Transformação Data Warehouse para MapReduce no projeto ProInfoData

Tiago Rodrigo Kepe e Felipe Bolsi

28 de junho de 2011

1 Introdução

O projeto ProInfoData tem como objetivo monitorar diariamente 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 como tempo.

Para atender esta demanda de máquinas 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 na fugura 1 seguir:

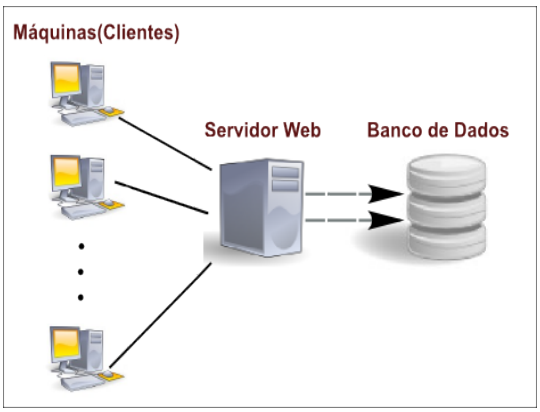

Figura 1: Visão geral da Arquitetura do ProInfoData

Como Sistema de Gerenciamento de Banco de Dados (SGBD) foi escolhido um SGBD relacional, um dos motivos dessa escolha é porque o modelo relacional facilita na associação dos dados. 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 um nova solução para o armazenamento dos dados e para realização das consultas disponíveis no [1].

A solução que proposmos 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 [2].

Acreditamos que o grande desafio dessa solução será a transformação do modelo relacional 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 utlizar 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 as conclusões dessa monografia.

2 Data Warehouse

Segundo Silberschatz e Korth [9] "As consultas ao banco de dados normalmente são projetadas para extrair informações específicas, como saldo de um conta ou a soma dos saldos de conta de um cliente. Porém, consultas projetadas para ajudar a formular um estratégia corporativa normalmente exigem a agregação em um escala muito maior e incluem análise estatística não expressa facilmente com os recursos da SQL que já vimos anteriormente. Essas cosultas normalmente precisam acessar dados vindos de várias origens."

Um DW é um repositório de dados oriundos de várias origens e armazenados sob um banco de dados comum e que, normalmente, será mantido por um longo período de tempo, permitindo acesso a dados históricos. Diferente dos bancos de dados transacionais, o DW têm a característica distinta direcionada principalmente ao suporte para tomada de decisões, segundo Navathe [7]. Os dados armazenados no DW são submetidos a análises e agregações complexas, por isso, ele 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 nele.

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

[1] portal do projeto

[2] http://labs.google.com/papers/mapreduce.html

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.

No DW os dados são extraídos de vários bancos de dados e podem estar em esquemas diferentes. É parte da tarefa dele agrupar toda a informação em um único esquema.

Existem diferentes formas de manter um DW atualizado. No modelo mais comum, a coleta de dados é orientada pela fonte, as fontes transmitem os novos dados. Segundo Ramakrishnan [8] 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 DW. Nesse modelo é enviada uma requisição as fontes sempre que seja necessário atualizar a base central. Ramakrishnan [8] menciona que outra forma é a de reconstrução total da base do DW 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 DW. Mas essa abordagem torna todo o processo muito caro em termos computacionais, segundo Ramakrishnan [8].

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

2.1 Data Mart

Data Mart (DM) é um subconjunto de dados do DW. O conjunto de DM agrupa dados específicos de um determinado assunto ou sumariza os dados para uma determinada finalidade, ou seja, 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.

Segundo Navathe [7], para fazer a análise de dados mais eficiente, o DW deve ter uma coleção de dados agregados ou sumarizados.

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

2.2 Arquiterura 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 conjundo de DM que foi projetado para otimizar as consultas.

