UNIVERSIDADE FEDERAL DA BAHIA

DISSERTAÇÃO DE MESTRADO

CORRELAÇÃO PROBABILÍSTICA IMPLEMENTADA EM SPARK PARA BIG DATA EM SAÚDE

ROBESPIERRE DANTAS DA ROCHA PITA

Mestrado Multiinstitucional em Ciência da Computação

Salvador
05 de março de 2015

MMCC-Msc-2015

ROBESPIERRE DANTAS DA ROCHA PITA

CORRELAÇÃO PROBABILÍSTICA IMPLEMENTADA EM SPARK PARA BIG DATA EM SAÚDE

Esta Dissertação de Mestrado foi apresentada ao Programa de Mestrado Multiinstitucional em Ciência da Computação da Universidade Federal da Bahia, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: MARCOS ENNES BARRETO

Salvador
05 de março de 2015

Ficha catalográfica.

PITA, ROBESPIERRE D. R

CORRELAÇÃO PROBABILÍSTICA IMPLEMENTADA EM SPARK PARA BIG DATA EM SAÚDE/ ROBESPIERRE DANTAS DA ROCHA PITA– Salvador, 05 de março de 2015.

73p.: il.

Orientador: MARCOS ENNES BARRETO.
DISSERTAÇÃO DE MESTRADO– UNIVERSIDADE FEDERAL DA BAHIA, INSTITUTO DE MATEMÁTICA, 05 de março de 2015.

TOPICOS PARA FICHA CATALOGRAFICA.
I. BARRETO, MARCOS E.. II. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA. III Título.

NUMERO CDD

TERMO DE APROVAÇÃO

ROBESPIERRE DANTAS DA ROCHA PITA

CORRELAÇÃO PROBABILÍSTICA IMPLEMENTADA EM SPARK PARA BIG DATA EM SAÚDE

Esta Dissertação de Mestrado foi julgada adequada à obtenção do título de Mestre em Ciência da Computação e aprovada em sua forma final pelo Programa de Mestrado Multiinstitucional em Ciência da Computação da Universidade Federal da Bahia.

Salvador, 05 de Março de 2015

Prof. Dr. Ricardo Araújo Rios
DCC/UFBA

Prof. Dr. Murilo do Carmo Boratto
UNEB

Prof. Dr. Carlos Antonio de Souza Teles Santos
UEFS

Para aquela que desde sempre participou de todas a minhas lutas, dedico a primeira vitória que não pude comemorar ao seu lado. Esteja com Deus vó, eu sempre lhe amarei.

AGRADECIMENTOS

O que é a gratidão, senão a maior das virtudes do homem? Reconhecer e agradecer a uma mão estendida é um ato que perpetua as boas ações ao redor do mundo. Por isso mesmo, não poderei deixar de agradecer aqueles que estiveram toda a minha vida ao meu lado, torcendo por mim, vibrando cada vitória e compartilhando cada lágrima. Minha amável família, mãe, pai, irmã, irmão, primos, tios(as), etc. Obrigado, de coração. Todo suporte complementar de amor, afeto e companheirismo veio do único lugar que eu poderia esperar, minha doce namorada, muito obrigado.

Sendo esta vitória uma consagração de anos de dedicação à carreira acadêmica, não posso deixar de exercer a gratitude com aqueles que possibilitaram a existência desta seara do conhecimento: Leobino Sampaio, Carolina Souza e Luis Michelon. Sem vocês dificilmente eu teria dado tantos passos, e sei que consideram tão curta a caminhada feita até agora.

Outro agradecimento especial deve ser dado ao meu professor Marcos Ennes Barreto. Detentor de uma capacidade criativa, atenção e paciência invejáveis, teve papel mais que importante para o desenvolvimento deste trabalho. Usarei os ensinamentos provenientes da sua atuação como orientador o resto da minha carreira. Sem esquecer das indiscutíveis contribuições, motivações, cobranças e ideais do professor Davide Rasella, peça importantíssima deste projeto e trabalho e da solicitude de Antony Peter Stevens, colaborador que direcionou os primeiros caminhos a serem seguidos para obter os conhecimentos necessários.

Por último, mas não menos importante, agradeço Àquele que colocou em meu caminho todas essas pessoas maravilhosas e que sempre esteve junto a mim em todos os momentos. Deus, muito obrigado.

Tás a ver a linha do horizonte?
A levitar, a evitar que o céu se desmonte
Foi seguindo essa linha que notei que o mar
Na verdade é uma ponte
Atravessei e fui a outros litorais
E no começo eu reparei nas diferenças
Mas com o tempo eu percebi
E cada vez percebo mais
Como as vidas são iguais
Muito mais do que se pensa

Mudam as caras
Mas todas podem ter as mesmas expressões
Mudam as línguas mas todas têm
Suas palavras carinhosas e os seus calões
As orações e os deuses também variam
Mas o alívio que eles trazem vem do mesmo lugar
Mudam os olhos e tudo que eles olham
Mas quando molham todos olham com o mesmo olhar.

Seja onde for uma lágrima de dor
Tem apenas um sabor e uma única aparência
A palavra "saudade"só existe em português
Mas nunca faltam nomes se o assunto é ausência.

A solidão apavora mas a nova amizade encoraja
E é por isso que a gente viaja
Procurando um reencontro uma descoberta
Que compense a nossa mais recente despedida
Nosso peito muitas vezes aperta
Nossa rota é incerta
Mas o que não incerto na vida?

—GABRIEL O PENSADOR (Tás a Ver)

RESUMO

A aplicação de técnicas de correlação probabilística em registros de saúde ou socioeconômicos de uma população tem sido uma prática comum entre epidemiologistas como base para suas pesquisa não-experimentais. Entretanto, o crescimento do volume dos dados comum ao cenário imposto pelo *Big Data* provocou uma carência por ferramentas computacionais capazes de lidar com esses imensos repositórios. Neste trabalho é descrita uma solução implementada no framework de processamento em cluster Spark para a correlação probabilística de registros de grandes bases de dados do Sistema Público de Saúde brasileiro. Este trabalho está vinculado a um projeto que visa analisar a relação entre o Programam Bolsa Família e a incidência de doenças associadas à pobreza, tais como hanseníase e tuberculose. Os resultados obtidos demonstram que esta implementação provê qualidade competitiva em relação a outras ferramentas e abordagens existentes, comprovada pela superioridade das métricas de tempo de execução.

Palavras-chave: Correlação probabilística, Computação intensiva de dados, Sistemas de saúde pública, Apache Spark

ABSTRACT

The application of probabilistic correlation techniques in health or socioeconomics records of a population has been a common practice among epidemiologists as a basis for non-experimental research. However, the growth data volume, common to the Big Data scenario has caused a shortage of computational tools able to handle these huge repositories. We describe a solution based on the Spark clustered processing framework for probabilistic correlation of records from large bases of the Brazilian public health system. This work is bound to a project that aims at to analyze the relationship between the Bolsa Familia and the incidence of diseases associated with poverty, such as leprosy and tuberculosis. The results show that this implementation provides competitive quality compared to other existing tools and approaches, proven by the superiority of the runtime metrics.

Keywords: Probabilistic record linkage, Data-intensive computing, Public healthcare, Apache Spark

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

Capítulo 1

Neste primeiro capítulo, é passada uma visão global acerca dos principais conceitos, métodos e resultados desta dissertação.

INTRODUÇÃO

Nos dias de hoje, uma espécie de "*Big Data Analytics*"permeia as áreas do comércio, finanças e governo. Seja no monitoramento de transações feito pelas companhias de cartões de crédito afim de encontrar atividades fraudulentas, ajudando analistas financeiros a vasculhar dados do mercado para identificar boas oportunidades de investimentos ou permitir o rastreamento de comportamentos dos cidadãos através da internet ou tráfego telefônico (MILLER, 2012). Atualmente, a área de saúde encara uma grande quantidade de conteúdo proveniente de pontos de atendimento (postos, hospitais, etc.), instrumentos médicos sofisticados e comunidades médicas baseadas na Web (CHEN; CHIANG; STOREY, 2012).

Apesar de todas essas oportunidades, a análise de grandes fontes de dados de saúde populacional ainda tem uma evolução muito mais lenta que aquelas voltadas à outras áreas. As áreas de biomedicina, epidemiologia e saúde coletiva ainda carecem de aplicações competitivas que atendam seus requisitos dentro do contexto de *Big Data*. Alguns autores (SCANNAPIECO; VIRGILLITO; ZARDETTO, 2013) apontam o uso do pareamento, aliado a limpeza de dados e uma remodelagem dos métodos estatísticos de análise como uma tendência de uso por institutos nacionais de saúde para compreensão de eventos estudo de impacto de intervenções em saúde.

1.1 MOTIVAÇÃO

Neste contexto, o Instituto de Saúde Coletiva da Universidade Federal da Bahia (ISC-UFBA), dispondo de uma equipe multidisciplinar de pesquisadores relevantes (nas áreas de epidemiologia, biologia e estatística), propôs um projeto que tem como objetivo a avaliação do impacto do Programa Bolsa Família (PBF) na tuberculose e hanseníase utilizando *correlação* do Cadastro único (CadÚnico) com dados de morbimortalidade (presentes em outras bases governamentais). O termo *record linkage* teve sua origem na área de saúde pública a partir da necessidade de convergir prontuários distintos de pacientes usando atributos como *nome*, *data de nascimento*, entre outros. Métodos cada vez

mais sofisticados, motivados principalmente pelas aplicações de estudos epidemiológicos, incluíram as áreas da ciência da computação e estatística (WINKLER, 2006).

Considerando o imenso volume das bases de dados e a quantidade de tuplas em cada uma delas, a tarefa de parear registros que representem um mesmo indivíduo através dos repositórios facilmente se enquadraria no contexto de *Big Data*, o que se justifica pela necessidade de se utilizar um ambiente que permita o emprego pleno das propriedades de: escalabilidade, que possibilita o crescimento do uso de recursos de acordo com a demanda; distribuição, para diminuir os tempos de execução; e tolerância à falhas, para garantir a recuperação de cada etapa já concluída em caso de falhas na execução. Todos esses requisitos motivaram estudos sobre conceitos e ferramentas que pudessem dar suporte à implementação de todo um conjunto de rotinas capazes de fazer a correlação probabilística em grandes fontes de dados.

1.2 OBJETIVOS

O cenário imposto pelo projeto proposto pelo ISC o incluiu no contexto de *Big Data* por conta dos requisitos de velocidade no processamento de um imenso volume de dados. Por conta disso, foi eleita uma ferramenta com alto poder de processamento, capaz de prover abstração na tolerância a falhas, distribuição de tarefas e arquivos em *cluster*, o Apache Spark, um sistema de computação distribuído, confiável e de alto desempenho.

Dessa forma, o principal objetivo deste trabalho é atender os requisitos da levantados pelo ISC-UFBA para uma ferramenta de correlação probabilística das bases de dados disponíveis, servindo como suporte computacional ao estudo de análise do impacto dos programas de transferência condicional de renda mantidos pelo Governo Federal do Brasil em doenças relacionadas a pobreza, como Hanseníase e Tuberculose. Almeja-se desenvolver um fluxo de trabalho capaz de lidar com grandes fontes de dados populacionais dentro do contexto de *Big Data* que tenha resultados competitivos em relação a outras soluções já existentes na literatura no que se refere a acurácia dos métodos utilizados.

1.3 METODOLOGIA

O passo inicial foi o levantamento do estado da arte e ferramentas que pudessem contribuir no atendimento aos requisitos encontrados no contexto do projeto de colaboração. Estes estudos preliminares, além de discussões e investigações sobre as principais soluções já utilizadas pelos orgãos interessados em iniciativas anteriores tiveram importância substanciais na definição de cada etapa do fluxo de trabalho proposto para a correlação probabilística de grandes bases. Uma das principais contribuições foi a escolha pelo *framework* Spark (ZAHARIA et al., 2010), justificada pela sua capacidade de promover processamento em *cluster* com alto poder de abstração, escalabilidade, tolerância á falhas e superioridade em relação a ferramentas já existentes.

A Figura 1.1 ilustra o método proposto por este trabalho que tem em sua composição as etapas: de *análise descritiva*, cujo objetivo é buscar entendimento pleno sobre as bases de dados envolvidas e seus contextos; de *pré-processamento*, responsável por preparar os registros de cada fonte que será pareada, possibilitando a reformatação, blocagem e

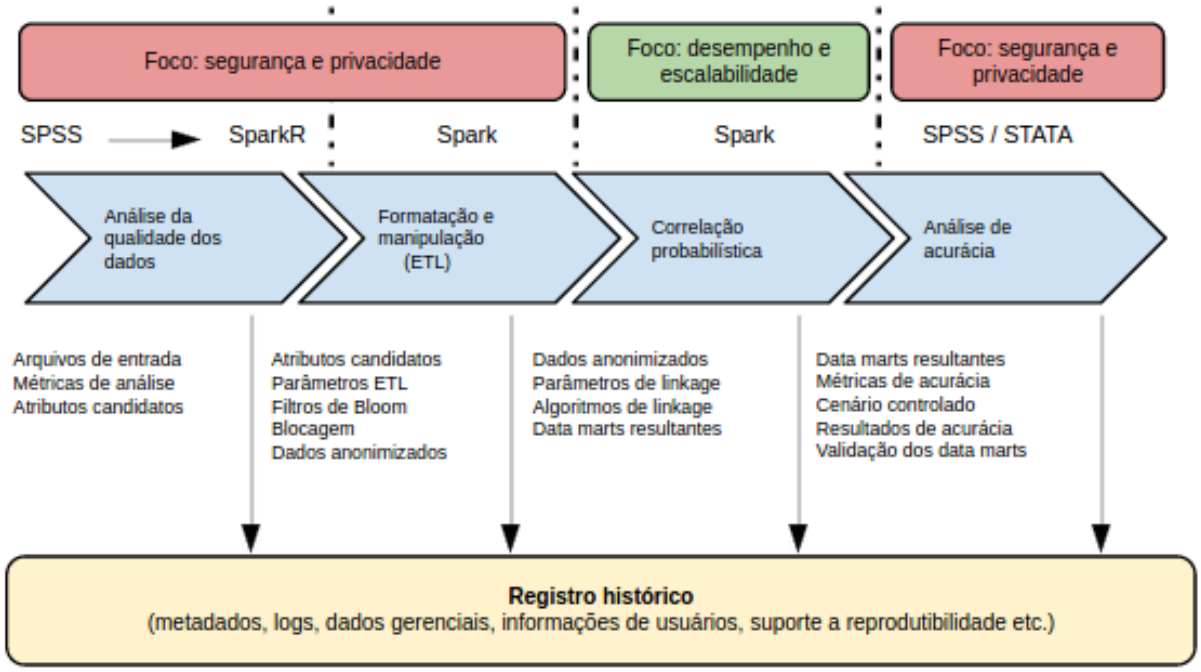

Figura 1.1 Fluxo de execução proposto para a correlação.

anonimidade dos dados envolvidos; de *correlação*, onde acontece a verificação de similaridade entre um par de registros e a decisão sobre seu pareamento; e a geração do data mart, que recupera os pares com decisão positiva pela etapa anterior e disponibiliza em um arquivo que será utilizado incialmente pela equipe de estatísticos do projeto e para os pesquisadores que visam o estudo do impacto das políticas de transferência condicional de renda na morbimortalidade das doenças ligas a pobreza. A Figura 1.1 mostra o arcabouço proposto para efetuar o pareamento entre duas bases de dados. Todas as etapas da metodologia estão compreendidas em três fases interdependentes.

1.4 RESULTADOS

Resultados preliminares mostram uma qualidade competitiva quando comparado a alternativas já existentes, ainda que não tenha sido medida a acurácia resultante do método (sendo este estudo uma responsabilidade de componentes estatísticos dentro da equipe multidisciplinar que compõe o projeto de colaboração), a Correlação em Spark teve resultados superiores as principais ferramentas alternativas encontradas na literatura quanto as métricas de sensibilidade, pares falsos positivos e negativos. A Figura 1.2 mostra o resultado do comparativo entre essas soluções num cenário controlado.

Em outra perspectiva, a Correlação em Spark se apresenta como alternativa ideal para pareamento de grandes bases. Sua capacidade de lidar com fontes de dados no contexto de *Big Data* se dão graças ao pleno emprego dos conceitos de distribuição de tarefas, tolerância a falhas e escalabilidade.

Por outro lado, o processamento de bases compostas de poucos registros apresentou a

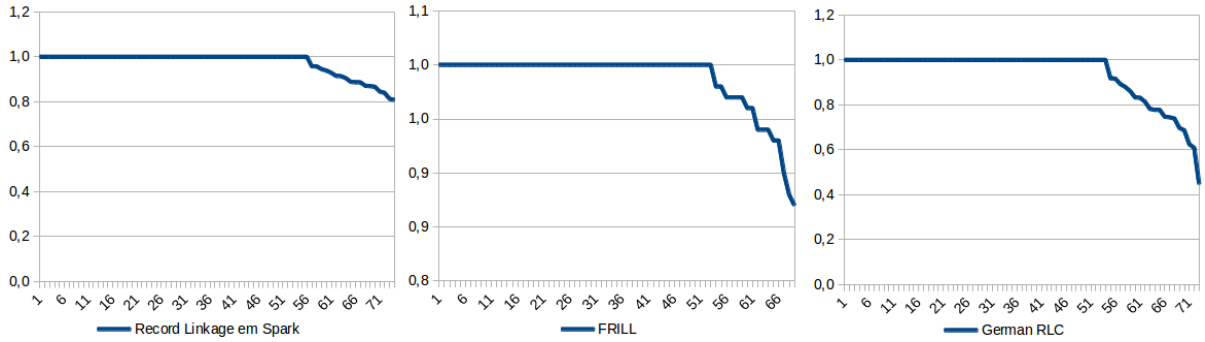

Figura 1.2 Comparativo entre a Correlação em Spark e outras ferramentas.

	CadÚnico	**SIH**
Tamanho (registros)	~76 milhões	~57 mil
Normalização (UTF8)	2310,4 s	36,5 s
Anonimização (Bloom)		
Blocagem (cidades)		
Correlação	9,03 h	
Geração do Data Mart	1,31 h	

Tabela 1.1 Tempos de execução de todo o fluxo de trabalho.

necessidade de considerar uma combinação de técnicas de na sua execução, pelo fato de não permitirem um uso satisfatório dos recursos. Neste caso, abordagens *multithreading* se mostraram mais eficazes.

1.5 ORGANIZAÇÃO DO TRABALHO

No próximo capítulo são apresentados os conceitos de *Big Data*, sua perspectiva na saúde e o projeto proposto pelo ISC-UFBA dentro deste contexto. Um levantamento do estado da arte para os tópicos de ferramentas de processamento de grandes conjuntos de dados e pareamento probabilístico de registros estarão expostos no Capítulos 3 e **??**, respectivamente. A solução proposta por este trabalho será mostrada em detalhes no Capítulo 4 e seus resultados alcançados nas primeiras execuções, assim como um comparativo com outras ferramentas já existentes podem ser encontradas no Capítulo 5. Uma discussão sobre as conclusões, desafios e trabalhos futuros deste trabalho estão no Capítulo 6.

Capítulo

2

Este capítulo tem o objetivo de contextualizar o Big Data, suas implicações e desafios na área da saúde. São apresentados alguns projetos neste contexto, com foco no estudo de caso que motivou este trabalho.

BIG DATA NA ÁREA DE SAÚDE

2.1 INTRODUÇÃO

O termo *Business Inteligence* nasceu nos anos 1990 e representava um conjunto de tecnologias que permitiam suportar o estudo de dados históricos para compreensão de eventos no passado de alguma organização ou população. Nos anos 2000, se popularizou o "*Business Analytics*", uma extensão que permitia um melhor apoio às decisões empresariais. Entende-se por *analytics* o extensivo uso de um conjunto de dados, alcançado através de sua análise estatística e quantitativa gerenciada a partir de modelos preditivos ou baseado em fatos para dirigir ações e decisões (DAVENPORT; HARRIS, 2007).

Algumas soluções computacionais foram desenvolvidas para dar suporte a esta tendência às análises, se tornando tradicionais em algumas áreas. O *Data Warehouse* (DEVLIN, 1996) abrange soluções de armazenamento de um grande conjunto de dados, provendo tolerância à falhas e uma velocidade considerável na entrega das informações processadas. O *Data Mining* (BERRY; LINOFF, 1997) por sua vez, é um termo que incluía técnicas de exploração de um conjunto de dados para detecção de comportamentos, satisfação e curiosidade de consumidores ou grupo de usuários.

Com o crescimento absurdo dos dados, gerados *online* através de mídias sociais, transações, dispositivos baseados em sensores ou por estudos científicos, novos desafios tornaram os métodos tradicionais de armazenamento, processamento e análise desses dados obsoletos (AHUJA; MOORE, 2013). Neste contexto, surge o *Big Data*, termo que converge técnicas inovadoras para manipulação de grandes conjuntos de dados eficientemente. Todavia, o termo ainda carece de um consenso sobre seus significados ou implicações. Autores como (ZIKOPOULOS; EATON et al., 2011) o julgam impróprio, pois implica que o os dados pré-existentes sejam pequenos de alguma forma ou sugere (errôneamente) que o "tamanho"desses dados é o único desafio a ser enfrentado.

As soluções baseadas em *Big Data* não sobrepõem aquelas fundamentadas em mineração de dados, *data warehouse*, *business intelligence* ou *data analytics*, porém, são

ideais quando o cenário requisita: **a)** além da análise de dados estruturados (organizados em blocos semânticos. Ex.: bases relacionais), também semi-estruturados (baseado em estruturas auto-descritiva. Ex.: hipertexto) e não-estruturados (não possuem uma estrutura definida. Ex.: dados multimídia); **b)** quando todos ou uma grande porção dos dados devem ser analisados, e não somente uma pequena amostra; **c)** uma análise iterativa ou exploratória (ZIKOPOULOS; EATON et al., 2011).

Num cenário típico deste termo é possível observar, ao menos, algumas das seguintes propriedades (ou requisitos) nos dados que serão processados (popularmente chamadas de quatro V's do *Big Data*): **a)** Produção ou coleção de um *Volume* massivo de dados; **b)** Alta *Velocidade* no processamento; **c)** *Variedade* na apresentação e formato do dado (texto, imagem, vídeo); **d)** *Veracidade* das informações colecionadas. Desta forma, o projeto exposto na Seção 2.3 requer uma solução no contexto de *Big Data* devido aos seus requisitos de processar um grande volume de dados estruturados (em sua totalidade), com uma velocidade de processamento iterativo rápido o suficiente para suportar decisões acerca de novas intervenções nos sistemas sociais ou de saúde providos pelo Governo Federal, respeitando e garantindo a veracidade das informações contidas nas bases utilizadas.

2.2 BIG DATA NA SAÚDE

Nos dias de hoje, o "*Big Data Analytics*"permeia as áreas do comércio, finanças e governo. Seja no monitoramento de transações feito pelas companhias de cartões de crédito afim de encontrar atividades fraudulentas, ajudando analistas financeiros a vasculhar dados do mercado para identificar boas oportunidades de investimentos ou permitir o rastreamento de comportamentos dos cidadãos através da internet e tráfego telefônico (MILLER, 2012). Atualmente, a área de saúde encara uma enxurrada de conteúdo proveniente de pontos de atendimento (postos, hospitais, etc.), instrumentos médicos sofisticados e comunidades médicas baseadas na Web (CHEN; CHIANG; STOREY, 2012).

Autores como (MURDOCH; DETSKY, 2013) sugerem que a análise dos dados gerados por pacientes e médicos podem contribuir para a entrega de um serviço de saúde melhorado. De forma análoga, (MILLER, 2012) aponta duas fontes para uso das tecnologias de *Big Data* na saúde. A primeira é impulsionada pelo estudo de dados genômicos (genotipagem, a expressão do gene ou dados de sequenciamento). A segunda se baseia na análise dos pacientes (registros eletrônicos ou de seguradoras, prescrições, resultados, etc).

Apesar de todas essas oportunidades, a análise de grandes fontes de dados de saúde populacional ainda tem uma evolução muito mais lenta que aquelas voltadas à outras áreas. As áreas de biomedicina, epidemiologia e saúde coletiva ainda carecem de aplicações competitivas que atendam seus requisitos dentro do contexto de *Big Data*. Autores como (SCANNAPIECO; VIRGILLITO; ZARDETTO, 2013) apontam o uso do pareamento, aliado a limpeza de dados e uma remodelagem dos métodos estatísticos de análise como uma tendência de uso por institutos nacionais de saúde para compreensão de eventos estudo de impacto de intervenções em saúde.

Num contexto semelhante ao projeto exposto na póxima seção, o uso do pareamento

de bases de dados administrativas no Canadá conseguiu promover um satisfatório suporte para estudos longitudinais e observacionais baseados em coortes populacionais, fomentando notável avanço à comunidade de pesquisas na área de saúde coletiva (DOIRON; RAINA; FORTIER,). O trabalho (RUBIN, 1997) evidencia a complexidade dos estudos não-experimentais[1] em grandes fontes de dados.

2.3 ESTUDO DE CASO: ANÁLISE DE IMPACTO DO PROGRAMA BOLSA FAMÍLIA (ESTUDO DE CASO)

Seguindo a hipótese de que a melhor alternativa para redução da morbimortalidade de doenças ligadas a pobreza não está condicionada somente numa atuação direta e eficaz na assistência à saúde (WHO, 2006), mas também com uma preocupação e planejamento em intervenções que abordem os determinantes sociais de uma população (FRIEDEN, 2010), o Instituto de Saúde Coletiva da Universidade Federal da Bahia (ISC-UFBA), dispondo de uma equipe multidisciplinar de pesquisadores relevantes (nas áreas de epidemiologia, biologia e estatística), propôs um projeto que tem como objetivo a avaliação do impacto do Programa Bolsa Família (PBF) na tuberculose e hanseníase utilizando pareamento do Cadastro único (CadÚnico) com dados de morbimortalidade (presentes em outras bases governamentais).

Classificado como um programa de transferência de renda condicional (PTRC), o Programa Bolsa Família pode ser caracterizado como um plano de distribuição de renda que tem como alvo famílias pobres, permitindo que pais invistam no capital humano dos filhos (FISZBEIN; SCHADY; FERREIRA, 2009). Os objetivos deste tipo de intervenção são, à curto prazo, estabelecer um piso mínimo de consumo, atenuando os efeitos da pobreza e, à longo prazo, interromper a transmissão da pobreza entre gerações. As adoções pelos PTRC tiveram início na década de 90, principalmente na América-Latina, destacando países como México e Brasil, que possuem os maiores programas de assistência (FISZBEIN; SCHADY; FERREIRA, 2009).

