Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

Tipo de artículo: Artículo original
Temática: Tecnologías de bases de datos
Recibido: 25/03/2016 | Aceptado: 30/06/2016

Algoritmo para corregir anomalías a nivel de instancia en grandes volúmenes de datos utilizando *MapReduce*

Algorithm to correct instance level anomalies in large volumes of data using MapReduce

Yaisel Nuñez-Arcia [1*], Lisandra Díaz-de-la-Paz [1], Juan Luis García-Mendoza [1]

[1] Universidad Central "Marta Abreu" de Las Villas. Carretera a Camajuaní km 51/2. Santa Clara, Villa Clara, Cuba. Teléfono: 42-281515

[*] Autor para correspondencia: ynunes@uclv.cu

Resumen

Los problemas de calidad de datos a nivel de instancia tienen un impacto directo en la toma de decisiones de las organizaciones y afectan su desempeño. A medida que crece desmedidamente la información es mayor la probabilidad de que se encuentren dichos problemas en los datos. En este trabajo se presenta un algoritmo para corregir anomalías a nivel de instancia en fuentes de datos *big data* con formato estructurado o semi-estructurado. Como método de agrupamiento se utiliza el algoritmo *K-means*, para calcular la distancia de edición entre las cadenas se aplica la modificación de Levenshtein y para manejar el volumen de los datos se utiliza el modelo de programación distribuida *MapReduce*. Además, con el fin de mejorar la calidad de los datos se propusieron las siguientes cuatro fases: identificación del tipo de fuente de datos, el formato de los datos y el problema a corregir; pre-procesamiento de los datos; agrupamiento de los datos y limpieza de los datos.

Palabras clave: calidad de datos, limpieza de datos, *big data*, algoritmo *K-means*, *MapReduce*

Abstract

Data quality problems at instance level have a direct impact on decision making of organizations and affect their performance. As information grows unreasonably it is greater the probability that such problems occur in data. This paper presents an algorithm to correct instance level anomalies in big data sources with semi-structured or structured format. As a clustering method, K-means algorithm was used. To calculate the edit distance between strings the modification of Levenshtein was applied, and to handle the volume of the data, MapReduce model for distributed programming was used. Besides, in order to improve data quality, the following four phases were proposed:

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

identification of the data source type, data format and the problem to be solved; pre-processing of the input data; data clustering and data cleansing.

Keywords: *data quality, data cleansing, big data, K-means algorithm, MapReduce*

Introducción

Las organizaciones actualmente manejan grandes cantidades de datos que provienen de una única fuente o de la fusión de varias fuentes heterogéneas. Usualmente al integrar fuentes diversas surgen problemas de calidad que necesitan ser tratados, por tanto, se hace imprescindible contar con algoritmos y herramientas que permitan mantener un control adecuado de la calidad de los datos.

La existencia de estos problemas degrada significativamente la calidad de la información con un impacto directo en la eficiencia de una empresa. Esto se debe a múltiples razones que generalmente se expresan durante la inserción de los datos, estas son: la falta de reglas implementadas, el incorrecto manejo de los datos cambiantes y los errores de escritura tales como: errores lexicográficos y transposiciones de caracteres. Como consecuencia de ello, se encuentran con más frecuencia de lo deseado anomalías tales como: inconsistencias, valores perdidos, registros duplicados y representaciones no convencionales de datos (Koudas et al., 2006). La eliminación de estos problemas en los sistemas de información generalmente se conoce por el término "limpieza de datos" y tiene como principal objetivo mejorar la calidad de los datos (Barateiro and Galhardas, 2005).

Una de las técnicas existentes para corregir las anomalías de tipo cadena en diferentes fuentes de datos es a través del agrupamiento basado en el cálculo de las distancias entre cadenas. En el Centro de Estudios de Informática (CEI) de la Universidad Central "Marta Abreu" de Las Villas (UCLV) se han obtenido buenos resultados en bases de datos relacionales con el trabajo presentado por López (2011). En este último se utiliza como método de agrupamiento el algoritmo PAM (*Partition Around Medoides*) introducido por Kaufman and Rousseeuw (2009) y se propone una modificación a la distancia de edición Levenshtein (1966), de tal manera que se construye un grafo a partir de la distancia entre los diferentes caracteres del teclado más usado en Cuba, el tipo QWERTY (López, 2011).

