Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Proposta de Arquitetura NoLAP para um Sistema de Apoio à Decisão Acadêmico

Rodrigo da Fonseca Silveira

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Orientador
Prof. Dr. Marcio de Carvalho Victorino

Brasília
2020

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

da Fonseca Silveira, Rodrigo
dF676p Uma proposta de arquitetura NoLAP para um Sistema de Apoio à Decisão Acadêmico / Rodrigo da Fonseca Silveira; orientador Marcio Victorino. -- Brasília, 2020.
78 p.

Dissertação (Mestrado - Mestrado Profissional em Computação Aplicada) -- Universidade de Brasília, 2020.

1. Sistemas de Apoio à Decisão. 2. Bancos de Dados NoSQL. 3. Big Data. I. Victorino, Marcio, orient. II. Título.

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Proposta de Arquitetura NoLAP para um Sistema de Apoio à Decisão Acadêmico

Rodrigo da Fonseca Silveira

Dissertação apresentada como requisito parcial para conclusão do
Mestrado Profissional em Computação Aplicada

Prof. Dr. Marcio de Carvalho Victorino (Orientador)
Faculdade de Ciência da Informação

Prof.a Dr.a Maristela Terto de Holanda
Departamento de Ciência da Computação

Prof. Dr. Gibeon Soares de Aquino Junior
Departamento de Informática - DIM/UFRN

Prof. Dr. Marcelo Ladeira
Coordenador do Programa de Pós-graduação em Computação Aplicada

Brasília, 17 de dezembro de 2020

Dedicatória

Dedico esse trabalho aos meus pais, Francisco e Iracema, que durante toda a vida se esforçaram muito para que eu pudesse chegar a esse momento de ter condições de apresentar uma dissertação de conclusão do Mestrado Profissional em Computação Aplicada, à minha irmã, Vanessa, ao meu sobrinho, Lucas, aos meus avós, Mariana e João, e aos meus tios e primos que formam a estrutura da minha base familiar e que sempre torceram muito por mim. Também dedico à minha companheira Janaína, que sempre me deu o suporte, a motivação e a paciência necessárias para que eu conseguisse superar todos os desafios inerentes ao curso. Por fim, e não mais importante, dedico ao meu inesquecível primo Marcos Fonseca, criado como um irmão e o amigo mais presente em boa parte da minha vida, e que infelizmente nos deixou no dia 3 de outubro de 2020, restando muitas saudades.

Agradecimentos

Gostaria de agradecer ao meu orientador Marcio de Carvalho Victorino, por ter me acompanhado desde o início nessa jornada, não só com orientações técnicas, mas também com palavras de incentivo e companheirismo nos momentos mais difíceis, e à professora Maristela, por ter me permitido fazer parte de seu grupo de pesquisa, o que me ajudou muito na minha ainda inicial vida acadêmica de pesquisador. Também gostaria muito de agradecer aos meus colegas da linha de Ciência dos Dados por dividirem comigo seus conhecimentos, dicas e apoio moral em todas as situações, assim como também agradeço aos demais professores do curso de Pós-Graduação em Computação Aplicada pela seriedade, respeito pelos alunos e por nos exigirem o máximo de dedicação e empenho para o cumprimento das disciplinas.

Resumo

Este trabalho tem por objetivo apresentar uma proposta de migração da arquitetura do *Data Warehouse* de dados acadêmicos da Universidade de Brasília - UnB, desenvolvido em bancos de dados relacionais, conhecido na literatura como arquitetura ROLAP - *Relational On Line Analytical Processing* , para uma abordagem em bancos de dados *NoSQL*, mais precisamente para bancos de dados *NoSQL* de família de colunas. As abordagens consideradas neste trabalho levaram em consideração o que se observou de mais relevante no estado da arte da literatura relacionada ao tema, como migrações de sistemas de *Data Warehouse* para os bancos de dados de família de colunas, como HBase, por exemplo, em conjunto com soluções para o processamento de grandes volumes de dados em um *cluster* de servidores, como *Apache Hadoop*. Essa migração parte da necessidade de serem realizados estudos de novos paradigmas de arquiteturas para o Sistema de Apoio à Decisão Acadêmico face à nova realidade dos problemas que surgiram pelo crescimento massivo do volume de dados gerados pelos sistemas de informação da Universidade de Brasília - UnB.

Palavras-chave: NoSQL, NoLAP, Data Warehouse, Big Data

Abstract

This research aims to propose a migration project to the Data Warehouse of the University of Brasilia - UnB , developed in a relational database, that is, *ROLAP architecture* to a column family NoSQL database architecture. The approache envolved in this work was considered by researches from the literature state of the art about migrations from Data Warehouse systems to column family databases, such as *HBase*, with solutions for large volumes of data processing on a server cluster, such as *Apache Hadoop*. This migration project emerged from the need of studying new architectural paradigms for the Academic Data Warehouse due to problems raised by massive growth of data volume stored in the University of Brasilia relational databases.

Keywords: NoSQL, NoLAP, Data Warehouse, Big Data

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

DW Data Warehouse.

ETL Extract, Transform and Load.

JSON JavaScript Object Notation.

LAI Lei de Acesso à Informação.

MOLAP Multidimensional Online Analytical Processing.

NoLAP NoSQL Online Analytical Processing.

OLAP OnLine Analytical Processing.

OLTP Online Transaction Processing.

ROLAP Relational Online Analytical Processing.

SAD Sistema de Apoio à Decisão.

SEI Sistema Eletrônico de Informações.

SQL Structured Query Language.

STI Secretaria de Tecnologia da Informação.

TEMAC Enfoque Meta Analítico Consolidado.

UnB Universidade de Brasília.

XML Extensible Markup Language.

Capítulo 1

Introdução

1.1 Definição do Problema

A Secretaria de Tecnologia da Informação - STI - tem como missão viabilizar soluções de tecnologia da informação que promovam a disponibilidade, integridade, confiabilidade e autenticidade das informações dos ativos relacionados aos sistemas informatizados da Universidade de Brasília *(http://www.sti.unb.br/)*. Esta secretaria desenvolve as atividades de caráter permanente de apoio, necessárias ao desenvolvimento do ensino, da pesquisa e da extensão no que se refere ao processamento de dados da universidade.

A STI UnB já desenvolveu vários sistemas operativos informatizados nas diversas áreas administrativas, de ensino, de pesquisa e de extensão. Tais sistemas geram e armazenam uma expressiva quantidade de dados relacionados aos processos, porém esses dados não estão relacionados entre si, impedindo uma análise conjunta em função da falta de integração destes dados.

Com o objetivo de se minimizar essa limitação, a STI iniciou, em 2018, o desenvolvimento de um Sistema de Apoio à Decisão - SAD - denominado BI-UnB . Esse sistema está organizado de acordo a arquitetura OLAP proposta por Kimball [1], constituída basicamente por quatro camadas:

1. Camada composta por fontes de dados
2. Camada de Extração de Dados, Transformação e Carga (ETL)
3. Camada de armazenamento
4. Camada de apresentação

Nessa arquitetura, os dados oriundos dos sistemas fontes passam pelos processos de extração e transformação para serem carregados em repositórios multidimensionais deno-

minados *Data Warehouse - DW*. O objetivo é integrar dados provenientes de várias fontes e transformá-los em informação [1].

O software utilizado para implementar o DW foi o Sistema Gerenciador de Banco de Dados Relacional - SGBDR - Postgres. No entanto, com o aumento do volume das bases de dados da Universidade de Brasília, principalmente após a implantação do sistema de acesso eletrônico de documentos - SEI - em março de 2016, a tecnologia relacional poderá não ser capaz de apresentar um desempenho satisfatório em função desse crescimento massivo do volume de dados armazenados. Por outro lado, soluções para armazenamento de dados distribuídos, como os bancos de dados NoSQL e frameworks como *Hadoop*, apresentam-se como um recurso para o armazenamento massivo de dados altamente escalonáveis, tornando-se uma alternativa aos bancos de dados relacionais em determinadas situações [2]. Ou seja, para estender o conceito da abordagem tradicional ROLAP (arquitetura OLAP em bancos relacionais), surge também a arquitetura OLAP em bancos NoSQL - NoLAP.

Em relação ao aumento do volume das bases de dados da UnB, este foi refletido no tamanho, em número de linhas armazenadas, da tabela de Fatos do *SAD* desenvolvido :

1. **2018** - aproximadamente 7.000.000 linhas
2. **2019** - aproximadamente 13.000.000 linhas
3. **2020** - aproximadamente 22.000.000 linhas

Além desse aumento do volume de armazenamento, desde a sanção em 18 de novembro de 2011 da Lei 12.257/2011, que ficou conhecida como Lei de Acesso a Informação -LAI -, foi regulamentado o direito ao acesso às informações públicas governamentais a todos os cidadãos. Nesse contexto, foi estabelecida a política de publicação de dados abertos pelo decreto Nº 8.777, de 11 de maio 2016 [3]. A LAI permite o acesso a um grande volume de dados gerados pelas organizações públicas por parte dos gestores das organizações públicas ou privadas e também da população. A partir do marco de criação dessa lei, houve um aumento do número de requisições de extrações de dados à STI.

Diante do exposto, este trabalho tem por objetivo propor a substituição do Postgres em uma arquitetura OLAP tradicional, denominada arquitetura ROLAP, por um Data Warehouse de arquitetura não convencional, NoLAP, com o propósito de atender à necessidade de armazenamento massivo de dados do BI-UnB, e com isso possibilitar a análise integrada de dados gerados pela universidade de forma rápida, prática e altamente escalonada [2]. Entende-se por *convencional* neste trabalho a arquitetura de *Data Warehouse* que surgiu como conceito acadêmico ainda no final da década de 80 e início dos anos 90 [4], que utilizava bancos de dados relacionais.

A necessidade dessa nova abordagem de migração parte da nova realidade enfrentada pelos Sistemas de Apoio à Decisão da Universidade de Brasília face aos possíveis problemas gerados pelo aumento constante do volume das bases de dados, já que o modelo lógico adotado não se adapta automaticamente ao novo ambiente com esse novo volume massivo de informações [5].

1.2 Justificativa

Atualmente, os gestores da UnB não são capazes de realizar uma análise integrada das várias áreas que compõem a universidade de maneira otimizada por meio de uma única ferramenta. Além disso, há diversos pedidos de extrações de dados direcionados à STI, tanto embasados na LAI e direcionados a obterem dados para pesquisas, quanto originados dos diversos departamentos para o atendimento de demandas, como o Censo Escolar do Ministério da Educação, por exemplo. Na maior parte dos casos, essas demandas são respondidas por meio do envio de planilhas eletrônicas para posteriormente os próprios interessados analisarem e tirarem suas conclusões.

As extrações de dados são feitas atualmente diretamente nos bancos de dados relacionais acadêmicos, e não raramente são utilizados códigos complexos na linguagem *SQL* que resultam em dados brutos, ou seja, que ainda precisam ser computados ou derivados para serem transformados em informação. Isso ocorre devido ao fato de os sistemas legados da universidade serem OLTP - *Online Transaction Processing*, ou seja, são utilizados para as operações de inserção, atualização e exclusão de dados, mas não são apropriados para a análise.

Como parte da motivação, seguem algumas quantidades de pedidos de extrações de dados para a STI a partir de 2012 :

1. 2012 - 35 pedidos de extrações
2. 2013 - 59 pedidos de extrações
3. 2015 - 101 pedidos de extrações
4. 2016 - 154 pedidos de extraçoes
5. 2017 - 160 pedidos de extrações
6. 2018 - 165 pedidos de extrações
7. 2019 - 197 pedidos de extrações

A seguir, alguns exemplos de sistemas acadêmicos não integrados disponibilizados pela STI em 2020:

1. Sistema de Informações Acadêmicas de Alunos de Graduação: SIGRA
2. Sistema de Informações Acadêmicas de Alunos de Pós-Graduação Stricto Sensu: SIPPOS
3. Sistema de Auxílio Estudantil (SAE)
4. Sistema de Pesquisa e Iniciação Científica (Sipic)
5. Sistema Integrado de Gestão de Pessoas - Dados funcionais de servidores (técnicos e docentes)

Diante do exposto, faz-se necessária uma nova proposta de arquitetura para atender à necessidade de armazenamento massivo de dados e com isso possibilitar a análise integrada de dados gerados pela UnB de forma rápida, prática e altamente escalonada [2], não só pelo aumento observado do tamanho das bases, mas também pelo aumento considerável do número de requisições por extrações de dados. Essa arquitetura proposta servirá como ponto central de integração dos diversos sistemas OLTP disponibilizados pela STI, além de possibilitar um escalonamento horizontal da arquitetura por meio das soluções de bancos de dados NoSQL [2]. Essa estrutura permitirá uma análise integrada das informações provenientes de várias áreas da UnB, além de permitir uma resposta mais rápida às demandas de requisições de dados feitas por departamentos, pesquisadores e cidadãos em geral. Para grandes volumes de armazenamento, os bancos de dados NoSQL são uma abordagem otimizada para gerenciamento de dados massivos e distribuídos.

1.3 Pergunta de pesquisa

Como proporcionar à comunidade uma análise integrada de dados acadêmicos da UnB de forma satisfatória quando a solução atual em SGBD Relacional começar a refletir uma perda de desempenho nas respostas às consultas em função do aumento do volume de dados gerados ?

1.4 Hipótese de Pequisa

Uma solução distribuída de banco de dados NoSQL com uma abordagem baseada em processamento de Big Data poderia melhorar o desempenho da entrega de informações gerenciais à comunidade acadêmica e garantir uma maior disponibilidade em relação ao crescimento dos dados ao longo dos próximos anos.

1.5 Objetivo Geral

Este trabalho tem por objetivo propor uma arquitetura não convencional para um Sistema de Apoio à Decisão - SAD, ou seja, uma arquitetura OLAP com bancos de dados NoSQL, capaz de suportar o processamento e geração de análises de um grande volume de dados. Esse SAD possibilitará consultas mais direcionadas, sumarizadas, e poderá promover um ganho de tempo considerável em análises de dados que antes seriam feitas ou de forma manual, ou utilizando técnicas menos otimizadas que a modelagem dimensional de dados distribuídos. Além disso, por meio dessa nova organização de armazenamento, haverá a possibilidade de criação de um ambiente propício à análise dos dados por meio de técnicas de mineração de dados, por exemplo.

1.6 Objetivos Específicos

De modo a alcançar o objetivo geral proposto, os seguintes objetivos específicos foram estabelecidos:

- Analisar diferentes modelos de bancos de dados NoSQL e compará-los em termos de desempenho, armazenamento e aproveitamento do que já foi desenvolvido para o BI-UnB até o momento;
- Analisar e comparar diferentes estratégias de arquiteturas de distribuição de dados;
- Propor e validar uma arquitetura de *Data Warehouse* não convencional a partir das pesquisas contidas no estado da arte da literatura [2] [6] [7] em bancos de dados NoSQL.
- Propor uma extensão do processo de desenvolvimento de um Sistema de Apoio à Decisão desenvovido por Kimball [1] para alinhar o processo à arquitetura proposta.

1.7 Estrutura do Trabalho

Este trabalho está estruturado nos seguintes capítulos:

- O **Capítulo 2** apresenta a fundamentação teórica dos principais conceitos e tecnologias utilizados para desenvolvimento desta pesquisa. São abordados os principais conceitos a respeito de Sistemas de Apoio à Decisão, uma visão geral sobre os princípios de bancos de dados NoSQL e o que os diferencia de um banco de dados relacional tradicional, alguns conceitos importantes para o entendimento de *Big Data* e

breve apresentação do *framework* Apache Hadop, muito utilizado em soluções com processamento de dados distribuídos

- O **Capítulo 3** aborda trabalhos relacionados a esta pesquisa, quais os pontos positivos, as limitações e seus objetivos.
- O **Capítulo 4** apresenta a metodologia desta pesquisa, que consistiu na revisão sistemática da literatura para levantamento dos principais tópicos relacionados à fundamentação teórica necessária e para a identificação dos principais estudos relacionados que serviram de base para o desenvovimento deste trabalho. Além disso, também aborda com mais detalhes um estudo de caso para a migração de uma arquitetura ROLAP para uma arquitetura NoLAP do Sistema de Apoio à Decisão Acadêmico da UnB e a escolha do SGBD para o projeto final desta pesquisa.
- O **Capítulo 5** apresenta a arquitetura e o processo utilizados no projeto e a configuração em mais detalhes do *cluster* Hadoop e do SGBD HBase.
- O **Capítulo 6** apresenta um pouco mais em detalhes como foi feita a implementação do *cluster* Hadoop e do SGBD HBase, assim como das ferramentas OLAP Mondrian, Pentaho e Saiku.
- O **Capítulo 7** apresenta as conclusões do trabalho realizado e as discussões relacionadas à confirmação da hipótese de pesquisa apresentada no Capítulo 1 e relacionadas ao cumprimento dos objetivos gerais e específicos.

Capítulo 2

Fundamentação Teórica

Neste capítulo é apresentada a Fundamentação Teórica dos principais conceitos e tecnologias consideradas importantes para desenvolvimento desta pesquisa. Na Seção 2.1, são apresentados os principais conceitos a respeito de Sistemas de Apoio à Decisão. A seção 2.2 aborda uma visão geral sobre os princípios de bancos de dados NoSQL e o que os diferencia de um banco de dados relacional tradicional. Na seção 2.3, são apresentados alguns conceitos importantes para o entendimento de *Big Data*, ao passo que seção 2.4 traz uma breve apresentação do *framework* Apache Hadoop, muito utilizado em soluções com processamento de dados distribuídos. A fundamentação teórica foi escolhida a partir das palavras-chaves mais citadas pelos artigos mais referenciados encontrados a partir da metodologia da Teoria do Enfoque Meta Analítico Consolidado-TEMAC, de Mariano e Rocha [8], detalhada no Capítulo 4.

2.1 Sistemas de Apoio à Decisão - SAD

A arquitetura utilizada pelo SAD da Universidade de Brasília é a arquitetura OLAP - *On Line Analytical Processing* [1]. Dentre seus conceitos, está o *Data Warehouse*, ou seja, um repositório de dados multidimensional capaz de armazenar dados originários de fontes diversas, como, por exemplo, bancos de dados, planilhas eletrônicas ou arquivos em formato de texto. Esses dados, que podem ser da organização responsável pelo SAD ou externos a ela, passam por um processo de extração, transformação e carga (*Extract, Transformation, Load* - ETL), para permitir uma análise integrada das informações. A arquitetura OLAP é composta por:

- Fontes de dados;
- Camada de ETL *(Extract, Transformation, and Load);*
- *Data Warehouse;*

- Servidor de relatórios analíticos

O servidor de relatórios analíticos possibilita consultas sob demanda, que permitem uma navegação pelos dados organizados em dimensões no *Data Warehouse* e a geração de *dashboards* ou relatórios sem a necessária geração de código. A figura 2.1 demonstra uma arquitetura típica de um Data Warehouse.

A arquitetura OLAP pode ser implementada de formas diferentes, e dependendo do repositório de dados, pode receber algumas classificações. Na arquitetura MOLAP, os dados ficam armazenados em um banco de dados multidimensional. Já a arquitetura ROLAP é uma simulação da tecnologia OLAP feita em banco de dados relacionais. Essa ferramenta tem a vantagem de utilizar tecnologia estabelecida, com arquitetura padronizada, beneficiando-se da diversidade de plataformas, escalabilidade e paralelismo de *hardware*. Sua desvantagem é o conjunto pobre de funções para análises dimensionais e o baixo desempenho da linguagem SQL na execução de consultas mais complexas [1].

Embora seja pouco utilizado na literatura, o termo *NoSQL Online Analytical Processing* - NOLAP tem sido utilizado mais recentemente em algumas pesquisas e trabalhos científicos. Além disso, é um termo intuitivo àqueles familiarizados às terminologias OLAP. Fourny [9] utiliza o termo NoLAP em seu artigo sobre bancos de dados NoSQL. Victorino et. al[10], embora não tenha usado o termo explicitamente, utilizou o conceito de banco de Dados NoSQL com arquitetura OLAP para uma proposta de ecossistema de *BIG DATA* para a análise de dados abertos governamentais. Em suma, trata-se de um novo conceito de OLAP que estende o paradigma do primeiro para se beneficiar da flexibilidade dos bancos NoSQL, já que esses não possuem algumas limitações dos bancos relacionais, como por exemplo, consultas a coleções de dados em uma única transação e facilidade nativa de operarem em arquitetura de *cluster*.

De acordo com Inmon[11], *Data Warehouse* é uma coleção de dados, orientada a assuntos, integrada, variável no tempo e não volátil, para suporte ao gerenciamento dos processos de tomada de decisão :

- **Integrado:** Os dados são reunidos no armazém de dados - *Data Warehouse* - a partir de várias origens distintas e consolidados em um ambiente íntegro e consistente. Essas origens podem ser sistemas internos à organização, sistemas externos, planilhas eletrônicas, etc.
- **Variável no tempo:** Todos os dados no DW são identificados com um período de tempo particular, que pode ser dia, mês, ano, período letivo, etc.
- **Não volátil:** Os dados são estáveis no DW, e sempre adicionados, mas nunca removidos. Uma mesma versão temporal de um dado seguirá a mesma indefinitivamente.

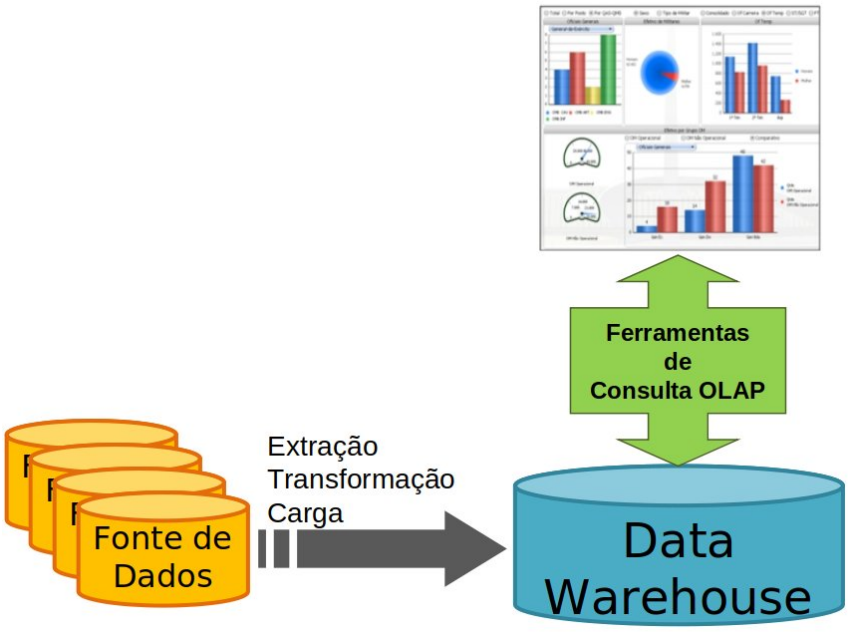


Figura 2.1: Arquitetura de um DW
Fonte: Adaptado de Kimball [1].

Para o desenvolvimento de um *Data Warehouse*, Kimball et al. [1] sugere o ciclo apresentado na Figura 2.2. Este ciclo é composto pelas seguintes atividades:

- **Planejamento do Projeto:** esta atividade consiste na elaboração do plano do projeto. Esse documento contém o planejamento detalhado da execução de todas as tarefas do projeto.
- **Definição dos Requisitos:** a definição dos requisitos pode ser dividida em duas tarefas: Levantamento das Necessidades do Negócio e Levantamento das Fontes de Dados.
- **Levantamento das Necessidades do Negócio:** consiste no levantamento das necessidades que o sistema deverá atender. Nesta etapa identifica-se o público-alvo e os requisitos funcionais e não funcionais para o DW.
- **Levantamento das Fontes de Dados:** consiste no levantamento e análise inicial das fontes de dados, incluindo o levantamento e análise da documentação existente sobre as fontes identificadas, tais como modelos de dados e correspondentes metadados, documentação sobre detalhes a respeito de regras de negócio, tabelas, arquivos e demais artefatos. Tal atividade irá promover o entendimento sobre o negócio e seus dados, bem como irá auxiliar no planejamento do projeto e na identificação de riscos correspondentes às fontes de dados.