Segundo (SILVA, 2007), o PBF surgiu em 2003 a partir da unificação de programas já existentes, como o Bolsa Escola, a Bolsa Alimentação, o Vale Gás e o Cartão Alimentação, integrando, posteriormente, o Programa de Erradicação do Trabalho Infantil. A transferência de renda, o estímulo do acesso à educação e o desenvolvimento da família são os três principais eixos deste programa (MDS, 2014). São alvos deste PTRC as famílias com renda *per-capita* de até R$ 70,00, que são beneficiadas com valores que vão de R$ 35,00 até R$ 306,00. Somente famílias pobres que possuem crianças com até 15 anos podem participar do programa, tendo a manutenção do benefício mensal associado à duas condicionalidades, na educação, onde todas as crianças devem estar matriculadas e possuir frequência escolar mínima de 85% (MDS, 2014), e na saúde, onde as crianças devem manter a sua vacinação em dia, e mulheres grávidas devem fazer consultas pré-natais e

[1] Os estudos **experimentais** caracterizam-se por apresentarem manipulação de intervenções diretas sobre os indivíduos em estudo e atribuição aleatória da intervenção em causa. O exemplo típico de estudo experimental é o ensaio clínico randomizado. Em contraponto, estudos **não-experimentais ou observacionais** pretendem avaliar se existe associação entre um determinado fator e um desfecho sem, entretanto, intervir diretamente na relação analisada. (SCHNEIDER, 2007)

participar de cursos educativos sobre gestação (MS, 2013).

Muitas doenças apontam a pobreza como o maior dos seus fatores de risco. Alguns trabalhos como (SANTOSI; CASTROII; FALQUETOII, 2008), concluem que as condições sanitárias e econômicas são capazes de contribuir para propensão da contração da Hanseníase, assim como seu agravo. Outros autores como (BARROSO et al., 2003) e (OLIVEIRA; FILHO, 2000) evidenciam a relação tanto do surgimento, quanto da reicidência ou do aumento da resistência do *Micobacterium* agente da Tuberculose a fatores socioeconômicos similares. Neste contexto, o projeto de estudo do impacto do Programa Bolsa Família na morbimortalidade de doenças relacionadas a pobreza tem três principais objetivos específicos, explicitados a seguir.

O primeiro dos objetivos específicos será descrever as tendências dos indicadores epidemiológicos tais como, incidência, grau de incapacidade no diagnostico, percentual de cura e abandono, taxa de hospitalização, mortalidade, letalidade geral e letalidade hospitalar no grupo dos beneficiários do Bolsa Família, dos inscritos no Cadastro Único não beneficiários do PBF e na população não inscrita nesta base de dados.

Avaliar o impacto do Programa Bolsa Família nos indicadores de morbimortalidade pelas doenças citadas acima, comparando seus beneficiários e não beneficiários através de metodologias quase-experimentais representam os demais objetivos específicos.

2.3.1 Metodologias de Análise

Para possibilitar tais estudos, o grupo de pesquisa do ISC-UFBA solicitou e obteve as seguintes bases de dados governamentais que armazenam informações dos período compreendido entre 2007 à 2012: o Cadastro Único (CadÚnico), que representará um grupo de indivíduos com dados de composição familiar e socioeconômicos; a Folha de Pagamento do Bolsa Família (FPBF), que representará os indivíduos expostos a intervenção; e para possibilitar a verificação dos dados de morbimortalidade, bases dos Sistemas de Informação Hospitalar (SIH), Sistemas de Notificação e Agravo (SINAN) e Sistemas de Informação de Mortalidade (SIM). Todos esses repositórios de dados foram confiados a esta relevante equipe sob estritas regras de acesso.

Para possibilitar o estudo através de coortes, será necessária a aplicação de rotinas de pareamento entre as bases citadas anteriormente, a finalidade desta etapa será determinar registros que representam as mesmas pessoas nos diferentes conjuntos de dados. Para a Epidemiologia, uma coorte é um conjunto de indivíduos que é seguido ao longo do tempo no que concerne exposição de risco/proteção e aos seus desfechos em saúde.

Após a construção de um banco de dados único através de técnicas de pareamento de registros, o impacto do PBF nos desfechos de saúde será avaliado utilizando o *Regression Discontinuity Design* (RDD) (IMBENS; LEMIEUX, 2007), que possibilitará o estudo comparativo entre beneficiários e não-beneficiários com uma renda *per capita* de *baseline* próxima ao limiar de elegibilidade, e o *Propensity Score Matching* (PSM) (DEHEJIA; WAHBA, 1998), que permitirá avaliar os indicadores de morbimortalidade por tuberculose e hanseníase, comparando os contemplados do PBF e os não-contemplados pareados, assim como reportado em recente estudo sobre impacto dos programas de transferência de renda condicional (PTRC) no Uruguai (AMARANTE et al., 2011), que combinou pa-

reamento e RDD para avaliar o efeito do Programa de *Alimentación y Nutrición Escolar* (PANES) no baixo peso a nascer.

2.3.2 Desafios Tecnológicos

Considerando o imenso volume das bases de dados e a quantidade de tuplas[2] em cada uma delas, a tarefa de parear registros que representem um mesmo indivíduo através dos repositórios facilmente se enquadraria no contexto de *Big Data* e de *Many-Task Computing* (MTP) (RAICU; FOSTER; ZHAO, 2008), o que se justifica pela necessidade de se utilizar um ambiente que permita o emprego pleno das propriedades de: escalabilidade, que possibilita o crescimento do uso de recursos de acordo com a demanda; distribuição, para diminuir os tempos de execução; e tolerância à falhas, para garantir a recuperação de cada etapa já concluída em caso de falhas na execução.

Por conta das diferentes origens destas bases de dados, cedidas por variados órgãos governamentais com necessidades e padrões heterogêneos, a presença de atributos que tratem uma tupla de forma unívoca em mais de uma base é praticamente nula. Desta forma, um outro desafio é utilizar, planejar e implementar algoritmos que verifiquem a similaridade de um conjunto de atributos entre dois registros e decida se eles representam um mesmo indivíduo, de forma probabilística.

Todavia, é necessária uma preocupação com a qualidade das fontes de dados utilizados para as comparações. Segundo (MAGGI, 2008), métricas que evidenciam quantidade de valores ausentes ou mau preenchimento, falhas na manutenção de formato (integridade), inconsistência entre as tabelas e tuplas duplicadas caracterizam uma qualidade insuficiente de dados para execução dos algoritmos de pareamento. Além destas métricas, uma preocupação acerca da aptidão dos dados é a heterogeneidade, que pode ser estrutural: quando as bases que serão pareadas são manipuladas por sistemas de informação de diferentes contextos, o que implica o uso de um mesmo dado através de diferentes representações ou formatos; e a heterogeneidade semântica, que se apresenta quando duas entidades distintas numa fonte de dados são representadas de uma mesma forma, o que pode gerar duplicação de dados (MAGGI, 2008).

Essas informações justificam a inserção de etapas de pré-processamento como análise descritiva, reformatação e deduplicação de dados em cada uma das bases. Todas essas fases podem ser consideradas como um substrato de uma metodologia que antecede a execução do pareamento de registros e configuram um aumento na complexidade da tarefa.

Outro importante desafio que permeia toda a manipulação das bases de dados, considerando o teor descritivo e nominal das informações armazenadas além do controle de acesso exigido no momento de sua concessão pelos órgãos governamentais, é a aplicação da anonimidade sobre os atributos que podem identificar um indivíduo, sua condição social ou de saúde.

[2]Linhas de uma base de dados formada por uma lista ordenada de colunas ou atributos.

2.3.3 Trabalhos Relacionados

A maioria dos estudos de impacto do PBF sobre a saúde dos seus beneficiários foi concentrada no estado nutricional e na saúde das crianças. Resultados da segunda rodada do estudo de Avaliação de Impacto do Bolsa Família (AIBF), que utilizando dados coletados em dois momentos diferentes nos mesmos participantes e escolhendo o grupo controle por PSM asseguram uma boa robustez metodológica, demonstraram um impacto do PBF em alguns indicadores de saúde (SAGI, 2013). No trabalho de (BRAUW et al., 2012) foi visto que o número de partos prematuros entre as gestantes beneficiárias do programa foi inferior em comparação com as não beneficiarias, sendo o número de crianças nascidas a termo maior que 10,7%. No que diz respeito à nutrição, encontrou-se um Índice de Massa Corporal (IMC) maior entre os beneficiários.

O AIBF (SAGI, 2013) concluiu também que o PBF conseguiu ter um efeito no aumento de crianças vacinadas no período apropriado para vacinações contra a Poliomielite e o Tétano, Difteria e Coqueluche (DPT), sendo 11,6% e 26% maior - respectivamente - nos beneficiados pelo programa. Estes resultados contrastam com outro trabalho (ANDRADE et al., 2012), que não conseguiu evidenciar nenhum efeito do PBF no calendário vacinal utilizando só os dados da linha de base de 2005.

Em relação a mortalidade, um recente estudo (RASELLA et al., 2013) demonstrou um forte impacto do PBF na redução da mortalidade em menores de cinco anos, chegando a reduzir - nos municípios de maior cobertura - de 17% a mortalidade geral, de 65% a mortalidade por desnutrição e de 53% por diarréia, achados confirmados por outros estudos (SHEI, 2013) que demonstrou também uma sinergia com a expansão do Programa Saúde da Família (GUANAIS, 2013). Apesar da pobreza se apresentar como o principal fator de risco para algumas doenças, nenhum estudo até o momento tem analisado o efeito do PBF em doenças fortemente correlacionadas ao baixo nível socioeconômico, como Tuberculose e Hanseníase.

Capítulo 3

A evolução das ferramentas de processamento assim como o framework Spark, escolhido para dar suporte a implementação das soluções são apresentados neste capítulo.

PROCESSAMENTO DE BIG DATA

O crescimento dos volumes de dados tem exigido uma evolução nas aplicações, *frameworks* e ferramentas utilizadas em sua análise. Requisitos como alto desempenho, escalabilidade e tolerância a falhas para execuções longas impuseram a adoção de um processamento distribuído em um aglomerado de máquinas, chamados *clusters*, considerando que as capacidades de cômputo ou entrada e saída (I/O) de máquinas tradicionais não eram mais suficientes para tal demanda.

Os ambientes baseados em *clusters* trouxeram alguns desafios para programabilidade. O primeiro deles é o paralelismo, que requeria a reescrita das aplicações atendendo ao paradigma de programação paralela que dividia uma execução entre processos, *threads* ou núcleos de processador. Outro desafio dos programas baseados em *cluster* era lidar com as falhas, tanto pela parada de uma ou mais máquinas (ou *threads*) participantes do aglomerado, quanto por nós lentos. Finalmente, o compartilhamento dos recursos por vários usuários provendo o mínimo de interferência entre eles é outro grande desafio desta tendência tecnológica (ZAHARIA, 2014).

Todos esses desafios impulsionaram uma onda evolutiva de plataformas, paradigmas e modelos de programação distribuída. A próxima seção descreverá algumas destas evoluções, iniciando pelo *Google MapReduce* (DEAN; GHEMAWAT, 2008), que se apresenta como um modelo simples de processamento capaz de lidar com falhas num *cluster* sem exigir conhecimento acerca de distribuição das tarefas num aglomerado de máquinas.

3.1 O MODELO MAPREDUCE

Os desafios de como paralelizar a computação entre centenas ou milhares de máquinas em *cluster*, distribuir uma grande quantidade de dados entre esse agrupamento de ativos, lidar com falhas na execução e prover abstração suficiente para simplificar o desenvolvimento de aplicações complexas foram os principais impulsionadores do desenho do *Google MapReduce* (DEAN; GHEMAWAT, 2008). Esse *framework* foi exposto numa publicação em

2008 pela *Google* e aparentemente substituiu uma série de implementações de propósito específico capazes de processar grandes quantidades de dados brutos, como: páginas visitadas, logs de requisição *Web*, índices invertidos, rastrear componentes em rede usando uma representação através de grafos, sumarização de páginas por *host* ou *ranking* de buscas mais frequêntes num determinado período.

Além de um *framework*, o *MapReduce* introduziu um paradigma de programação baseado no modelo funcional onde, de forma abstrata, um desenvolvedor escreve aplicações utilizando duas funções básicas que manipulam os dados sem a preocupação de lidar com a divisão das tarefas num *cluster* (ou em *threads* de uma máquina local), a distribuição dos dados a serem processados e as possíveis falhas na execução. Estas funções, que serão expostas a seguir, tem propósitos específicos que desenham o fluxo de execução e necessitam que haja um tratamento no conjunto bruto de dados de entrada para a construção de uma estrutura baseada em pares chave-valor [1]: **i)** a função *map*, escrita pelo usuário, processa o par de entrada e cria pares chave-valor intermediários. Antes de finalizar a execução, a biblioteca do *MapReduce* agrupa todos os valores com a mesma chave; **ii)** a função *reduce*, também implementada pelo usuário, aceita cada uma das chaves e seus respectivos valores e os agrupa.

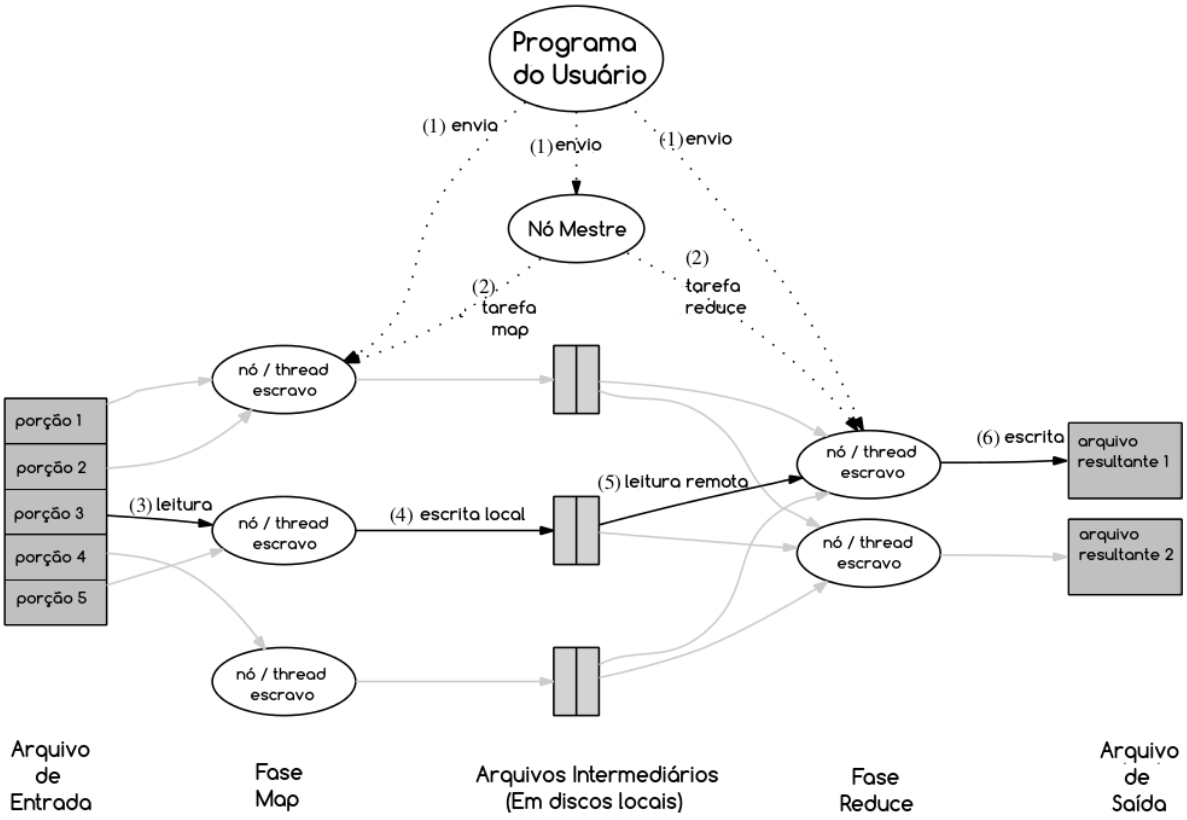

Figura 3.1 Fluxo de execução do *MapReduce* (DEAN; GHEMAWAT, 2008)

[1]Modelo de dados que corresponde o valor a uma chave, a velocidade de consulta é maior do que banco de dados relacional, suporta armazenamento em massa e alta concorrência. (HAN et al., 2011)

Na Figura 3.1 é possível entender o fluxo de execução do *MapReduce* proposto pela *Google*. São sete etapas básicas, explicitadas por (DEAN; GHEMAWAT, 2008): **i)** a biblioteca do MapReduce dividirá os dados de entrada em *splits* (pedaços) de 16-64MB e iniciará os processos em *threads* numa ou várias máquinas; **ii)** uma dessas *threads* ou máquinas será eleita como Master da execução, que deverá delegar as tarefas *Map* ou *Reduce* para os *workers* ociosos; **iii)** estes nós escravos, responsáveis pela fase *Map* lerão e analisarão o conteúdo dos *splits* a que foi atribuído, seu papel será organizar todos os pares que tiverem a mesma chave de modo que fiquem próximos; **iv)** todo dado intermediário gerado pelo processamento da fase *Map* é armazenado localmente nos nós *workers* e sua localização enviada para o *Master* que notificará isso para os nós escravos *Reduce*; **v)** após a notificação, os nós responsáveis pela etapa *Reduce* acessarão, através de *Remote Procedure Protocol* (RPC), e ordenarão todas as chaves numa porção de dados intermediários, agrupando todas as chaves iguais; **vi)** numa penúltima etapa, todos os dados resultantes são concatenados mantendo todas as chaves idênticas associados aos seus valores intermediários, gerando um arquivo resultado único; **vii)** por fim, quando todos os nós terminaram sua execução, a biblioteca MapReduce enviará um sinal ao programa principal, escrito pelo usuário, lhe fornecendo o resultado.

A necessidade de executar análises em enormes conjuntos de dados em *clusters* com centenas ou até milhares de máquinas torna imprescindível que se planeje um modelo de tolerância à falhas. O *framework MapReduce* consegue identificar paradas por *crash* das máquinas envolvidas na execução porquê o nó mestre faz uma verificação períodica em cada nó escravo, determinando que aquele que não respondê-la numa janela de tempo é falho. Encontrado um nó com problemas, caso ele pertença a fase *Map*, o *split* destinado ao computador defeituoso deve ser reprocessado, já que os dados intermediários gerados pelo nó defeituoso não poderão ser acessados. Caso a parada tenha acontecido num nó *Reduce* com tarefas em progresso, o tratamento é o mesmo que o descrito para os nós *Map*, porém, se ele já tiver sido concluído não faz sentido reexecutá-lo uma vez que não se utiliza o sistema de arquivo local para armazenar o arquivo de saída, e sim em sistemas de arquivos globais.

Uma outra abstração provida por este *framework* é a capacidade de lidar com nós que não são falhos, porém apresentam uma lentidão na execução de suas demandas (seja por problemas com o disco, comunicação de rede ou baixa capacidade de processamento). Para dirimir este problema quando percebida a lentidão, o *Google MapReduce* demanda a alguns nós ociosos cópias dos dados que estão sendo processados pelos retardatários. Desta forma, é possível garantir que a execução não sofrerá mais atrasos, o resultado do primeiro nó que concluir a execução é armazenado, e os demais são descartados.

3.2 O APACHE HADOOP

O *Hadoop* (HADOOP, 2009) nasceu através de um esforço da *Yahoo!* em desenvolver um mecanismo de busca na *web* mais robusto que o *Nutch* (KHARE et al., 2004), uma plataforma de código fonte aberto que era executado num cluster com 20 máquinas e suportado por uma equipe limitada de desenvolvedores. Atraídos pelas vantagens apresentadas pelo *MapReduce* um ano antes, os principais pesquisadores da *Yahoo!* se empenharam em

construir uma ferramenta sob os mesmos moldes. Pouco tempo depois, o projeto foi incubado e licenciado pela *Apache*, fazendo nascer o *Apache Hadoop* (VENNER; CYRUS, 2009). A idéia de resolver problemas de processamento com a magnitude de *petascale computing*[2] utilizando um grande conjunto de máquinas de baixo custo, provendo alto nível de abstração para desenvolvedores e com provimento de tolerância a falhas similar ao proposto pela ferramenta do *Google*, porém, todas as vantagens de uma ferramenta com código fonte aberto são incluídos a este novo *framework*.

O fluxo de execução e escrita de aplicações sobre a plataforma *Hadoop* se assemelha bastante ao explicitado neste trabalho sobre o *MapReduce*, porém, para garantir o ordenamento das chaves geradas pelo processamento da fase *map*, armazenada em dados temporários, foi necessária a criação de uma etapa que antecedesse a fase *reduce*, a *shuffle*. Esta implementação é crucial quando existem quando existem mais nós envolvidos com a fase *map* que com a *reduce*. A terminologia também sofreu algumas alterações, no *Hadoop* o papel do *Master* no *MapReduce* é assumido pelo *JobTracker*, este também é responsável por lidar com as requisições e gerência do sistema de arquivos distribuídos. Os nós escravos ilustrados na Figura 3.1 que podem ser responsáveis por uma das duas fases da execução são chamados de *TaskTracker* na proposta do framework de código fonte aberto da *Yahoo!*, estes podem assumir diferentes papéis numa mesma execução, podendo executar tarefas *map* e, posteriormente, *reduce*.

3.3 O USO DE SISTEMAS DE ARQUIVOS DISTRIBUÍDOS

O fato de prover um *framework* confiável focou os esforços dos desenvolvedores do *Google MapReduce* na implementação de um modelo de tolerância à falhas exclusivo para o fluxo de execução e seus principais componentes, abstraindo por completo os possíveis problemas no acesso e manutenção dos dados de entrada e saída deste processamento. Isso foi possível graças a aplicação de um sistema distribuído de arquivos capaz de atender os requisitos de distribuição, escalabilidade, confiabilidade e disponibilidade, o *Google File System* (GFS) (GHEMAWAT; GOBIOFF; LEUNG, 2003).

Esse sistema de arquivos foi proposto em 2003 e vem atendendo todos os serviços da *Google* durante todos esses anos, o objetivo é prover abstração no acesso de arquivos ou conjunto de dados num aglomerado de centenas ou milhares de máquinas de baixo custo. O projeto foi desenhado sobre quatro afirmativas básicas: falhas não são exceções, e devem ser contornadas; grandes arquivos abrigam ou representam objetos menores, o que exige um estudo de como dividi-los e tratar seu constante uso; a maioria das escritas em arquivos são através da inserção (*append*) de mais conteúdo; e a flexibilização de alguns tópicos como a consistência ou o uso de Interfaces de Programação de Aplicações (API) devem evitar o redesenho e promover o co-desenho de aplicações.

Numa abordagem prática, uma aplicação *MapReduce* utiliza a API cliente do GFS e faz as requisições dos arquivos que deverão ser processados de acordo com os parâmetros passados pelo desenvolvedor dos métodos *map* e *reduce*. Este cliente, como ilustrado na Figura 3.2 deve solicitar a localização desses arquivos ao *GFS Master*, responsável

[2]Refere-se a sistemas computacionais com desempenho capaz de processar na ordem de petaflops, ou seja, um quadrilhão de operações de pontos flutuantes por segundo.

por manter todos os metadados da infraestrutura assim como obter as atualizações sobre conteúdo, localidade e versionamento dos dados alocados nesta arquitetura. Após obter a resposta sobre a localização dos dados pelo *Master*, o cliente poderá se conectar diretamente com o *chunckserver* para obtê-lo, este último é um servidor de baixo custo capaz de armazenar um máximo de 64MB de conteúdo (o que explica o tamanho máximo de cada split no *MapReduce*).

As vantagens diretas dessa infraestrutura de armazenamento estão associadas às propriedades de localidade, sabendo a localização dos *splits* que serão processados, o ideal é que o *Master* delegue a um *worker* um conteúdo mais próximo dele, geralmente local (considerando que o *cluster* que executa o GFS é o mesmo que aloca os nós do *MapReduce*).

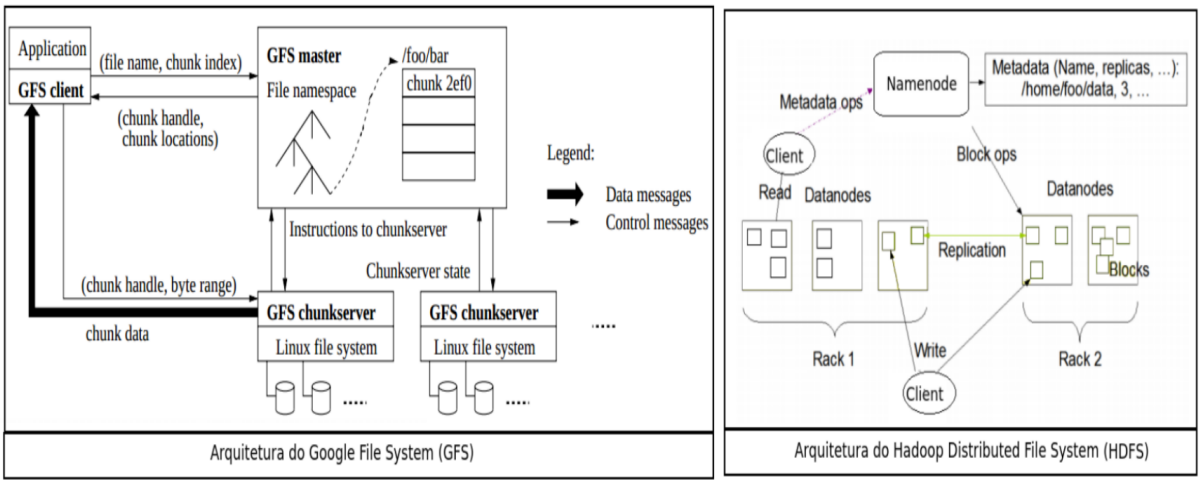

Figura 3.2 Arquitetura dos sistemas de arquivos distribuídos que suportam o Google MapReduce e Apache Hadoop, adaptado de (GHEMAWAT; GOBIOFF; LEUNG, 2003) e (BORTHAKUR, 2008)

.