Por otra parte, con el crecimiento exponencial que ha experimentado la información y la gran variedad de formatos de datos existentes, se hace necesario analizar las posibilidades de extender dicho algoritmo hacia otras fuentes de datos, por ejemplo, *big data* con formato estructurado o semi-estructurado. Actualmente existe un amplio conjunto de tecnologías y marcos de trabajo para procesar grandes volúmenes de datos (*big data*), los cuales continúan

evolucionando. No obstante, se decide utilizar el modelo de programación distribuida *MapReduce*, por ser uno de los más utilizados en la literatura y más simples de implementar (Gregory, 2016). Por tanto, el presente trabajo tiene como objetivo proponer un algoritmo para corregir anomalías a nivel de instancia en fuentes de datos *big data* con formato estructurado o semi-estructurado.

Materiales y métodos

Durante el procesamiento de grandes volúmenes de datos, la presencia de anomalías e impurezas en dichos datos conduce a la obtención de resultados erróneos, los cuales como consecuencia provocan la elevación de los costos y la disminución de los beneficios de su análisis e interpretación (López et al., 2010, López, 2011).

Por consiguiente, se hace necesario analizar la información presente en fuentes de datos *big data*, luego detectar posibles anomalías e inconsistencias en los datos y por último realizar el proceso de corrección de errores, mejorando así la calidad de los datos.

Fuentes de datos *big data*

En la literatura existen varias definiciones de *big data*, en el presente trabajo se aborda la siguiente definición. "El término *big data* hace referencia a una inmensa y compleja colección de datos. Es un término aplicado a conjuntos de datos que superan la capacidad habitual del software para capturarlos, gestionarlos y procesarlos en un tiempo razonable" (Hashem et al., 2014). Para poder trabajar con *big data*, necesario conocer las posibles representaciones de los datos. En (Batini and Scannapieco, 2006) se refiere a tres tipos de datos: estructurados, semi-estructurados y no estructurados.

- Los datos estructurados son aquellos que cada elemento tiene una estructura asociada. Son datos en los cuales se permite realizar fácilmente las operaciones de entrada, consulta y análisis en las fuentes de datos. Un ejemplo de fuentes datos estructurados es el SQL (*Structured Query Language*), creado para la gestión y consulta de datos en el RDBMS (*Relational Data Base Management System*) (Hashem et al., 2014), (Batini and Scannapieco, 2006) (Batini and Scannapieco, 2006, Hashem et al., 2014).
- Los datos semi-estructurados son aquellos que presentan información procesada y con un formato definido, pero no estructurado. De esta manera se puede tener la información definida, pero con una estructura variable. Dos ejemplos son las bases de datos basadas en columnas y los ficheros con información en un lenguaje de etiquetas HTML o XML (Hashem et al., 2014).

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

- Los datos no estructurados son aquellos que se expresan en lenguaje natural y no presentan ningún tipo de estructura. Ejemplo de ellos son los mensajes de texto, datos de los medios sociales, videos, entre otros (Batini and Scannapieco, 2006).

El presente trabajo se enfoca en el análisis de posibles errores en las fuentes de los datos con representación estructurada o semi-estructurada en grandes volúmenes de datos. En cada fuente de datos independientemente del formato en que estén representados dichos datos se pueden presentar diversos errores, los cuales necesitan ser clasificados y caracterizados para luego ser corregidos con el propósito de gestionar la información de manera precisa y confiable.

Taxonomía de errores en los datos

Existen varios niveles para tratar los errores o anomalías presentes en los datos. Entre las que se encuentra anomalías a nivel de esquema o a nivel de instancia.