- **Projetar Arquitetura Técnica:** a atividade projetar a arquitetura técnica consiste no levantamento dos volumes envolvidos, tanto no que tange às bases de dados, quanto aos volumes de processamento e quantidade de usuários simultâneos. Também são organizados os softwares que comporão as camadas da arquitetura OLAP.
- **Modelagem Dimensional:** consiste na elaboração do modelo dimensional de dados do DW de acordo com o escopo da iteração do projeto. Nessa atividade são identificadas as dimensões e fatos relativos aos temas da iteração. Para cada fato são definidas tanto as dimensões envolvidas, quanto o seu conteúdo e o *grão* dos dados, ou seja, no menor nível de detalhe.
- **Projetar Aplicação de Business Intelligence (BI):** esta atividade consiste na definição das consultas previstas para a iteração e a documentação dos indicadores identificados. Inclui o levantamento das consultas e análises demandadas pelos usuários e que serão a base para a homologação correspondente ao ciclo.
- **Projeto Físico:** consiste na construção do modelo físico relacional e multidimensional. O modelo físico deriva do modelo lógico e de tratamentos de performance e controle; o modelo multidimensional depende das visões e consultas a serem liberadas para os usuários (consultas solicitadas, perfil de usuário, etc).
- **Selecionar e Instalar Produtos:** consiste em preparar o ambiente de desenvolvimento, disponibilizando o software e o hardware configurados.
- **Implementar Rotinas ETL:** consiste na especificação e documentação dos processos de extração, transformação e carga (ETL) dos dados. Inclui a obtenção dos dados que alimentarão o DW a partir da extração dos dados das bases transacionais.
- **Implementar Aplicação de Business Intelligence (BI):** consiste na implementação das consultas e das fórmulas que gerarão os indicadores solicitados pelos usuários.
- **Implantação:** consiste na disponibilização dos temas desenvolvidos na iteração para o DW no ambiente de produção e avaliação de performance. Esta fase inclui também a passagem da aplicação do ambiente de desenvolvimento para o ambiente de produção, a documentação do processo e o treinamento dos técnicos que darão sustentação ao aplicativo.
- **Nova Iteração:** consiste em planejar a próxima iteração levando em conta aspectos ressaltados pelos desenvolvedores ou usuários dos produtos já disponibilizados pelas iterações finalizadas.

- **Manutenção:** consiste em manutenir algum componente da aplicação instalada que não esteja em conformidade com as especificações do projeto ou evoluir algum requisito por solicitação dos usuários.
- **Gerenciamento do Projeto:** concentra as tarefas relacionadas ao acompanhamento e controle da execução do projeto.

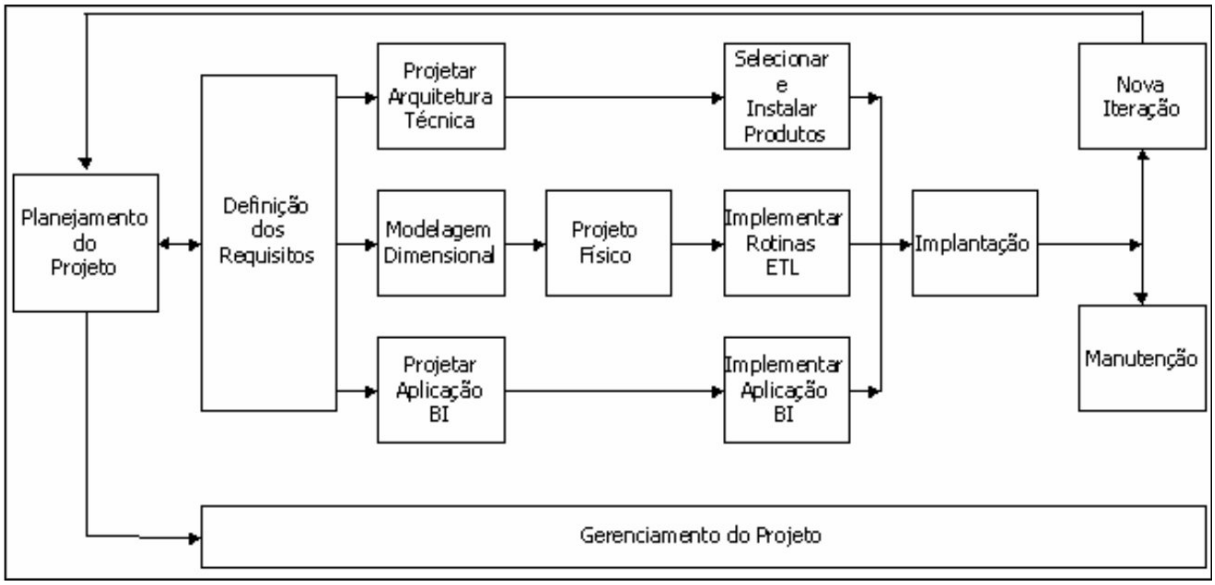


Figura 2.2: Ciclo de vida de um DW
Fonte: Adaptado de Kimball [1].

2.2 Bancos de Dados NoSQL

Um banco de dados NoSQL pode ser definido como aquele que não se encaixa em todos os paradigmas da teoria dos bancos de dados relacionais [12]. Como por exemplo, bancos de dados NoSQL podem incluir modelos de dados sem esquema, sem normalizações, coleções de dados em uma única coluna de uma tabela, novos modelos de distribuição ou até mesmo linguagens de consultas diferentes e mais enxutas que a consagrada SQL - *Structured Query Language*. Por causa do grande aumento do número de usuários e do volume de dados gerados, principalmente a partir dos anos 2000, surgiu um dilema em relação à forma de se tratar o processamento desse aumento tanto do volume de dados quanto do número de usuários: adquirir máquinas com maiores recursos, e mais caras, ou utilizar máquinas com menores recursos em um *cluster*. Sadalage e Fowler [12] ainda descrevem que o cenário com *cluster* não foi previsto no desenvolvimento dos bancos relacionais e por isso estes não foram projetados para trabalhar dessa forma, ao contrário dos bancos de dados

NoSQL. Sendo assim, em virtude deste mesmo cenário descrito ocorrer na Universidade de Brasília nos últimos anos, escolheu-se seguir o caminho do processamento distribuído dos dados no projeto final deste trabalho.

De forma geral, bancos de dados NoSQL são *softwares* de código aberto, são orientados à execução em *clusters*, não possuem esquemas rígidos e trabalham com consistência eventual, ou seja, em algum momento, todos os nós do *cluster* de um sistema NoSQL podem ter inconsistências de replicação de um determinado dado, mas, quando cessarem-se as atualizações, todos esses nós ficarão atualizados com o mesmo valor em algum momento, em função dos requisitos do sistema. Essa característica surge da implementação do teorema CAP - *Consistency* (Consistência), *Availability* (Disponibilidade) e *Partitioning* (Particionamento):

- Entre as três propriedades, Consistência, Disponibilidade e Particionamento, somente é possível a obtenção de duas delas ao mesmo tempo em um sistema distribuído.

Quanto à classificação, os bancos de dados NoSQL podem ser categorizados em quatro classes, conforme suas características [5]. Cada uma dessas classes possui sistemas que atendem a diferentes limitações dos bancos relacionais tradicionais e complementam-se entre si:

- Chave-valor *(key-value)*;
- Orientados a Documentos ;
- Família de Colunas (*column family* ou *columnar*)
- Orientados a Grafos (*graph database* ou *triple*)

Os bancos de dados do tipo chave-valor são aqueles cujo todo o armazenamento e acesso ao banco de dados são feitos por meio de uma chave primária, por isso geralmente apresentam ótimos desempenho e escalabilidade. São utilizados, por exemplo, para gerenciar funcionalidades como listas de itens mais vendidos, carrinhos de compras, preferências do consumidor, gerenciamento de produtos em diversos sites de compras, entre outras aplicações. A maioria dos bancos de dados que possuem armazenamento em memória RAM são da categoria chave-valor especialmente pela simplicidade de sua estrutura e possibilitam a visualização da base de dados como uma tabela *hash*. Neste modelo, o armazenamento dos dados é a combinação de um conjunto de chaves, e estas estão ligadas a um valor, *string*, ou seja, campos em formato de texto, ou binário [5]. No entanto, não devem ser utilizados quando houver necessidade de consultas pelo valor do dado ou então por relacionamentos entre os dados. Entre alguns exempos, estão Riak e Redis.

Nos bancos orientados a documentos, as informações são armazenadas sobre uma estrutura de árvore, em formatos como *XML* ou *JSON*. São semelhantes aos bancos de chave-valor, porém, neste caso o valor é estruturado e possui hierarquia. Diferentemente dos bancos tradicionais, estes não possuem um esquema ou estrutura pré-definida. É bastante comum a utilização dos formatos JSON e XML para representar os documentos nestes tipos de bancos de dados e o suporte ao processamento e análise de dados é bastante variado dentre os representantes da categoria [12]. São utilizados para registros de eventos de *Logs*, em sistemas gerenciadores de conteúdo, em sistemas de comércio eletrônico, entre outros. Entre alguns exemplos, estão MongoDB e RavenDB.

Os bancos de dados orientados a Grafos possuem foco na criação de relacionamentos entre os dados, porém, diferentemente dos bancos relacionais tradicionais, os dados não são tabulares, mas armazenados em registros triplos compostos por sujeito, predicado e objeto, ou seja, armazenam entidades e relacionamento entre essas entidades. Entidades são nós que possuem propriedades e os relacionamentos são arestas que podem possuir várias propriedades. As arestas possuem significado direcional e uma consulta é conhecida como travessia do grafo. Com estes bancos, é possível construir redes complexas de relacionamentos e revelar novos dados através da inferência [12]. Entre alguns exemplos, estão Neo4J e Infinite Graph.

Os bancos de dados de famílias de colunas possuem a sua estrutura similar à estrutura tradicional de tabelas dos bancos de dados relacionais, mas são otimizados de modo que as informações armazenadas estão organizadas em colunas ao invés de linhas. Têm sido utilizados para diversas funcionalidades, como armazenar as notificações de usuários, gerir os itens das páginas ou indicação de favoritos, entre outras aplicações em sistemas *Web*. Embora o modelo de colunas compartilhe o conceito de armazenamento dos sistemas relacionais, a diferença é que ele foi criado para arquiteturas que estão distribuídas massivamente. No armazenamento em colunas, cada chave está associada a um ou mais atributos (colunas) e as informações são guardadas de tal modo que as informações possam ser agregadas rapidamente. Os bancos colunares trabalham naturalmente com o uso da desnormalização através de duplicidade dos dados. Com isso é possível obter um ganho na velocidade de leitura, pois não são utilizados relacionamentos que demandam um trabalho de reconstituição dos dados [12]. De acordo com a literatura [13] [14] [6], os bancos de dados de família de colunas são os mais indicados para a arquitetura de *Data Warehouse*. Esses bancos de dados permitem o armazenamento de dados com chaves mapeadas para valores, e os valores são agrupados em múltiplas famílias de colunas, cada uma dessas famílias de colunas funcionando como um mapa de dados. Famílias de colunas são grupos de dados relacionados que frequentemente são acessados juntos. Linhas diferentes de famílias de colunas não precisam possuir as mesmas colunas, e novas colunas

podem ser adicionadas a qualquer linha, a qualquer momento .Entre alguns exemplos, estão Cassandra e HBase. Como será explicado nos próximos capítulos, esses dois bancos de dados foram selecionados para o experimento preliminar, em que um dos dois foi escolhido para o projeto final deste trabalho.

2.2.1 Apache Cassandra

O Cassandra foi inicialmente desenvolvido pelo Facebook para ser utilizado no motor de busca de sua caixa de entrada de mensagens. Em 2008, ele se tornou *opensource* e em 2009 passou a ser mantido pela *Apache Foundation*. Seu modelo de distribuição do sistema é baseado no Dynamo (desenvolvido pela *Amazon*) enquanto a forma de organização dos dados é baseado no BigTable, desenvolvido pelo Google.

Esse banco de dados possui uma topologia em anel, em que não há a abstração de um nó principal e outros secundários. O número total de dados armazenados pelo *cluster* é representado pelo anel, que é dividido em intervalos iguais ao número de nós, e cada nó pode ser responsável por um ou mais intervalos de dados. Há também o conceito de *Keyspace*, semelhante a um esquema em um banco de dados relacional, em que no *Keyspace* há as colunas e as famílias de colunas, formadas pelo nome do campo, seu valor e seu *timestamp*, utilizado pelos nós para realizar a sincronização eventual dos dados. Cada linha de dados é identificada de forma única pela chave primária da linha. Esse banco de dados foi projetado para trabalhar com uma alta taxa de disponibilidade, sendo que dessa forma torna-se inviável a existência de transações e possibilita o envio de vários registros em uma mesma família de colunas ao mesmo tempo. O *timestamp* existente é utilizado para se identificar qual a versão mais recente do dado, para resolver conflitos de gravação ou para identificarem-se dados desatualizados [15].

O Cassandra também foi projetado para permitir a replicação da informação, tanto para permitir melhora de desempenho quanto para ser tolerante a falhas. No momento de criação de um *keyspace*, é definido a quantidade de nós nos quais a informação será replicada. A toda requisição, tanto de escrita quanto de leitura, também será feita no valor de réplica. Em relação à consistência, há formas variadas de garanti-la, do menor para o maior nível, sendo que no maior, o processo só será considerado finalizado quando todos os nós forem atualizados. Cassandra particiona de forma transparente os dados pelos nós participantes do *cluster*, e cada nó fica responsável por uma parte do *keyspace*.

Outra característica desse banco de dados é que não há relacionamentos, ou seja, a base deve ser desnormalizada, uma mesma informação poderá aparecer em diversas colunas. Além disso, diferentemente da linguagem SQL, Cassandra não permite alguns operadores, como *case* e *when*, por exemplo, ou funções de agregação, como a função

max. Para esse tipo de informação, é necessária a criação de diversas *Views* contendo a informação pré-processada em uma família de colunas [12].

2.2.2 Apache HBase

O Apache HBase é um banco de dados de código aberto NoSQL de família de colunas que fornece acesso de leitura e gravação em tempo real a grandes conjuntos de dados. Esse banco de dados foi projetado de forma a fornecer escalabilidade linear de modo a permitir o processamento de grandes conjuntos de dados com bilhões de linhas, além de combinar fontes de dados com variedades de estruturas e esquemas diferentes.

HBase possui alta escalabilidade, ou seja, um novo nó pode ser adicionado ao *cluster* a qualquer momento para melhorar o processamento. Outra característica é que todas as tabelas devem possuir uma chave primária, mesmo no caso da existência de uma única família de colunas. O espaço da chave está dividido em blocos sequenciais que são então atribuídos a uma *region*, ou região. Os *RegionServers* possuem uma ou mais regiões, de modo que a carga está distribuída uniformemente em todo o *cluster*. Se as chaves dentro de uma região forem freqüentemente acessadas, o HBase pode subdividir a região dividindo-a automaticamente, para permitir um processamento paralelo dos dados [16]. Ou seja, a unidade mais simples e fundamental da escalabilidade horizontal no HBase é a região, um conjunto contíguo e ordenado de linhas que são armazenadas. A arquitetura do HBase possui um único nó mestre e vários nós secundários, ou seja, *RegionServers*.

Cada *RegionServer* contempla um conjunto de regiões, e cada região pertence somente a um único *RegionServer*. No momento de uma solicitação, o nó mestre, chamado *HMaster,* recebe e encaminha o processamento do dado ao *RegionServer* correspondente.

HBase também utiliza o componente do *framework Hadoop* chamado Zookeeper como um serviço de coordenação distribuída. Alguns desses serviços fornecidos pelo Zookeeper são os de estabelecer a comunicação do cliente com os servidores da região, acompanhar falhas do servidor ou de partições de rede, além de manter informações de configurações.

HBase e Cassandra possuem diferenças e similaridades. Entre as similaridades, os dois são bancos de dados de famílias de colunas, possuem escalabilidade horizontal, permitem replicação dos dados para melhora do desempenho e suporte às falhas, além de serem escritos na linguagem Java. Dentre as diferenças, estão suas arquiteturas, já que Cassandra possui uma topologia em anel, sem diferenças de hierarquia entre nós, ao passo que HBase segue a arquitetura do Apache Hadoop, em que há um nó mestre e outros nós seguidores. Os dois também diferem entre suas linguagens de consultas. HBase possui linguagem própria, mas pode ser utilizado em conjunto com o Apache Hive ou Apache Phoenix para criação de uma camada de comandos SQL de manipulação de suas tabelas, enquanto o Cassandra possui a CQL, com algumas limitações em relação aos comandos

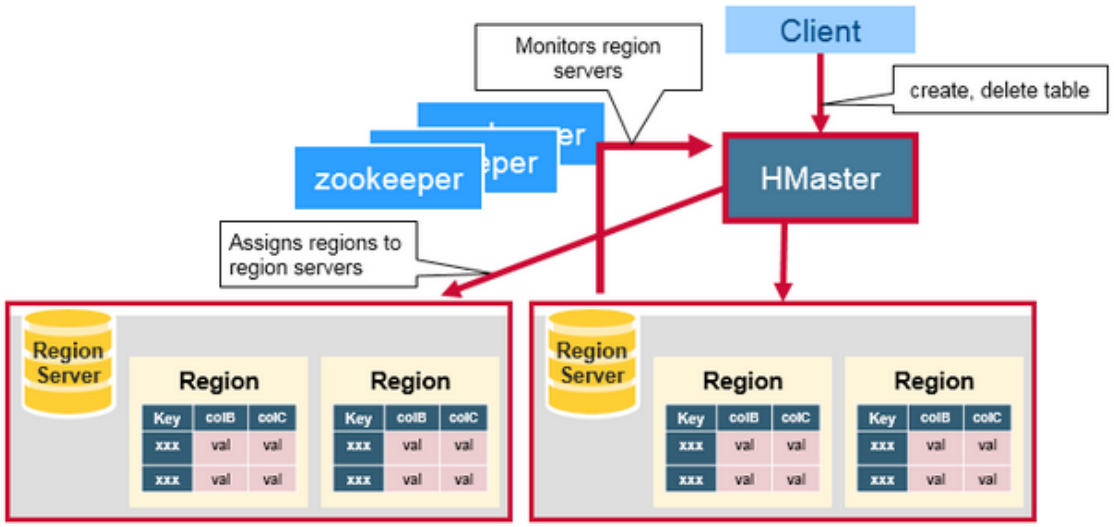


Figura 2.3: Arquitetura HBase
Fonte: Adaptado de Loukides [16].

dos Apache Hive ou Apache Phoenix. Em consequência, para consultas com dados agregados, Cassandra oferece um melhor desempenho se forem criadas *Views* materializadas. Ou seja, o modelo de banco de dados em Cassandra deve ser criado baseado nas consultas que serão executadas [17]. De forma geral, assim como a grande maioria dos bancos de dados NoSQL, tanto Cassandra quanto HBase podem fornecer excelentes soluções caso o problema a ser resolvido possua características que favoreçam às especificações de um dos dois bancos de dados. Dessa forma, é muito importante entender o problema antes da escolha do banco de dados que será utilizado em qualquer projeto.

A integração entre Hive e HBase pode ser feita por meio do uso de *Storage Handlers*, que são códigos de configurações de formatos de entrada e saída de dados do Hive, que identificam uma tabela do HBase como uma tabela externa ao Hive. Dessa forma, consultas em linguagem SQL podem ser realizadas para acessarem dados nas tabelas do banco NoSQL HBase sem restrições, incluindo junções entre diferentes tabelas. De forma semelhante, também ocorre a integração entre HBase e Apache Phoenix, que fornece um *driver JDBC* que permite a criação e exclusão de tabelas, além de comandos de manipulação de dados por meio de comandos SQL em tabelas criadas nos esquemas do Phoenix dentro banco HBase, o que permite uma utilização muito próxima a um banco de dados relacional, com todos os benefícios de um banco de dados distribuído.

A Figura 2.5 traz um esquema do HBase que demonstra de forma simples sua arquitetura. Como faz parte do *framework* Hadoop, esse banco de dados utiliza o sistema

de arquivos distribuído HDFS para armazenar seus metadados e tabelas, e utiliza os comandos *MapReduce* para manipulação de dados, como nos caso de *bulking inserts*, por exemplo.

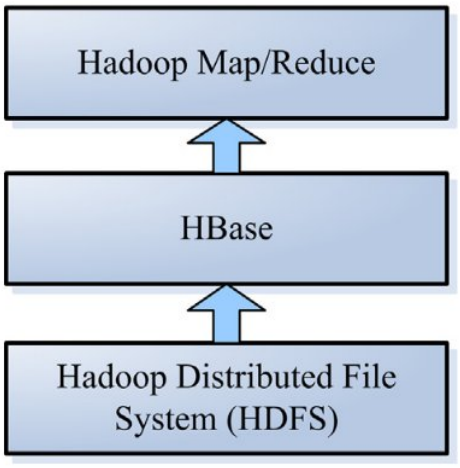


Figura 2.4: Esquema HBase
Fonte: Adaptado de Chen [18].

2.3 Big Data

Há várias variáveis associadas a *Big Data* e, de forma geral, os autores associam essas variáveis a características que se iniciam com a letra *V*, como nos trabalho de Laney [19] e Zikopoulos [20]. Algumas definições abrangem os termos volume, velocidade e variedade [19]. Outros autores definem valor e veracidade como características a mais [20]. A definição de *Big Data* envolve um conjunto de dados muito grande com uma grande diversidade de tipos, de difícil processamento por parte das arquiteturas tradicionais [18].

De acordo com o site ***www.enterprise2020.com***, o volume dados armazenado no mundo cresceu aproximadamente de 1,2 ZB em 2012 para 40 ZB em 2020. Em 2017, estimava-se que em um minuto apenas, ocorriam aproximadamente 3,8 milhões de buscas no *Google*, 3,3 milhões de postagens no *Facebook*, quase 450 milhões de *tweets* no *Tweeter* e 18 milhões de mensagens enviadas pelo *Whatsapp*, por exemplo. Dessa forma, surgiram as questões: Como armazenar esse grande volume de dados ? Como consultar de maneira eficiente ? Como obter informação? Como diminuir o custo das soluções?

A partir desse cenário, surgiram os bancos de dados NoSQL como parte da solução do problema no quesito armazenamento, pois possuem modelos flexíveis de dados, não tem o foco de normalização, melhoram seus desempenhos com a utilização de *clusters* e também com o suporte à replicação dos dados. Além dos bancos NoSQL, também surgiram outras

tecnologias, como o Apache Hadoop e *Spark* que, em conjunto com soluções para análises de grande volumes de dados, como, por exemplo, as ferramentas *Pentaho*, *Tableau* e *Power BI* permitiram grande avanço na análise de dados em muitas áreas causado pelas possibilidades a partir do grande volume de informações geradas, como na pesquisa científica, administração pública, área médica, física quântica, bioinformática, agronegócio, entre outras. Além disso, houve o aumento da eficiência operacional, melhor atendimento ao cliente e identificação de novos produtos e serviços a partir dessas novas análises.

Entre alguns conceitos de *Big Data*, estão:

1. Boas arquiteturas e *frameworks* devem ser prioridades.
2. Suporte a vários métodos analíticos
3. Nenhum tamanho será suficiente para todas as soluções
4. Levar a análise aos dados, e não o contrário
5. Processamento deve ser feito de forma distribuída em memória
6. O armazenamento de dados deve ser distribuível para memória.
7. Coordenação necessária entre nós de processamento

Um conceito muito importante relacionado a *Big Data* é o *MapReduce*, com o objetivo de dividir os problemas complexos do *Big Data* em pequenas unidades de trabalho e processa-las em paralelo. O *Map-Reduce* pode ser dividido em dois estágios:

1. **Passo de mapeamento:** o nó mestre divide os dados em vários subconjuntos menores, um nó trabalhador (*worker*) processa um subconjunto de dados menor sob o controle de um rastreador de trabalho (*Jobtracker*) e armazena o resultado no sistema de arquivos local, onde um redutor (*reducer*) será capaz de acessá-lo.
2. **Passo de redução:** esta etapa analisa e reúne os dados de entrada a partir das etapas de mapeamento. Podem haver múltiplas tarefas de redução para paralelizar a agregação, e essas tarefas são executadas nos nós trabalhadores (*workers*) sob o controle do rastreador de trabalho (*JobTracker*).

A figuras 2.7 e 2.8 trazem um exemplo, em que há uma nota fiscal de um cliente com os itens comprados, e os valores do preço gasto e as quantidades compradas. Deseja-se saber os totais de quantidade e preço. A função map lê registros do banco de dados e emite pares de chave-valor, e a função reduce recebe diversos pares de chave-valor com a mesma chave e os agrega em um, no caso, agregou os valores das variáveis preço e quantidade. Assim, na etapa de redução, o redutor reconhece as variáveis em comum e realiza a soma destas para gerar o valor final da consulta recebida.

