MÉTODOS Y HERRAMIENTAS PARA DETECCIÓN DE DUPLICADOS EN PARALELO

Víctor Guevara Valdez
e-mail: vguevar@hotmail.com
Iván Amón Uribe
e-mail: ivan.amon@upb.edu.co

RESUMEN: *La detección de duplicados busca en una o más fuentes de datos aquellas entidades que debiendo ser únicas tienen representaciones distintas. Las funciones de similitud existentes en general tienen un costo computacional alto, por lo cual se han desarrollado técnicas y herramientas para mejorar la eficiencia computacional de este proceso. En este artículo se hace un compendio de técnicas y herramientas que buscan mejorar la eficiencia del proceso utilizando procesamiento paralelo.*

PALABRAS CLAVE: detección de duplicados, procesamiento paralelo.

ABSTRACT: *duplicate detection searches one or more data sources entities that must be unique with different representations. The similarity functions existing in general have a high computational cost, which have developed techniques and tools to improve the computational efficiency of this process. This paper presents a compendium of techniques and tools that using parallel processing seek to improve the efficiency of the process.*

KEYWORDS*: record linkage, parallel record linkage*.

1 INTRODUCCIÓN

La calidad de los datos en las fuentes de datos empresariales constituye un factor clave para la eficiencia operativa, rentabilidad y sostenibilidad de las organizaciones [3], ya que los datos de calidad son imprescindibles para una acertada toma de decisiones [1]. Uno de los problemas que afecta la calidad de los datos es la existencia de duplicados conocida como *Record Linkage*, es decir, de diferentes representaciones de una misma entidad que no cuentan con un identificador único [2] [4].

Pueden ser varias las diferentes formas de representación de una entidad que dan origen a los duplicados. Ariel en [24] presenta la necesidad de unificar la información de diferentes organizaciones del sector de la salud. Las columnas altamente discriminantes que permiten identificar los diferentes pacientes, tales como número y tipo de identificación, nombres, apellidos, entre otros, pueden presentar diferencias de nomenclatura, abreviaturas, omisión de caracteres, errores ortográficos y de digitación, lo cual puede provocar una distorsión de la realidad que no permite tomar decisiones de negocio acertadas. En [41], Amón y Jiménez, comparan varias funciones de similitud bajo diferentes situaciones problemáticas como errores ortográficos y tipográficos, abreviaturas, tokens faltantes, prefijos y sufijos y espacios en blanco.

El proceso de detección de duplicados [23] hace uso de *funciones de similitud* [5]. Entre las más conocidas se encuentran la distancia de edición [13] [14] [15], acronyms [16], Guth [17], Jaccard [18], Jaro [19], Jaro Winkler [20], N-gram [21], TF-IDF [22], algoritmos fonéticos como Soundex [42] [45], PhoneticSpanish para el idioma Español [44], Doble Metaphone [43], que permiten detectar cadenas de texto similares más no idénticas, de tal forma que posteriormente se pueda ejecutar un proceso de depuración que unifique los duplicados en entidades únicas.

Cuando dos fuentes de datos **A** y **B** son comparadas para detectar duplicados, una técnica elemental consiste en comparar cada fila de **A** con cada fila de **B**, resultando un número máximo de **|A|x|B|** comparaciones entre filas (donde **|.|** denota el número de filas en la fuente de datos). De la misma forma, cuando se depura una única base de datos **A**, el número máximo de comparaciones es $|A| \times \frac{|A|-1}{2}$, porque cada fila de **A** debe compararse con el resto de filas.

Como puede verse, la cantidad de comparaciones que debe realizarse bajo este proceso puede llegar a ser muy alto, convirtiéndolo en prohibitivamente costoso computacionalmente, haciendo que este enfoque no sea factible en grandes bases de datos e incluso en bases de datos de tamaño moderado. Por ejemplo, la comparación de dos bases de datos con un millón de filas cada una, resulta en 10^{12} (un billón) de posibles comparaciones entre pares de filas.

Al mismo tiempo, asumiendo que no hay duplicados en las fuentes de datos que se van a comparar (por ejemplo, una fila de **A** aparece sólo una vez en **B** y viceversa), entonces el número máximo de verdaderos positivos corresponde a **min (|A|, |B|)**. De la misma forma, para una detección de duplicados el número de entidades únicas (y por lo tanto sus verdaderos positivos) en una base de datos es siempre menor o igual al número de registros en ella. Por consiguiente, mientras los esfuerzos computacionales de comparación de registros se incrementan cuadráticamente con el crecimiento de las fuentes de datos, el número de verdaderos positivos sólo aumenta linealmente [46]. Es evidente entonces que la gran mayoría de comparaciones serán entre registros que

corresponden a entidades diferentes pudiéndose considerar como esfuerzo improductivo.

Para mejorar el desempeño del proceso de detección de duplicados, se han desarrollado *técnicas de indexación* [6], cuyo objetivo es reducir este gran número de posibles comparaciones, reduciendo la cantidad posible de pares de filas que no corresponden a la misma entidad.

Dichas técnicas a su vez pueden ser paralelizadas, de tal forma que se pueda mejorar aún más la eficiencia del proceso, mediante el uso de la potencia de múltiples procesadores y/o de varios sistemas de cómputo de forma simultánea (procesamiento multihilo).

Al momento de la elaboración de este trabajo, aunque se conoce de artículos tipo *survey* sobre las técnicas de indexación [6] [7], no se encontró evidencia de un artículo que presente de forma unificada las técnicas y herramientas que hacen uso del procesamiento paralelo aplicado al *record linkage*, constituyéndose en el principal aporte de este artículo el poder encontrar en un solo sitio la información sobre técnicas y herramientas existentes para mejorar la eficiencia computacional en la detección de duplicados utilizando procesamiento paralelo, que se presentan de forma dispersa en la literatura, reduciendo así sustancialmente el tiempo requerido por los investigadores interesados en la temática.

Este artículo está organizado como sigue: en la sección 2 se mencionan las distintas técnicas de indexación de forma breve por no ser el foco de este trabajo. En la sección 3 se presentan diferentes técnicas de procesamiento paralelo para la detección de duplicados. En la sección 4 se presentan algunas herramientas que utilizan procesamiento paralelo para la detección de duplicados. Por último, en la sección 5, se presentan las conclusiones y posibles trabajos futuros.

