TÉCNICO LISBOA

ETL na era do Big Data

Luis Miguel Monteiro Silva

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e de Computadores

Orientadores: Prof.ª Helena Isabel de Jesus Galhardas
Dr. João Carlos Pereira Damásio

Júri

Presidente: Prof. Mário Jorge Costa Gaspar da Silva
Orientador: Prof.ª Helena Isabel de Jesus Galhardas
Vogal: Prof.ª Maribel Yasmina Santos

Julho 2016

Resumo

A quantidade de dados existente hoje em dia está a crescer de forma cada vez mais rápida, uma vez que as empresas produzem cada vez mais dados, a velocidades cada vez mais elevadas. Torna-se assim necessária a existência de ferramentas que sejam capazes de lidar com esse volume de dados crescente. A presente dissertação tem como objectivo averiguar em que condições podemos beneficiar da utilização de tecnologias de *Big Data* para o processamento de informação. Numa primeira parte dessa dissertação, serão definidos os principais conceitos relacionados com o processamento de grandes conjuntos de dados, apelidados de *Big Data*, e as tecnologias para processamento destes conjuntos. Serão apresentadas como trabalho relacionado algumas ferramentas para transformação de dados, que incluem algumas das mais utilizadas. A segunda parte da dissertação irá detalhar o trabalho implementado, a sua validação fazendo por fim uma conclusão e análise de trabalho futuro.

Palavras-chave: Big Data, ETL, MapReduce, Hadoop, Pig.

Abstract

The amount of data that exists nowadays is growing faster, since the companies produce more and more data, in increasingly higher speeds. This makes it necessary to have tools which are able to cope with this increased volume of data. This thesis aims to ascertain under what conditions we can benefit from the use of Big Data's technologies for information processing. In the first part of this dissertation, the main concepts related to the processing of large data sets, nicknamed Big Data, and technologies for processing these sets will be defined. Will be presented as work related some tools for data transformation, which include some of the most used. The second part of the thesis will detail the implemented work, validation and finally a conclusion and analysis of future work.

Keywords: Big Data, ETL, MapReduce, Hadoop, Pig.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

Vivemos num mundo em que o volume de dados está a crescer de forma exponencial e a velocidade elevada (Chen *et al.* , 2014). O aumento do número de dispositivos físicos ligados à Internet, em particular os dispositivos móveis, a larga utilização de redes sociais, e a maior necessidade de ter sistemas de suporte à decisão, potenciam o aumento do volume dos dados, a velocidades elevadas, assim como a heterogeneidade dos tipos de dados (vídeos, texto, música, etc.). A este volume, variedade e velocidade de dados dá-se o nome de *Big Data* (Chen *et al.* , 2014). Em comparação com os conjuntos de dados tradicionais, *Big Data* geralmente inclui uma grande quantidade de dados, estruturados e não estruturados, que para darem suporte a decisões em tempo real requerem normalmente uma análise detalhada de forma a extrair informação útil (Chen *et al.* , 2014).

Big Data traz um conjunto de novas oportunidades em relação à extração e transformação de dados em informação, uma vez que, associados ao termo *Big Data* começam a surgir novas tecnologias capazes de suportar um volume grande de dados e efetuar um processamento/análise de dados em tempo útil. *Big Data* não traz só novas oportunidades, também traz novos desafios, como por exemplo, a forma de organizar e gerir esses conjuntos de dados de forma eficaz. Surge assim a necessidade de possuir ferramentas que permitam processar dados de *Big Data* de forma rápida e eficaz. Novos paradigmas de computação foram desenvolvidos para lidar com estes volumes de dados, sendo o mais popular o *MapReduce* (Dean & Ghemawat, 2004). O paradigma de *MapReduce* é um modelo de programação para computação distribuída eficiente. Fornece um modelo de processamento paralelo e várias implementações associadas para processar grandes quantidades de dados. O *MapReduce* é um dos componentes do framework *Hadoop*[1], que é uma tecnologia que nasceu no seio das grandes empresas do ramo da Internet, tais como a Google e Facebook, tendo sido depois disponibilizada para a comunidade *open source* (havendo, no entanto, também disponíveis distribuições comerciais). Para cada distribuição de *Hadoop* existe uma concretização de *MapReduce*. *Hadoop* é um framework de computação que possibilita distribuir o processamento de grandes volumes de dados por *clusters* de computadores. Esta tecnologia traz grandes vantagens em termos de desempenho para o processamento de grandes volumes de dados no sentido em que o processamento em si é feito de forma muito

[1] `http://hadoop.apache.org`

mais rápida em comparação com as denominadas tecnologias convencionais, como alias se poderá comprovar mais a frente nesta dissertação.

O que se pretende efetuar com esta dissertação insere-se no contexto de processos de *Extract, Transform e Load (ETL)* que são um conjunto de técnicas e ferramentas utilizadas para transformar dados oriundos de múltiplas fontes de dados em informação útil e com significado para fins de análise de negócio. Os processos *ETL* enquadram-se em tecnologias de *Business Intelligence* (BI), que são tecnologias capazes de lidar com grandes quantidades de dados e o seu objectivo é permitir uma fácil interpretação desses dados (Rud, 2009). O processo *ETL* é utilizado sempre que queremos extrair e transformar dados de um esquema origem para um esquema destino, sendo a arquitetura de *data warehouse* um dos cenários onde a sua utilização é mais frequente. Este processo é modelado como um grafo de transformações de dados que se aplicam aos dados de entrada e produzem os dados de saída. Uma *data warehouse* é um repositório de dados resultante da integração de várias fontes de dados, e desenhado para ajudar a efetuar a análise de dados. As ferramentas que executam processos *ETL* são utilizadas para transformar os dados extraídos de várias fontes para um formato exigido pelo repositório de dados destino (Hurwitz *et al.* , 2013).

Existe uma variante do processo de *ETL*, que se designa de ELT, que desempenha as mesmas funções de um processo *ETL*, mas em ordem diferente. Num processo ELT ao invés de transformar os dados antes de estes serem armazenados no repositório de destino, o que se faz é aproveitar o sistema de destino para fazer a transformação. Esta variante faz sentido quando o repositório de destino é algo como um *cluster* onde o Hadoop esteja a ser executado ou uma aplicação que beneficia da *cloud*, uma vez que nestes casos o sistema final tem capacidade de suportar a transformação da informação. Utilizar processos de ELT em *Big Data* traz benefícios em termos de desempenho, que é melhor, e facilidade de escalabilidade. Muitas das ferramentas de ETL tradicionais também oferecem esta vertente de ELT, permitindo assim ao utilizador utilizar essas duas variantes dependendo de qual for melhor para determinada situação.

Com o volume dos dados a aumentar, os processos *ETL* têm a necessidade de processar cada vez mais dados, com diferentes formatos o que pode ser um problema para as ferramentas tradicionais para processos de *ETL*, uma vez que, podem não conseguir lidar com *Big Data* e dados não estruturados de forma eficiente. O paradigma *MapReduce* pode fazer com que o processo de *ETL* seja mais rápido permitindo que as transformações beneficiem do seu modelo de processamento que distribui os dados e o próprio processamento por vários nós de um *cluster* de máquinas.

1.1 Objectivos

O presente trabalho tem como objectivo averiguar em que contextos é útil utilizar tecnologias de Big Data num processo de *ETL*, ou seja pretende-se concluir em que circunstancias é mais adequado a utilização de um ambiente centralizado ou um distribuído para execução de operações de um processo *ETL*. Assim sendo o trabalho a desenvolver consiste nas seguintes etapas:

- Desenvolver um fluxo *ETL* utilizando *MapReduce* com a ferramenta de integração e análise de

dados *Pentaho*[2];

- Desenvolver o mesmo fluxo *ETL* com a plataforma de análise de dados *Pig*[3];
- Desenvolver o mesmo fluxo utilizando o *Pentaho* mas sem *MapReduce*;
- Comparar os três fluxos desenvolvidos tendo em conta alguns critérios como desempenho, facilidade de desenvolvimento, funcionalidades oferecidas, etc..

1.2 Principais Contribuições

As principais contribuições resultantes deste trabalho foram:

- Foi desenvolvido um processo *ETL* de três formas diferente e utilizando tecnologias *open source* diferentes. Os processos implementados podem ser explorados, utilizados e enriquecidos em trabalhos futuros;
- O trabalho foi realizado com o suporte e ajuda da Rede de Novas Licenciaturas (RNL) que nos permitiu utilizar o seu *cluster* para a realização dos testes. Esta parceria resultou numa troca de ajuda e de conhecimento que possibilitou instalar no *cluster* o *Pentaho* e configura-lo com o *Mapredure*, o que acaba por ser uma mais valia para a RNL e o Instituto Superior Técnico;
- O *Pentaho* instalado no *cluster* pode vir a ser utilizado em futuras disciplinas lecionadas no Instituto Superior Técnico, dando assim tanto aos alunos como aos professores a possibilidade de desenvolverem trabalhos relacionados com os temas abordados neste dissertação.

1.3 Estrutura do documento

Este documento está organizado em seis capítulos. O Capítulo 2 apresenta os principais conceitos necessários para a compreensão da dissertação. No Capítulo 3, são apresentadas algumas ferramentas para elaboração de fluxos *ETL*. No Capítulo 4 são detalhadas as três implementações do processo de *ETL*, no Capítulo 5 é efetuada a validação das três implementações. Finalmente, no Capítulo 6 apresentam-se as principais conclusões e possibilidade de trabalho futuro.

[2] `http://www.pentaho.com`
[3] `http://pig.apache.org`

Capítulo 2

Conceitos Fundamentais

Este capítulo apresenta os principais conceitos relacionados com o trabalho desenvolvido, necessários à compreensão do documento. Em particular na secção 2.1 apresenta-se o conceito de Big Data assim como os seus principais desafios. Nas secções 2.2 e 2.3 são apresentados os conceitos de *MapReduce* e *Hadoop*. Por fim na última secção des capítulo faz-se uma definição do que consiste cada uma das fases de um processo de *ETL*.

2.1 Big Data

O termo *Big Data* é usado principalmente para caracterizar grandes conjuntos de dados. Embora a importância de *Big Data* seja já amplamente reconhecida, ainda existem diferentes opiniões em relação à sua definição (Chen *et al.* , 2014). De uma forma geral, *Big Data* refere-se a conjuntos de dados que não podem ser adquiridos, geridos, e processados por ferramentas tradicionais de software/hardware e de Tecnologias de Informação dentro de um determinado limite de tempo (Chen *et al.* , 2014). No entanto, existem várias definições para *Big Data*, entre as quais podemos encontrar as seguintes:

- Em 2010, no contexto do projeto *Apache Hadoop*, *Big Data* foi definido como "conjuntos de dados que não podem ser capturados, geridos e processados por computadores comuns dentro de um limite temporal aceitável" (Chen *et al.* , 2014);
- Com base na definição anterior, McKinsey & Company, em Maio de 2011, anunciou *Big Data* como a próxima fronteira para inovação, competição, e produtividade, definindo *Big Data* como "conjuntos de dados que não podem ser adquiridos, armazenados e geridos pelo software clássico de base de dados" (Chen *et al.* , 2014).

Apesar destas definições serem relativamente recentes, o termo *Big Data* tinha já sido definido no início de 2001. Doug Laney, um analista da META[1], definiu *Big Data* como um modelo de 3V's. Volume para o aumento dos dados gerados e armazenados, Velocidade para a necessidade de armazenar e analisar os dados rapidamente de forma a tirar proveito do valor de *Big Data*, e Variedade no que

[1] presentemente Gartner

diz respeito aos vários tipos de dados que podem incluir dados semi-estruturados e não estruturados tais como áudio, vídeo, páginas web e texto assim como dados estruturados tradicionais (Chen *et al.* , 2014).

O aumento dos dados na era do *Big Data*, traz desafios na aquisição, armazenamento, gestão e análise de dados. Os sistemas tradicionais de gestão e consulta de dados são baseados nos Sistemas de Gestão de Base de Dados Relacionais (SGBDRs) (Chen *et al.* , 2014). Os SGBDRs, apesar de suportarem extensões para alguns tipos de dados não estruturados e semi-estruturados, as suas capacidades para processar dados não estruturados e semi estruturados são limitadas. Para além disso, como mencionado no Capítulo 1, o facto de a informação ser hoje produzida a grande velocidade e em enorme quantidade, leva a que os SGBDRs tenham também alguma dificuldade em lidar com o volume de dados (Chen *et al.* , 2014). Alguns dos principais desafios de *Big Data* dizem respeito a:

- **Integração, Agregação e Representação dos dados**: os dados extraídos de diferentes fontes de *Big Data* possuem muita heterogeneidade. Dessa forma apenas carregar os dados para um repositório destino é insuficiente para que se consiga interpretar esses dados e conseguir tirar proveito dos mesmos. A utilização de metadados adequados faz com que seja possível perceber melhor os dados, mas mesmo assim os desafios irão permanecer devido as diferenças na forma como os dados foram obtidos e na estrutura de registo de dados. Para uma análise em grande escala ser eficaz, o localizar, identificar e compreender dos dados tem de acontecer de forma totalmente automatizada. De maneira a que isto possa acontecer, a diferença na estrutura e semântica dos dados deve ser expressa de tal forma que possa ser percebida pelas tecnologias que irão efetuar a integração, agregação e representação dos dados (Agrawal *et al.* , 2011);

- **Privacidade**: Em Tecnologias de Informação a segurança e privacidade dos dados são dois factores importantes. No contexto de *Big Data*, devido ao facto da informação ser gerada em grande velocidade pode haver riscos de segurança mais graves. Concretamente a segurança em *Big Data* é confrontada com dois desafios. Um dos desafios está relacionado com a proteção da privacidade pessoal durante a fase de aquisição de dados, uma vez que, dados como interesses pessoais, hábitos, etc. podem ser adquiridos sem o utilizador saber. O outro desafio prende-se com o facto de dados pessoais poderem ficar de conhecimento público mesmo sendo adquiridos com o consentimento do utilizador. Um exemplo tipico onde podemos enquadrar essas preocupações é o Facebook. Nesta rede social os utilizadores partilham informações pessoais e muita dessa informação pode ser adquirida por terceiros caso o utilizador se esqueça de modificar as opções de privacidade (Chen *et al.* , 2014);

- **Colaboração**: a análise de *Big Data* requer o trabalho conjunto de pessoas de várias áreas de especialidade. Essas pessoas podem estar separadas no tempo e no espaço quando não for possível reuni-las num único lugar. Por essa razão quando se está a montar um sistema de dados para a análise de informação é preciso levar em conta este facto e suportar a colaboração (Agrawal *et al.* , 2011);

- ***Timeliness***: quanto maior for o conjunto de dados a ser processado, mais tempo irá ser necessário para o analisar. Há muitas situações em que o resultado da análise é necessário imediatamente para se poder tomar uma decisão em tempo real. Dessa forma, conceber um sistema que seja capaz de lidar tanto com o tamanho do conjunto de dados como com rápido processamento desse conjunto torna-se particularmente difícil quando o volume de dados está crescendo de forma rápida e as decisões que se têm de tomar possuem limites de tempo de resposta apertados (Chen *et al.* , 2014).