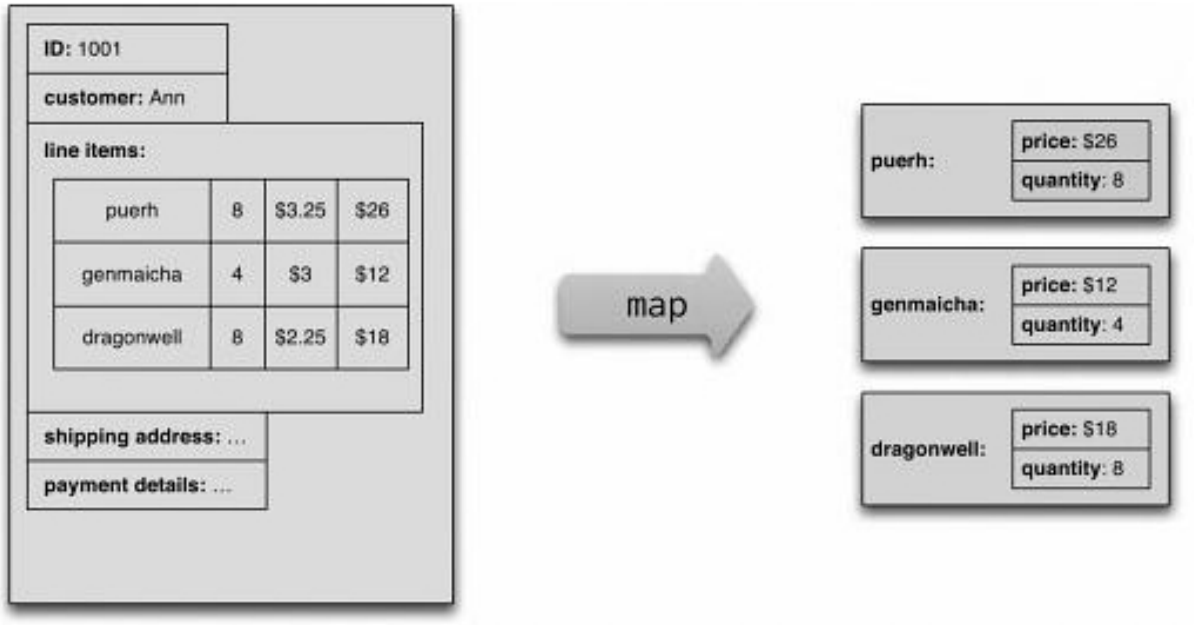


Figura 2.5: Etapa de Mapeamento
Fonte: Adaptado de Sadalage[12]

Figura 2.6: Etapa de Redução
Fonte: Adaptado de Sadalage[12]

2.4 Apache Hadoop

O Apache Hadoop foi criado por Doug Cutting e inspirado no BigTable, que é o sistema de armazenamento de dados do Google, baseado no *Google File System* e no conceito *MapReduce* [21]. O Apache Hadoop é um *framework* baseado em Java, *open source* e trata-se de uma plataforma heterogênea com diversas tecnologias e soluções.

Dentre suas características, é um framework que permite o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores que utilizam modelos de programação simples, projetado para fornecer escalabilidade a partir de um único servidor

para milhares de máquinas, cada uma oferecendo processamento e armazenamento local, e concebido para detectar e tratar falhas, de modo a fornecer um serviço o mais disponível possível.

Principais módulos do hadoop, conforme Figura 2.9:

1. **Hadoop Common:** Contém as bibliotecas e arquivos comuns e necessários para todos os módulos Hadoop.
2. **Hadoop Distributed File System (HDFS):** Sistema de arquivos distribuído que armazena dados em máquinas dentro do cluster, sob demanda, permitindo uma largura de banda muito grande em todo o cluster.
3. ***Hadoop Yarn:*** Trata-se de uma plataforma de gerenciamento de recursos responsável pelo gerenciamento dos recursos computacionais em cluster, assim como pelo agendamento dos recursos.
4. ***Hadoop MapReduce:*** Modelo de programação para processamento em larga escala.
5. **ZooKeeper:** serviço centralizado para coordenação de aplicações distribuídas. Mantém informações de configuração das aplicações distribuídas, além de fornecer a sincronização entre elas. Como já citado para o HBase, o Zookeeper é o responsável pela coordenação e sincronização das regiões nos *Region Servers*.
6. **Banco de dados HBase**
7. **Apache Hive**

A arquitetura do Apache Hadoop pode ser definida da seguinte forma:

1. **Camada de armazenamento:** Hadoop Distributed File System (HDFS)
2. **Camada de processamento:** Yarn e Map Reduce
3. **Camada de acesso aos dados:** ferramentas como Apache Hive, para abstração da linguagem SQL como interface aos comandos *MapReduce*; e o banco de dados HBase, que pode ser utilizado em conjunto com o Apache Hive ou com o Apache Phoenix.

O *framework Hadoop* utiliza *MapReduce* para dividir os problemas complexos de *Big Data* em pequenas unidades de trabalho , *jobs* , e as processa em paralelo.

Um componente muito importante do Hadoop é o **YARN- *Yet Another Resource Negotiator***, formado pelos *Resource Manager*, gerenciador de recursos, e pelo *Node Manager*, gerenciador de nó, que gerenciam os recursos e processamento de dados no

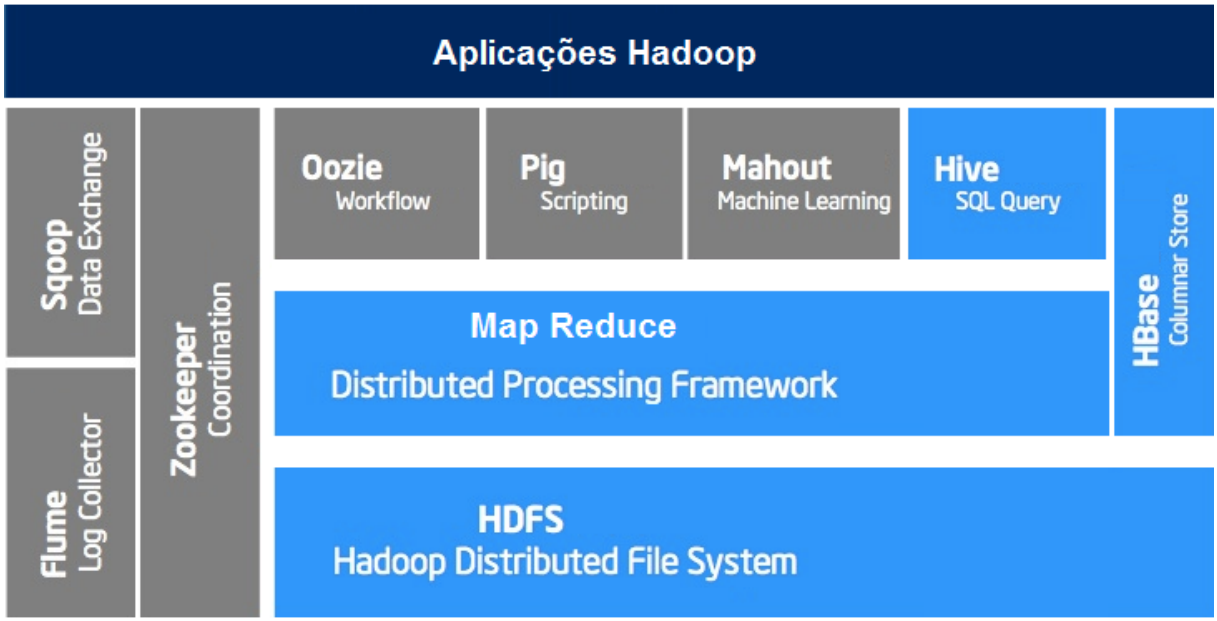


Figura 2.7: *Framework Hadoop*
Fonte: Adaptado de White [21].

sistem de arquivos HDFS. O *Resource Manager* é responsável por receber tarefas de *MapReduce* e submetê-las aos *Node Managers*, e também deve comunicar-se com o *Name Node*, ou nó master, para a localização dos dados a serem processados nos *Data Nodes*, que estão distribuídos pelos nós do *cluster*. O *Resource Manager* deve ser acionado pelos *Node Managers* a intervalos regulares, de modo a estes reportarem o estado de seus processamentos.

2.4.1 Apache Zookeeper

O Apache Zookeeper é um serviço centralizado para manter informações de configurações e nomenclaturas entre serviços distribuídos do *framework* Hadoop. Ele sincroniza as configurações entre diferentes nós do *cluster*, e é um componente obrigatório do banco de dados HBase, já que atua na sincronização dos dados desse SGBD. Outra característica desse componente é que seus dados ficam em memória e assim os nós do *cluster* Zookeeper conseguem obter informações uns dos outros o mais rápido possível.

Entre os exemplos de serviços do Apache Zookeeper, estão:

1. Manter arquivos de configuração em todos os nós para que a sincronização dos dados seja realizada
2. Serviços de nomes para encontrar uma máquina em um cluster

3. Prover sincronização distribuída: gerenciamento de filas, bloqueio e liberação de recursos
4. Serviços de grupos para *eleição* de um nó líder. Existe o conceito do *Zookeeper Quorum*, que trata-se de um quorum mínimo de nós para escolherem o novo *nó mestre* em caso de falha do atual.
5. Gerenciar quais nós estão ativos no cluster

Em um *cluster* Zookeeper, o nó mestre é escolhido dinamicamente por consenso dentro do conjunto, e se este falhar, sua função será migrada para outro nó. A gravação de dados é feita de forma linear, no nó mestre primeiro, que por sua vez replica para os demais nós, de forma assíncrona. A leitura pode ser feita em todos os nós, mas nem sempre o nó disponível tem a versão mais atual do dado.

2.4.2 Apache Hive

O Apache Hive é um componente do *framework* Hadoop, conforme figura 2.9, que abstrai o conceito de *Data Warehouse*, permitindo o resumo de dados, consultas e análises em HiveQL, uma linguagem de consulta semelhante à linguagem SQL. O Hive pode ter tabelas internas, em que os dados são armazenados em seu sistema interno de armazenamento, ou externas, quando os dados são originados de um arquivo externo, que pode ser um arquivo texto, ou inclusive podem ser tabelas do banco de dados HBase. Ou seja, na integração entre HBase e Hive, as tabelas do primeiro são mapeadas como tabelas externas do segundo, e dessa forma, permite-se consultar os dados das tabelas do HBase por meio de comandos SQL.

Hive faz parte da pilha de tecnologias do Apache Hadoop e foi desenvolvido em função da complexidade de comandos para executar consultas nos arquivos armazenados dentro do sistema HDFS por meio de operações *MapReduce*. Dessa forma, Hive transforma as sentenças SQL em *Jobs MapReduce* executados no *cluster* Hadoop para consultas analíticas com fontes de dados arquivos de texto ou tabelas do HBase.

2.4.3 Apache Phoenix

O Apache Phoenix tornou-se um projeto da empresa Apache em 2014. Trata-se de uma solução de código aberto, que fornece um *driver JDBC*, ou seja, um componente que permite que uma aplicação Java consiga acessar um banco de dados, para tornar possível a manipulação de dados do SGBD HBase por meio de comandos da linguagem SQL, incluindo a utilização de transações. O Apache Phoenix traduz os comandos SQL em comandos da linguagem *HBase Query Language*, que por sua vez são traduzidos em

comandos *MapReduce* nas regiões dos *Region Servers* do HBase. Os metadados das tabelas criadas pelo Apache Phoenix são armazenados em uma tabela do HBase. Assim que uma tabela for criada no esquema do Apache Phoenix, uma tabela vinculada com mesmo nome é criada no HBase, onde os dados serão armazenados de fato. São permitidas as criações de Views e Índices para a melhoria do desempenho das consultas.

Apache Phoenix possui dois tipos de índices:

- **Índices locais:** indicados para sistemas com alta carga de escrita e baixa carga de leitura, ideais para sistemas OLTP. Nesse tipo de índice, as tabelas de índices e as tabelas dos dados originais são armazenados no mesmo *Region Server*.
- **Índices Globais:** indicados para sistemas com alta carga de leitura e baixa carga de escrita, ideais para *Data Warehouses*. São criadas tabelas de índices que são lidas como quaisquer outras tabelas do HBase, distribuídas em várias regiões, em diferentes *Region Servers*, e que respondem caso estejam mais próximas ao local de onde ocorreu a requisição. Nos índices globais, as tabelas de índices podem ser armazenadas em *Region Servers* diferentes das tabelas que possuem o dado de origem, justamente para melhorar o desempenho de leitura.

Apache Phoenix também possui a possibilidade de se criarem **índices cobertos**, ou *covered*, que permitem a associação entre duas ou mais colunas em um mesmo índice, caso essas duas colunas sejam muito utilizadas ao mesmo tempo como filtros em consultas ao sistema.

Capítulo 3

Trabalhos Relacionados

Neste capítulo, serão apresentados os principais trabalhos relacionados a esta pesquisa. A seleção iniciou-se por meio de buscas a partir de duas importantes fontes científicas acadêmicas: *Web of Science* e *Scopus*. A metodologia de pesquisa será detalhada no Capítulo 4, em que foi utilizada a metodologia da Teoria do Enfoque Meta Analítico Consolidado-TEMAC, de Mariano e Rocha [8]. Os resultados nas fontes científicas foram os seguintes, em julho de 2019:

1. **Web of Science**
    - termo NoSQL a partir de 2012: 1.556 resultados
    - *NoSQL and (OLAP or 'DATA WAREHOUSE')* a partir de 2012: 57 resultados
2. **Scopus**
    - termo NoSQL a partir de 2012: 2.583 resultados
    - *NoSQL and (OLAP or 'DATA WAREHOUSE')* a partir de 2012: 141 resultados

Os autores escolhidos pela relevância de suas obras foram Max Chevalier, Khaled Dehdouh, El Malki e Kopliku, pois seus trabalhos foram os mais próximos ao desta pesquisa pelo fato de darem ênfase a um modelo de migração de um *Data Warehouse* em bancos de dados relacionais para uma abordagem em bancos de dados NoSQL, além de realizarem comparações entre arquiteturas envolvendo mais de um banco de dados NoSQL. Além destes, outros autores foram selecionados por referenciarem aqueles principais autores e por complementarem os estudos relacionados a *Data Warehouse* com NoSQL a partir de estudos de caso, e por isso foram citados para o enriquecimento do conteúdo acadêmico desta pesquisa. A seleção dos artigos foi feita a partir da leitura dos resumos e das introduções, quando os resumos não ficavam muito claros, e buscaram-se estudos de casos

envolvendo transformações de sistemas de *Data Warehouse* em bancos de dados relacionais para bancos de dados NoSQL ou então para pesquisas com ênfase em propostas de arquiteturas envolvendo soluções para o tratamento de volumes massivos de dados. Além de artigos com foco em bancos de dados NoSQL, também foram escolhidos aqueles com foco no *framework* Hadoop e em suas teconologias, como Hive e Hbase.

3.1 Novas propostas de Arquitetura de Data Warehouse com NoSQL

Khaled et. al [2] realizou um estudo em que soluções para armazenamento de dados distribuídos, como os bancos de dados NoSQL e frameworks como *Hadoop* apresentam-se como um recurso para o armazenamento massivo de dados altamente escalonáveis, tornando-se uma alternativa aos bancos de dados relacionais em determinadas situações. O autor trouxe uma proposta de três tipos de modelos conceituais colunares para um sistema de apoio à decisão: *NLA* (abordagem lógica normalizada), *DLA* (abordagem lógica desnormalizada com dados simples em colunas); e *DLA-CF* (abordagem lógica desnormalizada transformando tabelas de dimensões em família de colunas). Nesse caso, foi utilizado o banco de dados Hbase, em que as três propostas foram comparadas por meio do desempenho em duas categorias de consultas. A abordagem *NLA* obteve o pior desempenho em todas as situações. As abordagens *DLA* e *DLA-CF* obtiveram resultados semelhantes, sendo a *DLA-CF* com um desempenho um pouco melhor. A Figura 2.8 demonstra a transformação entre os modelos conceitual, lógico e físico por Khaled [2]:

A abordagem *NLA (Normalized Logical Approach)* simula um esquema normalizado como em um banco relacional tradicional. As tabelas de dimensões e de fatos são separadas em tabelas distintas. Para simular os relacionamentos, que inexistem nos bancos NoSQL de famílias de colunas, as chaves identificadoras das dimensões são duplicadas na tabela de fatos, e a integridade referencial fica sob responsabilidade do desenvolvedor.

Já a abordagem *DLA (Denormalized Logical Approach)* propõe a transformação do modelo conceitual em um modelo baseado em uma grande estrutura chamada *BigFactTable*, ou grande tabela de fatos, em que dimensões e fatos ficam armazenados juntos. Essa abordagem traz uma grande desnormalização dos dados em colunas de dados simples. Há uma única chave identificadora de dimensões e fatos, sem junções entre tabelas.

Por fim, a *DLA-CF (Denormalized Logical Approach by using Column Family)* diferencia-se da *DLA* por utilizar dados agregados em família de colunas para as dimensões. Isso possibilita que os atributos de uma mesma dimensão compartilhem o mesmo espaço em disco e melhore o tempo de acesso aos dados.

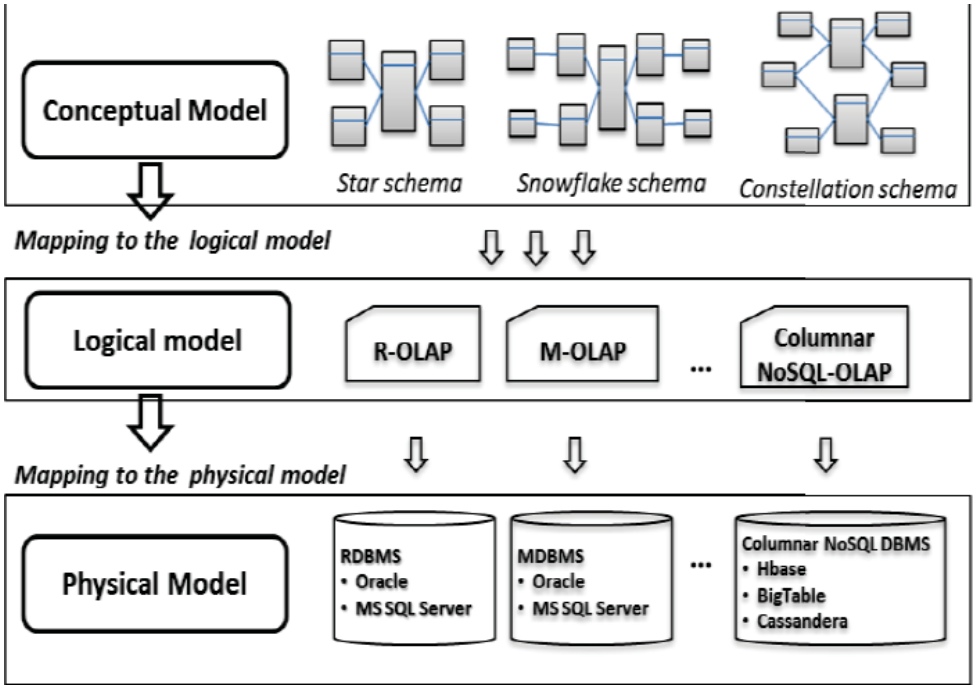


Figura 3.1: Modelos *ROLAP, MOLAP e NoLAP*
Fonte: Adaptado de Khaled [2].

Chevalier et al. [6] realizou um trabalho em 2015 com uma proposta muito semelhante à de Khaled, e também com três abordagens: *MLC0* (todas as tabelas de dimensões e a tabela de fatos transformadas em uma única tabela); *MLC1* (dimensões transformadas em famílias de colunas com dados agrupados); *MCL2* (tabelas separadas para as dimensões e para a tabela de fatos). O foco nesse estudo foram as regras de transformação do modelo conceitual para o modelo lógico a partir dessas três abordagens e também as regras de transformação dos modelos entre eles, ou seja, do *MCL0* para *MCL1*, para *MCL0* para *MCl2* e nos casos contrários. Como resultado final, o modelo *MCL2* precisou de menor espaço em disco, mas obteve pior desempenho nas consultas, ao passo que os outros *MCL0* e *MCL1* obtiveram desempenho semelhantes, com vantagem para *MCL1*, de família de colunas. Ou seja, há relatos na literatura que atestam o bom desempenho da abordagem de transformação de um sistema de apoio à decisão para o esquema em uma única tabela.

Como o foco de Chevalier nesse trabalho são as regras de transformações para os modelos, o estudo traz propostas de conversões de fatos e dimensões em família de colunas de dados agregados ou simples. Em relação aos experimentos, a abordagem *MCL2* sempre ocupou o menor espaço em disco, mas obteve o pior desempenho em todas as situações. Ao contrário do estudo do autor Khaled, a abordagem *MCL1*, de família de colunas, obteve um melhor desempenho de forma mais notável em relação à *MCL0*, a abordagem com colunas simples, em função de as famílias de colunas ocuparem o mesmo espaço em disco, o que economiza tempo de busca para determinadas consultas.

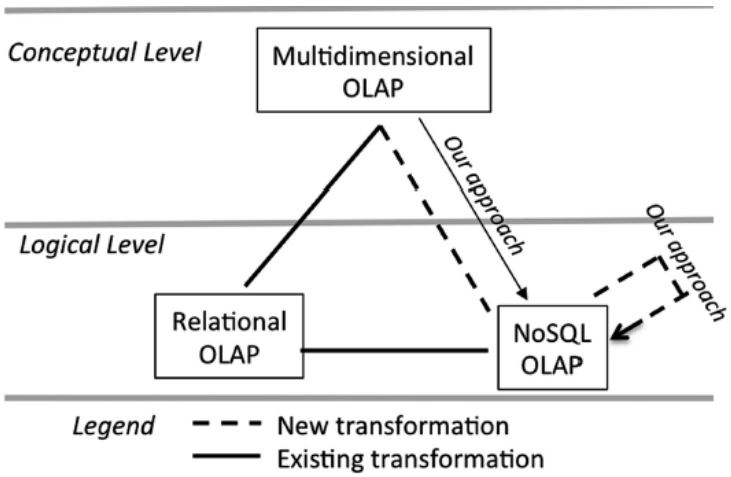


Figura 3.2: Transformação do modelo conceitual para lógico de Chevalier
Fonte: Chevalier [6]

As regras de transformação de Chevalier são as seguintes:

- **Modelo Conceitual para MCL0:**
  - Cada cubo de Fatos e Dimensões é transformado em uma tabela com uma única família de colunas
  - Cada métrica é transformada em um atributo na única família de colunas
  - Para todas as dimensões, cada atributo é convertido em um atributo da única família de colunas
- **Modelo Conceitual para MCL1:**
  - Cada cubo de Fatos e Dimensões é transformado em uma tabela com várias famílias de colunas
  - A tabela contém uma única família de colunas para os fatos.
  - A tabela contém uma família de colunas para cada dimensão.
  - Cada métrica é convertida em um atributo da família de colunas dos fatos
  - Para todas as dimensões, cada atributo é convertido em um atributo para cada família de colunas de cada dimensão
- **Modelo Conceitual para MCL2:**
  - A tabela de Fatos e as tabelas de Dimensões são transformados em tabelas distintas
  - Cada atributo da dimensão é convertido em um atributo (coluna) simples

– Cada métrica é convertida em uma coluna simples na tabela de fatos

– Cada identificador dos atributos nas dimensões é convertido em um atributo na tabela de fatos

3.2 Análise Consolidada dos Trabalhos Relacionados

Foram analisados 23 trabalhos relacionados à implementação de um Data Warehouse com bancos NoSQL a partir da pesquisa em importantes bases científicas: Scopus e Web of Science. A maioria dos trabalhos, mais precisamente 14, utilizaram o Apache Hive pela facilidade de uso da linguagem HiveQL, seja em uma simples análise de viabilidade de construção de um *Data Warehouse* com essa solução, seja na comparação com bancos de dados NoSQL, entre eles Cassandra e Hbase, e até mesmo em comparações com a solução tradicional de *Data Warehouse* com banco relacional, nesse caso, o banco de dados MySQL.

