Instituto de Ciência e Tecnologia – ICT
Universidade Federal de São Paulo - UNIFESP

UC Redes de Computadores

Prof. Bruno Kimura
bruno.kimura@unifesp.br
13/11/2018

Trabalho 4 (MapReduce com Sockets)

- **Metodologia:** Trabalho individual ou em grupo de no máximo 2 (dois) alunos a ser desenvolvido através de codificação na linguagem C.
- **Data de entrega:** 23/11/18
- **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 os mecanismos comunicação entre processos providos pela API de Sockets, conforme discutidos em aula, implemente uma aplicação distribuída de contagem de palavras em arquivos através do 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].

Esse 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)
```

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". 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 do modelo MapReduce [1]. 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). Há várias entidades são executadas em processos únicos. Desconsidere a operação de *fork*. As entidades trabalhadoras (*workers*) são processos responsáveis pelas tarefas de *map* e/ou *reduce*. A entidade especial é a mestre (*master*), que é o processo responsável por alocar aos trabalhadores as tarefas de *map* ou *reduce*, enviando-os os respectivos dados para serem processados por eles. Note que a aplicação (*use program*) e o mestre podem uma única entidade, nesse caso, um mesmo processo.

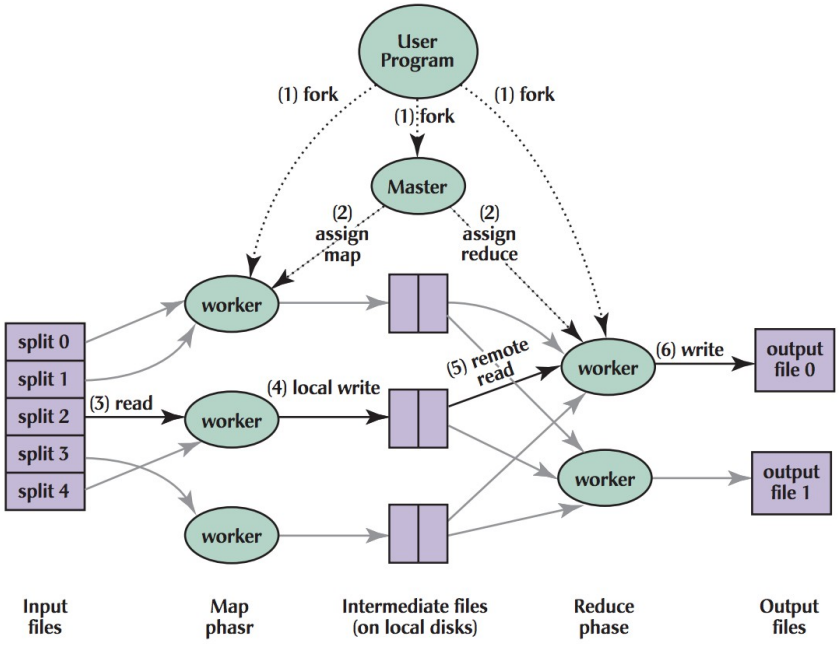

Figura 1: Visão de execução de uma implementação baseada em MapReduce. Fonte: [1]

Pares intermediários de <chave, valor> gerados pelos processos trabalhadores na tarefa de *map* são buferizados em memória local dos processos. 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 buffer. Periodicamente, esse buffer deve ser transmitido para o processo mestre.

O processo mestre então particionará esses dados recebidos (os pares intermediários de <chave, valor>) através de uma função de partição, e transmitirá as partições para um ou mais trabalhadores realizarem o *reduce*. Um processo trabalhador alocado na tarefa de *reduce* então recebe uma ou mais partições e ordena os pares pelo valor das chaves. Ao ordenar os pares, todas as ocorrências (valor) de uma mesma chave serão agrupadas, o que facilita a operação de redução. Finalmente, os trabalhadores em *reduce* iteram sobre os pares ordenados para reduzi-los. A saída de uma função *reduce* (dados reduzidos) é enviada ao nó mestre.

Note que, se houver mais de um processo *reducer*, chegarão ao processo mestre os blocos distintos dos dados reduzidos (cada bloco referente a um pedaço distinto dos dados de entrada alocado ao respectivo *reducer*). Nesse caso, há opções para o mestre prosseguir, por exemplo: (1) o próprio nó mestre itera sobre os blocos distintos recebidos dos processos *reducers*, fazendo ele mesmo a redução final em um único bloco resultante; (2) ou o mestre agrupa os blocos recebidos em partições maiores e as aloca para uma ou mais processos *reducers*. Note que a opção (2) requer laço de particionamento e alocação até que os blocos resultantes das partições sejam reduzidos em único bloco final.

Nesta prática, o objetivo é distribuir o modelo Map Reduce de tal forma que as entidades (master e workers) sejam executados em processos únicos. Para tanto, implemente a comunicação/sincronização entre processos trabalhadores (clientes) e processo mestre (servidor) e através de Sockets TCP.

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