2.2 Hadoop

O *Hadoop* é um *framework* que permite efetuar o processamento distribuído de grandes conjuntos de dados num *cluster* de máquinas usando modelos de programação como o *MapReduce*. É desenhado para se escalar de um servidor para milhares de servidores, cada um oferecendo processamento e armazenamento local. O *Hadoop* é comummente definido como sendo constituído pelo *MapReduce* e pelo *Hadoop Distributed File System* (HDFS), um sistema de ficheiros distribuído que gere o armazenamento dos dados nas máquinas que compõem o *cluster Hadoop*. Como ilustrado na Figura 2.1, tipicamente, o *Hadoop* lê ficheiros de entrada localizados no *HDFS*, faz alguma computação com *MapReduce* e armazena os resultados no *HDFS*.

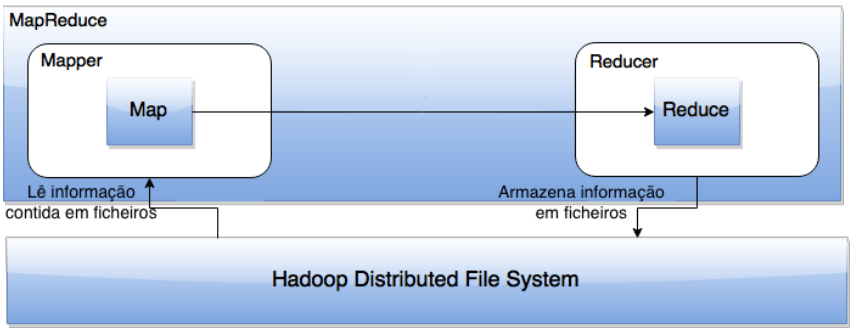

Figura 2.1: Arquitectura básica do Hadoop.

O *HDFS* é responsável por dividir os ficheiros em blocos (tipicamente 128 megabytes) e distribuí-los por máquinas diferentes, fazendo várias cópias de cada bloco, de forma a evitar a perda de informação se uma máquina falhar.

O *HDFS* tem uma arquitetura mestre/escravo. Tipicamente, o *HDFS* de um *cluster* é constituído por dois tipos de nós: um *NameNode* que é o mestre que gere os metadados do *cluster* e o acesso aos ficheiros, e um conjunto de *DataNodes*, normalmente um por cada nó do *cluster*, que armazenam os dados. O *NameNode* executa operações de *namespace* do sistema de ficheiro como abrir, fechar e renomear ficheiros e diretorias, mas também tem como função fazer o mapeamento de blocos para os *DataNodes*. Os *DataNodes* por sua vez são responsáveis por executar pedidos de escrita e leitura provenientes dos clientes do sistema de ficheiro. Têm ainda como função fazer a criação, eliminação e replicação de blocos, respondendo assim a instruções do *NameNode*.

2.3 MapReduce

MapReduce é um modelo de programação utilizado para o processamento, de forma paralela e distribuída, de grandes conjuntos de dados. Os utilizadores para utilizarem o *MapReduce* especificam uma função de *map* que processa um par chave/valor e gera um conjunto intermédio de pares chave/valor, e uma função de *reduce* que agrupa todos os valores intermédios associados a uma mesma chave intermédia e devolve um conjunto menor de valores (Dean & Ghemawat, 2004).

Os programas escritos neste paradigma são automaticamente paralelizados e executados num *cluster* de computadores. *MapReduce* é responsável por particionar os dados de entrada, agendar a execução do programa utilizando um conjunto de máquinas, tratar das falhas das máquinas, e gerir a comunicação entre as máquinas. *MapReduce* permite que os utilizadores sem experiência com sistemas paralelos e distribuídos possam utilizar facilmente os recursos de um sistema distribuído (Dean & Ghemawat, 2004).

O *cluster* onde o *MapReduce* é instalado possui dois tipos de nós, um nó *master* e vários nós *workers*. O nó *master* escolhe *workers* inativos e atribui a cada um uma tarefa de *map* ou *reduce*. O nó *master*, para cada tarefa de *map* ou *reduce* guarda o estado (inativo, em progresso ou concluído) e a identidade do *worker* que está a realizar a tarefa. Para além disso possui informação sobre a localização e o tamanho dos ficheiros intermédios produzidos pelas funções de *map*.

MapReduce é tolerante a falhas porque cada nó *worker* tem de periodicamente reportar o seu funcionamento. Se um *worker* permanecer em silêncio por um tempo mais do que o esperado, o *master* marca-o como estando em falha e atribui o trabalho a outro *worker*.

O modelo de programação *MapReduce* recebe um conjunto de dados como entrada, e converte-o num outro conjunto de dados, onde os elementos individuais são transformados em tuplos (pares chave/valor). Como o próprio nome indica, o utilizador deste modelo de programação expressa o processamento em duas funções distintas: *Map* e *Reduce*:

- A função de *map* recebe um conjunto de dados como entrada e produz um conjunto de pares chave/valor intermédios. O *MapReduce* agrupa depois todos os valores intermédios associados a uma dada chave I e passa essa chave e os seus valores para a função de *reduce* (Dean & Ghemawat, 2004);

- A função de *reduce* aceita uma chave intermédia I e um conjunto de valores a ela associada, ou seja, recebe como entrada os tuplos que foram gerados pelo *map*. De seguida, combina esses dados formando um conjunto possivelmente menor de valores. Tipicamente, apenas zero ou um valor de saída é produzido por cada invocação da função de *reduce*. Os valores intermédios são fornecidos à função de *reduce* através de um iterador permitindo assim lidar com listas de valores que sejam difíceis de manter em memória (Dean & Ghemawat, 2004).

O exemplo mais comum de utilização de *MapReduce* é a contagem do número de ocorrências de cada palavra numa coleção de documentos. O pseudo código apresentado a seguir, apresenta as funções de *map* e *reduce* para esse problema.

```
map(String key, String value):
        //key: document name
        //value: document contents
        for each word w in value:
                EmitIntermediate(w, "1");

reduce(String key, Iterator values):
        //key: a word
        //values: a list of counts
        int result = 0;
        for each v in values:
                result += ParseInt(v);
        Emit(AsString(result));
```

A função de *map* recebe dois parâmetros: uma chave que é nome do documento (*key*), e um valor associado a essa chave, neste caso o seu conteúdo (*value*). Para cada palavra presente no documento devolve uma contagem associada, neste exemplo a contagem devolvida é 1. O par chave/valor intermédio produzido é a palavra e a sua contagem "*EmitIntermediate(w, "1")*".

A função de *reduce* recebe também dois parâmetros: a chave intermédia (*key*) produzida pela função de *map* e uma lista de valores associados a essa chave (*values*). De seguida, soma todos os valores produzindo assim o número total de ocorrências de uma determinada palavra e escreve o resultado final como sendo uma *string* "*Emit(AsString(result))*".

Existem várias implementações de *MapReduce* em várias linguagens de programação como Java, C++, Perl, Python, C e Ruby, que estão disponíveis com as várias distribuições de *Hadoop* existentes, como Apache Hadoop, Cloudera, Hortonworks, IBM InfoSphere BigInsights, MapR, etc..

2.4 ETL

Extract Transform and Load (ETL), como referido no Capítulo 1, é um processo responsável por executar a extração, transformação e carregamento de dados de um ou mais sistemas de dados fonte para um sistema de dados destino. Este processo é efetuado por ferramentas de software especializadas em *ETL*, que têm como objectivos, extrair os dados de várias fontes de dados heterógeneas, transformar e corrigir os dados, se forem detectados erros, para poder armazená-los num formato ou estrutura adequada para consulta e análise, e carregar os dados transformados num sistema de dados destino.

2.4.1 Extracção

Extracção é a operação que consiste em obter os dados de um sistema fonte para posterior utilização num sistema destino. O desenho e criação de um processo de extração é, muitas vezes, uma das tarefas mais demoradas do processo *ETL*. Os sistemas onde residem os dados fonte, podem ser muito

complexos e possuir pouca documentação, o que acaba por tornar difícil a tarefa de determinar quais os dados que precisam ser extraídos (Kimball & Caserta, 2004). Em adição, os dados precisam ser extraídos não apenas uma única vez, mas várias vezes e de forma periódica para poder manter o repositório destino atualizado.

Os sistemas onde residem os dados a extrair não podem ser modificados, nem o seu desempenho pode ser ajustado para satisfazer as necessidades do processo de extração. Por esta razão, a extração deve ser projetada de forma a não afetar negativamente os sistemas fonte em termos de desempenho e tempo de resposta (Lane, 2002).

Durante o carregamento inicial, detectar alterações no conteúdo dos dados não é muito importante porque o mais provável é extrair todos os dados de uma determinada fonte. Mas assim que este carregamento inicial for feito, a capacidade de detectar alterações nos dados passa a ser de enorme importância. Detectar alterações não é uma tarefa trivial e por esta razão, é preciso definir desde do início uma estratégia para detectar alterações incrementais. Algumas formas de detectar alterações nos dados que podemos encontrar são (Kimball & Caserta, 2004):

- ***Audit columns***: presentes na maioria dos sistema fonte, são colunas que são colocadas em cada tabela para guardar a data e hora em que um registo foi adicionado ou modificado. Estas colunas são preenchidas por *triggers*, que são disparados automaticamente assim que um registo é inserido ou atualizado;
- **Extracções baseadas no tempo**: por exemplo se quisermos extrair todos os registos do dia de ontem, basta selecionar os registos em que a data de criação ou modificação seja igual a *SYSDATE-1*. Apesar deste método parecer simples, tem muitas desvantagens. Executar seleções de dados baseadas no tempo pode implicar o carregamento de dados duplicados para o sistema destino quando ocorrem falhas a meio do processo;
- **Processo de eliminação**: este processo preserva uma cópia de cada extração anterior. Na iteração seguinte, é feita uma extração completa dos dados, que, de seguida é comparada com a extração anterior. Apenas as diferenças são enviadas para o repositório destino. Apesar de não ser a técnica mais eficiente, este processo de eliminação é o mais confiável de todas as técnicas de carregamento incremental para detecção de alterações.

2.4.2 Transformação

A fase de transformação aplica um conjunto de regras para transformar os dados extraídos de forma a carregá-los num repositório destino. A transformação dos dados não consiste apenas num simples mapeamento de colunas e tabelas para o local correto no repositório destino. Muitas vezes, é necessário alterar os dados para poderem corresponder ao esquema destino e respectivas restrições de integridade. Algumas das tarefas comuns desta etapa de transformação são (Kimball & Caserta, 2004):

- Filtrar os dados;
- Substituir chaves naturais por *surrogate keys*;

- Combinar e eliminar entidades duplicadas;
- Confirmar atributos comummente usados nas dimensões;
- Normalizar cálculos, criar indicadores de performance chave;
- Corrigir os dados nas rotinas de limpeza de dados.

2.4.3 Carregamento

Após executar todas as tarefas que compõem a fase de transformação, os dados estão prontos para serem carregados para o repositório de dados destino. Antes de executar o carregamento é necessário ter em conta algumas considerações, como por exemplo saber com que frequência o processo de *ETL* será executado. É importante saber a frequência porque o carregamento de informação pode ter um impacto negativo no desempenho do sistema de dados destino. Não só o sistema pode passar a ter dificuldade em executar as suas tarefas como o próprio processamento pode ficar mais lento à medida que os dados vão sendo armazenados. Assim sendo, é necessário garantir que o carregamento é efetuado de forma a causar o menor impacto possível no sistema de dados destino.

Capítulo 3

Trabalho Relacionado

No Capítulo 3 serão apresentadas um conjunto de tecnologias relacionadas com o tema desta dissertação. O capítulo está dividido em três secções. A primeira secção apresenta duas ferramentas do projeto *Apache Hadoop*, o *Apache Pig* e o *Apache Hive* que são duas das ferramentas que o *Hadoop* disponibiliza e que podem ser utilizadas para processos de *ETL*. A segunda secção deste capítulo apresenta duas *frameworks* que exploraram o uso de *ETL* para *Big Data*. A terceira secção exibe as principais características do Pentaho e de duas das mais populares ferramentas utilizadas para processos de *ETL*, *Oracle Data Integrator(ODI)* e *Informatica PowerCenter*. Por fim a ultima secção compara algumas das funcionalidades oferecidas por estas tecnologias.

3.1 Ferramentas do Projecto Apache Hadoop

Esta secção apresenta duas ferramentas do Projecto *Apache Hadoop*, que podem ser utilizadas para desenvolver fluxos *ETL*, o *Apache Pig* e o *Apache Hive*.

3.1.1 Apache Pig

O *Apache Pig*[1] é uma plataforma para analisar grandes conjuntos de dados, que fornece um motor para execução de fluxos de dados de forma paralela sobre o *Hadoop* (Gates, 2011). O *Pig* executa-se sobre o *Hadoop*, utiliza o sistema de ficheiros fornecido pelo *Hadoop*, o *HDFS*, e executa todas as suas operações com o *MapReduce*. Esta plataforma inclui a sua própria linguagem, *Pig Latin*, para especificar fluxos de dados. A linguagem *Pig Latin* inclui operadores para muitas das operações de dados tradicionais (junção, ordenação, filtragem, etc.), mas também possui a possibilidade dos utilizadores desenvolverem as suas próprias funções, em linguagens como Java, Python, JavaScriot ou Ruby, para leitura, processamento e escrita de dados.