Tabela 3.1: Análise Consolidada entre Estudos de *Data Warehouse* com NoSQL

Estudo/Autor	Tecnologia Utilizada	Propósito
Chao [22]	Hive	DW com Processamento paralelo
Yongqiang [23]	Hive	DW com RCFile
Camacho [24]	Hive	Viabilidade de DW com Hive
Martinho [25]	Hive	DW com Hive
Li Xi [26]	Hive	DW com Hive
Garg [27]	Hive	Otimização de consultas
Kang [28]	Hive	Desempenho com MapReduce
Gadiraju [29]	Hive	Comparativo: Hive ou MySQL
Kumar [30]	Hive	Comparativo: Hive ou MySQL
Czajkowski [31]	Hive	DW para base de dados meteorológicos
Costa [32]	Cassandra ou Hive	Comparativo: Cassandra ou Hive
Khaled [14]	HBase juntamente com Hive	DW: de BD Relacional para NoSQL
Ibrahim Gad [33]	HBase juntamente com Hive	DW para dados climáticos
Song [34]	Hive ou HBase	Comparativo: Hive ou HBase
Yangui [7]	HBase ou MongoDB	Comparativo: HBase ou MongoDB
Khaled [2]	HBase	DW: de BD Relacional para NoSQL
Chevalier [6]	HBase	DW: de BD Relacional para NoSQL
Scabora [35]	HBase	DW: de BD Relacional para NoSQL
Mohamed Boussahoua [36]	HBase	DW: de BD Relacional para NoSQL
Franciscus [37]	HBase	DW: de BD Relacional para NoSQL
Sugam Sharma [17]	HBase ou Cassandra	Comparativos entre bancos NoSQL
Chevalier [38]	MongoDB ou HBase	Comparativo: HBase ou MongoDB
Alekseev [13]	Cassandra ou MongoDB	Comparativo: Cassandra ou MongoDB

O banco de dados HBase também foi bastante utilizado, em 10 trabalhos, seja para uma análise de migração de um *Data Warehouse* em bancos relacionais para uma abordagem com NoSQL, seja para comparações entre Hive ou MongoDB. Em alguns trabalhos, Hive e Hbase foram utilizados conjuntamente, para aproveitamento da arquitetura de família de colunas do Hbase e da linguagem HiveQL.

O banco de dados NoSQL Cassandra foi pouco utilizado nos trabalhos relacionados encontrados nas pequisas relacionadas à *Data Warehouse*, e quase sempre associado ao *streaming* de dados.

Pela Tabela 3.1, percebe-se a consolidação dos trabalhos relacionados pesquisados. Como já dito, nas pesquisas em que foram analisadas a possibilidade de *Data Warehouse* com carga de dados via *streaming*, e não em *batch*, como é a forma mais tradicional, Casssandra obteve um desempenho superior ao Hive e ao MongoDB. No entanto, nos testes de desempenho de carga de dados em *batch*, Hbase obteve resultados melhores.

Sugam Sharma [17] realizou em 2015 um estudo comparativo entre vários soluções NoSQL, e chegou à conclusão que Cassandra possui melhor desempenho em processamento via *streaming*, enquanto o Hbase possui melhor desempenho em processamento *batch*.

Em relação à transformação de um *Data Warehouse* em arquitetura *ROLAP* para uma arquitetura OLAP com bancos de dados NoSQL - NOLAP, os já citados Chevalier [6] e Khaled [14] juntam-se a Rania Yangui et. al[7] em propostas de migração. Os três autores utilizaram o banco de dados Hbase em seus experimentos e chegaram à conclusão que o melhor desempenho foi obtido com a arquitetura em uma única tabela com fatos e dimensões agregados em famílias de colunas. Além de Hbase, Khaled também utilizou Hive em seu experimento pela facilidade da linguagem HiveQL. Dessa forma, consegue-se aproveitar o conhecimento nessa última linguagem juntamente com os recursos do banco de dados colunar Hbase. Os anos de experiência com SQL podem representar um dos fatores que indicam a quantidade de pesquisas que utilizaram Hive observada na Tabela 3.1.

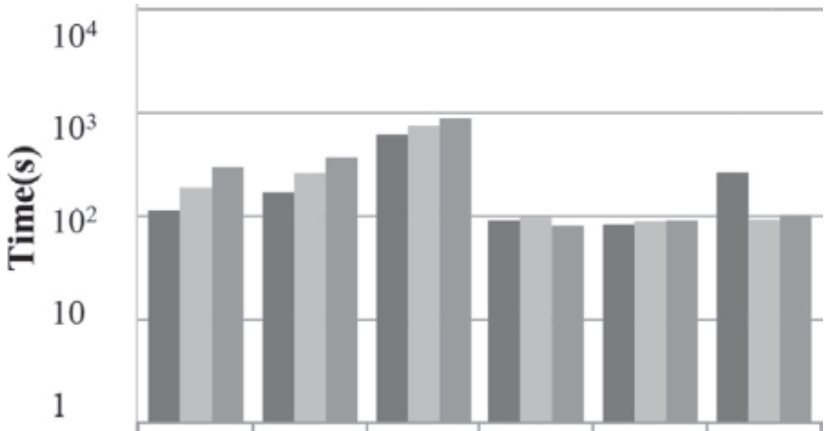
Em relação ao banco de dados orientado à documentos *MongoDB*, este foi utilizado em alguns trabalhos em comparativos a outros bancos de dados NoSQL, como Hbase [38] e Cassandra [13]. Yangui [7] concluiu que MongoDB obteve um melhor desempenho que HBase na arquitetura hieráquica para a consultas do experimento. No entanto, o próprio autor reconheceu que as consultas eram simples e pouco variadas, ou seja, mais consultas complexas deveriam ser realizadas para melhores análises. Chevalier[38] concluiu que Hbase obteve um desempenho melhor que MongoDB nos quesitos carga de dados e nas consultas com dados agregados. Em relação à comparação com Cassandra, Alekseev et. al [13] realizou experimento com alguns critérios comparativos entre Cassandra, MongoDB e Hive. A Figura 3.3 mostra um resumo do que foi feito.

	Evaluation Criteria Weight						
	50	10	10	20	5	5	100
NoSQL-system	query execution time for fetching	NoSQL-system monitoring	ease of writing queries	additional tools for processing data	ease of system configuration and deployment	completeness of documentation and manuals	rank
Hadoop + Hive	30	10	9	20	2	5	76
MongoDB	35	6	5	8	5	4	63
Cassandra	50	8	10	10	3	5	86

Figura 3.3: Cassandra x MongoDB x Hive
Fonte: Adaptado de Alekseev [13].

Segundo o autor Alekseev [13], Cassandra foi considerado a melhor solução para *Data Warehouse* segundo os critérios do experimento, com Hive logo em seguida.

Song et. al [34] realizou uma comparação entre Hive e Hbase em três consultas de *slice and dice*, ou seja, de cortes e mudanças de perspectivas de dimensões, e Hbase obteve um desempenho superior:



	Hive			HBaseLattice		
	S1	S2	S3	S1	S2	S3
C1	116	172	615	92	83	266
C2	187	261	751	96	90	94
C3	296	365	880	81	92	103

Figura 3.4: HBase x Hive - 3 consultas de *slice and dice*
Fonte: Adaptado de Song [34].

Capítulo 4

Metodologia

A metodologia desta pesquisa consiste na revisão sistemática da literatura para levantamento dos principais tópicos relacionados à fundamentação teórica necessária e para a identificação dos principais estudos relacionados que serviram de base para o desenvovimento deste trabalho, além de um estudo de caso realizado na Secretaria de Tecnologia da Informação da Universidade de Brasília - STI/UnB.

4.1 Teoria do Enfoque Meta Analítico Consolidado-TEMAC

A revisão sistemática foi realizada por meio da Teoria do Enfoque Meta Analítico Consolidado-TEMAC, de Mariano e Rocha [8]. O TEMAC possui três etapas:

1. **Etapa 1 - Preparação da pesquisa:** Resposta às questões sobre o termo de pesquisa, espaço temporal e base de dados. Foram utilizados os termos *'NOSQL' AND (OLAP OR 'DATA WAREHOUSE')*, entre os anos de 2012 a 2019. A base de dados utilizada foi a *Web of Science* em virtude da excelente organização de seus metadados. Ao todo foram retornados 57 artigos por meio da pesquisa em junho de 2019:

    - Web of Science
        - termo *NoSQL* a partir de 2012: 1.556 resultados
        - *NoSQL and (OLAP or 'DATA WAREHOUSE')* a partir de 2012: 57 resultados

2. **Etapa 2 - Apresentação e Interrelação dos dados:** Após a análise dos registros, observou-se que:

- **Palavras-chave** mais utilizadas pelos artigos
  - 'database' - 19 vezes
  - 'model' - 16 vezes
  - 'processing' - 9 vezes
  - 'analytics', 'olap' e 'system' - 8 vezes
  - 'mapreduce' - 7 vezes
  - 'hadoop' e 'query' - 6 vezes
  - mining' e 'multidimensional' - 5 vezes
  - 'hbase', 'hive' e 'colum-oriented' - 4 vezes
- **Autores** que mais publicaram:
  - R. Tournier, O. Teste, El Malki e Kopliku - 5 artigos
  - Max Chevalier - 4 artigos
  - Khaled Dehdouh e Boussaid - 3 artigos
- **Países** que mais publicaram:
  - França - 11 artigos
  - Estados Unidos - 9 artigos
  - China e Portugal - 7 artigos

3. **Etapa 3 - Detalhamento do modelo integrador e validação por evidências:** Nessa etapa, são realizadas análises bibliométricas, como *co-citation* e *coupling* , as principais abordagens e linhas de pesquisa. *Co-citation* é o termo referente aos artigos frequentemente citados juntos, o que pode ser entendido como a perspectiva de que abordam o mesmo assunto, ou sejam, criam um núcleo de abordagem e auxiliam no entendimento do que vem sendo estudado nos últimos anos. Já o *coupling* identifica artigos que tem sido citados com frequencia por vários outros artigos, o que pode significar que o assunto abordado pelos estudos tem sido aceito e levado adiante por outros pesquisadores [8].

A Figura 4.1 traz a nuvem de palavras-chave mais utilizadas e a 4.2 traz a rede dessas palavras utilizadas em conjuntos, ou *clusters*. Ao todo, podem ser identificados cinco *clusters*:

1. **Cluster 1:** 'Data Warehouse' + 'Big Data' + Hive + 'Colum-Oriented model'
2. **Cluster 2:** 'Data Denormalization' + 'Data Warehousing' + 'Databases' + 'Data mining'
3. **Cluster 3:** OLAP + NoSQL + SQL + 'Document-oriented'

4. **Cluster 4:** 'Data Warehouses' + HBase + MongoDB + MapReduce + DBMS + Scalability

5. **Cluster 5:** 'NoSQL Database' + Analytics + 'Big Data Warehouse' + 'Business Intelligence' + Generation

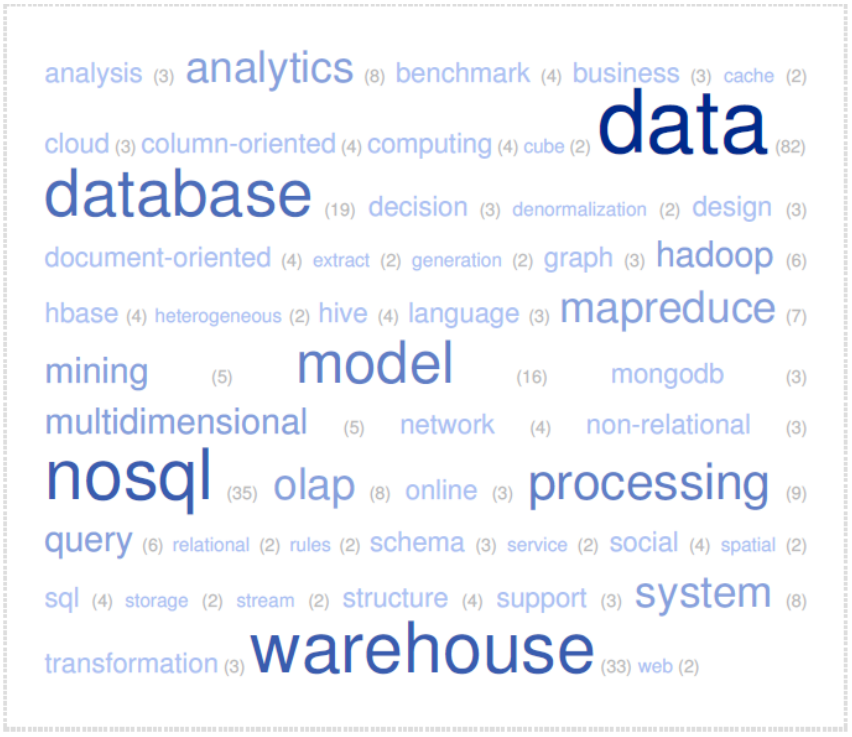


Figura 4.1: TEMAC - Nuvem de palavras-chave

Além da rede de palavras, também foi analisada a rede de co-citações entre trabalhos dos autores, trazida pela Figura 4.3. Nessa análise, é possível observar dois *clusters*, cada um representando épocas diferentes. O primeiro, traz trabalhos mais recentes de autores como Chevalier [6] [39], Kimball [1] e Khaled Dehdouh [40] [2] mais citados conjuntamente, sendo o mais antigo de Kimball, de 2013 e os mais recentes de Chevalier e Khaled Dehdouh, de 2015. Já o segundo *cluster* traz trabalhos de Chaudhri [41], Golfarelli [42], Cuzzocrea [43], E. Dede [44], Cattell [45] e Leavitt [46], sendo o mais antigo o de Chaudhri, de 1997, e os mais recentes de E. Dede e Cuzzocrea, de 2013. Segundo a metodologia TEMAC, essa abordagem diz respeito aos trabalhos que vinham sendo considerados até o presente momento, ou seja, do passado ao presente.

Outro aspecto do TEMAC diz respeito à análise de *coupling*, ou seja, os artigos mais citados pelos artigos retornados pela busca no portal Web of Science nos últimos três anos e, segundo a metodologia, tratam-se das tendências de assuntos abordados para os próximos anos. O que pode-se analisar a partir da Figura 4.4 é que existem três *clusters*.

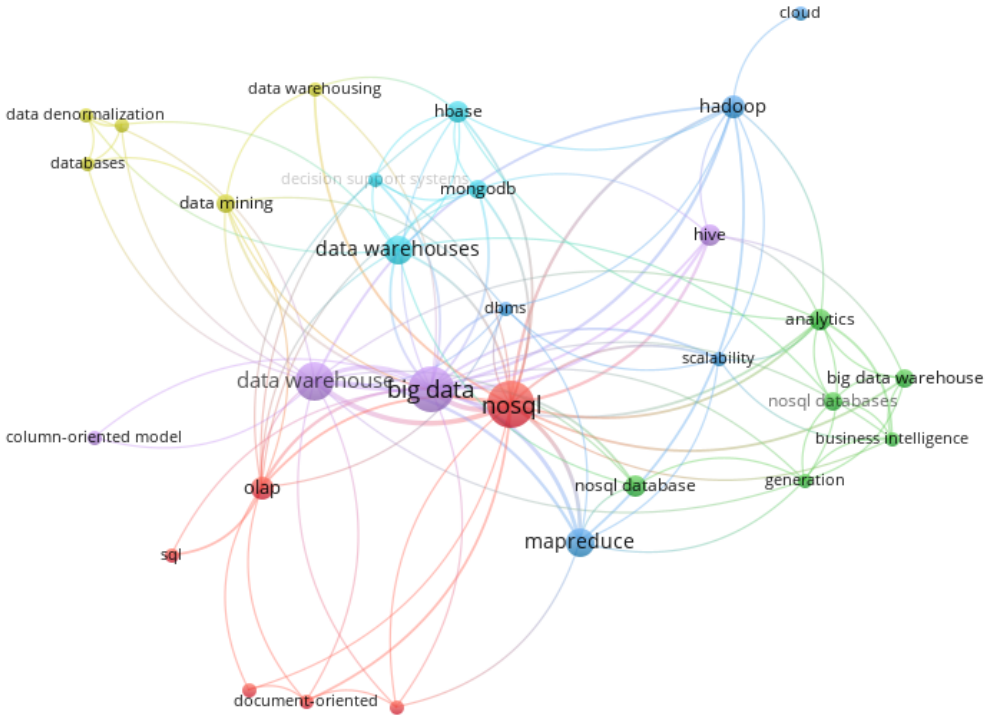


Figura 4.2: TEMAC - Rede de palavras-chave

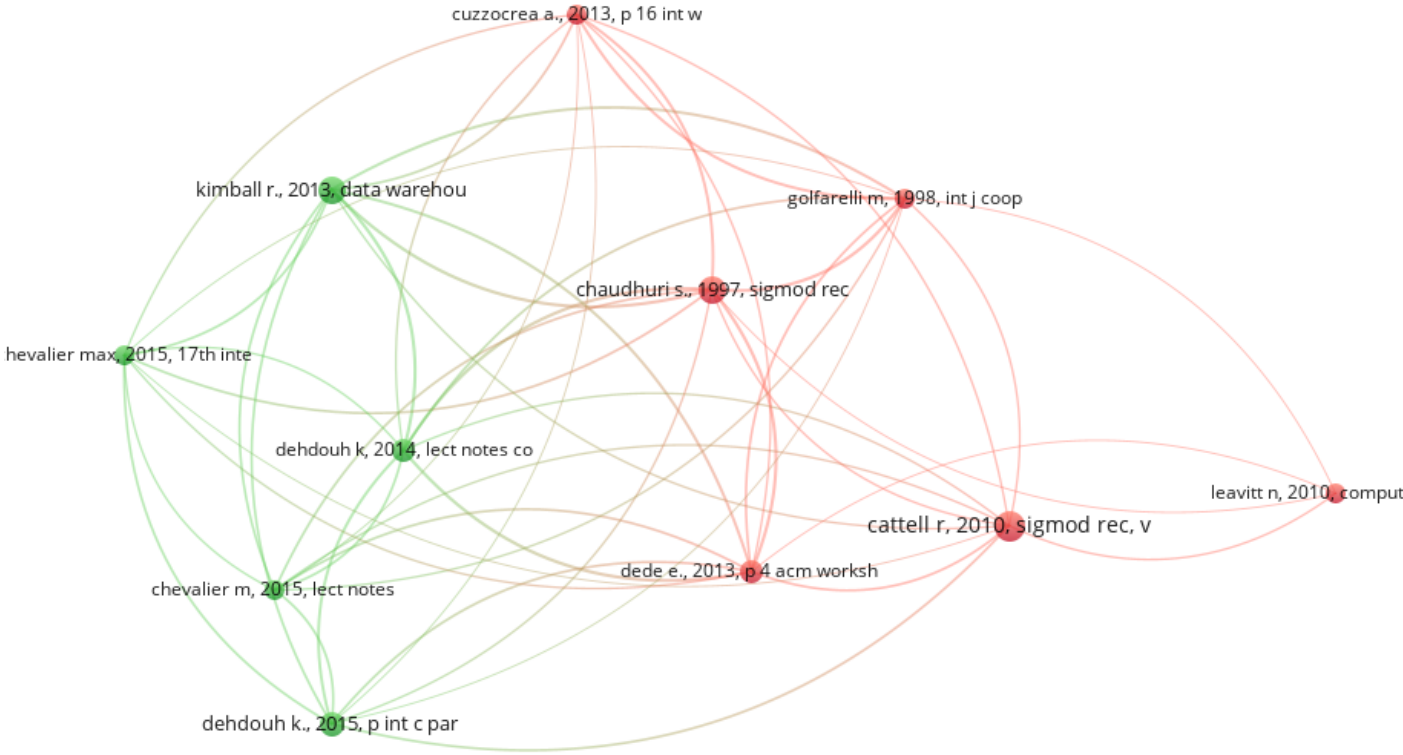


Figura 4.3: Co-citações entre trabalhos dos autores

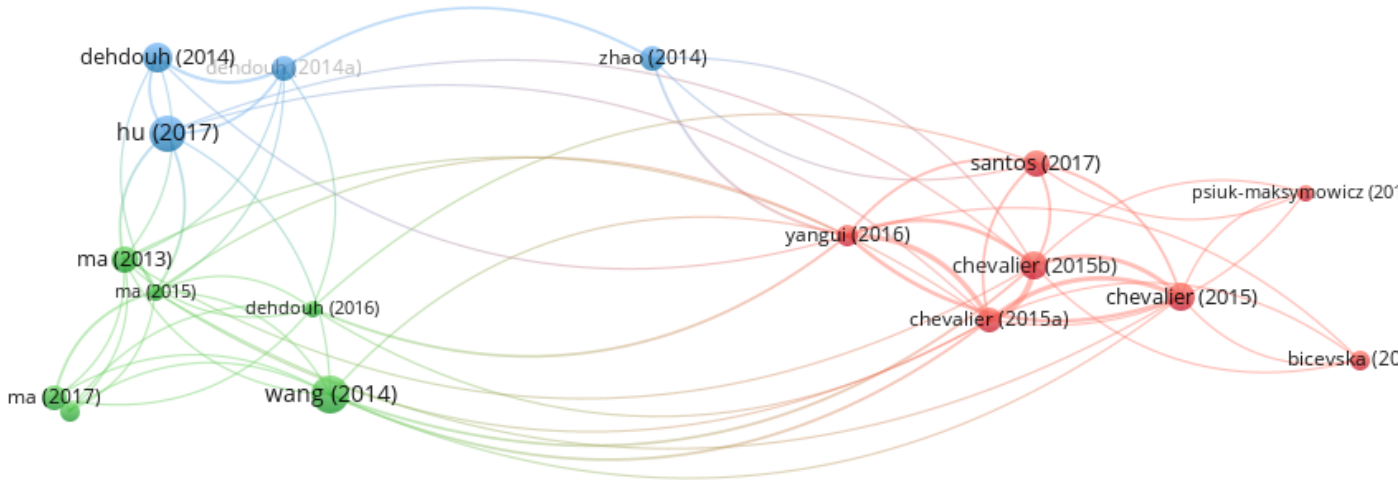


Figura 4.4: *Coupling* dos trabalhos

O primeiro traz os trabalhos de Bicevska [47], Chevalier [6] [38] [39], Psiuk [48], Santos [49] e Yangui [7]. O segundo aborda os estudos de Khaled Dehdouh [14], Liu [50], Kun Ma [51] [52] [53] e Wang [54]. O terceiro traz referências de Khaled Dehdouh [40] [2], Yang Hu [55] e Zhao [56].

Como complemento ao TEMAC, também foi realizada a mesma busca na base científica Scopus, para enriquecer a revisão sistemática deste trabalho. Sendo assim, também foram utilizados os termos *NoSQL* e posteriormente *NoSQL AND (OLAP OR 'DATA WAREHOUSE')*:

- Scopus
  - termo *NoSQL* a partir de 2012: 2.583 resultados
  - *NoSQL and (OLAP or 'DATA WAREHOUSE')* a partir de 2012: 141 resultados

A Figura 4.5 traz um gráfico gerado pela Scopus com os autores que mais publicaram a respeito do tema nos últimos anos. Assim como na Web of Science, os autores O. Teste, El Malki, Kopliku, Max Chevalier e Khaled Dehdouh também aparecem entre os principais nesta última pesquisa.

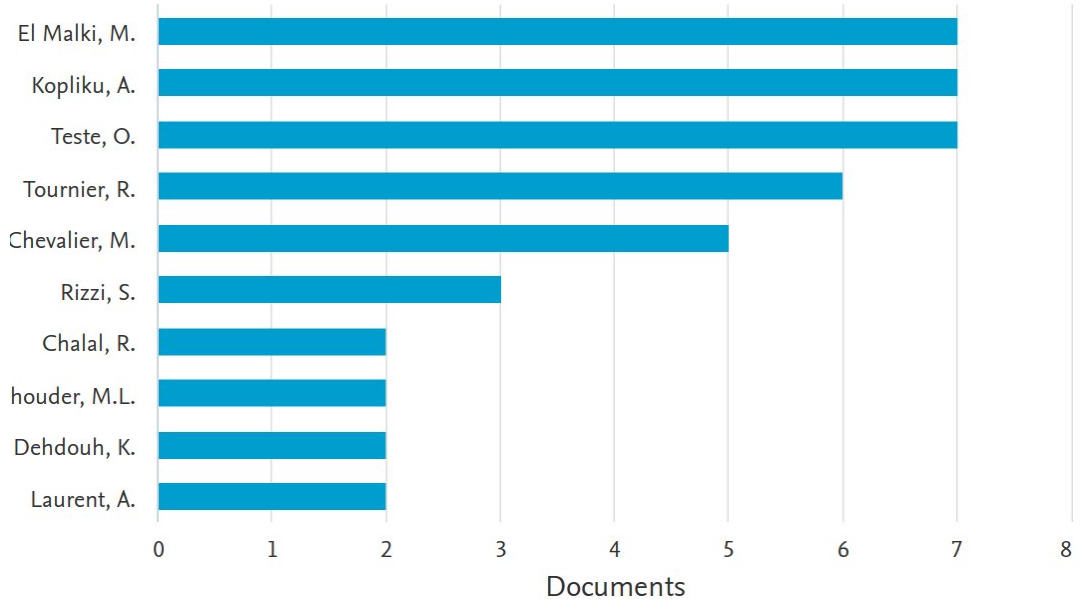


Figura 4.5: Principais autores do tema *OLAP com NoSQL* desde 2012
Fonte: Scopus.

