Enunciado do Trabalho Prático

LEI – Sistemas Operativos

2012/2013

1 Contexto

Para o processamento de grandes volumes de dados é corrente a utilização do modelo de programação conhecido por **MapReduce**.[1] Este modelo de programação permite, de uma forma relativamente simples, aplicar um processamento modular a grandes conjuntos de dados (*datasets*), normalmente uniformes, promovendo o paralelismo e por isso permitindo bons desempenhos.

A ideia base é proceder ao processamento dos dados em duas fases. Inicialmente, dividimos o dataset e processamos cada porção em paralelo. Este processamento, a que chamamos **Map**, produz um conjunto de pares "chave, valor" para cada porção processada, consoante os dados e a operação que se pretende,

Imagine por exemplo um conjunto de páginas web, uma para cada freguesia do país, em que em cada página tem informação sobre os cidadãos e é possível extrair relações *(nome, desporto praticado)*. Considere um processamento de Map que a partir dessa informação gera pares do tipo "desporto praticado, número de praticantes".

Posteriormente, numa segunda fase, a que chamamos **Reduce**, processamos os resultados do Map de forma a consolidar a informação obtida em cada uma das porções do dataset. O resultado é, mais uma vez, um conjunto de pares. Também o Reduce pode e deve ser feito por vários processos em paralelo.

Para o exemplo anterior, o Reduce produziria o número de praticantes no país por tipo de desporto, consolidando para cada tipo de desporto um par também do tipo *(desporto praticado, número de praticantes)*. Resumindo, partindo de um conjunto de URLs, invocaríamos vários comandos Map aos quais passaríamos um dos URL e obteríamos um conjunto de pares "desporto praticado, número de praticantes" para cada freguesia. Depois, a vários comandos Reduce forneceríamos, a cada, todos os pares obtidos para cada um dos desportos praticados (a chave destes pares). O resultado final seria o total de praticantes por desporto.

Enquanto o Reduce tende a ser um processamento de consolidação, tipicamente fazendo a acumulação de valores da primeira fase, o Map destina-se a processamentos mais complexos quer no tipo de mapeamentos que faz (por exemplo, agregações e combinações de vários tipos de informação) quer na manipulação dos dados de entrada.

2 Exercício

Pretende-se que implemente um programa `mapreduce` capaz de coordenar vários processos de Map e de Reduce.

O programa `mapreduce` deverá receber como argumento os comandos responsáveis por efectuar o Map e o Reduce, da seguinte forma:

```
$ ./mapreduce map reduce
```

[1] Ver http://en.wikipedia.org/wiki/MapReduce como introdução.

Assume-se que o comando *map* faz o seguinte:

1. Recebe um argumento que usará para aceder ou obter uma porção do dataset,
2. Imprime para o *stdout* linhas de texto como resultado, e
3. Cada uma destas linhas tem o formato "*chave valor*", ou seja, duas palavras separadas por um espaço.

Assume-se que o comando *reduce* faz o seguinte:

1. Não recebe argumentos,
2. Recebe texto no seu *stdin*, e
3. Imprime para o *stdout* uma única linha de texto.

Pretende-se que o `mapreduce`, na fase de Map, faça o seguinte:

1. Receba texto do seu *stdin*,
2. Para cada linha lida, execute o comando *map*, usando essa linha como argumento, e
3. Capture o respetivo resultado apresentado no *stdout*.

Na fase de Reduce, espera-se que por cada chave distinta observada nos resultados da primeira fase, faça o seguinte:

1. Execute o comando *reduce*, fornecendo-lhe no *stdin* todos os valores associados a essa chave, um por linha,
2. Capture o resultado produzido no *stdout* desse comando, e
3. Imprima para o seu *stdout*, por cada chave, uma linha contendo a chave e o resultado do *reduce*, separados por um espaço.

A concretização do `mapreduce` deve:

1. Utilizar primitivas de gestão de processos, ficheiros, sinais e *pipes* do sistema Unix na linguagem C.
2. Executar em cada fase concorrentemente várias instâncias do comando respetivo.
3. Nunca exceder um número predeterminado (`MAXCOM`) de comandos em execução.
4. Terminar imediatamente todos os comandos caso seja detetado algum erro.
5. Eliminar no final quaisquer ficheiros intermédios que possam ter sido criados
6. Permitir resolver problemas em que os resultados intermédios (do *map*, a fornecer ao *reduce*), sejam maiores do que é possível armazenar eficientemente em memória.

3 Exemplo

O seguinte exemplo mostra como o `mapreduce` poderá ser utilizado para calcular o histograma dos tamanhos em blocos dos ficheiros na diretoria `/tmp`, ou seja, para cada tamanho, quantos ficheiros foram encontrados:

```
$ find /tmp | mapreduce du "wc -l" | sort -n
16 2
44 2
1332 1
1460 1
$
```

4 Equipas

Para a realização do trabalho os alunos deverão organizar-se em equipas de 2 elementos.

5 Entrega

Os trabalhos deverão ser submetidos no sítio "http://elearning.uminho.pt/" até às 00:00 do dia 1 de junho de 2013. A submissão deverá consistir num único ficheiro (eg. tar, zip) que contenha todos os ficheiros necessários à criação do executável `mapreduce` através do comando `make`. Deverá ainda conter um ficheiro de nome `equipa` com o nome e número dos elementos da equipa.

6 Avaliação

A avaliação será marcada imediatamente após a entrega do trabalho e coordenada com as restantes UCs.