

Transformação Data Warehouse para MapReduce no projeto ProInfoData

Tiago Rodrigo Kepe e Felipe Bolsi

27 de junho de 2011

1 Introdução

O projeto ProInfoData tem como objetivo monitorar os computadores de todas as escolas públicas do Brasil. O monitoramento visa disponibilizar dados para que o MEC e a sociedade acompanhem o estado de funcionamento dos computadores. Inicialmente este parque computacional foi estimado em 500.000 computadores, atualmente, essa estimativa aumentou para mais de 1.000.000 de máquinas, a tendência é que continue crescendo com o tempo.

Para atender esta demanda o sistema foi estruturado da seguinte forma: todo computador de escola pública brasileira terá um agente (cliente) que diariamente envia informações de uso e de hardware para o servidor central, além das informações do uso de rede que foram incluídas recentemente. O servidor tem duas camadas: o WebService que recebe informações dos agentes e as armazena no Banco de Dados (BD). Essa arquitetura pode ser visualizada a seguir:

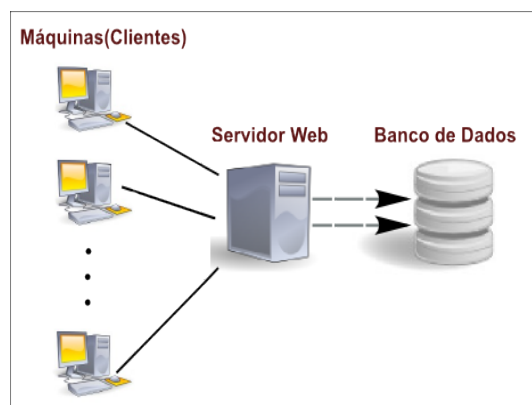


Figura 1: Visão geral da Arquitetura do ProInfoData

Como Sistema de Gerenciamento de Banco de Dados foi escolhido PostgreSQL, um dos motivos dessa escolha é por que o PostgreSQL é um software livre, entretanto, para comportar o volume de dados gerado foi necessário desenvolver uma arquitetura de armazenamento robusta e escalável. A arquitetura proposta foi baseada em armazém de dados Data Warehouse (DW) que é direcionada às operações de leitura, favorecendo a

análise de grandes volumes de dados e a geração de relatórios complexos, entraremos em mais detalhes no capítulo 2.

No entanto, com o aumento emergente do número de máquinas e com o acréscimo das informações do uso de rede, o volume de dados tornou-se extremamente grande, fato que nos levou a propor uma nova solução para o armazenamento dos dados e para realização das consultas disponíveis no portal do projeto.

Nossa proposta baseia-se em uma tecnologia emergente, chamada MapReduce. Essa solução mostrou-se eficiente em diversas implementações, a mais conhecida é o sistema de armazenamento e busca do Google, ver [1] ou artigo MapReduce Google.

Acreditamos que o grande desafio dessa solução será a transformação do modelo baseado em Data Warehouse para um modelo MapReduce.

No capítulo 2 descrevemos com mais detalhes o conceito de Data Warehouse e a arquitetura do projeto ProInfoData. No capítulo 3 descrevemos o conceito de MapReduce, as tecnologias que serão utilizadas Hive e Hadoop e porque utilizar MapReduce no ProInfoData. No capítulo 4 deparamos com nosso grande desafio em propor um método de transformação de Data Warehouse para MapReduce. Enfim, no capítulo 5 chegamos às conclusões dessa monografia.

2 Data Warehouse

Segundo Silberschatz e Korth [4] “As consultas ao banco de dados normalmente são projetadas para extrair informações específicas, como saldo de uma conta ou a soma dos saldos de conta de um cliente. Porém, consultas projetadas para ajudar a formular uma estratégia corporativa normalmente exigem a agregação em uma escala muito maior e incluem análise estatística não expressa facilmente com os recursos da SQL que já vimos anteriormente. Essas consultas normalmente precisam acessar dados vindos de várias origens.”