Os *scripts* escritos pelos utilizadores em *Pig Latin*, são compilados pelo *Pig* e convertidos para um ou mais programas *MapReduce* que depois são executados utilizando o *Hadoop*.

[1] `http://pig.apache.org/`

A linguagem *Pig Latin* tem como principais propriedades a sua facilidade de programação, uma vez que é relativamente fácil conseguir executar tarefas simples de análise de dados de forma paralela, a sua extensibilidade, os utilizadores podem criar as suas próprias funções para fazer algum tipo de processamento especial, e no facto de que o utilizador não tem que especificar no código como é que as suas instruções serão convertidas para programas *MapReduce*.

A plataforma *Pig* foi desenhada para executar uma longa sequência de operações de dados, tornando-o assim ideal para três categorias de tarefas de *Big Data*: operações de *ETL*, pesquisa em dados ainda não tratados, e processamento iterativo (Gates, 2011).

O caso de uso mais frequente para a utilização desta plataforma são as operações de *ETL*. Um exemplo comum são empresas web que recolhem os ficheiros de log dos seus servidores web, limpam os dados, e calculam agregados antes de carregar os dados para as suas *data warehouses*. Neste caso, o *Pig* é usado para limpar registos com dados corrompidos (Gates, 2011).

Um outro exemplo de utilização da plataforma *Pig* para executar operações de *ETL* é usar o *Pig* para construir modelos de previsão de comportamento. A plataforma Pig pode ser usada para analisar toda a interação que os utilizadores têm com um *website* e classificá-los em vários segmentos. Feita essa classificação será possível produzir um modelo matemático para prever a reação dos utilizadores a determinados tipos de publicidade ou novos artigos. Dessa forma o *website* pode mostrar apenas conteúdos específicos que sejam do interesse dos utilizadores de cada segmento (Gates, 2011).

3.1.2 Apache Hive

O *Apache Hive*[2] é uma plataforma de *software* para *data warehouse* que facilita a consulta e gestão de grandes conjuntos de dados armazenados de forma distribuída. O *Hive* possui um mecanismo que permite estruturar os dados e fazer consultas sobre os mesmos utilizando uma linguagem semelhante ao *SQL* denominada *HiveQL*[3]. Esta linguagem também permite aos programadores de *MapReduce* utilizarem as suas funções de *map* e *reduce* quando for inconveniente ou ineficiente utilizar o *HiveQL*. Construído sobre o *Apache Hadoop*, o *Hive* fornece as seguintes funcionalidades:

- Ferramentas para extrair, transformar e armazenar os dados (*ETL*);
- Um mecanismo que permite estruturar os dados de forma a poder analisá-los com a linguagem *HiveQL*;
- Acesso a ficheiros armazenados no *HDFS* ou em outros sistemas de armazenamento como o *Apache Hbase*[4], uma base de dados do *Hadoop* desenhada para ser utilizada quando se precisa efetuar acessos em tempo real de escrita ou leitura aos dados de *Big Data*;
- Execução de consultas através do *MapReduce*.

A linguagem *HiveQL* é semelhante ao *SQL*, o que permite aos utilizadores familiarizados com *SQL* consultar os dados. Além disso, esta linguagem permite aos programadores que estão familiarizados

[2] `https://hive.apache.org/`
[3] `https://hive.apache.org/`
[4] `http://hbase.apache.org/`

com a framework *MapReduce*, utilizar as suas funções de *map* ou *reduce* de forma a fazer um tipo de análise que pode não ser suportada pelas capacidades da linguagem *HiveQL*.

O *Hive* não é projetado para processamento de transações em tempo real e não oferece consultas em tempo real. É melhor utilizado para trabalhos em *batch* sobre grandes conjuntos de dados em que nunca se retira ou modifica informação, apenas acrescentasse, por exemplo web logs. Os aspectos que mais importância têm para o *Hive* são aspectos como a escalabilidade (escala-se com mais máquinas adicionadas de forma dinâmica ao *cluster Hadoop*), extensibilidade (com *MapReduce* e funções definidas pelo utilizador), tolerância a falhas, não ficar dependente com o formato dos dados de entrada[5].

O *Hive* possui dois componentes, *HCatalog* e *WebHCat*. *HCatalog* é uma tabela e uma camada de gestão de armazenamento para *Hadoop* que dá aos utilizadores de diferentes ferramentas de processamento de dados (incluindo *Pig* e *MapReduce*) uma forma mais fácil de ler e escrever dados para o ambiente do *Hive*. *WebHCat* fornece um serviço que pode ser utilizado para executar programas *MapReduce*, *Pig*, *Hive* ou efetuar operações aos metadados *Hive* utilizando uma interface HTTP.

Ao contrário do *Pig* e *MapReduce*, o *Hive* faz com que seja mais fácil para os utilizadores de SGBDRs tradicionais ou outros utilizadores que saibam *SQL*, aceder e transformar dados no *Hadoop*. O *Pig* pode não ser tão fácil de aprender como o *Hive* mas ambas as tecnologias têm um tempo de aprendizagem para aqueles que não têm um *background* em termos de desenvolvimento de *software*. O *MapReduce* é uma tecnologia que programadores de Java, C++ e Python podem assimilar de forma relativamente rápida. Mas sem uma base numa linguagem como Java, o *MapReduce* pode ser difícil de aprender (Jamack, 2014).

3.2 Frameworks ETL

A Secção 3.2 apresenta duas frameworks para efetuar fluxos *ETL* utilizando *MapReduce*.

3.2.1 CloudETL

CloudETL (Liu *et al.* , 2012) é uma framework para *ETL* que usa *Hadoop* para paralelizar fluxos *ETL* e processar dados em *Hive*. O utilizador define o processo *ETL* através de construções e transformações de alto nível e não precisa de se preocupar com detalhes técnicos relacionados com *MapReduce*. A framework *CloudETL* suporta esquemas de *data warehouse* diferentes, tais como esquemas em estrela e *slowly changing dimensions* (Liu *et al.* , 2012).

O *Hive* tem muitas das operações e funções *standard* de *SQL* e consegue ler dados de vários formatos, mas possui limitadas capacidades de *ETL* quando se quer fazer o processamento com base em várias dimensões de análise. É utilizado para processos *ETL* simples, mas para cenários mais complexos não é a ferramenta apropriada. A plataforma *Hive* não tem suporte por exemplo para procurar um elemento de uma dimensão e caso não o encontrar, atualizar a tabela de dimensão. A framework *CloudETL* suporta este tipo de operações e também tem suporte para as *slowly changing dimensions*.

[5] `https://cwiki.apache.org/confluence/display/Hive/Home`

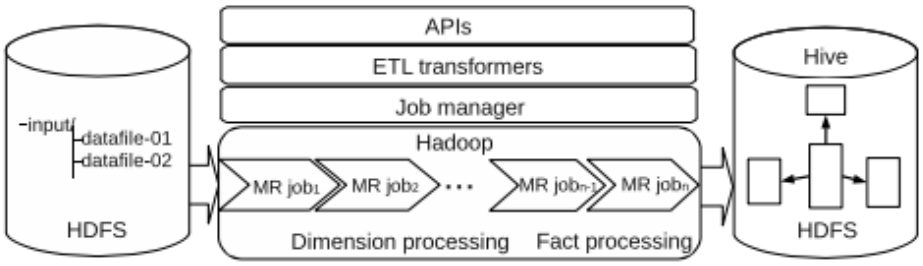

Figura 3.1: Arquitectura CloudETL

Esta framework pode ser vista como uma camada que se situa em cima do *Hive* e tem o objectivo de tornar mais fácil e mais rápida a criação de processos *ETL* escaláveis para carregar dados em *data warehouses Hive*. O *CloudETL* permite que os programadores de *ETL* possam facilmente converter um programa *ETL* em programas *MapReduce* sobre *Hadoop* utilizando instruções de alto nível, ou seja instruções com um elevado grau de abstração, num programa Java e sem ter que lidar com os detalhes de *MapReduce* (Liu *et al.* , 2012).

A framework *CloudETL* usa *Hadoop* como a plataforma *ETL* de execução, *Hive* como um sistema de *data warehouse* e possui os seguintes componentes: interfaces de programação de aplicação (APIs) utilizadas pelos programas *ETL* dos utilizadores, um conjunto de elementos para efetuar transformações nos dados denominados de *ETL transformers*, e um gestor de programas que controla a execução de programas submetidos no *Hadoop*. Esta arquitetura é demonstrada na Figura 3.1 (Liu *et al.* , 2012).

Num fluxo de trabalho *CloudETL* existem duas fases sequenciais: processamento de dimensão e processamento de factos. As fontes dos dados precisam estar presentes no *HDFS* quando os programas *MapReduce* são iniciados. A framework *CloudETL* permite que no decorrer de um *job* os dados sejam processados em várias tabelas. Os dados de entrada são transformados em valores de dimensão ou de factos através das transformações especificadas pelo utilizador e que são executadas por *mappers* e *reducers*. A componente denominada de *ETL transformer* integra um conjunto de transformações de processamento de dados como conversões, *lookups* (para obter chaves de alguns valores de dimensão), entre outros. O gestor de programas submete os *jobs* numa ordem sequencial. Primeiramente faz-se o processamento de dimensão e só depois é que é executado o processamento de factos, uma vez que as tabelas de factos necessitam de consultar as chaves das tabelas das dimensões. A plataforma Hive utiliza o *HDFS* para armazenar fisicamente os dados mas apresenta os dados contidos em ficheiros como tabelas lógicas.

3.2.2 ETLMR

ETLMR (Liu *et al.* , 2011) é uma framework de programação para *ETL* que usa *MapReduce* para atingir escalabilidade. A framework *ETLMR* tem suporte nativo para esquemas específicos de *data warehousing*, tais como esquemas em estrela, *snowflakes*, e *slowly changing dimensions*. O facto de suportar estes esquemas específicos faz com que seja possível aos utilizadores construir fluxos de *ETL* com base em *MapReduce* utilizando poucas linhas de código (Liu *et al.* , 2011).

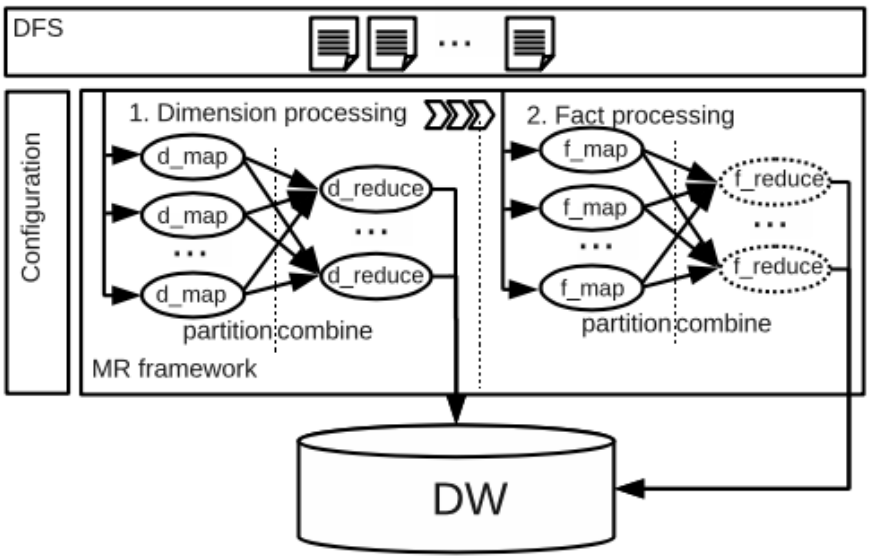

Figura 3.2: Fluxo ETL sobre MapReduce

Esta framework oferece construções especificas de *ETL* de alto nível, sobre tabelas de factos e dimensões tanto em esquemas em estrela como em esquemas *snowflakes*. Um utilizador pode implementar programas de *ETL* paralelos utilizando essas construções sem ser necessário ter conhecimento dos detalhes da execução paralela de processos de *ETL*. Esta funcionalidade facilita o trabalho de programação uma vez que o utilizador apenas tem que fazer um ficheiro de configuração com poucas linhas de código para declarar objectos de factos e dimensões e as funções de transformação necessárias (Liu *et al.* , 2011). O *ETLMR* utiliza uma framework de programação baseada em Phyton para tornar a programação mais fácil, chamada de *pygrametl* (Thomsen & Pedersen, 2009).

A Figura 3.2 ilustra um processo *ETL* paralelo utilizando *ETLMR*. O fluxo *ETL* consiste em duas fases sequenciais: processamento das dimensões e processamento dos factos, cada uma executada como um programa *MapReduce*. Os dados são lidos das fontes residentes num sistema de ficheiros distribuído, depois são transformados, carregados em tabelas de dimensão e factos por instruções de *ETLMR* paralelas que consolidam os dados numa *data warehouse*. Para escrever um programa de *ETL* paralelo são necessárias apenas algumas linhas de código que declaram tabelas de destino e funções de transformação (Liu *et al.* , 2011).

A concretização de *MapReduce* utilizada por *ETLMR* é o *Disco*[6] que é uma concretização *MapReduce open source* desenvolvida pela Nókia, que utiliza o Erlang e Phython como linguagens de programação. Os criadores da framework *ETLMR* escolheram o *Disco* por três razões. Primeiro porque usa Phython o, que para eles, facilita um *scripting* rápido de programas de processamento de dados. Segundo, o *Disco* consegue atingir um alto grau de flexibilidade porque fornece muitas interfaces *MapReduce* personalizáveis. Terceiro, oferece suporte direto para programas distribuídos escritos em Phython (Liu *et al.* , 2011).