Tanto los datos estructurados como semi-estructurados pueden presentar errores a nivel de instancia. Los problemas a nivel de instancia se pueden dividir en problemas de un único registro y problemas de múltiples registros (Gschwandtner et al., 2012).

Único registro: se refiere a los problemas que se presentan en uno o varios campos de un solo registro también denominado registro simple (Barateiro and Galhardas, 2005, López, 2011).

A continuación, se enumeran dichos problemas:

- Ausencia de datos en un campo no nulo: atributos que son rellenados con algún valor ficticio; por ejemplo, un número de carnet de identidad 99999999999 es un valor indefinido utilizado para superar la restricción no nula.
- Datos erróneos: datos que son válidos, pero no se ajustan a la entidad real; un ejemplo de datos erróneos es 31 que indica la edad de un empleado cuando este en realidad tiene 30 años.
- Errores ortográficos: palabras mal escritas en campos de las fuentes de datos; por ejemplo, “Calos Rodrigues” en lugar de “Carlos Rodríguez”.
- Valores implícitos: existencia de datos extraños en algún campo de datos; un ejemplo común de valores implícitos es la inserción de un título en un campo de nombre, por ejemplo, “Licenciado Carlos Rodríguez”.
- Valores de campos perdidos: datos que son almacenados en el campo equivocado. Por ejemplo, el valor “Villa Clara” ubicado en el atributo Municipio.
- Datos ambiguos: datos que pueden ser interpretados en más de una manera, con diferentes significados. La existencia de datos ambiguos puede ocurrir debido a la presencia de abreviatura: por ejemplo: el nombre

abreviado “J. González” puede ser ampliado de diferentes formas, como: “Juan González”, “Jorge González”, “Julio González”, etc.

Múltiples registros: los problemas con los datos en múltiples registros no pueden ser detectados considerando cada registro por separado, ya que como el nombre lo indica, se está haciendo referencia a más de un registro (Barateiro and Galhardas, 2005, López, 2011).

- Registros duplicados: son aquellos que no contienen información contradictoria; por ejemplo, los siguientes registros de empleados se consideran duplicados: Empleado1 (Nombre = “Carlos Rodríguez”, Dirección = “25, Calle Primera, Municipio de Santa Clara”, Nacimiento = 07/10/1990); Empleado2 (Nombre = “Carlo Rodríguez”, Dirección = “25, Calle 1ra, Sta Clara”, Nacimiento = 07/10/1990).
- Contradicción de registros: son aquellos que contienen información contradictoria; por ejemplo, si se consideran los registros Empleado1 y Empleado2, pero con la siguiente información: Empleado1 (Nombre = “Carlos Rodríguez”, Dirección = “25, Calle 1ra, Santa Clara”, Nacimiento = 07/10/1990); Empleado2 (Nombre = “Carlos Rodríguez”, Dirección = “25, Calle 1ra, Santa Clara”, Nacimiento = 07/10/1980).
- Datos no estandarizados: diferentes registros que no utilizan las mismas representaciones de datos, invalidando así su comparación:
    - Transposición de palabras: en un solo campo, las palabras pueden aparecer en diferente orden; por ejemplo, los nombres “Carlos Rodríguez” y “García, Julio” no utilizan la misma regla de ordenación.
    - Formato de codificación: uso de diferentes formatos de codificación, por ejemplo, ASCII, UTF-8, entre otros.
    - Formato de representación: diferentes representaciones de un formato para la misma información; por ejemplo, el formato de moneda puede ser $ 10.5, 10.5 $, etc.
    - Unidad de medida: diferentes unidades utilizadas en registros distintos, por ejemplo, las distancias dadas en cm y pulgadas.

