a 1 é normal, se estiver a 2 duplica, a 3 triplica, etc.

Conclusão

O objetivo de juntar todas as fotos da pasta dada foi alcançado assim como a resiliência a falhas, a eficiência de processamento, escolhendo preferencialmente os workers esperadamente mais rápidos, o balanceamento de operações, uma vez que que não são excluídos workers mas dando-se preferência à velocidade de execução, qualquer worker pode realizar qualquer operação, há transferência de imagens entre os nós da rede e a comunicação entre broker e workers. Deste modo acreditamos que o trabalho foi bem conseguido e concluído com sucesso já que para além dos pontos enumerados acima foram cumpridas as regras para os alcançar e foram abordados os temas referidos.