Computação Distribuída – Guião de Avaliação 02

Diogo Gomes & Mário Antunes

Maio 2019

1 Introdução

O objectivo deste guião de avaliação é implementar um sistema distribuído que implemente o algoritmo Map/Reduce (ver Figura 1).

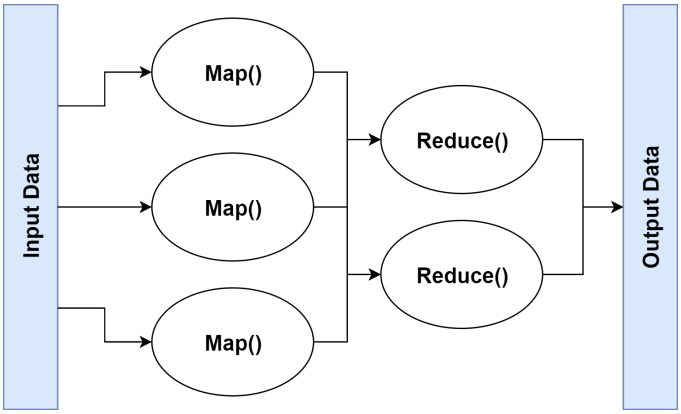

Figura 1: Esquema simplificado do modelo Map/Reduce.

2 Map/Reduce

Map/Reduce é uma técnica de processamento distribuída e um modelo de programação. O algoritmo contém duas tarefas importantes, de seus nomes *Map* e *Reduce*. A tarefa de *Map* é converter um conjunto de dados noutro,

onde os elementos individuais são divididos em tuplos. A tarefa de *Reduce* é receber os tuplos da operação de *Map* e combinar estes num conjunto de tuplos mais pequeno. Como indicado pelo nome, a tarefa de *Reduce* é sempre executada depois da tarefa de *Map*.

A maior vantagem deste modelo de programação é a capacidade do mesmo em escalar horizontalmente em função dos recursos computacionais disponíveis. No entanto, nem sempre é trivial converter um função numa sequência de Map/Reduce. Mas uma vez descrita como tal, a operação pode escalar para centenas de nós de processamento.

3 Problema

Um dos problemas clássicos resolvidos com Map/Reduce é a contagem de palavras num texto de grandes dimensões. Para este trabalho deverá desenvolver uma solução que permita criar um histograma de palavras usadas num texto passado por argumento a um processo coordenador.

Deverá implementar uma arquitectura distribuída composta por um coordenador e múltiplos processos trabalhadores (*workers*), ver Figura 2. Os *workers* serão responsáveis por implementar as tarefas de *Map* e *Reduce*. Uma vez que os *workers* são genéricos, deverão receber a sua tarefa (Map/Reduce) do coordenador. A tarefa do coordenador é pois garantir uma distribuição de esforço pelos *workers* alocando trabalho de acordo com as necessidades.

Assumindo que o coordenador recebe um caminho (*path*) para um ficheiro texto (.txt) que contem o texto literário, este deve ler o ficheiro e partir o mesmo em pedaços (blobs) para que os *workers* possam sucedânea-mente desempenhar as seguintes tarefas:

- Mapear parágrafos de texto (a partir do texto original) numa lista de *tokens*. Nesta tarefa o coordenador entrega *blobs* de texto aos *workers*. Estes fazem a *Tokenização* dos parágrafos de texto, sendo o resultado uma lista de palavras (com contagem 1 para cada palavra individual, podendo existir entradas repetidas na lista). Nesta tarefa são ainda removidas as pontuações e outros caracteres que não constituem palavras.

- Contar ocorrências de palavras numa dada lista de palavras. Nesta tarefa a lista de palavras anteriormente gerada é transformada numa

contagem (dicionário em que a chave é a palavra e o valor o número de ocorrências dessa palavra).

- Reduzir as contagens a uma única contagem (texto total). Nesta tarefa são agregados os dicionários da tarefa anterior até eventualmente existir apenas um dicionário - o resultado final.