Um depósito de dados (data warehouse - DW) é um repositório de dados oriundos de várias fontes e armazenados sob um banco de dados comum, unificado, e que, normalmente, será mantido por um longo período de tempo, permitindo acesso a dados históricos. Diferente dos bancos de dados tradicionais que são transacionais, os DW têm a característica distinta direcionada principalmente ao suporte para tomada de decisões, segundo Navathe [3]. Os dados armazenados no depósito são submetidos a análises e agregações complexas, por isso, o data warehouse é um banco de dados direcionado a consultas e análise de dados. Também é possível utilizar técnicas para descobrir regras e padrões a partir dos dados, facilitando, assim, a tomada de decisão. As informações são de alto nível obtidas a partir de dados detalhados armazenados no depósito.

A modelagem de dados dos DW é baseada em modelos multidimensionais que aproveitam informações dos dados para visualização em estruturas denominadas cubos, os cubos também são conhecidos como matrizes multidimensionais, na representação em matrizes o desempenho pode ser muito melhor do que em modelos relacionais, além de facilitar no processamento analítico on-line (OLAP – online analytical processing) e na visualização dos dados.

Nesse modelo multidimensional são definidas tabelas de dimensão e tabelas fatos, as tabelas de dimensão armazenam dados não voláteis, isto é, dados que não são alterados frequentemente com o tempo, já as tabelas fatos armazenam dados voláteis que são alterados constantemente e estão relacionadas as tabelas de dimensão.

Nos DW os dados são extraídos de vários bancos de dados e podem estar em esquemas diferentes. É parte da tarefa do Data Warehouse agrupar toda a informação em um único esquema.

Existem diferentes formas de manter o Data Warehouse atualizado. No modelo mais comum, a coleta de dados é orientada pela fonte, as fontes transmitem os novos dados. Essa transmissão de dados é estipulada para ocorrer de forma contínua para manter o repositório mais atualizado possível em relação as suas diversas fontes. Uma outra forma possível é que a coleta dos dados seja orientada pelo Data Warehouse. Nesse modelo é enviada uma requisição as fontes sempre que seja necessário atualizar a base central. Outra forma é a de reconstrução total da base do Data Warehouse periodicamente. Apesar de ser uma abordagem mais simples, ela não é eficiente, pois deve tratar de grandes quantidades de dados todas as vezes em que a reconstrução for realizada.

Vale observar que em nenhuma das abordagens a base central de dados estará sempre atualizada em relação as todas as suas fontes. Uma forma de contornar esse problema é a atualização em duas fases, onde as modificações feitas nas bases são então replicadas no Data Warehouse. Mas essa abordagem torna todo o processo muito caro em termos computacionais.

Em geral essa pequena desatualização não significa um problema para os sistemas de suporte a decisão.

2.1 Data Mining

Data mining é uma técnica que tornou-se muito popular nos últimos anos, consiste no processo de extrair informações válidas, ou melhor, permite inferir e explorar conhecimento a partir dos dados armazenados. Em conjunto com o DW compõe uma funcionalidade crucial para tomada de decisões estratégicas.

Segundo Navathe [3], “Para fazer a data mining mais eficiente, o data warehouse deve ter uma coleção de dados agregados ou sumarizados”. Nesse ponto entra o conceito de Data Mart (DM), consiste em criar dentro do DW um conjunto de dados agregados ou sumarizados com diversos objetivos, mas principalmente em facilitar a mineração de dados, focando na agregação para melhorar o desempenho das consultas mais comuns ou frequentes.

No ProInfoData foram definidos gráficos e relatórios para o MEC e a sociedade consultarem no portal do projeto, com base nessas consultas foram criados data marts específicos para atender essa demanda, como veremos a seguir.

2.2 Arquitetura do Banco de Dados no ProInfoData

No BD do ProInfoData existem três etapas essenciais que podemos chamar de grandes transações: carregamento, armazenamento e leitura de dados. O carregamento consiste em receber e consolidar os dados no DW, o armazenamento é o próprio histórico de dados do DW e a etapa de leitura organiza os dados para otimizar as consultas. Essas etapas são implementadas em três componentes: staging area, DW e Data Marts.

A staging area é uma tabela de armazenamento temporário, ela é responsável por receber os dados dos clientes sem nenhuma manipulação, esses dados são inseridos pelo servidor webservice, eles são armazenados temporariamente e após o carregamento da staging area, são extraídos, transformados e consolidados no DW, então são retirados da staging area. Com os dados armazenados no DW, eles, finalmente, são sumarizados e agregados no conjunto de DM que foi projetado para otimizar as consultas.

