Redblock: Uma ferramenta para a deduplicação de grandes bases de dados em tempo real

Luan Félix Pimentel[1], **Igor Lemos Vicente**[1], **Guilherme Dal Bianco**[1]

[1]Universidade Federal da Fronteira Sul (UFFS)
Caixa Postal 101 – 89802-112
Departamento de Ciência da Computação
Chapecó, SC - Brasil.

luanfelixpimentel@gmail.com, guilherme.dalbianco@uffs.edu.br

Abstract. *Online data blocking aims to identify records that represent the same purpose on a streaming data environment. Real-time data blocking must be able to process a range of informations with a high effectiveness and no delays. The purpose of this paper is to introduce a developed tool named Redblock for real-time data deduplication using a distributed platform for online processement combined with a data blocking technique that make use of an Inverted Index. The tool managed to provide good preliminary results in terms of effectiveness in a synthetic database.*

Resumo. *A blocagem online de dados tem como propósito identificar registros que representam um mesmo objetivo em ambientes com fluxo contínuo de dados. A blocagem online deve ser capaz de processar volumes variados de informações, sem atrasos e com uma alta eficácia. Este trabalho, propõe uma ferramenta intitulada Redblock para a deduplicação de dados em tempo real. A ferramenta utiliza uma plataforma distribuída de processamento online em conjunto com um método de blocagem utilizando índice invertido. Na experimentação, Redblock demonstrou bons resultados preliminares em relação a sua eficácia em uma base de dados sintética.*

1. Introdução

A integração de dados tem como objetivo facilitar o acesso a informações a partir da consolidação de diferentes fontes de dados em um único repositório. Serviços como bibliotecas virtuais, *media streaming* e redes sociais dependem de um processo de integração com uma alta qualidade. Para isto, uma tarefa fundamental é a identificação de entidades (registros, documentos, textos, etc.) que já estão armazenadas na base de dados, portanto não devem ser novamente inseridas. Tal etapa, conhecida como deduplicação, tem por objetivo melhorar a qualidade das bases de dados, identificando entidades duplicadas.

A deduplicação envolve três etapas principais: blocagem, comparação e a classificação [Christen 2012]. A blocagem corresponde ao processo de geração de pares candidatos. Ou seja, todos os registros devem ser analisados em busca de potenciais duplicatas. Somente registros pertencentes a um mesmo bloco são utilizados para a criação dos pares candidatos com custo quadrático de processamento. Na etapa de comparação, a partir do conjunto de pares candidatos é computada a similaridade de cada par utilizando funções de similaridade (ex: Jaccard, Jaro, Ngram) [Mitra et al. 2005]. Por fim, na etapa

de classificação são utilizados algoritmos de classificação (árvores de decisão, máquinas de vetores de suporte, *Naive Bayes*, entre outros [Manning et al. 2008] ou heurísticas configuradas a partir de limiares manualmente definidos, para identificar quais pares representam uma duplicata.

Tradicionalmente, a deduplicação é tratada em bancos de dados estáticos (ou em lote). Isto quer dizer que a base de dados é processada, na sua totalidade, em busca de registros que representam dados duplicados. No entanto, o crescimento de serviços e aplicações com fluxo contínuo de dados em tempo real impulsiona a demanda por soluções capazes de suportar tal fluxo de dados. A deduplicação online, diferente da versão estática, deve ser capaz de lidar com picos de processamento sem que sejam evidenciados gargalos e ao mesmo tempo deve ser capaz de se adaptar a possíveis alterações nos padrões dos dados.

Este artigo propõe uma ferramenta, intitulada *Redblock*, para a deduplicação de dados online utilizando a plataforma *Apache Storm*. Essa plataforma é um sistema de computação distribuído e tolerante a falhas, que permite processar dados em tempo real. O principal desafio da *Redblock* é garantir que o maior número possível de pares duplicados sejam identificados. Apesar da *Redblock* implementar as principais etapas de deduplicação (blocagem e classificação), a principal contribuição é no processo de blocagem, no qual utilizamos um índice invertido que é armazenado em um banco de dados chave-valor. O objetivo desse banco de dados é otimizar o processo de busca das informações necessárias para a deduplicação dos dados. Para o processo de classificação, a *Redblock* utiliza um algoritmo supervisionado baseado em árvore de decisão. Tal algoritmo, por ser supervisionado, depende de um treinamento que deve ser minimizado o máximo possível, devido ao custo de rotulação. Dessa forma, a ferramenta utiliza uma amostragem aleatória para compor o treinamento do método.