[6] `http://discoproject.org`

3.3 Ferramentas de Integração de Dados

Nesta secção são apresentadas três das ferramentas de integração de dados utilizadas para executar o processo *ETL*, *Pentaho, Oracle Data Integrator e PowerCenter*. A primeira ferramenta a ser apresentada é o *Pentaho*, uma ferramenta *open source*, bastante representativa e que será utilizada neste trabalho. De seguida, é apresentado o *ODI* uma ferramenta que se designa por ser uma ferramenta para *ELT* (transformações acontecem no repositório de dados destino) e não de *ETL*, bastante conhecida e utilizada, e por fim o *PowerCenter* uma das ferramentas mais utilizadas.

3.3.1 Pentaho Data Integration

O *Pentaho Data Integration PDI*[7] é uma ferramenta de *ETL* que permite ao utilizador aceder, preparar, analisar e extrair informação útil de dados tanto tradicionais como de *big data*. O *Pentaho* tem como objectivo simplificar a gestão de grandes volumes de dados que entram nas empresas, a velocidades e variedade cada vez mais elevadas, independentemente do tipo dos dados e número de fontes de dados. O *PDI* possui uma interface gráfica que oferece aos utilizadores uma alternativa à opção de terem de codificar programas, em *SQL* ou *MapReduce*, para processar os dados.

O *Pentaho* permite o desenvolvimento de processos *ETL* utilizando operadores convencionais de *ETL* ou utilizando operadores de *Big Data*, onde se insere o *MapReduce*.

Integrar e combinar Big Data com dados existentes

O *Pentaho* permite ler e escrever dados de uma variedade de fontes e formatos e isso acaba por simplificar e acelerar o processo de integrar base de dados existentes com novas fontes de dados. O *PDI* possui uma interface gráfica com as seguintes características:

- *Drag and drop* intuitivo;
- Biblioteca completa com componentes pré-construídos para aceder e transformar dados de um conjunto completo de fontes;
- Transformações dinâmicas;
- *Debugger* integrado para testar e melhorar execução de tarefas;
- Capacidade de se conectar com *Hadoop*, base de dados *NoSQL* e base de dados analíticas;
- Ferramentas visuais para elaborar programas; *MapReduce* que permitem reduzir os ciclos de desenvolvimento;
- Mecanismos para preparar, modelar e explorar conjuntos de dados não estruturados.

[7] `http://www.pentaho.com/product/data-integration`

O *Pentaho* tem ainda um motor de integração de dados que fornece: (i) um mecanismo de multi-tarefas para uma execução mais rápida, (ii) suporte para *clusters* de máquinas permitindo distribuir o processamento por vários nós, e (iii) execução sobre o *Hadoop* para um desempenho mais rápido.

O *PDI* possui uma capacidade de se conectar a uma grande variedade de dados, incluindo fontes de dados estruturados, não estruturados e semi-estruturados. Como alguns exemplos podemos encontrar:

- Base de dados relacionais, Oracle, DB2, MySQL, SQL Server;
- Hadoop, Apache Hadoop, Cloudera, HortonWorks, MapR;
- Base de dados NoSQL, MongoDB, Cassandra, HBase;
- Base de dados analíticas, Vertica, Greenplum, Teradata;
- SAP;
- Aplicações baseadas na cloud e SaaS;
- Ficheiros, XML, Excel, flat file e APIs web services.

3.3.2 Oracle Data Integrator

Oracle Data Integrator(ODI)[8] é uma plataforma de integração de dados que cobre vários dos requisitos de integração de dados, desde grandes quantidades de dados ao carregamento de dados em *batch*.

No *ODI* os utilizadores podem definir o transporte e transformação de dados criando graficamente mapeamentos lógicos. Os utilizadores criam um fluxo de dados dos sistemas que contêm as fontes dos dados para os sistemas de destino utilizando diferentes tecnologias. Os sistemas tanto de origem como destino podem incluir base de dados relacionais, aplicações, ficheiros XML, tabelas *Hive*, *HBase*, ficheiros *HDFS*, etc. Os utilizadores podem também inserir filtros, junções, agregações, e outros componentes de transformação. Este desenho lógico pode ser feito sem ter a necessidade de escolher a implementação física para efetuar a transferência e transformação dos dados[9].

Depois de o utilizador finalizar o desenho lógico do fluxo de dados, o *ODI* sugere uma implementação física, que inclui mecanismos para transferir os dados, como *Sqoop* ou *Oracle Loader* para *Hadoop*, e o mecanismo em que as transformações são executadas, tal como *Hive* ou uma base de dados relacional. O *ODI* gera código para cada uma das tecnologias subjacentes utilizando o conceito de *Knowledge Module*, que correspondem a um conjunto de *templates* de código que são reutilizáveis e podem ser editáveis pelos utilizadores para encapsular as melhores práticas de organização de desenvolvimento do utilizador. Os utilizadores do *ODI* são isolados dos detalhes das linguagens *Hadoop* subjacentes e dos ficheiros de configuração, que de outra forma, teriam de ser desenvolvidos manualmente.

A metodologia declarativa do *ODI* de separar o desenho lógico da implementação física permite manter-se atualizado com as mais recentes ferramentas *Hadoop*. À medida que as tecnologias subjacentes para o transporte e transformação de dados são enriquecidas e novos projetos vão emergindo,

[8] `http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html`
[9] `http://hortonworks.com/blog/oracle-data-integrator-hortonworks-data-platform/`

os *Knowledge Modules* podem ser desenvolvidos para executar o desenho lógico existente utilizando os recursos do *Hadoop* mais recentes.

3.3.3 PowerCenter

Informatica PowerCenter[10] é uma ferramenta de integração de dados utilizada em processos de *ETL*. Esta ferramenta permite extrair dados de várias fontes, transformar esses dados de acordo com a lógica de negócio desejada e carrega-los para sistemas de destino como *datamarts* ou *data warehouses*.

A ferramenta PowerCenter é composta pelos seguintes componentes:

- *PowerCenter Repository*: este repositório é a componente central de todo o *PowerCenter*. Agrega na sua base de dados um conjunto de tabelas de metadados que são acedidas pelos *PowerCenter Client* e *Server* para consultar e guardar informação;
- *PowerCenter Repository Server*: esta componente gere as conexões entre o repositório e as aplicações cliente. A sua função é inserir, atualizar, ir buscar informação das tabelas da base de dados do repositório e manter a consistência dos objecto que contêm informação;
- *PowerCenter Client*: o cliente é utilizado para gerir os utilizadores, definir os sistemas fonte e destino, criar os mapeamentos entre esses sistemas e criar fluxos de execução para a lógica de mapeamento definida. Esta componente está dividida em várias subcomponentes:
  - *Repository Manager*
  - *Repository Server Administration Console*
  - *Designer*
  - *Workflow Manager*
  - *Workflow Monitor*
- *PowerCenter Server*: o *PowerCenter Server* é responsável por extrair os dados das fontes, executar as transformações definidas e carregar os dados transformados para o sistema de destino.

Fontes de dados

O PowerCenter pode aceder as seguintes fontes de dados:

- Relacionais: Oracle, Sybase, Informix, IBM DB2, Microsoft SQL Server e Teradata.
- Ficheiros: *Flat Files* fixos e delimitados, ficheiros COBOL e XML.
- Aplicações: tendo produtos adicionais da *PowerCenter Connect* é possível aceder a fontes de negócio como PeopleSoft, SAP R/3, Siebel, IBM MQSeries, and TIBCO.
- Mainframe: *PowerExchange* permite um acesso mais rápido à IBM DB2 Multiple Virtual Storage (MVS).
- Outros: Microsoft Excel e Access.

[10] `http://www.informatica.com/pt/products/data-integration/enterprise/powercenter/`

Sistemas Destino

O PowerCenter poder carregar dados para os seguintes sistemas destino:

- Relacionais: Oracle, Sybase, Sybase IQ, Informix, IBM DB2, Microsoft SQL Server, and Teradata;
- Ficheiros: *Flat Files* fixos e delimitados e XML;
- Aplicações: tendo produtos adicionais da *PowerCenter Connect* é possível carregar dados para SAP BW, IBM MQSeries e TIBCO;
- Outros: Microsoft Access.

3.4 Comparação das ferramentas

A Tabela 3.1 apresenta uma comparação entre as ferramentas apresentadas tendo em conta algumas características que normalmente estão presentes em ferramentas de *ETL*. A tabela inclui tanto ferramentas próprias para processos de *ETL* como também ferramentas do projeto *Apache Hadoop*, que embora possam ser consideradas de mais baixo nível, podem ser utilizadas para desempenhar funções de processos de *ETL*, uma vez que suportam algumas funcionalidades próprias de ferramentas de *ETL*. Ao analisar a tabela podemos ver que as ferramentas desenhadas mesmo para processos de *ETL* suportam todas as funcionalidades levadas em conta na comparação. O Pig e o Hive são mais limitados, não oferecem algumas das funcionalidades de raiz, mas sim através de funções escritas pelos utilizadores.

Tabela 3.1: Comparação entre ferramentas para processos ETL.

	Pentaho	**PowerCenter**	**ODI**	**Pig**	**Hive**
Filtrar Dados	✓	✓	✓	✓	✓
Combinar e Eliminar Entidades Duplicadas	✓	✓	✓	✓	✓
Limpeza de Dados	✓	✓	✓	Não suporta limpeza absoluta	Não suporta limpeza absoluta
Facilidade de Utilização	O Pentaho possui uma interface simples e intuitiva	—	—	Para utilizadores com background em termos de desenvolvimento de software	Para utilizadores de SGBDRs tradicionais
Processamento Paralelo	✓	✓	✓	Dados têm de estar no *Hadoop*	Dados têm de estar no *Hadoop*
Comercial vs Open Source	Open Source	Comercial	Comercial	Open Source	Open Source
Transformações Complexas	✓	✓	✓	Funções definidas pelo utilizador	Funções definidas pelo utilizador
Particionamento de dados	✓	✓	✓	—	—
Lookups Complexos	✓	✓	✓	Funções definidas pelo utilizador	Funções definidas pelo utilizador
Data lineage	✓	✓	✓	—	—

Capítulo 4

Processo de *ETL*

O objectivo desta dissertação foi averiguar em que condições é mais adequado o uso de tecnologias de Big Data para o desenvolvimento de um processo de *ETL*. Com esse objectivo em mente foi decidido que seria implementado um processo de *ETL* de três formas diferentes. Duas das implementações foram desenvolvidas com tecnologias de Big Data num ambiente distribuído de forma a tirar partido das vantagens de processamento do *Hadoop* e do *Mapreduce*. A terceira implementação foi feita num ambiente centralizado de forma a poder analisar e comparar o seu desempenho com as restantes implementações e assim determinar a partir de que condições é mais vantajoso utilizar tecnologias como o *MapReduce* para o desenvolvimento de processos de *ETL*.

Neste capítulo serão detalhadas as três implementações do processo de *ETL* utilizando o *Pentaho* com e sem *MapReduce* e o *Pig*. O capítulo começa por apresentar a arquitetura que foi estabelecida para desenvolver do processo de *ETL*. De seguida serão apresentados os sistemas operacionais, onde residem os dados de origem, e o repositório analítico que é o local onde é armazenada a informação extraída dos dados após terem sido transformados pelo processo de *ETL*. A terceira secção deste capítulo apresenta as transformações lógicas que foram aplicadas aos dados. Por fim na última secção são apresentadas as três implementações do processo de *ETL*.

4.1 Arquitectura da Solução proposta

Tal como referido no primeiro capítulo, o objectivo deste trabalho consiste em verificar em que contextos é mais vantajoso utilizar um ambiente centralizado ou distribuído para a execução de um processo de *ETL*. Para efetuar esta verificação foi desenvolvido de raiz um sistema de suporte à decisão, onde se enquadram as respectivas implementações do processo de *ETL*. A Figura 4.1 mostra a arquitetura da solução proposta. Os componentes desta arquitetura são os seguintes: (i) sistemas operacionais em que os dados estão organizados segundo um modelo de dados origem; (ii) canalização de dados (Processo *ETL*), onde são efetuadas as transformações entre o modelo de dados origem e o modelo de dados destino; (iii) repositório de dados analítico, onde os dados estão organizados de acordo com um modelo analítico (modelo de dados destino), em indicadores e dimensões de análise; e (iv) uma

componente de análise de dados para se poder consultar com facilidade o conteúdo do repositório analítico. Este capítulo foca-se na canalização dos dados, ou seja, no processo *ETL*.

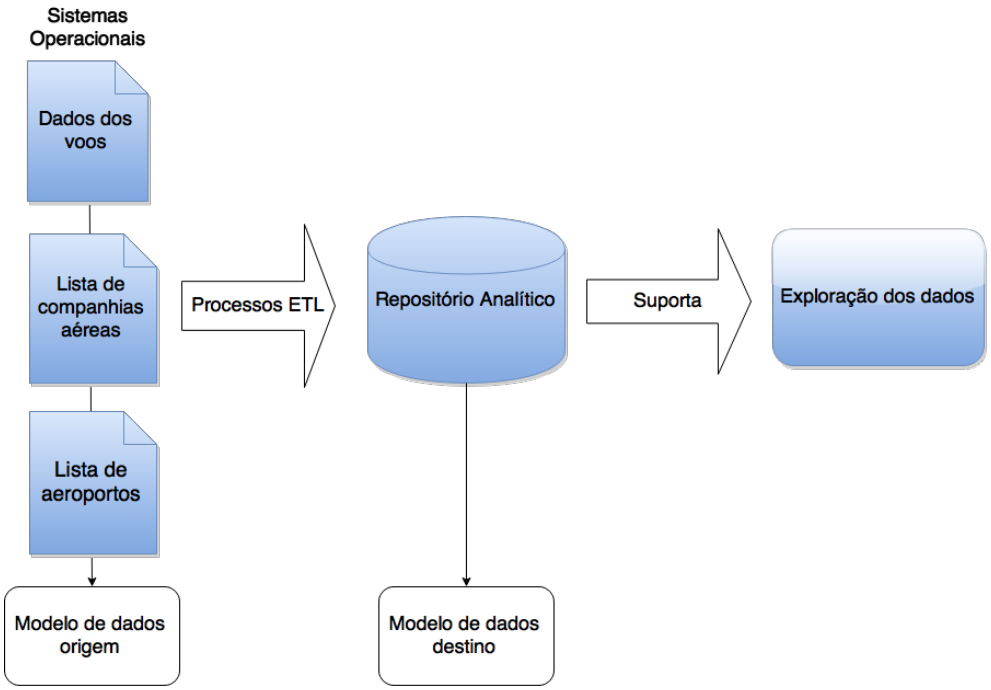

Figura 4.1: Arquitetura lógica da solução proposta

Os dados dos sistemas operacionais que foram utilizados para alimentar o processo de *ETL* foram dados públicos com informação estatística relativa a voos comerciais realizados nos Estados Unidos, desde Outubro de 1987 até Abril de 2008 (cerca de 120 milhões de registos)[1]. Foi selecionado esse conjunto de dados por conter informação interessante sobre a qual se poderia calcular o conjunto de indicadores. Por isso teve de ser construído um modelo multidimensional, secção 4.2 constituído por tabelas de factos e dimensão.