No DW existem dois tipos de tabelas de armazenamento: as tabelas de dimensão e as tabelas fatos, as principais tabelas de dimensão são: tabela de cadastro de dados das escolas, a tabela de cadastro da máquina associada a escola correspondente, a tabela de catálogo de hardware e a tabela de data. As principais tabelas fatos são: a tabela de armazenamento de disponibilidade das máquinas, tabela de inventário de hardware e tabela de uso de rede.

O conjunto de DM é formado por quatro Data Mart: o DM para classificar a disponibilidade das máquinas em verde (último contato a menos de 11 dias), amarelo (último contato entre 11 e 30 dias) e vermelho (último contato com mais de 31 dias). Outro DM agrupa por escola a disponibilidade das máquinas. Existe outro para agregar informações de hardware e outro para detectar alteração de hardware.

Talvez inserir uma imagem da arquitetura do BD????

Cada componente descrito têm sua complexidade de arquitetura, carregamento e armazenamento. Por isso, em todos esses componentes foram realizados testes de desempenho, utilizando uma metodologia baseada em um modelo incremental de hardware e software. O objetivo é avaliar o sistema partindo de um ambiente menos complexo para o mais complexo, usando cargas intermediárias até o ponto limite do sistema. Com isso, além de encontrar a carga máxima que o sistema suporta, fornece também resultados parciais que facilitam a avaliação de estresse de hardware. Os testes de escrita no BD mostraram que a arquitetura proposta chega a atender 334 transações por segundo. Já os testes de consulta alcançou o número de 142 transações por segundo. Com estes resultados, a arquitetura mostrou-se eficiente, atendendo as conexões e as consultas de dados esperadas.

Entretanto, um ponto crucial do BD não entrou nessa bateria de testes que é o carregamento dos dados para o DW e depois a agregação dos dados no conjunto de DM. O carregamento é considerado crucial porque foi determinado um período máximo para executá-lo que é de oito horas (entre 00h:00m até 8h:00m) porque nesse horário o número de consultas é relativamente menor e conseqüentemente o impacto no BD é

menor. Além disso, os dados tem um delay de um dia, ou seja, os dados no DW são sempre atualizados do dia anterior e não do dia atual.

O carregamento é realizado por funções de carregamento executadas no PostgreSQL, são procedures (store de carregamento) que realizam as junções, sumarizações e agregações necessárias. Essa etapa não entrou nos testes porque depende dos dados já inseridos no BD, como a cada dia são inseridos mais dados, o carregamento tende a demorar mais e, assim, aumenta gradualmente o tempo de execução conforme cresce o volume de dados.

Não é difícil de perceber que o carregamento um dia demorará mais de oito horas para finalizar, ocasionando impacto negativo nas consultas do BD, pois enquanto ele é carregado as tabelas são bloqueadas (lock realizado pelo SGBD) pelo PostgreSQL, esse mecanismo é implementado no kernel do SGBD para que as consultas não retornem informações desatualizadas.

Por isso, torna-se necessário usar novas tecnologias para resolver esse tipo problema.

3 MapReduce

MapReduce é um framework criado para permite que a tarefa de analisar uma grande quantidade de dados possa ser dividida e designada a um conjunto de máquinas, denominando cluster de computadores, afim de tirar proveito da performance obtida pelo paralelismo, omitindo-se toda a complexidade para tornar a execução em paralelo possível, de modo que o usuário foque no problema principal, que é o processamento dos dados.

O modelo MapReduce é composto por duas fases, a de mapeamento dos dados e redução. Para execução dessas fases o framework designa uma das máquinas do cluster como master; essa máquina então define um conjunto de máquinas para desempenhar a função de mapeamento e um outro conjunto para executar a tarefa de redução. Na primeira fase a máquina master tem a função de dividir os dados de entrada em várias partes menores e então designar cada parte a uma máquina que estará desempenhando a atividade de mapeamento. Após o término dessa fase inicia-se a fase de redução. Nessa segunda fase a máquina master notificará as máquinas que desempenham a atividade de redução sobre a localização dos dados produzidos pela fase anterior, para que os dados sejam condensados em informações úteis para que possam ser interpretados e utilizados para o fim necessário.

Cada uma dessas fases, a de mapeamento e redução, implementa uma função, Map e Reduce respectivamente, ambas funções são implementadas pelo usuário.