O DW do ProInfoData é composto por tabelas de dimensão que armazenam dados com pouca atualização, foram criadas para armazenar informações das escolas, máquinas e catálogo de hardware. Também é composto por tabelas fatos que são atualizadas diariamente, essas tabelas armazenam informações de disponibilidade, inventário de hardware e consumo de banda de rede.

O conjunto de DM é formado por quatro Data Marts: um DM para classificar a disponibilidade das máquinas por cores. Outro DM agrupa por escola a disponibilidade das máquinas. Existe outro para agregar informações de hardware e um para detectar alteração de hardware.

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 consequentemente 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 SGBD, 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.

O desempenho das funções de carregamento depende da quantidade de dados armazenados no BD, o volume de dados cresce diariamente, por isso, não é difícil de perceber que o carregamento chegará ao ponto de demorar mais de oito horas para finalizar, ocasionando maior demora na realização de consultas no BD.

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

3 MapReduce

MapReduce é um framework criado para analisar uma grande quantidade de dados, os dados são divididos e designados a um conjunto de máquinas, denominado cluster de computadores, afim de tirar proveito da performance obtida pelo paralelismo, omitindo 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. Podemos ver a execução na figura 2.

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. Com a conclusão dessa fase o framework agrupará todos os 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.

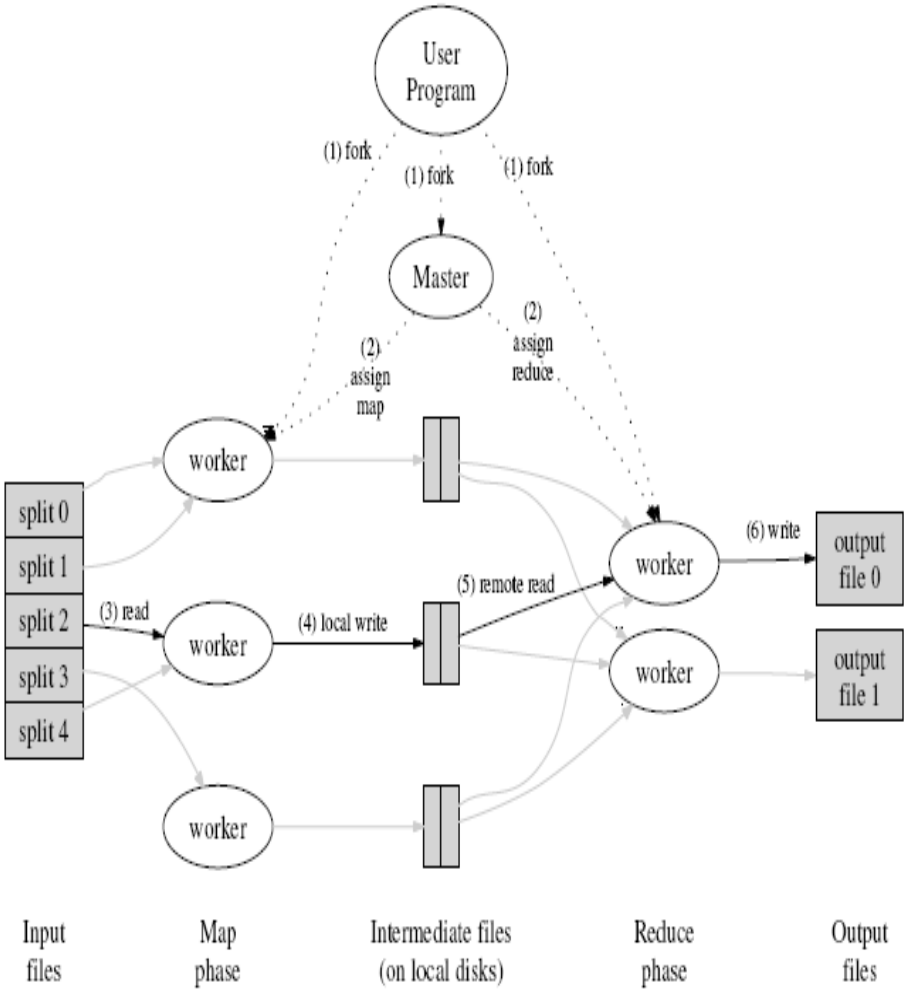

Figura 2: Visão geral da execução do framework MapReduce
Fonte: MapReduce: Simplified Data Processing on Large Clusters (2004, p. 3)

3.1 Hadoop

Hadoop [4] 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) ver [5]. Também oferece uma interface para implementar as funções de Map e Reduce o que facilita a programação.