Se apresentando como um componente evoluído do Apache Nutch e subprojeto Apache Hadoop, o *Hadoop Distributed File System* (HDFS) tem inspiração nas vantagens do GFS e no entendimento das vantagens do uso de um sistema de arquivo distribuído aliado a um *framework* de processamento de larga escala. Propondo mudanças básicas à solução proprietária da *Google*, o HDFS se apresenta como uma alternativa de software livre para qualquer aplicação que tenha como requisito o armazenamento escalável, tolerante a falhas, confiável e robusto.

Na Figura 3.2 é possível perceber o quanto são similares em algumas estratégias e divergentes na terminologia. O *HDFS Datanode* tem papel similar ao *GFS chunckserver* assim como o existe uma semelhança nos papéis executados pelo *HDFS Namenode* e o *GFS Master*. Ambas as soluções fornecem tolerância à falhas através de replicação, relaxamento das regras POSIX e consistência, maior vazão no acesso aos dados e possibilidade de escalar o armazenamento.

3.4 FRAMEWORK SPARK

O framework de código aberto do *Apache Hadoop* impulsionou uma série de variações que buscavam satisfazer requisitos específicos alheios aos propósitos originais do *framwork* proposto pela *Google*. O GraphLab (LOW et al., 2010), Apache Hive (THUSOO et al., 2009), Yahoo Pig (OLSTON et al., 2008) e o Twister (EKANAYAKE et al., 2010) são exemplos de inciativas *Beyond Hadoop* (além do *Hadoop*) (MONE, 2013) que basicamente mantém as primitivas do *MapReduce* mas possuem o objetivo de gerar novos níveis de abstração. Porém, alguns autores apontaram para uma insuficiência deste paradigma percussor para aplicações destinadas ao processamento de dados relacionados, com fortes requisitos de iterações, *machine learning* ou até mesmo com diferentes exigências no desempenho. Esses novos frameworks classificados como *Beyond MapReduce* (além do *MapReduce*) (RÖSCH, 2012) tem como exemplo o Google Dremel (DREMEL, 1955), Jumbo (GROOT; KITSUREGAWA, 2010), Shark (XIN et al., 2013), Apache Spark (ZAHARIA et al., 2010).

O Apache Spark (ZAHARIA et al., 2010) se apresenta como um sistema de computação em *cluster* de alto desempenho. Este *framework* pemite o uso de linguagens como o Python, Java e Scala para construção de aplicações de propósito geral, provendo suporte a ferramentas dos mais altos níveis de abstração, como manipulação de dados usando *structured query language* (SQL), aprendizado de máquina e fluxo de processamento orientado a grafos.

É possível usar esta ferramenta numa máquina local, distribuindo tarefas entre núcleos de um processador ou ainda num aglomerado de máquinas sob supervisão de sistemas gerenciadores de *clusters*, como o Apache Mesos (HINDMAN et al., 2011), o YARN (VAVILAPALLI et al., 2013) ou um gerenciador nativo ao próprio Spark. Os critérios que levaram ao seu uso, em detrimento das demais alternativas citadas neste capítulo estão centradas: na adoção de linguagens de programação de propósito geral (como o Python, Java e Scala), que permite a criação de aplicações que atendam a cenários com especificidades e requisitos diferentes dos demais; no alto poder de abstração sobre as propriedades de distribuição de tarefas, dados e tolerância à falhas nas *threads* ou máquinas num *cluster*; no alto desempenho provido pelo fluxo de execução baseado em *working sets* de memória; e na evolução ativa por conta do número crescente de usuários e desenvolvedores envolvidos no seu projeto.

3.4.1 Resilient Distributed Datasets

Os *resilient distributed datasets* (RDDs), representa uma coleção de objetos que será particionada e distribuída entre máquinas participantes da execução de uma tarefa *Spark*, garantindo a possibilidade de reconstrução de uma porção de dados no caso de perda de partição. Os principais benefícios do uso do *Apache Spark* podem ser explorados com a criação de um RDD a partir de um conjunto de dados que serão processados. Existem duas formas básicas de criar essa coleção resiliente de dados, ambas utilizam a classe *SparkContext* para criação dos objetos: paralelizando um vetor de itens iteráveis criado em tempo de execução ou referenciando um conjunto de dados presente num

sistema de armazenamento externo, como os sistemas de arquivos distribuídos: HDFS (BORTHAKUR, 2008), HBase (GEORGE, 2011), ou qualquer outra fonte de dados que obedeça a estrutura *Hadoop Format* (ZAHARIA et al., 2010).

A manipulação de um RDD pode ser feita através de duas classes de operações básicas: **transformações**, que criam um novo conjunto de dados a partir de um já existente. Implementado usando *lazy evaluation*, que promove melhor desempenho e administração de grandes conjuntos de dados; e as **ações**, que retornam valor ao programa do usuário depois de processar um RDD. As transformações apenas são computadas quando uma ação requisita seus resultados (ZAHARIA et al., 2010). Na tabela 3.1, é possível entender um pouco sobre as principais funções do *framework* Apache Spark.

Na Tabela 3.1, nota-se que algumas funções do *framework*, como *reduceByKey* estão apenas disponíveis em RDDs de pares chaves/valor. Os nomes das funções se referem a nomenclatura usada em outras APIs em Scala e linguagens funcionais. Por exemplo, o *map* percorre itens um-a-um, enquanto o *flatMap* mapeia cada valor de entrada em uma ou mais saídas (muito similar ao MapReduce) (ZAHARIA, 2014). Além disso, é possível que o desenvolvedor de uma aplicação escolha qual RDD deve estar contido em memória, ou qual o nível de armazenamento ele deseja, como mostrado na Tabela 3.2. É possível definir explicitamente quando e qual RDD será armazenado em memória, usando as funções *cache()* ou *persist()*.

A escolha sobre o nível de armazenamento ideal promove um *trade-off* entre uso de memória e eficiência da CPU. Os autores e desenvolvedores do Spark (SPARK, 2014a) recomendam o seguinte processo para apoio na eleição do que melhor se encaixa a um determinado cenário:

Memória suficiente: Se existe recurso suficiente para armazenar os RDDs em memória confortavelmente, o nível de armazenamento padrão (MEMORY_ONLY) é a melhor opção. Esta é a alternativa mais eficiente em termo de uso de CPU.

Memória razoável: Recomenda-se experimentar o MEMORY_ONLY_SER que manipula os objetos de forma a utilizar eficientemente o espaço disponível, mantendo uma velocidade razoável no seu acesso.

Uso dos discos: Deve-se evitar o uso do disco para armazenar as partições de um RDD (ou mesmo toda a coleção de dados) utilizando a constante DISK_ONLY a menos que a função que processará os dados seja bastante custosa ou não haja a necessidade de re-computar os dados.

3.4.1.1 Vantagens do RDD

O autor (ZAHARIA, 2014) apresentou em seu trabalho uma breve comparação entre o uso dos RDDs e tecnologias que suportam aplicações e trabalhos relacionados, como as baseadas em Memória Distribuída Compartilhada (MDC)[3]. A principal diferença nestes modelos de abstração é que o RDD apenas é criado através de transformações enquanto a MDC permite leitura e escrita em cada local compartilhado.

[3]Modelo de memória distribuída onde aplicações lêem e escrevem espaços de endereçamentos globais. Promove boa abstração mas configura um desafio sua implementação com tolerância à falhas em clusters de alto desempenho. (NITZBERG; LO, 1991)

Transformações	
Função	**Descrição**
map(func)	Retorna uma novo conjunto de dados distribuídos através do processamento de cada elemento de um RDD pela função *func*.
flatMap(func)	Similar ao *map*, mas fiel a função Map do paradigma *MapReduce*, onde cada valor de entrada pode ser mapeado a nenhuma ou mais chaves de saída.
filter(func)	Retorna um novo conjunto de dados composto pelos itens de entrada que sejam de interesse da função *func*.
distinct([numTasks])	Retorna um novo conjunto de dados composto apenas por elementos distintos na fonte.
reduceByKey(func, [numTasks])	Processa um dado de entrada no formato de pares (chave,valor) e retorna um RDD com todos os valores de cada chave distinta são agregadas usando a redução implementada na função *func*.
Ações	
Função	**Descrição**
collect()	Retorna todos os elementos de um conjunto de dados em uma estrutura de dados (vetor ou matriz) para o programa do usuário. Utilizado para trazer o resultado do processamento das transformações.
count()	Executa a contagem de todos os elementos de um RDD.
foreach(func)	Executa uma função *func* em cada elemento de um RDD.
reduce(func)	Agrega os elementos de um RDD segundo a implementação da função *func*.

Tabela 3.1 Principais funções do *framework* Spark. (SPARK, 2014a)

Um benefício sobre o uso de RDDs é a maior eficiência no uso da tolerância à falhas, o re-processamento de uma partição isenta o *framework* da sobrecarga inerente ao uso de *checkpoints* durante a execução. Além disso, as partições perdidas ou falhas podem ser redistruibuídas entre as máquinas do cluster. Uma segunda vantagem seria a possibilidade de tratar, assim como no MapReduce, os nós lentos através de cópias das tarefas em outros nós, operação não trivial quando se usa o MDC.

Além disso, duas outras vantagens importantes no uso de RDDs são: a possibilidade de escalonar tarefas para os nós do cluster se baseando na localidade dos dados que

Nível de armazenamento	Descrição
MEMORY_ONLY	Armazena RDDs de forma desserializada. Caso não caiba em memória, algumas partições não serão guardadas em cache.
MEMORY_AND_DISK	*Similar ao MEMORY_ONLY, mas as partições que não cabem em memória são armazenadas em disco.*
MEMORY_ONLY_SER	Armazena RDDs como objetos Java serializados (uma matriz de bits por partição).
MEMORY_AND_DISK_SER	*Similar ao MEMORY_ONLY_SER, porém armazena as partições que não cabem na memória em disco.*
DISK_ONLY	Guarda todas as partições de um RDD em disco.

Tabela 3.2 Níveis de armazenamento em memória do Spark. (SPARK, 2014a)

serão processados, o que incrementa o desempenho e diminui os problemas com o tráfego de rede; além disso, como mostrado na Tabela 3.2, o Spark lida satisfatoriamente com memória insuficiente, pois quando não há recurso o suficiente, as partições sobressalentes são armazenadas em disco mantendo desempenho similar aos sistemas de processamento paralelo de dados atuais (ZAHARIA, 2014).

3.4.2 Variáveis Compartilhadas

Para facilitar a passagem de variáveis ou coleções de dados que serão manipuladas por todas as máquinas de um aglomerado de máquinas remotas, comumente feito por tarefas Spark (como *map* ou *reduce*), este framework provê dois padrões de variáveis compartilhadas que são copiadas para todos os participantes de uma execução sob a condição de somente leitura: *variáveis broadcast* e *acumuladores*.

3.4.2.1 Variáveis Broadcast Criado para repassar de forma eficiente uma cópia somente leitura de um grande conjunto de dados para a memória de cada nó trabalhador num cluster. O Spark utiliza algoritmos de distribuição de variáreis eficazes para reduzir os custos com a comunicação entre processos (SPARK, 2014b).

```
>>> broadcastVar = sc.broadcast(["Salvador", "Recife", "Sergipe"])
<pyspark.broadcast.Broadcast object at 0x102f09f10>

>>> broadcastVar.value
["Salvador", "Recife", "Sergipe"]
```

3.4.2.2 Acumuladores Para suportar contadores ou somadores de forma confiável e paralela, o *framwork* Spark possibilita o uso de *acumuladores*. Estas variáveis apenas suportam operações de adição e são nativamente implementadas no formato numérico, mas é possível que programadores adicionem novos tipos (SPARK, 2014b).

```
>>> accum = sc.accumulator(0)
Accumulator<id=0, value=0>

>>> sc.parallelize([1, 2, 3, 4]).foreach(lambda x: accum.add(x))
...
10/09/29 18:41:08 INFO SparkContext: Tasks finished in 0.317106 s

>>> accum.value
10
```

3.4.3 Características Técnicas

Para usar o Spark, o desenvolvedor deve escrever e executar um *programa mestre* que irá se conectar ao *cluster* de nós trabalhadores (*workers*). Neste programa deverão estar definidos os RDDs, suas transformações e ações. Enquanto o papel dos *workers* é armazenar, operar partições dos RDDs e notificar o programa mestre sobre as etapas concluídas, este último deve monitorar as atualizações e garantir que todo o fluxo de execução dos dados acontecerá sem problemas.

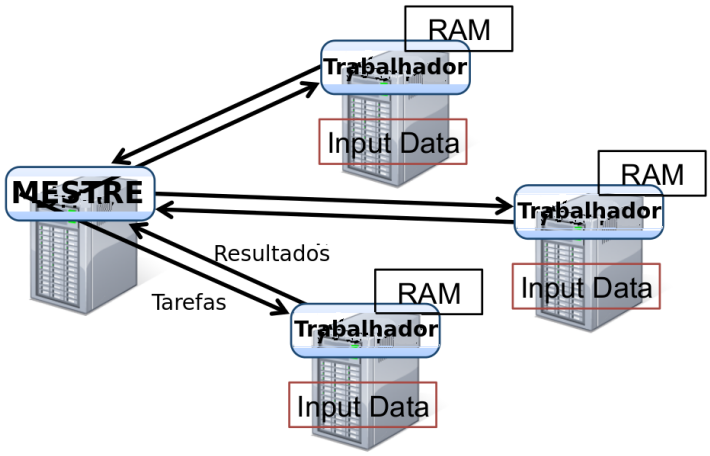

Figura 3.3 Arquitetura de execução do Spark (ZAHARIA, 2014)

A Figura 3.3 mostra a arquitetura de execução do Spark. A proposta é que um programa executado no nó mestre carregue múltiplos trabalhadores que lerão blocos de dados de entrada a partir de um sistema de arquivos distribuído e possa persistir partições RDD em memória.

3.4.4 Exemplo de Aplicação

Um grande arquivo CSV, com variáveis separadas pelo delimitador ";"(ilustrado pela Figura 3.4), alocado no sistema de arquivos distribuídos HDFS (BORTHAKUR, 2008), armazena todas as solicitações feitas a uma equipe de suporte em TI pelos seus usuários. Supondo que haja a necessidade de efetuar uma tarefa de mineração destes dados, surge a oportunidade de mostrar como o Spark lida com suas aplicações.

```
cod;nome;solicitacao;status
65657;JOAO PEREIRA SOUZA;MANUTENCAO;CONCLUIDO
21565;PEDRO DE JESUS;CONFIGURACAO;EM ANDAMENTO
...
78456;EZEQUIEL PENA;MANUTENCAO;EM ANDAMENTO
65989;JOAO PEREIRA SOUZA;CONFIGURACAO;EM ANDAMENTO
96548;MARIA FLOR SILVA;MANUTENCAO;CONCLUIDO
```

Figura 3.4 Amostra da base fictícia usada para mineração

As motivações para análise desta base de conhecimento são inúmeras, portanto, se fará necessária uma diminuição no escopo deste exemplo de aplicação. Será então definido como objetivo inicial a busca pelos chamados do usuário "JOAO PEREIRA SOUZA":

```
from pyspark.context import SparkContext

sc = SparkContext(...)

linhas = sc.textFile("hdfs://...")
colunas = linhas.map(lambda x: x.split(';'))(1))
nome = colunas.filter(_startsWith("JOAO PEREIRA SOUZA"))
nome.cache()
```

Utilizando o *SparkContext* é possível criar um RDD a partir de um arquivo alocado num repositório HDFS. A variável *nome* usa o delimitador do arquivo para extrair apenas a segunda coluna da base de dados através da função *map()*, logo depois, filtra os registros que iniciam com o nome do usuário que está sendo buscado usando o *filter()*. Como explicado através da Tabela 3.1, o trecho de código acima ainda não torna disponíveis os resultados da busca, para que isso seja possível é necessário aplicar alguma ação que retorne ao programa mestre a matriz resultante desta execução.

```
registrosUsuario = nome.collect()
cont = nome.count()
```

A estratégia de persistir em memória um determinado RDD permite um ganho significativo de processamento a cada ação feita a partir dele. Por exemplo, além de coletar a lista de chamados deste usuário através do *collect()*, pode-se contar todos os registros deste conjunto resiliente de dados com o *count()*.

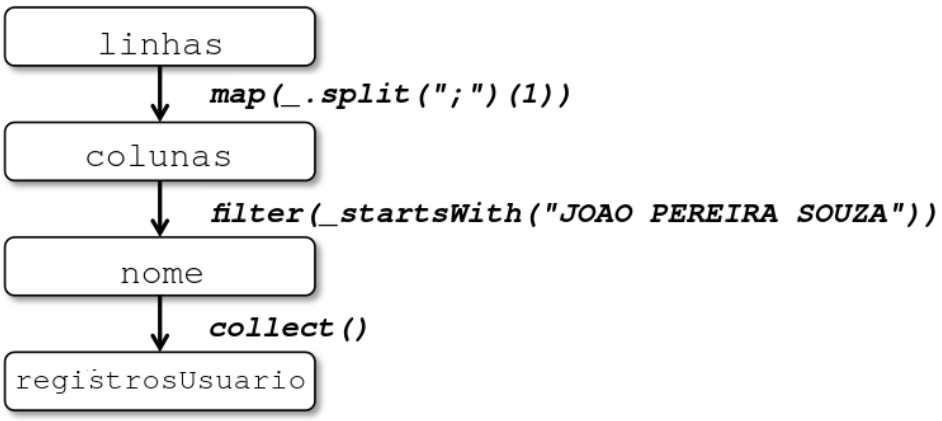

Figura 3.5 Grafo de fluxo dos RDDs gerados durante a execução. Adaptado de (ZAHARIA, 2014)

.

Os quadros presentes Figura 3.5 representam os RDDs gerados a cada transformação implementada na aplicação, com exceção da matriz *registrosUsuario* que é um resultado de uma ação sobre o RDD *nome*. Este exemplo de implementação foi feita com base na API Python para o Spark, o PySpark (ROSEN, 2012), que melhor detalhada na próxima seção.

3.4.5 PySpark. Implementando Tarefas Spark Usando Python.

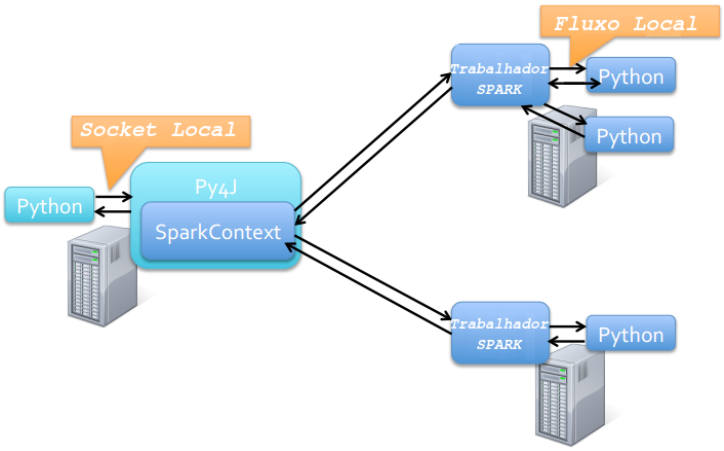

Figura 3.6 Arquitetura do PySpark, adaptado de (ROSEN, 2012)

.

Desenvolvido por (ROSEN, 2012), o PySpark é uma API que permite a escrita de tarefas Spark usando a linguagem de propósito geral Python. Como mostrado na Fi-

gura 3.6, o *programa mestre* escrito nesta linguagem utiliza das bibliotecas do Py4J[4] para se utilizar dos benefícios do *framework*. Em cada nó trabalhador, a execução da tarefa fica por conta do interpretador Python local.

O uso do PySpark não altera o substrato exposto pela Figura 3.3 do fluxo de execução do Spark, apenas inclui um passo de interpretação pelo Py4J. O fluxo local presente nos trabalhadores representam as transformações efetuadas em cada RDD manipulado pelo programa mestre, assim como mostrado na Figura 3.6.

Neste trabalho, todas as implementações foram feitas utilizando o PySpark por conta do alto poder de abstração além da curva de aprendizado confortável provida por ele. Em contato com seu principal desenvolvedor, ficou claro que as perdas no desempenho são mínimas na comunicação entre processos ou processamento de dados, mesmo considerando a inclusão do Py4J na estrutura original do Spark.

A correlação probabilística de registros, também conhecido como *record linkage* é um meio de combinar informações de uma variedade de arquivos computadorizados (WINKLER, 2006). Seu uso se aplica quando não existem identificadores únicos para os registros de duas fontes de dados (FELLEGI; SUNTER, 1969). Na literatura, esta tarefa pode ser chamada de *identificação de objetos* (TEJADA; KNOBLOCK; MINTON, 2001), *data cleaning* (RAHM; DO, 2000), *combinação* ou *junção aproximada* (GRAVANO et al., 2001), *combinação difusa* (ANANTHAKRISHNA; CHAUDHURI; GANTI, 2002) e *resolução de entidade* (BENJELLOUN et al., 2009).

O termo *record linkage* teve sua origem na área de saúde pública a partir da necessidade de convergir prontuários distintos de pacientes usando atributos como *nome*, *data de nascimento*, entre outros. Métodos cada vez mais sofisticados, motivados principalmente pelas aplicações de estudos epidemiológicos, incluíram as áreas da ciência da computação e estatística (WINKLER, 2006).

3.5 FORMALIZAÇÃO DO RECORD LINKAGE

Os primeiros a propôr um modelo matemático formal para o *record linkage* foram (FELLEGI; SUNTER, 1969). Estes autores apresentaram um modelo rígido de decisão sobre o pareamento de registros presentes em dois arquivos (L_A e L_B), preenchidos a partir de um processo de geração de *unidades de população* (UPs) com um número de características associadas (por exemplo: nome, idade, sexo, estado civil, endereço, etc). Assume-se que esta etapa de geração de UPs incorre alguns erros ou incompletude aos registros resultantes em cada arquivo e ainda assim existem alguns elementos que são comuns em ambos os conjuntos de registros.

Desta forma são estabelecidas duas populações A e B compostas por elementos a e b respectivamente:

$$A \ X \ B = \{(a,b); a \in A, b \in B\}$$

Consequentemente, o conjunto de pares ordenados é a união de dois conjuntos disjuntos de pares verdadeiramente positivos (M ou *matched*) e pares verdadeiramente negativos (U ou *unmatched*):

[4]Py4J permite que programas Python dinamicamente acessem objetos Java. (PY4J, 2014)

$$M = \{(a,b); a = b,\ a \in A,\ b \in B\}$$

e

$$U = \{(a,b); a \neq b,\ a \in A,\ b \in B\}$$

A decisão sobre quais os componentes de $A\ X\ B$ irão compôr cada um dos seus conjuntos disjuntos ($(a,b) \in M$ ou $(a,b) \in U$) é suportada pela criação de um vetor de concordância ou índice de similaridade entre cada um dos atributos dos registros a e b comparados (uma melhor descrição sobre métodos de obtenção de similaridade serão expostos na Seção 3.6 deste capítulo):

$$\gamma[\alpha(a),\beta(b)] = \{\gamma^1[\alpha(a),\beta(b)],..., \gamma^K[\alpha(a),\beta(b)]\}$$

A função γ executada sobre $A\ X\ B$ também pode ser escrita $\gamma(a,b)$ ou $\gamma(\alpha,\beta)$ (ou apenas γ). Elas denotam a verificação da similaridade entre cada um dos atributos comuns aos dois registros. O conjunto de todas as verificações γ é chamada *espaço de comparação* e é representado por Γ. Observando $\gamma(a,b)$ busca-se decidir se esses são pares **verdadeiros positivos (A_1)** ($(a,b) \in M$) ou **verdadeiros negativos (A_3)** ($(a,b) \in U$). No entanto, existem casos em que não se pode julgar com propriedade sobre o *link* entre dois registros, o que permite que se haja uma terceira decisão chamada **possível par (A_2)**.

Por conta disso, foi definida uma *regra de linkage* L que mapeia todo *espaço de comparação* Γ para um conjunto funções de decisões randômicas $D = \{d(\gamma)\}$ onde:

$$d(\gamma) = \{P(A_1 \mid \gamma),\ P(A_2 \mid \gamma),\ P(A_3 \mid \gamma)\};\ \gamma \in \Gamma$$

e

$$\sum_{i=1}^{3} P(A_i \mid \gamma) = 1.$$

Cada valor observado de γ tem atribuído a sí a probabilidade de pertencer a cada um dos grupos de decisão (A_1, A_2, A_3) correspondente. Para alguns ou todos os possíveis valores de γ, a função de decisão pode se tornar uma variável de distribuição randômica degenerada [5], isto é, pode atribuir a um dos grupos uma probabilidade 1.

Com isso, considerando níveis de erros associados à *regra de linkage*, torna-se necessário o cálculo da probabilidade de γ, dado que $(a,b) \in M$ pelo $m(\gamma)$. Então:

$$\begin{aligned} m(\gamma) &= P\{\gamma[\alpha(a),\beta(b)] \mid (a,b) \in M\} \\ &= \sum_{(a,b)\in M} P\{\gamma[\alpha(a),\beta(b)]\}.P[(a,b)|M]. \end{aligned}$$

De forma análoga, o cálculo da probabilidade de γ, dado que $(a,b) \in U$ pelo $u(\gamma)$. Então:

[5]Em matemática, uma distribuição degenerada é a distribuição de probabilidade de uma variável aleatória discreta cujo suporte consiste de somente um valor.

$$u(\gamma) = P\{\gamma[\alpha(a),\beta(b)]|(a,b)\in U\} \\ = \sum_{(a,b)\in U} P\{\gamma[\alpha(a),\beta(b)]\}.P[(a,b)|U].$$

Dos níveis de erros citados acima associados a *regra de linkage*, existem dois tipos: os **falsos positivos** acontecem quando pares de registros que não correspondem ao mesmo indivíduo ou evento são definidas como participantes do conjunto A_1 (ou seja, não deveriam ser pareados); os **falsos negativos** são pares que se referem a registros que deveriam ser pareados mas foram incluídos no conjunto A_3. A Figura 3.7 ilustra como estão dispostas as classificações dos pareamentos probabilísticos propostos por (FELLEGI; SUNTER, 1969).