4.2 Estudo De caso

4.2.1 Secretaria de Tecnologia da Informação - STI

O estudo de caso desta pequisa foi realizado no ambiente da Secretaria de Tecnologia da Informação - STI, criada com o intuito de desenvolver as atividades de caráter permanente de apoio, necessárias ao desenvolvimento do ensino,da pesquisa e da extensão no que se refere ao processamento de dados.

De acordo com o site da secretaria, *http://www.sti.unb.br*, esta tem como objetivos:

1. Promover e incentivar a informática na Universidade de Brasília visando obter maior eficiência institucional em todos os níveis.
2. Promover e incentivar a informática na Universidade de Brasília para alcançar maior eficácia no suporte às atividades de ensino, pesquisa, extensão e administração da Instituição.
3. Promover meios para o compartilhamento de recursos computacionais entre a comunidade acadêmica da UnB e as redes de pesquisa nacionais e internacionais.
4. Desenvolver, implantar e manter sistemas em mainframe e em microcomputadores.
5. Supervisionar, coordenar e controlar as atividades relacionadas com pesquisa, desenvolvimento e manutenção de hardware, software e rede de teleprocessamento, assim como as relacionadas com a manutenção ambiental e operação de computadores.

6. Planejar e coordenar a execução de serviços relacionados com o tratamento eletrônico de informações.

Como parte da Secretaria de Tecnologia da Informação, há a Coordenadoria de Estratégia de Dados - CED, responsável pelo apoio à demais áreas da STI, e dentre suas tarefas, estão:

- Auxiliar as demais equipes com serviços e manutenção dos bancos de dados dos sistemas da STI
- Criação e validação de novos modelos de dados para os novos sistemas e para a manutenção dos sistemas atuais da STI
- Criação e manutenção do Sistema de Apoio à Decisão da UnB: BI-UnB
- Resposta às demandas de extrações de dados aos demais departamentos da UnB, aos pesquisadores e cidadãos em geral por meio do SIC- Serviço de Informação ao Cidadão, cujas demandas aumentaram consideravelmente após a sanção em 18 de novembro de 2011 da Lei 12.257/2011, que ficou conhecida como Lei de Acesso a Informação (LAI)
- Unidade responsável pela migração dos dados dos atuais sistemas acadêmicos e administrativos para a nova plataforma SIG, de origem da Universidade do Rio Grande do Norte, em produção desde o início de 2020.

A STI iniciou, por meio da área CED, em 2018, o desenvolvimento de um Sistema de Apoio à Decisão (SAD), denominado BI-UnB . Esse sistema está organizado segundo a arquitetura OLAP proposta por Kimball [1], constituída basicamente por quatro camadas:

1. Camada composta por fontes de dados
2. Camada de Extração de Dados, Transformação e Carga (*ETL*)
3. Camada de armazenamento
4. Camada de apresentação

Com o crescimento massivo do volume de dados das bases da Universidade de Brasília, e com a crescente demanda por novas extrações de dados, uma nova abordagem de arquitetura para o Sistema de Apoio à Decisão faz-se necessária.

4.2.2 Resultados Preliminares e Escolha do SGBD

Para os resultados preliminares, foi utilizado um subconjunto dos dados do Sistema de Apoio à Decisão da UnB para um estudo de viabilidade de migração de uma arquitetura ROLAP, ou seja, em banco de dados relacional Postgres, para uma arquitetura em banco de dados NoSQL, além da escolha de qual banco de dados NoSQL seria o ideal para o projeto final. Dessa forma, no estudo inicial, o objetivo foi testar a implementação de um *Data Warehouse* em uma única tabela e realizar algumas consultas a essa nova base. Assim, um subconjunto da tabela de fatos do modelo Relacional foi utilizado, com aproximadamente 5 milhões de registros, e foram realizados *full outer joins* com as 5 tabelas de dimensões e as 2 tabelas auxiliares de indicadores para a geração do arquivo *.csv*, que foi carregado nas grandes tabelas dos outros dois bancos de dados testados. Todas as colunas de dados foram copiadas, com exceção das chaves primárias das Dimensões e das chaves estrangeiras da tabela de Fatos. Assim, a única chave existente na tabela gerada foi a chave da única e extensa família de colunas contendo 272 colunas originárias do modelo Relacional. Os bancos de dados escolhidos foram o Cassandra e Hbase, juntamente com o Hive, para a implementaçao da camada SQL. Esses foram os escolhidos em virtude dos estudos de caso encontrados no Capítulo 4 para o HBase [22] [23] [24] [25] [26] [27] [29] e para o Cassandra, que apareceu em alguns estudos comparativos de bancos de dados NoSQL [32] [17] [13]. O banco de dados NoSQL orientado a documentos MongoDB foi preterido em virtude de estudos anteriores terem relatado seu pior desempenho frente aos bancos de dados NoSQL colunares para *Data Warehouse* [38] [13] ou então, no estudo em que foi superior, esse fato ocorreu por meio de consultas simples ou com volume de dados pouco expressivo [7].

O modelo escolhido para o experimento preliminar foi o *DLA* do autor Khaled [2], similar ao *MCL0* do autor Chevalier [6], ou seja, todas as dimensões e tabela de fatos integradas em uma única tabela, com as colunas com dados simples. Essa opção surgiu da observação do bom desempenho dessa abordagem na literatura, além da facilidade de implementação, já que os objetivos neste momento foram o de testar a vialidade da solução e escolher o melhor banco de dados para o projeto.

Para os dois casos, Cassandra e HBase/Hive, foram gerados dois códigos de consultas. Sendo a primeira com os seguintes campos:

1. Período acadêmico (desde 1994/1)
2. Campus do curso do aluno
3. Curso do aluno
4. Departamento da Disciplina

5. Nome da Disciplina
6. Quantidade de alunos na turma
7. Taxa de aprovação da disciplina para a turma - Campo realizado com a estrutura *case / when*. Caso a menção do aluno seja SS, MS ou MM, é considerado aprovado.
8. Taxa de reprovação da disciplina para a turma - Campo realizado com a estrutura *case / when*. Caso a menção do aluno seja SR, MI ou MI, é considerado reprovado.

A segunda consulta foi muito semelhante, mas sem as taxas de aprovação e reprovação, e somente com lista simples de alunos:

1. Período (desde 1994/1)
2. Campus do curso do aluno
3. Curso do aluno
4. Departamento da Disciplina
5. Nome da Disciplina
6. Alunos na turma

No estudo preliminar, chegou-se à conclusão do melhor banco de dados sem a necessidade de um teste de desempenho, pois, analisando as vantagens e desvantagens de cada abordagem, a solução HBase/Hive apresentou-se mais vantajosa já na primeira etapa qualitativa. Tanto Cassandra quanto Hbase possuem pontos positivos e negativos. Em relação ao Cassandra, a consulta com as taxas de aprovação e reprovação não pôde ser realizada, já que a linguagem CQL possui algumas limitações, e não permite o uso da estrutura *case / when* e consultas com dados agrupados, e isso impede sua utilização com a ferramenta Saiku, utilizada em conjunto com a ferramenta Pentaho pela Universidade de Brasília para a geração de consultas *adhoc*. A tabela 5.1 sintetiza as observações, sendo que o sinal de positivo indica que um SGBD foi melhor que o outro:

Em relação à instalação, HBase/Hive possuem uma compatibilidade com mais versões do Java que o Cassandra. O Cassandra exige determinada versão da linguagem *Java* e outra determinada versão da linguagem Python, o que pode ser um problema para o pesquisador que tiver o ambiente com versões diferentes das necessárias. Nem sempre um *downgrade* de versão é um processo simples.

Na comparação das linguagens, embora a CQL seja próximo à linguagem SQL, há certas limitações que não existem na linguagem HiveQL, como a possibidade de agrupamento dos resultados.

Tabela 4.1: Vantagens e Desvantagens entre Apache Hive e Cassandra

Solução	Cassandra	Hive
Instalação	-	+
Linguagem	-	+
Compatibilidade OLAP	-	+
Criação Esquema	-	+
Criação Tabela	-	+
Artigos Relacionados	-	+
Configuração	+	-
Topologia	+	-

Outra vantagem de HBase/Hive diz respeito à compatilidade maior com a ferramenta OLAP utilizada na Universidade de Brasília, o Pentaho, que possui conexão com Hive e Phoenix para o *plugin* de relatórios *ad hoc*, ao passo que o Cassandra só possui compatibilidade para consultas fixas por meio de *Views*. Ou seja, a cada novo relatório utilizando o Cassandra como SGBD, seria necessário ao usuário requisitar à equipe de Desenvolvimento por uma nova consulta.

Para a criação do esquema e das tabelas, o Hive/HBase volta a ter vantagem. No Cassandra, para a criação do esquema é necessário conhecer o tipo de estratégia de replicação do dados e o fator de replicação, ao passo que no HBase/Hive, a criação segue o mesmo modelo do SQL em banco de dados relacionais.

Outra vantagem do HBase encontra-se no quesito artigos de trabalhos relacionados a *Data Warehouse*, já que boa parte dos artigos encontrados por meio do TEMAC que abordam o tema migração de DW para bancos NoSQL trazem Hive ou HBase como escolha de banco de dados.

Na parte de configuração, HBase/Hive são penalizados por dependerem da prévia configuração do Hadoop, que possui muitos arquivos de configuração e que podem variar a cada versão do *framework*.

Sobre a topologia, os dois modelos são muito bons quanto à distribuição dos recursos. O Cassandra leva vantagem por não ter um nó principal, que no caso do Hbase pode representar a paralisação do sistema em caso de falha do nó principal, mas isso pode ser mitigado por um nó reserva que pode ser acionado a qualquer momento.

Dessa forma, foram adotadas duas abordagens preliminares: migração para Cassandra e para HBase. Confirmou-se a viabilidade para a migração da arquitetura, que foi realizada com sucesso, sendo feitas consultas que demonstraram o funcionamento na prática da solução pesquisada na literatura. Entre vantagens e desvantagens, o SGBD HBase demonstrou ser mais vantajoso principalmente pelo fato de ter mais compatibilidade com a ferramenta OLAP Pentaho, por meio do Apache Hive ou Phoenix, já de uso pela Uni-

versidade de Brasília, e por possibilitar o aproveitamento de grande parte do trabalho já feito em relatórios junto aos usuários. Além disso, foram encontrados mais estudos científicos relacionados a esta pesquisa que relatam o bom desempenho do HBase, o SGBD escolhido para o projeto final.

Capítulo 5

Arquitetura

5.1 Arquitetura Anterior Relacional do Sistema de Apoio à Decisão da UnB

O *Data Warehouse* da UnB estava previamente armazenado em um banco de dados relacional *Postgres* e foi desenvolvido de acordo com o modelo da Figura 5.1. Há a tabela de fatos, com aproximadamente 13 milhões de registros em 2019, mais 5 tabelas de dimensões, e 2 tabelas auxiliares de indicadores, sendo uma para alunos e outras para docentes. Dentre os indicadores, estão aqueles que apontam se determinado aluno recebe auxílio estudantil, se exerce atividade de monitoria em alguma disciplina, por exemplo. A tabela de fatos une as informações da vida acadêmica do aluno, como seu curso, suas disciplinas cursadas, turmas das disciplinas, professores, menções, coeficiente de rendimento acadêmico (IRA), entre outras, para cada período de tempo letivo. Os relatórios e o processo de ETL são feitos a partir do conjunto de ferramentas Pentaho.

As tabelas do BI-UnB são as seguintes:

- **Tabelas de Dimensões:**
  - **Aluno**: Dados relativos ao aluno. Ex: matrícula, nível acadêmico, opção escolhida, etc
  - **Curso**: Dados relativos ao curso. Ex: Nome, nível acadêmico, código, departamento, datas de início e fim, etc
  - **Disciplina**: Dados relativos às disciplinas. Ex: Nome, código, número de créditos, departamento, etc
  - **Opção**: Dados relativos às opções dos cursos. Ex: Grau acadêmico, código, nome, créditos para formatura, etc

  - **Pessoa**: Dados pessoais relativos a docentes e alunos. Ex: Nome, data de nascimento, passaporte, CPF, RG, etc.
  - **Período**: Dados relativos ao período letivo. Ex: período, nível do perído, indicador de período atual, etc.

- **Tabela de Fatos:**
  - **Dados Acadêmicos:** Dados relativos à vida acadêmica do aluno. Ex: Menções nas disciplinas, turmas, rendimento acadêmico (IRA), quantidade de reprovações, quantidade de trancamentos, posição no fluxo do curso, etc.

- **Tabela de Indicadores:** Tabelas com colunas binárias *sim/não*, feitas para filtros rápidos
  - **Indicadores de Alunos**: Indicadores a respeito dos alunos. Ex: Se o aluno está com curso trancado, se possui auxílio estudantil, se possui monitoria, se o aluno é possível formando, etc.
  - **Indicadores de Docentes**: Indicadores a respeito dos docentes. Ex: Se o docente ocupa função administrativa, se participa de algum curso de Extensão, se orienta algum aluno, se está afastado por saúde, etc.

A arquitetura do BI-UnB segue a arquitetura tradicional proposta por Kimball [1], de acordo com a Figura 5.2. As fontes de dados do SAD são os sistemas acadêmicos da UnB citados no Capítulo 1: Sigra, Sippos, SAE, Sipic e Sipes. Os processos de ETL são realizados por meio da ferramenta PDI Kettle do conjunto de soluções Pentaho. Os dados são carregados nas tabelas do modelo apresentado na Figura 5.1 no banco de dados relacional PostgreSQL. Há uma camada intermediária chamada camada semântica, em que atua a ferramenta Mondrian em que as colunas dessas tabelas são mapeadas em dimensões e métricas para a ferramenta de geração de relatórios Saiku, ambas também integrantes das soluções Pentaho. Além disso, o SAD possui as características citadas por Inmon [11] de ser integrado, já que as informações são consolidadas em um ambiente íntegro, variável no tempo, pois todos os registros do SAD são identificados pelo período da UnB, e não volátil, a partir do momento em que o processo de ETL desenvolvido é sempre incremental, os registros nunca são excluídos.

O processo de criação dessa arquitetura segue o processo de desenvolvimento proposto pelo ciclo de vida de DW de Kimball [1] e apresentado pela Figura 2.2 do Capítulo 2.

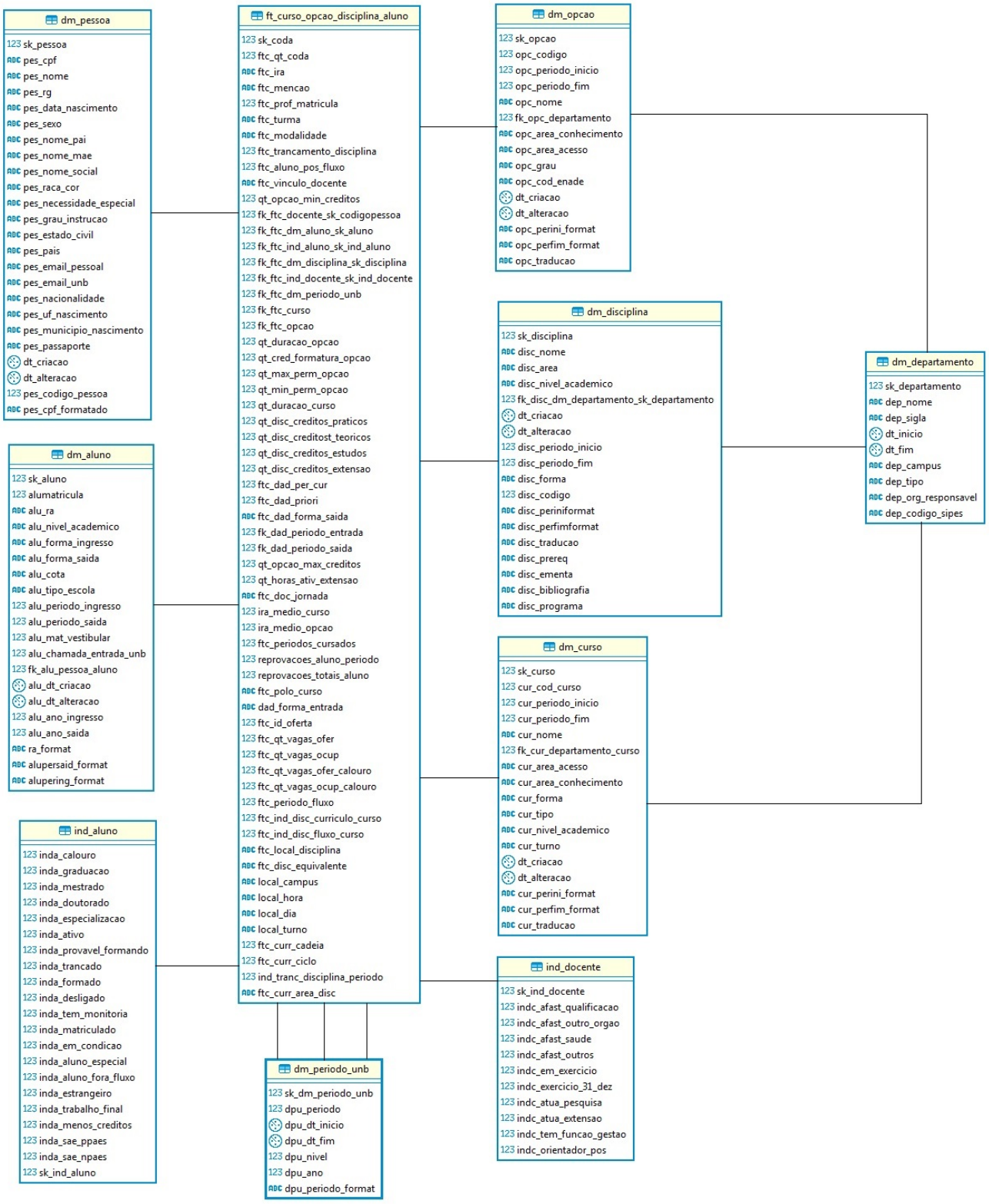


Figura 5.1: Modelo DW-UnB

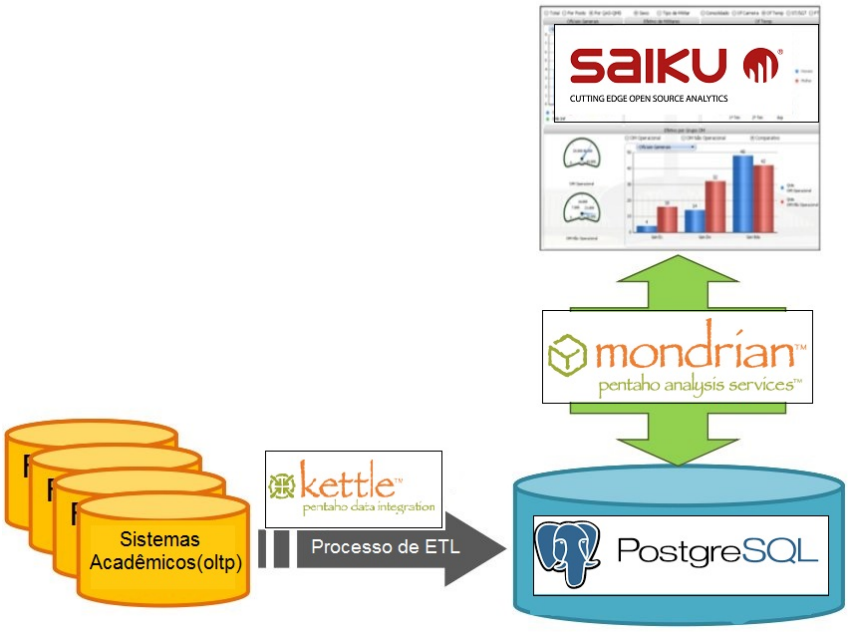


Figura 5.2: Arquitetura tradicional do BI-UnB

5.2 Extensão da Arquitetura

A extensão da arquitetura para o BI-DW possui como base a pesquisa na literatura pelos estudos que demonstram um melhor desempenho para o banco de dados HBase em arquitetura de única tabela em família de colunas com dados agregados, de acordo com Chavalier [6] e Khaled Dehdouh [40], em suas abordagens *MLC1* e *DLA-CF*, respectivamente. Como citado no Capítulo 2, as tabelas do banco de dados HBase podem ser mapeadas como tabelas externas tanto do Apache Hive quanto do Apache Phoenix , de forma que consultas na linguagem SQL podem ser utilizadas para acesso aos dados das tabelas HBase.

Outra vantagem no uso tanto do Apache Hive quanto do Apache Phoenix diz respeito à sua capacidade de integração à ferrameta Mondrian da solução Pentaho. Assim, houve uma alteração na camada semântica dessa ferramenta, mas os relatórios até então produzidos foram mantidos praticamente inalterados. O banco de dados HBase foi escolhido pelo fato de permitir as soluções propostas por Kahled Dehdouh [40] e Chevalier [6] e por nativamente permitir uma integração com Hive e Phoenix. Houve uma etapa adicional na configuração física do SGBD, já que o HBase possui algumas estratégias de particionamento dos dados. Em relação à etapa de ETL, houve uma alteração na reunião das dimensões em famílias de colunas lógicas para então serem carregadas na grande tabela do HBase. A Figura 5.3 permite verificar as diferenças em relação à Figura 5.2 que representa a arquitetura atual do BI-UnB. Na arquitetura extendida, que trata o processamento distribuído dos dados, há a presença do *HBase Master* e do Hive, ou Phoenix,

no nó principal do *cluster* Hadoop e dos *Region Servers* do HBase nos nós secundários do *cluster*, juntamente com os *datanodes* do Hadoop, representados pelas caixas roxas na Figura 5.3.

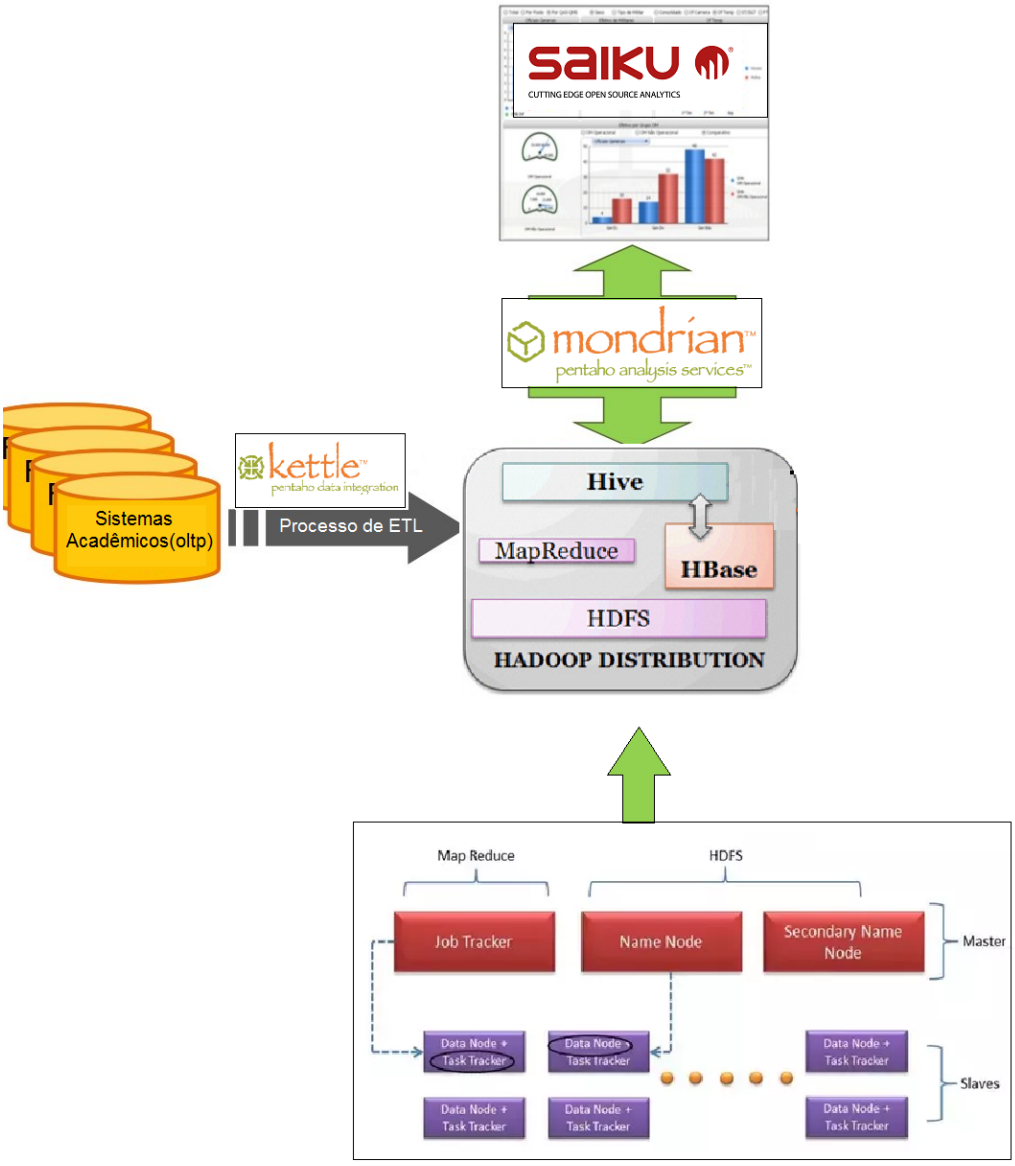


Figura 5.3: Extensão da arquitetura do BI-UnB

5.3 Extensão do Processo de Desenvolvimento

A arquitetura extendida demandou algumas alterações no processo de desenvolvimento proposto por Kimball [1] e representado na Figura 2.2 do Capítulo 2. Além de algumas

alterações nas etapas já existentes, uma nova etapa foi proposta ao processo. A Figura 5.4 apresenta o novo esquema com a nova etapa *Projeto do Cluster* em destaque.