En el presente trabajo se analizan específicamente anomalías provenientes de un único registro tales como: errores ortográficos, valores implícitos y datos ambiguos (abreviaturas); y de múltiples registros los datos no estandarizados (transposición de palabras). Luego de detallar los diferentes problemas que pueden encontrarse en las fuentes de datos, se deben aplicar técnicas de limpieza de datos que permitan corregir dichos errores en los datos con el fin de mejorar su calidad.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

Técnicas de limpieza de datos

El proceso de limpieza de datos es una tarea crucial para mejorar el resultado de la gestión de la información en una empresa. Es importante su uso para corregir diferentes problemas como la integridad, estandarizar valores, completar los datos que faltan, consolidar ocurrencias duplicadas, entre otros (Barateiro and Galhardas, 2005).

Un enfoque de limpieza de datos debe satisfacer los siguientes requisitos: detectar y eliminar los principales errores en las fuentes de datos. Además el enfoque debe estar apoyado por herramientas automatizadas, en aras de limitar la labor de inspección y la programación manual (Rahm and Do, 2000).

Para la detección de errores se pueden utilizar varios métodos tales como: estadísticos, basados en patrones y reglas de asociación y algoritmos de agrupamiento basados en distancias. Los métodos estadísticos, aunque simples y rápidos, tienen como principal desventaja que generan muchos falsos positivos. Los métodos basados en patrones y en reglas de asociación detectan posibles errores a partir del análisis de los registros que incumplen los patrones y las reglas descubiertas; sin embargo, su principal desventaja radica en no ser escalable para grandes volúmenes de datos. Por otra parte, los métodos de agrupamiento basados en distancias ofrecen buenos resultados en la comparación de cadenas de texto, reconocimiento óptico de caracteres y reconocimiento de patrones, a pesar de tener una gran complejidad computacional (Dean and Ghemawat, 2008; Jhaver et al., 2014).

En el presente trabajo se utiliza el método de agrupamiento basado en distancias. Para aplicar esta técnica de limpieza de datos se aplica algoritmo *K-means*, la modificación a la distancia de edición de Levenshtein propuesta por López (2011) y el modelo de programación *MapReduce*.

Algoritmo *K-means*

El algoritmo *K-means* fue introducido por MacQueen (1967) para el cálculo de clústeres. Es un algoritmo simple, directo y está basado en el análisis de las varianzas que se encarga de agrupar un conjunto de datos en un número predefinido de clústeres denominado K. Se comienza con un conjunto aleatorio de centroides de cada uno de los clústeres y se continúa reasignando los datos del conjunto de datos a los centroides, basándose en la similitud entre el dato y el centroide. El proceso de reasignación no se detiene hasta que se converge al criterio de parada (por ejemplo, se alcanzó un número fijo de iteraciones o los clústeres encontrados no cambian luego de cierto número de iteraciones) (Cambronero and Moreno, 2006; Villagra et al., 2009). Este proceso se diferencia del algoritmo PAM utilizado en (López, 2011) en la forma de seleccionar los centros, pues en el PAM se hace de manera estática, mientras que en el *K-means* se va re-calculando. Para realizar el re-cálculo de centroides en el *K-means* se tiene la siguiente ecuación:

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

$$cj = \frac{1}{|C_j|}\sum_{\forall z \in C_j} z \qquad (1)$$

Donde

Cj representa los clústeres.

z se refiere a un elemento del conjunto de datos que pertenece Cj.

cj es un centroide de Cj.

$|\ Cj\ |$ corresponde al número de elementos en Cj.

Para la conformación de los grupos se calcula la similitud entre cadenas. En este trabajo se utiliza la modificación de la distancia de edición de Levenshtein propuesta en (López, 2011).

Modificación de la distancia de edición Levenshtein

La distancia de edición entre dos cadenas fue introducida por Levenshtein (1966) como la cantidad mínima de operaciones de inserción, eliminación y sustitución que hay que hacer para transformar una cadena en la otra. La función así definida se demuestra que constituye una métrica y ha tenido algunas variaciones en el tiempo (Zobel and Dart, 1996). Posteriormente en (López, 2011) se adicionó una nueva operación: la transposición, intercambio de caracteres adyacentes, la cual es un error frecuente en las personas que teclean rápido.