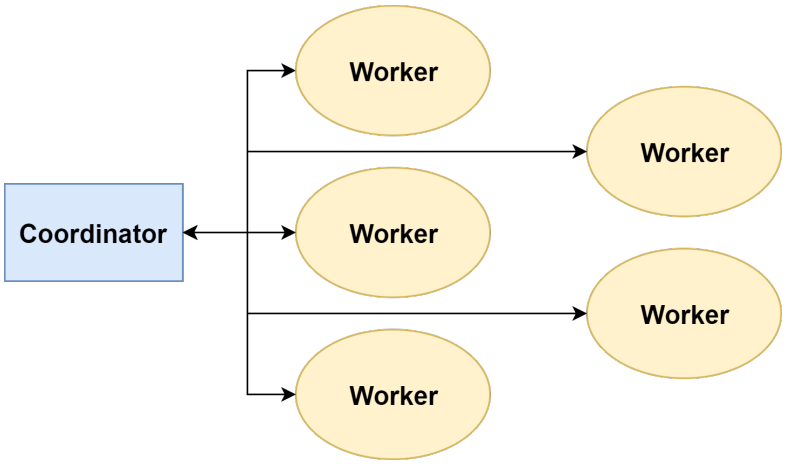

Figura 2: Arquitectura simplificada da solução.

Para notas superiores a 14 valores, é necessária a implementação de mecanismos de **tolerância a falhas**. Será fornecido um *script* adicional que periodicamente irá matar um dos processos com uma probabilidade baixa. Nenhuma mensagem será corrompida nem perdida (devido ao requisito de uso do protocolo TCP). O **coordenador** deverá ter um processo de **backup** e comunicar o endereço deste aos seus *workers*.

3.1 Requisitos

A implementação deverá ser feita em Python 3.5, e deverá consistir de dois ficheiros principais: `coordinator.py` e `worker.py` (podem existir outros ficheiros que são incluídos por estes). Cada um destes ficheiros deverá lançar um processo independente que poderá inclusive executar numa máquina distinta (em rede local).

Os *workers* conhecem o endereço do coordenador (passado por argumento).

A comunicação entre processos deverá ser feita utilizando protocolo TCP no porto 8765.

O coordenador deverá ser o primeiro processo a iniciar, sendo que um segundo coordenador (backup) deverá detectar a existência do primeiro e tornar-se backup deste.

Em caso de falha o sistema deverá recuperar e apresentar um resultado final correcto (sendo a escolha dos mecanismos de recuperação responsabilidade dos alunos - a documentar na apresentação)

3.2 Protocolo

Nesta secção são apresentados os objectos JSON a ser trocados entre o coordenador e os *workers*, ver Figura 3.

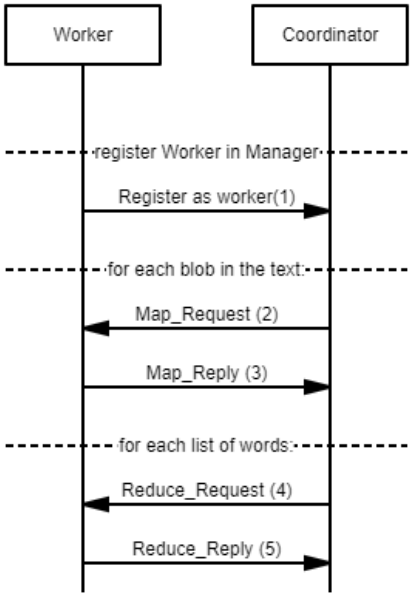

Figura 3: Protocolo para um coordenador e um *worker*.

A primeira mensagem (Mensagem 1) é usada para um *worker* registar-se no coordenador e estar apto a receber tarefas:

```
{
  "task": "register",
  "id": <worker_id>
}
```

A escolha do tipo/formato do *id* é sua responsabilidade.

A próximas mensagens representa a tarefa de Map (Mensagem 2 e 3 respectivamente). A primeira representa a atribuição da tarefa pelo coordenador ao *worker*. A segunda representa a resposta que o *worker* envia quando termina a tarefa.

```
{
  "task": "map_request",
  "blob": <encoded_text>
}
```

O blob é determinado pelo manager, o codigo fornecido já parte o texto em blobs.

```
{
  "task": "map_reply",
  "value": [(<word1>, 1),(<word2>, 1),...]
}
```

Repare que o valor retornado é uma lista de tuplos com as palavras e o numero 1 (uma ocorrencia). Esta lista constitui um histograma base.