O repositório analítico para as implementações distribuídas foi o *HDFS* e para implementação centralizada foi o sistema de ficheiro local da máquina onde o processo foi executado. O processo de *ETL* foi implementado recorrendo a três cenários tecnológicos distintos:

- *Pentaho*, utilizando operadores convencionais de *ETL*;
- *Pentaho*, utilizando operadores de *MapReduce*;
- *MapReduce*, utilizando a linguagem de *scripting Pig*.

Estes cenários serão comparados de acordo com o desempenho e funcionalidades oferecidas (em particular as primitivas de transformação) pelas ferramentas, possibilitando desta forma aferir em que contextos é mais adequado a utilização de um certo cenário em detrimento dos restantes.

Como já foi mencionado nesta secção, foi desenvolvido um modelo multidimensional constituído por tabelas de factos e dimensão. A opção por desenvolver esse modelo multidimensional de raiz, bem

[1] http://stat-computing.org/dataexpo/2009/the-data.html

como o universo de dados escolhido, justifica-se de três formas: (i) poder efetuar uma comparação dos vários cenários ao nível do critério de desenvolvimento; (ii) poder construir um conjunto de indicadores e dimensões de análise que "obriguem" as implementações do processo de *ETL* a exercitarem diferentes primitivas de transformação; (iii) Poder efetuar testes com grandes volumes de dados.

Nas secções seguintes apresentam-se os componentes da arquitetura mais em detalhe.

4.2 Sistemas Operacionais e Repositório Analítico

Como referido na secção anterior, o conjunto de dados sobre o qual o trabalho é feito é um conjunto de dados público com detalhes sobre a partida e chegada de voos comerciais realizados nos Estados Unidos. Esta informação está organizada em três ficheiros distintos:

- Um ficheiro principal com os dados relativos aos voos, nomeadamente: período de tempo em que o voo aconteceu, origem e destino, hora de partida e chegada, causas de atraso, etc;
- Um ficheiro com o código e nome das companhias aéreas;
- Um ficheiro com informação sobre os aeroportos, nomeadamente: código, nome, cidade, estado, etc.

O repositório analítico está organizado num conjunto de indicadores e dimensões de análise. Os indicadores são armazenados em duas tabelas de factos, denominadas de Indicadores Diretos (indicadores cuja fórmula de cálculo é direta) e Indicadores Calculados (indicadores cuja fórmula de calculo é mais complexa envolvendo várias operações) e as dimensões de análise dividem-se em seis tabelas de dimensões: período, companhia aérea, aeroporto de partida, aeroporto de chegada, motivo de cancelamento do voo e motivo de atraso do voo. A Figura 4.2 ilustra o modelo multidimensional do repositório analítico. Conforme se pode ver pela figura, os indicadores diretos estão agrupados segundo um conjunto maior de dimensões do que os indicadores calculados.

4.3 Transformações Lógicas

Como já foi mencionado os processos *ETL* são os responsáveis por fazer a transformação entre o modelo de dados origem e o modelo de dados destino. A lógica das três implementações é a mesma. O que difere é a forma como foram implementados, que, devido aos ambientes e tecnologias envolvidos, tinha de ser necessariamente diferente. A lógica do processo de *ETL* consiste em três etapas:

- Calcular o indicador: nesta primeira etapa é feito o cálculo dos indicadores, agrupando os dados pelas dimensões exemplificadas na Figura 4.2;
- Pesquisar os identificadores: após calcular os indicadores, faz-se uma pesquisa dos identificadores das dimensões sobre as quais os indicadores foram agrupados e calculados;
- Armazenar os resultados: por fim é feito o armazenamento do resultado.

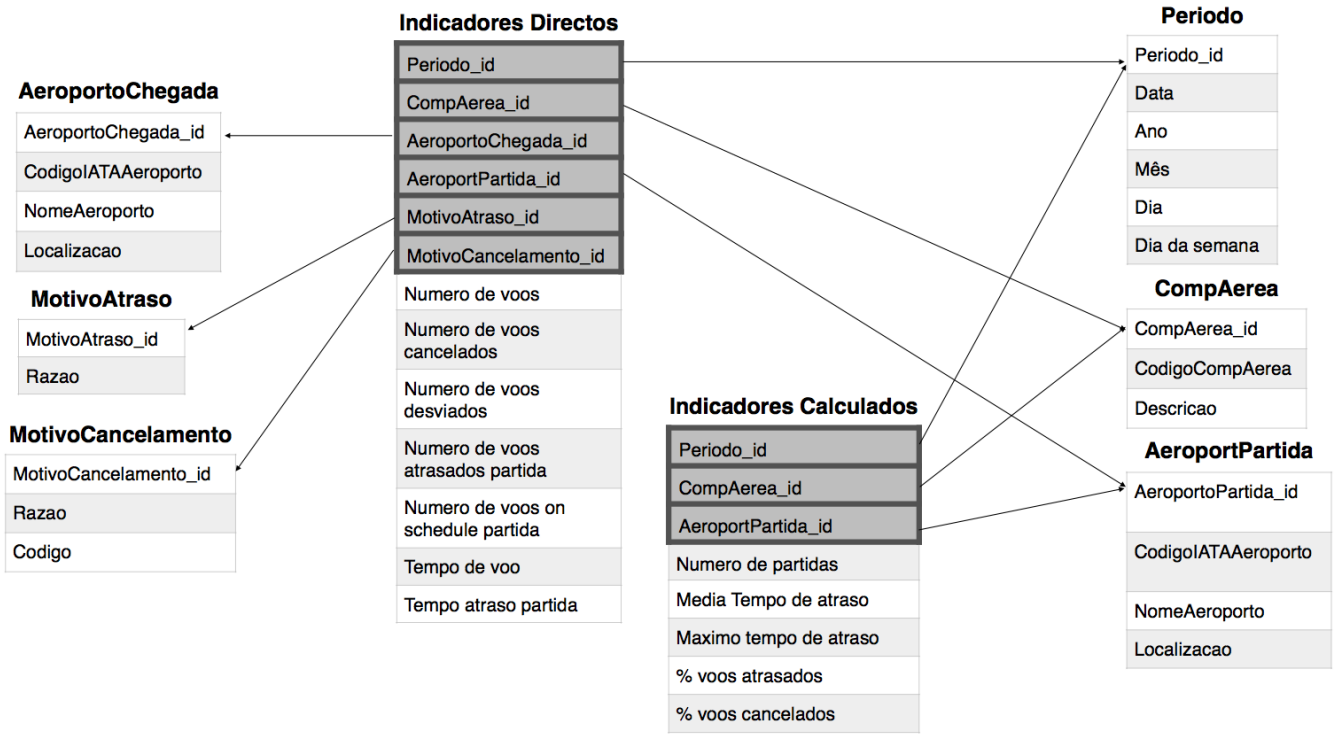

Figura 4.2: Repositório Analítico

O cálculo dos indicadores é efetuado pela execução de interrogações SQL aos sistemas operacionais, agrupadas mediante um conjunto de atributos. A Figura 4.3 ilustra algumas dessas interrogações e o seu mapeamento para os indicadores respectivos.

SELECT year, month, dayofmonth, uniquecarrier, dest, origin, cancellationcode, **COUNT**(flightnum) as NumVoosCancelados **FROM** flight **WHERE** cancelled=1 **GROUP BY** year, month, dayofmonth, uniquecarrier, dest, origin, cancellationcode	Número de voos cancelados
SELECT year, month, dayofmonth, uniquecarrier, dest, origin, **COUNT**(flightnum) as NumVoosDesviados **FROM** flight **WHERE** diverted=1 **GROUP BY** year, month, dayofmonth, uniquecarrier, dest, origin	Número de voos desviados
SELECT periodo_id, carrier_id, depairport_id, (**SUM**(numvooscancelados) / **SUM**(numvoos))*100 as PercVoosCancelados **FROM** Indicadores_Directos **GROUP BY** periodo_id, carrier_id, depairport_id	Percentagem de voos cancelados

Figura 4.3: Mapeamento entre os sistemas operacionais e o repositório analítico

4.4 Implementações do processo de *ETL*

Esta secção irá apresentar as três implementações do processo de *ETL* detalhando a forma como cada uma faz o cálculo dos indicadores de análise.

4.4.1 Pentaho Data Integration (PDI) utilizando primitivas convencionais de ETL

O desenvolvimento de um fluxo de transformações de dados no *PDI* está organizado em *jobs* e transformações. Uma transformação está relacionada com a movimentação e alteração de registos da fonte para o destino. Um *job*, por sua vez, está relacionado mais com aspectos de alto nível de controlo do fluxo das transformações. Um *job* pode ser visto como uma orquestração de transformações.

O cálculo dos indicadores de análise está organizado em dois *jobs*, um para os indicadores diretos e outro para os indicadores calculados. Ambos os *jobs* possuem um conjunto de transformações que calculam um ou mais indicadores. Nesta primeira implementação do processo de *ETL* os indicadores são obtidos através da execução das interrogações, como as exemplificadas na secção 4.3. Nesse sentido, os indicadores que partilham a mesma cláusula "*where*" são calculados numa mesma transformação. A Figura 4.4 ilustra o fluxo de transformações do *job* para o cálculo dos indicares diretos. Ao analisar a figura, podemos reparar que o *job* é composto por um ponto de início, um ponto de término de execução e por várias transformações responsáveis por calcular os indicadores.

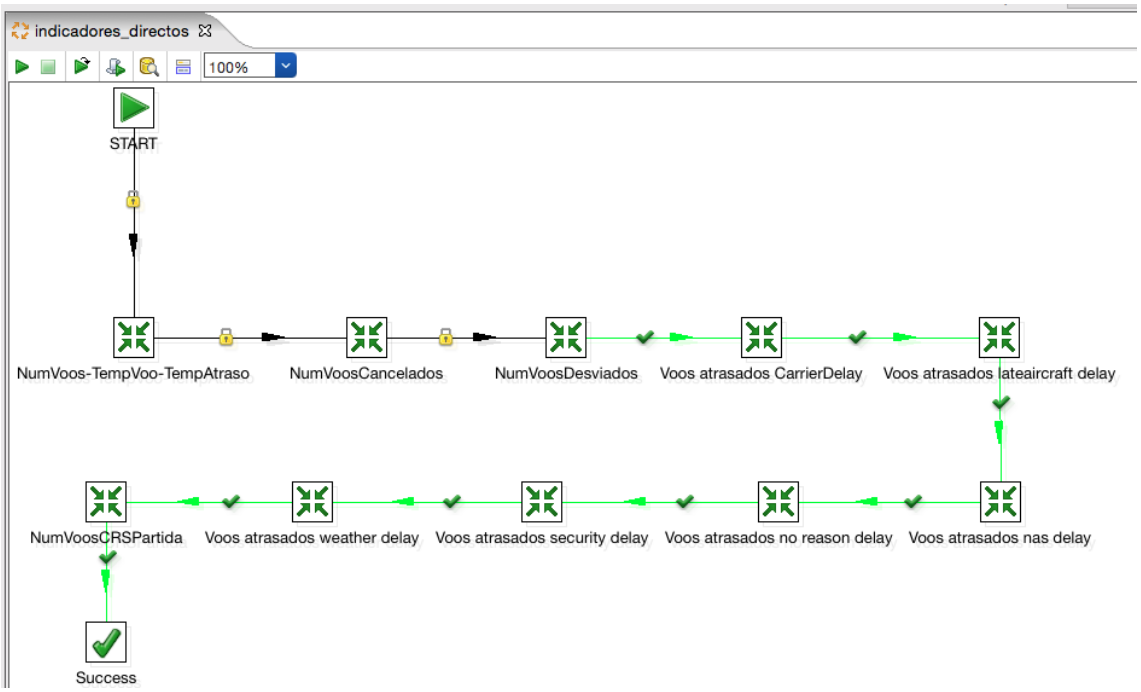

Figura 4.4: Fluxo de transformações do Job para o cálculo dos Indicadores Diretos

As transformações são compostas por vários componentes tendo um fluxo de execução semelhante ao ilustrado na Figura 4.5. A transformação começa por executar o operador responsável pela execução da interrogação que calcula o indicador. Com o objectivo de armazenar o indicador de análise na tabela de factos respectiva, é executado um conjunto de pesquisas às dimensões de análise de forma a obter o identificador de cada dimensão. Esta implementação do processo vais buscar os dados de entrada numa base de dados *MySQL* e armazena os resultados em ficheiros no sistema de ficheiros local do computador.

Os vários componentes de uma transformação executam-se em paralelo, ou seja, um componente para iniciar a sua execução não precisa esperar que o componente anterior termine a sua execução. À

Figura 4.5: Fluxo de execução de uma transformação

medida que os dados vão sendo produzidos, vão sendo passados para os componentes seguintes.

4.4.2 Pentaho Data Integration (PDI) utilizando primitivas de MapReduce

A segunda implementação do processo de *ETL* utiliza o *PDI* com primitivas de *MapReduce*. O cálculo dos indicadores está, à semelhança da implementação anterior, organizado em dois *jobs*, mas os jobs são agora uma orquestração de programas *MapReduce*.

Como é utilizado o *MapReduce* para efetuar o cálculo dos indicadores, os dados de origem têm de ser transferidos, sob a forma de ficheiros, para o sistema de ficheiros distribuído do *Hadoop*, o *HDFS*. A saída gerada por cada programa *MapReduce* é também armazenada no *HDFS*. A Figura 4.6 ilustra a arquitetura dos *jobs* utilizando *MapReduce*.