2 TÉCNICAS DE INDEXACIÓN PARA DETECCIÓN DE DUPLICADOS

El objetivo de la etapa de indexación es reducir el número de posibles comparaciones, reduciendo la mayor cantidad posible de pares de registros que no corresponden a la misma entidad. A continuación se presentan las técnicas de indexación más conocidas.

2.1 Bloqueo

Consiste en subdividir la fuente de datos original en subconjuntos mutuamente excluyentes (bloques), utilizando campos altamente discriminantes (por ejemplo el nombre completo, la fecha de nacimiento, entre otros), bajo el supuesto de que no se produzcan coincidencias entre bloques distintos. Por lo tanto, con el fin de localizar duplicados, es suficiente comparar sólo los registros de un mismo bloque, sin tener que comparar con los elementos de otros bloques reduciendo considerablemente la cantidad de comparaciones a realizar.

Aunque esta técnica puede aumentar sustancialmente la velocidad del proceso [25], los errores cometidos durante la etapa de armado de bloques pueden conducir a la detección de falsos duplicados o a la pérdida de duplicados reales [7].

La etapa de armado de bloques utiliza un criterio de ablocamiento, el cual está basado en una columna (campo) o en la concatenación de varios campos o columnas. Un criterio importante para un buen ablocamiento es poder agrupar valores similares dentro del mismo bloque. La similitud depende de las características de los datos a procesar. Para cadenas que contienen nombres propios, por ejemplo, puede usarse similitud fonética con funciones como Soundex [42], NYSIIS [47], Doble-Metaphone [43], PhoneticSpanish [44].

2.2 Enfoque de barrio ordenado

Consiste en crear una clave para cada registro mediante la extracción de campos. Por ejemplo, en una lista de nombres propios la clave de registro puede ser el apellido. Luego se ordenan los datos por medio de esta clave y finalmente una ventana de tamaño fijo se mueve a través de la lista con el fin de limitar la búsqueda a los registros ubicados dentro de la ventana. Esta técnica se basa en el supuesto de que los duplicados estarán cercanos dentro de la ventana y su eficacia depende de la clave que se seleccione [26].

2.3 Indexación basada en q-gram

El objetivo de esta técnica es indexar las fuentes de datos de tal forma que los registros que son similares más no idénticos sean insertados dentro del mismo bloque. Asumiendo que las claves son cadenas de texto, la idea es crear variaciones en cada clave usando q-grams (sub cadenas de longitud q) e insertar identificadores de registros en más de un bloque [27]. La Tabla 1 muestra un ejemplo de indexación con bi-grams presentado en [6], el cual utiliza el apellido como clave:

Tabla 1: Ejemplo de indexación utilizando bi-grams.

Fila	Clave (Apellido)	Lista de bi-grams	Claves indexadas
1	Smith	[sm,mi,it,th], [mi,it,th], [sm,it,th], [sm,mi,th], [sm,mi,it]	**smmiitth**, miitth, smitth, smmith, smmiit
2	Smithy	[sm,mi,it,th,hy], [mi,it,th,hy], [sm,it,th,hy], [sm,mi,th,hy], [sm,mi,it,hy], [sm,mi,it,th]	smmiitthhy, miitthhy, smitthhy, smmithhy, smmiithy, **smmiitth**

3	Smithe	[sm,mi,it,th,he], [mi,it,th,he], [sm,it,th,he], [sm,mi,th,he], [sm,mi,it,he], [sm,mi,it,th]	smmiiitthhe, miitthhe, smitthhe, smmithhe, smmiithe, **smmiitth**

Fuente: Christen, P., *"A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication"*, 2012.

Como puede verse la clave *smmiitth* está presente en los identificadores 1,2 y 3 y por tanto estos serían candidatos a duplicados.

2.4 Indexación basada en arreglos de sufijos

Esta técnica fue propuesta como un enfoque eficiente de integración de información proveniente de múltiples fuentes. La idea es insertar las claves y sus sufijos en un arreglo basado en un índice invertido. Un arreglo de sufijos tiene cadenas o secuencias y sus sufijos ordenados alfabéticamente. La indexación basada en arreglos de sufijos fue usada exitosamente en bases de datos bibliográficas inglesas y japonesas [28]. La tabla 2 muestra un ejemplo de indexación basada en arreglos de sufijos presentada en [6]:

Tabla 2: Ejemplo de indexación basada en arreglos de sufijos

Fila	Clave (nombre)	Sufijos
1	Catherine	catherine, atherine, therine, herine, rine
2	Katherina	katherina, atherina, therina, herina, erina, **rina**
3	Catherina	catherina, atherina, therina, **rina**
4	Catrina	catrina, atrina, trina, **rina**
5	Katrina	katrina, atrina, trina, **rina**

Fuente: Christen, P., *"A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication"*, 2012.

El sufijo *rina* al está presente en los identificadores 2, 3, 4 y 5 y por tanto estos serían candidatos a duplicados.

2.5 Agrupaciones de campana

Esta técnica de indexación se basa en la idea de usar un enfoque de agrupamiento computacional barato para crear agrupamientos superpuestos de alta dimensionalidad, desde los cuales se pueden generar pares de registros candidatos. Los grupos (*clusters*) son creados calculando la similitud entre claves utilizando medidas tales como Jaccard TF-IDF/coseno. Ambas medidas están basadas en tokens, los cuales pueden ser caracteres, q-grams o palabras. Estos pueden ser implementados eficientemente usando un índice invertido en el cual los tokens son los índices llave en lugar de las claves actuales [29].

3 TÉCNICAS DE PROCESAMIENTO PARALELO

3.1 Algoritmo de dos fases usando cálculo rápido de distancia de edición

Este enfoque emplea como base algoritmos de minería de datos para realizar agrupamiento jerárquico. Una idea clave es la clasificación *radix* (ordenamiento de raíz [51]) en ciertos atributos para eliminar registros idénticos antes de cualquier proceso posterior, es decir, armando para cada registro una clave utilizando columnas altamente discriminantes. Otra idea novedosa es la de formar un gráfico que relaciona registros similares y así encontrar los componentes conectados [9].

El algoritmo secuencial básico consiste en reunir todos los registros de todos los conjuntos en una lista única X, ordenarlos por la clave, eliminar los registros idénticos por clave, aplicar bloqueo sobre el conjunto resultante y eliminar los duplicados con el criterio seleccionado.

