HAL
open science

Procesamiento de big data en Hadoop usando el repartition join

Néstor Iván Escalante Fol, Alberto Flores Portilla, Genoveva Vargas-Solar, Carolina Rocío Sánchez Pérez, Marva Angélica Mora Lumbreras

To cite this version:

Néstor Iván Escalante Fol, Alberto Flores Portilla, Genoveva Vargas-Solar, Carolina Rocío Sánchez Pérez, Marva Angélica Mora Lumbreras. Procesamiento de big data en Hadoop usando el repartition join. Programación Matemática y Software, 2015, 7, pp.52 - 58. hal-01584806

HAL Id: hal-01584806

https://hal.science/hal-01584806v1

Submitted on 12 Sep 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Procesamiento de *big data* en Hadoop usando el *repartition join*

Implementig the repartition join for processing big data using Hadoop

Néstor Ivan Escalante Fol,*[1] Alberto Portilla Flores,[1] Genoveva Vargas del Solar,[2]
Carolina Rocío Sánchez Pérez,[1] Marva Angélica Mora Lumbreras[1]

[1] Facultad de Ciencias Básicas, Ingeniería y Tecnología, Universidad Autónoma de Tlaxcala.
Calzada Apizaquito s/n, Apizaco. CP 90300. Tlaxcala, México
[2] French Council of Scientific Research, LIG-LAFMIA.
681 rue de la Passerelle BP 72, 38402. Saint Martin d'Heres, Francia
* Correo-e: nestorescalantefol@gmail.com

PALABRAS CLAVE:

big data, Map Reduce, *Hadoop*, *join*

RESUMEN

El objetivo principal de este trabajo es el procesamiento de grandes volúmenes de información, conocidos como *big data*. Presentamos la implementación del algoritmo *repartition join* para realizar la operación *join* en un conjunto grande de datos. El algoritmo fue programado bajo el modelo de programación Map Reduce. Implementar un *join* en el contexto de *big data* resulta ser complejo y costoso; por ello, apoyados en la plataforma Hadoop, herramienta que ofrece las utilidades necesarias para el manejo de grandes volúmenes de información, analizamos el comportamiento del algoritmo para evaluar su rendimiento. El algoritmo planteado se evaluó en un clúster conformado por tres nodos. Los resultados de ejecución se analizaron para su posterior uso en aplicaciones con datos reales.

KEYWORDS:

big data, Map Reduce, Hadoop, join

ABSTRACT

The main objective of this work concerns the processing of big data. Therefore repartition implementation algorithm is proposed to perform the join operation in a large data set, applying under the Map Reduce programming model. Implementing a join in the context of big data is complex and costly, therefore we use Hadoop platform, which provides the necessary tool for managing large volumes of information utilities to analyze the behavior of the algorithm and to evaluate its performance. The algorithm was tested in a cluster consisting of 3 nodes, analyzing the execution results for later use with real data.

Recibido: 31 de julio de 2014 • **Aceptado:** 6 de noviembre de 2014 • **Publicado en línea:** 30 de junio de 2015

1 *BIG DATA*

En los últimos años se han generado datos a una escala sin precedentes, por lo que hoy en día estamos sumergidos en un mar de información; esto significa que el volumen de datos que necesitan ser extraídos, transformados, almacenados y analizados se incremente exponencialmente. De estas grandes cantidades surge el término *big data*, que se refiere al procesamiento de enormes cantidades de datos no estructurados o semiestructurados que los sistemas convencionales no son capaces de gestionar. El *big data* también describe el tratamiento de información en el que el volumen, la velocidad y la variedad de los datos exceden las posibilidades de organización para calcular una adecuada y oportuna toma de decisiones.

La investigación sobre *big data* pretende solucionar desafíos que se presentan al momento de capturar, almacenar y dar mantenimiento a los datos, así como los presentes en la búsqueda de información para su posterior análisis y visualización de resultados. A medida que avanza la tecnología, encontramos que el universo digital comprende todo tipo de datos; sin embargo, la mayoría de nuevos datos que se generan son de tipo no estructurado. Esto significa que muchas veces sabemos poco acerca de ellos, ya sea porque no poseen definiciones de tipos, no están organizados de acuerdo con ningún patrón o simplemente no contemplan el concepto de variables o atributos. Los metadatos permiten enfrentar la información no estructurada, ya que sus características permiten obtener información, contenido, calidad, condición y otros elementos sobre los datos. La única manera en la que una empresa puede ser competitiva es teniendo capacidad para procesar su principal activo: la información que genera.

Este artículo está organizado como sigue: la sección 2 introduce el paradigma Map Reduce; la sección 3 presenta la plataforma Hadoop; la sección 4 presenta el algoritmo *join*; la sección 5 es sobre la implementación; la sección 6 trata brevemente los resultados y pruebas, y finalmente, la sección 7 presenta las conclusiones.