O presente trabalho está organizado da seguinte forma: Na segunda seção é apresentados o embasamento para o entendimento da *Redblock*. Na seção 3 é descrito o funcionamento da ferramenta. Na quarta seção são exemplificados alguns trabalhos relacionados envolvendo o processo blocagem online. Na seção 5 apresentamos os experimentos e resultados obtidos pela terceira seção. Por fim, são apresentadas as considerações finais e como serão os próximos passos de evolução e aprimoramento da ferramenta.

2. Referencial Teórico

Com o objetivo de efetivar o entendimento do trabalho proposto, nesta seção serão apresentados alguns métodos presentes na literatura envolvendo a blocagem de dados. Em seguida, serão descritos os principais conceitos envolvendo o *framework* de processamento online *Apache Storm*, utilizado para o desenvolvimento da *Redblock*.

2.1. Técnicas de Blocagem

Um método mais simplista de produzir grupos ou blocos de dados é definir um dos atributos como o critério ou chave de blocagem. Somente registros que apresentarem uma mesma chave de blocagem serão inseridos em um mesmo bloco, reduzindo substancialmente o número de comparações. No entanto, em situações em que o atributo chave apresentar variações ou erros, pares duplicados poderão não ser agrupados corretamente reduzindo assim a qualidade do método [Elmagarmid et al. 2007].

Já a técnica de *Blocagem por Vizinho Ordenado* (BVO) busca ordenar a base de dados de acordo com os valores-chave dos blocos e sequencialmente move uma janela de tamanho fixo que irá criar os grupos de dados sobre os valores. Inicialmente, a base de dados é ordenada a partir do valor-chave selecionado (por exemplo, o atributo nome em uma tabela de cadastro) e em seguida, é organizado em ordem alfabética. A partir disso, os pares candidatos são gerados utilizando uma janela de tamanho fixo. Na Tabela 1 é apresentado um exemplo de base de dados contendo 4 registros, que ao aplicar uma janela deslizante com tamanho 3 produz um primeiro bloco contendo os primeiros 3 registros da base de dados (R1, R2 e R3) e outro bloco contendo os registros (R2, R3 e R4), conforme ilustrado na Tabela 2.

Identificador	Nome
R1	João Silva
R2	João
R3	Jo
R4	Teresinha Souza

Tabela 1. Exemplo de dados para aplicação da Indexação por vizinho ordenado.

Intervalo	Pares Candidatos
R1 - R3	(R1, R2), (R2,R3), (R1,R3)
R2 - R4	(R2,R3), (R3,R4), (R2,R4)

Tabela 2. Resultado da Indexação por vizinho ordenado, utilizando os registros da Tabela 1.

Um problema identificado por [Christen 2012] no método BVO é que a ordenação dos valores chave dos blocos são sensíveis perante erros e variações nas primeiras posições dos valores. Por exemplo, se a base de dados fosse ainda maior, ”Christina”e ”Kristina”estariam bem distantes na lista organizada em ordem alfabética, mesmo sendo nomes bem semelhantes que talvez se refiram a mesma pessoa. Para contornar esse problema o método *Q-Gram* utiliza como chave de blocagem *substrings* de um atributo com tamanho *Q*. Os valores-chave do bloco geram *strings* utilizando os *grams* que se tornarão o valor-chave em um índice e seus componentes comparados posteriormente. Apesar do método *Q-Gram* obter bons resultados, o alto custo de se indexar substrings [Baxter et al. 2003] limita a aplicação do método a um atributo em específico.