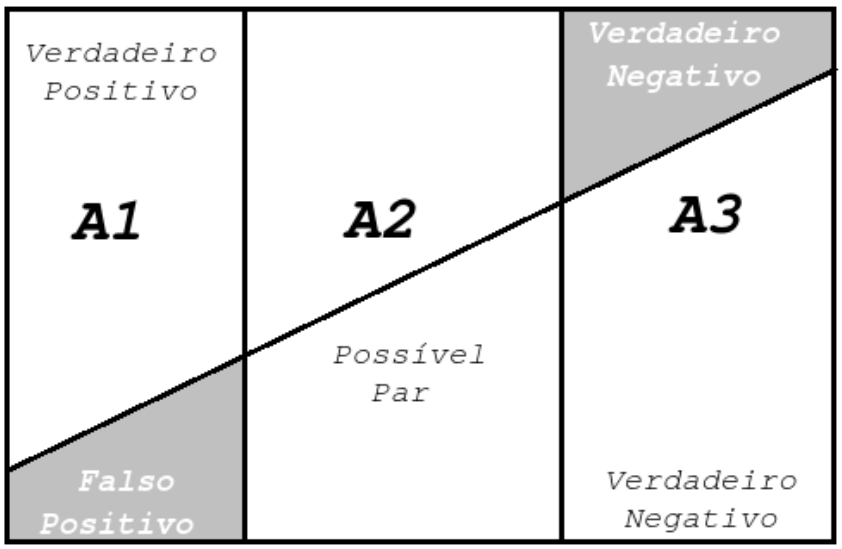

Figura 3.7 As três regiões do modelo de probabilidade proposto por (FELLEGI; SUNTER, 1969). Adaptado de (RACCHUMI-ROMERO, 2008)

O calculo da probabilidade de ocorrer um falso positivo é:

$$P(A_3|U) = \sum_{\gamma} \in \Gamma u(\gamma.P(A_1|\gamma).$$

Os falsos negativos, por sua vez, tem sua probabilidade calculada através da seguinte fórmula:

$$P(A_1|M) = \sum_{\gamma} \in \Gamma u(\gamma.P(A_3|\gamma).$$

Uma *regra linkage* num espaço Γ estará em níveis μ, λ (onde $0 < \mu < 1$ e $0 < \lambda < 1$), denotado por $L(\mu, \lambda, \Gamma)$ se:

$$P(A_1|U) = \mu$$

e

$$P(A_3|M) = \lambda.$$

Calcular estes níveis são necessários para definir se foi alcançada a ***regra linkage* ótima**. Esta última maximiza a probabilidade de comparações positivas (decisões de registros para os grupos A_1 e A_3) ou minimiza a probabilidade de falha por fazer uma disposição positiva. Para que isso seja possível, é necessário que se satisfaça:

$$\mathrm{P}(A_2 \mid \mathrm{L}) \leqslant \mathrm{P}(A_2 \mid L^{'}) \; \forall \; L^{'}(\mu, \lambda, \Gamma)$$

O autor (FELLEGI; SUNTER, 1969) considerou esta uma boa abordagem considerando os custos da verificação manual sobre os pares do grupo A_2. Contudo, se a probabilidade dos registros pertencentes a A_2 não é pequena, o processo de pareamento é de utilidade questionável. O emprego de métodos que aumentem a sensibilidade do algoritmo utilizado se faz necessário neste caso.

3.6 ÍNDICES DE SIMILARIDADE

A verificação de similaridade é o substrato da tarefa de *correlação*, especificamente na etapa $\gamma(a,b)$ ou $\gamma(\alpha,\beta)$, onde cada elemento dentre os atributos de um determinado registro num arquivo é relacionado e comparado com as demais tuplas em outro. O objetivo é dar suporte à decisão sobre o pareamento destes registros, minimizando o número de possíveis pares do conjunto A_2. No modelo proposto por (FELLEGI; SUNTER, 1969), os atributos são comparados na sua forma literal, permitindo a aplicação de alguns pesos, em seu trabalho, ele também define uma probabilidade de encontrar atributos ou registros análogos com esta abordagem.

Entretanto, considerando fontes de dados administrativos distintos, é possível encontrar formatos e contextos diferentes para representar valores idênticos. Além disso, outros problemas acerca da entrada e manutenção de dados em um repositório de registros podem ser ocasionados por fatores humanos (como erros de digitação, omissão ou ambiguidade), de aplicação (má implementação na forma de popular as bases, mal emprego das regras que bancos de dados) e obsolecência (considerando a dinamicidade do mundo-real, alguns dados perdem seu valor com o passar do tempo).

A seguir serão apresentadas algumas das principais propostas para verificação de similaridade classificadas entre três classes: as **baseadas em edição** como as propostas pelo *SOUNDEX* (MORTIMER; SALATHIEL, 1995), *distância de edição* (RISTAD; YIANILOS, 1998) e *Jaro* (COHEN; RAVIKUMAR; FIENBERG, 2003); as **baseadas em *tokens*** como os coeficientes de *Jaccard* (REAL; VARGAS, 1996) e *Sorensen-Dice* (DICE, 1945); e **híbridos** como o *fuzzy string matching* (FSM ou *approximate matching*) (CHAUDHURI et al., 2003). O índice de Sorensen-Dice terá uma abordagem mais detalhada por ser a que será empregada no fluxo de trabalho proposto por esta dissertação, sob justificativas que serão expostas no próximo capítulo.

3.6.1 Códigos fonéticos

Algoritmos capazes de lidar com códigos fonéticos buscam simplificar a representação das palavras, o que pode servir para indexação de bases de dados, verificação de seme-

lhança entre dois nomes ou agrupamento de palavras semelhantes. O objetivo é encontrar palavras iguais ou equivalentes minimizando suas diferenças de escrita e mantendo sua fonética ou pronúncia.

Nesta subseção serão expostos três principais algoritmos baseados em códigos fonéticos. O emprego deles no contexto do pareamento pode ser feito nas etapas de verificação de similaridade, indexação, blocagem ou pré-processamento de texto (estas duas últimas serão melhor descritas nas próximas seções ou capítulos):

3.6.1.1 SOUNDEX O SOUNDEX (ODELL, 1918) (PFEIFER et al., 1995) se apresenta como um algoritmo para indexação de nomes através de sua pronúncia e fonemas. Originalmente desenvolvido por Margaret Odell and Robert Russel na *U.S. Bureau of Archives*, teve suas primeiras implementações e evoluções usando a lingua inglesa como base para construção de códigos a partir das letras de um determinado nome.

A implementação desta técnica seguindo os fonemas brasileiros precisou dirimir uma série de desafios, como por exemplo, a presença de uma letra representando vários fonemas como o "X"com som de "Z"em exame, "csi"em hexa ou "cha"em enxame. Existem também, os casos em que várias letras representam o mesmo fonema como "S", "SS"e "Ç"ou ainda "C", "Q"e "K". Também são encontradas palavras de origem estrangeiras que utilizam letras inexistentes no alfabeto oficial brasileiro como "K", "W"e "Y", ou ainda, letra que não representa fonema como a letra "H"(LUCENA, 2006). Como resultado, segue a Tabela 3.3 usada para e implementação desse algorítmo em Português do Brasil:

Codificação SOUNDEX	
Código fonético	**Letras**
1	B,F,P,V
2	C, G, J, K, Q,S, X, Z—
3	D, T
4	L
5	M, N
6	R

Tabela 3.3 Distribuição fonética das letras segundo o código SOUNDEX (LUCENA, 2006)

Para funcionamento deste algoritmo fonético deve-se: transformar todas as letras para maiúsculas, ignorando a acentuação; manter sempre a primeira letra da palavra; substituir por 0 todas as ocorrências das vogais, letras não nativas da língua portuguesa e o 'H' ('A', 'E', 'I', 'O', 'U', 'W', 'Y', 'H'); substituir as letras pelos seus códigos seguindo a Tabela 3.3; e remover todos os códigos repetidos; remover todos os códigos 0.

3.6.1.2 BuscaBR Dadas as insuficiências do SOUNDEX com muitas especifidades da lingua portuguesa, algumas iniciativas nacionais buscaram desenvolver alternativas ao indexador fonético americano. A codificação do BuscaBR executa substituições de conjuntos de letras por seu fonema correspondente, diferente dos códigos usados pelo

SOUNDEX, removendo letras repetidas em sequência, e também letras mudas (BOHM, 2010).

Codificação BuscaBR				
Fonema	**Letras**		**Fonema**	**Letras**
I	I		M	N, RM, GM, MD, SM e terminação em AO
B	BR		N	NH
F	PH		P	P
G	GR, MG, NG, RG		S	Ç, X, TS, C, Z, RS
J	GE, GI, RJ, MJ, NJ		T	LT, TR, CT, RT, ST
K	Q, CA, CO, CU, C		V	W
L	LH		L	R

Tabela 3.4 Distribuição fonética segundo o BuscaBR

No algoritmo do BuscaBR, deve-se: converter todas as letras para maiusculo, eliminar a acentuação, efetuar as substituições seguindo a Tabela 3.4, eliminar todas as terminações no conjunto T{'S', 'Z', 'R', 'M', 'N', 'L'}; eliminar todas as vogais e o 'H' além das letras em duplicidade.

3.6.1.3 Metaphone-pt_BR Assim como as alternativas acima, o Metaphone-pt_BR (aO; ROSA, 2012) gera chaves (índices) de palavras a partir do modo como elas são pronunciadas. Porém um grande avanço em comparação as demais, é que este algoritmo tem suas substituições baseadas em expressões regulares além de outras semelhantes expostas nas Tabelas 3.3 e 3.4. Desta forma

Codificação Metaphone_pt-BR				
Fonema	**Letras**		**Fonema**	**Letras**
R	CR, *R*,		Z	EX*
K	Q, CA, CO, CU, C, K, CHR		T	TH, T, T*
S	CE, CI, Ç, SS, SCE, SCI		F	PH
G	GA, GO, GU, GH		L	L*
M	M, N*,		2	R, R$, RR
J	GE, GI, GHE, GHI		3	NH
X	SCH, SH, CH, EXE, EXI,		1	R

Tabela 3.5 Parte da distribuição fonética do Metaphone-pt_BR

Os passos do algoritmo do Metaphone-pt_BR são bastante semelhantes aos demais, porém o número de substituições é muito maior. A Tabela 3.6 mostra um comparativo da aplicação de cada um dos códigos fonéticos abordados. O número de variações a partir do SOUNDEX é bastante vasta, porém, foram trazidos para esta dissertação apenas aqueles mais relevantes capazes de tratar fonemas em português do Brasil.

Por ser mais antigo e não ser nativo ao Português brasilieiro, o SOUNDEX se mostra menos eficiente que as demais soluções expostas nesta subseção. Estudos preliminares

	SOUNDEX	BUSCABR	METAPHONE-PT_BR
Kubitscheck	K132	KBSK	KBTXK
Kubixeque	K122	KBSK	KBXK
Walter	W436	VT	VLTR
Valter	V436	VT	VLTR
Teresina	T625	TRSM	TRZN
Terezina	T625	TRSM	TRZN

Tabela 3.6 Tabela comparativa da aplicação dos códigos fonéticos em Português do Brasil

apontam uma maior capacidade do BuscaBR de amenizar erros de digitação com mesmo fonema, porém, até o término da construção deste texto, não houve um estudo aprofundado sobre a acurácia de cada um deles.

3.6.2 Distância de Edição

O objetivo de se calcular a Distância de Edição, ou Índice de Levenshtein, é definir qual o menor número de inserções, deleções ou substituições para que duas porções de texto (*strings*) se tornem idênticas. Dessa forma é possível definir o quanto essas variáveis são similares (RISTAD; YIANILOS, 1998).

Originalmente proposto por (LEVENSHTEIN, 1966), o índice resultante da Distância de Edição recebeu várias aplicações na área de Ciência da Computação, principalmente nas buscas em bancos de dados ou cálculo de similaridade entre duas *strings*. O algoritmo abaixo, escrito em Python, foi retirado do site (WIKIBOOKS, 2014) e mostra como calcular o menor número de edições (inserções, deleções ou substituições) entre duas palavras:

```
def lev(a, b):
    if not a: return len(b)
    if not b: return len(a)
    return min(lev(a[1:], b[1:])+(a[0] != b[0]), \\
               lev(a[1:], b)+1, lev(a, b[1:])+1)
```

Neste mesmo site existem outras implementações do mesmo algoritmo com propósitos ou métricas de desempenho diferentes. Na Tabela 3.7, o código acima é empregado nos mesmos nomes da Tabela 3.6.

	Distância de Edição
Kubitscheck X Kubixeque	6
Walter X Valter	1
Teresina X Terezina	1

Tabela 3.7 Índice de Levenshtein aplicado para calcular a distância de edição entre dois nomes.

É possível perceber uma sensibilidade considerável deste método em comparação aos

algorítmos fonéticos, pois nomes com mesma pronúncia podem gerar distâncias relativamente altas em alguns casos.

3.6.3 Índice de Jaro-Winkler

O indice de Jaro-Winkler calcula a similaridade aproximada entre duas *strings* e foi criada especificamente para aplicações de correlação (WINKLER, 2006) como uma alternativa ao modelo de comparação proposto por (FELLEGI; SUNTER, 1969).

```
jaro(s1,s2) = 1/3( #comum/#str\_tam1 + #comum/#str\_tam2 +
                0.5 #tranposicoes/#comum)
```

O algoritmo desta proposta segue três passos simples: computar o tamanho das variáveis *strings* (#str_tam1, #str_tam2); encontrar o número de caracteres comuns (#comum); e encontrar o número de transposições (#tranposicoes) necessárias para que as duas strings se tornem idênticas. Seguindo a fórmula acima.

	Índice de Jaro-Winkler
Kubitscheck X Kubixeque	0.6700
Walter X Valter	0.8888
Teresina X Terezina	0.95

Tabela 3.8 Índice de Jaro-Winkler entre pares de strings.

Apesar do estudo de (WINKLER, 1985) apontar algumas vantagens de uma versão menos aprimorada deste algoritmo sobre outros estudos em sua época, é possível perceber que ainda há uma insuficiência deste índice no que tange a sensibilidade para fonemas.

3.6.4 Coeficiente de Jaccard

O índice proposto por Jaccard (REAL; VARGAS, 1996) teve o objetivo inicial de calcular a similaridade entre áreas biológicas e sua relação com as espécies que as ocupam para determinar o raio de sua preservação em reservas naturais. Seu uso na computação também é capaz de prover uma comparação entre duas *strings*. A fórmula usada para calcular este índice de similaridade entre duas variáveis de texto é:

$$J = \frac{C}{A + B + C`}$$

A variável A representa o número de caracteres na *String1* que não está na *String2*. B é inversamente proporcional ao A. Já C ilustra todos os caracteres em ambas as *strings*.

Utilizando a classe que emprega o algoritmo de Jaccard no pacote Distance (MEYER, 2013) nos pares de nomes da Tabela 3.9, é possível perceber que o tamanho das *strings* é um fator importante no cálculo da sua similaridade, pois quanto maior é o conjunto de caracteres, mas sensível aos erros se torna o coeficiente.

	Índice de Jaccard
Kubitscheck X Kubixeque	0.5833
Walter X Valter	0.2857
Teresina X Terezina	0.25

Tabela 3.9 Índice de Jaccard calculado. *Strings* similares possuem coeficientes mais próximos do 0.

3.6.5 Fuzzy String Matching

A *Fuzzy String Matching*, também conhecido como *Approximate String Matching* (HALL; DOWLING, 1980) foi originalmente desenvolvida para busca de conteúdo na internet com base em padrões pré-estabelecidos. Este algoritmo segue a lógica *fuzzy*, que retorna valores aproximados ao invés dos absolutos 1 ou 0 da lógica tradicional. Desta forma, é possível lidar com dois problemas clássicos: encontrar um subconjunto de *strings* em uma porção de texto ou encontrar um dicionário que combine com um padrão de forma aproximada.

	Coeficiente FSM
Kubitscheck X Kubixeque	0.4
Walter X Valter	0.83
Teresina X Terezina	0.88

Tabela 3.10 *Fuzzy String Matching* aplicado para definir o nível de similaridade entre duas strings.

Essa ferramenta de cálculo de similaridade é dita híbrida, pois sua busca por similares é feita de forma semelhante ao coeficiente de Levenshtein (LEVENSHTEIN, 1966), porém, seu resultado aproximado é diferente dos absolutos providos pela distância de edição. Os resultados alcançados na Tabela 3.10 foram obtidos a partir do uso da ferramenta FuzzyWuzzy (COHEN, 2011).

3.6.6 Índice de Sorensen ou Dice

Este coeficiente nasceu sob o objetivo de suportar estudos ecológicos para expressar de maneira quantitativa como duas espécies estão associadas na natureza. Em outras palavras, a idéia é prover uma métrica que calcule a similaridade na relação entre espécies em diferentes ecossistemas (DICE, 1945). Esta medida é alcançada através de um coeficiente chamado indice de coincidência, posteriormente conhecido como índice de Sorensen ou Dice.

$$Dice = \frac{2h}{a + ab}$$

A expressão acima mostra como é possível efetuar o cálculo de similaridade. A variável a corresponde ao número de ocorrências de uma determinada espécie A, assim como b para

o B, já a variável h representa o somatório dos números de espécies a e b que coincidem. Aplicando esta técnica para comparação de *strings* no contexto da computação pode-se substituir o termo 'espécie' por 'caractere'. Desta forma, a e b serão contadores de um determinado caractere nas strings A e B respectivamente e a h seria uma variável incremental para cada caractere numa mesma posição em A e B.

	Sorensen-Dice
Kubitscheck X Kubixeque	0.58
Walter X Valter	0.83
Teresina X Terezina	0.85

Tabela 3.11 Índice de Sorensen-Dice aplicado para pares de *strings*

Os valores alcançados pelo Índice de Sorensen-Dice mostrados na Tabela 3.11 revelam uma maior sensibilidade em comparação ao Índice de Jaro-Winkler para nomes com maior número de caracteres, porém, seus resultados em comparação ao Jaccard mostram uma sensibilidade inferior em todos os nomes comparados.

3.7 BLOCAGEM

Considerando que uma tarefa de pareamento probabilístico de dados efetua a comparação de cada registro de uma primeira fonte de dados a todos de uma segunda, a escalabilidade se torna uma preocupação muito relevante. Ao parear duas bases de dados hipotéticas A e B, cada uma delas com 5 mil registros, vão ser necessárias 25 milhões (AxB) de comparações detalhadas. Isso é claramente impraticável (MICHELSON; KNOBLOCK, 2006).

Esta questão trás à tona a necessidade de uma etapa de pré-processamento, chamada *blocagem*, que antecede o pareamento e é capaz de analisar e gerar grupos de registros candidatos em cada base. O nome desta tarefa se deve ao fato de que ela particiona o produto cartesiano de comparações em blocos mutualmente exclusivos (NEWCOMBE, 1967).

Para a escolha do melhor método de blocagem a ser empregado numa tarefa de pareamento, deve-se ponderar entre dois principais itens: primeiro, o número de candidatos em cada bloco deve ser pequeno para garantir a redução no número de comparações; em segundo lugar, um conjunto candidato deve ser inclusivo o suficientes para impedir que pares de registros verdadeiros sequer sejam comparados. Esse *trade-off* faz desta uma tarefa bastante desafiadora (MICHELSON; KNOBLOCK, 2006). Desde os modelos propostos por (FELLEGI; SUNTER, 1969) e (NEWCOMBE et al., 1986), os métodos de blocagem para propósitos gerais ou específicos evouíram bastante. Nas próximas subseções serão apresentados alguns métodos de blocagem mais relevantes ao contexto desta dissertação encontradas na literatura.

3.7.1 Blocagem Padrão

A idéia mais básica da blocagem padrão é aglomerar todos os registros de cada base que compartilham uma *chave de blocagem* idêntica (JARO, 1989). Esta chave pode ser um atributo presente em todos os conjuntos de registros que serão pareados, como por exemplo o *ano de nascimento*, *cidade natal*, *sobrenome* ou *código postal*.

A escolha sobre a chave de blocagem deve ser feita com cuidado para evitar blocos muito grandes, por exemplo, usar a variável *sexo* pode dividir uma base de dados em dois grandes blocos. Além disso, blocos pequenos irão interferir na acurácia da correlação (BAXTER; CHRISTEN; CHURCHES, 2003), como discutido no início desta seção.

O maior problema associado a este método de blocagem é o fato dele ter uma sensibilidade a erros muito grande. Isso significa que qualquer diferença (por erro da digitação, formato ou omissão) na chave de blocagem de dois registros deveriam participar do mesmo bloco impediria que isso acontecesse. Desta forma, uma etapa de correção de erros e formatação de atributos se faz necessária para dirimir esta possibilidade.

3.7.2 Blocagem por Vizinhança Ordenada

Dada uma coleção entre duas ou mais bases de dados, a idéia deste método de blocagem consiste em concatená-las em uma lista sequencial de N registros, definir uma *chave de ordenamento* comum a elas, ordenar as bases a partir desta chave e mover uma janela de tamanho fixo e arbitrário através da N registros limitando a comparação àqueles que estão dentro do intervalo criado, como mostrado na Figura 3.8.

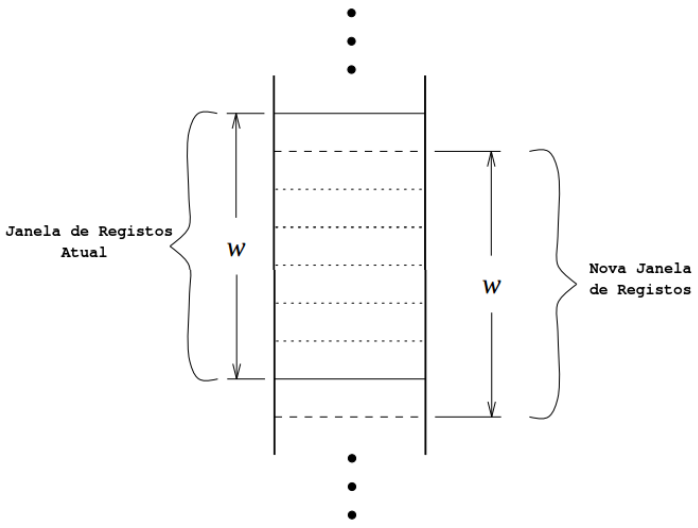

Figura 3.8 Comportamento da janela w na blocagem por vizinhança ordenada. Adaptado de (HERNÁNDEZ; STOLFO, 1998)

Se a janela tem tamanho w, então, a cada novo registro adicionado a ela será comparado com o intervalo anterior $w-1$ e a primeira tupla sai de w (HERNÁNDEZ; STOLFO, 1998). O tamanho do bloco dependerá do que foi definido para a constante w e o número de comparações será calculado por $(N \log_N)$. Os riscos desta abordagem é semelhante

aos da blocagem padrão, com a adição da preocupação acerca do tamanho desta janela, uma *chave de ordenamento* pode conter tantos registros que acaba separando mais dois registros que deveriam ser pareados que os limites conseguem incluí-los no mesmo bloco.

Em seu trabalho, o autor (HERNÁNDEZ; STOLFO, 1998) ainda explica a importância da escolha, desenho e construção da *chave de ordenamento* para bom funcionamento deste método. Ele propõe também a eleição de mais de um atributo para participar desta chave e apresenta uma teoria equacional, que calcula a Distância de Edição entre dois registros para verificar sua equivalência.

3.7.3 Blocagem por Passos Múltiplos ou por Predicados

A blocagem por múltiplos passos (ou *multi-pass*) foi proposta por (HERNÁNDEZ; STOLFO, 1998) como uma alternativa à blocagem por vizinhança ordenada, considerando as limitações deste método ao permitir a não comparação de dois atributos com leves erros em suas chaves de ordenamento (ocasionados por omissão ou falha na digitação). Trata-se de uma estratégia onde se executa algumas vezes, de forma independente, o método exposto na última subseção. Cada vez usando um atributo principal diferente para compor a *chave de ordenamento* e uma janela w relativamente menor.

Por exemplo, numa primeira rodada, a *chave de ordenamento* pode ter como principal atributo o *nome* de um indivíduo enquanto numa segunda execução o *último nome* pode ser utilizado. Cada uma destas etapas independentes irão gerar um conjunto de registros que poderão ser comparados e, eventualmente, pareados. Porém, antes que seja aplicada a correlação destas bases, elas serão submetidas a uma aplicação de cálculo do fecho transitivo[6] e então entregues às próximas etapas do pareamento.

Na literatura, o autor (MICHELSON; KNOBLOCK, 2006) apresenta uma implementação avançada que define do método *multi-pass* com *esquemas de blocagem* (*Blocking Schemes* ou BS) através da conjunção {*primeira letra do nome* (p), *primeiro nome* (n)} $\wedge$ { *token de combinação1 (*$t1$*)*, *ultimo nome* (u)} e o outro {*token de combinação2 (*$t2$*)*,*código postal* (z)}. Considerando o *token* como um atributo criado a partir da *chave de ordenamento* feita pela primeira parte desta conjunção, responsável por permitir o cálculo do fecho transitivo. Desta forma o esquema de blocagem se torna:

$$BS = \{p, n\} \wedge \{t1, u\} \cup \{t2, z\}$$

Este autor julga e demonstra o quanto este método é inclusivo, capaz de maximizar o número de pares verdadeiros mantendo todas as vantagens inerentes ao uso de blocagem. Além disso, sua implementação conta com o uso de algoritmos de aprendizado de máquina para definir os BSs. Em seu trabalho, o autor (FILHO, 2008) chama o modelo descrito acima de *blocagem por predicados*, e o implemnenta usando uma abordagem mais simplista.

[6] Teorema usado no estudo de grafos para encontrar relações e caminhos entre os vértices. No contexto do autor (HERNÁNDEZ; STOLFO, 1998), a idéia é unir os dois conjuntos de registros sem duplicatas.

3.8 SEGURANÇA E ANONIMIDADE

Com a necessidade de submeter bases de dados administrativas à correlação para estudos epidemiológicos surge a preocupação com a segurança das informações privadas assim como tornar anônimo cada indivíduo nestes repositórios. Esse problema pode, em algumas ocasiões, restringir dramaticamente a utilização de dados administrativos para estudos estatísticos ou sociais (QUANTIN; RIANDEY,).

Em seu trabalho, o autor (QUANTIN et al., 1998) defende a aplicação de algoritmos hash[7] em atributos que possam identificar indivíduos antes de usar as bases de dados em rotinas de pareamento que suportarão acompanhamentos e estudos epidemiológicos. Desde então, alguns métodos avançados vem sendo aplicados para tornar anônimos os indivíduos em bases que serão pareadas.