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, real-

mente 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 [6] 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 [3]. Ele fornece uma linguagem SQL-like chamado HiveQL mantendo total apoio para o MapReduce. Para acelerar consultas, fornece índices como o índice de bitmap. Um exemplo de arquitetura com o Hive é do facebook conforme a figura 3.

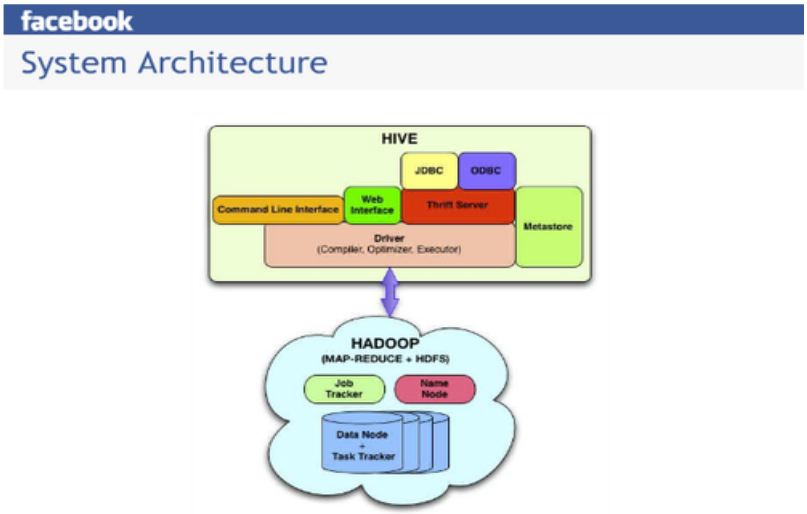

Figura 3: Facebook Data Infrastructure
Fonte: http://nosql.mypopescu.com/post/681603154/presentation-hive-a-petabyte-scale-data-warehouse

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 mape-

amento 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
           SimpleDateFormat("dd/MM/yyyy");
18
19     /* Pega data currente e atribui a uma string */
20     strDate = formatador.format(currentDate);
21
22     public static class Map extends MapReduceBase
23       implements Mapper<LongWritable , Text , LongWritable ,
             Text> {
24
25         private LongWritable k = new LongWritable();
26         private Text v = new Text();
27
28         /* Implementacao da funcao de mapeamento . */
29         public void map( LongWritable key , Text value,
30           OutputCollector<LongWritable, Text> output,
                Reporter reporter
31           ) throws IOException {
32
33             /* Separa os campos da linha em um array. */
34             String[] fields = value.toString().split("\\s");
35
```

```
36              /* Pega o MAC da maquina do primeiro campo da
                    linha. */
37              Long machine = new Long (fields[1]);
38
39
40              /* Verifica se o campo data-contato e data atual
                    */
41              if (fields[6].equals(strDate)) {
42
43                  /* Emite um par chave/valor intermediario,
                        contendo a data de
44                      contato(chave) e regiao como string(valor)
                            . */
45                  k.set(strData);
46
47                  /* regiao */
48                  v.set(fields[5]);
49
50                  output.collect(k,v);
51              }
52          }
53      }
54
55      public static class Reduce extends MapReduceBase
56        implements Reducer<LongWritable, Text, LongWritable,
              Text> {
57
58          /* Implementacao da funcao de reducao. */
59          public void reduce (LongWritable key , Iterator <Text
                > values,
60            Output Collector <LongWritable, Text> output,
                  Reporter reporter
61            ) throws IOException {
62
63              count = new Long(0);
64              sumSul = new Long(0);
65              sumNorte = new Long(0);
66              sumSudeste = new Long(0);
67              sumCentroOeste = new Long(0);
68              sumNordeste = new Long(0);
69
70              /* Percorre os registro do dia atual e classifica
                     o numero de
71              contatos por regiao */
72              while (values.hasNext()) {
73
74                  String region = values.next().toString();
75                  /* Pega o regiao e data. */
76
77                  if (region.equals("Sul"))
78                      sumSul++;
79
80                  if (region.equals("Norte"))
81                      sumNorte++;
82
83                  if (region.equals("Sudeste"))
84                      sumSudeste++;
85
86                  if (region.equals("Centro Oeste"))
87                      sumCentroOeste++;
```