Por fim, o método de Índice Invertido, utiliza palavras como forma de indexação de uma coleção de registros. As nomenclaturas utilizadas por esse método são separadas por Vocabulário (diferentes palavras do documento) e Ocorrência (frequência que determinada palavra aparece no documento). Abaixo, exemplificamos na Tabela 3 como estariam organizados os dados recebidos, com registro separado por vírgulas. Já na Tabela 4, apresentamos uma simples visualização do resultado da aplicação do método do Índice Invertido, usando como referência a primeira linha da tabela 3. Como pode ser notado, o método foi capaz de agrupar o registro 3 junto do restante, utilizando como critério o nome da rua.

Docs	Texto
1	João Silva, (49) 987453214, Avenida Gusmão Freire
2	João, (49) 987453214, Avenida Gusmão Freire
3	Jo Silvo, (49) 987453218, Avenida Freire

Tabela 3. Exemplo de dados com base para aplicação do Índice Invertido.

Número	Termo	Docs
1	João	1, 2
2	Silva	2
3	(49)987453214	1,2
4	Avenida	1,2, 3
5	Gusmão	1,2
6	Freire	1,2,3

Tabela 4. Organização dos dados após aplicação do Índice Invertido utilizando as informações do Doc 1, Tabela 3.

No índice invertido, pode-se perceber que sua metodologia de aplicação é similar ao método tradicional da blocagem. Neste método, não escolhemos um determinado campo e com base nele produzimos índices. Pelo contrário, aqui utilizamos todos os campos e geramos índices para cada registro que foi processado [Christen 2012]. Devido a esta flexibilidade tal método de blocagem foi explorado pela ferramenta *Redblock*, conforme será descrito na Seção 3.

2.2. Apache Storm

A plataforma *Apache Storm* possibilita desenvolver aplicações que demandem processamento massivo de dados em tempo real. É utilizada uma metodologia própria que permite que uma coleção de tuplas (lista de valores) sejam distribuídas e processadas por *spouts e bolts*. Os *spouts* são similares a processos com a função de realizar a leitura de uma fonte de dados. Já os *bolts*, processam as tuplas recebidas para serem distribuídas a outros *bolts* ou armazenam as informações em fontes externas. Os *bolts e spouts* são integrados a partir de uma topologia que possibilita conectar um *bolt* e *spouts* a fim de se formar uma arquitetura. Como o objetivo é o processamento contínuo de dados, a topologia será executada até que o usuário interrompa o processo.

A Figura 1 ilustra um exemplo de topologia com 2 *spouts* e 5 *bolts*. No exemplo, os *spouts* recebem os dados e os distribui para os outros 3 *bolts* seguintes. Esses, por sua vez, irão realizar um processamento sobre os dados e encaminharão para outros 2 *bolts* finais. Estes serão responsáveis por transformar e processar os dados recebidos do *bolt* anterior e salvar as informações necessárias[1].

[1]Informações fornecidas pela Apache Storm. Disponível em: $< http : //storm.apache.org/ >$. Acesso em 6 de fevereiro de 2017.

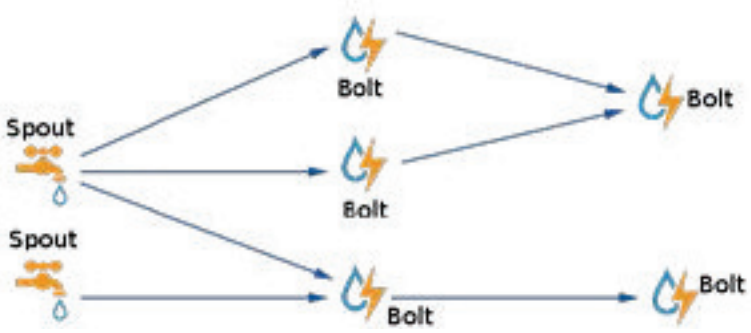

Figura 1. Exemplo básico do funcionamento de *Bolts* e *Spouts* em *Apache Storm*

3. Proposta - *Redblock*

Nesta seção, será descrita a ferramenta proposta para a deduplicação de dados em tempo real, denominada *Redblock*, com o objetivo de desenvolver um eficiente método para deduplicação online. A *Redblock* combina o método do índice invertido para o agrupamento de dados com a plataforma de processamento online *Apache Storm* para obter elasticidade no processamento massivo de dados.