3.8.1 O Uso dos Filtros de Bloom

Em seu artigo, o autor (SCHNELL; BACHTELER; REIHER, 2009) relata a insuficiência do uso de algoritmos criptográficos ou funções HASH para mascarar atributos capazes de identificar um indivíduo. Segundo ele, as ferramentas de *correlação* que usam estes métodos apenas conseguem parear registros com identificadores idênticos e são incapazes de tolerar erros de digitação ou omissão. Para lidar com esse problema, foi proposto o uso de filtros de Bloom (BLOOM, 1970).

Filtros de Bloom foram inicialmente propostos por (BLOOM, 1970) como um novo código hash baseado em *membership* (filiação). Consiste num vetor de bits (inicialmente definidos com valor 0) com tamanho t. Para transformar um conjunto de *strings* $S = x_1, x_2, ..., x_n$ num filtro de Bloom será necessário executar funções hash $h_1, ..., h_2$ em cada um dos elementos de S. As funções hash serão utilizadas para definir quais posições dentro do vetor de tamanho l serão definidas com valor 1. Dessa forma, o conjunto S pode ser partes de um nome, subsequências de *string* ou até mesmo atributos de um registro.

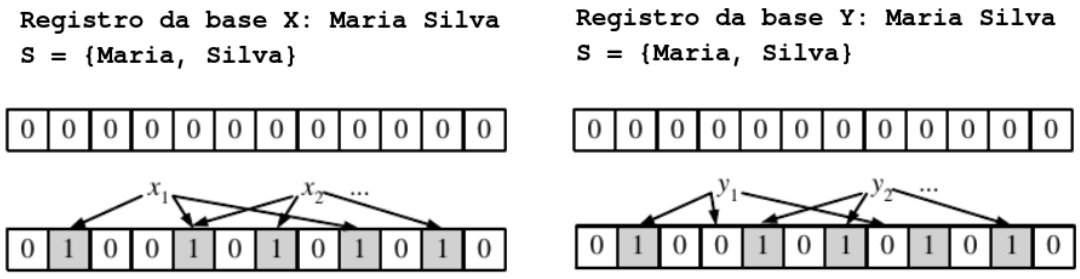

Figura 3.9 Exemplo de aplicação dos filtros de Bloom.

O uso das funções hash garantem um mesmo comportamento para entradas idênticas se o tamanho do filtro não for alterado, como mostrado na Figura 3.9, o que impede a existência de falsos negativos. Além disso, a forma de como aplicar as funções hash deve

[7]Função de via única que transforma qualquer dado de entrada em um resumo de tamanho fixo, impossibilitando o retorno ao dado original ou ataque de dicionário.

contemplar leve diferenças entre filtros de entradas semelhantes, isso garante um grau de similaridade entre pares verdadeiros com algum erro de digitação. Estas propriedades permitem que hajam falsos positivos.

Capítulo

4

Neste capítulo, todo o fluxo de trabalho e implementações da solução de correlação probabilística em Spark é apresentada.

CORRELAÇÃO EM SPARK

4.1 INTRODUÇÃO

O fluxo de trabalho proposto por esta dissertação teve inspiração inicial nos estudos de Antony Peter Stevens, responsável pelo pareamento de diversas bases governamentais através do Ministério da Saúde. Sua expertise e solicitude foram fundamentais para entendimento das bases e do desafio proposto pelo projeto de colaboração. O processo de evolução e melhoria das implementações são constantes, assim como o uso de alternativas para validar cada uma das escolhas feitas.

O cenário imposto pelo projeto descrito no Capítulo 2 o incluiu no contexto de *Big Data* por conta dos requisitos de volume e velocidade no processamento de um imenso volume de dados. Por conta disso, foi eleita uma ferramenta de processamento com alto desempenho capaz de prover abstração na tolerância a falhas, distribuição de tarefas e arquivos em *cluster*, o Apache Spark, melhor detalhado no Capítulo 3. Desta forma, todo um fluxo de trabalho de uma rotina de correlação probabilístico de registros (incluindo suas sub-tarefas abordadas no Capítulo **??**) implementado usando este *framework* para parear as bases governamentais disponibilizadas pela equipe de epidemiologistas e estatísticos do ISC-UFBA é detalhada neste capítulo.

A organização deste capítulo conta, na Seção 4.3, com uma abordagem sobre as iniciativas mais relevantes relacionadas ao nosso contexto sobre correlação encontradas no levantamento do estado da arte. A partir da Seção 4.2 são detalhadas todas as etapas necessárias para a execução do pareamento destas bases administrativas: uma análise descritiva das bases (Seção 4.2.1); pré-processamento dos dados, anomização e blocagem (Seção 4.2.2); teste de similaridade e implementação das rotinas de pareamento (Seção 4.2.3); e por fim, a recuperação do data mart gerado após o pareamento (Seção 4.2.4).

4.2 FLUXO DE TRABALHO DA CORRELAÇÃO EM SPARK

A contribuição principal deste trabalho é a proposta e a implementação de uma ferramenta de pareamento probabilístico de dados no contexto de *Big Data* aplicado na área de saúde. Desta forma, foram sugeridas e desenvolvidas rotinas e tarefas usando o *framework* Spark que pudessem relacionar bases de dados administrativas cedidas pelo Governo Federal através dos Ministério da Saúde e do Desenvolvimento para fomentar o estudo acerca da influência do programa de transferência de renda condicional Bolsa Família em doenças relacionadas a pobreza, sob aprovação dos seus respectivos Comitês de Ética.

Os requisitos principais apresentados pelo projeto, descrito no Capítulo 2, incluíam o desenvolvimento de um sistema que: de forma probabilística, consiga parear registros entre duas ou mais bases; seja capaz de processar uma enorme quantidade de dados em tempo satisfatório; consiga prover confidencialidade aos dados por conta de limitações no acesso as bases nominais imposta pelos orgãos cedentes; atinja um nível de acurácia [1] competitivo ante as principais ferramentas já existentes; seja o suporte para futuros estudos longitudinais; e que permita o avanço e integração de novas funções ou evolução.

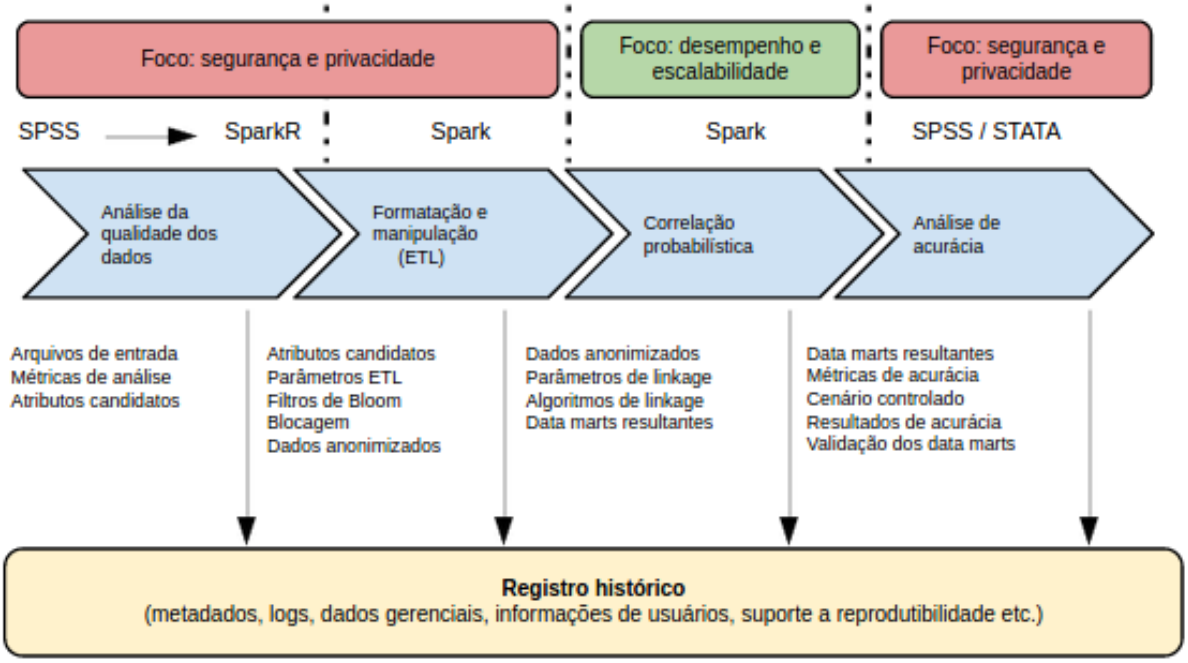

Figura 4.1 Fluxo de execução proposto para Correlação.

A Figura 4.1 ilustra o fluxo de trabalho subdividido em três fases. A primeira fase se refere ao conhecimento acerca das bases e o contexto onde estão inseridas, a relevância da análise descritiva (na Seção 4.2.1) de todas as bases é requisito mínimo para um bom resultado do pareamento. Na segunda fase, são aplicadas as rotinas de normalização (usando

[1] O termo acurácia trata do estudo sobre a exatidão do pareamento, feito através do levantamento das métricas de sensibilidade e especificidade (COUTINHO; COELI, 2006)

codificação UTF-8) e tratamento do texto, responsável por retirar acentos, espaços duplos substituindo todos os caracteres por caixa alta. Ainda nesta fase, é feita a deduplicação usando ferramentas estatísticas capazes de retirar registros repetidos nas bases, a anonimização destes indivíduos através da aplicação dos filtros de Bloom e a blocagem utilizada (ambas na Seção 4.2.2). Na terceira etapa, são feitos os cálculos de similaridade entre os registros e é decidido se o par de registros comparados podem ou não representar um mesmo indivíduo (detalhados na Seção 4.2.3). Por último, na terceira fase (Seção 4.2.4), são recuperados os pares tuplas com maior probabilidade de conterem informações sobre uma mesma pessoa dentro das bases A e B.

Considerando a composição da equipe multidisciplinar envolvida no projeto (cientistas da computação, estatísticos, biólogos e epidemiologistas) e a abrangência de cada uma dessas áreas de estudo, algumas tarefas inerentes a este fluxo foram delegadas a componentes com competências que melhor se relacionam a cada etapa. Em paralelo a este trabalho, existe um estudo aprofundado sobre a aplicação de métodos ETL (Extração, Transformação e Carga) que deram suporte aos resultados alcançados e descritos nesta dissertação. Dessa forma, apesar de haver uma participação ativa de cada participante em todo o *workflow*, o escopo deste texto é documentar a contextualização sobre pareamento de dados, implementação em Spark das rotinas utilizadas, execução e entrega de resultados da etapa de pareamento sobre as bases de dados disponibilizadas.

4.2.1 Análise Descritiva

Como explicado no Capítulo **??**, o *record linkage* é a alternativa ideal para correlacionar fontes de dados que não compartilham de uma chave única de identificação de registros. Desta forma, existe a necessidade de eleger quais os atributos destas bases serão comparados para determinar o nível de similaridade entre duas tuplas distintas. A escolha sobre esses atributos podem ser pela sua coexistência em ambas as bases, capacidade discrimatória, qualidade no preenchimento (indicado pela pouca incidência de *missings*[2]) e integridade de banco de dados.

O objetivo desta etapa no fluxo de trabalho proposto é gerar conhecimento suficiente sobre as bases e cada um de seus contextos para dar suporte às decisões acerca de quais atributos serão utilizados para o pareamento probabilístico dos registros. Essa necessidade se aplica para todo cenário onde se deseja executar as tarefas de correlação sobre duas fontes de dados, pois o entendimento sobre elas irá impactar diretamente na acurácia alcançada. Todas as ferramentas expostas na Seção 4.3 permitem que seu usuário defina os atributos que participarão do procedimento.

Como exposto no Capítulo 2, os pesquisadores obtiveram as bases de dados necessárias para aplicação de seus estudos epidemiológicos. A Tabela 4.1 mostra todas as bases disponibilizadas para as rotinas de pareamento probabilístico de dados. O CadÚnico (Cadastro Único) é uma base que armazena dados socioeconômicos dos indivíduos e suas famílias. As folhas de pagamento do Programa Bolsa Família (PBF) guardam informações sobre cada beneficiário e valor recebido a cada mês. Dados referentes a internação hospitalar, notificação de doença ou agravo por agente de saúde e mortalidade relacionas às doenças

[2]Atributos ausentes por erros de preenchimento ou omissão

Bases de Dados	Descrição	Ano
CadÚnico	Dados socio-econômicos	2007 a 2013
PBF	Folhas de Pagamento do Bolsa Família	2007 a 2013
SIH	Hospitalização	1998 a 2011
SINAN	Notificação de agravo	2000 a 2012
SIM	Mortalidade	2000 a 2012

Tabela 4.1 Bases de dados disponibilizadas para execução da correlação

Tuberculose e Hanseníase estão guardadas nas bases SIH, SINAN e SIM respectivamente. Levando em consideração a abrangência desses repositórios, uma decisão de projeto reduziu o escopo das primeiras atividades, levando em consideração a necessidade de validar a ferramenta de pareamento proposta por este trabalho e a melhor qualidade das bases mais novas. Desta forma, decidiu-se trabalhar inicialmente com as bases do ano de 2011. Esta dissertaąão traz os resultados preliminares do pareamento efetuado entre as bases CadÚnico$x$SIH e CadÚnico$x$SIM.

Bases	Qt. de Registros
CadÚnico	~106 milhões
SIH	~62 mil
SIM	~17 mil

Tabela 4.2 Bases de dados do ano 2011 submetidas a correlação e quantidade de registros de cada uma.

A Tabela 4.2 mostra a quantidade de registros para cada uma das bases que serão estudadas e pareadas. Num primeiro passo, para o *linkage* entre o CadÚnico e o SIH, o resultado da análise acerca do conjunto de atributos comuns as duas bases é exposto na Tabela 4.3, nela pode-se verificar a porcentagem de valores ausentes em cada uma das bases. Com base nestes resultados, foram escolhidos os atributos que se referem aos nomes, data de nascimento e municipios de residência do cidadão, paciente ou falecido registrados nos repositórios.

CadÚnico	
Atributo	Missing %
NOME	0
DT_NASC	0
MUNIC_RES	0
SEXO	0
RG	48,7

SIH	
Atributo	Missing %
NOME	0
NASC	0
MUNIC_RES	0
SEXO	0
LOGR	0,9

SIM	
Atributo	Missing %
NOME	9
NASC	1,2
RES	0

Tabela 4.3 Valores ausentes dos principais atributos de cada base estudada.

Após adquirir conhecimento sobre cada uma das bases e seu contexto, foi possível planejar um método que detectasse e removesse registros duplicados. No CadÚnico, após reunião com técnicos do Ministério da Saúde, confirmou-se a possibilidade de remover todos os duplicados usando uma variável cujo papel descreve um registro como ATIVO ou INATIVO, dessa forma, cidadãos re-cadastrados com mesmo NIS (Número de Identificação Social) por migração ou formação de uma segunda família tiveram seus dados

mais antigos removidos. No SIH, algumas tuplas com N_AIH (Número de Aprovação de Internação Hospitalar) duplicado por conta de renovação do prazo final para recebimento de alta tiveram os registros mais obsoletos apagados. No caso SIM, não haviam dados duplicados para o ano de 2011.

Bases	Qt. de Registros
CadÚnico	~76 milhões
SIH	~56 mil
SIM	~17 mil

Tabela 4.4 Tamanho das bases após a deduplicação.

Após a eleição dos atributos, resultado do trabalho da análise descritiva de cada base envolvida, os dados estão prontos para a próxima fase do fluxo de trabalho proposto. Na próxima seção será detalhada a etapa responsável pelo pré-processamento, anonimização e blocagem dos registros.

4.2.2 Pré-processamento, Blocagem e Privacidade

A fase de preparação dos dados, como mostrado na Figura 4.1, compreende três importantes etapas que antecedem o pareamento. Toda a execução das tarefas desta fase foram implementadas usando o Spark.

O pré-processamento tem como principal objetivo amenizar os problemas gerados na imputação dos registros, desta forma, existe a necessidade de normalizar todas as *strings* que serão usadas no pareamento, alterando-as para caixa alta e retirando a acentuação. Outra preocupação é a manipulação do formato utilizado pelos bancos de dados para atributos como *data de nascimento* (ex.: no CadÚnico o atributo DT_NASC tem formato $aaaa - mm - dd$, já no SIH e SIM o formato é $aaaammdd$).

O Algoritmo 4.1 se refere a implementação do fluxo de dados usando o *framework* Spark. Na *linha 10*, são disparadas todas as transformações que serão feitas no RDD (criado na *linha 8* a partir de uma base original). A função responsável por gerar os filtros de Bloom está descrita no Algoritmo 4.3 e a blocagem foi melhor detalhada no Algoritmo 4.2.

4.2.2.1 Blocagem Segundo (WINKLER, 1985), não existe um método infalível para realizar a blocagem de grandes bases de dados, esta é uma tarefa baseada em tentativa e erro. Desta forma, a decisão inicial sobre a blocagem foi utilizar a blocagem padrão, descrita na Seção 3.7.1. A justificativa principal foi a facilidade na implementação e a impossibilidade de um estado ter sido declarado de forma errada nestas bases, levando em consideração de que as prefeituras são as responsáveis pelo preenchimento de todas elas antes do repasse aos orgão federais. Assim, numa primeira tentativa de reduzir o número total de comparações, foi feita a blocagem por estados que separou em arquivos distintos de cada base registros que tinha mesma Unidade Federativa (UF).

Para alguns blocos, a divisão do banco de dados usando a blocagem por unidade federativa obteve resultados bastante vantajosos. Por exemplo, o processamento dos

Algorithm 4.1 Função Bloom

```
1: Entrada ← BaseOriginal_1.csv
2: PastaDeSaida ← '\home\anon\Base-1\'
3:
4: tamNome ← 50
5: tamNasc ← 40
6: tamCidade ← 20
7:
8: EntradaRDD = sc.textFile(Entrada) ▷ Transformando arquivo de entrada em RDD
9:
10: Resultado = EntradaRDD.cache().map(normaliza).map(chamaBloom).collect()
11:                                          ▷ Transformações e ação feitos ao RDD
12: função NORMALIZA(linha)
13:     atributos ← linha.split(";")
14:     para i ← 1 até len(atributos) faça
15:         atributos[1] ← atributos[1].normalize(UTF8)     ▷ Normalização via Python
16:         linha ← atributos.join';'
17:     fim para
18: devolve linha
19: fim função
20:
21: função CHAMABLOOM(linha)
22:     atributos ← linha.split(";")
23:     a ← bloom(atributos[1], tamNome)
24:     b ← bloom(atributos[2], tamNasc)
25:     c ← bloom(atributos[3], tamCidade)
26:     VetorDeBits ← str(a) + str(b) + str(c)
27:
28:     EscreveEmBloco(atributo[4], PastaDeSaida, atributos[5], VetorDeBits)
29: devolve 0
30: fim função
```

estados de menor densidade (mostrados na Tabela 4.4) como o Amapá levou em torno 96 segundos de execução do pareamento, usando o Spark. Porém, em estados com maior participação, como São Paulo e Bahia, levaram dias para ter sua execução concluída. Estas métricas para blocos muito grandes tornaram impraticável o uso da blocagem padrão por estados.

Como alternativa, foi implementada a blocagem padrão usando o atributo "MUNIC_RES"para as bases do CadÚnico e SIH, e "RES"do SIM. Como resultado, foram obtidos blocos de tamanhos razoáveis e gerando métricas de tempo de execução promissores. Além disso, esta estratégia garantia a inexistência de erros no repositório com informações socio-econômicas, já que as prefeituras são as responsáveis pela imputação dos dados.

Estado	CadÚnico Tam. (%)	SIH Tam. (%)	Estado	CadÚnico Tam. (%)	SIH Tam. (%)
Rondônia	0,8	1,1	Sergipe	1,7	0,5
Acre	0,5	0,2	Bahia	11,0	4,6
Amazonas	0,2	2	Minas Gerais	9,7	6,4
Roraima	0,4	0,3	Esp. Santo	1,8	1,4
Amapá	0,4	0,2	R. de Janeiro	4,9	11,2
Tocantins	1,0	0,6	São Paulo	13,0	25,4
Maranhão	6,1	1,6	Paraná	5,2	3,5
Piauí	3,2	0,9	St. Catarina	1,9	4,8
Ceará	8,0	3,5	R. G. do Sul	4,2	11,8
Rio G. do Norte	2,6	1,7	Mt. Gr. Sul	2,2	1,5
Paraíba	3,5	3,6	Mato Grosso	2,3	0,8
Pernambuco	7,3	7,8	Goiás	3,2	2,9
Alagoas	3,1	1,2	Dist. Federal	1,8	0,5

Tabela 4.5 Distribuição de registros em blocagem por estados.

	CadÚnico	SIH	SIM
Mínimo	13	1	1
Média	13.638,4	18,9	6,9
Mediana	6122,5	124,3	39,8
Máximo	1.662.057	3771	1148
Desvio Padrão	42.996,5	124,3	39,8
Qt. total de blocos	5566	2953	2422

Tabela 4.6 Análise descritiva dos blocos por cidades.

Ao analisar o A Tabela 4.6 e o gráfico da Figura 4.2, pode-se ter a idéia de como aconteceu a distribuição de blocos. O CadÚnico abrange a maior parte das cidades enquanto as demais bases contém um número menor de blocos, isso já representa uma grande quantidade de dados que sequer serão comparados.

Utilizando o Spark, uma nova métrica passou a participar da escolha sobre o método ideal de blocagem, além da diminuição do número total de comparações e manter a inclusão de pares que devem ser comparados, também será preciso garantir que os blocos sejam de tamanhos razoáveis para que as tarefas em *cluster* utilizem de forma otimizada todos os recursos de processamento e memória. Observando a quantidade de blocos com baixo número de registros nas bases menores, é possível inferir que não haverá um uso pleno dos recursos de paralelização, o que pode requisitar a adoção de novas abordagens de blocagem em trabalhos futuros.

Algorithm 4.2 Função EscreveEmBloco

```
1: função escreveEmBloco(chave, PastaDeSaida, atributo, VetorDeBits)
2:     ArquivoDeSaida = str(PastaDeSaida) + str(atributo) + .bloom
3:     escrita ← open(outputFile, a) ▷ Cria ou abre um arquivo adicionando mudanças.
4:     escrita.insere(VetorDeBits)
5: fim função
```

O Algoritmo 4.2 mostra a implementação simples de como se criam os blocos para cada

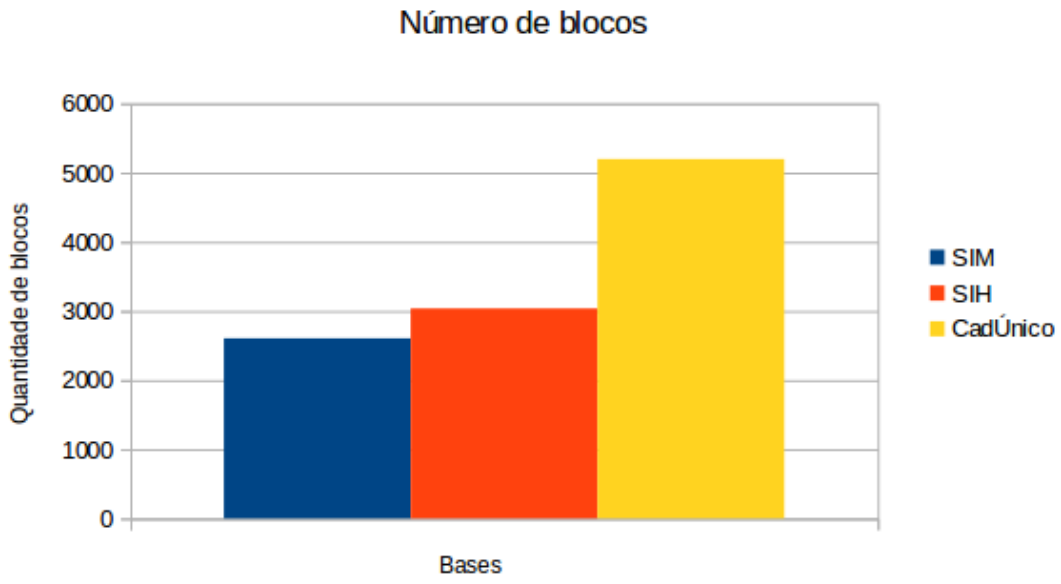

Figura 4.2 Quantidade de blocos/cidades em cada base.

base em arquivos separados. Registros com mesmos atributos de blocagem serão sempre escritos em um mesmo arquivo. Os estados ou cidades (ambos usados em algum momento como atributo de blocagem) são representados em todos os bancos pelo seu código junto ao Instituto Brasileiro de Geografia e Estatística (IBGE), e são desconsiderados os erros de preenchimento, visto sua improbabilidade por conta do processo de imputação nas bases.

4.2.2.2 Privacidade A privacidade é alcançada através da aplicação de um método semelhante ao proposto por (SCHNELL; BACHTELER; REIHER, 2009). O uso de *bigramas* (subsequências de *strings* compostas por dois caracteres) para gerar os filtros de Bloom (BLOOM, 1970) proporcionou além da anonimização, a possibilidade de ocasionar leves diferenças no filtro quando houver erro em algum dos atributos utilizados para o pareamento. Outra vantagem do uso do vetor de bits é a possibilidade de atribuir pesos aos atributos mais significativos de cada tupla.

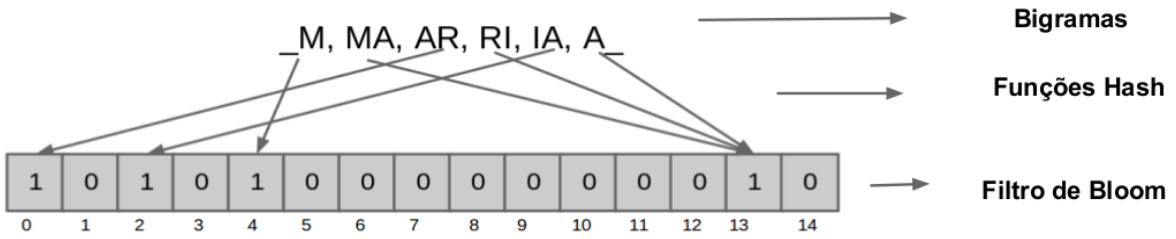

Figura 4.3 Uso dos bigramas para gerar filtros de Bloom.