Em relação às etapas existentes, o projeto físico tornou-se diferente ao tradicional pelo fato de a arquitetura em uma única tabela com famílias de colunas exigir alterações no modelo geralmente criado para um banco relacional. Entre as alterações, estão a ausência de relacionamentos, causando a retirada de chaves estrangeiras das tabelas físicas e a ausência de *constraints* para a verificação da integridade dos dados, já que essa verificação foi resolvida nas rotinas de ETL. Além disso, o formato em família de colunas também ocasionou mudanças no processo de carga dos dados.

As etapas *Projetar Aplicação de BI* e *Implementar Aplicação de BI* também foram alteradas pelo fato de serem necessárias alterações na camada semântica da ferramenta Mondrian, em que houve a necessidade de configuração da integração com HBase e reformulação do mapeamento dos objetos *OLAP*, ou seja, as dimensões e métricas.

Quanto à implantação, também houve a necessidade de configuração do *cluster*, instalação do sistema operacional Linux, do Apache Hadoop, Phoenix e HBase.

A nova etapa, *Projeto do Cluster*, surgiu da natureza distribuída do *framework Hadoop*, do qual fazem parte o banco de dados escolhido HBase e a solução Phoenix para abstração da linguagem SQL para as operações de *MapReduce*. Nessa etapa, foram mapeadas as necessidades e as características do particionamento dos dados em famílias de colunas e o fator de replicação, configurado para três, ou seja, há duas outras réplicas existentes dos dados distribuídas pelos nós do *cluster*.

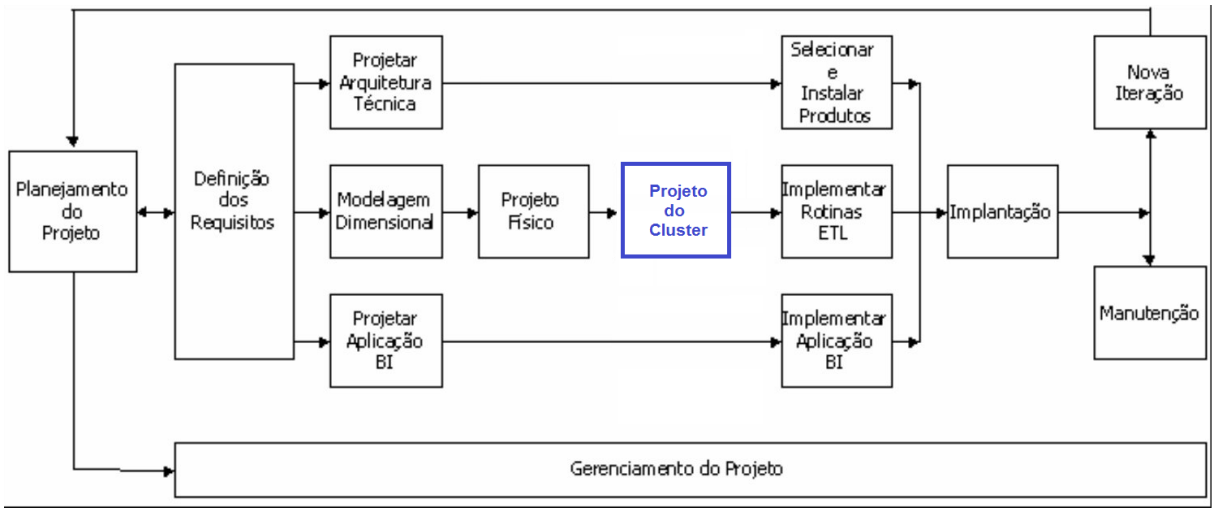


Figura 5.4: Extensão do processo de desenvolvimento de um *Data Warehouse*

Sendo assim, a nova proposta a partir do eixo da modelagem dimensional ficou da seguinte forma:

- Modelagem Dimensional

- Projeto Físico
- ***Projeto do Cluster***
- Implementar Rotinas de ETL

Capítulo 6

Implementação

Neste capítulo, será um pouco mais detalhada a implementação do *cluster* Hadoop/HBase/Phoenix, a criação da grande tabela neste banco de dados, a conexão com a ferramenta Pentaho, ou seja, as etapas requeridas para a realização do projeto distribuído do SAD da Universidade de Brasília.

6.1 Implementação do *Cluster*

O *cluster* utilizado nesse trabalho foi composto por seis máquinas, todas com 8 GB de memória RAM, 80 GB de disco e 4 núcleos ou *cores*, totalizando 48 GB de memória RAM, 480 GB de disco e 32 *cores*, e todas elas executando o Sistema Operacional Linux Centos 7. Em um *cluster* Hadoop, os nós precisam ser identificados por nomes próprios no arquivo */etc/hostname* para que possam ser incluídos nos arquivos de configuração do *cluster* e reconhecidos pelos outros nós por meio do arquivo */etc/hosts*:

- H-SSI-HADOOPMASTERBD (nó principal)
- H-SSI-HADOOPSLAVEBD1
- H-SSI-HADOOPSLAVEBD2
- H-SSI-HADOOPSLAVEBD3
- H-SSI-HADOOPSLAVEBD4
- H-SSI-HADOOPSLAVEBD5

As versões escolhidas dos *softtwares* para esse projeto foram:

- Apache Hadoop 2.7.0
- Apache Zookeeper 3.4.14

- Apache HBase 1.3.1
- Apache Phoenix 4.11.0

É importante notar que a versão do Apache Hadoop influencia nas versões de todos os outros *softwares*, pois a versão do Apache HBase tem que ser específica para a versão do Hadoop escolhida, assim como a versão do Apache Phoenix tem que ser compatível com a versão do Apache HBase. A versão do Apache Hadoop escolhida foi a **2.7.0** em função de ser uma versão bastante estável do *framework* e por ter um grande suporte da comunidade de *software livre* de fácil acesso na *Internet*.

Nas seções seguintes, serão explicadas as configurações efetuadas em cada componente do *cluster*, que são feitas por meio de arquivos *.xml*, que por sua vez possuem propriedades que definem o comportamento destes componentes durante a execução das requisições. Antes das configurações, cada nó teve que ter seu *firewall* alterado com regras para que todos os nós se comunicassem sem problemas de permissões. Além do *firewall*, também foram geradas chaves *ssh*, que foram compartilhadas entre os nós para a criação de uma rede de confiança segura entre as máquinas.

6.1.1 Implementação do *Hadoop: HDFS, YARN e MAPREDUCE*

Os arquivos do Apache Hadoop devem ser copiados para os seis nós do *cluster* em uma pasta do sistema operacional chamada */hadoop*. Primeiramente, configura-se o sistema de arquivos HDFS. No nó principal, deve ser criada a pasta */data/namenode*, que será a pasta em que ficarão os arquivos do *NameNode*, responsável por gerenciar os *DataNodes* do sistema de arquivos distribuído HDFS. Nos cinco nós secundários, deverá ser criada uma pasta */data/datanode*, para os arquivos dos *DataNodes*. Em todos os nós, deverá ser configurado o arquivo *hdfs-site.xml*, de acordo com a tabela 6.1, e o arquivo *slaves*, que deverá ser editado com os nomes de todos os nós do *cluster*, de acordo com listagem descrita no início da seção 6.1.

Tabela 6.1: Propriedades do arquivo *hdfs-site.xml*

Propriedade	**Valor**	**Descrição**
dfs.namenode.name.dir	/hadoop/data/nameNode	Pasta dos arquivos do *NameNode*
dfs.datanode.data.dir	/hadoop/data/dataNode	Pasta dos arquivos dos *DataNodes*
dfs.replication	3	Fator de replicação dos dados no HDFS

Após a configuração do arquivo *hdfs-site.xml*, utiliza-se o comando *hadoop namenode -format* para formatar o sistema HDFS e deixá-lo pronto para o uso.

Prosseguindo com as configurações, para o gerenciador de *jobs YARN*, deve ser utilizado o arquivo *yarn-site.xml*. A configuração da utilização de memória de cada nó pelo YARN é muito importante na configuração de todo o *cluster*, já que se for feita de forma incorreta, poderá comprometer todo o funcionamento dos *jobs mapreduce*, pois em caso de falta de memória, todo o proceso será suspenso. De acordo com o *site* oficial ***https://docs.cloudera.com***, em nós de 8 GB de RAM, devem ser reservados 2 GB para o sistema operacional e mais 1 GB para o Hbase, sobrando 5 GB totais para utilização do YARN. Também de acordo com o *site*, cada *container* utilizado para os *jobs* de *mapreduce* deve ter no mínimo 512 MB, o que já determina o número de *containers* por nó e a quantidade de memória RAM disponível para cada um, levando-se em consideração o total de 5 GB disponíveis. De posse dessas informações, as propriedades do arquivo *yarn-site.xml* ficam de acordo com a tabela 6.2.

Tabela 6.2: Propriedades do arquivo *yarn-site.xml*

Propriedade	**Valor**	**Descrição**
yarn.resourcemanager.hostname	H-SSI-HADOOPMASTERBD	Gerenciador de recursos
yarn.nodemanager.resource.memory-mb	5120	Memória do Yarn
yarn.scheduler.maximum-allocation-mb	5120	Máx mem. alocável
yarn.scheduler.minimum-allocation-mb	1024	Min mem. alocável
yarn.app.mapreduce.am.resource.mb	2048	Mem. para *mapreduce*
yarn.app.mapreduce.am.command-opts	-Xmx1648m	Mem. da *JVM*

Em relação à configuração do componente de *MapReduce*, esta é feita por meio do arquivo *mapred-site.xml*, de acordo com a tabela 6.3. Os valores também foram calculados da mesma forma que os do arquivo *yarn-site.xml*.

Tabela 6.3: Propriedades do arquivo *mapred-site.xml*

Propriedade	**Valor**	**Descrição**
mapreduce.framework.name	yarn	Define o Gerenciador de *jobs*
mapreduce.map.java.opts	-Xmx829m	Memória da *JVM* para *map*
mapreduce.reduce.java.opts	-Xmx1630m	Memória da *JVM* para *reduce*
mapreduce.map.memory.mb	1024	Memória para *map*
mapreduce.reduce.memory.mb	2048	Memória para *reduce*

Após essas configurações, o início do funcionamento do sistema HDFS, do *Yarn e MapReduce* são feitos pelos comandos *start-dfs.sh* e *start-yarn.sh*, respectivamente. O funcionamento do sistema HDFS pode ser verificado pelo endereço ***http://h-ssi-hadoopmasterbd:50070/***, conforme Figura 6.1, e o funcionamento do YARN pode

ser conferido pelo endereço ***http://h-ssi-hadoopmasterbd:8088/cluster***, conforme Figura 6.2.

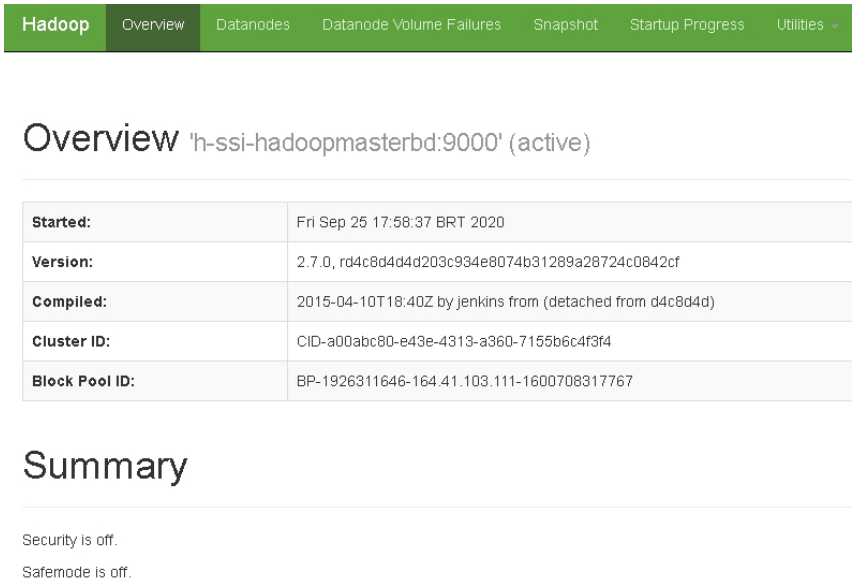


Figura 6.1: Tela de acompanhamento do sistema HDFS

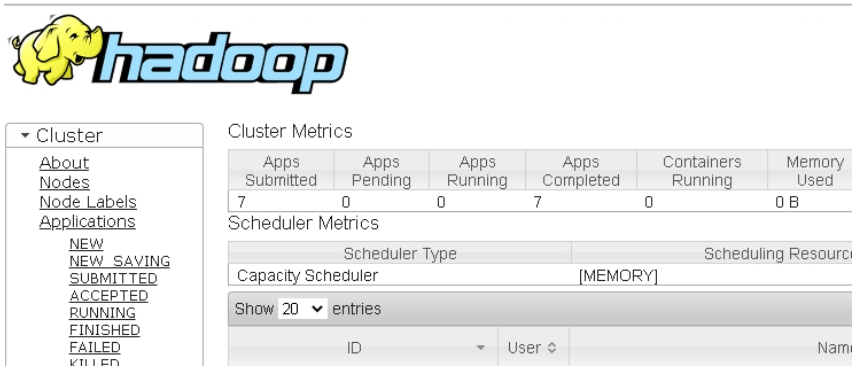


Figura 6.2: Tela de acompanhamento do YARN

6.1.2 Implementação do *Apache Zokeeper*

O Apache Zookeeper é um componente **obrigatório** para o funcionamento do Apache HBase, pois é este componente que monitora o funcionamento dos *RegionServers*, e também é responsável por outras tarefas, como a sincronização dos dados entre os nós do *cluster* HBase, por exemplo. Ao contrário do Apache Hadoop, os arquivos do Zookeeper devem ser instalados em um número ímpar de nós, justamente para se permitir a existência do *quórum Zookeeper* em caso de necessidade de eleição de um novo nó principal do *cluster* Zookeeper, caso o atual falhe de alguma forma. Sendo assim, os nós

escolhidos foram os H-SSI-HADOOPMASTERBD, H-SSI-HADOOPSLAVEBD1 e H-SSI-HADOOPSLAVEBD2. Em cada um dos três, foi criada uma pasta */zookeeper* em que os arquivos foram instalados e, nesta pasta, houve a criação de um arquivo chamado *myid*, com números de 1 a 3, para identificação da sequência de cada máquina. A tabela 6.4 traz a configuração do arquivo *zoo.cfg*, que define as propriedades do componente.

Tabela 6.4: Propriedades do arquivo *zoo.cfg*

Propriedade	**Valor**	**Descrição**
dataDir	/zookeeper/data	Pastas dos arquivos
dataLogDir	/zookeeper/dataLog	Pasta dos *Logs*
clientPort	2181	Porta de conexão do cliente
server.1	h-ssi-hadoopmasterbd:2888:3888	Servidor 1
server.2	h-ssi-hadoopslavebd1:2888:3888	Servidor 2
server.3	h-ssi-hadoopslavebd2:2888:3888	Servidor 3

Após as configurações, o comando *zkServer.sh start* nos três nós escolhidos inicia o serviço do Zookeeper nessas máquinas.

6.1.3 Implementação do Apache HBase

Para a configuração do Apache HBase, uma pasta */hbase* deve ser criada nos seis nós do *cluster* para a instalação dos arquivos. Na pasta */hbase/conf*, está o arquivo *regions*, que, assim como o arquivo *slaves* do Apache Hadoop, deve conter os nomes de todas as máquinas do *cluster*. As propriedades são definidas no arquivo *hbase-site.xml*, de acordo com a tabela 6.5. Para que a integração entre HBase e Hadoop ocorra de forma correta, esse arquivo deve ser copiado para a pasta */hadoop/etc/hadoop*. Além disso, a pasta */hbase/conf* deve ser inserida na propriedade *classpath* do Hadoop por meio do arquivo *hadoop-env.sh*.

Tabela 6.5: Propriedades do arquivo *hbase-site.xml*

Propriedade	**Valor**	**Descrição**
hbase.rootdir	hdfs://h-ssi-hadoopmasterbd/hbase	Pasta raíz HDFS
hbase.tmp.dir	/hbase/tmp	Arq. temporários
hbase.cluster.distributed	true	Modo distribuído
hbase.zookeeper.quorum	h-ssi-hadoopmasterbd, h-ssi-...	3 nós Zookeeper
phoenix.queryserver.serialization	JSON	Serialização Phoenix
hbase.region.replica.replication.enabled	true	Habilitar replicação
hbase.meta.replica.count	3	Replicação = 3

Além do arquivo *hbase-site.xml*, também é preciso definir a propriedade ***export HBASE-MANGES-ZK=false*** no arquivo *hbase-env.sh*, que indica que o HBase não irá gerenciar o Zookeeper, pois há um módulo que possui esse componente embutido nativamente, mas por questões de desempenho, é uma melhor estratégia instalar o Zookeeper separadamente, como foi feito neste trabalho.

Após as configurações, inicia-se o Apache HBase pelo comando *start-hbase.sh*. O funcionamento do Apache HBase pode ser verificado por ***http://h-ssi-hadoopmasterbd/master-status***, conforme Figura 6.3.

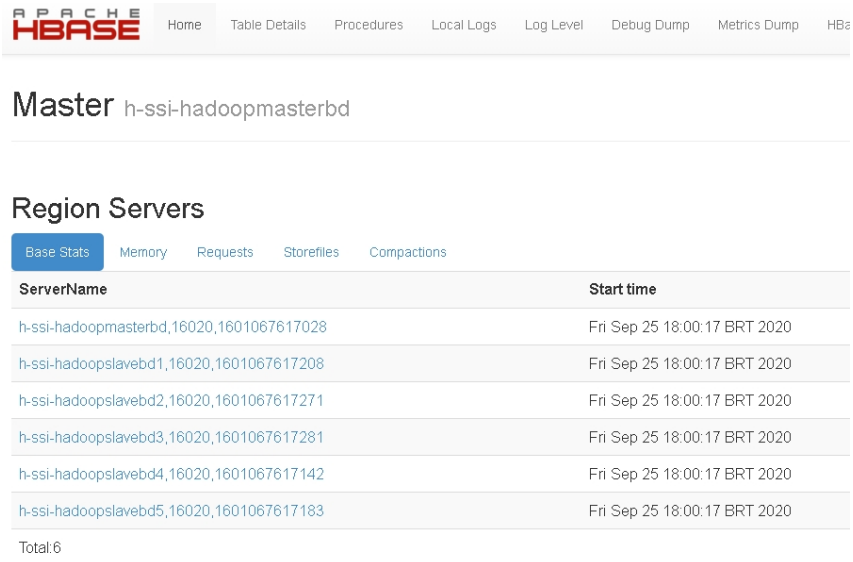


Figura 6.3: Tela de acompanhamento do HBase

6.1.4 Implementação do Apache Phoenix

Os arquivos de configuração do Apache Phoenix devem ser instalados na pasta */phoenix* do nó principal, em que funcionará o servidor de consultas Phoenix. No entanto, alguns de seus arquivos de bibliotecas devem ser copiados para todos os nós do *cluster*, para a pasta */hbase/lib* e */hadoop/lib*, já que o Apache Phoenix cria alguns *jobs MapReduce* tanto no momento de criação dos índices da tabela, quanto na inserção de dados de forma mais rápida pela estratégia conhecida como *Bulking Insert*, e todos os nós podem executar algumas requisições vindas desse componente. Como exemplo, o arquivo ***phoenix-core-4.11.0-HBase-1.3.jar*** deve ser copiado para a pasta */hbase/lib* e os arquivos ***tephra-api-0.14.0-incubating.jar, twill-api-0.8.0.jar, libthrift-0.9.3.jar*** para a pasta */hadoop/lib*. Consultas pontuais na linguagem *SQL* pelo Apache Phoenix podem ser feitas após a conexão do componente com o HBase por meio do comando *sqlline.py localhost:2181/hbase-unsecure*. A partir desse primeiro acesso, é criada uma tabela

no HBase chamada *SYSTEM.CATALOG* que possui os metadados de mapeamento entre as tabelas Phoenix e HBase. Já o servidor de consultas, ou *Query Server*, com o qual a ferramenta Pentaho se conecta, é inicado pelo comando *python /phoenix/bin/queryserver.py*.

6.2 Criação da Tabela e dos Índices no Apache HBase

6.2.1 Criação da Tabela

Criou-se uma tabela no Apache HBase por meio do Apache Phoenix chamada FT-DWSED-FATO, que seguiu as abordagens *DLA-CF* de Khaled [2] e *MLC1* de Chevalier et al. [6], ou seja, a tabela de Fatos e as tabelas de Dimensões do modelo relacional foram transformadas em uma única tabela, com uma Família de Colunas para cada tabela de Dimensões e outra para a tabela de Fatos. Sendo assim, foram criadas as Famílias de Colunas: *Fato*, *Aluno*, *Curso*, *Opção*, *Disciplina*, *Docente* e *Período*. Todas essas Famílias de Colunas possuem as propriedades da tabela 6.6.

Tabela 6.6: Propriedades das famílias de Colunas

Propriedade	**Valor**	**Descrição**
IN-MEMORY	true	blocos HDFS em *cache*
COMPRESSION	GZ	Arquivo da FC utiliza compactação gzip
BLOCKCACHE	true	blocos HDFS em *cache*
REPLICATION-SCOPE	3	Fator replicação 3

Além do fato de os dados de uma mesma Família de Colunas serem armazenados no mesmo espaço em disco para o ganho de desempenho, algumas propriedades ainda foram configuradas para um ganho ainda maior. Descritas na tabela 6.6, entre essas propriedades está a compressão por meio de compactação *gzip*, que possibilita uma ocupação menor de espaço dos arquivos, o que diminui o tempo de busca em arquivos de tamanhos massivos. O fator de replicação igual à 3, além de permitir uma maior disponibilidade e intolerância à falha, também permite um maior ganho de desempenho em leitura, já que o nó que responder primeiro à requisição será o responsável pelo processamento. A propriedade *BLOCKCACHE* em conjunto com *IN-MEMORY* permite que os blocos de dados do sistema HDFS sejam armazenados na memória *cache* do *Java Heap Space* dos *RegionServers*, sempre que possível, o que também oferece um ganho considerável no desempenho de leitura.

A Tabela 6.7 traz as principais colunas da Família de Colunas *FATO*, que representam ou métricas ou características dos alunos que variam ao longo do tempo, sendo que algumas podem ser definidas como *Dimensões Degeneradas* no modelo ROLAP, ou seja,

possuem uma característica de qualificar o dado, mas não estão em uma tabela separada, e sim na tabela de Fatos.

Tabela 6.7: Principais Colunas da Família de Colunas *FATO*

Coluna	**Tipo**	**Descrição**
QT-ALUNOS	integer	Métrica de quantidade de alunos
IRA	varchar(10)	Valor do IRA do aluno
MENCAO	char(2)	Menção do aluno na discipina
POS-FLUXO	integer	Posição no fluxo do aluno
MIN-CRED-FORMATURA	integer	Mínimo de créditos para formatura
MAX-PERM-OPCAO	integer	Máximo de períodos para permanecer no curso
MIN-PERM-OPCAO	integer	Mínimo de períodos para permanecer no curso
QT-OPCAO-MAX-CREDITOS	integer	Máximo de créditos em um mesmo período
QT-PERIODOS-CURSADOS	integer	Quant. de períodos cursados pelo aluno
QT-REPROVACOES-TOTAIS	integer	Quant. de reprovações do aluno no curso

A Tabela 6.8 traz as principais colunas da Família de Colunas *ALUNO*, que representam as características pessoais e acadêmicas dos alunos.