Na Figura 2, ilustramos a topologia proposta. A *Redblock* é composta por 7 etapas principais com o objetivo de fragmentar o processamento, possibilitar que sejam identificados gargalos de processamento e tratados através do aumento do nível de paralelismo. Por exemplo, se o *bolt* que trata do processo de blocagem sofrer sobrecarga, é possível aumentar o número de *bolts* para esta tarefa para evitar atrasos de processamento. As linhas pontilhadas representam processos que ocorrem em paralelo no funcionamento da ferramenta, sendo respectivamente os *Bolts* que salvam dados em memória e o *Bolt* de treinamento que é iniciado previamente ao *Decision Tree Bolt* e ao *Counter Bolt* uma única vez. A seguir cada uma das etapas é descrita.

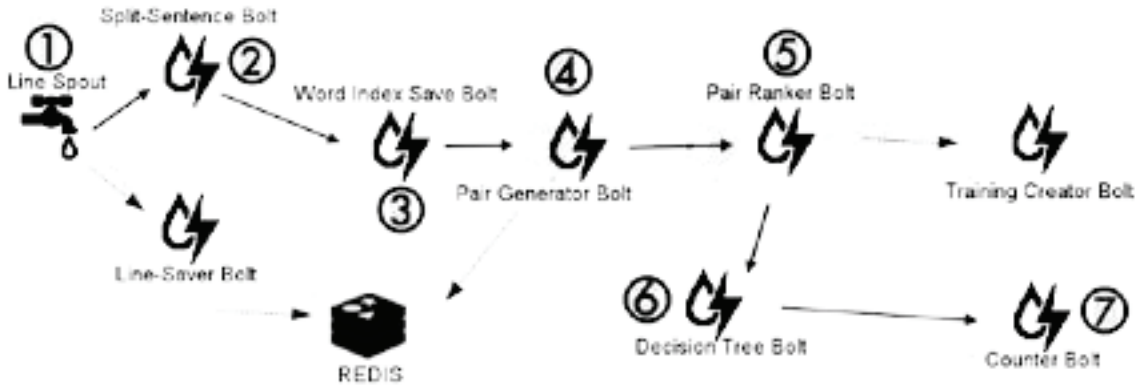

Figura 2. Topologia da ferramenta baseada na metodologia do *ApacheStorm*. Os números junto as etapas, ilustram o fluxo principal de execução da Redblock.

3.1. Spout - *Line Spout*

O *Line Spout* (1) representa o passo inicial da topologia. Ele desempenha a leitura da base de dados. Note que para desenvolver um ambiente de teste, o fluxo contínuo de dados é simulado através do carregamento de uma base de dados.

Após a leitura do arquivo, o *Line Spout* processa linha a linha da base de dados e envia a tupla gerada para o tratamento posterior do *bolt* destinatário. Por exemplo, a base de dados da Tabela 5 é carregada e enviada respectivamente para os *Bolts Line Saver* e *Split Sentence*. Note que uma cópia de cada registro é enviada para o *Bolt Line Saver* e outra cópia para o *Bolt Split Sentence*.

Registros do arquivo de entrada
01, joão, 6562348, estudante
02, joão, 98542138, professor
03, roberto, 52132157, estudante

Tabela 5. Exemplo de dados de entrada utilizados pela ferramenta.

3.2. Bolt - *Line Saver*

Como o processamento dos dados é em tempo real, um processo (ou um bolt) não congela enquanto outro está sendo executado. Portanto, o *Line Saver* realiza sua função em paralelo ao *Split Sentence Bolt*, recebendo a tupla e fazendo a sua tarefa.

O *Line Saver* tem como objetivo salvar a linha no banco de dados não relacional *REDIS*, armazenando os dados em memória para uma otimização da recuperação posterior. Ou seja, a tupla enviada pelo *spout* é salva no formato [ID][Linha].