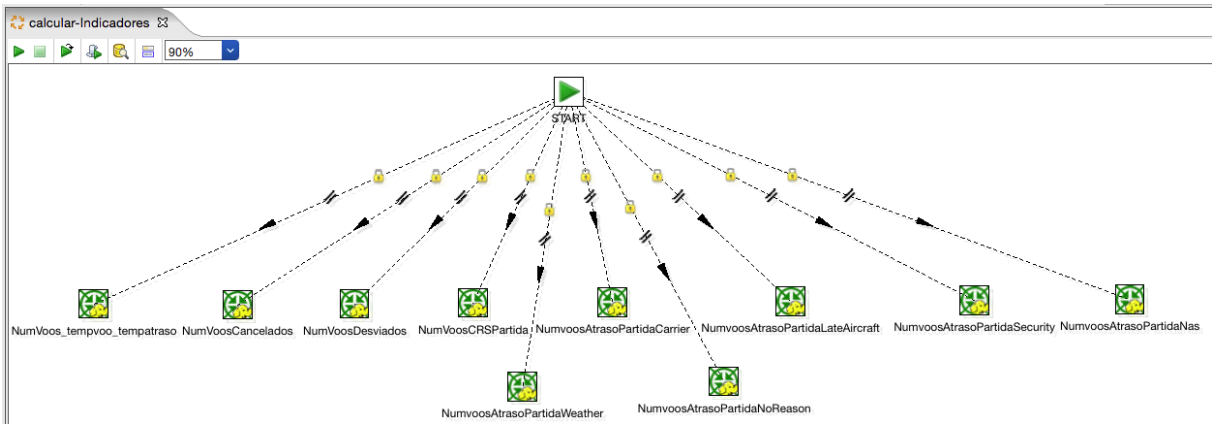

Figura 4.6: Job para o cáculo dos Indicadores Directos com MapReduce

Na implementação anterior os indicadores que partilham a mesma clausula *where* foram calculados na mesma interrogação. Nesta implementação utilizando *MapReduce* o mesmo acontece, ou seja, os indicadores que satisfaçam o mesmo critério de seleção são calculados num mesmo programa *MapReduce*.

As duas implementações do processo de *ETL* que utilizam o *PDI*, apesar de serem implementadas no *PDI* são bastante diferentes. No primeiro caso os dados são lidos de uma base de dados e o resultado final é armazenado em ficheiros no computador. No segundo caso, é utilizado o *HDFS* para armazenar a informação de entrada e de saída do processo de *ETL*. Pelo facto de as duas implementações terem utilizado o mesmo programa (*PDI*) podia-se pensar que converter uma na outra seria um processo simples, mas não é o caso. Devido ao facto das duas implementações pertencerem uma a um ambiente centralizado e outra a uma ambiente distribuído, os componentes do *PDI* que forma utilizados nos dois casos foram os adequados para os dois tipos de ambiente. Por essa razão, converter uma

implementação na outra implica refazer do o processo de forma a alterar os operadores utilizados.

Na implementação sem *MapReduce* o cálculo dos indicadores é efetuado por via de uma interrogação à base de dados e é nessa mesma interrogação que é aplicado qualquer tipo de filtragem e agrupamento aos dados. Utilizando *MapReduce* não é possível fazer essas operações utilizando um único operador, o que acaba por resultar num processo mais complexo envolvendo mais operadores para o cálculo dos mesmos indicadores.

Quando é utilizado o *MapReduce*, é preciso perceber que o processamento dos dados é efetuado em duas fases, *map* e *reduce*, e que tanto a entrada como a saída dessas duas fases tem de estar organizado segundo um par chave/valor. Tendo em conta estas características foi necessário analisar para os dados, e o que se queria calcular e decidir que parte do processamento seria efetuado no *mapper* e que parte é seria efetuado no *reducer*.

Decidiu-se então deixar a cargo do *mapper* a responsabilidade de ler os dados de entrada, identificar os atributos dos voos e definir o par chave/valor que será passado ao *reducer*. A chave passada ao *reducer* é composta pelos atributos sobre os quais se vai querer agrupar os dados e calcular os indicadores. A Figura 4.7 ilustrada, como exemplo, um dos *mappers* implementados. Na definição do *mapper* têm que constar sempre dois operadores que especificam o tipo dos dados que são lidos e o tipo dos dados que são passados ao *reducer* como par chave/valor. Na Figura 4.7 esses operadores são o *MapReduce Input* e o *MapReduce Output*. O segundo operador, intitulado de 'Identificação dos atributos' serve para fazer um parse aos dados de forma a identificar os atributos dos voos. Feita essa identificação passa a ser possível aplicar o critérios de seleção para o cálculo dos indicadores. Os dados que não satisfaçam o critério de seleção são enviados para um operador que não faz nenhum tipo de processamento com o objectivo de apenas retira-los do fluxo do processo. Como última etapa do *mapper* é definido o par chave/valor que será passado ao *reducer*. No exemplo da Figura 4.8, a chave era constituída por um conjunto de atributos (período em que o voo acorreu, companhia aérea, destino e origem) e o valor era o número um uma vez que se queria fazer uma contagem do número de voos.

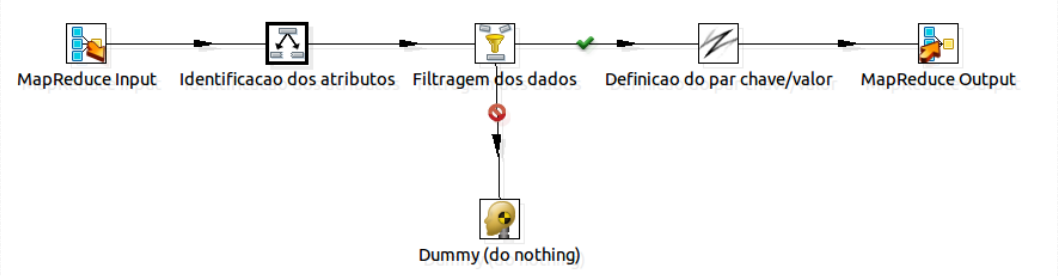

Figura 4.7: Definição do mapper

O *reducer* recebendo esse par chave/valor, agrupa os dados pela chave e aplica a operação necessária para calcular os indicadores (sum, count, max, etc.), conforme ilustrado nas Figuras 4.9 e 4.10 respectivamente. A saída dos vários *reducers* é composta por ficheiros que são armazenados no *HDFS*.

Os programas *MapReduce* presentes nos *jobs* têm de ser configurados com um conjunto de informações.

Step name: Definicao do par chave/valor

Fields:

#	New field	Java expression	Value type	Length	Precision	Replace value
1	new_key	Year + ' ' + Month + ' ' + DayofMonth + ' ' + UniqueCarrier + ' ' + Dest + ' ' + Origin	String			
2	new_value	1	Integer			

Figura 4.8: Definição do par chave/valor

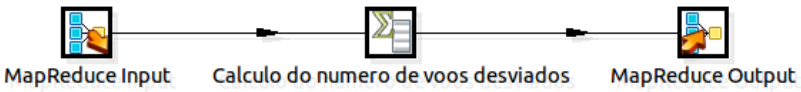

Figura 4.9: Definição do reducer

The fields that make up the group:

#	Group field
1	key

Get Fields

Aggregates :

#	Name	Subject	Type	Value
1	num_voos_desviados	value	Sum	

Get lookup fields

Figura 4.10: Exemplificação das operações efectuadas no *reducer*

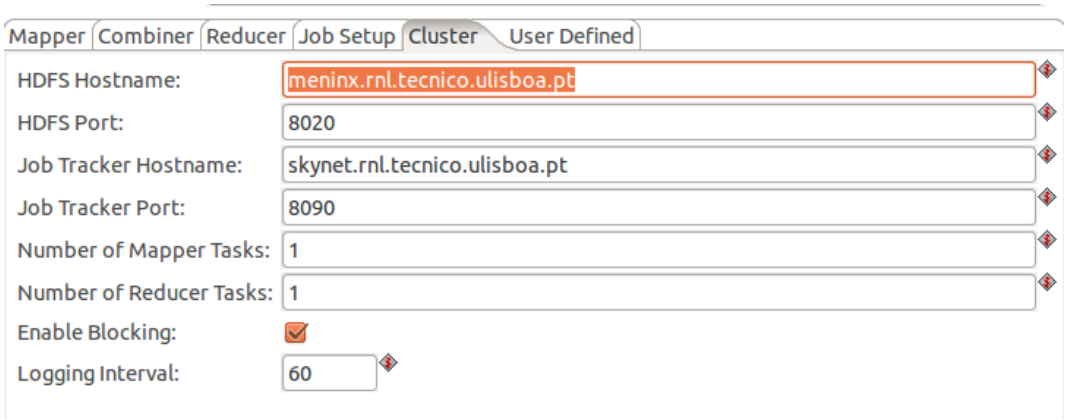

Figura 4.11: Configuração do programa MapReduce

É preciso especificar, para cada programa *MapReduce*, o *mapper* e *reducer* utilizados e ainda fornecer informação sobre o *cluster* onde o *Hadoop* e *MapReduce* estão instalados. A Figura 4.11 ilustra a configuração do *cluster*. É preciso o indicar os nomes e as localizações do *HDFS* e do componente do *Hadoop* responsável por executar um programa *MapReduce*, que na figura são os parâmetros *HDFS Hostname*, *HDFS Port*, *Job Tracker Hostname* e *Job Tracker Port*. Para além desses parâmetros é possível indicar o número de *mappers* e *reducers* por cada programa *MapReduce*.

4.4.3 MapReduce utilizando a linguagem de *scripting* Pig

Sendo o *Pig* uma linguagem de *script* a implementação do processo de *ETL* utilizando esta tecnologia teria de ser necessariamente diferente das duas outras implementações. O cálculo dos indicadores

está dividido em dois *scripts*, um para os indicadores diretos e outro para os indicadores calculados. A linguagem Pig tem algumas semelhanças com *SQL*, na medida em que é possível efetuar operações como junções, agrupamentos, etc.

Tal como na implementação anterior, os dados de entrada têm de ser transferidos para o *HDFS* de forma a poderem ser utilizados pelo *Pig*. As primeiras instruções dos *scripts* consistem em fazer o carregamento dos dados dos voos e aplicar a componente de limpeza dos dados, e carregar a informação relativa às dimensões de análise, conforme é ilustrado nas secções (1) a (7) do código.

```
-- carregamento dos dados relativos aos voos
(1) dados_voos = load '$input' using PigStorage(',') AS (Year:int, Month:int,
DayofMonth:int, DayOfWeek:int, ArrTime:chararray,
CRSArrTime:int, DepTime:chararray, CRSDepTime:int,
UniqueCarrier:chararray, FlightNum:int, TailNum:chararray, ActualElapsedTime:chararray,
CRSElapsedTime:chararray, AirTime:chararray, DepDelay:chararray, ArrDelay:chararray,
Origin:chararray, Dest:chararray, Distance:chararray, TaxiIn:chararray,
TaxiOut:chararray, Cancelled:int, CancellationCode:chararray,
Diverted:int, CarrierDelay:chararray,
WeatherDelay:chararray, NASDelay:chararray, SecurityDelay:chararray,
LateAircraftDelay:chararray);

(2) dados_voos_clean = foreach dados_voos generate Year, Month, DayofMonth,
UniqueCarrier, (int)REPLACE(AirTime,'NA','0') as AirTime_clean,
(int)REPLACE(DepDelay,'NA','0') as DepDelay_clean,
Diverted, Cancelled, Dest, Origin, CancellationCode,
(int)REPLACE(CarrierDelay,'NA','0') as CarrierDelay_clean,
(int)REPLACE(WeatherDelay,'NA','0') as WeatherDelay_clean,
(int)REPLACE(NASDelay,'NA','0') as NASDelay_clean,
(int)REPLACE(SecurityDelay,'NA','0') as SecurityDelay_clean,
(int)REPLACE(LateAircraftDelay,'NA','0') as LateAircraftDelay_clean;

-- carregamento dos dados relativos as dimensoes
(3) periodo_dim = load '/tmp/ist164166/dimensoes/periodo.csv' using
PigStorage(',') AS (Periodo_id:int, Ano:int, Mes:int, Dia:int,
DiaSemana:int, Data:chararray);

(4) carrier_dim = load '/tmp/ist164166/dimensoes/carrier.csv' using
PigStorage(',') AS (Carrier_id:int, CarrierCode:chararray,
Description:chararray);

(5) airport_dim = load '/tmp/ist164166/dimensoes/airport.csv' using
```

```
PigStorage(',') AS (Airport_id:int, IataAirportCode:chararray,
AirportName:chararray, Location:chararray);

(6) delay_dim = load '/tmp/ist164166/dimensoes/delay.csv' using
PigStorage(',') AS (Delay_id:int, Reason:chararray);

(7) cancellation_dim = load '/tmp/ist164166/dimensoes/cancellation.csv'
using PigStorage(',') AS (Cancellation_id:int, Reason:chararray,
Code:chararray);
```

Após o carregamento da informação, é iniciado o cálculo dos indicadores. Para calcular os indicadores é efetuado um *GROUP BY* pelas dimensões de análise e de seguida é aplicada a operação desejada, seja ela um somatório, uma contagem, determinar o máximo, etc. Para além disso, alguns indicadores necessitam que uma determinada filtragem aos dados seja feita. A secções (8), (9) e (10) do código ilustram o cálculo do indicador.

```
-- calculo do indicador numero de voos desviados
(8) Desviados1 = FILTER dados_voos_clean BY Diverted==1;
(9) Desviados2 = group Desviados1 by (Year, Month, DayofMonth,
UniqueCarrier, Dest, Origin);
(10) Desviados3 = foreach Desviados2 generate flatten($0),
COUNT($1) as numvoosdesviados;
```

As instruções seguintes têm como objectivo obter os identificadores das dimensões de análise, e armazenar os indicadores no *HDFS*.

```
(11) Desviados4 = join Desviados3 by ($0,$1,$2), periodo_dim by(Ano,Mes, Dia);
(12) Desviados5 = foreach Desviados4 generate Periodo_id,$3,$4,$5,
numvoosdesviados;

(13) Desviados6 = join Desviados5 by ($1), carrier_dim by (CarrierCode);
(14) Desviados7 = foreach Desviados6 generate Periodo_id, Carrier_id, $2, $3,
numvoosdesviados;

(15) Desviados8 = join Desviados7 by ($2), airport_dim by (IataAirportCode);
(16) Desviados9 = foreach Desviados8 generate Periodo_id, Carrier_id,
Airport_id as Dest, $3, numvoosdesviados;

(17) Desviados10 = join Desviados9 by ($3), airport_dim by (IataAirportCode);
(18) Desviados11 = foreach Desviados10 generate Periodo_id, Carrier_id, Dest,
Airport_id as Origin, numvoosdesviados;
```

```
(19) Desviados12 = ORDER Desviados11 BY Periodo_id, Carrier_id, Dest, Origin;
(20) store Desviados12 into '/tmp/ist164166_2/indicadores_directos/desviados';
```

Ao contrário da implementação do processo de *ETL* utilizando o PDI com *MapReduce*, não foi necessário fazer a divisão do processamento dos dados pelas duas fases do *MapReduce*, uma vez que, o próprio *Pig* encarrega-se de fazer essa divisão. Dessa forma o utilizador tem apenas de se preocupar com as operações que são aplicadas aos dados, cabendo ao *Pig* fazer a gestão das operações executadas no *mapper* e no *reducer*.