Para la utilización de este método de agrupamiento, la distancia entre dos elementos se determina empleando una modificación de la distancia de edición de Levenshtein planteada en (López, 2011), de tal manera que se tenga en cuenta la distancia en el teclado de los diferentes caracteres. Con este agrupamiento se logra que cadenas menos distantes queden reunidas en el mismo grupo.

Para buscar este costo se construye un grafo con el teclado de la computadora $G(V, A)$, donde V es el conjunto de teclas (vértices del grafo) y A es el conjunto de aristas. El vértice Vi está conectado con el vértice Vj si representan teclas adyacentes en el teclado.

Una porción de este grafo que representa el teclado se muestra en la Figura. **1**.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

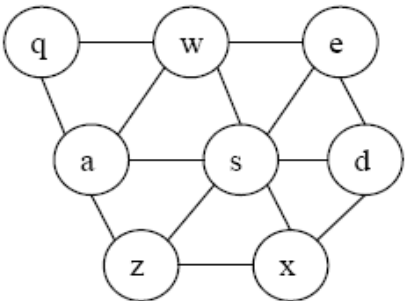

Figura. 1. Porción del grafo del teclado de la computadora. (Fuente (López, 2011))

De esta forma el costo w_s de sustituir el carácter *i* por el carácter *j*, es la longitud del camino más corto para ir de *i* a *j* en el grafo *G*.

La distancia de edición entre las cadenas *s* y *t* de longitud *n* y *m* respectivamente, la cual se denomina "EDUK" (*Edition Distance Using Keyboard*) (López, 2011), se define el cálculo de la distancia a través de una matriz de dimensión (n x m) de forma tal que:

$$d(i,0) = i, i = 0,1,\ldots,n \qquad (2)$$

$$d(0,j) = j, j = 0,1,\ldots,m \qquad (3)$$

Para el resto de los valores:

$$d(i,j) = \min \begin{cases} d(i-1,j) + w, \\ d(i,j-1) + w, \\ d(i-1,j-1) + R(s[i],t[j]), \\ d(i-2,j-2) + F(s[i-1],s[i],t[j-1],t[j]) \end{cases} \qquad (4)$$

Donde,

$R(s[i],t[j]) = w_s$, es la longitud del camino más corto en el grafo $G(V,A)$ del teclado para ir del carácter $s[i]$ al carácter $t[j]$, si son diferentes y $R(s[i],t[j]) = 0$, si los caracteres $s[i]$ y $t[j]$ son iguales,

$F(a,b,c,d) = w$, si $ab = dc$, en otro caso no se tiene en cuenta (*a*, *b*, son caracteres de la cadena s y *c*, *d* caracteres de la cadena t)

Entonces $EDUK(s,t) = d(n,m)$

En ocasiones las cadenas están compuestas por más de una palabra. Es frecuente también que cadenas similares se escriban poniendo en diferentes posiciones las palabras que la componen.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

Para tratar tales cadenas, en (López, 2011) se propone: dividir cada cadena en las palabras que la componen (se utiliza el término *token* para generalizar), buscar la distancia EDUK de cada *token* de una cadena con los *tokens* de la otra y tomar la distancia mínima, después estas distancias se suman y el resultado constituye la distancia entre las cadenas.

Formalizando esta idea:

Sean s y t dos cadenas y $S = \{s1, s2, \dots s_p\}$, $T = \{t1, t2, \dots t_q\}$ los conjuntos de los *tokens* que las forman, entonces:

$$d_{TOK}(s,t) = \begin{cases} 0 \; si \; s = t \\ d_{TOK}(t,s) \; si \; q < q \\ 1 + \sum_{i=1}^{p} m_i \;\; en \; otro \; caso \end{cases} \qquad (5)$$

Siendo

$$m_i = \min\left(d_{EDUK}(s_i, t_j)\right), j = 1 \dots q$$

La función toma valor cero solamente para el caso de cadenas iguales. Cuando se tengan dos cadenas con los mismos *tokens* en órdenes diferentes, la distancia entre ellos es uno.