3.3. Bolt - *Split Sentence*

Prosseguindo com o fluxo contínuo, o *Split Sentence Bolt* (2) recebe como entrada uma linha original (conforme a Tabela 5) e promove o processo de fragmentação. Ou seja, cada um dos atributos é emitido para a etapa seguinte para posterior indexação. Por exemplo, o primeiro registro da Tabela 5 é fragmentado em 3 partes: ”joão”, ”6562348”e ”estudante”e enviado para a próxima etapa juntamente com o ID de referência. Todos os campos serão emitidos para o *bolt* seguinte no formato [ID][campo].

3.4. Bolt - *WordIndex Save*

No *bolt- WordIndex Save* (3) é iniciado o processamento dos dados que são recebidos utilizando o método do índice invertido. Especificamente, a tupla enviada pelo *bolt Split Sentence* é recebida e processada de maneira que é criado um conjunto para cada palavra presente na base de dados. Isso evita a presença de termos duplicados. Caso a palavra já esteja no set, o ID de referência que veio na tupla é adicionado como uma lista de IDs que contém a mesma palavra. Abaixo, é ilustrado um exemplo simplificado do resultado do índice invertido das palavras ”João”e ”Silva”. Note que o termo ”joão”aparece nos registros com ID 01 e 02, já o termo ”6562348”está presente no registro com ID 01.

É possível notar a partir deste *bolt* pode-se identificar, através da simples contagem de ocorrências dos termos no índice invertido, a presença de palavras muito frequentes (conhecido como *stop word* [Manning et al. 2008]). RedBlock é definido um limiar para descartar tais termos. Por exemplo, se um termo estiver presente em mais de 50 registros este é ignorado pelo índice invertido.

Campo	Lista de IDs que contém o campo
João	01, 02
6562348	01
estudante	01,03

Tabela 6. Exemplo do *Bolt WordIndex Save* utilizando como entrada os registros da Tabela 5.

3.5. Bolt - *Pair Generator*

A função do *Pair Generator* (4) é de construir pares baseados no conjunto de palavras que foi salvo no banco de dados pelo *Bolt WordIndex Save*. A idéia é que todos os registros que compartilham uma palavra em comum devem ser comparados para verificar se representam uma duplicata. As principais funções desse *bolt* são recuperar o registro por completo (em sua totalidade) acessando o banco de dados e enviar os pares para a etapa seguinte.

Por exemplo, se o termo ”João”consta nos registros 01, 02 e 03, cada registro servirá de fonte para que o *Bolt Pair Generator* recupere o registro e envie para o próximo *bolt* os pares (1,2) e (1,3), conforme Tabela 6.

3.6. Bolt - *Pair Ranker*

O *bolt Pair Ranker* (5) tem como objetivo computar a similaridade de cada linha recebida pelo *bolt Pair Generator*, mensurando o grau de semelhança de cada atributo a partir de uma função de similaridade. As funções de similaridade utilizadas são as de *Jaccard* e *Leveshtein* [Manning et al. 2008] . Na função *Jaccard* é avaliado o nível de distância que duas *strings* se encontram. A distância *Levenshtein* entre duas *strings* é definida como o número mínimo de formatações necessárias para se transformar tal *string* na outra que está sendo comparada, sendo tais formatações representadas por inserção, substituição e eliminação de um ou mais caracteres.

3.7. Bolt - *Training Creator*

O objetivo do *bolt Training Creator* é construir um modelo de treinamento a partir do algoritmos de árvore de decisão. O modelo será utilizado posteriormente para identificar os pares duplicados e não duplicados. Note que o treinamento é configurado a partir de uma base de treinamento contendo um conjunto, de tamanho pré-definido, de pares rotulados pelo usuário. Dessa forma, identificar tais pares e rotulá-los é uma tarefa custosa que deve ser minimizada o máximo possível.

3.8. Bolt - *Decision Tree*

O *bolt Decision Tree* (6) utiliza o modelo de classificação, previamente criado, para identificar os pares como duplicatas ou não duplicatas. O *bolt* recebe como entrada os valores de similaridade de cada par e emite como saída a instância classificada. O algoritmo utilizado foi o J48.

3.9. Bolt - *Counter Bolt*

O último *bolt* presente na topologia é o *Counter Bolt* (7). Enquanto a topologia está em funcionamento, este bolt irá gerar ao usuário quantos pares Verdadeiro-Positivos,

Verdadeiro-Negativos, Falso-Positivos e quantos Falso-Negativos foram computados. Este bolt só terá funcionalidade na presença do gabarito da base de dados.