A função Map recebe como entrada um par chave-valor. Com o par serão realizadas as operações definidas (operações de Map e Reduce), e será produzido como saída uma lista de pares chave-valor intermediárias. No término dessa fase, isto é, cada função Map designada terminou sua execução, será então obtida a lista completa de chave-valor intermediárias. Com a conclusão dessa fase o framework agrupará todos os

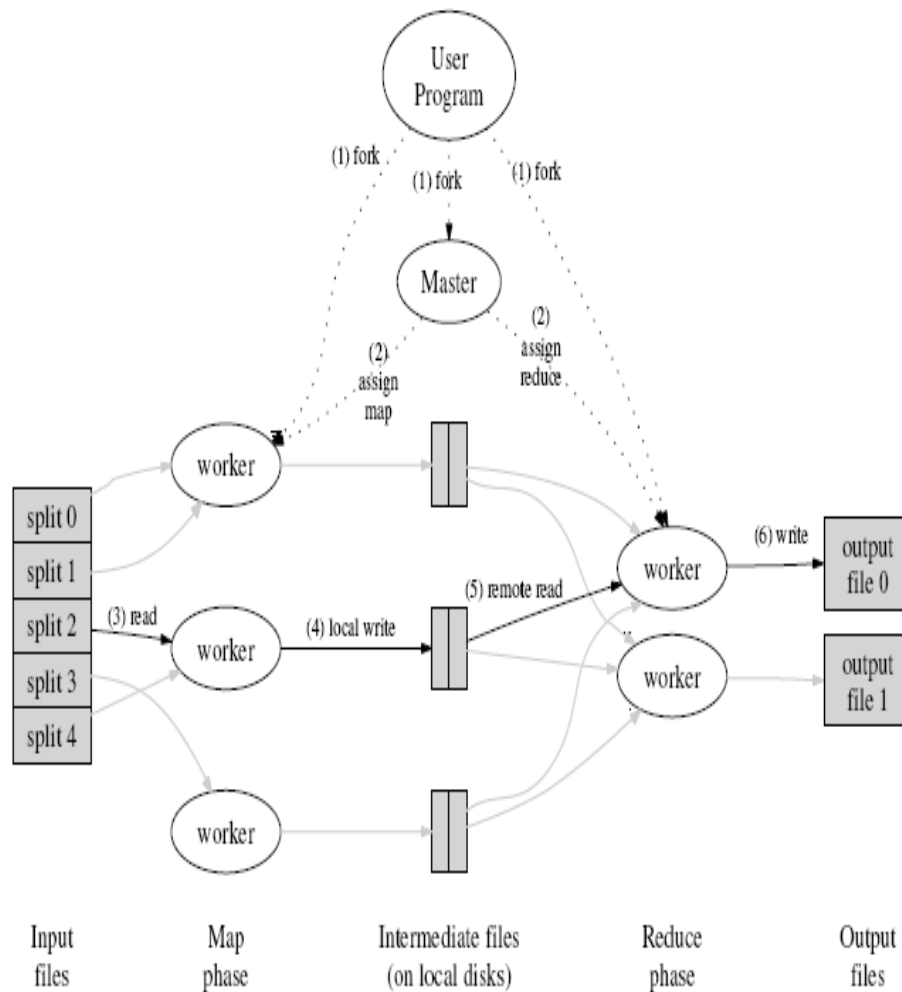


Figura 2: Visão geral da execução do framework MapReduce
 Fonte: MapReduce: Simplified Data Processing on Large Clusters (2004, p. 3)

valores associados a mesma chave, partindo para a próxima fase, que utiliza a função Reduce.

A função Reduce recebe como entrada o par chave-lista de valores associada a chave. O objetivo dessa função é realizar tarefas para agrupar ainda mais os valores associados a mesma chave, gerando, normalmente, uma única saída.

Algumas variações do modelo são possíveis para que ele se adapte melhor as características dos dados analisados e operações realizadas pelas funções Map e Reduce, visando uma melhor performance na utilização do modelo.

3.1 Hadoop

Hadoop é um framework que permite o processamento de dados em larga escala em clusters de computadores. Oferece um mecanismo de distribuição dos dados em um Sistema de Arquivo Distribuído (Hadoop Distributed File System - HDFS). Também oferece uma interface para im-

plementar as funções de Map e Reduce o que facilita a programação e proporciona um sistema de armazenamento simples em conjunto com o Hive.

