UC Sistemas Operacionais
ICT/UNIFESP

Prof. Bruno Kimura
bruno.kimura@unifesp.br
19/04/2018

LAB 2: Map Reduce

Metodologia: Trabalho individual ou em grupo de no máximo 3 (três) alunos a ser desenvolvido em laboratório de informática através de codificação na linguagem C.

Data de entrega: 03/05/18

Forma de entrega: Código .c deve ser enviado no SEAD. Insira como comentário no código o nome e matrícula de cada integrante do grupo.

Observação: Somente serão aceitos trabalhos **autênticos**. Cópias (entre grupos e/ou de fontes da Internet) serão anuladas.

Descrição:
Utilizando as ferramentas de sincronização e comunicação entre processos discutidas em aula, implemente uma aplicação de contagem de palavras em documentos baseada no modelo de programação **MapReduce**.

Milhares de códigos foram implementados utilizando o modelo MapReduce na Google, incluindo algoritmos para processamento de grafos de larga escala, processamento de texto, mineração de dados, aprendizado de máquina, traduções de máquina, etc [1].

Referência:
[1] Jeffrey Dean and Sanjay Ghemawat. 2008. ***MapReduce: simplified data processing on large clusters*****.** Commun. ACM 51, 1 (January 2008), 107-113. DOI: `https://doi.org/10.1145/1327452.1327492`

Sobre o Modelo MapReduce:
O modelo prevê conceitualmente duas funções:

- **Mapeamento**: produz uma lista de pares `<chave, valor>` a partir de entradas estruturadas de diferentes tipos de pares `<chave, valor>`:

```
map(k1, v1) → list(k2, v2)
```

- **Redução**: produz uma lista de valores a partir de uma entrada que consiste em uma chave e uma lista associada de valores:

```
reduce(k2, list(v2)) → list(v2)
```

Neste exercício de laboratório, as funções de *map* e *reduce* podem ser representadas pelos pseudo-códigos abaixo. A cada ocorrência da palavra *w*, a função *map* emite a sua ocorrência "1". A função *reduce* então realiza a contagem, somando todas as ocorrências emitidas de uma palavra específica *w*.

```
map(String key, String value):
     // key: document name
     // value: document contents
     for each word w in value:
          EmitIntermediate(w, "1");
```

```
reduce(String key, Iterator values):
     // key: a word
     // values: a list of counts
     int result = 0;
     for each v in values:
          result += ParseInt(v);
     Emit(AsString(result));
```

A Figura 1 ilustra a execução de uma implementação baseada no modelo MapReduce [1]. Nesta implementação, a aplicação (*user program*) divide as entradas em pedaços, cada um com um conteúdo diferente do outro (parte de um arquivo ou um arquivo distinto). Várias entidades são executadas de forma concorrente. Há entidades trabalhadoras (*workers*), que são responsáveis pelas tarefas de *map* ou *reduce*. A entidade especial é a mestre (*master*), que é responsável por alocar aos trabalhadores ociosos as tarefas de *map* e *reduce*. Pares intermediários de `<chave, valor>` gerados pelos trabalhadores na tarefa de *map* são buferizados em memória. Periodicamente, o buffer pode ser descarregado em disco, particionado-o em diferentes arquivos por uma função de partição. Um trabalhador alocado na tarefa de *reduce*, lê as partições e ordena os pares pelo valor das chaves. Ao ordenar, todas as ocorrências (valor) de uma mesma chave são agrupadas, facilitando a operação de redução. Finalmente, os trabalhadores em *reduce* iteram sobre os pares ordenados. A saída de uma função *reduce* é anexada ao arquivo de saída. No caso da aplicação de contagem de palavras, a tarefa de redução requer simplesmente a soma das ocorrências de cada palavra (chave), anexando o resultado <chave = *w*, valor = contagem> ao arquivo de saída.

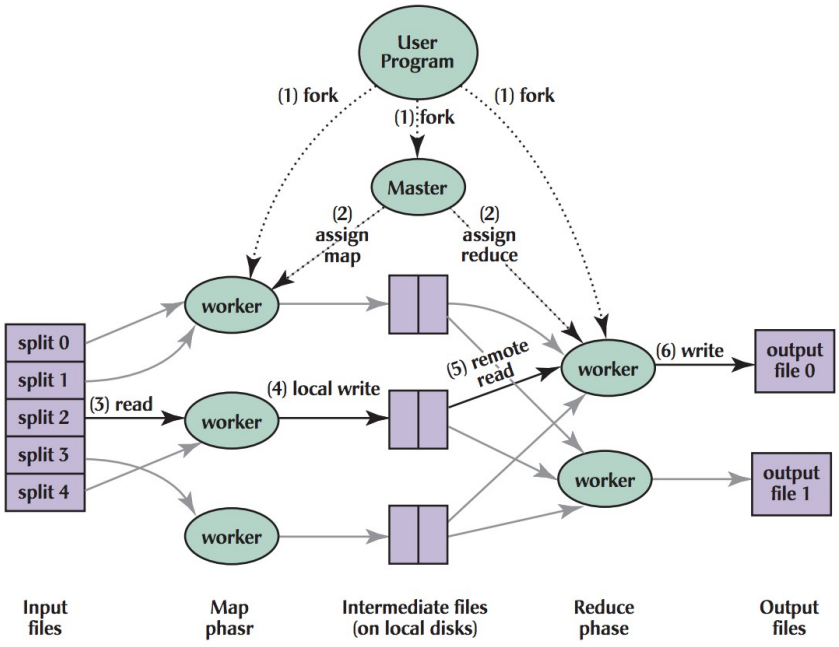

Figura 1: Visão de execução de uma implementação baseada em MapReduce. Fonte: [1]