A Figura 4.3 ilustra como funciona a construção de um filtro de Bloom de 14 posições

através dos bigramas extraídos de um nome. O funcionamento deste método também é mostrado no Algoritmo 4.3, como pode ser notado, cada atributo envolvido no pareamento será responsável por compôr uma porção do vetor de bits resultante. O uso de funções hash e matemática discreta para encontrar a posição no filtro que se tornará "1"torna improvável a inferência do valor original do registro a partir do vetor de bits. Estes mesmos fatores garantem que esta implementação se comportará de forma idêntica para entradas iguais, assim como o uso dos bigramas ocasionam apenas "pequenas diferenças"nos filtros gerados por registros similares.

Algorithm 4.3 Função Bloom

```
1:  função BLOOM(registro, tamanho_vetor)
2:      registro ← '_' + registro + '_'          ▷ Inclui espaços no início e fim.
3:      vetor ← []                                ▷ Inicia vetor vazio
4:      para i ← 1 até tamanho_vetor faça
5:          vetor.append(0)
6:      fim para
7:      para i ← 1 até len(registro)-1 faça
8:          A1 ← md5(registro[i] + registro[i + 1])   ▷ Aplica função hash em bigramas
9:          A2 ← int(A) mod tamanho                   ▷ Retorna o resto da divisão.
10:         vetor[A2] ← 1
11:     fim para
12: fim função
```

Outra característica da aplicação dos filtros de Bloom, que pode ser visto a partir da *linha 8* do Algoritmo 4.3, é a possibilidade de utilizar mais de uma função de hash repetidas vezes. Isso fortalece a privacidade do vetor de bits, dificultando ataques de força bruta ou dicionário na tentativa de obter os registros em texto claro.

Estudos feitos por componentes do projeto responsáveis por aplicar métodos ETL (Extração, Transformação e Carga de registros) e pré-processamento dos dados revelaram uma forte influência do tamanho do filtro de Bloom, assim como o peso dado a cada variável na composição do vetor de bits, na sensibilidade resultante do método proposto por esta dissertação. Neste experimento, foram construídas duas bases A e B com 50 e 20 registros, respectivamente, impostas a três cenários: No Cenário 1, nenhum dos indivíduos deveriam ser pareados, utilizando o Correlação em Spark; no Cenário 2, foram incluídos apenas 5 indivíduos idênticos em ambas as bases; e no Cenário 3, foi incluído um erro em cada um dos 5 indivíduos que deveriam ser pareados no Cenário 2. Os resultados mostrados na Tabela 4.7 mostram melhores métricas para filtros de Bloom compostos de 50 bits reservados para a variável $NOME$, 40 bits para $DATA\_NASC$ e 20 para $MUNIC\_RES$ (atributos usados no estudo). Apesar de alguns outros filtros obterem resultados semelhantes, sabe-se que quanto maior o filtro, maior também será a sensibilidade a erros. Portanto, essa escolha tende a prevenir um alto número de falsos positivos entre os pares encontrados.

Após esta etapa, os dados estão aptos a serem entregues à tarefa mais custosa (em termos computacionais). Os blocos das bases com registros anonimizados agora podem ser

Tamanho do Vetor e distribuição dos pesos	**Cenário 1** Nenhum par esperado		**Cenário 2** 5 pares esperados		**Cenário 3** 5 pares esperados*	
	Pares esperados	Pares encontrados	Pares esperados	Pares encontrados	Pares esperados	Pares encontrados
20x20x20	0	310	5	347	5	348
30x30x30	0	29	5	41	5	42
40x40x40	0	11	5	17	5	16
50x50x50	0	0	5	5	5	5
50x50x40	0	0	5	5	5	5
50x40x40	0	0	5	5	5	5
50x40x30	0	2	5	6	5	6
50x40x20	0	0	5	5	5	5

Tabela 4.7 Peso de cada atributo no filtro de Bloom.

processados em uma máquina ou *cluster* com alta capacidade de processamento. Caso não haja recurso suficiente para executar isso numa infraestrutura própria, é possível encaminhar o resultado deste fluxo para ativos de terceiros contando com a segurança provida pelo uso dos métodos descritos até aqui.

4.2.3 Correlação Probabilística

A etapa de pareamento é responsável por executar o teste de similaridade entre cada par de registro contidos em dois blocos de bases distintas e decidir acerca de seu pareamento. O produto cartesiano gerado por esta etapa aumenta, de forma absurda, o custo de sua execução. Por conta disso, cada uma das fases descritas até aqui tem importante papel para garantir seu sucesso.

Dessa forma, para cada bloco com mesmo atributo de blocagem (código de município de residência) será aplicada uma verificação de similaridade baseada no cálculo do Índice de Sorensen/Dice (DICE, 1945), como exposto na Seção 3.6.6. A escolha deste coeficiente é justificada por ela ser uma das únicas, além do proposto por Jaccard (REAL; VARGAS, 1996), capazes de verificar a semelhança entre vetores de bits. Além disso, a aplicação deste método foi inspirado pelo trabalho exposto por (SCHNELL; BACHTELER; REIHER, 2009), que também já vinha sendo praticado em algumas demandas de pareamento probabilístico pelo Ministério da Saúde.

A primeira parte do pareamento probabilístico é descrito no Algoritmo 4.4. A maior preocupação é encontrar um par de blocos de bases distintas que segue o mesmo atributo de blocagem, garantido na *linha 6*. A estratégia de se utilizar a base menor numa variável *broadcast* e definir o fluxo de execução de dados a partir do bloco maior foi aprovada pelos desenvolvedores do Spark num contato acerca de algumas dúvidas de implementação, e é responsável por diminuir os custos de entrada e saída de *hardware*, comunicação entre processos e mapeamento de registros em cada *thread* envolvida. Um arquivo de resultado será gerado para cada par de blocos, dessa forma, foi necessário definir qual o diretório e nome este arquivo receberá.

Na segunda parte do pareamento, para cada par de linhas dos blocos selecionados é feito um mapeamento de todas as suas posições com a finalidade de encontrar 1s em

Algorithm 4.4 Implementação do pareamento - preparação

```
1: dirBlocos1 ← '\temp\blocos1\'
2: dirBlocos2 ← '\temp\blocos2\'
3: dirResult ← '\temp\result\'
4:
5: para blocoMenor ← 1 até len(dirBlocos2) faça
6:     se existe blocoMaior = blocoMenor em dirBlocos1 então
7:         baseMenor ← sc.textFile(blocoMenor)
8:         baseMaior ← sc.textFile(blocoMaior)
9:         fileResult ← str(dirResult + blocoMenor)
10:
11:        var ← baseMenor.cache().collect()
12:        varbc ← sc.broadcast(var)
13:
14:        pareados = baseMaior.cache().map(compare).collect()
15:    fim se
16: fim para
```

posições de vetores, coincidentes ou não. Essa tarefa é feita para preencher a fórmula usada para calcular o coeficiente de Sorensen. Em caso de vetores completamente iguais, o resultado da variável *dice* será 1.0, e caso haja alguma diferença entre eles, este valor irá decrescer.

A variável de corte pode ser configurada de forma arbitrária considerando que ainda não houve um estudo aprofundada sobre a acurácia do método para definir qual o valor otimizado para ela. Dessa forma, o valor foi fixado em $0,87$ apenas para garantir que nenhum verdadeiro positivo seja perdido neste pareamento. A aplicação de um método similar ao empregado no RecLink (JUNGER, 2006) para inferir qual a melhor definição para conseguir uma melhor acurácia pode ser empregado nesta etapa, porém, o foco deste trabalho se mantém na implementação de um método de pareamento probabilístico de dados usando o framework Apache Spark aplicado ao projeto de análise de impacto do Programa Bolsa Família em doenças relacionadas a pobreza.

4.2.4 Geração do Data Mart

Como foi visto no Algoritmo 4.2, as linhas que compõem cada bloco carregam consigo uma chave de identificação unívoca de cada uma das bases. Essas chaves são utilizadas na etapa de geração de data smart, que recupera os registros pareados de todos os blocos para uma única base. Este é o produto de todo o fluxo de trabalho responsável pela correlação exposta nesta dissertação e é entregue à equipe de estatísticos para medir a acurácia do resultado e em seguida para os epidemiologistas e biólogos para estudos e análises de interesse.

A estratégia de manter essas chaves é importante, pois permite que a recuperação de dados seja feita por uma tarefa Spark ou até mesmo através de buscas num SGBD.

Algorithm 4.5 Implementação do pareamento - comparação e decisão

```
1: função COMPARE(linha)
2:     para posix1 ← 1 até len(varbc.value) faça
3:         h ← 0                                   ▷ contador de 1s na mesma posição
4:         a ← 0                                   ▷ contador de 1s no vetor do blocoMaior
5:         b ← 0                                   ▷ contador de 1s no vetor do blocoMenor
6:         corte ← 0.87                            ▷ ponto de corte para decisão de pareamento
7:
8:         para posix2 ← 1 até len(linha) faça
9:             se varbc.value[posix1] = 1 então
10:                b ← b + 1
11:            fim se
12:            se linha[posix1] = 1 então
13:                a ← a + 1
14:            fim se
15:            se varbc.value[posix1] = linha[posix2] então
16:                h ← h + 1
17:            fim se
18:
19:            dice ← (2 ∗ h)/(a + b)
20:
21:            se dice > corte então                ▷ onde acontece a decisão.
22:                fileResult.escreve(varbc.value[posix1], linha[posix2])
23:            fim se
24:        fim para
25:
26:    fim para
27: fim função
```

De qualquer forma, ainda existe a preocupação em anonimizar também estas variáveis para evitar qualquer possibilidade de identificação de um indivíduo através das bases. No Capítulo 5 serão apresentados os resultados preliminares das execuções do pareamento proposto assim como uma análise comparativa entre ele e outras propostas descritas na Seção 4.3.

Após os primeiros pareamentos entre o CadÚnico e o SIH, foram percebidas muitas duplicatas entre os registros pareados de ambas as bases. Isso devido ao fato de que havia uma alta similaridade entre um indivíduo do CadÚnico com vários do SIH, e vice-versa. Entre pouco mais de 427 mil pares encontrados pelo pareamento, haviam mais de 162 mil registros participando de um ou mais pares de cada uma das bases. Esses dados justificaram a inclusão de uma etapa de deduplicação na fase de geração de data mart, esta tarefa foi feita pelos estatísticos do grupo mantendo os pares de maior coeficiente de similaridade em caso de duplicata.

4.3 TRABALHOS RELACIONADOS

Nesta seção serão apresentadas as principais soluções de pareamento probabilístico encontradas na literatura. O objetivo é apresentá-las mostrando seus prós e contras antes de expôr com mais detalhes a Correlação em Spark proposto para que se possa compará-los no Capítulo 5.

4.3.1 German RLC

O Centro de *Record Linkage* Alemão (RLC, 2011) tem como objetivo principal promover a pesquisa em pareamento de dados oferencendo ferramentas práticas abertas para uso acadêmico. Suas publicações mais relevantes incluem a proposta de um fluxo de pareamento capaz de prover a preservação da privacidade dos dados através dos Filtros de Bloom (SCHNELL; BACHTELER; REIHER, 2009), o uso de *multibit trees* para blocagem e checagem de similaridade (BACHTELER; REIHER; SCHNELL, 2013a), e a aplicação destas ferramentas em estudos de caso (SCHNELL, 2013) (ANTONI, 2013). As principais soluções disponibilizadas por este centro estão disponíveis em (RLC, 2011), são elas:

MTB O *Merge Toolbox* (SCHNELL; BACHTELER; REIHER, 2005) é um programa capaz de prover deduplicação e pareamento probabilísticos de dados usando a técnica de similaridade baseada na Distância de Edição, melhor descrita no Capítulo **??**.

Safelink O *Safelink* (SCHNELL; BACHTELER; REIHER, 2007) introduz um protocolo capaz de gerar filtros de Bloom gerados a partir de bases ou atributos de entrada.

TDGen O *Test Data Generator* (BACHTELER; REIHER, 2012) permite a criação de dados genéricos incluindo erros para validação de métodos de correlação.

Muitos conceitos introduzidos por este grupo se tornaram inspirações para equipes ou indivíduos responsáveis pelo pareamento probabilístico de dados dentro do Ministério da Saúde brasileiro. Neste trabalho, algumas etapas do fluxo de execução do pareamento são baseadas nas estratégias introduzidas pelas publicações citadas acima.

4.3.2 RecLink

Iniciativa brasileira usada para pareamento probabilístico de bases de dados administrativas de alguns orgãos públicos, o RecLink (JR; COELI, 2000) é uma aplicação com interface interativa que permite, de forma flexível, que o usuário manipule as regras da correlação. Em seu método, estão incluídas: a etapa de blocagem baseada em SOUNDEX (PFEIFER et al., 1995); e o teste de similaridade que pode ser feito de forma rigorosa (para registros idênticos), através de sequências de caracteres e comparação aproximada de atributos.

Algumas extensões a este aplicativo foram propostas em (JUNGER, 2006) para uso do algoritmo EM (expectation-maximisation) para estimar quais os parâmetros devem ser utilizados na configuração do pareamento probabilístico de bancos de dados. Outra versão

do RecLink (JUNIOR; COELI, 2006) incluiu no programa a possibilidade de utilizar a técnica de associação probabilística de registros.

4.3.3 FRIL

O *Fine-grained record integration and linkage tool* (JURCZYK et al., 2008) é uma ferramenta *open-source* que permite o pareamento de dados de forma rápida e fácil. Os parâmetros e regras para melhora sobre os resultados do pareamento são permitidos aos seus usuários através de um conjunto de definições sitemáticas que podem ser feitas de forma interativa.

A utilização do método de vizinhança ordenada para busca e blocagem de registros e a possibilidade de ser executado em arquiteturas que suportam *multicores* ou *multithreading* são ótimas vantagens dessa ferramenta, além dos benefícios do software livre.

4.3.4 Dedoop

Apesar de não ser uma ferramenta voltada a integração de dados, o Dedoop (KOLB; THOR; RAHM, 2012) foi uma das poucas alternativas encontradas na literatura capaz de lidar com bases de dados no contexto de *Big Data*, baseada no paradigma *MapReduce*, desenvolvida em *Hadoop*. Em seu fluxo de trabalho, o *Dedoop* inclui blocagem, pareamento e o uso de aprendizado de máquina para encontrar duplicatas em uma base de dados.

Capítulo 5

Os resultados acerca da comparação da solução proposta por esta dissertação e alguns trabalhos relacionados, assim como a apresentação de métricas de tempo de execução são feitas neste Capítulo

RESULTADOS

Como foi comentado no Capítulo 4, já foram executadas algumas rotinas de pareamento probabilístico nas bases com informações socioeconômicas (CadÚnico), acerca das internações hospitalares (SIH) e mortalidade (SIM) das doenças relacionadas a pobreza, Hanseníase e Tuberculose.

Sendo o estudo sobre acurácia do método proposto neste trabalho uma etapa que sucede a entrega dos resultados do pareamento, os dados que podem ser apresentados como produto desta solução se assemelham aos apresentados na Seção 5.3, onde diferentes ferramentas foram comparadas. Porém, uma verificação acerca da presença de falsos positivos ou falsos negativos seria bastante onerosa e impraticável, limitando a apresentação de resultados a distribuição dos coeficientes de similaridade para os pares encontrados. Espera-se dos resultados dos estudos sobre a acurácia a definição de um ponto de corte que maximize a presença de verdadeiros positivos em detrimento de falsos positivos ou falsos negativos e em paralelo um intenso processo de *feedback* e redesenho ou extensão da Correlação em Spark.

Inicialmente, serão apresentados os resultados dos pareamentos probabilístico entre CadÚnico*x*SIH (Seção 5.1.1 e CadÚnico*x*SIM (Seção 5.1.3). Algumas métricas sobre os tempos de execução alcançados, além de um breve comparativo com a implementação *multicore* serão apresentadas na Seção 5.2.

5.1 PRIMEIROS PAREAMENTOS

Nesta seção serão apresentados os resultados para os três primeiros pareamentos entre bases de dados distintas. O primeiro, mais básico, foi feito entre todos os registros do CadÚnico e SIH[1]. O segundo usou um recorte na base de internação hospitalar que representasse os indivíduos com Tuberculose como diagnóstico principal para ser *linkado*

[1]Com todas as informações disponibilizadas ao ISC-UFBA, com dados sobre a hospitalização por HIV, Hanseníase e Tuberculose.

com o CadÚnico. Os óbitos causados por essa mesma doença presentes no SIM foram usados no terceiro pareamento com a base de dados socio-econômicos.

5.1.1 Pareamento 1: CadÚnico x SIH (completo)

Este primeiro pareamento foi feito com objetivo de iniciar um *feedback* preliminar sobre a qualidade do método e resolver problemas pontuais na implementação. As informações sobre a blocagem por cidades das duas bases envolvidas podem ser verificadas no gráfico da Figura 4.2 e na Tabela 4.6.

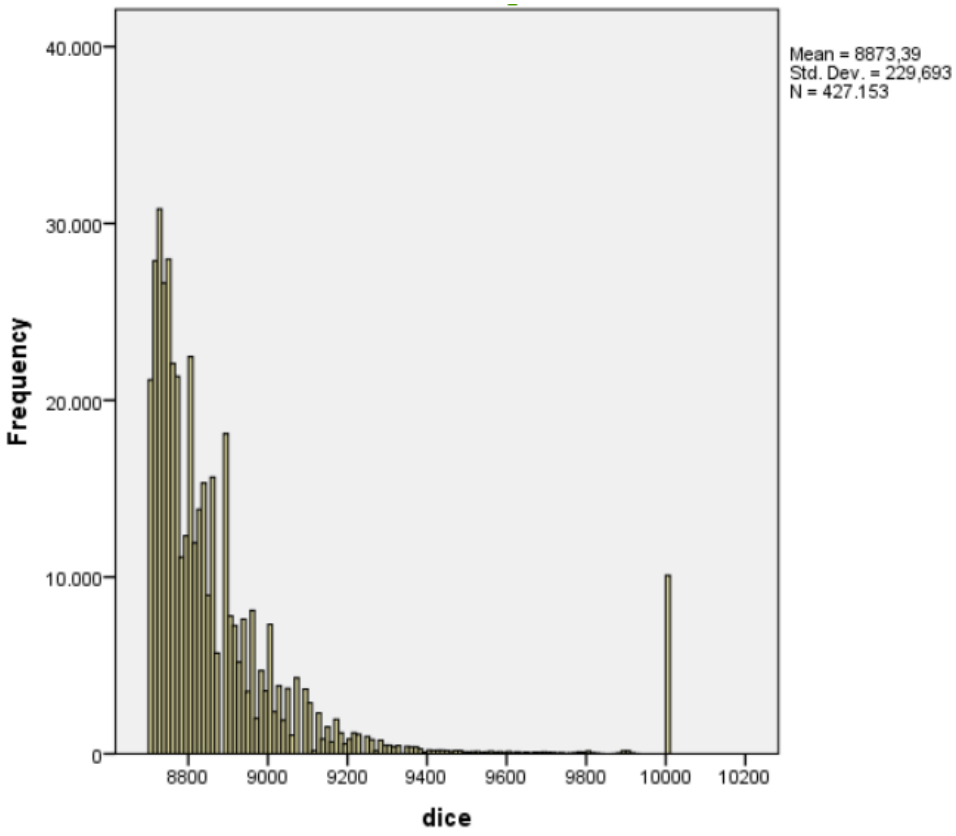

Figura 5.1 Distribuição da similaridade entre os pares - CadÚnico x SIH (completo).

O histograma presente na Figura 5.1 mostra a distribuição entre os graus de similaridade encontrados entre os pares. Pouco mais que 427 mil registros foram pareados com coeficiente de Dice maiores que 0.87 (representados como 8.700 pelo algoritmo). Abaixo, a Tabela 5.1 mostra a distribuição absoluta e proporcional desses registros entre faixas de similaridade.

Não foi executado nenhum procedimento de deduplicação na base que resultou deste pareamento, mas análises superficiais encontraram pouco mais que 162 mil registros participando de mais de um par dentro do *data mart* gerado. Naturalmente, a probabilidade dos pares presentes entre as faixas mais altas serem verdadeiros é cada vez maior. Da mesma forma, a ocorrência de falsos positivos aumenta de acordo com a diminuição do

Faixa	Quantidade	%
Dice igual a 10000	10089	2,4
9700-9999	651	0,2
9600-9799	1277	0,3
9400-9599	2502	0,6
9200-9399	9273	2,2
9000-9199	40252	9,4
8700-8999	363109	85,0

Tabela 5.1 Distribuição de registros no Pareamento 1.

coeficiente Dice.

5.1.2 Pareamento 2: CadÚnico x SIH (Tuberculose)

Para possibilitar análises preliminares acerca da ocorrência da Tuberculose entre indivíduos do CadÚnico, foi feito um pareamento entre esta base e um recorte do SIH contendo apenas os hospitalizados que tiveram a referida doença como diagnóstico principal. Dessa forma, uma base de 14824 dos 56 mil registros do SIH foi pareada com o repositório de aproximadamente 76 milhões de indivíduos do CadÚnico.

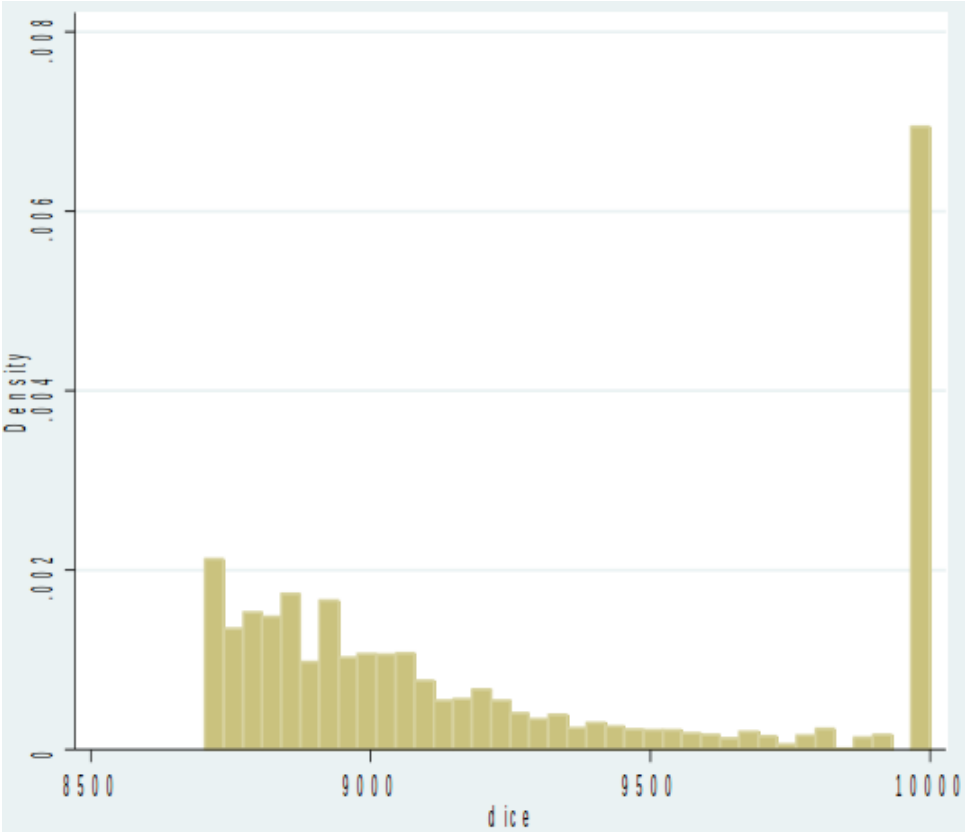

Figura 5.2 Distribuição da similaridade entre os pares - CadÚnico x SIH.

O histograma ilustrado na Figura 5.2 mostra como estão distribuídos os pares de

maior similaridade. Foram encontrados 7827 pares únicos, distribuídos entre as faixas de similaridade expostas na Tabela 5.2. Os registros com coeficiente de similaridade Dice máximo certamente são verdadeiros positivos, espera-se um aumento suave na incidência de falsos positivos de acordo com a diminuição da similaridade, fazendo necessário o estudo e eleição de um ponto de corte que potencialize a presença de pares verdadeiros.

Faixa	**Quantidade**	**%**
Dice igual a 10000	1854	23,7
9700-9999	137	1,8
9600-9799	224	2,9
9400-9599	357	4,6
9200-9399	629	8,0
9000-9199	1271	16,2
8700-8999	3355	42,9

Tabela 5.2 Distribuição de registros no Pareamento 2.

Numa breve comparação entre este e o Pareamento 1 pode-se atribuir à deduplicação feita após a obtenção do *data mart* a responsabilidade pela diminuição de diversos falsos positivos e uma atenuação na distribuição de registros com índices de Dice mais baixos.

5.1.3 Pareamento 3: CadÚnico x SIM (Tuberculose)

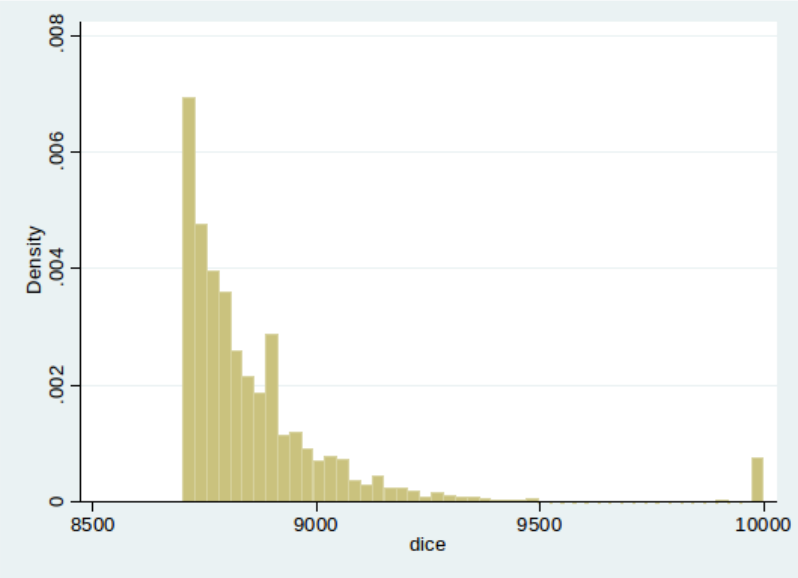

Figura 5.3 Distribuição da similaridade entre os pares - CadÚnico x SIM.