As premissas do MapReduce consiste na simplificação do armazenamento, comparada as estruturas de armazenamento do SGBD, o armazenamento utilizado deve ser simples e normalmente armazenar uma chave e um valor para os dados.

O Hadoop pode ser instalado em três modos: Standalone, Pseudo-Distributed e Fully-Distributed. A primeira é útil para testar a aplicação e depurar o código, e roda como um único processo Java. O modo Pseudo-Distributed, assim como no Standalone, é executado em apenas uma máquina, porém cada daemon do Hadoop roda em um processo distinto. Já o modo Fully-Distributed é utilizado em sistemas de produção, realmente distribuídos.

Preocupações com falhas de hosts no meio de processamento de tarefas são desconsideradas. Todos os problemas de distribuição ficam a cargo do framework.

3.2 Hive

Hive é uma infra-estrutura de data warehouse construída em cima do Hadoop.

Ele suporta convenientemente a análise de grandes conjuntos de dados armazenados em sistemas de arquivos compatíveis com o Hadoop, como exemplo o sistema de arquivos da Amazon S3. Ele fornece uma linguagem SQL-like chamado HiveQL mantendo total apoio para o MapReduce. Para acelerar consultas, fornece índices como o índice de bitmap.

Atualmente, existem três formatos de arquivos suportados no Hive, que são textfile, SEQUENCEFILE e RCFILE.

4 Proposta de transformação de Data Warehouse em um modelo MapReduce

Uma abordagem possível para nossa solução seria baseada em ATL (model transformation technology), consiste em realizar transformações usando modelos, essa abordagem facilita na automatização do processo de transformação, além de ter suporte de ferramentas e APIs.

Para realizar transformações entre modelos é necessário criar correspondência entre elementos de um modelo com elementos do outro modelo, segundo [2]. Essa correspondência é realizada através de regras de associação. Em nosso ambiente é necessário associar as tabelas, atributos, associações, funções de carregamento e demais componentes da arquitetura do BD do ProInfoData, a elementos correspondentes no modelo do Hadoop, Hive e MapReduce.

No BD do ProInfoData existem tabelas de dimensão e tabelas fatos, inicialmente temos que criar regras de transformação para associar as tabelas essências do DW ao modelo de armazenamento do hadoop.

Um dos formatos que o hadoop suporta é o arquivo de texto, transformaremos as tabelas fatos e tabelas de dimensão em um arquivo, onde um

tupla corresponde a uma linha e cada atributo das tabelas corresponderia a um campo do arquivo. Uma possível dificuldade seria representar as associações entre as tabelas fatos e as tabelas de dimensão, no DW essa associação ocorre com chaves primárias e estrangeiras. Para resolver esse problema teremos que inserir no arquivo todos os atributos das tabelas associadas, esta abordagem pode impactar diretamente no tamanho do arquivo, mas esse problema não é preocupando, uma vez que o MapReduce foi projetado para suportar grande volume de dados, na faixa de terabytes e até pentabytes.

Então uma ou mais tabelas corresponderiam a um arquivo de texto, suas linhas a concatenação das tuplas associadas às tabelas e os campos do arquivo corresponderiam a atributos das tabelas.

Um exemplo seria a transformação das tabelas: mectb01-escola-dim, mectb03-data-dim, mectb04-maquina-dim e mectb05-disponibilidade-fact, em um único arquivo chamado de “arq-disponibilidade”, cada linha deste arquivo teria os campos:

MAC-ADDRESS, INEP, NOME-ESCOLA, CIDADE, ESTADO, REGIÃO, DATA-CONTATO

A staging area seria transformada em um arquivo, pois é uma tabela simples de armazenamento temporário, poderiam ser criadas rotinas de atualização dos arquivos de armazenamento permanente a partir do arquivo da staging area. Na verdade teria que ser estudado melhor a própria existência do arquivo de armazenamento temporário, pois outra solução seria inserir os novos dados vindos do servidor Webservice diretamente nos arquivos de armazenamento, com isso evitaria a manutenção do arquivo temporário. Essa abordagem seria mais consistente com o ATL, pois se houvesse algum incremento ou alteração na estrutura do arquivo temporário, essa modificação impactaria nas rotinas de atualização que precisariam ser adaptadas.

Essa última abordagem sem o armazenamento temporário eliminaria a etapa de carregamento dos dados, contudo teria que ser avaliado o desempenho de inserções nos arquivos de armazenamento, mesmo se existir o arquivo temporário também será necessário testar o desempenho de inserções nele.

