Universidade Federal do ABC
Programa de Pós-graduação em Ciência da Computação
Disciplina de Inteligência na Web e Big Data

PROVA

Patrícia Dias dos Santos
RA 23201810211
patricia.santos@ufabc.edu.br

São Paulo, 30 de Abril de 2018

Questão 1: Descreva em uma página os conceitos de programação funcional necessários para o desenvolvimento de algoritmos distribuídos para trabalhar com grande massa de dados.

Problemas que envolvem o tratamento de conjuntos de dados massivos têm como solução ideal o uso de um modelo de processamento paralelo e distribuído que se adapta a qualquer volume e grau de complexidade. O paradigma funcional permite um maior nível de abstração, tornando possível que o programador ignore aspectos de arquitetura, e dedique seu tempo para pensar sobre os resultados e não como gerá-los.

A Programação funcional é um paradigma que trata a computação como uma avaliação de funções matemáticas evitando estado ou dados mutáveis, o que torna o trabalho com threads, concorrência e paralelismo muito mais fácil. Na programação funcional *utilizamos apenas funções que podem ser separadas em módulos* de acordo com suas responsabilidades e semântica, além de podermos passá-las através de argumentos para outras funções (funções que recebem outras funções como argumento são chamadas de funções de alta ordem) e também utilizá-las em atribuições e declarações de tipo.

Um exemplo bem comum de função de alta ordem é o *map*. Comumente utilizada em linguagens funcionais, *esta função recebe como argumentos uma expressão lambda de um parâmetro e uma lista*. Pode-se entender uma expressão lambda como sendo uma função anônima. Não nomeamos ela no corpo do nosso código, apenas reconhecemos o que ela faz, a expressamos dentro de um contexto específico, como no caso do map. Mais especificamente, map é uma função que aplica uma função a uma lista de valores e retorna a lista resultante. A função transforma um elemento do tipo a em um do tipo b e, por meio desta função, uma lista de elementos do tipo a é transformada em uma lista de elementos do tipo b. Assim a map pode ser claramente paralelizada, pois tomando-se os devidos cuidados para que a ordenação final dos elementos seja mantida, a função de transformação ($a \rightarrow b$) pode ser aplicada paralelamente a cada elemento da lista de entrada [2].

Um algoritmo funcional é formado única e exclusivamente pela composição de funções. O que leva o nosso código a ser *escrito de uma maneira declarativa*, e não imperativa. Na programação funcional pura *não existem efeitos colaterais*. Todas as funções programadas devem ter o mesmo retorno a partir de um mesmo parâmetro. Isso é bom porque ganhamos em consistência. Mesmo que na programação funcional a sua função use um valor externo, sabemos que esse *valor é imutável*. Logo, temos certeza de que a *função não irá quebrar e se manterá pura*.

O **MapReduce** (MR) é um importante modelo de programação de dados em grande escala, tais como mineração de dados e simulações científicas, e fornece um modelo simples que permite que os desenvolvedores utilizem algoritmos distribuídos altamente sofisticados. Segundo [2] o MR permite ao desenvolvedor da aplicação focar em aspectos importantes do algoritmo para resolver o problema, permitindo que este ignore questões referentes à distribuição de dados, sincronização, execução paralela, tolerância a falhas e monitoramento. O MR divide uma computação em pequenas tarefas executadas em paralelo em várias máquinas e pode ser escalável facilmente para clusters de computadores de baixo custo.

O MR procura criar uma abstração que melhore a performance de programas para sistemas distribuídos, utilizando de forma implícita a filosofia do "dividir para conquistar". Para isso, o MapReduce utiliza de conceitos da programação funcional. Um típico programa de MR consiste em apenas duas funções: Map (map(chave, valor): recebe um par de entrada de chave e valor e produz um conjunto intermediário de chaves e valores) e Reduce (reduce(chave, valores): recebe uma entrada chave e um conjunto de valores relacionados àquela chave). O conjunto de dados de entrada é armazenado em uma coleção de partições em um sistema de arquivos distribuído implementado em cada nó no cluster. O programa é então injetado em uma estrutura de processamento distribuído e executado. A função Map lê um conjunto de "registros" de um arquivo de entrada, faz algumas filtragens e/ou transformações e, em seguida, gera um conjunto de registros intermediários na forma de novos pares de chave/valor. A segunda fase de um programa MR executa instâncias R do programa Reduce (onde R é tipicamente o número de nós). Todos os registros de saída da fase Map com o mesmo valor de hash são consumidos pela mesma instância Reduce, independentemente de qual instância do Map produziu os dados. Cada instância Reduce processa ou combina os registros atribuídos a ela de alguma forma e, em seguida, grava registros em um arquivo de saída (no sistema de arquivos distribuído), que faz parte da saída final da computação. O conjunto de dados de entrada existe como uma coleção de uma ou mais partições no sistema de arquivos distribuído. É o trabalho do planejador de MR decidir quantas instâncias do Map serão executadas e como alocá-las aos nós disponíveis. Um dos principais benefícios do Map Reduce é que ele lida automaticamente com falhas, ocultando a complexidade da tolerância a falhas do programador. Se um nó falhar, o MapReduce automaticamente executará novamente suas tarefas em uma máquina diferente. Da mesma forma, se um nó estiver disponível, mas com desempenho ruim, o MapReduce executará uma "tarefa de backup" em outra máquina para concluir a computação mais rapidamente [1].

Assim, conforme dito anteriormente, os conceitos de funções puras, funções de alta ordem, dados imutáveis, uso de expressões lambda, resistência a efeitos colaterais, linguagem declarativa, avaliação preguiçosa e o Map-Reduce estabelecem uma abstração que permite construir aplicações com operações simples, escondendo os detalhes da paralelização, que é o ideal em aplicações que necessitem dessas características, a exemplo do tratamento de "big data" e do processamento de algoritmos de alta complexidade e escalabilidade.

Referências

[1] Ciprian Dobre and Fatos Xhafa. Parallel programming paradigms and frameworks in big data era. *International Journal of Parallel Programming*, 42(5):710–738, 2014.

[2] Sabir Ribas, Mário Henrique de Paiva Perché, Igor Machado Coelho, Paulo Luiz Araújo Munhoz, Marcone Jamilson Freitas Souza, and André Luiz Lins de Aquino. Mapi: um framework para paralelização de algorítimos. 2010.