Tabela 6.8: Principais Colunas da Família de Colunas *ALUNO*

Coluna	**Tipo**	**Descrição**
MATRICULA	integer	Matrícula do aluno
ALU-NIVEL	varchar(20)	Nível acadêmico do aluno
ALU-COTA	varchar(20)	Cota racial ou social do aluno
ALU-TIPO-ESCOLA	varchar(10)	Tipo de escola: pública ou privada
ALU-PERIODO-INGRESSO	integer	Período de ingresso do aluno na UnB
ALU-PERIODO-SAIDA	integer	Período de saída do aluno na UnB
ALU-CPF	char(11)	CPF do aluno
ALU-NOME	varchar(200)	Nome do aluno
ALU-SEXO	varchar(10)	Sexo do aluno
ALU-NACIONALIDADE	varchar(20)	Nacionalidade do aluno
IND-ALU-CALOURO	bit	Indica se o aluno é calouro
IND-ALU-FORMANDO	bit	Indica se o aluno é provável formando
IND-ALU-MONITORIA	bit	Indica se o aluno é monitor de disciplina
IND-ALU-SAE	bit	Indica se o aluno recebe auxílio
IND-ALU-FORA-FLUXO	bit	Indica se o aluno está fora do fluxo
IND-ALU-TRANCADO	bit	Indica se o aluno está trancado no período
IND-ALU-ESPECIAL	bit	Indica se o aluno é especial

A Tabela 6.9 traz as principais colunas da Família de Colunas *CURSO*, que representam as características dos Cursos da Unversidade de Brasília. Além das características do próprio curso, também há informações sobre a Unidade Acadêmica do curso.

Tabela 6.9: Principais Colunas da Família de Colunas *CURSO*

Coluna	Tipo	Descrição
COD-CURSO	integer	Código do curso
NOME-CURSO	varchar(200)	Nome do curso
PERIODO-INICIO-CURSO	integer	Período de início do curso
FORMA-CURSO	varchar(10)	Curso presencial ou à distância
TIPO-CURSO	varchar(20)	Curso profissional ou acadêmico
TURNO-CURSO	varchar(20)	Curso diurno ou noturno
NIVEL-CURSO	varchar(20)	Nível acadêmico do curso
DEP-NOME-CURSO	varchar(200)	Nome do departamento do curso
DEP-SIG-CURSO	char(3)	Sigla do departamento do curso
DEP-CAMP-CURSO	varchar(30)	Campus do curso

A Tabela 6.10 traz as principais colunas da Família de Colunas *OPCAO*, que representam as características das Opções, ou Habilitações, dos curso da Unversidade de Brasília.

Tabela 6.10: Principais Colunas da Família de Colunas *OPCAO*

Coluna	Tipo	Descrição
COD-OPCAO	integer	Código da opção
NOME-OPCAO	varchar(200)	Nome da opção
PERIODO-INICIO-OPCAO	integer	Período de início da opção
GRAU-OPCAO	varchar(30)	Grau da opção
COD-ENADE-OPCAO	integer	Código ENADE da opção
DEP-NOME-OPCAO	varchar(200)	Nome do departamento da opção
DEP-SIG-OPCAO	char(3)	Sigla do departamento da opção
DEP-CAMP-OPCAO	varchar(30)	Campus da opção

A Tabela 6.11 traz as principais colunas da Família de Colunas *PERIODO*, que representam as características dos períodos da Unversidade de Brasília, já que praticamente todas as consultas requisitadas possuem um limite temporal.

Tabela 6.11: Principais Colunas da Família de Colunas *PERIODO*

Coluna	Tipo	Descrição
PER-PERIODO	integer	Período acadêmico
PER-NIVEL	varchar(20)	Nível acadêmico do período
PER-ANO	integer	Ano acadêmico

A Tabela 6.12 traz as principais colunas da Família de Colunas *DISCIPLINA*, que representam as características das disciplinas da Unversidade de Brasília.

A Tabela 6.13 traz as principais colunas da Família de Colunas *DOCENTE*, que representam as características pessoais e acadêmicas dos docentes da Universidade de Brasília.

Tabela 6.12: Principais Colunas da Família de Colunas *DISCIPLINA*

Coluna	Tipo	Descrição
COD-DISCIPLINA	integer	Código da disciplina
NOME-DISCIPLINA	varchar(200)	Nome da disciplina
PERIODO-INICIO-DISCIPLINA	integer	Período de início da disciplina
NIVEL-DISCIPLINA	integer	Nível acadêmico da disciplina
FORMA-DISCIPLINA	integer	Disciplina presencial ou à distância
DEP-NOME-DISCIPLINA	varchar(200)	Nome do departamento da disciplina
DEP-SIG-DISCIPLINA	char(3)	Sigla do departamento da disciplina
DEP-CAMP-DISCIPLINA	varchar(30)	Campus da disciplina

Tabela 6.13: Principais Colunas da Família de Colunas *DOCENTE*

Coluna	Tipo	Descrição
DOC-MATRICULA	integer	Matrícula do docente
DOC-CPF	char(11)	CPF do docente
DOC-NOME	varchar(200)	Nome do docente
DOC-SEXO	varchar(10)	Sexo do docente
DOC-NACIONALIDADE	varchar(20)	Nacionalidade do docente
DOC-ESCOLARIDADE	varchar(20)	Escolaridade do docente
DOC-PAIS	varchar(20)	País de nascimento do docente
IND-DOC-CEDIDO	bit	Indica se o docente está cedido a outro órgão
IND-DOC-FUNC-GESTAO	bit	Indica se o docente tem função de gestão
IND-DOC-ORIENTA	bit	Indica se o docente orienta algum aluno
IND-DOC-PESQ	bit	Indica se o docente atua em pesquisa
IND-DOC-AFAST-QUAL	bit	Indica se o docente está afastado para qualificação
IND-DOC-AFAST-SAUDE	bit	Indica se o docente está afastado por saúde
IND-DOC-EXTENS	bit	Indica se o docente atua em Extensão

6.2.2 Criação dos Índices

A criação dos índices para a tabela criada no Apache HBase levou em consideração a busca por filtros na ferramenta de relatórios Saiku/Pentaho e também o fato de que quase todas as consultas possuem filtros de períodos, ou seja, são limitadas por um intervalo de tempo. Dessa forma, foram criados índices a partir dos campos da tabela que representam os objetos mais acessados nas consultas, e também índices *cobertos* do Apache Phoenix com esses campos em conjunto com a informação de período acadêmico.

A criação dos índices é feita por comandos *MapReduce* do componente Apache Phoenix, e a partir de cada índice é criada uma tabela de mesmo nome no HBase, que é utilizada sempre que possível pelo plano de execução do Apache Phoenix nos momentos em que as colunas dos índices forem referenciadas nas consultas.

A tabela 6.14 traz os índices criados para otimização das consultas dos filtros na

ferramenta de geração de relatórios Saiku/Pentaho. No modelo relacional, ROLAP, independemente do tamanho da tabela de FATOS, essas consultas aos filtros são realizadas diretamente nas tabelas de Dimensões, que são muito menores em tamanho, o que não ocorre no modelo NoSQL OLAP, que possui apenas uma única grande tabela. Dessa forma, a criação dos índices para os objetos mais utilizados visa diminuir essa desvantagem da abordagem distribuída no momento da pesquisa pelos filtros na construção dos relatórios *ad hoc*.

Tabela 6.14: Índices simples criados no Apache Phoenix

Nome	**Coluna**	**Descrição**
ix-aluno-alumatricula	matricula	Matrícula do Aluno
ix-curso-codigo	cod-curso	Código do Curso
ix-curso-nome	nome-curso	Nome do Curso
ix-opcao-codigo	cod-opcao	Código da Opção
ix-opcao-nome	nome-opcao	Nome da Opção
ix-disciplina-nome	nome-disciplina	Nome da disciplina
ix-disciplina-codigo	cod-disciplina	Código da disciplina
ix-periodo	per-periodo	Período acadêmico
ix-docente-cpf	doc-cpf	CPF do docente
ix-docente-mat	doc-matricula	Matrícula do docente

A tabela 6.15 traz os índices *cobertos* criados para otimização das consultas em que são utilizaos intervalos temporais. Na Universidade de Brasília, a referência temporal é o período acadêmico. Dessa forma, assim como na tabela 6.14, foram criados índices com as mesmas colunas, mas com a inclusão da coluna de período em todos eles.

Tabela 6.15: Índices *cobertos* criados no Apache Phoenix

Nome	**Coluna**	**Descrição**
ix-aluno-alumatricula-per	matricula e per-periodo	Matrícula do Aluno
ix-curso-codigo-per	cod-curso e per-periodo	Código do Curso
ix-curso-nome-per	nome-curso e per-periodo	Nome do Curso
ix-opcao-codigo-per	cod-opcao e per-periodo	Código da Opção
ix-opcao-nome-per	nome-opcao e per-periodo	Nome da Opção
ix-disciplina-nome-per	nome-disciplina e per-periodo	Nome da disciplina
ix-disciplina-codigo-per	cod-disciplina e per-periodo	Código da disciplina
ix-docente-cpf-per	doc-cpf e per-periodo	CPF do docente
ix-docente-mat-per	doc-matricula e per-periodo	Matrícula do docente

6.3 Implementação das ferramentas *Mondrian, Pentaho e Saiku*

6.3.1 Criação das Métricas e Dimensões na ferramenta Mondrian

A ferramenta Mondrian é a responsável pelo mapeamento entre as colunas da tabela criada *FT-DWSED-FATO* no Apache HBase e os objetos OLAP, métricas e dimensões, disponibilizados aos usários pela ferramenta de geração de relatórios Saiku. Para que a conexão entre Mondrian e a tabela criada seja efetuada, é preciso que o servidor de consultas do Apache Phoenix, *Query Server*, esteja ativo, e que o arquivo ***phoenix-4.11.0-HBase-1.3-thin-client.jar***, presente na pasta */phoenix*, seja copiado para a pasta */mondrian/lib*.

No Mondrian, é necessário configurar a conexão com o Apache Phoenix antes do início do mapeamento, conforme Figura 6.4.

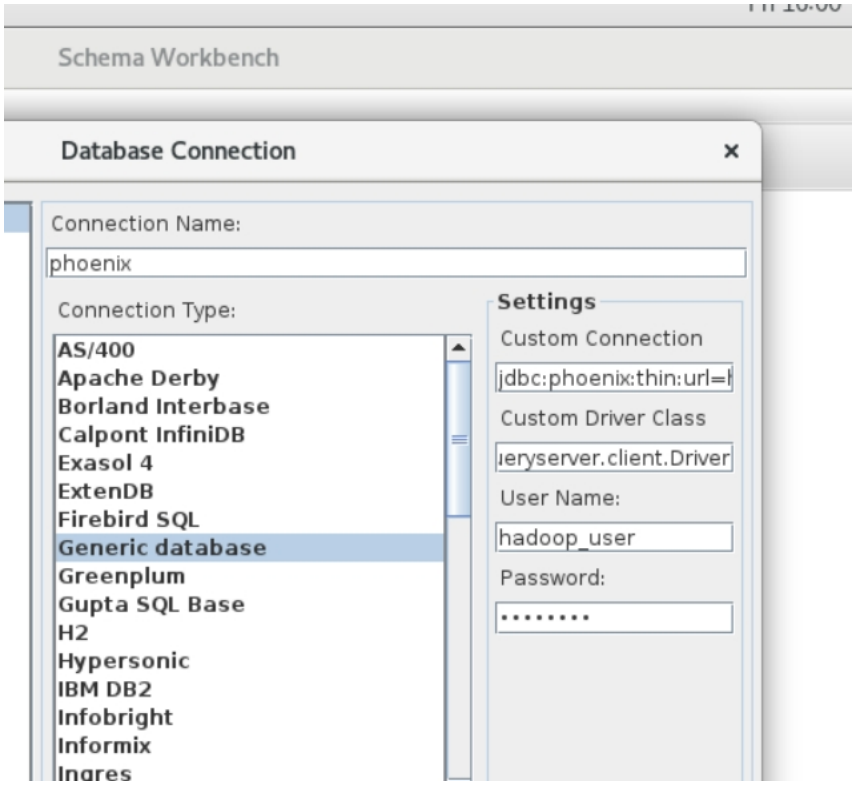


Figura 6.4: Configuração da conexão no Mondrian

Após a configuração da conexão, é feito o mapeamento entre as colunas da tabela e os objetos OLAP, conforme Figura 6.5. Cria-se então o Modelo, ou seja, um arquivo *.xml*

com o mapeamento dos objetos e conexão criada para o para o servidor *Query Server* do Apache Phoenix. Após esse passo, é feito o *upload* desse modelo para o servidor Pentaho.

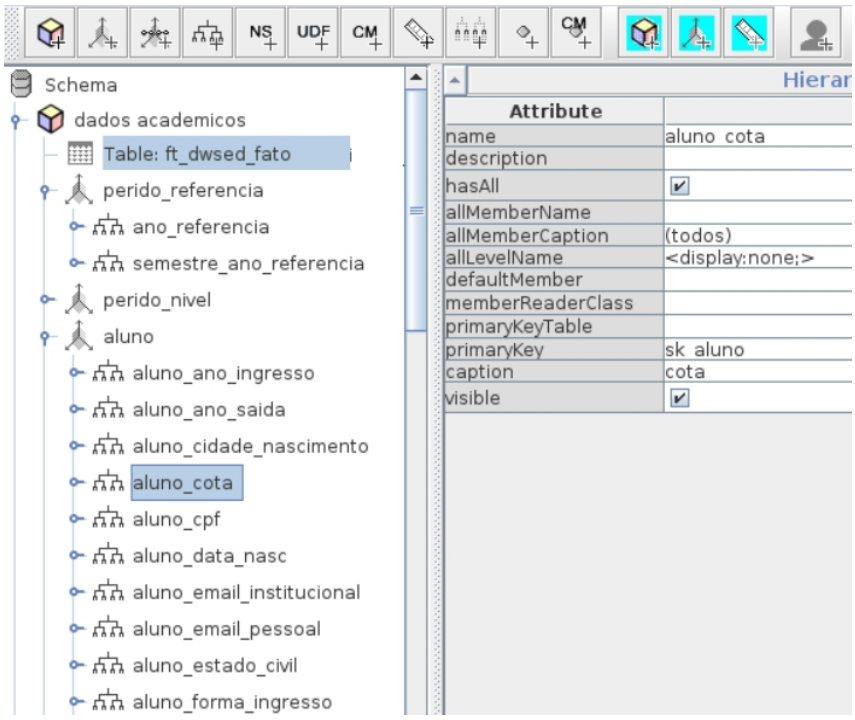


Figura 6.5: Configuração dos objetos OLAP no Mondrian

6.3.2 Conexão ao servidor *OLAP Pentaho* e *API* de relatórios Saiku

O servidor OLAP Pentaho fornece diversos serviços aos usuários para que estes sejam capazes de realizar consultas analíticas e com isso tomarem decisões por meio das informações geradas em relatórios ou paineis, chamados *Dashboards*, que podem ser armazenados em pastas personalizadas para cada usuário. Além disso, essa solução também disponibiliza meios para que sejam armazenados arquivos, imagens ou outros documentos de interesse. O Pentaho fornece nativamente uma ferramenta de relatórios *ad hoc*, porém, a comunidade de *software livre* desenvolve extensões que podem ser adicionadas ao servidor com interface mais amigável ao usuário, além de algumas capacidades extras na criação de relatórios. Na Universidade de Brasília, foi adotada a extensão Saiku para a geraçao de relatórios OLAP, com boa aceitação por parte dos usuários.

Assim como na ferramenta Mondrian, no servidor Pentaho são necessárias algumas configurações para que haja a conexão ao *Query Server* do Apache Phoenix. O mesmo arquivo ***phoenix-4.11.0-HBase-1.3-thin-client.jar*** deve ser copiado para a pasta

/pentaho/tomcat/lib. Além disso, uma fonte de dados deve ser criada a partir do Modelo enviado pela ferramenta Mondrian, conforme Figura 6.6.

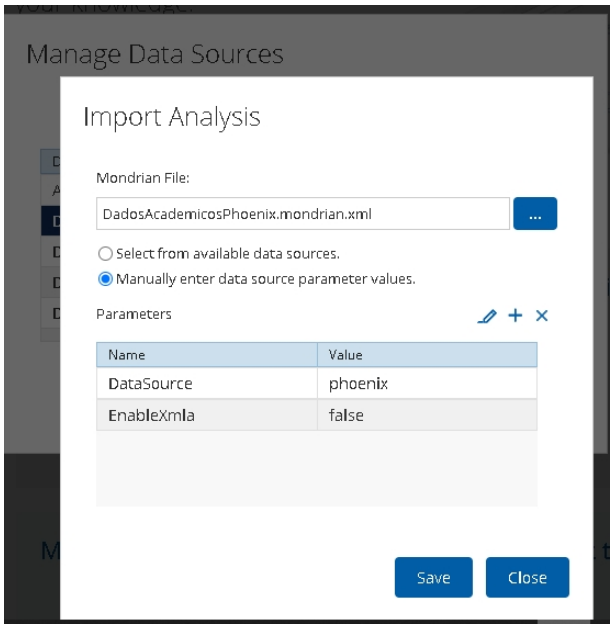


Figura 6.6: Configuração da fonte de dados no Pentaho

Após a configuração da fonte de dados, na ferramenta de geração de relatórios Saiku esta já ficará disponível para escolha do usuário. Conforme a Figura 6.7, é possível perceber que as duas fontes de dados estão disponíveis, a que faz o mapeamento para a abordagem relacional OLAP, no banco de dados Postgresql, e a abordagem NoSQL OLAP, que faz o mapeamento para o banco de dados distribuído HBase.

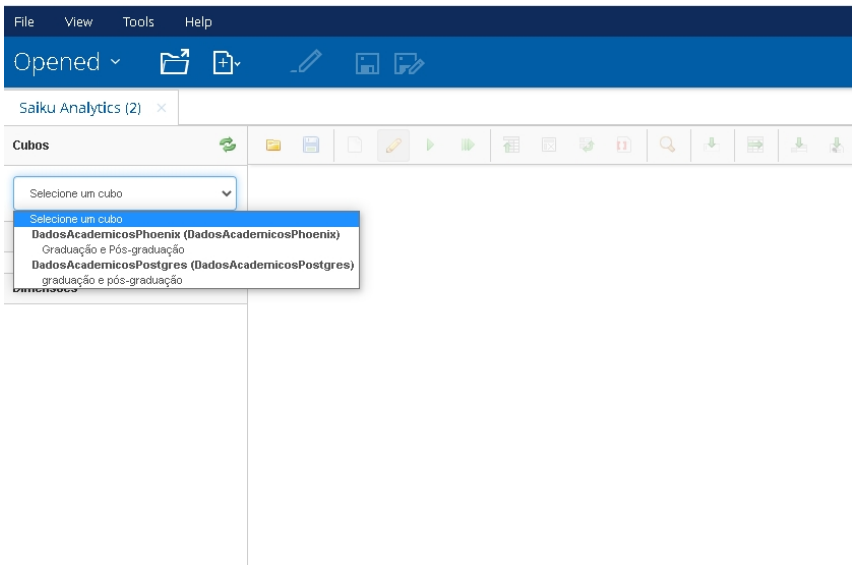


Figura 6.7: Escolha do Cubo na ferramenta Saiku

A partir da escolha pela fonte de dados relacionada à abordagem distribuída Hadoop/HBase/Phoenix, o usuário poderá iniciar a criação de relatórios *ad hoc*, escolhendo os objetos na ferramenta Saiku, de acordo como foram mapeados na ferramenta Mondrian, conforme Figura 6.8.

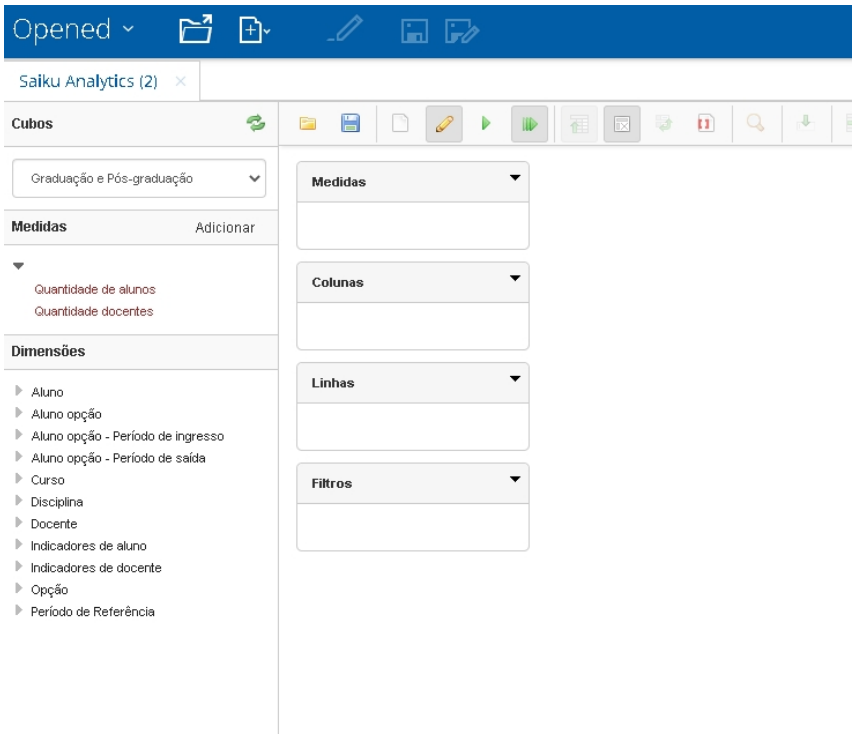


Figura 6.8: Criação de relatórios na ferramenta Saiku

Capítulo 7

Resultados

Este capítulo apresenta os resultados obtidos pela aferição dos tempos em segundos de consultas utilizadas como comparação entre as duas arquiteturas: a tradicional ROLAP, e a nova NoSQL OLAP, ou simplesmente NoLAP. Foram selecionadas consultas simples e complexas, de forma a simularem necessidades de análises de demandas do cotidiano da equipe CED da STI da UnB. As consultas foram executadas em alguns blocos de milhões de registros, com o objetivo de se observar a capacidade de resposta de ambas arquiteturas e comprovar a hipótese de pesquisa deste trabalho, ou seja, com o aumento da quantidade de registros, a abordagem distribuída apresentará um desempenho proporcionalmente melhor em relação à abordagem relacional.

A configuração do servidor da abordagem ROLAP é a seguinte:

- Um único servidor
- 48 GB de RAM
- 8 núcleos ou 8 *cores*
- Sistema Operacional **Windows Server 2012**
- Bancos de Dados **PostgreSQL 10**
- Índices criados para todas as chaves estrangeiras da tabela de Fatos relacionadas às tabelas de Dimensões e de Indicadores, conforme Figura 5.1, do **Capítulo 5**.

7.0.1 Consultas Utilizadas

Foram utilizadas as seguintes consultas:

- ***Consulta 1:*** **select count(*) from...** => Essa consulta foi utilizada como parâmetro de desempenho entre as duas abordagens, pois trata-se de uma consulta que

"varre" toda a tabela, e reflete a queda de desempenho com o aumento do número de registros.