El algoritmo multihilo consiste en procesar paralelamente distintas porciones del conjunto X.

ALGORITMO SECUENCIAL

1. Reunir todos los registros de todos los conjuntos de datos en una única lista X.
2. Ordenar los registros en X y formar grupos tales que cada grupo conste de claves idénticas. Elegir un registro de cada grupo tal que se obtenga un conjunto de registros resultante X'.
3. Hacer bloqueo en X', específicamente, podría existir un bloque por cada posible l.mer (subcadena de longitud m, hay 26^l posibles l-mer para el idioma inglés [38]). Por ejemplo, si se define un l-mer "y" y dos registros tienen "y" como l-mer en su apellido, estos dos registros estarán en el bloque correspondiente a "y".
4. Agrupar cada bloque empleando agrupamiento jerárquico con vinculación singular. Específicamente, dos registros r_1 y r_2 se llevarán al mismo bloque si la distancia entre ellos no es mayor a t (donde t es el umbral especificado). Se utiliza un algoritmo rápido de distancia de edición entre dos registros [39].
5. Se genera una gráfica G(V, E) donde V es la colección X'. Dos registros tienen un borde entre ellos si existe por lo menos una agrupación en al menos un bloque al que ambos registros pertenecen.

6. Buscar los componentes conectados de G(V, E).
7. Registrar en la salida cada componente conectado como un grupo.

ALGORITMO PARALELO

1. El procesador maestro emite todos los registros de entrada a los procesadores esclavos.
2. Cada procesador ordena una porción de X en paralelo. Específicamente, los registros de X son agrupados con base en los dos primeros caracteres de los apellidos. Hay 262 posibles 2-mer (subcadenas de longitud 2), por lo tanto no son muchos grupos posibles (algunos de los cuales podrían estar vacíos). Cada procesador ordena 262/p grupos. Como subproducto de esta clasificación, cada procesador escoge un representante de cada grupo de registros idénticos que son ordenados. En otras palabras, se forma X´. Los procesadores esclavos informan al procesador maestro acerca de los resultados.
3. El maestro asigna |X´|/p número de registros de X´ a cada procesador para hacer bloqueo. Cada procesador hace bloqueo sobre estos registros y envía información de los bloques al procesador maestro.
4. El procesador maestro agrega los bloques. Por ejemplo, si "*y*" es un l-mer, partes del bloque correspondientes a "*y*" podrían estar en múltiples procesadores. El procesador maestro fusiona estos bloques parciales.
5. Sean B_1, B_2,…, B_t los bloques en X´. Tener en cuenta que t <= 26l, donde l es el tamaño del bloque. Sea $n_i=|B_i|$, para $1 \le i \le t$. el maestro ordena $n_1^2, n_2^2, ..., n_t^2$ valores en orden descendente. Sea $s = \sum_{i=1}^{t} n_i^2$. El maestro distribuye los bloques entre los procesadores de tal forma que el trabajo asignado a cada procesador sea parecido. Específicamente, la distribución es tal que la suma de los cuadrados de los tamaños de los bloques asignados a cada procesador sea cercana a *s/p*.
6. La tarea siguiente es generar el gráfico G(V, E). Para esto, cada procesador encuentra los bordes en los bloques a lo largo de la misma línea como en el algoritmo secuencial. Todos los bordes de todos los procesadores son enviados al maestro.
7. El maestro encuentra los componentes conectados en la gráfica. Estos componentes conectados, junto con las copias de los registros eliminados al principio, nos dan los grupos de interés.

3.2 MapReduce

Consiste en utilizar el paradigma de programación *MapReduce* [50] con el nuevo algoritmo desarrollado en esta técnica, llamado RLP [10], para distribuir dinámicamente la carga de trabajo del proceso de detección de duplicados, de forma paralela entre una red de equipos en ambiente Hadoop [48]. A cada equipo se le asigna una determinada cantidad de bloques de datos previamente construidos con el método de bloqueo para que los procese simultáneamente en multihilo con *MapReduce*. A su vez, cada bloque es particionado en subconjuntos que son procesados de forma independiente en Hadoop.

El proceso consta de dos fases: Equilibrio de sesgo en el bloqueo basado en el paralelismo y desplazamiento de paralelismo basado en coincidencia .

Paso 1: Equilibrio de sesgo en el bloqueo basado en el paralelismo

El objetivo general es reducir los efectos del sesgo de datos en tiempo de ejecución, es decir, la posible tendencia a que se presente desequilibrio en las particiones, quedando algunas cargadas con demasiados datos y otras con pocos. En la primera pasada, esto se hace equilibrando el número de comparaciones dadas a cada partición de datos en el algoritmo de particionamiento RLP, propio de esta técnica. Desde cada tarea reducida se asigna una partición separada, lo cual permite equilibrar la carga de trabajo computacional de la tarea, asegurando con ello tareas terminadas en aproximadamente el mismo tiempo.

Paso 2: Desplazamiento de paralelismo basado en coincidencia

El objetivo es asegurar la detección de los duplicados que se crucen desde diferentes particiones. El sistema reduce el tiempo de ejecución a través de un enfoque de "divide y vencerás" que aumenta la utilización de las máquinas disponibles en Hadoop.

3.3 Algoritmo FERAPARDA para detección de duplicados en paralelo

Algoritmo que utiliza relacionamiento probabilístico con procesamiento paralelo de bloques para detección de duplicados [12].

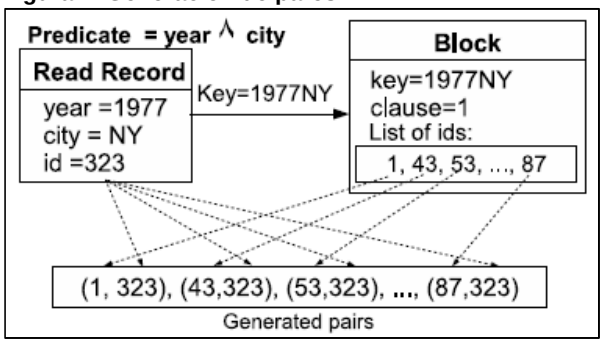
Este algoritmo ha sido utilizado exitosamente para detectar duplicados en conjuntos de datos sintéticos con más de 1 millón de registros en 7 minutos, con un grupo de 20 ordenadores, logrando una aceleración casi lineal y proporcionando una mejor escalabilidad y eficiencia que otras soluciones como Febrl [4], p-DC, p-DD1, P-DD2 [6] y P-Swoosh [7].