Al aplicar el algoritmo *K-means* y la técnica de limpieza de datos basada en la distancia de edición entre cadenas, a medida que va creciendo el volumen de datos aumenta la latencia en conformar los resultados. Con el propósito de disminuir este tiempo, se decide utilizar el modelo de programación distribuida *MapReduce* por la popularidad alcanzada en la actualidad dentro de las tecnologías *big data* y por ofrecer soluciones para este tipo de problema (Yan et al., 2012, Jhaver et al., 2014).

Tecnología *MapReduce*

MapReduce es un modelo de programación para el procesamiento y generación de grandes volúmenes de datos en sistemas distribuidos y está especialmente pensado para tratar ficheros de gran tamaño -del orden de gigabytes y terabytes-. También mejora el tratamiento de los datos no estructurados, ya que trabaja a nivel de sistema de ficheros. Está inspirado por las funciones *map* y *reduce* presentes en *Lisp* y otros lenguajes funcionales. Los programas escritos en este estilo funcional son automáticamente paralelizados y ejecutados en un clúster de máquinas de bajo costo. El sistema se encarga de dividir los datos de entrada, planificando la ejecución del programa en un conjunto de máquinas, ocupándose de las fallas y manejando la comunicación requerida entre las mismas. Esto permite a los programadores sin experiencia con este tipo de sistema utilizar fácilmente los recursos de un sistema distribuido (Ramet et al., 2011).

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

La función *map* se aplica en paralelo a cada par llave/valor del conjunto de datos de entrada produciendo una lista de pares *(k2, v2)*.

map (k1, v1) –> list (k2, v2)

Por cada llamada a dicha función se agrupan todos los valores intermedios asociados con la misma clave *k* y son enviados a la función *reduce*. Esta última recibe la clave con su conjunto de valores asociados y los fusiona para formar un conjunto de valores posiblemente más pequeño:

reduce (k2, list (v2)) -> list (v3)

Típicamente cada llamada *reduce* produce un valor (*v3* o un valor vacío), aunque una misma llamada puede devolver más de un valor (Dean and Ghemawat, 2008). Los resultados de las llamadas se recopilan en la lista de resultados.

Este modelo de programación ofrece múltiples ventajas como el poder trabajar sobre cualquier formato de datos (estructurado, semi-estructurado o no estructurado), no tener requisitos de hardware elevados, ser escalable a nivel de tamaño de clúster y de rendimiento, y funciona bien con grandes cantidades de datos ya que trabaja en bloques independientes del tamaño (Yan et al., 2012).

Utilizando el modelo que se plantea con *MapReduce* se propone un algoritmo de agrupamiento para la limpieza de datos a través del cálculo de la distancia entre cadenas para el procesamiento de grandes conjuntos de datos.

Resultados y discusión

Para limpiar grandes volúmenes de datos se propone dividir dicho proceso en las siguientes cuatro fases siguiendo un orden lógico.

Fase 1 Identificación

Inicialmente es necesario conocer el tipo de fuente de datos, su formato de representación y el (los) problema(s) a corregir.

Fase 2 Pre-procesamiento

En la propuesta del algoritmo para agrupar datos es necesario tener un pre-procesamiento del conjunto de los datos de entrada para poder realizar con éxito el agrupamiento a través del cálculo de las distancias.

Luego de la etapa inicial como primer paso es necesario calcular la cantidad de clústeres (k), para así conformar los clústeres y sus respectivos centros asociados. Para poder realizar el cálculo de la distancia entre las cadenas, estas se van a ir leyendo de un fichero. Como segundo paso tomar k cadenas aleatoriamente del fichero, las cuales van a ser los centros del clúster y se van a ubicar en un vector de centroides.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

Fase 3 Agrupamiento

En se esta etapa se aplica el siguiente algoritmo:

