**Executando o mapeador e o redutor localmente**

**Objetivo: obter feedback rapidamente sobre o código do mapeador e do redutor sem sobrecarregar o Hadoop.**

Sumário

Organização

Execução do código no Hadoop

Há uma esperança: a alternativa

Execução do código localmente

Conclusão

***Sumário***

Este tutorial mostra como testar o código mapeador e redutor localmente, em um terminal Unix, sem iniciar uma tarefa do Hadoop. Ao evitar a sobrecarga de inicialização do Hadoop, podemos obter rapidamente feedback útil sobre a precisão de nosso código mapeador e redutor. Também apresentamos alguns recursos úteis do terminal para trabalhar com grandes arquivos de texto.

***Organização***

Digamos que recebemos o arquivo purcharses.txt e queremos calcular qual o montante foi pago por meio de cada um dos métodos de pagamento (por exemplo, Amex, Visa, dinheiro, etc.). A configuração padrão na pasta udacity\_training é a seguinte:

code/

*sample code*

data/

purchases.txt

access\_log

*Execução do código no Hadoop*

A fim de manter tudo organizado, vamos criar uma nova pasta chamada *tutorial*.

[training@localhost udacity\_training]$ mkdir tutorial

Colocaremos os arquivos mapper.py e reducer.py nessa pasta. Agora, vamos executar diretamente a tarefa do Hadoop, a partir da pasta tutorial, conforme descrito a seguir:

Neste ponto, já colocamos o arquivo purchases.txt no HDFS, em um diretório chamado *input\_1*.

[training@localhost tutorial]$ hs mapper.py reducer.py

input\_1 tutorial\_out

Enquanto o Hadoop é prolixo durante o progresso da tarefa, a maioria dos logs se refere à estrutura de mapeamento/redução e não fornece um feedback útil sobre a precisão de nosso código.  
 Infelizmente, neste caso, nossa tarefa falha:

13/11/11 19:13:28 ERROR streaming.StreamJob: Job not

successful. Error: NA

13/11/11 19:13:28 INFO streaming.StreamJob: killJob...

Streaming Command Failed!

***Há uma esperança: a alternativa***

Após uma hora olhando nosso código, decidimos testá-lo em um terminal. Primeiro, precisamos observar nossos dados. O comando *head* permite reproduzir as primeiras linhas de um arquivo de texto. Esse comando é especialmente útil com arquivos grandes, quando queremos dar uma olhada em algumas linhas do arquivo, sem reproduzir todo o conteúdo.  
 Isso é usado conforme descrito a seguir (a partir da pasta *data*):

[training@localhost data]$ head -n 5 purchases.txt

2012-01-01 09:00 San Jose Men's Clothing

214.05 Amex

2012-01-01 09:00 Fort Worth Women's Clothing

153.57 Visa

2012-01-01 09:00 San Diego Music 66.08 Cash

2012-01-01 09:00 Pittsburgh Pet Supplies

493.51 Discover

2012-01-01 09:00 Omaha Children's Clothing

235.63 MasterCard

Usamos o recurso *head* com uma opção -n 5, que reproduz as primeiras cinco linhas do arquivo. Agora, podemos pegar uma pequena amostra do arquivo grande purchases.txt e usá-la para executar um teste local. O comando para fazer isso é:

[training@localhost data]$ head -n 1000 purchases.txt

>> ../tutorial/test\_in.txt

Estamos acessando o arquivo purchase.txt no diretório de dados: o recurso *head* pegará as primeiras 1.000 linhas do arquivo. O comando >> pegará a saída a partir do *head* e simultaneamente criará um arquivo test\_in.txt (caso não exista um na pasta tutorial) e gravará nesse arquivo essas 1.000 linhas. Observe que acessamos o arquivo original no diretório de dados, mas criamos o arquivo novo na pasta tutorial, que está fora do diretório de dados. Para que os comandos funcionem de forma adequada, a pasta de treinamento udacity deve ser organizada conforme segue:

code/

*sample code*

data/ (aqui é onde acessamos o arquivo de dados)

purchases.txt

access\_log

tutorial/ (aqui é onde criamos o novos arquivo:

test\_in.txt)

mapper.py

reducer.py

test\_in.txt

***Execução do código localmente***

Agora estamos prontos para executar o teste localmente. O comando para fazer isso a partir da pasta tutorial é:

[training@localhost tutorial]$ cat test\_in.txt | python

mapper.py | sort | python reducer.py

Vamos dividir isso em duas partes. Certamente, um comando que envolva um comando cat e um python será útil!

Usamos o comando *cat* para acessar todo o conteúdo do test\_in.txt. O comando | diz para coletar a saída padrão do comando à esquerda (*cat*) e o insere como entrada padrão no comando à direita. Dessa forma, o script do mapeador recebe como entrada padrão o conteúdo do arquivo e pode o ler linha por linha.

Então, passamos a saída do mapeador (linhas dos pares de valor de chave) para o recurso *sort*. Esse recurso classificará a saída de forma que todas as linhas com a mesma chave sejam agrupadas. Observe que, ao executar uma tarefa do Hadoop, não precisamos nos preocupar em classificar os resultados, uma vez que o Hadoop faz isso (durante a fase de classificação *shuffle*). Aqui, precisamos inclui-lo, pois estamos testando nosso mapeador e redutor localmente.

Então, inseriremos os pares de valor de chave no redutor. Ele reproduz (como saída padrão, no terminal) a saída reduzida final. Como alternativa, podemos salvá-lo em arquivo ao incluir o comando >> test\_out.txt no final.

O resultado da execução do comando completo do mapeador e redutor será:

File "reducer.py", line 29

salesTotal+ = float(thisSale)

^

SyntaxError: invalid syntax

Finalmente um feedback útil! E ficamos sabendo que o problema é apenas um erro de sintaxe. Podemos corrigi-lo e executar o teste localmente mais uma vez.

[training@localhost tutorial]$ cat test\_in.txt | python

mapper.py | sort | python reducer.py

Amex 43138.99

Cash 49065.39

Discover 52008.03

MasterCard 47020.69

Visa 52353.4

Excelente! Obtemos alguns resultados razoáveis com base em nossa pequena amostra de 1.000 registros. Agora, podemos executar a tarefa no Hadoop com o conjunto completo de dados com mais confiança.

***Conclusão***

O presente tutorial apresentou uma forma de testar o código mapeador e redutor localmente em um terminal Unix. O erro encontrado durante a execução local foi um simples erro de sintaxe. No entanto, ao estudar cuidadosamente o resultado final produzido localmente, podemos identificar erros complicados na lógica do nosso código, antes mesmo de executar uma tarefa do Hadoop.