Exercício Programa 2 - Fase II
MAC0431 - Introdução à Computação Paralela e Distribuida

Fernando Omar Aluani 6797226
Jefferson Serafim Ascaneo 6431284

1 Descrição do Problema

Hoje em dia muitos desenvolvedores usam sistemas de controle de versão para gerenciar seus projetos, e entre esses sistemas, um que é amplamente utilizado é o *Git*. Uma das funcionalidades do Git é a possibilidade de exibir um relatório de informações de um repositório, listando os commits feitos, quem é o autor de cada commit, as alterações realizadas, e mais.

Como pode existir grande variabilidade dos métodos e locais de trabalho dos desenvolvedores, notar quais pessoas estão trabalhando mais ou menos torna-se um desafio. No entanto, o Git pode nos ajudar fornecendo informações detalhadas sobre o repositório, possibilitando a análise do trabalho de cada desenvolvedor.

Além disso, em certas metodologias de desenvolvimento de software é importante que os commits tenham um número reduzido de modificações, permitindo a integração contínua do software sendo desenvolvido e um menor número de conflitos. Um outro ponto interessante é a análise da estabilidade do código, e um indicador para isso é a quantidade de linhas de um arquivo modificadas por commit. Um arquivo que sofre, em média, uma grande quantidade de modificações é mais propenso a erros de programação por conter código mais recente e que ainda não foi executado em um grande número de cenários. Todas estas informações podem ser extraídas dos relatórios fornecidos pelo Git.

1.1 Formato da Entrada de Dados

O relatório do Git é um arquivo texto contendo a descrição de cada commit feito no repositório de forma ordenada, do commit mais recente para o mais antigo. Tal relatório é dado pelo comando **git log**, que contém diversos parâmetros para ajustar o formato do relatório.

Para o nosso projeto, é esperado que a descrição de cada commit no relatório tenha o seguinte formato:

```
<commit hash code> <author email>
<número de linhas adicionadas>      <número de linhas removidas>      <caminho/nome do arquivo modificado>
<número de linhas adicionadas>      <número de linhas removidas>      <caminho/nome do arquivo modificado>
<número de linhas adicionadas>      <número de linhas removidas>      <caminho/nome do arquivo modificado>
...
```

Para tal, o seguinte comando é usado:

```
git log --numstat --pretty=format:"%H %ae"
```

Dessa entrada, extraímos as seguintes estatísticas:

- Número de commits por desenvolvedor;

- Média, desvio padrão e total de linhas modificadas de cada desenvolvedor;
- Média, desvio padrão e total de linhas modificadas por commit;
- Média, desvio padrão e total de linhas modificadas de cada arquivo.

2 Implementação

2.1 Map

O Map basicamente coleta as amostras de dados do arquivo de entrada. Ele recebe um par $(chave, valor)$ de *Text* (classe do Hadoop representando um texto, como uma *string*), e coleta pares $(chave, valor)$ onde a chave é uma string com o nome do dado, e valor é um *double* com o seu valor.

A chave do par recebido é o hash code do commit, enquanto o valor é o próprio texto do commit, separado do arquivo de entrada. O Map então processa esse texto do commit, lendo linha por linha do texto e analisando cada linha para extrair os dados que vamos usar, que são:

- Número de linhas modificadas (adicionadas + removidas) para cada arquivo;
- Número de commits feitos pelo autor;
- Número de linhas modificadas no commit;
- Número de linhas modificadas pelo autor;

2.2 Reduce

O Reduce pega os pares de dados coletados pelo Map e calcula as estatísticas (soma, média e desvio padrão) para cada conjunto de pares com a mesma chave. As estatísticas são coletadas e compõem a saída do programa, adicionando o tipo da estatística como prefixo da chave a qual ela se refere.

O algoritmo para calcular as estatísticas é um simples algoritmo para cálculo de soma, média e desvio padrão de uma lista de valores.

Há alguns casos onde o resultado do desvio padrão é *NaN* (not-a-number), devido a como ele é calculado. Nesses casos, o valor do desvio padrão é setado como 0.

2.3 InputFormat

O padrão do Hadoop para leitura do arquivo de entrada e distribuição dos dados dele para os vários nós de processamento é ler linha-por-linha, e distribuir as linhas. Isto não funcionaria no nosso programa, pois precisamos analisar a entrada de forma *commit-por-commit*, e cada commit é composto por várias linhas de texto do arquivo de entrada.

Portanto, derivamos a classe padrão do Hadoop que gerencia a leitura e distribuição dos dados do arquivo de entrada, *FileInputFormat*, e criamos a nossa *CommitInputFormat* que separa o arquivo de entrada nos blocos relativos à cada commit, distribuindo tais blocos para o Map.

2.4 RecordReader

O *InputFormat* usa internamente uma classe do Hadoop chamada *RecordReader* para a leitura do arquivo de entrada e divisão em blocos. Para criar nosso *CommitInputFormat* também tivemos que implementar um *CommitRecordReader*.

Ele lê linha-por-linha do arquivo de entrada, separando os blocos quando encontra a linha que indica um novo commit.

3 Conclusão

Ao implementar a solução em Hadoop, nos deparamos com os seguintes pontos:

3.1 Pontos Positivos

- Fácil implementar seu algoritmo;

3.2 Pontos Negativos

- Complicado para entender a princípio;
- Difícil testar/*debuggar* seu programa;
- Saída do programa é problemática, gerando uma pasta com alguns arquivos estranhos e dando erros se a pasta já existe;