```
88
89                  if (region.equals("Nordeste"))
90                      sumNordeste++;
91
92              }
93
94              String sul = new String();
95              String norte = new String();
96              String sudeste = new String();
97              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",
                    sumCentroOeste) ;
111
112             nordeste+="Nordeste";
113             nordeste+= String.format("% 15d", sumNordeste) ;
114
115             /* Emite o Data atual(chave), e total de
                    disponibilidade para cada
116                 regiao */
117             output.collect(key, new Text(sul)) ;
118             output.collect(key, new Text(norte)) ;
119             output.collect(key, new Text(sudeste)) ;
120             output.collect(key, new Text(centroOeste)) ;
121             output.collect(key, new Text(nordeste)) ;
122         }
123     }
124
125     public static void main (String[] args) throws Exception
            {
126         JobConf conf = new JobConf (Availability.class) ;
127         conf.setJobName("Availability") ;
128
129         /* Define a quantidade de tarefas de reducao. */
130         conf.setNumReduceTasks(1) ;
131
132         /* Define o tipo dos pares chave/valor de saida. */
133         conf.setOutputKeyClass(LongWritable.class);
134         conf.setOutputValueClass(Text.class);
135
136         /* Indica as classes que implementam o mapeamento e
                reducao. */
137         conf.setMapperClass(Map.class);
138         conf.setReducerClass(Reduce.class);
139
140         /* Indica o tipo dos arquivos de entrada e saida. */
141         conf.setInputFormat(TextInputFormat.class);
142         conf.stOutputFormat(TextOutputFormat.class);
143
```

```
144             /* Le da linha de comando a origem e destino dos
                    arquivos no HDFS. */
145             FileInputFormat.setInputPaths(conf, new Path(args[0])
                    );
146             FileOutputFormat.setOutputPath(conf, new Path (args
                    [1]));
147
148             JobClient.runJob(conf);
149         }
150 }
```

Este código implmenta 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 esplicitamente 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] hadoop.apache.org. Amazon s3. http://wiki.apache.org/hadoop/AmazonS3, 2011. Acessado em 28 de Junho de 2011.

[4] hadoop.apache.org. Apache hadoop. http://hadoop.apache.org/, 2011. Acessado em 28 de Junho de 2011.

[5] hadoop.apache.org. Hadoop distributed file system. http://hadoop.apache.org/hdfs/, 2011. Acessado em 28 de Junho de 2011.

[6] hadoop.apache.org. What is apache hive. https://cwiki.apache.org/confluence/display/Hive/Home%3bjsessionid=4B4A95CA588FF5 2011. Acessado em 28 de Junho de 2011.

[7] S. B. Navathe and Elmasri R. *Sistema de Banco de Dados. 4ª ed.* São Paulo: Person Addison Wesley, 2005.

[8] R. Ramakrishnan and Gehrke J. *Sistemas de Gerenciamento de Banco de Dados.* Ed. McGraw-Hill, 2007.

[9] Silberschatz, A. and Korth, H. and Sudarshan, S. *Sistema de Banco de Dados. 2ª ed.* São Paulo: Makron Books, 1999.