Por fim são apresentadas as mensagens usadas para a tarefa de Reduce (Mensagem 4 e 5 respectivamente). A primeira representa a atribuição da tarefa de Reduce pelo coordenador ao *worker*. É enviado uma lista de listas, cada lista representa um histograma de palavras. A segunda representa a resposta (histograma) que o *worker* envia quando termina a tarefa.

```
{
  "task": "reduce_request",
  "value": [[(<word1>, <count1>),(<word2>, <count2>),...],
  [(<word1>, <count1>), (<word2>, <count2>), ...]]
}
```

```
{
  "task": "reduce_reply",
  "value": [(<word1>, <count1>),(<word2>, <count2>),...]
}
```

O programa termina quando todos histogramas devem ser reduzidos a apenas um.

4 Objectivos

- Implementar um sistema distribuído capaz de contar palavras num texto literário (exemplo: Os Lusíadas)
- A codificação de mensagens deverá ser feita em JSON [1], ver protocolo em Subseção 3.2.
- Comunicação através do protocolo TCP [2].
- Implementar *workers* (que podem actuar como *Map* ou *Reduce*).
- Implementar o coordenador que gere a distribuição de tarefas entre os *workers*.
- Implementar um protocolo que permita tolerância a falhas.

5 Notas

- **Atenção:** durante as últimas semanas de aulas o guião poderá sofrer algumas alterações, verifiquem periodicamente actualizações ao mesmo,
- É fornecido um código de ajuda que deve ser usado para implementar e testar a solução [3].
- O código fornecido poderá ser melhorado ao longo do tempo, por isso é recomendado que configurem o git upstream [4] e verifiquem periodicamente a existência de novo código.

6 Avaliação e Data de Entrega

O trabalho deverá ser submetido, até dia 10/06/2019, para o repositório git [5] correspondente (`https://classroom.github.com/g/RgMu8-tU`). O trabalho pode ser realizado em grupos de dois ou individualmente. Os alunos são incentivados a publicar periodicamente o código do projecto. Além do código os alunos devem entregar uma apresentação (em PDF) onde expõem os pontos mais importantes da implementação. A apresentação deve ter no máximo 5 diapositivos (capa não incluída).

Os seguinte pontos serão considerados para a avaliação de forma cumulativa:

- Solução base (9 valores)
  - Apenas um coordenador e um *worker*.
  - Comunicação (TCP) entre o coordenador e o *worker*.
  - Terminar o problema enunciado na Seção 3
- Solução distribuída (1 valores)
  - Existe um coordenador e 4 *workers* cada qual a desempenhar a sua tarefa.
  - Implementação do protocolo segundo o enunciado.
- Solução distribuída robusta (4 valores)
  - Permitir um número indefinido de *workers*.
  - Implementar um coordenador capaz de distribuir tarefas entre todos os *workers* de forma indistinta (não existe pré alocação de tarefas a cada *worker*).
  - Segundo coordenador torna-se backup do primeiro, implementação de um protocolo de replicação
- Tolerância a falhas (4 valores)
  - Coordenador distribui endereço do(s) backup(s) pelos *workers*, eleição de coordenador (algoritmo definido por cada grupo e documentado na apresentação)
  - Sistema é capaz de tolerar a morte de qualquer processo (apenas um processo está em falha a cada momento) sem perdas de dados e com resultado final correcto (explicação dos metodos/tecnicas/algoritmos utilizados na apresentação)
- Performance, elegância e robustez da solução (2 valores)
  - Apresentação com demonstração do impacto das alterações efectuadas em termos de performance (com respectiva analise estatística)
  - Uso correcto da linguagem Python, recorrendo a funcionalidades modernas desta linguagem.

Referências

[1] Python. json – json encoder and decoder. https://docs.python.org/3/library/json.html, 2019. [Online; accessed 2019-02-08].

[2] Python. Socket programming howto. https://docs.python.org/3/howto/sockets.html, 2019. [Online; accessed 2019-02-08].

[3] mariolpantunes. mapreduce. https://github.com/mariolpantunes/mapreduce, 2019. [Online; accessed 2019-05-19].

[4] GitHub. Configuring a remote for a fork. https://help.github.com/en/articles/configuring-a-remote-for-a-fork, 2019. [Online; accessed 2019-04-15].

[5] Roger Dudler. git – the simple guide. http://rogerdudler.github.io/git-guide/, 2019. [Online; accessed 2019-02-08].