4. Trabalhos Relacionados

Um simples processo de deduplicação de dados pode ser realizado comparando todos os registros contra todos os outros presentes na base de dados. No entanto, tal método resulta em um custo quadrático de processamento, o que seria inviável no caso de uma base de dados média ou grande. Nesse contexto, a blocagem surge como uma alternativa para se reduzir o espaço de busca, somente processando os registros que possuem algum indício de representar uma duplicata.

No cenário de bases de dados incrementais, na qual os registros são inseridos ao longo do tempo, é importante que o método de deduplicação seja capaz de se autoajustar a novos padrões e reduzir ao máximo o tempo de processamento de uma nova entrada para atender a demanda online. Dessa forma, é importante que o processo de blocagem, que representa a maior fatia de processamento [Dal Bianco et al. 2015], seja suficientemente eficiente para não resultar em atrasos de processamento.

Um dos primeiros trabalhos envolvendo a deduplicação online foi proposto por [Bhattacharya and Getoor 2007], na qual é desenvolvido um método para que consultas envolvendo dados deduplicados sejam respondidas em tempo real. No método, dados são pré-processados para possibilitar a resposta em tempo real. Já no contexto da blocagem online, em [Bizer et al. 2009] é proposta uma técnica na qual as similaridades entre os atributos são pré-calculadas quando um índice invertido é criado. Dessa forma, o método depende de que boa parte dos dados estejam presentes estaticamente para que o índice seja populado apropriadamente, diferente do contexto online. Já [Ramadan et al. 2013] aprimora a estrutura de índice proposta por [Whang et al. 2009] para o seu funcionamento de forma dinâmica, também pré-calculando as similaridades entre os atributos.

Não foram identificados trabalhos que explorem plataformas distribuídas de código aberto para o processamento em tempo real, como o *Spark* ou *Apache Storm*. Tais plataformas são capazes de processar milhões de entradas por segundo, utilizando um único computador [Srikanth and Reddy 2016]. No *Apache Storm*, o fluxo de dados é definido utilizando uma estrutura de topologia para gerenciar o que deve ser processado a cada instante.

5. Experimento

Nesta seção, será descrito um experimento inicial com objetivo de avaliar se a ferramenta desenvolvida foi capaz de identificar corretamente os pares duplicados e qual a demanda de pares rotulados para configurar o método. Tal experimento é importante para identificar se a Redblock é capaz de agrupar corretamente os pares e de posteriormente construir os pares candidatos.

Para se realizar o experimento foram utilizadas métricas tradicionais. A Precisão avalia, dos pares recuperados, a taxa de pares que foram corretamente identificados. A Revocação mede a taxa de pares recuperados comparando com o total de pares duplicados que estão presentes na base de dados. Por fim, o F1 combina a precisão e a revocação em uma medida única. Além disso, foi utilizada uma base de dados sintética contendo 10.000

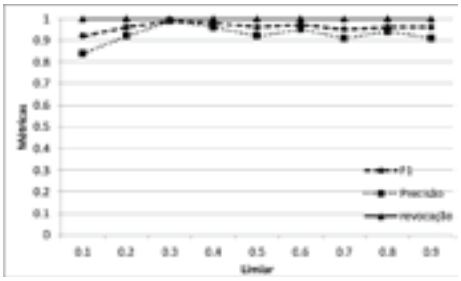

Figura 3. Experimento em uma base de dados sintética com objetivo de avaliar a precisão, revocação e o F1.

registros sendo 1.000 deles duplicatas. A base de dados foi gerada com a ferramenta Febrl [Christen 2008]. Bases sintéticas são importantes devido a possibilidade de controlar o número de pares duplicados e o total de registros que serão inseridos na base de dados.

A Figura 3 apresenta os resultados das métricas F1, precisão e revocação obtidos com a Redblock. O eixo X define o limiar que determina o tamanho do conjunto de treinamento. Por exemplo, o limiar 0.1 define um conjunto de treinamento com 10% de pares positivos (100 pares) e o mesmo número de pares negativos.