La estrategia de paralelización está basada en 4 filtros: *CompareRecord*, Bloqueo, Clasificador y Fusión.

El filtro ***CompareRecord*** lee cada registro del conjunto de datos, asigna un identificador al registro y genera una llave de bloque por cada conjunción en el predicado de bloque. Un predicado de bloque se entiende como una disyunción de conjunciones. Cada parte de una conjunción define una función de transformación sobre uno o más atributos del registro. Un ejemplo de predicado es *P = (nombre ∧ año de nacimiento) ∪ (apellido ∧ país)*. Cuando se aplica a un registro, el predicado de bloque generará una clave de bloque para cada conjunto. Las comparaciones sólo se ejecutan para los registros con la misma clave de bloque. El identificador de registro se genera de tal forma que se pueda identificar qué instancia de filtro lo leyó, lo que es necesario para instancias posteriores del algoritmo [52].

Una vez generada la clave de bloque, se envía al filtro de **Bloqueo** junto al identificador de registro y al identificador de conjunción. Esta comunicación emplea una cadena de marcado basada en la clave de bloqueo, de modo que cada mensaje con la misma clave de bloqueo será enviado a la instancia apropiada del filtro de bloqueo. El filtro de bloqueo mantiene una lista de identificadores de registro para todos los registros que generan la misma llave de bloqueo. Cuando un nuevo mensaje llega, el filtro de bloqueo identifica si tiene que crear un nuevo bloque o simplemente añadir a uno ya existente. Además, durante la recepción del mensaje, el filtro de bloqueo genera un par de identificadores de registro formados por el identificador del mensaje y cada identificador presente en la lista, como se puede verse en la Figura 2 [52].

Figura 2. Generación de pares.



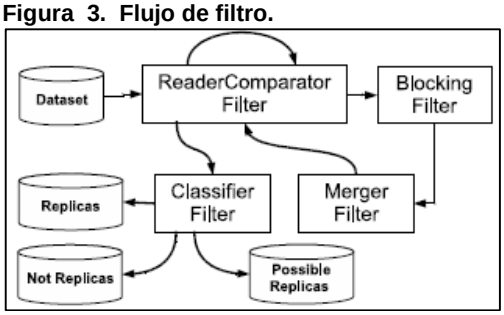
Fuente: Santos et. al., "A Scalable Parallel Deduplication Algorithm", 2007.

Conjunciones de predicados pueden superponerse y generar pares redundantes, lo que resultaría en el procesamiento redundante. Para evitar esto, se emplea un filtro de **Fusión**, lo que elimina pares redundantes. Para identificar pares redundantes, el filtro de fusión mantiene una tabla *hash* definida por la combinación de los identificadores, donde el más pequeño se utiliza siempre primero. Una optimización interesante en este filtro surge del hecho de que la generación de identificador de registro es siempre creciente, por lo tanto, después de un tiempo, ningún par estará más asociado con ese identificador. En los primeros experimentos, cuando se estaban manteniendo todos los pares en este filtro, la utilización de la memoria fue alta. Ahora, se define una lista circular que guarda las llaves hash y limita el tamaño de la tabla hash, mientras que garantiza el acceso a todas las llaves pertinentes.

Todas las parejas que nunca han sido procesadas son enviadas nuevamente al filtro CompareRecord. El filtro de Fusión utiliza secuencias basadas en el identificador de registro más grande etiquetado. Se debe recordar que, a través de un identificador, es posible saber qué instancia de *CompareRecord* lo generó. Se Utiliza siempre el identificador más grande porque se asume que el filtro *CompareRecord* que lo generó será capaz de comparar el par. Se debe tener en cuenta que el filtro de Fusión siempre enviará un par a una instancia *CompareRecord* que tiene al menos uno de los registros.

La Figura 3 muestra este bucle. Se decide implementar Lector y Comparador como el mismo filtro, ya que durante la etapa de comparación se tienen todos los registros necesarios en la memoria [53].

Figura 3. Flujo de filtro.



Fuente: Santos et. al., "A Scalable Parallel Deduplication Algorithm", 2007.

Es importante darse cuenta de que no se podía encontrar una estrategia eficaz que produjera una partición arbitraria perfecta ordenada de un conjunto de datos que garantizara un equilibrio de carga entre los filtros *CompareRecord*. Por lo tanto, un par de registros a comparar pueden estar en cualquiera de los casos, y no se sabe de antemano si es posible explotar cualquier localidad de referencia. Cuando un *CompareRecord* recibe un mensaje de tipo de comparación desde el filtro de Fusión, comprueba si tiene dos registros en la partición de datos. Si este es el caso, los compara y envía un mensaje con el resultado de la comparación al filtro **Clasificador**. Cuando una instancia de *CompareRecord* tiene sólo uno de los registros, tiene que enviar su registro a la instancia que es propietaria de la otra, que, por supuesto, implica la comunicación. Teniendo esto en la mente, se han hecho varias optimizaciones para reducir la comunicación. Se definen dos nuevos tipos de mensajes: *ReceiveAndCompareRecord* (RCRmsg) y *CompareAlreadyReceivedRecord* (CRRmsg). *ReceiveAndCompareRecord* lleva el registro completo y no reduce los costos por sí mismo. Sin embargo, después de que un registro ha sido enviado a otra instancia CompareRecord, nunca será enviado de nuevo a esa instancia. Cada vez que un nuevo par llega a una

instancia de CompareRecord y no tiene ambos registros, se comprueba si ya se ha enviado su registro a la otra instancia. Si este es el caso, envía un mensaje de CompareAlreadyReceivedRecord. Por último, hay otra optimización: si una instancia de ReaderComparator recibe un par, que tiene sólo uno de los registros, y el otro es recibido en su caché, no es necesaria la comunicación y el resultado de la comparación se envía al clasificador.

El último filtro es el **clasificador**. Se clasificará el par de registros de acuerdo con el resultado del filtro CompareRecord y dice si el par es una conicidencia o no o si es una posible coincidencia (necesita supervisión humana).