A correlação do repositório de dados sobre a mortalidade por Tuberculose com o CadÚnico foi a demanda mais recente feita pela equipe de epidemiologistas com intuito de permitir estudos preliminares sobre a mobimortalidade e avançar com o pareamento entre todas as bases disponíveis com informações socioeconômicas. Assim como no Pareamento

2, foi feito um recorte dos óbitos de 2011 cuja causa principal tenha sido a Tuberculose, resultando num substrato contendo quase 17 mil registros. Por outro lado, todos os registros sem duplicação do CadÚnico foram utilizados, totalizando aproximadamente 76 milhões.

O histograma ilustrado na Figura 5.3 reflete a distribuição de similaridade encontrado no Pareamento 3, mostrado na Tabela 5.3. Dada a natureza desta base de dados, considerando que no momento do óbito nem sempre é possível uma imputação plena dos dados por conta da carência de informações em muitos casos (por exemplo, um óbito de indigente ou recém-nascido poderá gerar atributos não preenchidos).

O reflexo da má qualidade da base de mortalidade pode ser vista na péssima distribuição do coeficiente de Dice maiores que $8,700$. Neste repositório, apenas 243 observações (0,29% dos registros) não continham nenhum *missing*. Além disso, o número de tuplas duplicadas para o SIM foi 74.378 e para o CadÚnico 5.222, o que representa um número muito grande, representando 91,3% do *data mart* gerado pelo pareamento.

Faixa	**Quantidade**	**%**
Dice igual a 10000	1656	2,03
9700-9999	118	0,14
9600-9799	214	0.26
8700-9599	79485	97,55

Tabela 5.3 Distribuição de registros no Pareamento 3.

Os problemas observados neste pareamento evidenciam a necessidade de haver uma tratativa distinta para cada cenário imposto. Desta forma, cabe um planejamento com diferentes estratégias possibilitadas no fluxo de trabalho que possa amenizar cada erro existente na base original (uso de outros atributos para o pareamento, melhor configuração da construção dos filtros de Bloom, melhor distribuição dos pesos, melhor tratamento dos atributos ausentes, etc.)

5.2 TEMPOS DE EXECUÇÃO

Sendo o Apache Spark um framework de processamento distribuído em *cluster*, é natural que a escalabilidade seja uma das suas principais vantagens. Portanto, sabe-se que quanto maior o número de ativos de processamento e memória estão envolvidos na execução de uma tarefa Spark, melhores serão suas métricas de desempenho. Porém, para que a distribuição das tarefas no aglomerado de máquinas façam pleno uso dos recursos disponíveis, é necessário que a demanda seja grande o suficiente para garantir que cada núcleo de processador esteja responsável por pelo menos quatro tarefas. Dessa forma, quanto maior a demanda ou quantidade de dados a serem processados melhor será o desempenho alcançado pelo Spark numa determinada execução.

Nesta etapa, o aglomerado com uma máquina que possui 8 pro- cessadores Intel Xeon E74820, 16 núcleos, 64 *threads*, 126 GB de RAM conectada a um *storage* com discos capazes de armazenar mais que 10 TB utilizando protocolo NFS para acesso remoto aos dados será chamado de $CLUSTER$. Um ativo com um processador Intel Core i5 com 4 núcleos, 16 threads e 4 GB de memória receberá o nome $MAQUINA1$. A $MAQUINA3$

possui um processador Intel Core i7 com 4 núcleos, 32 threads e 4 GB de memória. As métricas de tempo serão comparadas segundo a quantidade de dados processados e o recurso computacional utilizado no processamento.

Tamanho das bases (registros)	**Tempo execução em MAQUINA1**	**Tempo de execução em CLUSTER**
1 x 1 milhão	93,49s	299s
10 x 1 milhão	395,87s	305s
100 x 1 milhão	2044,78s	477s

Tabela 5.4 Comparativo entre tempo de execução em máquinas distintas para diferentes tamanhos de bases

As métricas mostradas na Tabela 5.4 foram obtidas a partir da execução do pareamento baseado em Spark (descritos pelos Algoritmos 4.4 e 4.5) em bases de dados fictícias com registros anonimizados em filtros de Bloom de mesmo tamanho que o sugerido pelo estudo na Seção 4.2.2.2. Os resultados sobre pares encontrados foram ignorados. Apreciando os tempos de execução para cada um dos tamanhos das bases envolvidas, é possível observar que as vantagens no uso do Apache Spark são melhor exploradas em demandas maiores, principalmente em ativos com mais recursos para efetuar o processamento dessas tarefas.

Para demonstrar como o número de comparações influencia no tempo total de execução, foram retirados três blocos das base de dados separada por estados. O *sample A* foi construído uma quantidade de vetores fictício de tamanho similar ao bloco do Amapá, outro bloco com tamanho similar ao de Sergipe das duas bases estão no *sample B*, já o *sample C* contém os uma mesma quantidade de filtros do bloco de Tocantins. Os dados mostrados na Tabela 5.5 refletem como o número total de comparações influencia no tempo de execução alcançado, todas as amostras foram processadas no $CLUSTER$.

Nome do Sample	**Tamanho (registros) CadÚnico x SIH**	**Comparações (milhões)**	**Tempo de execução**
A	367,892 x 147	~54	96,26s
B	1,6 milhão x ~171	~289,5	479s
C	1,02 mi x 389	~397,6	656,79s

Tabela 5.5 Tempos de execução orientados ao número de comparações do pareamento.

Uma implementação similar do método proposto por este trabalho utilizando a linguagem de processamento *multithreading* de memória compartilhada OpenMP (DAGUM; MENON, 1998) foi utilizada para efetuar o pareamento do sample A em diferentes máquinas para que se pudesse comparar os tempos de execução alcançados para cada uma das ferramentas.

Apesar da Tabela 5.6 trazer resultados favoráveis ao OpenMP para todas as máquinas utilizadas no experimento, o Spark ainda se mantém como principal alternativa quando existem os requisitos de fácil manutenção do código, distribuição de tarefas em diver-

Ferramentas	MAQUINA1	MAQUINA2	CLUSTER
Spark	507.5 s	235.7 s	96.26 s
OpenMP	104.9 s	65.5 s	13.36 s

Tabela 5.6 Comparação entre tempos de execução para implementações em Spark e OpenMP no processamento do Sample *A*.

sas máquinas e tolerância a falhas. Ainda assim, o OpemMP pode vir a ser uma boa alternativa no pareamento de bases pequenas.

5.2.1 Métricas de Tempo no Fluxo de Trabalho

Os tempos de execução alcançados em cada etapa do fluxo de trabalho tiveram influência direta do número de registros em cada par de blocos que passaram pelo pareamento. A Tabela 5.7 mostra as métricas de tempo alcançadas para as bases mostradas na Tabela 4.4 usando a blocagem descrita na Seção 4.2.2.1 para o SIH e CadÚnico. Considerando o tamanho médio dos blocos e a execução das tarefas no $CLUSTER$, pode-se perceber que para grande parte dos blocos houve um bom uso dos recursos, pois haviam registros suficientes de indivíduos por cidades para paralelizar as tarefas de pareamento entre os 64 da máquina.

<table>
<tr><th></th><th>CadÚnico</th><th>SIH</th></tr>
<tr><td>Tamanho (registros)</td><td>~76 milhões</td><td>~57 mil</td></tr>
<tr><td>Normalização (UTF8)</td><td rowspan="3">2310,4 s</td><td rowspan="3">36,5 s</td></tr>
<tr><td>Anonimização (Bloom)</td></tr>
<tr><td>Blocagem (cidades)</td></tr>
<tr><td>Pareamento</td><td colspan="2">9,03 h</td></tr>
<tr><td>Geração do Data Mart</td><td colspan="2">1,31 h</td></tr>
</table>

Tabela 5.7 Tempos de execução para fluxo de trabalho sobre bases absolutas.

As métricas alcançadas na correlação entre o CadÚnico e cada uma das bases de saúde também tiveram tempos satisfatórios. Para que isso fosse possível, foi necessária uma alteração no algoritmo de correlação que verificasse qual o maior dos arquivos que passariam pelo pareamento para garantir uma melhor distribuição das tarefas. Esta alteração teve suma importância considerando que bases que geram blocos muito pequenos (principalmente com menos registros que núcleos disponíveis) não gozam do paralelismo de forma plena.

A preocupação sobre a quantidade de registros em blocos pequenos reforça a propriedade inerente a solução proposta e implementada neste trabalho de atender ao processamento de grandes bases de dados, no contexto de *Big Data*. Desta forma, abordagens *multithreading* podem ser consideradas como alternativa para bases pequenas que não requisitem escalabilidade, distribuição de tarefas em cluster ou tolerância à falhas.

	CadÚnico x SIH_TB		**CadÚnico x SIM_TB**	
Tamanho (registros)	~76 milhões	~15 mil	~76 milhões	~17 mil
Normalização (UTF8)	2310,4 s	7,9 s	2310,4 s	8,3 s
Anonimização (Bloom)				
Blocagem (cidades)				
Pareamento	2,9 h		3,6 h	
Geração do Data Mart	1268,98 s		456 s	

Tabela 5.8 Tempos de execução alcançados para bases recortadas.

5.3 COMPARATIVO ENTRE FERRAMENTAS DE PAREAMENTO

O estudo sobre a acurácia resultante da solução de pareamento proposta neste trabalho ainda acontecerá, sendo de responsabilidade da equipe de estatísticos envolvidos no projeto. Porém, ainda é necessário verificar se a ferramenta de correlação desenvolvida para dar suporte à análise descrita no Capítulo 4 atende ao requisito de oferecer um resultado de acurácia competitiva entre os aplicativos já existentes.

Nesta seção, serão comparadas algumas soluções capazes de realizar o pareamento probabilístico (comentadas na Seção 4.3) com a proposta descrita nesta dissertação. Será construído um cenário onde bases de dados devem ser relacionadas considerando alguns erros entre seus registros e serão avaliados os resultados para cada uma das ferramentas. As métricas observadas para esta metodologia serão: quantidade de falsos positivos, falsos negativos e curva de similaridade de acordo com os erros entre um par de registros.

5.3.1 Cenário de Validação

Duas bases de dados foram construídas a partir do resultado das duas fases do vestibular da Universidade Federal da Bahia do ano de 2013 [2]. A lista dos candidatos aptos à segunda fase contém 126 registros e foram armazenados numa base chamada $FASE1$, destes, apenas 73 foram aprovados para ingressar nos próximos períodos letivos, contidos no repositório $BASE2$. Considerando que todos os aprovados na base menor estão presentes na maior, então, espera-se que uma rotina de pareamento entre essas listas retornasse um mínimo de 73 pares de registros.

Os atributos de ambas as bases são $NOME$ e RG, desconsiderando a possibilidade de realizar o pareamento através do simples relacionamento do último atributo, deseja-se executar a correlação para encontrar quais são os indivíduos comuns aos dois repositórios. Para avaliar o comportamento cada ferramenta utilizada neste procedimento, foram incluídos alguns erros numa porção limitada de registros.

De forma aleatória, foram eleitos 20 registros da $FASE2$ que sofreriam algum tipo de edição, estas podem ser feitas através de substituição ou supressão de caracteres. Para cada um desses 20 registros, foi sorteado quantas edições (de 1 a 5) e quais atributos ($NOME$, RG ou ambos) passariam por estas alterações.

O comportamento esperado para este experimento é que 53 registros que não foram

[2] As listas podem ser acessadas em: ⟨http://www.vestibular.ufba.br/⟩

ORIGINAIS		SORTEADO	
NOME	**RG**	**ERROS**	**ATTR.**
AIRTON SERRA RIBEIRO SENA	1195945280	4	AMBOS
ALESSANDRO SANTOS DA SILVA	701030623	2	NOME
ALLAN THALES RAMOS OLIVEIRA	1559162384	1	NOME
ALVARO DE JESUS PEREIRA	1376435942	1	AMBOS
AYRTON SILAS SILVA GUIMARÃES	1615085882	1	RG
CAIO VICTOR OLIVEIRA DA SILVA	1527979040	1	RG
DANIEL PINA DAVANZO	1111540063	5	RG
FERNANDO EDUARDO SANTANA MOREIRA AGUIAR JUNIOR	1320169406	1	NOME
JUNOT FREIRE DOS SANTOS NETO	1266387137	3	NOME
KAIO RODRIGO SANTOS PORTO	1483406792	4	RG
LAHERCE GOMES GONÇALVES	1362007960	3	NOME
LEONARDO SANTOS PEREIRA	1572365870	4	RG
LUCAS MAGALHAES DA SILVA LORDELO	1367053323	2	AMBOS
MARCOS RAMOS CONCEICAO	1379550220	4	NOME
MARINA PEIXOTO SANTOS	1327854120	5	RG
OSVALDO LIMA SANTANA FILHO	1451499515	3	NOME
PALOMA SANTIAGO MERCES	1375193546	5	RG
RAFAEL LIMA LELIS COELHO	1190940531	4	AMBOS
ROBERTO SALES CALDEIRA	1365641279	4	NOME
UÁLEX SILVA DE JESUS	1629435201	2	AMBOS

Tabela 5.9 Registros com número de erros e atributos que serão alterados eleitos aleatoriamente.

alterados da base $FASE2$ sejam pareados com seus representantes da $FASE1$ com o máximo de similaridade, enquanto, de acordo com a distância imposta pelas alterações, os 20 registros sorteados obtenham coeficientes de similitude cada vez menores. Para uma quantidade tão pequena de indivíduos, a possibilidade de se obter falsos positivos é menor.

O ponto de corte usado neste pareamento será feito manualmente, retirando da base resultante apenas as 73 primeiras linhas, o número exato de registros da base $FASE2$. Para registros que receberam mais alterações em ambos os atributos, é possível que se obtenha um índice de semelhança tão baixo que estes darão lugar a falsos positivos, se tornando, dessa forma, falsos negativos. Nas próximas subseções, serão aplicadas as melhores configurações para a correlação em diferentes aplicações, o objetivo deste experimento é verificar a competitividade da solução proposta por esta dissertação quanto as suas alternativas.

5.3.2 Resultados da Correlação em Spark

Aplicando os passos descritos pelo Capítulo 4, exceto pela blocagem, foi feito um pré-processamento das bases, aplicação de bigramas para construção dos filtros de Bloom e pareamento utilizando o coeficiente de Sorensen\Dice entre os registros das bases $FASE1$ e $FASE2$. O resultado deste método não gerou nenhum falso negativo ou falso positivo

ALTERADOS	
NOME	**RG**
AIRTON SENNA RIBEIRO DECA	1195945280
ALECSANDRO SANTOS DA SELVA	701030623
ALAN THALES RAMOS OLIVEIRA	1559162384
ALVARRO DE JESUS PEREIRA	1376435942
AYRTON SILAS SILVA GUIMARÃES	1615085882
CAIO VICTOR OLIVEIRA DA SILVA	1527979040
DANIEL PINA DAVANZO	1111540063
FERNANDO EDUARDO SANTANNA MOREIRA AGUIAR JUNIOR	1320169406
JUNOL FREYRE DOS SATOS NETO	1266387137
KAIO RODRIGO SANTOS PORTO	1483406792
LAERTE GOMES GONÇAUVES	1362007960
LEONARDO SANTOS PEREIRA	1572365870
LUCAS NAGALHAES DA SIVA LORDELO	1367053323
MARKOS RANOS CONCEISSAO	1379550220
MARINA PEIXOTO SANTOS	1327854120
OSVALDO L SANTANA FILHO	1451499515
PALOMA SANTIAGO MERCES	1375193546
RAZIEL LIMA LELYS COENHO	1190940531
RORBETO SALLE CALDEIRA	1365641279
WÁLEX SILVA DE JEZUS	1629435201

Tabela 5.10 Registros alterados após sorteio.

dentro do ponto de corte de 73 pares. A Figura 5.4 mostra um gráfico que ilustra a distribuição entre a similaridade dos pares, numa comparação visual, é possível perceber que a linha que representa a similaridade não é tão acentuada quanto a apresentada pelas demais soluções, o que proporciona um melhor agrupamento de registros semelhantes.

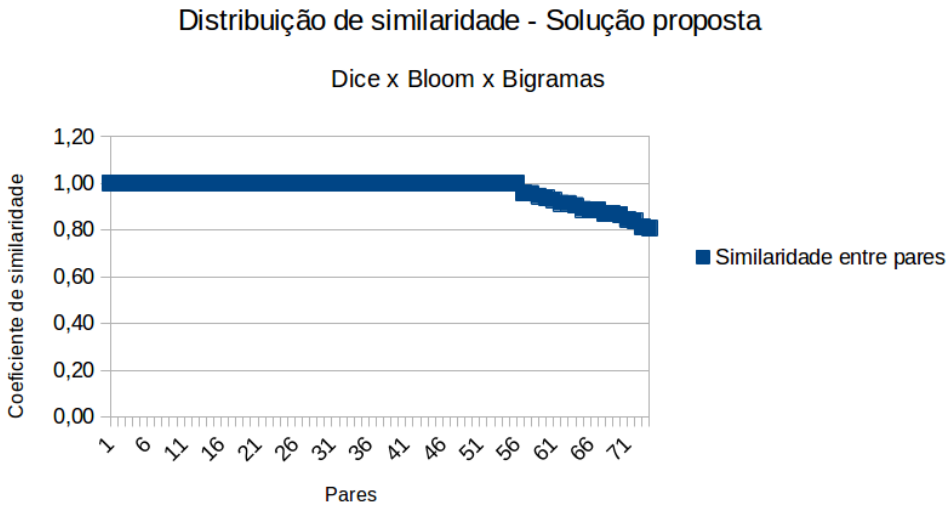

Figura 5.4 Resultado do pareamento utilizando a Correlação em Spark.

Apesar deste método ter sido inspirado no proposto pelos autores do *Merge Tool Box*

não fica claro no uso da ferramenta, ou no seu manual, se a construção dos filtros de Bloom através do *SAFELINK* é feita utilizando bigramas, em caso negativo, essa pode ser uma explicação para o fato desta aplicação demonstrar uma maior sensibilidade às diferenças encontradas nos registros da base $FASE2$. A Tabela 5.11 mostra uma comparação entre os resultados obtidos por cada ferramenta analisada.

	FRIL	MTB	Correlação em Spark
Total de pareados	69	73	73
Falsos positivos	0	1	0
Falsos negativos	4	1	0

Tabela 5.11 Tabela comparativa entre ferramentas de pareamento analisadas.

No quesito *tempo de execução*, a Correlação em Spark demonstrou superioridade em bases de dados com maior número de registros a serem pareados, nenhuma das ferramentas alternativas testadas obtiveram um bom funcionamento na tentativa de parear repositórios com alguns milhões de indivíduos. Dessa forma, é possível considerar a solução proposta por este trabalho competitiva entre as demais ferramentas existentes, principalmente considerando tarefas no contexto de *Big Data*.

5.3.3 Resultados do FRILL

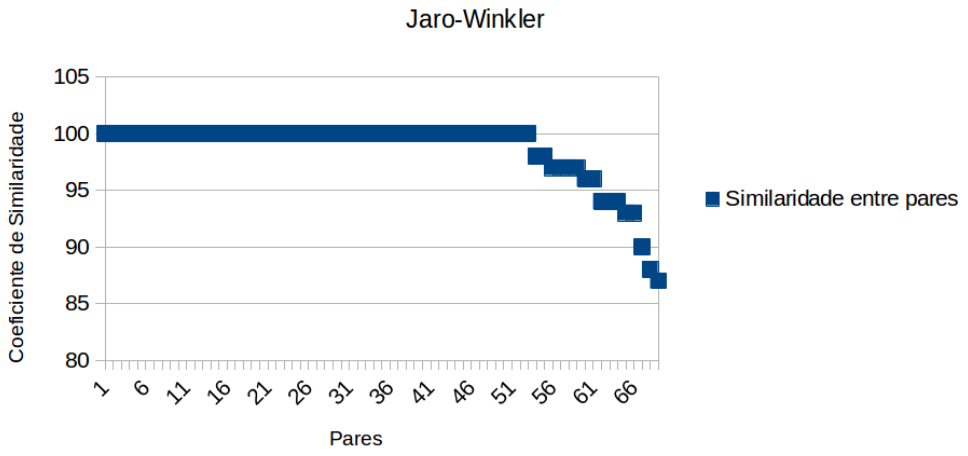

Figura 5.5 Resultado do pareamento utilizando o FRILL.

A melhor configuração encontrada utilizando o FRILL (JURCZYK et al., 2008) comparou cada um dos dois atributos das bases distintas considerando pesos iguais, utilizando a métrica de similaridade Jaro-Winkler (COHEN; RAVIKUMAR; FIENBERG, 2003) e o método de busca (ou blocagem) baseado em vizinhança ordenada. Essas definições ocasionaram inexistência de falsos positivos e menor número de falsos negativos. Nesta ferramenta, conseguiu-se parear 69 dos 73 registros esperados. A Figura 5.5 mostra um gráfico que ilustra o comportamento do aplicativo quanto a determinação de similaridade entre os pares.

Todos os quatro registros não encontrados neste pareamento possuem ao menos 3 erros no atributo *RG* (ou ambos). Em contrapartida, nenhum falso positivo obteve similaridade suficiente para ser considerado um par verdadeiro pelo aplicativo..

5.3.4 Resultados do Merge Toolbox (German RLC)

O *Merge Toolbox* (SCHNELL; BACHTELER; REIHER, 2005) trás a possibilidade de aplicar o pareamento probabilístico utilizando o cálculo de similitude entre um par de registros anonimizados pelos filtros de Bloom (BLOOM, 1970) através dos coeficientes de Dice (DICE, 1945) ou Jaccard (REAL; VARGAS, 1996). Esta ferramenta inclui um conjunto de variáveis capazes de configurar o comportamento probabilístico da correlação, pesos para atributos e combinações de verificações de similaridade.

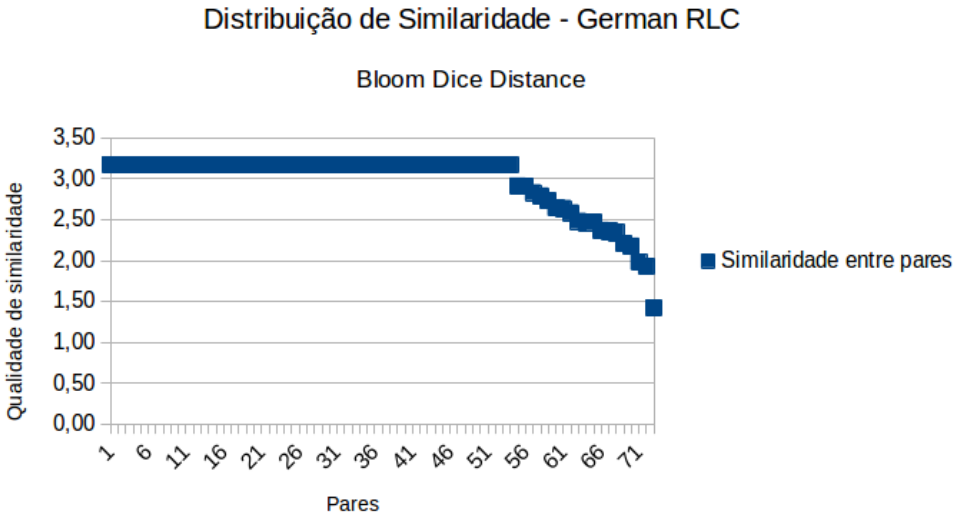

Figura 5.6 Resultado do pareamento utilizando o Merge Toolbox.

A melhor configuração do pareamento feito neste aplicativo inclui a comparação dos registros anonimizados em filtros de Bloom[3] através do "*Bloom Dice Distance*", sendo definida a opção "*One to One Matching*"que suprime qualquer pareamento duplicado. Todas as demais configurações foram mantidas em seu padrão.

Os resultados desta configuração geraram a ocorrência de um falso positivo (dois registros de bases distintas que não deveriam ser pareados) e um falso negativo (um registro que deveria ter sido pareado, mas não foi). Apesar disso, numa comparação visual, é possível notar uma semelhança na forma como as similaridades entre pares estão distribuídas nos gráficos das Figuras 5.5 e 5.6.

[3]Gerados a partir do pré-processamento feito através do SAFELINK (SCHNELL; BACHTELER; REIHER, 2007),outra ferramenta provida pelo Centro de *Record Linkage* Alemão.

Capítulo 6

Este capítulo tem foco na discussão sobre as conclusões, desafios e trabalhos futuros deste trabalho.

CONCLUSÕES

Motivada pelo atendimento aos requisitos apresentados pelo ISC-UFBA de uma aplicação de correlação probabilística de fontes de dados com grandes quantidades de registros provendo boas métricas de execução e comportamento competitivo quanto a ferramentas existentes, a proposta apresentada por esta dissertação foi implementada sobre o *framework* de processamento distribuído em *clusters* de alto desempenho Apache Spark utilizando um fluxo de trabalho capaz de apoiar a decisão sobre a combinação de pares de registros de bases de dados diferentes que representam um mesmo indivíduo.

As métricas de tempo de execução encontradas foram satisfatórias considerando o tamanho das bases de dados e a carência de ferramentas capazes de lidar com elas. A solução proposta também se mostrou competitiva, e até superior as alternativas, quanto a acurácia resultante do método, considerando um estudo básico em ambiente controlado. Estes resultados preliminares, assim como o método apresentado nesta dissertação tiveram um artigo aceito para publicação no *Workshop Beyond MapReduce 2015*, em conjunção com *18th International Conference on Extending Database Technology* (PITA et al.,).

O uso do framework de processamento distribuído Apache Spark, descrito no Capítulo 3 foi fundamental para que se conseguisse bons resultados de tempos de execução. Toda a implementação do fluxo de trabalho contou com o desenvolvimento orientado a transformações e ações nas coleções de dados persistentes e somente-leitura, chamados de RDDs. Apesar de obter métricas de tempo inferiores a abordagem *multithread* implementada em OpenMP, o Spark ainda mantém as vantagens de escalabilidade, distribuição, tolerância à falhas e facilidade de uso.

6.1 LIMITAÇÕES DO TRABALHO

A principal limitação no desenvolvimento deste trabalho foi a extração e decisão sobre os melhores métodos que comporiam o fluxo de trabalho a ser proposto da vasta literatura encontrada. Ademais, vencer a curva de aprendizado imposta pelo framework Spark e seu conceito de RDDs foi um importante obstáculo.