Algoritmo: Agrupamiento de las cadenas

```
clase Mapper
map(llave, valor, contexto)
1:      mejor_indice=0
2:      k=cantidad de clústeres
3:      mejor_dist=Distancia(c_0,valor)
4:      para i=1 hasta k hacer
5:          dist=Distancia(c_i,valor)
6:          si dist<mejor_dist entonces
7:              mejor_dist=dist
8:              mejor_indice=i
9:          fin si
10:     fin para
11:     escribir(mejor_indice,valor)
fin
clase Reducer
reduce(llave, lista(valores), contexto)
1:      m=cantidad de elementos de la lista de valores
2:      para i=0 hasta m hacer
3:          nuevo_Centro=Re-calcular nuevo centro del clúster
4:      fin para
5:      para i=0 hasta m hacer
6:          escribir(nuevo_Centro,lista_i)
7:      fin para
8:      si nuevo_Centro != viejo_Centro entonces
9:          Incrementar en 1
10:     fin si
fin
```

En el algoritmo planteado para lograr el agrupamiento de las cadenas se utiliza la función *map*, la cual recibe una llave referente a los centros de los clústeres y un valor que va a ser una cadena del fichero de entrada. En la línea 3 se calcula la distancia con la función (5) del centroide C_0 del primer clúster del vector con el valor pasado por parámetro a la función. Luego en la línea 4 se recorre el vector de clústeres comenzando con la segunda posición, en cada *i-ésimo* elemento se calcula la distancia del centro de clúster con el valor. En la línea 6 se va a ir comparando las distancias

Revista Cubana de Ciencias Informáticas
Vol. 10, No. 3, julio-septiembre, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 97-110

hasta obtener la mejor en relación con el centro del clúster (líneas 7 y 8). En la línea 11 se escribe en el sistema de archivos el índice del centro del clúster referente con su resultado asociado.

Por otra parte, en la función *reduce* se recibe la lista de los valores asociados a los centros de los clústeres. En las líneas de la 2 a la 4 se recorre la lista de valores y se re-calcula el centro del clúster escogiendo el mejor valor de los elementos. Este proceso se realiza utilizando la función (1) descrita anteriormente.

Posteriormente entre las líneas de la 5 a la 7 se escribe en el sistema de archivos el nuevo clúster y sus valores asociados.

Por último, en las líneas de la 8 a la 9 se comprueba la convergencia entre el nuevo centro calculado y el viejo centro. Si al realizar la comprobación los centros no son iguales, se incrementa un contador de actualización (quiere decir que se vuelve a ejecutar nuevamente el algoritmo con los nuevos centros), este procedimiento se realiza hasta que no se actualice más.

Fase 4 Limpieza

Por último, se deben verificar los diferentes grupos obtenidos en la etapa anterior. El proceso ahora se reduce a la cantidad de grupos definidos, pues en cada uno de ellos se encuentran las cadenas menos distantes.

En caso de presentar ambigüedades como las que se muestran en la

Tabla , se recomienda determinar de forma interactiva las sustituciones a realizar, o sea preguntarle directamente al usuario (López, 2011).

Tabla 1. Ejemplo de dos clústeres a los cuales puede pertenecer la cadena J. González.

Clúster 1	Clúster 2
…	…
Jorje Gonzáles	Jwlio Gonzáles
Jorge González	Julio González
Jorge Gonsales	Julio Gomzales
…	…

Por ejemplo si se procesa la cadena “J. González”, esta puede pertenecer a cualquiera de los dos clústeres que se muestran en la

Tabla , por tanto se recomienda la interacción analista y sistema para asignarla al clúster correcto.