3.4 Fusión paralela p-DC, p-DD1 y P-DD2

Este enfoque [30] busca solucionar el problema de tener que repetir comparaciones de registros que ya han sido fusionados, es decir, cuando se ha determinado que dos registros corresponden a la misma entidad y se fusionan en un único registro, este nuevo registro fusionado debería ser nuevamente comparado con el resto de registros del conjunto.

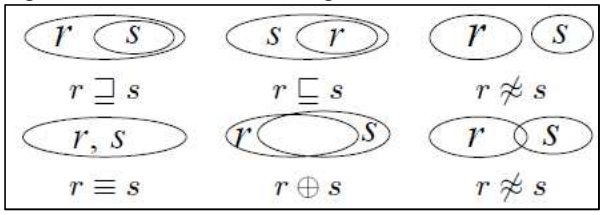
Para lograr este objetivo se contempla la explotación de las características de tres escenarios para lograr una buena paralelización: dos conjuntos limpios, sólo uno limpio o ambos "sucios".

Cuando dos registros, *r* y *s*, se refieren a la misma entidad del mundo real, son coincidentes y se denota como $r \approx s$ (de otra forma $r \not\approx s$).

Cuando una colección A no tiene coincidencias en ella, se llama limpia. De lo contrario se llama sucia. Esto es, *(1) A es limpia sii* $\forall r, s \in A, r \not\approx s$, *y (2) A es sucio sii* $\exists r, s \in A, r \approx s$.

Cuando dos registros *r* y *s* son coincidentes, cuatro relaciones pueden ocurrir, como se muestra en la figura 4: (1) $r \sqsupseteq s$: toda la información de *s* aparece en *r*, (2) $r \sqsubseteq s$: toda la información de *r* aparece en *s*, (3) $r \equiv s$: la información de *r* y *s* es idéntica, y (4) $r \oplus s$: ni (1) ni (2), pero la superposición de la información de *r* y *s* está más allá de un umbral θ.

Figura 4. Relaciones entre registros.



Fuente: Kawai et. al., "*P-swoosh: Parallel algorithm for generic entity resolution*", 2006.

El mejor de los escenarios es comparar dos conjuntos limpios, ya que el número de comparaciones se reduce al máximo. Considérese cuatro registros: a_i, a_l en *A* $(i < l \leq m)$, b_j, b_k en *B* $(j < k \leq n)$, y dos conjuntos *E* (para mantener una instancia de dos registros idénticos) y *C* (para mantener uniones c_{ij} de a_i y b_j). Entonces, si:

- $a_i \neq b_j$: proceder a la siguiente comparación (a_i, b_{j+1}).
- $a_i \equiv b_j$: agregar a_i a *E*, luego quitar a_i de *A* y b_j de *B*. Como *A* y *B* son limpios, no se necesitan comparaciones (a_i, b_k) ni (a_l, b_j). Al final, por lo tanto, se ahorran $m - i + n - j$ operaciones de comparación. Proceder a la siguiente comparación (a_{i+1}, b_1).
- $a_i \sqsubseteq b_j$: remover a_i de *A*. Se puede omitir la comparación (a_i, b_k). Proceder a la siguiente comparación (a_{i+1}, b_1).
- $a_i \sqsupseteq b_j$: quitar b_j de *B*. No se puede omitir la comparación (a_i, b_k). Proceder a la siguiente comparación (a_{i+1}, b_1).
- $a_i \oplus b_j$: quitar a_i de *A* y b_j de *B*. Agregar c_{ij} (unión de a_i, b_j) a *C*. Proceder a la siguiente comparación (a_i, b_{j+1}).

Se asume que se cuenta con *k* procesadores. Los conjuntos de datos se subdividen en *k* subconjuntos y cada subconjunto se asigna a un procesador. El objetivo de la paralelización es ejecutar simultáneamente la depuración en los *k* procesadores para mejorar el rendimiento.

Se depura cada conjunto de datos de tal forma que se obtenga un conjunto limpio y al final se logran hacer comparaciones sólo entre conjuntos de este tipo.

3.5 P-swoosh: algoritmo paralelo para la resolución de entidad genérica

El algoritmo paralelo P-Swoosh [31] utiliza funciones genéricas de detección de duplicados y permite el balanceo de carga a través de múltiples procesadores. Los resultados de la evaluación del algoritmo demuestran una escalabilidad casi lineal de 2 a 15 procesadores.

Se tiene un conjunto de registros $R = \{r_1, ..., r_n\}$ para resolver. Si $r_i \approx r_j$, una función de unión genera $\langle r_i, r_j \rangle$, el compuesto de r_i y r_j.

Se tienen dos funciones: una "responsable" encargada de determinar en qué procesador se compara cada par de registros y otra de "alcance" encargada de asignar registros a cada procesador.

En lugar de dividir el conjunto inicial de registros *R*, el enfoque de paralelismo divide el conjunto de no coincidencias *R'*, asignando *R* a un nodo maestro que genera *R'*, para distribuirlo en *p* nodos esclavos que evalúan todos los posibles pares de registros y generan tablas T_p en las cuales aparecen las coincidencias y los registros para los cuales no se encontraron coincidencias, de tal forma que estos últimos se asignen nuevamente a un nodo maestro que los subdivide para aplicarles nuevamente el proceso.

En este algoritmo paralelo, es importante el balanceo de carga para una utilización efectiva de los procesadores. El algoritmo P-Swoosh puede explotar dos mecanismos de balanceo de carga para distribuir los procesos de comparación equilibradamente a múltiples nodos. Uno es "balanceo de carga horizontal", el cual balancea la carga de trabajo entre nodos esclavos; el otro es "balanceo de carga vertical", el cual balancea la carga de trabajo entre nodos maestros y esclavos.

3.6 Enfoque de procesamiento paralelo con múltiples procesadores

Este enfoque llamado MD [32], combina un método de bloqueo eficiente con un modelo robusto de programación en paralelo. La fase de bloqueo genera grandes bloques con registros que tienen bajo grado de similitud. De dichos bloques se generan segmentos más precisos. La programación en paralelo se utiliza para resolver estas fases y disminuir el número de operaciones. Este enfoque ha demostrado ser dos veces más rápido que BTO-TK [9], que es una solución escalable para eliminación de duplicados en paralelo en entorno distribuido. Los pasos que se siguen en este enfoque son Paso de dos bloqueos y fusión en paralelo:

Paso 1: Paso de dos bloqueos

Se segmenta el conjunto de datos buscando balancear el tamaño de los bloques. La creación de bloques balanceados puede determinar el desempeño de la aplicación. En la figura 5 se ilustra la creación de bloques balanceados.