Considerando a abrangência dos estudos que utilizarão os resultados dos pareamentos feitos com a solução proposta neste trabalho, tem-se como um eminente desafio a necessidade de lidar com uma equipe multidisciplinar composta por profissionais renomados na área de biologia, epidemiologia e estatística.

Assim como toda ferramenta que permite a execução distribuída de uma tarefa sobre um conjunto de dados, todo um planejamento acerca do escalonamento de tarefas e o entendimento sobre o melhor uso do recurso deve ser considerado. Dessa forma, a validação dos códigos gerados foi crucial para garantir boa qualidade e corretude dos resultados. As métricas de tempo alcançadas na execução das tarefas *Spark* em pequenas bases também incluíram um melhor tratamento para blocos gerados com baixa quantidade de registros, específicamente aqueles com menos filtros de Bloom que núcleos disponíveis para o processamento. Porém, essas métricas não competitivas no emprego do *framework* fortalece a superioridade dele sobre fontes de dados no contexto de *Big Data* e abre o requisito por métodos complementares para demais cenários.

6.2 TRABALHOS FUTUROS

Considerando o contexto em que este trabalho está inserido, sua aplicação transcende o estudo de caso apresentado no Capítulo 2. A ambição do grupo multidisciplinar numa visão mais ampla é refinar a ferramenta desenvolvida para ser a base de um Centro Nacional de *Record Linkage*. A posse, manutenção e uso das bases de dados presentes na Tabela 4.1 já estão asseguradas por um convênio entre os ministérios envolvidos, a Fundação Oswaldo Cruz (FIOCRUZ, 2015) e ISC-UFBA. Porém, novas concessões e estudos que necessitem utilizar as base nominais (sem anonimização) terão maiores obstáculos por conta da Lei Da Informação (CIVIL, 2015), sancionada em 2011. Isso dará maior relevância aos centros de referência em correlação capazes de executar o pareamento entre duas ou mais bases usando um protocolo de segurança e anonimidade dos dados.

No contexto computacional, em paralelo a evolução do cenário exposto acima, cresce a necessidade da construção de uma plataforma capaz de prover suporte tecnológico aos requisitos de alto desempenho, segurança, tolerância à falhas, escalabilidade e abstração. Dessa forma, supõe-se que o emprego dos conceitos e ferramentas do Cloud Computing (MELL; GRANCE, 2011) e suas três principais camadas de serviço seriam ideais para atender as futuras demandas deste projeto.

A infraestrutura proposta pela Figura 6.1 contempla as três camadas de serviço da Computação em Nuvem. Na camada de Infraestrutura como Serviço (IaaS) serão providas as tecnologias necessárias para a manutenção do *cluster* de processamento e armazenamento de dados. A programabilidade e definições de desenvolvimento contarão com os serviços prestados pela Plataforma como Serviço (PaaS). O usuário final terá todo o apoio de um portal construído para uma necessidade específica, cada um desses portais atenderão demandas distintas (pareamento, busca, pré-processamento, testes de similaridade, etc.) e estarão presentes na camada de Software como Serviço (SaaS).

Para que toda essa proposta seja possível, maiores transformações deverão ser feitas para o fluxo de trabalho da Correlação em Spark, principalmente nas alternativas ofere-

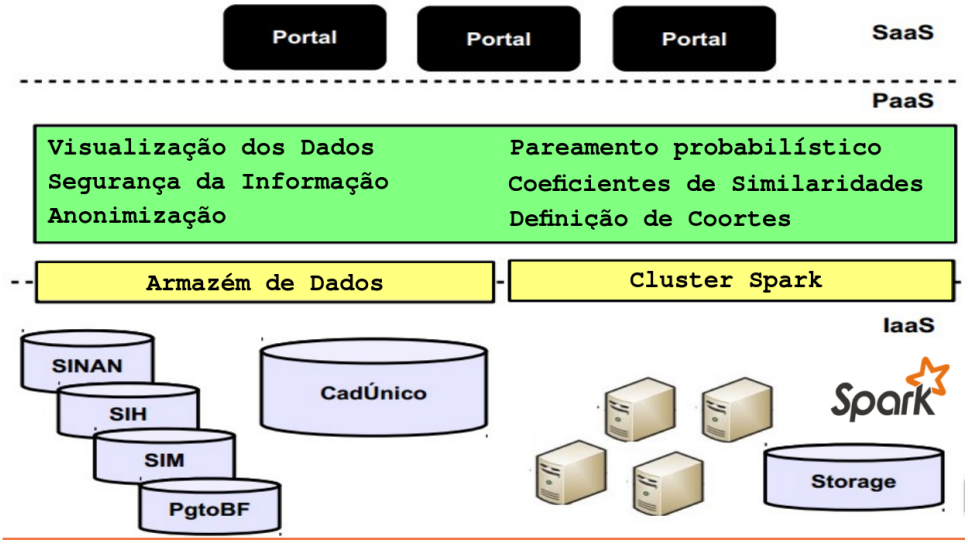

Figura 6.1 Proposta de infraestrutura em nuvem para prestação de serviços de Correlação

cidas para blocagem, coeficientes de similaridade e pareamento. A possibilidade escolher um método de blocagem pode amenizar os problemas de baixo número de registros que serão processados para cada par de blocos, abordagens mais inclusivas como a blocagem por predicados (TEJADA; KNOBLOCK; MINTON, 2001) ou *multibit trees* (BACHTELER; REIHER; SCHNELL, 2013b) já estão sendo estudadas. Para tornar a ferramenta mais competitiva a inclusão de novos testes de similitude deve ser possível, todas as alternativas apresentadas na Seção 4.3 possibilitam ao usuário final escolher qual coeficiente de similaridade será utilizado. Considera-se, num primeiro passo, implementar o algoritmo baseado em Jaccard (REAL; VARGAS, 1996) para testes em vetores de bits como é possível no *Merge Toolbox* (SCHNELL; BACHTELER; REIHER, 2005). Para definições no pareamento, variáveis capazes determinar as melhores configurações (como o algoritmo EM usado no RecLink (JR; COELI, 2000)), tolerância de falsos positivos e negativos serão outros desafios no desenvolvimento de novas versões desta proposta.

Outra melhoria que será implmentada com premência será a execução da correlação utilizando algoritmos fonéticos na etapa de pré-processamento. Os resultados esperados para essa mudança é uma atenuação nos problemas de imputação dos dados, o que pode impactar diretamente na acurácia resultante do método.

REFERÊNCIAS BIBLIOGRÁFICAS

AHUJA, S. P.; MOORE, B. State of big data analysis in the cloud. *Network and Communication Technologies*, v. 2, n. 1, p. p62, 2013.

AMARANTE, V. et al. *Social Assistance and Birth Outcomes: Evidence from the Uruguayan PANES*. [S.l.], 2011.

ANANTHAKRISHNA, R.; CHAUDHURI, S.; GANTI, V. Eliminating fuzzy duplicates in data warehouses. In: VLDB ENDOWMENT. *Proceedings of the 28th international conference on Very Large Data Bases*. [S.l.], 2002. p. 586–597.

ANDRADE, M. V. et al. Income transfer policies and the impacts on the immunization of children: the bolsa familia program. *Cadernos de Saúde Pública*, SciELO Brasil, v. 28, n. 7, p. 1347–1358, 2012.

ANTONI, M. German record linkage center (grlc). 2013.

aO, C. C. J.; ROSA, J. a. L. G. Metaphone-pt_br: The phonetic importance on search and correction of textual information. In: GELBUKH, A. F. (Ed.). *CICLing (2)*. Springer, 2012. (Lecture Notes in Computer Science, v. 7182), p. 297–305. ISBN 978-3-642-28600-1. Disponível em: ⟨http://dblp.uni-trier.de/db/conf/cicling/cicling2012-2.html\#JordaoR12⟩.

BACHTELER, T.; REIHER, J. *Tdgen: A test data generator for evaluating record linkage methods*. [S.l.], 2012.

BACHTELER, T.; REIHER, J.; SCHNELL, R. *Similarity filtering with multibit trees for record linkage*. [S.l.], 2013.

BACHTELER, T.; REIHER, J.; SCHNELL, R. *Similarity filtering with multibit trees for record linkage*. [S.l.], 2013.

BARROSO, E. C. et al. Fatores de risco para tuberculose multirresistente adquirida. *J Pneumol*, v. 29, n. 2, p. 89–97, 2003.

BAXTER, R.; CHRISTEN, P.; CHURCHES, T. A comparison of fast blocking methods for record linkage. In: CITESEER. *ACM SIGKDD*. [S.l.], 2003. v. 3, p. 25–27.

BENJELLOUN, O. et al. Swoosh: a generic approach to entity resolution. *The VLDB Journal?The International Journal on Very Large Data Bases*, Springer-Verlag New York, Inc., v. 18, n. 1, p. 255–276, 2009.

BERRY, M. J.; LINOFF, G. *Data Mining Techniques: For Marketing, Sales, and Customer Support*. New York, NY, USA: John Wiley & Sons, Inc., 1997. ISBN 0471179809.

BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, ACM, v. 13, n. 7, p. 422–426, 1970.

BOHM, L. F. Elaboração de uma estratégia de deduplicação de dados utilizando técnicas de blocagem em um cadastro hospitalar de pacientes. 2010.

BORTHAKUR, D. Hdfs architecture guide. *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf*, 2008.

BRAUW, A. de et al. The impact of bolsa familia on child, maternal, and household welfare. *mimeografía. Washington DC: Instituto Internacional de Investigaciones sobre Políticas Alimentarias*, 2012.

CHAUDHURI, S. et al. Robust and efficient fuzzy match for online data cleaning. In: ACM. *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. [S.l.], 2003. p. 313–324.

CHEN, H.; CHIANG, R. H.; STOREY, V. C. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, v. 36, n. 4, p. 1165–1188, 2012.

CIVIL, C. *LEI No 12.527, DE 18 DE NOVEMBRO DE 2011. (Lei da Informação)*. 2015. Disponível em: ⟨http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2011/lei/l12527.htm⟩.

COHEN, A. *FuzzyWuzzy: Fuzzy String Matching in Python*. 2011. Disponível em: ⟨http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/⟩.

COHEN, W.; RAVIKUMAR, P.; FIENBERG, S. A comparison of string metrics for matching names and records. In: *KDD Workshop on Data Cleaning and Object Consolidation*. [S.l.: s.n.], 2003. v. 3, p. 73–78.

COUTINHO, E. S. F.; COELI, C. M. Acurácia da metodologia de relacionamento probabilístico de registros para identificação de óbitos em estudos de sobrevida accuracy of the probabilistic record linkage methodology to ascertain deaths. *Cad. Saúde Pública*, SciELO Public Health, v. 22, n. 10, p. 2249–2252, 2006.

DAGUM, L.; MENON, R. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, IEEE, v. 5, n. 1, p. 46–55, 1998.

DAVENPORT, T. H.; HARRIS, J. G. *Competing on analytics: The new science of winning*. [S.l.]: Harvard Business Press, 2007.

DEAN, J.; GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, ACM, v. 51, n. 1, p. 107–113, 2008.

DEHEJIA, R. H.; WAHBA, S. *Propensity Score Matching Methods for Non-experimental Causal Studies*. [S.l.], 1998. (Working Paper Series, 6829). Disponível em: ⟨http://www.nber.org/papers/w6829⟩.

DEVLIN, B. *Data Warehouse: From Architecture to Implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996. ISBN 0201964252.

DICE, L. R. Measures of the amount of ecologic association between species. *Ecology*, JSTOR, v. 26, n. 3, p. 297–302, 1945.

DOIRON, D.; RAINA, P.; FORTIER, I. Linking canadian population health data: maximizing the potential of cohort and administrative data. *Canadian journal of public health= Revue canadienne de sante publique*, PMC Canada manuscript submission, v. 104, n. 3, p. e258.

DREMEL, A. J. *DREMEL*. [S.l.]: Google Patents, 1955. US Patent 2,721,587.

EKANAYAKE, J. et al. Twister: a runtime for iterative mapreduce. In: ACM. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. [S.l.], 2010. p. 810–818.

FELLEGI, I. P.; SUNTER, A. B. A theory for record linkage. *Journal of the American Statistical Association*, Taylor & Francis Group, v. 64, n. 328, p. 1183–1210, 1969.

FILHO, W. D. S. *ALGORITMO PARALELO E EFICIENTE PARA O PROBLEMA DE PAREAMENTO DE DADOS*. 2008. Disponível em: ⟨http://www.dcc.ufmg.br/pos/cursos/defesas/808M.PDF⟩.

FIOCRUZ, F. ao O. C. *Fundação Osvado Cruz (FIOCRUZ): Ciência e tecnologia em saúde para população brasileira*. 2015. Disponível em: ⟨http://portal.fiocruz.br/⟩.

FISZBEIN, A.; SCHADY, N. R.; FERREIRA, F. H. *Conditional cash transfers: reducing present and future poverty*. [S.l.]: World Bank Publications, 2009.

FRIEDEN, T. R. A framework for public health action: the health impact pyramid. *American journal of public health*, American Public Health Association, v. 100, n. 4, p. 590–595, 2010.

GEORGE, L. *HBase: the definitive guide*. [S.l.]: "O'Reilly Media, Inc.", 2011.

GHEMAWAT, S.; GOBIOFF, H.; LEUNG, S.-T. The google file system. In: ACM. *ACM SIGOPS Operating Systems Review*. [S.l.], 2003. v. 37, n. 5, p. 29–43.

GRAVANO, L. et al. Approximate string joins in a database (almost) for free. In: *VLDB*. [S.l.: s.n.], 2001. v. 1, p. 491–500.

GROOT, S.; KITSUREGAWA, M. Jumbo: Beyond mapreduce for workload balancing. In: *36th International Conference on Very Large Data Bases, Singapore*. [S.l.: s.n.], 2010.

GUANAIS, F. C. The combined effects of the expansion of primary health care and conditional cash transfers on infant mortality in brazil, 1998–2010. *American journal of public health*, American Public Health Association, v. 103, n. 11, p. 2000–2006, 2013.

HADOOP, A. *Hadoop*. 2009.

HALL, P. A.; DOWLING, G. R. Approximate string matching. *ACM computing surveys (CSUR)*, ACM, v. 12, n. 4, p. 381–402, 1980.

HAN, J. et al. Survey on nosql database. In: IEEE. *Pervasive computing and applications (ICPCA), 2011 6th international conference on*. [S.l.], 2011. p. 363–366.

HERNÁNDEZ, M. A.; STOLFO, S. J. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data mining and knowledge discovery*, Springer, v. 2, n. 1, p. 9–37, 1998.

HINDMAN, B. et al. Mesos: A platform for fine-grained resource sharing in the data center. In: *NSDI*. [S.l.: s.n.], 2011. v. 11, p. 22–22.

IMBENS, G.; LEMIEUX, T. *Regression Discontinuity Designs: A Guide to Practice*. [S.l.], 2007. (Technical Working Paper Series, 337). Disponível em: ⟨http://www.nber.org/papers/t0337⟩.

JARO, M. A. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, Taylor & Francis, v. 84, n. 406, p. 414–420, 1989.

JR, K. R. d. C.; COELI, C. M. Reclink: aplicativo para o relacionamento de bases de dados, implementando o método probabilistic record linkage. *Cadernos de Saúde Pública*, SciELO Public Health, v. 16, n. 2, p. 439–447, 2000.

JUNGER, W. L. Estimação de parâmetros em relacionamento probabilístico de bancos de dados: uma aplicação do algoritmo em para o reclink. *Cad. saúde colet.,(Rio J.)*, v. 14, n. 2, p. 225–232, 2006.

JUNIOR, K. R. d. C.; COELI, C. M. Reclink 3: nova versão do programa que implementa a técnica de associação probabilística de registros (probabilistic record linkage). *Cad. saúde colet.,(Rio J.)*, v. 14, n. 2, p. 399–404, 2006.

JURCZYK, P. et al. Fine-grained record integration and linkage tool. *Birth Defects Research Part A: Clinical and Molecular Teratology*, Wiley Online Library, v. 82, n. 11, p. 822–829, 2008.

KHARE, R. et al. Nutch: A flexible and scalable open-source web search engine. *Oregon State University*, v. 1, p. 32–32, 2004.

KOLB, L.; THOR, A.; RAHM, E. Dedoop: efficient deduplication with hadoop. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 5, n. 12, p. 1878–1881, 2012.

LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions and reversals. In: *Soviet physics doklady*. [S.l.: s.n.], 1966. v. 10, p. 707.

LOW, Y. et al. Graphlab: A new framework for parallel machine learning. *arXiv preprint arXiv:1006.4990*, 2010.

LUCENA, F. *BUSCA FONÉTICA EM PORTUGUÊS DO BRASIL*. 2006. Disponível em: ⟨http://www.brunoportfolio.com/arquivos/pdf/BuscaBR_Fonetica.pdf⟩.

MAGGI, F. A survey of probabilistic record matching models, techniques and tools. *Scienti_c Report TR-2008*, 2008.

MDS, M. d. D. *Bolsa Família*. 2014. ⟨http://www.mds.gov.br/bolsafamilia/⟩. [Online; accessed 05-Nov-2014].

MELL, P.; GRANCE, T. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology, 2011.

MEYER, M. *Distance 0.1.3: Utilities for comparing sequences*. 2013. Disponível em: ⟨https://pypi.python.org/pypi/Distance⟩.

MICHELSON, M.; KNOBLOCK, C. A. Learning blocking schemes for record linkage. In: MENLO PARK, CA; CAMBRIDGE, MA; LONDON; AAAI PRESS; MIT PRESS; 1999. *Proceedings of the National Conference on Artificial Intelligence*. [S.l.], 2006. v. 21, n. 1, p. 440.

MILLER, K. Big data analytics in biomedical research. *Biomedical Computation Review*, p. 14–21, 2012.

MONE, G. Beyond hadoop. *Communications of the ACM*, ACM, v. 56, n. 1, p. 22–24, 2013.

MORTIMER, J.; SALATHIEL, J. 'soundex'codes of surnames provide confidentiality and accuracy in a national hiv database. *Communicable disease report. CDR review*, v. 5, n. 12, p. R183–6, 1995.

MS, M. d. S. *Manual de orientações sobre o Bolsa Família na Saúde. Brasília: MS*. 2013. ⟨http://bolsafamilia.datasus.gov.br/documentos_bfa/MANUAL_PBF_BOLSAFAMILIA_SAUDE2009.PDF⟩. [Online; accessed 17-Jul-2013].

MURDOCH, T. B.; DETSKY, A. S. The inevitable application of big data to health care. *JAMA*, American Medical Association, v. 309, n. 13, p. 1351–1352, 2013.

NEWCOMBE, H. et al. Automatic linkage of vital records. In: DEPT. OF THE TREASURY, INTERNAL REVENUE SERVICE, STATISTICS OF INCOME DIVISION.

Record linkage techniques, 1985: proceedings of the Workshop on Exact Matching Methodologies, Arlington, Virginia, May 9-10, 1985: co-sponsored with the Washington Statistical Society and the Federal Committee on Statistical Methodology. [S.l.], 1986. v. 1299, p. 7.

NEWCOMBE, H. B. Record linking: the design of efficient systems for linking records into individual and family histories. *American Journal of Human Genetics*, Elsevier, v. 19, n. 3 Pt 1, p. 335, 1967.

NITZBERG, B.; LO, V. Distributed shared memory: A survey of issues and algorithms. *Distributed Shared Memory-Concepts and Systems*, p. 42–50, 1991.

U.S. BUREAU OF ARCHIVES. Robert Russel Margareth Odell. *SOUNDEX Patent*. 1918. 01 261 167.

OLIVEIRA, H. B. de; FILHO, D. d. C. M. Recidivas em tuberculose e seus fatores de risco. *Rev Panam Salud Publica*, SciELO Public Health, v. 7, n. 4, p. 233, 2000.

OLSTON, C. et al. Pig latin: a not-so-foreign language for data processing. In: ACM. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. [S.l.], 2008. p. 1099–1110.

PFEIFER, U. et al. Searching proper names in databases. In: CITESEER. *HIM*. [S.l.], 1995. p. 259–275.

PITA, R. et al. A spark-based workflow for probabilistic record linkage of healthcare data.

PY4J. *Py4J — SourceForge.net*. 2014. Disponível em: ⟨http://sourceforge.net/projects/py4j/⟩.

QUANTIN, C. et al. How to ensure data security of an epidemiological follow-up: quality assessment of an anonymous record linkage procedure. *International journal of medical informatics*, Elsevier, v. 49, n. 1, p. 117–122, 1998.

QUANTIN, C.; RIANDEY, B. Epidemiological and statistical secured matching in france.

RACCHUMI-ROMERO, J. A. *Utilizando o Relacionamento de Bases de Dados para Avaliação de Políticas Públicas: uma Aplicação para o Programa Bolsa Família*. Tese (Doutorado) — Tese de Doutorado em Demografia, Centro de Desenvolvimento e Planejamento Regional, Universidade Federal de Minas Gerais.[Links], 2008.

RAHM, E.; DO, H. H. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, v. 23, n. 4, p. 3–13, 2000.

RAICU, I.; FOSTER, I. T.; ZHAO, Y. Many-task computing for grids and supercomputers. In: *IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS08) 2008*. [S.l.: s.n.], 2008.

RASELLA, D. et al. Effect of a conditional cash transfer programme on childhood mortality: a nationwide analysis of brazilian municipalities. *The lancet*, Elsevier, v. 382, n. 9886, p. 57–64, 2013.

REAL, R.; VARGAS, J. M. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, JSTOR, p. 380–385, 1996.

RISTAD, E. S.; YIANILOS, P. N. Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 20, n. 5, p. 522–532, 1998.

RLC, G. *German Record Linkage Centers*. 2011. Disponível em: ⟨http://record-linkage.de/⟩.

RÖSCH, S. Beyond map reduce. 2012.

ROSEN, J. *Python API for Spark*. 2012. Disponível em: ⟨http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/josh-rosen-amp-camp-2012-spark-python-api-final.pdf⟩.

RUBIN, D. B. Estimating causal effects from large data sets using propensity scores. *Annals of internal medicine*, Am Coll Physicians, v. 127, n. 8_Part_2, p. 757–763, 1997.

SAGI, M. d. D. *Avaliação do Impacto do Programa Bolsa Família - 2a rodada (AIBF II)*. 2013. ⟨http://aplicacoes.mds.gov.br/sagi/PainelPEI/Publicacoes/AvaliacaodeImpactoProgramaBolsaFamiliaII.pdf⟩. [Online; accessed 7-Nov-2014].

SANTOSI, A. S. dos; CASTROII, D. S. de; FALQUETOII, A. Fatores de risco para transmissão da hanseníase. *Revista Brasileira de Enfermagem*, SciELO Brasil, v. 61, p. 738–743, 2008.

SCANNAPIECO, M.; VIRGILLITO, A.; ZARDETTO, D. Placing big data in official statistics: A big challenge. *URL: Paper for the NTTS*, v. 201, 2013.

SCHNEIDER, B. *Estimating causal effects: using experimental and observational designs: a think tank white paper*. [S.l.]: Amer Educational Research Assn, 2007.

SCHNELL, R. Linking surveys and administrative data. 2013.

SCHNELL, R.; BACHTELER, T.; REIHER, J. Mtb: Ein record-linkage-programm für die empirische sozialforschung. Deutschland, 2005.

SCHNELL, R.; BACHTELER, T.; REIHER, J. Improving record-linkage-software for survey-data. 2007.

SCHNELL, R.; BACHTELER, T.; REIHER, J. Privacy-preserving record linkage using bloom filters. *BMC Medical Informatics and Decision Making*, BioMed Central Ltd, v. 9, n. 1, p. 41, 2009.

SHEI, A. Brazil's conditional cash transfer program associated with declines in infant mortality rates. *Health Affairs*, Health Affairs, v. 32, n. 7, p. 1274–1281, 2013.

SILVA, M. O. d. S. The family allowance program: reflecting on core issues in brazil's income transfer policy. *Ciência & Saúde Coletiva*, SciELO Public Health, v. 12, n. 6, p. 1429–1439, 2007.

SPARK, A. *Spark Programming Guide*. 2014. Disponível em: ⟨http://spark.apache.org/docs/latest/programming-guide.html\#rdd-operations⟩.

SPARK, A. *Spark Programming Guide*. 2014. Disponível em: ⟨http://spark.apache.org/docs/latest/programming-guide.html\#shared-variables⟩.

TEJADA, S.; KNOBLOCK, C. A.; MINTON, S. Learning object identification rules for information integration. *Information Systems*, Elsevier, v. 26, n. 8, p. 607–633, 2001.

THUSOO, A. et al. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, VLDB Endowment, v. 2, n. 2, p. 1626–1629, 2009.

VAVILAPALLI, V. K. et al. Apache hadoop yarn: Yet another resource negotiator. In: ACM. *Proceedings of the 4th annual Symposium on Cloud Computing*. [S.l.], 2013. p. 5.

VENNER, J.; CYRUS, S. *Pro Hadoop*. [S.l.]: Springer, 2009.

WHO. e-publication. *Report of the global forum on elimination of leprosy as a public health problem*. Geneva: WHO, 2006. 27 p. p. (WHO/CDS/NTD/2006.4). Disponível em: ⟨http://whqlibdoc.who.int/hq/2006/WHO_CDS_NTD_2006.4_eng.pdf⟩.

WIKIBOOKS. *Algorithm Implementation/Strings/Levenshtein distance*. 2014. Disponível em: ⟨http://en.wikibooks.org/wiki/Algorithm_Implementation/Strings/Levenshtein_distance\#Python⟩.

WINKLER, W. E. Preprocessing of lists and string comparison. *W. Alvey and B*, 1985.

WINKLER, W. E. Overview of record linkage and current research directions. In: CITESEER. *Bureau of the Census*. [S.l.], 2006.

XIN, R. S. et al. Shark: Sql and rich analytics at scale. In: ACM. *Proceedings of the 2013 international conference on Management of data*. [S.l.], 2013. p. 13–24.

ZAHARIA, M. An architecture for fast and general data processing on large clusters. 2014.

ZAHARIA, M. et al. Spark: cluster computing with working sets. In: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. [S.l.: s.n.], 2010. p. 10–10.

ZIKOPOULOS, P.; EATON, C. et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. [S.l.]: McGraw-Hill Osborne Media, 2011.