Na figura, o limiar 0.1 resultou um valor de F1 de 92%, aumentando para 0.3 o valor é melhorado em 7% atingindo um valor máximo. Percebe-se que o aumento no tamanho do treinamento impacta diretamente na precisão do método. Em outras palavras, quanto mais pares rotulados, mais preciso é o treinamento do método de classificação. É importante notar que a revocação se mantém no valor máximo em todos os limiares, demostrando que a Redblock foi capaz de encontrar todos os pares duplicados da base de dados. Os limiares acima de 0.3 resultaram em uma eficácia bastante instável, ou seja, o aumento no conjunto de treinamento não resultou em uma melhora significativa dos resultados.

Por fim, pode-se notar com essa experimentação inicial que a Redblock foi capaz de promover uma blocagem com alta eficácia, encontrando todos os pares duplicados, assim como o método de classificação, que foi capaz de selecionar boa parte dos pares duplicados[2]

6. Considerações Finais

Neste trabalho, foi proposta uma nova ferramenta para a deduplicação online com foco no processo de blocagem. A ferramenta, denominada *Redblock*, combina o *framework Apache Storm* juntamente com o banco de dados *Redis* para possibilitar um processamento massivo de dados. No experimento inicial, foi possível constatar que a *Redblock* manteve uma alta qualidade (alta eficácia), ou seja, as etapas de blocagem e a classificação foram capazes de recuperar um alto número de pares duplicados sem perdas substanciais de registros positivos.

Os próximos passos envolvem testar a ferramenta com bases de dados contendo milhões de registros para analisar sua eficiência no processamento massivo de dados.

[2]Para visualização contínua dos aprimoramentos no código, o mesmo encontra-se disponível no *Github*. Link para acesso: https://github.com/luanfelixpimentel/storminho

Dessa forma, será possível comparar com outras ferramentas no estado-da-arte da bibliografia. Além disso, será avaliado o desempenho da ferramenta quando aplicada a uma base de dados real, como por exemplo, o *twitter* que é uma ferramenta online e gera uma indeterminada quantidade de dados em tempo real.

Referências

Baxter, R., Christen, P., Churches, T., et al. (2003). A comparison of fast blocking methods for record linkage. In *ACM SIGKDD*, volume 3, pages 25–27. Citeseer.

Bhattacharya, I. and Getoor, L. (2007). Query-time entity resolution. *Journal of Artificial Intelligence Research*, 30:621–657.

Bizer, C., Heath, T., and Berners-Lee, T. (2009). Linked data-the story so far. *Semantic services, interoperability and web applications: emerging concepts*, pages 205–227.

Christen, P. (2008). Febrl: a freely available record linkage system with a graphical user interface. In *HDKM '08: Proceedings of the second Australasian workshop on Health data and knowledge management*, pages 17–25, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Christen, P. (2012). A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering*, 24(9):1537–1555.

Dal Bianco, G., Galante, R., Gonçalves, M. A., Canuto, S., and Heuser, C. A. (2015). A practical and effective sampling selection strategy for large scale deduplication. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2305–2319.

Elmagarmid, A. K., Ipeirotis, P. G., and Verykios, V. S. (2007). Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1).

Manning, C. D., Raghavan, P., Schütze, H., et al. (2008). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

Mitra, P., Kang, J., Lee, D., and On, B.-w. (2005). Comparative study of name disambiguation problem using a scalable blocking-based framework. In *Digital Libraries, 2005. JCDL'05. Proceedings of the 5th ACM/IEEE-CS Joint Conference on*, pages 344–353. IEEE.

Ramadan, B., Christen, P., Liang, H., Gayler, R. W., and Hawking, D. (2013). Dynamic similarity-aware inverted indexing for real-time entity resolution. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 47–58. Springer.

Srikanth, B. and Reddy, V. K. (2016). Efficiency of stream processing engines for processing bigdata streams. *Indian Journal of Science and Technology*, 9(14).

Whang, S. E., Menestrina, D., Koutrika, G., Theobald, M., and Garcia-Molina, H. (2009). Entity resolution with iterative blocking. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 219–232. ACM.