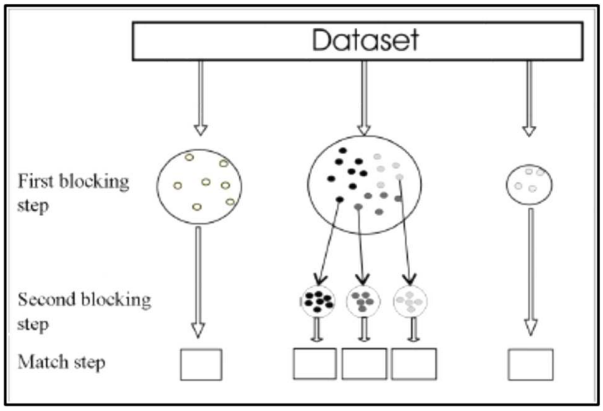
El primer paso de bloqueo maximiza la generación de pares candidatos. Algunos bloques pueden ser mucho más grandes que otros, lo que puede resultar en un bloqueo desbalanceado. Los bloques balanceados se envían a la fase de comparación. El segundo paso de bloqueo divide los bloques desbalanceados utilizando llaves más específicas. La salida de este paso se envía al paso de comparación.

Paso 2: Fusión en paralelo

En la figura 6 se ilustra el enfoque *map reduce* [49] que se utiliza en este paso.

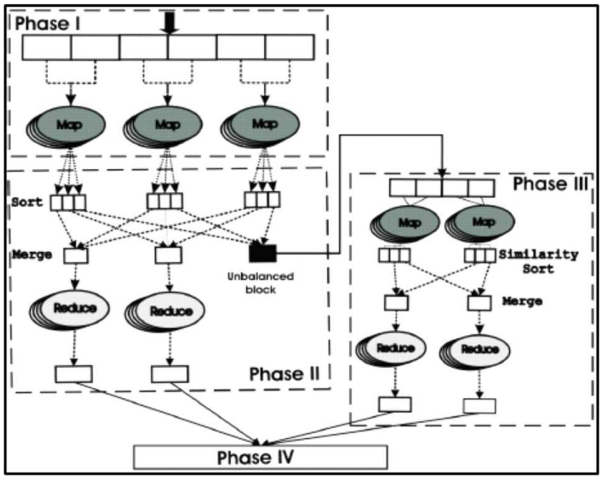
La fusión en paralelo a su vez consta de cuatro fases:

Figura 5. Descripción general del método de la etapa de dos bloqueos.



Fuente: Dal Bianco et. al., "*A fast approach for parallel deduplication on multicore processors*", 2011.

Figura 6. Secuencia de las fases del enfoque MD.



Fuente: Dal Bianco et. al., "*A fast approach for parallel deduplication on multicore processors*", 2011.

- *Fase 1*: El conjunto de datos es fragmentado secuencialmente. Cada fragmento es procesado en paralelo. Al mismo tiempo, las llaves del primer paso de bloqueo son creadas. Todas las fases siguientes son procesadas en paralelo, siguiendo el modelo *MapReduce* (MR) [40].

- *Fase 2:* los bloques son fusionados y los bloques desbalanceados son enviados a la fase 3. Los registros en bloques balanceados son comparados. Los pares identificados como duplicados son enviados a la fase 4.

- *Fase 3*: Los bloques desbalanceados son fragmentados en sub bloques. La función de bloqueo es identificada y la llave es creada usando el segundo paso de bloqueo. La función de ventana corrediza es usada para bloquear llaves altamente similares. Los registros al interior de los sub

bloques son comparados y los duplicados son enviados a la siguiente fase.

- *Fase 4*: los pares son fusionados en el archivo de salida y los duplicados son removidos.

4 HERRAMIENTAS PARA PROCESAMIENTO PARALELO

4.1 Febrl (Freely Extensible Biomedical Record Linkage)

Febrl [11] es un sistema abierto con interfaz gráfica de usuario que utiliza el enfoque de estandarización y fusión con procesamiento paralelo [8] y relacionamiento probabilístico [35], y que tiene como objetivo mejorar la depuración, normalización, eliminación de duplicados y fusión de los datos, haciendo énfasis en bases de datos de salud.

Está implementado en Phyton, lenguaje de programación abierto orientado a objetos que está disponible para la mayoría de plataformas y sistemas operativos.

Dispone de una interfaz que mejora la accesibilidad de usuarios no técnicos en detección de duplicados y que les permite seleccionar los métodos y sus correspondientes parámetros para todos los pasos del proceso: inicialización, exploración, limpieza y estandarización, definición de indexación (bloqueo), comparación, vector de clasificación por peso, procesamiento de archivos de salida de pares, evaluación y revisión "clerical", que consiste en determinar cuáles de los prospectos de duplicados efectivamente lo son, y generación de log. La figura 7 ilustra el proceso de vinculación de registros.

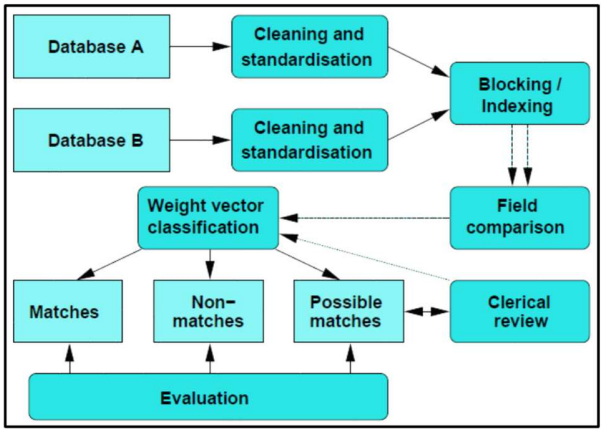
Las salidas de la etapa de bloqueo son pares de registros candidatos, mientras la etapa de comparación produce vectores con pesos numéricos similares, que luego se clasifican en coincidencias, no coincidencias y posibles coincidencias.

Esta herramienta está disponible en http://sourceforge.net/projects/febrl.