O carregamento do DW tem como tarefa principalmente a classificação dos dados, ele realiza o catálogo de hardware, associa as máquinas as sua escola correspondente e também associa a disponibilidade com a data de contato, além de detectar alteração de hardware. Essa etapa corresponde às funções de Map, ou seja, o mapeamento dos dados é responsável por essas classificações, já que o objetivo inerente do mapeamento é descrever o modelo dos dados, consequentemente essa tarefa será de responsabilidade do programador quando escrever as funções de Map.

Outra etapa importante é a agregação e sumarização dos dados nos Data Marts, essa etapa é uma das mais cruciais, pois impacta diretamente no desempenho do Data Warehouse, ela corresponde as funções de Reduce, pois no Reduce podemos agregar vários Maps, ou seja, as entradas dos arquivos de armazenamento são mapeadas e repassadas ao

Reduce para manipular os resultados. Desta forma os DM não existiriam mais, o programador assume toda responsabilidade ao escrever as funções de Reduce conforme a necessidade das consultas.

As consultas são de responsabilidade do Hive que oferece uma interface para recuperação dos dados, esse é um ponto de extrema importância, pois o Hive não implementa todas as funções SQL, dependendo da consulta o programador terá que escrevê-la manualmente.

Um código protótipo de uma função MapReduce usando o arquivo “arq-disponibilidade” seria:

Code 1: Disponibilidade por região

```
1  /* Prototipo de implementacao de map e reduce
2    * Tiago Rodrigo Kepe
3    */
4
5  package br.ufpr;
6  import java.io.IOException;
7  import java.util.*;
8  import java.text.SimpleDateFormat;
9
10 import org.apache.hadoop.fs.Path ;
11 import org.apache.hadoop.io.*;
12 import org.apache.hadoop.mapred.*;
13
14 public class Availability {
15     private Date currentDate = new Date();
16     private String strDate;
17     private SimpleDateFormat formatador = new
18         SimpleDateFormat("dd/MM/yyyy");
19
20     /* Pega data corrente e atribui a uma string */
21     strDate = formatador.format(currentDate);
22
23     public static class Map extends MapReduceBase
24         implements Mapper<LongWritable , Text , LongWritable ,
25             Text> {
26
27         private LongWritable k = new LongWritable();
28         private Text v = new Text();
29
30         /* Implementacao da funcao de mapeamento . */
31         public void map( LongWritable key , Text value ,
32             OutputCollector<LongWritable , Text> output ,
33             Reporter reporter
34         ) throws IOException {
35
36             /* Separa os campos da linha em um array. */
37             String[] fields = value.toString().split("\\s");
38
39             /* Pega o MAC da maquina do primeiro campo da
40                 linha. */
41             Long machine = new Long ( fields[1]);
42
43             /* Verifica se o campo data-contato e data atual
44                 */
45             if ( fields[6].equals(strDate)) {
46
47                 /* Emite um par chave/valor intermediario ,
```

```

44         contendo a data de
45         contato(chave) e regioao como string(valor)
46         . */
47         k.set(strData);
48
49         /* regioao */
50         v.set(fields[5]);
51
52         output.collect(k,v);
53     }
54 }
55
56 public static class Reduce extends MapReduceBase
57 implements Reducer<LongWritable, Text, LongWritable,
58 Text> {
59
60     /* Implementacao da funcao de reducao. */
61     public void reduce (LongWritable key, Iterator <Text
62 > values,
63     OutputCollector <LongWritable, Text> output,
64     Reporter reporter
65 ) throws IOException {
66
67         count = new Long(0);
68         sumSul = new Long(0);
69         sumNorte = new Long(0);
70         sumSudeste = new Long(0);
71         sumCentroOeste = new Long(0);
72         sumNordeste = new Long(0);
73
74         /* Percorre os registro do dia atual e classifica
75         o numero de
76         contatos por regioao */
77         while (values.hasNext()) {
78
79             String region = values.next().toString();
80             /* Pega o regioao e data. */
81
82             if (region.equals("Sul"))
83                 sumSul++;
84
85             if (region.equals("Norte"))
86                 sumNorte++;
87
88             if (region.equals("Sudeste"))
89                 sumSudeste++;
90
91             if (region.equals("Centro Oeste"))
92                 sumCentroOeste++;
93
94             if (region.equals("Nordeste"))
95                 sumNordeste++;
96
97         }
98
99         String sul = new String();
100        String norte = new String();
101        String sudeste = new String();
102        String centroOeste = new String();