Capítulo 5

Validação

No Capítulo 5 é feita a validação dos processos de *ETL* desenvolvidos e apresentados no capítulo anterior. O capítulo começa por apresentar a infra-estrutura que foi utilizada e que serviu de suporte para o desenvolvimento do trabalho. De seguida são detalhados os resultados obtidos nas diferentes implementações do processo que foram desenvolvidas.

5.1 Infraestrutura Utilizada

Como infraestrutura de suporte à realização do trabalho, foi utilizado o *cluster Hadoop* pertencente à RNL do Instituto Superior Técnico. Esta rede dispõe atualmente de uma infraestrutura moderna, flexível, dentro dos condicionalismos de fiabilidade e segurança normais nestes ambientes e é composta por um parque de máquinas distribuído por vários laboratórios com acesso a software especifico para as atividades lectivas. O *cluster* encontra-se a executar nas *workstations* dos laboratórios que utilizam Debian 7 64bits. Dependendo das horas do dia e da atividade dos laboratórios, estão disponíveis entre 40 a 360 *cores*, com cerca de 2GB de RAM por *core*. O *Hadoop* encontra-se disponível em 90 nós do *cluster*. Ao todo existe um máximo possível de 360 *slots* distribuídos com o mesmo peso para *Map* e *Reduce*. Os computadores do *cluster* possuem as seguintes características:

- Grupo 1: 60 computadores
  - Processador Intel Core i5 760 2.80ghz (Quad-core);
  - RAM 8GB (8192MB);
  - Sistema Operativo Debian 7.0/Windows 7;
  - HDD 500 GB.
- Grupo 2: 30 computadores
  - Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz (Quad-core);
  - RAM 8GB (8192MB);
  - Sistema Operativo Debian 7.0/Windows 7.

As versões do *Hadoop* e *Pig* utilizadas foram respectivamente *HadoopVersion* 2.2.0 e *PigVersion* 0.14.0. Para os testes com Pentaho sem *MapReduce* foi utilizado um computador do Grupo 1. Foi também usado o seguinte conjunto de software:

- Pentaho (kettle Spoon) 5.4.0.1-130;
- MySQL Workbench 5.2.40;
- MySQL 5.5.43 for Debian-linux-gnu.

5.2 Descrição dos dados utilizados e métricas de validação

Para esta dissertação foram utilizados dados públicos com detalhes sobre a partida e chegada de voos comerciais realizados nos Estados Unidos. Como mencionado no Capítulo 4 os dados estão divididos em três ficheiros, informação sobre os voos, informação sobre as companhias aéreas e por fim informação sobre os aeroportos.

O conjunto de dados de teste contém aproximadamente 120 milhões de registos no total, com um espaço ocupado de 1.6 gigabytes quando comprimido e 12 gigabytes sem estar comprimido. No trabalho que se pretende desenvolver, foram usados os seguintes critérios de validação:

- Desempenho: verificar o desempenho dos processos implementados mediante um volume de dados crescente;
- Comparar as tecnologias utilizadas nos três processos mediante a sua facilidade de utilização;
- Facilidade de desenvolvimento.

5.3 Experiências

Os testes realizados com as três implementações do processo de *ETL* foram divididos em duas partes. A diferença dos dois conjuntos de experiencias reside no número de implementações do processo utilizadas e na utilização de uma componente de limpeza de dados cujo objectivo foi substituir dados não disponíveis (NA) por um valor *null*. Em cada um dos testes foram calculados um conjunto de indicadores diretos e um conjunto de indicadores calculados. As subsecções seguintes apresentarão com mais detalhe os resultados obtidos.

5.3.1 Resultados

Processo sem a componente de limpeza de dados

Numa primeira experiência, foi medido o tempo de execução dos processos de *ETL* implementados envolvendo operações de transformação, agrupamento e cálculo dos indicadores, e descartando o tempo de armazenar os dados de entrada na base de dados e no *HDFS*. Com o objectivo de testar a

escalabilidade dos três processos foi medido o tempo de execução para diferentes volumes de dados. Os resultados são apresentados na Tabela 5.1 e nas Figuras 5.1 e 5.2, onde é ilustrado o tempo de execução que se obtém para os diferentes volumes de informação.

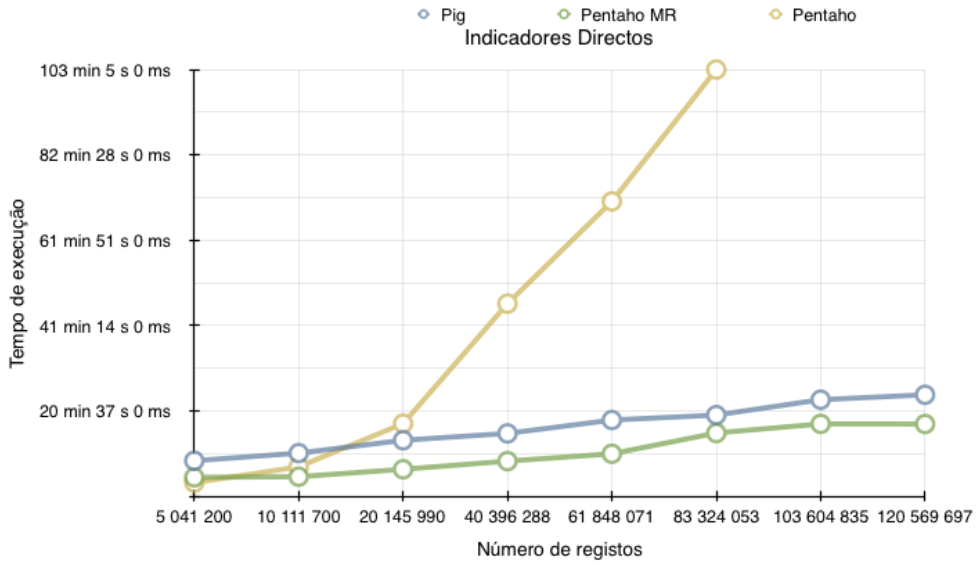

Figura 5.1: Primeira experiência Indicadores Directos

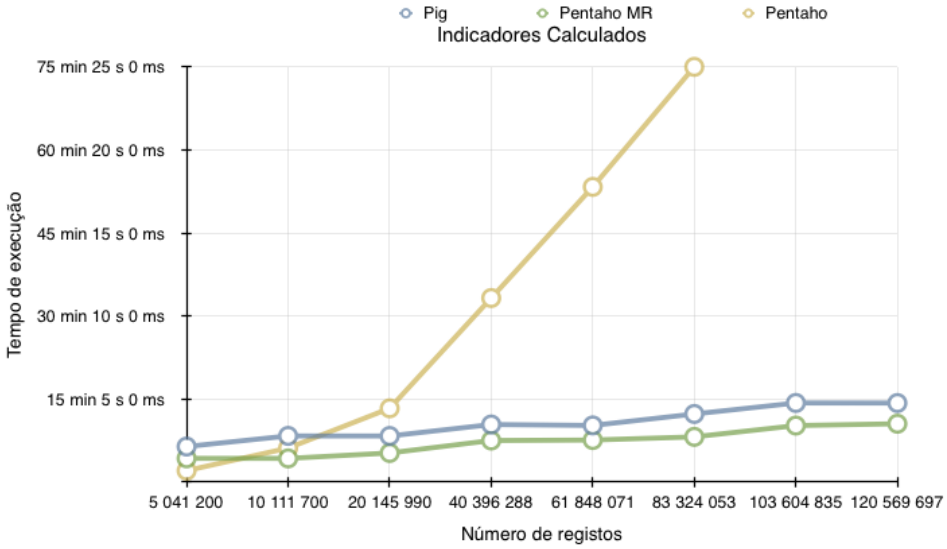

Figura 5.2: Primeira experiência Indicadores Calculados

Analisando os resultados podemos ver que os dois processos que beneficiam do ambiente distribuído (*Pig* e *Pentaho* com *MapReduce*) apresentam melhores resultados à medida que se vai escalando os dados. Por outro lado, o processo que usa um ambiente centralizado (*Pentaho* sem *MapReduce*), teve um resultado pior em comparação com os outros dois processos, uma vez que aqui não é possível tirar proveito do processamento distribuído dos dados. Para volumes de dados mais pequenos (5 e 10 milhões de registos) os dois processos que utilizam o *Pentaho* apresentam melhores resultados que o *Pig*. Conclui-se que, para volumes de dados até essa ordem, a utilização de um ambiente

distribuído não traz grandes vantagens sobre um ambiente centralizado, uma vez que o processo que usa o *Pentaho* sem *MapReduce* consegue ter bons tempos de execução. O ponto de quebra acontece quando o volume de dados atinge a casa dos 20 milhões de registos. A partir daqui a utilização de um ambiente distribuído apresenta melhores resultados como é facilmente visível. O *Pentaho* sem *MapReduce* tem muitas dificuldades em processar volumes de dados maiores num tempo aceitável. Entre os dois processos que usam *MapReduce* apesar de ambos apresentarem bons resultados, os melhores tempos de execução são obtidos pelo *Pentaho*.

Tabela 5.1: Comparação dos resultados obtidos - Indicadores Directos.

	Pig Indicadores Directos	**Pentaho Indicadores Directos (com MR)**	**Pentaho Indicadores Directos (sem MR)**	**Pig Indicadores Calculados**	**Pentaho Indicadores calculados (com MR)**	**Pentaho Indicadores Calculados (sem MR)**
5 041 200	8 min 36 s 534 ms	4 min 36,534 s	3 min 19 s	6 min 27 s 588 ms	4m 19s	2 min 5 s
10 111 700	10 min 27 s 417 ms	4 min 45,6 s	7 min 8 s	8 min 23 s 441 ms	4 min 18,4 s	6 min 11 s
20 145 990	13 min 33 s 557 ms	6 min 33,2 s	17 min 35 s	8 min 22 s 778 ms	5 min 17,6 s	13 min 24 s
40 396 288	15 min 12 s 681 ms	8 min 31 s	46 min 33 s	10 min 27 s 691 ms	7 min 33,4 s	33 min 26 s
61 848 071	18 min 26 s 538 ms	10 min 17,8 s	71 min 15 s	10m 17s 636ms	7 min 38,6 s	53 min 37 s
83 324 053	19 min 38 s 523 ms	15 min 19 s	103 min 5 s	12 min 22 s 547 ms	8 min 13,2 s	75 min 25 s
103 604 835	23 min 21 s 248 ms	17 min 30,823 s		14 min 22 s 523 ms	10 min 16,2 s	
120 569 697	24 min 32 s 665 ms	17 min 28,8 s		14 min 21 s 646 ms	10 min 36,2 s	

Processo com a componente de limpeza de dados

Numa segunda experiencia, foi considerado apenas os dois processos que usam o *MapReduce* e foi introduzido nesses processos uma componente de limpeza de dados, que substitui dados não disponíveis (NA) por um valor *null*. O processo que usa o Pentaho sem *MapReduce* foi descartado nesta segunda fase, uma vez que já não fazia sentido continuar a testar esse processo depois de analisar os resultados dos primeiros testes. Para este segundo conjunto de experiências o número de registos utilizados foi aumentado à custa da duplicação dos dados mas fomos sempre mudando o ano dos registos de forma a não ter dados com a mesma dimensão temporal.

Os resultados obtidos estão apresentados na Tabela 5.2 e nas Figuras 5.3 e 5.4. O tempo medido foi o tempo de execução dos processos envolvendo operações de transformação, limpeza, agrupamento, cálculo dos indicadores e armazenamento dos dados no *HDFS*. A ideia inicial era escalar os dados até 1 bilião de registos, mas tal não foi possível porque a execução dos *jobs* no *Pentaho* teve muitos problemas. As vezes para poder ter uma execução com sucesso era preciso várias tentativas. E isso estava a acontecer porque a carga colocada nos *reducers* era tão grande que estes esgotavam a memória RAM da máquina onde estavam a correr e iam abaixo, o que acabava por fazer com que os *jobs* ou falhassem ou demorassem muito tempo a executar.