4.2 BTO-TK solución escalable eliminación de duplicados en paralelo en entorno distribuido

En esta herramienta [33] se lleva a cabo de manera eficiente en paralelo la detección de duplicados usando el popular *framework MapReduce*. Se toma como entrada un conjunto de registros y como salida otro conjunto de registros fusionados sobre la base de una condición de similitud. Se distribuyen eficientemente los datos a través de los nodos con el fin de equilibrar la carga de trabajo y minimizar la necesidad de replicación.

Figura 7. Proceso general de vinculación de registros.



Fuente: Peter Christen, "*Febrl – A Freely Available Record Linkage System with a Graphical User Interface*", Department of Computer Science, The Australian National University, 2008.

Se controla cuidadosamente la cantidad de datos almacenados en la memoria principal en cada nodo. También se proponen soluciones para el caso en que los datos no quepan en la memoria principal de un nodo cuando se utiliza un particionamiento más fino. Los autores realizaron extensos experimentos en conjuntos de datos reales, evaluando las propiedades de *speedup* y *scaleup* de los algoritmos propuestos utilizando *Hadoop* [48].

En MapReduce, los datos los son particionados a través de los nodos de un grupo y almacenados en un sistema de archivos distribuido. El algoritmo decide qué datos deben ser particionados y replicados.

Algoritmos de similitud eficientes de unión basados en filtros efectivos pueden disminuir el número de pares candidatos a ser comparados. Se generan identificadores de registros y se procede a la unión de los mismos, sin tener que cargar todos los prospectos en memoria, sólo los necesarios dependiendo del fragmento que corresponda al rango de longitud necesaria.

Los algoritmos están disponibles en: http://asterixdb.ics.uci.edu/fuzzyjoin/.

4.3 BigMatch Software

BigMatch [34] es el programa de detección de duplicados utilizado por la Oficina del Censo de Estados Unidos. Se basa en técnicas de bloqueo que permiten identificar posibles coincidencias entre los registros de grandes conjuntos de datos. El objetivo principal de *BigMatch* es generar un conjunto de pares candidatos que deben ser procesados por algoritmos más sofisticados de detección de duplicados.

El propósito del programa *BigMatch* es extraer subarchivos de los registros de las posibles parejas desde un archivo muy grande al hacer una sola pasada

secuencial a través de él. El archivo grande no tiene que ser resuelto. Supóngase que se quiere encontrar registros coincidentes en un archivo maestro muy grande A, que corresponden a los registros en un archivo de tamaño moderado B. Se define un archivo para ser de tamaño moderado si puede encajar cómodamente en la memoria principal. El programa *BigMatch* permite al usuario especificar varios criterios de bloqueo y los parámetros de campo correspondientes. El programa lee el archivo B y crea una tabla de las claves para cada criterio de bloqueo.

Entonces, el programa procede a leer en los registros del archivo de gran tamaño A. Para cada criterio de bloqueo, el programa determina si hay algunos registros del archivo B, que tienen llaves que coinciden con la clave del registro de archivo A. Para todos los registros B con valor clave coincidente, se calcula un peso de comparación correspondiente. Si alguna de las comparaciones de pesos excede un valor límite, el registro A se escribe en un subarchivo correspondiente a este criterio de bloqueo.

Aunque el software *BigMatch* ejecuta en paralelo 10 hilos de bloqueo, es casi tan rápido como un programa clásico que sólo hace una sola pasada. Según el autor procesa 300.000 pares por segundo. Se ahorra el tiempo de procesador de múltiples tipos de archivo de gran tamaño que pueden contener mil millones de registros o más. Un archivo de miles de millones de registros en una máquina excepcionalmente rápida puede tardar más de 8 horas. El mayor ahorro es a menudo la reducción de la intervención calificada de los programadores que deben realizar un seguimiento de un gran número de archivos, hacer varias corridas, y juntar la información a través de múltiples ejecuciones [34].

Más información sobre el software se puede obtener de
william.e.yancey@census.gov o
william.e.winkler@census.gov.

5 CONCLUSIONES

Se ha hecho un compendio de las técnicas y herramientas que aparecen de forma aislada en la literatura y que contemplan el procesamiento paralelo en la detección de duplicados para mejorar la eficiencia computacional de este proceso.

En la mayoría de los casos se utiliza la técnica de bloqueo para reducir al máximo el número de comparaciones. El procesamiento multihilo permite paralelizar tanto la fase de bloqueo como la fase de vinculación de registros, fragmentando el conjunto de datos inicial en unidades más pequeñas que se pueden procesar simultáneamente. Esto le permite al proceso de detección de duplicados operar con mayor velocidad en sistemas de cómputo con múltiples procesadores o a través de un grupo de máquinas.

Aunque existen más herramientas comerciales como *AutoMatch* (IBM), GRLS (*Generalized Record Linkage System*, Oracle), LinkageWiz, RELAIS (*Record Linkage At Istat*), DataFlux (SAS) [54], The Link King (SAS), Trillium Software, Link Plus, no se evidenció documentación en la cual se trate específicamente acerca del uso del paralelismo en dichas herramientas.

Como trabajos futuros se plantea realizar un comparativo entre algunas de estas herramientas que utilizan procesamiento paralelo, utilizando como parámetro la eficiencia computacional.

6 REFERENCIAS

[1] Herzog, T.N., Scheuren, F. J., and Winkler, W. E., *"Data Quality and Record Linkage Techniques"*. New York: Springer, 2007.

[2] Winkler, W. E.,,*"Matching and record linkage"*, B. G. Cox et al. (ed.) Business Survey Methods. New York: Wiley, 1995.

[3] Winkler, W. E., *"Methods for evaluating and creating data quality"*, Information Systems 29(7) 531–550. 2004.

[4] Köpcke, H. and Rahm, E., *"Frameworks for entity matching: a comparison"*, Data & Knowledge Engineering 69(2) 197–210, 2010.

[5] Dewendra Bharambe, Susheel Jain, Anurag Jain, *"A Survey: Detection of Duplicate Record"*, 2012.

[6] Christen, P., *"A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication"*, 2012.

[7] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, Vassilios S. Verykios, *"Duplicate record detection: A survey",* 2007.

[8] Peter Christen, Markus Hegland, Stephen Roberts and Ole M. Nielsen, Tim Churches and Kim Lim, *"Parallel computing techniques for high-performance probabilistic record linkage",* 2002.

[9] Abdullah-Al Mamun, Tian Mi, Robert Aseltine, Sanguthevar Rajasekaran, *"Efficient sequential and parallel algorithms for record linkage",* 2013.