```

```

98         String nordeste = new String();
99
100        sul+="Sul";
101        sul+= String.format("% 15d", sumSul);
102
103        norte+="Norte";
104        norte+= String.format("% 15d", sumNorte);
105
106        sudeste+="Sudeste";
107        sudeste+= String.format("% 15d", sumSudeste);
108
109        centroOeste+="Centro Oeste";
110        centroOeste+= String.format("% 15d",
111                                   sumCentroOeste) ;
112
113        nordeste+="Nordeste";
114        nordeste+= String.format("% 15d", sumNordeste) ;
115
116        /* Emite o Data atual(chave), e total de
117           disponibilidade para cada
118           regioao */
119        output.collect(key, new Text(sul)) ;
120        output.collect(key, new Text(norte)) ;
121        output.collect(key, new Text(sudeste)) ;
122        output.collect(key, new Text(centroOeste)) ;
123        output.collect(key, new Text(nordeste)) ;
124    }
125}
126
127public static void main (String [] args) throws Exception
128{
129    JobConf conf = new JobConf (Availability.class) ;
130    conf.setJobName("Availability") ;
131
132    /* Define a quantidade de tarefas de reducao. */
133    conf.setNumReduceTasks(1) ;
134
135    /* Define o tipo dos pares chave/valor de saida. */
136    conf.setOutputKeyClass(LongWritable.class);
137    conf.setOutputValueClass(Text.class);
138
139    /* Indica as classes que implementam o mapeamento e
140       reducao. */
141    conf.setMapperClass(Map.class);
142    conf.setReducerClass(Reduce.class);
143
144    /* Indica o tipo dos arquivos de entrada e saida. */
145    conf.setInputFormat(TextInputFormat.class);
146    conf.setOutputFormat(TextOutputFormat.class);
147
148    /* Le da linha de comando a origem e destino dos
149       arquivos no HDFS. */
150    FileInputFormat.setInputPaths(conf, new Path(args[0])
151                                   );
152    FileOutputFormat.setOutputPath(conf, new Path (args
153                                                    [1]));
154
155    JobClient.runJob(conf);
156}

```

Este código implementa uma função Map e uma função Reduce. O Map recebe com entrada o arq-disponibilidade que contem os campos: **Mac-Address, Inep(identificador da escola, nome da escola, cidade, estado, região, data de contato**, gera uma saída intermediária para o Reduce com os campo data(chave) e região(valor). O Reduce, por sua vez, agrega a disponibilidade por regiões do Brasil, e gera como saída o par data(chave) e região seguida pelo quantidade de máquinas(valor).

Nota-se na linha 138 foi definida explicitamente a quantidade de Reduce, nesse caso somente uma máquina realizará todo o processo, pois foi implementado para um ambiente de teste inicial.

5 Conclusão

Pode-se notar que a implementação das funções Map e Reduce é bastante simples, permitindo que o programador se atenha apenas a geração dos dados, sem preocupar-se com a comunicação entre nós.

A transformação mostrou-se simplória, pois é necessário apenas associar elementos dos dois modelos, no nosso caso ocorrerão as seguintes transformações: tabelas serão mapeadas em arquivos, tuplas corresponderão a linha dos arquivos, os atributos a campos dos arquivos, as funções de carregamento corresponderão as funções Map, as funções de agregação (carregamento dos DMs) funções Reduce e as consultas serão realizadas pelo Hive.

Um ponto importante ainda a ser estudado, são as consultas feitas pelo Hive, mas essa parte focaremos na continuação desse trabalho.

Referências

- [1] Dean, J. and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8(6):679 –698, nov. 1986.
- [2] Marcos D. D. Fabro. *Metadata management using model weaving and model transformations*. PhD thesis, University of Nantes, September 2007.
- [3] S. B. Navathe and Elmasri R. *Sistema de Banco de Dados. 4ª ed.* São Paulo: Person Addison Wesley, 2005.
- [4] Silberschatz, A. and Korth, H. and Sudarshan, S. *Sistema de Banco de Dados. 2ª ed.* São Paulo: Makron Books, 1999.