

**UC Sistemas Operacionais  
ICT/UNIFESP**

Prof. Bruno Kimura  
bruno.kimura@unifesp.br  
19/05/2022

**LAB 2: Map Reduce**

<b>Objetivo:</b>	Implementar um contador de palavras com o modelo map-reduce.
<b>Metodologia:</b>	Trabalho em grupo de 4 (quatro) alunos(as) para desenvolvimento de protótipo em linguagem C.
<b>Entregáveis:</b>	<ul style="list-style-type: none"><li>• Protótipo em arquivos .c</li><li>• Relatório escrito (enviar no doc “Relatório-projeto-2” anexado na tarefa do classroom)</li></ul>
<b>Data de entrega:</b>	<b>02/06/2022</b>
<b>Observação:</b>	A autenticidade do trabalho será verificada. Cópias (entre grupos e/ou de fontes da Internet) implicam em anulação.

**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>:  
$$\text{map}(k1, v1) \rightarrow \text{list}(k2, v2)$$
- **Redução:** produz uma lista de valores a partir de uma entrada que consiste em uma chave e uma lista associada de valores:  
$$\text{reduce}(k2, \text{list}(v2)) \rightarrow \text{list}(v2)$$

Neste trabalho, 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). 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 às tarefas de *map* ou *reduce*. Pares intermediários de <chave, valor> gerados pelos trabalhadores na tarefa de *map* são buferizados em memória. Um trabalhador alocado na tarefa de *reduce*, lê os buffers e agrupa os pares pelas chaves. Ao agrupar, todas as ocorrências (valor) de uma mesma chave são somadas, realizando a operação de redução.

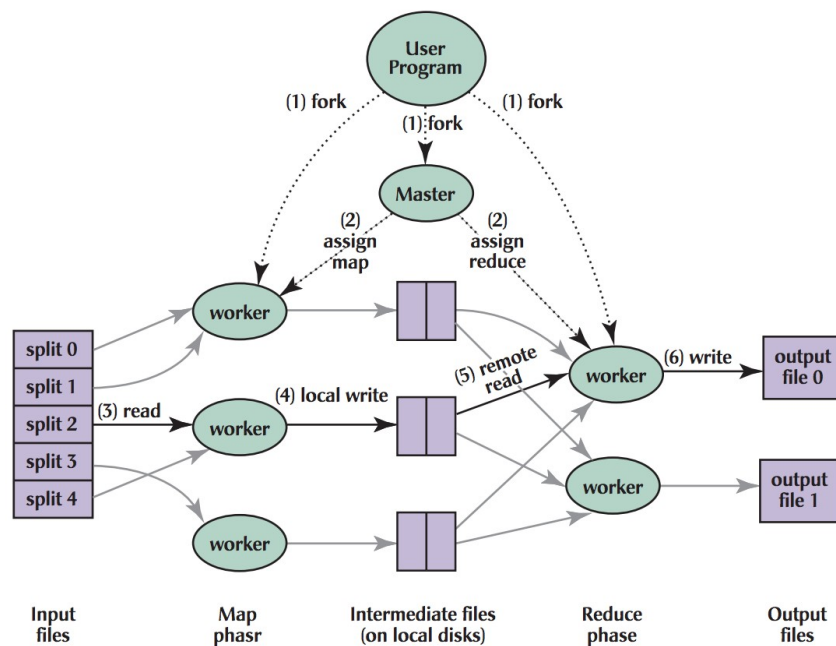


Figura 1: Visão de execução de uma implementação baseada em MapReduce. Fonte: [1]