- ***Consulta 2:*** **select distinct nome-curso from..** => Essa consulta foi utilizada como uma simulação de escolha de um curso no filtro de cursos da ferramenta Saiku. Em tese, esse tipo de consulta beneficia a abordagem relacional normalizada, pois é feita diretamente nas tabelas de Dimensões, que não sofrem o aumento observado na tabela de Fatos. No entanto, conforme visto no **Capítulo 6**, foram criados índices no Apache Phoenix para mitigar essa desvantagem da abordagem distribuída.
- ***Consulta 3:*** **select distinct nome-curso from ... where per-periodo = 20162** => Essa consulta, muito parecida com a consulta 2, mas com filtro de período, foi utilizada para forçar a utilização da tabela de Fatos na abordagem relacional normalizada, de modo a obter uma comparação em uma consulta **aleatória** à base de dados, considerado um ponto forte do SGBD Apache HBase.
- ***Consulta 4*****: select count(distinct matricula), count(distinct case when alu-cota <> 'Universal' then matricula else null end), nome-curso from... group by nome-curso** => Essa consulta representa uma requisição frequentemente feita à equipe CED da STI, que é a informação relacionada aos alunos cotistas na Universidade de Brasília. Nesse caso, escolheu-se uma consulta de quantidade de alunos totais e quantidade de alunos cotistas históricas dos cursos da UnB, ou seja, percorrendo a base inteira de dados. Apesar de a política de cotas ter iniciado recentemente, essa consulta tem por objetivo medir o tempo de uma consulta que envolve agregação de dados em todas as linhas da tabela.
- ***Consulta 5*****: select count(distinct matricula), count(distinct case when alu-cota <> 'Universal' then matricula else null end), nome-curso from... where per-periodo = 20171 group by nome-curso** => Essa consulta é muito parecida com a consulta anterior, porém, foi adicionado o filtro de período, assim como na consulta de número 3, para além de verificarmos o desempenho de uma consulta com agregação de dados, também verificarmos o comportamento das duas abordagens em relação à resposta de consulta com filtros aleatórios.

7.0.2 Resultados obtidos

As cinco consultas foram executadas em quatro cenários distintos para as quantidades de registros na tabela de *Fatos* da abordagem ROLAP e na única tabela da abordagem NoLAP: **7 milhões**, **23 milhões**, **46 milhões** e **58 milhões de registros**.

Os desempenhos das duas abordagens em relação às cinco consultas nos quatro cenários estão representados pelas Tabelas 7.1, 7.2, 7.3 e 7.4. Já a Tabela 7.5 traz as diferenças de tempo, em segundos, observadas entre essas cinco consultas para essas duas abordagens, ou seja, os tempos das consultas da arquitetura distribuída subtraídos dos tempos das consultas da arquitetura relacional.

Para uma melhor visualização, a Figura 7.1 traz as diferenças entre as consultas da Tabela 7.5 em forma de um gráfico. Pelo gráfico, é possível observar o ganho de desempenho que a abordagem NoLAP apresenta em relação à abordagem ROLAP à medida que a quantidade de registros aumenta, já que a diferença de tempo entre as consultas equivalentes nas duas arquiteturas aumenta substancialmente com o aumento do volume da carga de dados.

Tabela 7.1: Tempo em segundos (s) das abordagens para **7 milhões** de registros

Consulta	**ROLAP - Tempo (S)**	**NoLAP - Tempo (s)**
1	2s	6s
2	0.056s	0.476s
3	0.714s	3.51
4	12s	24s
5	3s	8.8s

Tabela 7.2: Tempo em segundos (s) das abordagens para **23 milhões** de registros

Consulta	**ROLAP - Tempo (S)**	**NoLAP - Tempo (s)**
1	4.7s	7s
2	0.055s	0.541s
3	19s	21s
4	75s	68s
5	28s	24s

Tabela 7.3: Tempo em segundos (s) das abordagens para **46 milhões** de registros

Consulta	**ROLAP - Tempo (S)**	**NoLAP - Tempo (s)**
1	32s	21s
2	0.058s	0.741s
3	27s	24s
4	217s	82s
5	42s	35s

Tabela 7.4: Tempo em segundos (s) das abordagens para **58 milhões** de registros

Consulta	**ROLAP - Tempo (S)**	**NoLAP - Tempo (s)**
1	138.0s	34.4s
2	0.056s	0.831s
3	97.00s	51.07s
4	322.00s	119.43s
5	92.1s	56.7s

Tabela 7.5: Diferenças em segundos (s) entre os tempos (Δt)

Consulta	**7 milhões**	**23 milhões**	**46 milhões**	**58 milhões**
1	-4	-2.3	11	104
2	-0.42	-0.486	-0.683	-0.775
3	-2.796	-2	3	46
4	-12	7	135	202.57
5	-5.8	4	7	35.4

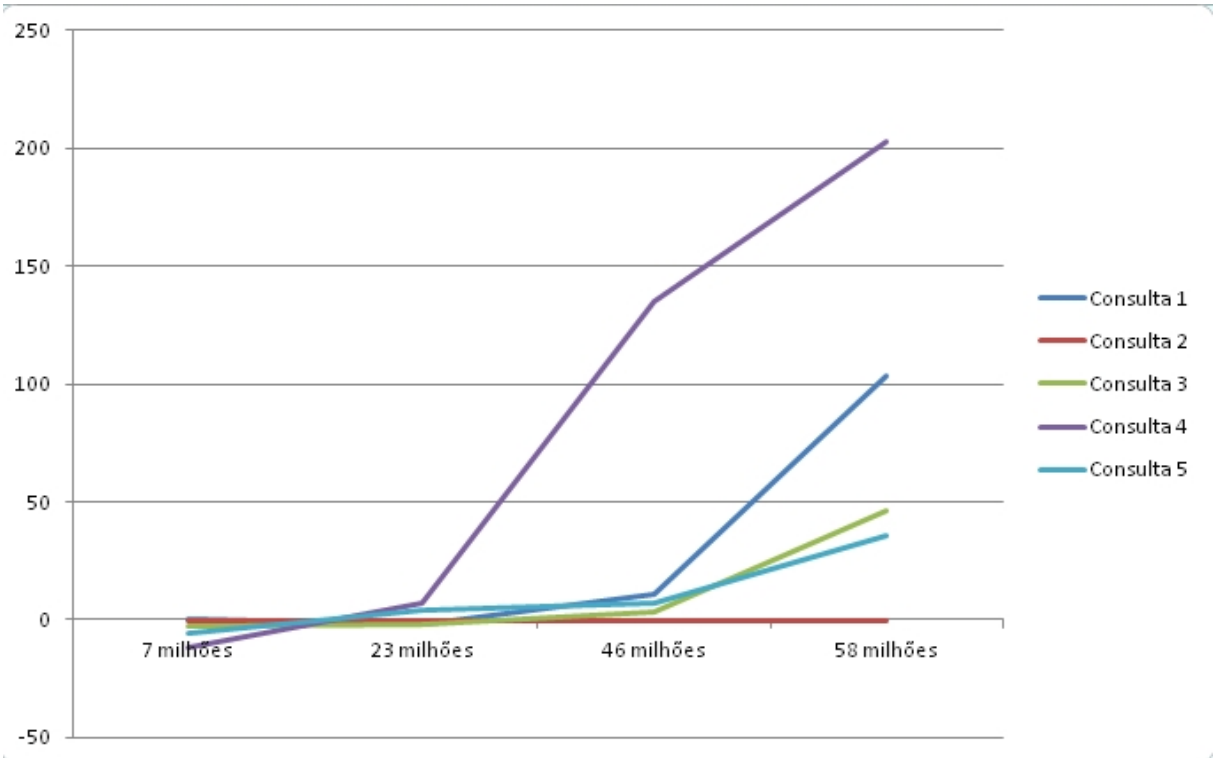


Figura 7.1: Gráfico das diferenças entre os tempos das consultas (Δt)

Capítulo 8

Conclusão

Historicamente, não havia uma ferramenta disponível na Universidade de Brasília para tornar seus gestores capazes de realizar uma análise integrada das várias áreas que compõem a Universidade de maneira otimizada. A partir dos diversos pedidos de extrações de dados direcionados à Secretaria de Tecnologia da Informação da UnB - STI, tanto embasados na LAI e direcionados a obterem dados para pesquisas, quanto originados dos diversos departamentos para o atendimento de demandas, como o Censo Escolar do Ministério da Educação, por exemplo, iniciou-se , em 2018, o desenvolvimento de um Sistema de Apoio à Decisão - SAD - denominado BI-UnB em arquitetura ROLAP, ou seja, com a utilização de bancos de dados relacionais. No entanto, nos últimos anos, foi notório o aumento do volume do armazenamento das bases de dados acadêmicos da Universidade, que foi refletido no crescimento constante do tamanho das tabelas deste SAD desenvolvido. Dessa forma, surgiu a questão de como proporcionar à comunidade uma análise integrada de dados acadêmicos de forma satisfatória quando a solução em SGBD Relacional começar a perder o bom desempenho em função do crescimento do volume de dados gerados.

A partir de uma metodologia constituída pela revisão sistemática da literatura e por um estudo de caso realizado na Secretaria de Tecnologia da Informação da Universidade de Brasília - STI/UnB, uma nova arquitetura em bancos de dados NoSQL foi proposta e testada para confirmar a hipótese de que uma solução distribuída de banco de dados com uma abordagem baseada em processamento de Big Data poderia melhorar o desempenho da entrega de informações gerenciais à comunidade acadêmica e garantir uma maior disponibilidade em relação ao crescimento dos dados ao longo dos próximos anos.

Pelos resultados do Capítulo 7, foi possível observar, a partir da implementação de um *cluster* Hadoop, o ganho de desempenho nas consultas à medida do aumento do número de registros, em função do processamento paralelo do banco de dados NoSQL HBase. Observou-se também uma grande diferença em desempenho principalmente nas consultas em que não foram utilizados filtros, ou seja, naquelas em que a análise esteve relacionada à

totalidade de registros da tabela. A vantagem da solução tradicional ROLAP foi observada e confirmada na consulta pontual a um objeto OLAP para a criação de filtros, já que o tempo observado nos quatro cenários foi sempre inferior para essa arquitetura. No entanto, com a utilização de filtros criados para o SGBD HBase, mesmo no pior caso, para 58 milhões de registros, a diferença entre as consultas não passou de 1 segundo, um tempo muito razoável e tolerável pelos usários e que notou-se imperceptível no gráfico da Figura 7.1 do Capítulo 7.

Além do fator desempenho, a solução NoLAP deste trabalho também oferece uma maior disponibilidade do serviço, uma vez que trabalha com a replicação dos dados, que podem ser armazenados em servidores físicos distintos.

Outra premissa citada no Capítulo 1 deste trabalho também foi confirmada, ou seja, a partir da utilização do componente Apache Phoenix, foi possível a utilização do SGBD HBase com a manutenção das ferramentas OLAP já utilizadas pela Universidade de Brasília: Mondrian, Pentaho e Saiku. Como a cultura da utilização de uma ferramenta de análise integrada ainda é relativamente recente na Universidade de Brasília, o requisito pela manutenção da solução Pentaho era bastante desejável pela boa aceitação por parte dos usuários, que passaram por alguns meses de adaptação e atualmente alcançaram independência na construção de relatórios para suas análises.

Além dessas características citadas, conseguiu-se provar que uma solução distribuída pode ser superior a uma solução tradicional ROLAP com uma configuração muito semelhante. Por exemplo, a memória RAM total do *cluster* NoLAP, que trata-se do componente mais caro, foi exatamente a mesma do servidor utilizado para a abordagem ROLAP.

Sendo assim, as principais contribuições deste trabalho são: (i) o levantamento dos principais referenciais teóricos sobre o tema envolvendo Sistemas de Apoio à Decisão com bancos de dados NoSQL, (ii) realização de um estudo de caso aplicando uma arquitetura referenciada por outros estudos científicos, (iii) implementação de um cluster Hadoop com HBase conectado a um servidor Pentaho para a construção de análises gerencias, (iv) comparação de desempenho entre consultas da arquitetura com SGBD Relacional e SGBD NoSQL em quatro cenários de crescimento do volume de dados, simulando possíveis situações futuras.

Ao final deste trabalho, chega-se à conclusão de que os resultados esperados foram alcançados. A hipótese de pesquisa de que uma solução distribuída de banco de dados com uma abordagem baseada em processamento de Big Data poderia melhorar o desempenho da entrega de informações gerenciais à comunidade acadêmica foi confirmada pela simulação apresentada. Além da nova arquitetura, este trabalho também contribuiu para um melhor entendimento do tema dentro da Secretaria de Tecnologia da Informação, uma vez que despertou o interesse de demais servidores que participaram de alguma forma da

pesquisa e abre novas possibilidades de trabalhos futuros relacionados à incorporação de novas bases ao novo SAD NoLAP, além das já incluídas bases acadêmicas, para se criar um extenso repositório unificado de dados para a Universidade de Brasília.

Referências

[1] Kimball, Ralph e Margy Ross: *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons, 2013. 1, 2, 5, 7, 8, 9, 11, 33, 37, 43, 46

[2] Dehdouh, Khaled, Fadila Bentayeb, Omar Boussaid e Nadia Kabachi: *Using the column oriented nosql model for implementing big data warehouses*. Em *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, página 469. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2015. 2, 4, 5, 25, 26, 28, 33, 35, 38, 55

[3] Heinen, Juliano: *Comentários à lei de acesso à informação*. Belo Horizonte, 2011. 2

[4] Inmon, William H: *Building the data warehouse*. 1992. 2

[5] Moniruzzaman, ABM e Syed Akhter Hossain: *Nosql database: New era of databases for big data analytics-classification, characteristics and comparison*. arXiv preprint arXiv:1307.0191, 2013. 3, 12

[6] Chevalier, Max, Mohammed El Malki, Arlind Kopliku, Olivier Teste e Ronan Tournier: *Implementation of multidimensional databases in column-oriented nosql systems*. Em *East European conference on advances in databases and information systems*, páginas 79–91. Springer, 2015. 5, 13, 26, 27, 28, 29, 33, 35, 38, 45, 55

[7] Yangui, Rania, Ahlem Nabli e Faiez Gargouri: *Automatic transformation of data warehouse schema to nosql data base: comparative study*. Procedia Computer Science, 96:255–264, 2016. 5, 28, 29, 35, 38

[8] MARIANO, Ari Melo e Maíra Santos ROCHA: *Revisão da literatura: Apresentação de uma abordagem integradora*. Em *XXVI Congreso Internacional de la Academia Europea de Dirección y Economía de la Empresa (AEDEM), Reggio Calabria*, volume 26, 2017. 7, 24, 31, 32

[9] Fourny, Ghislain: *Cell stores*. arXiv preprint arXiv:1410.0600, 2014. 8

[10] Carvalho Victorino, Marcio de, Marcelo Shiessl, Edgard Costa Oliveira, Edson Ishikawa, Maristela Terto de Holanda e Marçal de Lima Hokama: *Uma proposta de ecossistema de big data para a análise de dados abertos governamentais concetados*. Informação Sociedade, 27(1), 2017. 8

[11] Inmon, William H: *Building the data warehouse*. John wiley & sons, 2005. 8, 43

[12] Sadalage, Pramod J e Martin Fowler: *NoSQL essencial: um guia conciso para o mundo emergente da persistência poliglota*. Novatec Editora, 2013. 11, 13, 15, 19

[13] Alekseev, AA, VV Osipova, MA Ivanov, A Klimentov, NV Grigorieva e HS Nalamwar: *Efficient data management tools for the heterogeneous big data warehouse*. Physics of Particles and Nuclei Letters, 13(5):689–692, 2016. 13, 28, 29, 30, 38

[14] Dehdouh, Khaled: *Building olap cubes from columnar nosql data warehouses*. Em *International Conference on Model and Data Engineering*, páginas 166–179. Springer, 2016. 13, 28, 29, 35

[15] Hewitt, Eben: *Cassandra: the definitive guide*. " O'Reilly Media, Inc.", 2010. 14

[16] Loukides, Mike e Julie Steele: *Hbase: The definitive guide*, 2009. 15, 16

[17] SHARMA, TARUNA, MANJU PAYAL, KIRANDEEP KAUR, POOJA DIXIT e PRATEEK DADHICH: *An introduction to data warehousing and olap: Pros & cons*. Eureka, 2581:5172, 2018. 16, 28, 29, 38

[18] Chen, CL Philip e Chun Yang Zhang: *Data-intensive applications, challenges, techniques and technologies: A survey on big data*. Information sciences, 275:314–347, 2014. 17

[19] Laney, Doug: *3d data managment: controlling data volume, velocity and variety, appl*. Delivery Strategies Meta Group (949), 2001. 17

[20] Zikopoulos, Paul, Chris Eaton *et al.*: *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011. 17

[21] White, Tom: *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012. 19, 21

[22] Chao, Lu, Chundian Li, Fan Liang, Xiaoyi Lu e Zhiwei Xu: *Accelerating apache hive with mpi for data warehouse systems*. Em *2015 IEEE 35th International Conference on Distributed Computing Systems*, páginas 664–673. IEEE, 2015. 28, 38

[23] He, Yongqiang, Rubao Lee, Yin Huai, Zheng Shao, Namit Jain, Xiaodong Zhang e Zhiwei Xu: *Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems*. Em *2011 IEEE 27th International Conference on Data Engineering*, páginas 1199–1208. IEEE, 2011. 28, 38

[24] Camacho-Rodríguez, Jesús, Ashutosh Chauhan, Alan Gates, Eugene Koifman, Owen O'Malley, Vineet Garg, Zoltan Haindrich, Sergey Shelukhin, Prasanth Jayachandran, Siddharth Seth *et al.*: *Apache hive: From mapreduce to enterprise-grade big data warehousing*. arXiv preprint arXiv:1903.10970, 2019. 28, 38

[25] Martinho, Bruno e Maribel Yasmina Santos: *An architecture for data warehousing in big data environments*. Em *International conference on research and practical issues of enterprise information systems*, páginas 237–250. Springer, 2016. 28, 38

[26] Xi, Li, Wang Hongkai, Li Jinhu Pei, Yu Zhanpeng *et al.*: *Research on multi-dimensional analysis method of power equipment condition monitoring based on olap.* Em *2019 IEEE 4th International Conference on Big Data Analytics (ICBDA)*, páginas 384–388. IEEE, 2019. 28, 38

[27] Garg, Varun: *Optimization of multiple queries for big data with apache hadoop/hive.* Em *2015 International Conference on Computational Intelligence and Communication Networks (CICN)*, páginas 938–941. IEEE, 2015. 28, 38

[28] Kang, Woo Lam, Hyeon Gyu Kim e Yoon Joon Lee: *Reducing i/o cost in olap query processing with mapreduce.* IEICE TRANSACTIONS on Information and Systems, 98(2):444–447, 2015. 28

[29] Gadiraju, Krishna Karthik, Manik Verma, Karen C Davis e Paul G Talaga: *Benchmarking performance for migrating a relational application to a parallel implementation.* Future Generation Computer Systems, 63:148–156, 2016. 28, 38

[30] Kumar, A Sunny: *Performance analysis of mysql partition, hive partition-bucketing and apache pig.* Em *2016 1st India International Conference on Information Processing (IICIP)*, páginas 1–6. IEEE, 2016. 28

[31] Czajkowski, Bartosz e Teresa Zawadzka: *Duabi-business intelligence architecture for dual perspective analytics.* Em *International Conference: Beyond Databases, Architectures and Structures*, páginas 527–538. Springer, 2017. 28

[32] Costa, Carlos e Maribel Yasmina Santos: *Evaluating several design patterns and trends in big data warehousing systems.* Em *International Conference on Advanced Information Systems Engineering*, páginas 459–473. Springer, 2018. 28, 38

[33] Gad, Ibrahim, BR Manjunatha *et al.*: *Hybrid data warehouse model for climate big data analysis.* Em *2017 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, páginas 1–9. IEEE, 2017. 28

[34] Song, Jie, Chaopeng Guo, Zhi Wang, Yichan Zhang, Ge Yu e Jean Marc Pierson: *Haolap: a hadoop based olap system for big data.* Journal of Systems and Software, 102:167–181, 2015. 28, 30

[35] Scabora, Lucas C, Jaqueline J Brito, Ricardo Rodrigues Ciferri e Cristina Dutra de Aguiar Ciferri: *Physical data warehouse design on nosql databases.* Em *Proceedings of the 18th International Conference on Enterprise Information Systems*, páginas 111–118. SCITEPRESS-Science and Technology Publications, Lda, 2016. 28

[36] Boussahoua, Mohamed, Omar Boussaid e Fadila Bentayeb: *Logical schema for data warehouse on column-oriented nosql databases.* Em *International Conference on Database and Expert Systems Applications*, páginas 247–256. Springer, 2017. 28

[37] Franciscus, Nigel, Xuguang Ren e Bela Stantic: *Answering temporal analytic queries over big data based on precomputing architecture.* Em *Asian Conference on Intelligent Information and Database Systems*, páginas 281–290. Springer, 2017. 28

[38] Chevalier, Max, Mohammed El Malki, Arlind Kopliku, Olivier Teste e Ronan Tournier: *How can we implement a multidimensional data warehouse using nosql?* Em *International Conference on Enterprise Information Systems*, páginas 108–130. Springer, 2015. 28, 29, 35, 38

[39] Chevalier, Max, Mohammed El Malki, Arlind Kopliku, Olivier Teste e Ronan Tournier: *Benchmark for olap on nosql technologies*. 2015. 33, 35

[40] Dehdouh, Khaled, Fadila Bentayeb, Omar Boussaid e Nadia Kabachi: *Columnar nosql cube: Agregation operator for columnar nosql data warehouse*. Em *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, páginas 3828–3833. IEEE, 2014. 33, 35, 45

[41] Chaudhuri, Surajit e Umeshwar Dayal: *An overview of data warehousing and olap technology*. ACM Sigmod record, 26(1):65–74, 1997. 33

[42] Golfarelli, Matteo, Dario Maio e Stefano Rizzi: *The dimensional fact model: A conceptual model for data warehouses*. International Journal of Cooperative Information Systems, 7(02n03):215–247, 1998. 33

[43] Cuzzocrea, Alfredo, Ladjel Bellatreche, Il Yeol Song *et al.*: *Data warehousing and olap over big data: current challenges and future research directions.* Em *DOLAP*, volume 13, páginas 67–70, 2013. 33

[44] Dede, Elif, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon e Lavanya Ramakrishnan: *Performance evaluation of a mongodb and hadoop platform for scientific data analysis*. Em *Proceedings of the 4th ACM workshop on Scientific cloud computing*, páginas 13–20. ACM, 2013. 33

[45] Cattell, Rick: *Scalable sql and nosql data stores*. Acm Sigmod Record, 39(4):12–27, 2011. 33

[46] Leavitt, Neal: *Will nosql databases live up to their promise?* Computer, 43(2):12–14, 2010. 33

[47] Bicevska, Zane e Ivo Oditis: *Towards nosql-based data warehouse solutions*. Procedia Computer Science, 104:104–111, 2017. 35

[48] Psiuk-Maksymowicz, Krzysztof, Aleksander Płaczek, Roman Jaksik, Sebastian Student, Damian Borys, Dariusz Mrozek, Krzysztof Fujarewicz e Andrzej Świerniak: *A holistic approach to testing biomedical hypotheses and analysis of biomedical data*. Em *Beyond Databases, Architectures and Structures. Advanced Technologies for Data Mining and Knowledge Discovery*, páginas 449–462. Springer, 2015. 35

[49] Santos, Maribel Yasmina, Bruno Martinho e Carlos Costa: *Modelling and implementing big data warehouses for decision support*. Journal of Management Analytics, 4(2):111–129, 2017. 35

[50] Liu, Yunkai e Theresa M Vitolo: *Graph data warehouse: Steps to integrating graph databases into the traditional conceptual structure of a data warehouse*. Em *2013 IEEE International Congress on Big Data*, páginas 433–434. IEEE, 2013. 35

[51] Ma, Kun e Runyuan Sun: *Introducing websocket-based real-time monitoring system for remote intelligent buildings*. International Journal of Distributed Sensor Networks, 9(12):867693, 2013. 35

[52] Ma, Kun e Bo Yang: *Introducing extreme data storage middleware of schema-free document stores using mapreduce*. International Journal of Ad Hoc and Ubiquitous Computing, 20(4):274–284, 2015. 35

[53] Ma, Kun e Bo Yang: *Column access-aware in-stream data cache with stream processing framework*. Journal of Signal Processing Systems, 86(2-3):191–205, 2017. 35

[54] Wang, Shicai, Ioannis Pandis, Chao Wu, Sijin He, David Johnson, Ibrahim Emam, Florian Guitton e Yike Guo: *High dimensional biological data retrieval optimization with nosql technology*. Em *BMC genomics*, volume 15, página S3. BioMed Central, 2014. 35

[55] Hu, Yang, Venkat Yashwanth Gunapati, Pei Zhao, Devin Gordon, Nicholas R Wheeler, Mohammad A Hossain, Timothy J Peshek, Laura S Bruckman, Guo Qiang Zhang e Roger H French: *A nonrelational data warehouse for the analysis of field and laboratory data from multiple heterogeneous photovoltaic test sites*. IEEE Journal of Photovoltaics, 7(1):230–236, 2016. 35

[56] Zhao, Hongwei e Xiaojun Ye: *A practice of tpc-ds multidimensional implementation on nosql database systems*. Em *Technology Conference on Performance Evaluation and Benchmarking*, páginas 93–108. Springer, 2013. 35