Conclusiones

En este trabajo se propuso un algoritmo para corregir las anomalías a nivel de instancia en fuentes de datos *big data* con formato estructurado o semi-estructurado. Para la implementación del algoritmo propuesto se utilizaron tres técnicas fundamentales, (i) la técnica de agrupamiento, (ii) el cálculo de la distancia de edición y (iii) la programación

en paralelo. Específicamente, para formar los grupos se usó el algoritmo *K-means*. Para el cálculo de las distancias entre cadenas se utilizó la modificación a la distancia de edición Levenshtein, y para manejar el volumen de los datos se empleó el popular modelo de programación en paralelo *MapReduce*, lo cual constituye un elemento novedoso. Además, se propusieron las siguientes cuatro fases: identificación del tipo de fuente de datos, el formato de los datos y el problema a corregir; pre-procesamiento de los datos; agrupamiento de los datos y limpieza de los datos, con el propósito de mejorar la calidad de los datos. En trabajos futuros se debe probar el algoritmo en conjuntos de datos *big data* con formato estructurado o semi-estructurado para casos de estudio reales y se deben adaptar otras técnicas de limpieza para corregir las anomalías no expuestas en este trabajo.

Referencias

BARATEIRO, J. & GALHARDAS, H. 2005. A Survey of Data Quality Tools. *Datenbank-Spektrum,* 14**,** 48.

BATINI, C. & SCANNAPIECO, M. 2006. *Data quality: concepts, methodologies and techniques*, Springer.

CAMBRONERO, C. G. & MORENO, I. G. 2006. Algoritmos de aprendizaje: knn & kmeans. *Inteligencia en Redes de Comunicación, Universidad Carlos III de Madrid*.

DEAN, J. & GHEMAWAT, S. 2008. MapReduce: simplified data processing on large clusters. *Communications of the ACM,* 51**,** 107-113.

GREGORY, S. 2016. *Top Big Data Processing Frameworks* [Online]. KDnuggets ™. Available: http://www.kdnuggets.com/2016/03/top-big-data-processing-frameworks.html 2016].

GSCHWANDTNER, T., GÄRTNER, J., AIGNER, W. & MIKSCH, S. 2012. A taxonomy of dirty time-oriented data. *Multidisciplinary Research and Practice for Information Systems.* Springer.

HASHEM, I. A. T., YAQOOB, I., ANUAR, N. B., MOKHTAR, S., GANI, A. & KHAN, S. U. 2014. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems,* 47**,** 98-115.

JHAVER, S., KHAN, L. & THURAISINGHAM, B. 2014. Calculating Edit Distance for Large Sets of String Pairs using MapReduce.

KAUFMAN, L. & ROUSSEEUW, P. J. 2009. *Finding groups in data: an introduction to cluster analysis*, John Wiley & Sons.

KOUDAS, N., SARAWAGI, S. & SRIVASTAVA, D. Record linkage: similarity measures and algorithms. Proceedings of the 2006 ACM SIGMOD international conference on Management of data, 2006. ACM, 802-803.

LEVENSHTEIN, V. I. Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady, 1966. 707-710.

LÓPEZ, B. 2011. Limpieza de Datos: Reemplazo de valores ausentes y Estandarización. *Resumen de la tesis*.

LÓPEZ, B., PÉREZ, R. & BATULE, M. 2010. Las reglas de asociación ordinales en la detección de errores en los datos. *Revista Cubana de Ciencias Informáticas*, 4.

MACQUEEN, J. Some methods for classification and analysis of multivariate observations. Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, 1967. Oakland, CA, USA., 281-297.

RAHM, E. & DO, H. H. 2000. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23**,** 3-13.

RAMET, D., LAGO, J., KARLSSON, J., FALGUERAS, J. & TRELLES, O. 2011. Mr-Cirrus: Implementación de Map-Reduce bajo MPI para la ejecución paralela de programas secuenciales. *Proceedings of XXII Jornadas de Paralelismo, Las Palmas de Gran Canaria, España*.

VILLAGRA, A., GUZMÁN, A., PANDOLFI, D. & LEGUIZAMÓN, G. 2009. Análisis de medidas no-supervisadas de calidad en clusters obtenidos por K-means y Particle Swarm Optimization.

YAN, C., YANG, X., YU, Z., LI, M. & LI, X. Incmr: Incremental data processing based on mapreduce. Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 2012. IEEE, 534-541.

ZOBEL, J. & DART, P. Phonetic string matching: Lessons from information retrieval. Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval, 1996. ACM, 166-172.