[10] *Huang Yipeng, "Record Linkage in a Hadoop Environment", Columbia University, Department of Computer Science. School of Computing,* 2011.

[11] Peter Christen, "*Febrl – A Freely Available Record Linkage System with a Graphical User Interface*", Department of Computer Science, The Australian National University, 2008.

[12] Walter Santos, Thiago Teixeira, Carla Machado, Wagner Meira Jr., Altigran S. Da Silva, Renato Ferreira, Dorgival Guedes, "*A Scalable Parallel Deduplication Algorithm*", 2007.

[13] Patrick A. V. Hall, Geoff R. Dowling, "*Approximate string matching*", 1980.

[14] Rishin Haldar and Debajyoti Mukhopadhyay, "*Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach*", 2011.

[15] V. I. Levenshtein, "*Binary codes capable of correcting deletions, insertions and reversals*", 1966.

[16] Dorneles et al., "*Measuring Similarity Between Collection of Values*", 2004.

[17] Guth, G. J., "*Surname spellings and computerized record linkage*", 1976.

[18] Jaccard P., "*The distribution of flora in the alpine zone*", 1912.

[19] Jaro, M., "*Advances in record linking methodology as applied to the 1985 census of Tampa Florida*", 1989.

[20] Winkler,W., "*The state of record linkage and current research problems*", 1999.

[21] Navarro, G., Baeza-Yates, R., Sutinen, E., & Tarhio, J., "*Indexing methods for approximate string matching*", 2001.
[22] Salton, G., & McGill, M. J., "*Introduction to modern information retrieval*", 1983.
[23] Winkler,W., *"Overview of Record Linkage and Current Research Directions"*, 2006.
[24] Ariel, A. et al., *"Record Linkage in Health Data: a simulation study"*, 2014.
[25] Shin J., "*Comparative Study On Blocking Methods In Record Linkage*", 2009.
[26] Yan S. et al., "*Adaptive Sorted Neighborhood Methods for Efficient Record Linkage*", 2007.
[27] Ukkonen, E., "*Approximate string matching with q-grams and maximal matches*", 1992.
[28] de Vries T., Ke H., Chawla, S., Christen, P., "*Robust Record Linkage Blocking using Suffix Arrays*", 2011.
[29] Baxter R., Christen P., Churches T., "*A Comparison of Fast Blocking Methods for Record Linkage*", 2010.
[30] *Kim, H.-S., and Lee, D.,* "*Parallel Linkage*", 2007.

[31] H. Kawai, H. Garcia-Molina, O. Benjelloun, D. Menestrina, E. Whang, and H. Gong, "*P-swoosh: Parallel algorithm for generic entity resolution*", 2006.
[32] Guilherme Dal Bianco, Renata Galante, Carlos A. Heuser, "*A fast approach for parallel deduplication on multicore processors*", 2011.
[33] R. Vernica, M. J. Carey, and C. Li., "*Efficient parallel set-similarity joins using mapreduce*", 2010.
[34] William E. Yancey**,** "*BigMatch: A Program for Extracting Probable Matches from a Large File for Record Linkage*", 2002.
[35] I. Fellegi and A. Sunter, "*A theory of record linkage*". In Journal of the American Statistical Society, 1969.
[36] Dewendra Bharambe, Susheel Jain, Anurag Jain. "*A Survey: Detection of Duplicate Record*", 2012.
[37] E. H. Porter and W. E. Winkler, "*Approximate String Comparison and its Effect on an Advadced Record Linkage System*", 1997.
[38] Clark DE. "*Practical introduction to record linkage for injury research"*. 2004.
[39] Mi T, Rajasekaran S, Aseltine R. "*Efficient algorithms for fast integration on large data sets from multiple sources*". BMC Med Inform Decis Mak 2012.
[40] Dean J, Ghemawat S. "*Mapreduce: simplified data processing on large clusters"*. In Proceedings of the 6th conference on Symposium on Operating Systems Design and Implementation, Berkeley, CA, USA, 2004
[41] Amón I, Jiménez C. Detección de Duplicados: Una Guía Metodológica. Revista Colombiana de Computación. Volumen 11. No. 2. Diciembre 2010 p7-23. ISSN 1657-2831.
[42] Else, Willis I. "*The Complete Soundex Guide: Discovering the rules Used by the Census Bureau and the Inmigration and Naturalization Service When These organization Indexed Federal Records*". Apollo, PA: Closon Press. 2002.
[43] L. Philips. "*The double-metaphone search algorithm. C/C++ User's Journa*l", 18(6), 2000.
[44] Amón I, Jiménez C, Moreno F, Echeverri J. "A*lgoritmo fonético para detección de cadenas de texto duplicadas en el idioma español*". 2012.
[45] Holmes D, McCabe, M. "*Improving Precision and Recall for Soundex Retrieval*". 2014.
[46] Christen, P., *"A Survey of Indexing Techniques for Scalable Record Linkage and Deduplication"*, p3. 2012.
[47] R.L. Taft, "Name Search Techniques", Technical Report Special Report No. 1, New York State Identification and Intelligence System, Albany, N.Y., Feb. 1970.
[48] White, T., "*Hadoop the definitive guide*", 2011.
[49] Dean, J., Ghemawat, S., *"MapReduce: Simplified Data Processing on Large Clusters"*, 2008.
[50] Borthakur, D., "*The Hadoop distributed file system: Architecture and Design*", 2007.
[51] "*Ordenamiento de raíz (radix sort)"*. http://ict.udlap.mx/people/ingrid/Clases/IS211/Radix.html.
[52] Walter Santos, Thiago Teixeira, Carla Machado, Wagner Meira Jr., Altigran S. Da Silva, Renato Ferreira, Dorgival Guedes, "*A Scalable Parallel Deduplication Algorithm*", 2007. Cap. 2.1, p. 2.
[53] Walter Santos, Thiago Teixeira, Carla Machado, Wagner Meira Jr., Altigran S. Da Silva, Renato Ferreira, Dorgival Guedes, "*A Scalable Parallel Deduplication Algorithm*", 2007. Cap. 3.2, p. 5.
[54] Dataflux: http://www.dataflux.com/.
[55] LINKAGEWIZ: http://www.linkagewiz.com.
[56] LINKKING: http://www.the-link-king.com.
[57] LINKPLUS: http://www.cdc.gov/cancer/npcr.