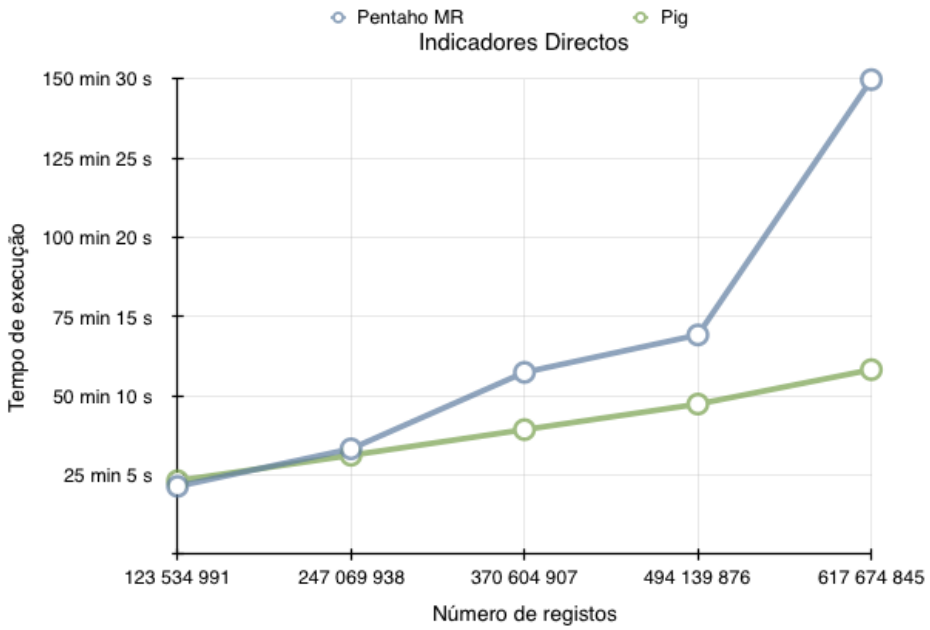

Figura 5.3: Segunda experência Indicadores Directos

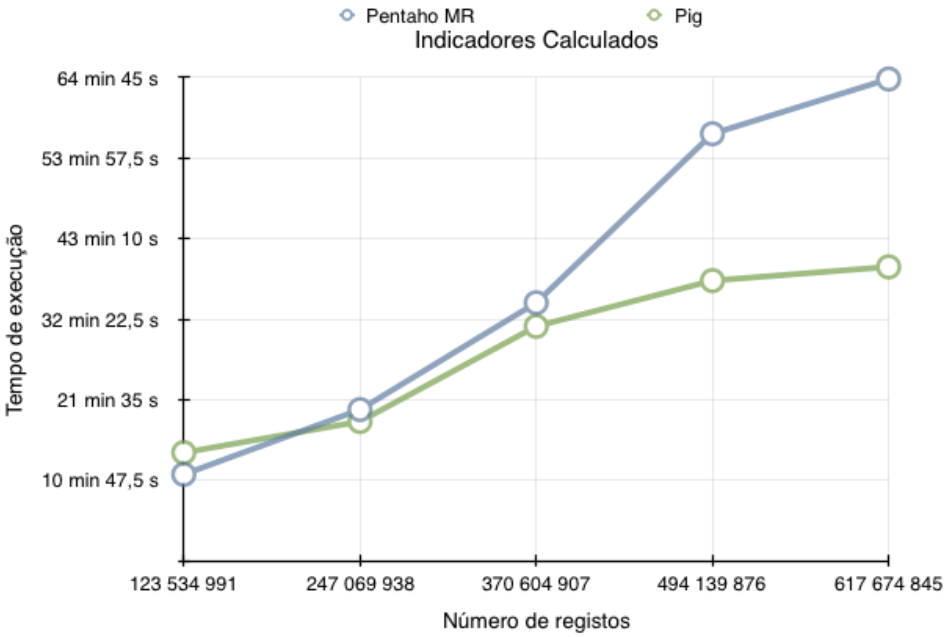

Figura 5.4: Segunda experiência Indicadores Calculados

As pessoas da RNL com quem estive em contacto também acharam que os processos estavam a falhar porque ao esgotar a memória da máquina, o sistema operativo matava automaticamente os processos que usavam mais memória. Por essa razão só foi possível escalar os dados até aos 617.674.845 de registos.

Analisando os resultados podemos concluir que o *Pig* teve melhores tempos que o *Pentaho* com *MapReduce*, muito por culpa do problema acima mencionado. O *Pentaho* utilizava muito mais *mappers* que o *Pig* mas apenas um *reducer*, e isso fazia com que o *reducer* ficasse sobrecarregado com toda a

informação que saía dos *mappers*. Já o *Pig* fez um balanceamento melhor de *mappers* e *reducers* do que o *Pentaho*. O *Pig* conseguiu utilizar uma quantidade de *mappers* e *reducers* suficiente de forma a poder fazer todo o processamento sem problemas. O *Pig* determina a quantidade de *mappers* e *reducers* a serem utilizados tendo em conta a quantidade de dados processados por cada instrução do *script*. Outra diferença entre o *Pig* e o *Pentaho*, é que no *Pig* para além do numero de *mappers* e *reducers* ser determinado levando em consideração a quantidade de dados, as instruções que usam *reducers* pertencem a um grupo restrito de instruções (COGROUP, CROSS, DISTINCT, GROUP, JOIN (inner), JOIN (outer), and ORDER.). As restantes instruções usam apenas *mappers* fazendo com que o processamento feito nessas instruções seja mais rápido. No *Pentaho* o processamento dos dados tinha sempre as duas fases, *map* e *reduce*.

Tabela 5.2: Comparação dos resultados obtidos - Indicadores Calculados.

	Pig Indicadores Directos	**Pentaho Indicadores Directos**	**Pig Indicadores Calculados**	**Pentaho Indicadores calculados**
123 534 991	23 min 20 s 375 ms	21 min 33 s	14 min 29 s 375 ms	11 min 34 s
247 069 938	31 min 23 s 481 ms	33 min 26 s	18 min 39 s 557 ms	20 min 15 s
370 604 907	39 min 30 s 523 ms	57 min 38 s	31 min 24 s 639 ms	34 min 34 s
494 139 876	47 min 35 s 642 ms	69 min 29 s	37 min 32 s 465 ms	57 min 13 s
617 674 845	58 min 29 s 506 ms	150 min 18 s	39 min 22 s 614 ms	64 min 33 s

Capítulo 6

Conclusões

Esta dissertação apresentou um estudo experimental e um conjunto de testes, sobre um processo de *ETL*, nos quais foram abordadas diferentes tecnologias com o objectivo de aferir em que condições é mais vantajoso o uso de tecnologias de *Big Data* para o processamento de dados. As tarefas específicas consideradas nesta tese de mestrado foram:

- Desenvolver uma implementação do processo *ETL* no *Pentaho*, utilizando operadores convencionais de *ETL*;
- Desenvolver uma implementação do processo *ETL* no *Pentaho*, utilizando operadores de *MapReduce*;
- Desenvolver uma implementação do processo *ETL* utilizando a linguagem de *scripting Pig*.

O objectivo passava por perceber a partir de que volume de dados podíamos tirar melhor proveito do uso de tecnologias que utilizam o *MapReduce* como mecanismo de processamento de dados. Tendo em conta o tipo e volume de dados que foram utilizados para esta dissertação existe de facto um ponto de quebra a partir do qual torna-se impraticável a não utilização de tecnologias de *Big Data*, uma vez que o tempo de processamento começa a ser cada vez maior.

Nas secções seguintes são apresentados os aspectos qualitativos do desenvolvimento das três implementações do processo ETL, as dificuldades sentidas na instalação do PDI e por último são apresentadas algumas diretrizes que podem vir a ser desenvolvidas no futuro.

6.1 Apectos qualitativos do desenvolvimento do processo ETL utilizando as diferentes alternativas

6.1.1 Pentaho Data Integration

O *PDI* oferece uma interface gráfica com um *drag-and-drop* bastante intuitivo, onde os utilizadores têm ao seu dispor uma vasta gama de operadores que lhes permitem construir grafos de transformações.

Apesar de ter um conjunto de operadores pré-definidos, o *PDI* permite utilizar código feito pelo utilizador o que enriquece e estende a funcionalidade do processo *ETL*.

Os processos de *ETL* que foram implementados no *Pentaho* diferem um do outro por causa do ambiente onde estão inseridos, um num ambiente centralizado e outro num ambiente distribuído. A utilização de *MapReduce* e do *HDFS* obrigam a refazer todo o processo porque os operadores utilizados têm de ser os adequados para este tipo de ambiente. Com *MapReduce* o processo torna-se mais complexo, na medida em que é preciso definir o processamento do *mapper* e do *reducer* e configurar o acesso ao *cluster* onde o *Hadoop* está instalado. Não utilizando *MapReduce* o processo torna-se mais simples, uma vez que, o cálculo dos indicadores é feito num único operador.

O *PDI* é uma boa solução para utilizadores pouco experientes, é fácil de usar, e fornece a capacidade de se conectar com diferentes fontes de dados tanto para entrada como para saída. A meu ver uma grande desvantagem da utilização do *PDI* com *MapReduce* vem do facto de nem sempre ser fácil transformar os problemas em pares chave/valor, e estabelecer que conjunto de transformações é feito no *mapper* e no *reducer*.

O *Pig* por sua vez faz a sua própria gestão das instruções fazendo com que não seja necessário o utilizador indicar que instruções são executadas no *mapper* e no *reducer*.

6.1.2 Pig

Trabalhar com o *Pig* é diferente do que trabalhar com o *PDI* na medida que não temos ao nosso dispôr uma interface gráfica. O que temos é um editor de texto onde escrevemos o código necessário para as operações que queremos realizar. Ao contrário do processo especificado no *PDI* que utiliza *MapReduce*, aqui não temos de nos preocupar em definir que conjunto de operações são feitas no *mapper* e no *reducer*, pois o próprio *Pig* faz essa divisão. Mas tal como o *PDI*, o *Pig* também permite enriquecer e estender as funcionalidades do processo *ETL* através de código feito pelo utilizador.

Para quem está habituado a linguagens como *SQL* irá encontrar no *Pig* algumas semelhanças, uma vez que este oferece muitas das operações *standard* que normalmente fazemos com os dados (Join, Count, Group, etc.), o que ajuda no processo de adaptação. De qualquer forma para utilizar o *Pig* é necessário aprender a linguagem de *script Pig Latin*, o que, para utilizadores sem experiência, pode implicar uma aprendizagem mais lenta.

6.2 Instalação do Pentaho

O *PDI* suporta diferentes versões de distribuições *Hadoop* e vem equipado com configurações para algumas distribuições como Cloudera, Hortonworks e MapR. De forma a suportar todas essas versões, o *Pentaho* usa uma camada de abstração, denominada *shim*, que se conecta com as diferentes distribuições do *Hadoop*. Uma *shim* é uma biblioteca que intercepta chamadas de API e redireciona-as, manipula-as ou altera os parâmetros das chamadas. Periodicamente, o *Pentaho* desenvolve novas *shims* à medida que novas distribuições *Hadoop* vão sendo desenvolvidas.

A RNL usa a distribuição da Apache. A maior dificuldade sentida estava no facto do *PDI* não vir com uma *shim* para as versões mais recentes da distribuição da Apache. A *shim* para Apache que vem por omissão com o *PDI* diz respeito a uma versão bastante antiga do *Hadoop* e o próprio *Pentaho* não tem suporte para *shims* de versões mais recentes da distribuição *Hadoop* da Apache. A forma indicada para resolver esse problema, é ser o utilizador a desenvolver uma *shim* modificando uma de outra distribuição que melhor se assemelha com a distribuição da Apache. No final a solução encontrada acabou por não implicar ter de desenvolver uma *shim* de raiz. O que foi necessário fazer foi duplicar a *shim* da distribuição da Cloudera que já vem com o *PDI*, e acrescentar nessa *shim* alguns ficheiros que se encontram localizados no diretório de instalação do *Hadoop*. Feito essas modificações foi possível colocar o *PDI* a funcionar com o *Hadoop* e assim poder executar o processo de *ETL* que tinha sido desenvolvido.

6.3 Trabalho Futuro

Apesar de termos atingido resultados interessantes, há algumas ideias a considerar em termos de trabalho futuro. Um caminho interessante a seguir seria poder realizar os testes não só com mais do que um conjunto de dados e com mais processos de *ETL* desenvolvidos em outras tecnologias, mas também num *cluster* maior e onde pudéssemos ter a totalidade dos nós disponíveis para os testes.

Os dados usados foram dados estruturados onde a informação já vinha organizada. Ter dados não estruturados e que implicassem uma maior componente de tratamento e limpeza de informação seria um ótimo teste para complicar o processo *ETL* e tirar um conjunto diferente de conclusões.

Bibliografia

Agrawal, Divyakant, Bernstein, Philip, Bertino, Elisa, Davidson, Susan, & Dayal, Umeshwas. 2011. Challenges and Opportunities with Big Data. *Cyber Center Technical Reports,Purdue University, Purdue e-Pubs*.

Chen, Min, Mao, Shiven, & Liu, Yunhao. 2014. Big Data: A Survey. *Mobile Netw Appl*.

Dean, Jeffrey, & Ghemawat, Sanjay. 2004. MapReduce: Simplified Data Processing on Large Clusters. *Symposium on Operating System Design and Implementation*.

Gates, Alan. 2011. *Programming Pig*. O'Reilly Media, Inc.

Hurwitz, Judith, Nugent, Alan, Halper, Fern, & Kaufman, Marcia. 2013. *Big Data For Dummies*. Wiley.

Jamack, Peter J. 2014. Hive as a tool for ETL or ELT. *IBM developers works*.

Kimball, Ralph, & Caserta, Joe. 2004. *The Data Warehouse ETL Toolkit*. Wiley Publishing, Inc.

Lane, Paul. 2002. Oracle9i Data Warehousing Guide. *Oracle Corporation*.

Liu, Xiufeng, Thomsen, Christian, & Pedersen, Torben Bach. 2011. ETLMR: A Highly Scalable Dimensional ETL Framework based on MapReduce. *Proceedings of 13th International Conference on Data Warehousing and Knowledge, Toulouse, France*.

Liu, Xiufeng, Thomsen, Christian, & Pedersen, Torben Bach. 2012. CloudETL: Scalable Dimensional ETL for Hive. *Department of Computer Science, Aalborg University, 1DB Technical Report*.

Rud, Olivia. 2009. *Business Intelligence Success Factors: Tools for Aligning Your Business in the Global Economy*. John Wiley and Sons, Inc.

Thomsen, C., & Pedersen, T. B. 2009. pygrametl: A Powerful Programming Framework for Extract-Transform-Load Programmers. *In Proc. of DOLAP*.