2 MAP REDUCE

Map Reduce es un modelo de programación utilizado para manejar grandes cantidades de datos en sistemas distribuidos. Proporciona un marco de ejecución para el procesamiento de datos a gran escala [3]. Su principio básico es dividir un problema en pequeñas tareas independientes para que sean atendidas en paralelo por distintos procesos; por ejemplo, en diferentes máquinas de un clúster. Luego, los resultados de cada proceso son combinados y mostrados como salida final. Los conjuntos de datos de entrada pueden provenir de una base de datos o un fichero y sus valores pueden ser enteros, flotantes, cadenas, bytes o estructuras complejas como listas, tuplas o arreglos. La estructura básica de trabajo en Map Reduce son pares clave-valor. Un trabajo de Map Reduce se divide en cuatro etapas:

- Inicialización. En esta etapa se preparan los datos de entrada (BD, archivos html, etcétera) y son divididos en procesos o pequeñas tareas.
- Map. Esta función recibe como parámetros un par (clave-valor) y devuelve una lista de pares. Esta función se encarga del mapeo y se aplica a cada elemento de la entrada de datos, por lo que se obtendrá una lista de pares por cada llamada a la función Map.
- Agrupación y ordenación. Como su nombre lo dice, en esta etapa se crean diferentes grupos dentro de cada proceso y son ordenados para un manejo más sencillo y adecuado de los datos.
- Reduce. Esta función se aplica en paralelo para cada grupo creado por la función Map. Reduce se llama una vez para cada clave única de la salida de la función Map. Junto con esta clave, se pasa una lista de todos los valores asociados con la clave para que pueda realizar alguna fusión y producir un conjunto más pequeño de los valores.

2.1 Multiplicador de matrices

A continuación se describe el ejemplo de cómo funciona un algoritmo en Map Reduce para llevar a cabo la multiplicación de dos matrices AxB (ver figura 1). El algoritmo matemático para realizar el producto de matrices se presenta en la figura 2.

El algoritmo de la multiplicación para las matrices bajo el modelo de Map Reduce es el siguiente:

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} 19 & 24 \\ 50 & 10 \\ 8 & 21 \end{pmatrix} \cdot \begin{pmatrix} 7 & 9 & 60 \\ 59 & 31 & 20 \end{pmatrix} = \begin{pmatrix} 1549 & 915 & 1620 \\ 940 & 760 & 3200 \\ 1295 & 723 & 900 \end{pmatrix}$$

Figura 1. Matrices de tamaño `ixj` y `jxk`

$$\mathbf{AB} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}b_{j1} & \sum_{j=1}^{n} a_{1j}b_{j2} & \cdots & \sum_{j=1}^{n} a_{1j}b_{jp} \\ \sum_{j=1}^{n} a_{2j}b_{j1} & \sum_{j=1}^{n} a_{2j}b_{j2} & \cdots & \sum_{j=1}^{n} a_{2j}b_{jp} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{n} a_{mj}b_{j1} & \sum_{j=1}^{n} a_{mj}b_{j2} & \cdots & \sum_{j=1}^{n} a_{mj}b_{jp} \end{bmatrix}$$

Figura 2. Algoritmo para multiplicar matrices

En la fase Map:

- para cada elemento `(i,j)` de `A`, crear un par clave-valor donde la clave es `(i,k)` y el valor `A[i,j]` para `k` desde `1` hasta `N`.
- para cada elemento `(j,k)` de `B`, crear un par clave-valor donde la clave es `(i,k)` y el valor `A[j,k]` para `i` desde 1 hasta `L`.

En esta fase obtendremos como resultado claves que son compuestas por la posición dada por la fila y la columna de cada elemento de la matriz `A` y `B`; el resultado está compuesto por el valor de las matrices en esa posición. En el caso del ejemplo mostrado en la figura 1, los valores obtenidos son:

```
Map output
A[1,1]->((1,1),19)-((1,2),19)-((1,3),19)
A[1,2]->((1,1),24)-((1,2),24)-((1,3),24)
A[2,1]->((2,1),50)-((2,2),50)-((2,3),50)
A[2,2]->((2,1),10)-((2,2),10)-((2,3),10)
A[3,1]->((3,1),08)-((3,2),08)-((3,3),08)
A[3,2]->((3,1),21)-((3,2),21)-((3,3),21)

B[1,1]->((1,1),07)-((2,1),07)-((3,1),07)
B[1,2]->((1,2),09)-((2,2),09)-((3,2),09)
B[1,3]->((1,3),60)-((2,3),60)-((3,3),60)
B[2,1]->((1,1),59)-((2,1),59)-((3,1),59)
B[2,2]->((1,2),31)-((2,2),31)-((3,2),31)
B[2,3]->((1,3),20)-((2,3),20)-((3,3),20)
```

En la fase Reduce:

Se revisan y comparan todos los pares generados en la fase anterior, lo que da como resultado nuevos pares clave-valor, donde la clave es `(i,k)` y el valor son los datos para realizar la operación de `A[i,j]*B[j,k]`. Así se obtienen los elementos de la matriz producto. En el caso del ejemplo de la figura 1 los valores obtenidos son:

```
((1,1), (19*07, 24*59))
((2,1), (50*07, 10*59))
((3,1), (08*07, 21*59))

((1,2), (19*09, 24*31))
((2,2), (50*09, 10*31))
((3,2), (08*09, 21*31))

((1,3), (19*60, 24*20))
((2,3), (50*60, 10*20))
((3,3), (08*60, 21*20))

((1,1), (133 + 1416))
((2,1), (350 + 590))
((3,1), (56 + 1239))


((1,2), (171 + 744))
((2,2), (450 + 310))
((3,2), (72 + 651))

((1,3), (1140 + 480))
((2,3), (3000 + 200))
((3,3), (480 + 420))

Reduce output
((1,1), (1549))
((2,1), (940))
((3,1), (1295))

((1,2), (915))
((2,2), (760))
((3,2), (723))

((1,3), (1620))
((2,3), (3200))
((3,3), (900))
```

3 HADOOP

Hadoop es una plataforma que permite desarrollar *software* escalable y confiable para computación distribuida bajo el modelo de programación MapReduce. Puede ejecutarse en uno o más nodos y, en ambos casos, su funcionamiento se basa en la ejecución de cuatro procesos que se comunican bajo el modelo cliente-servidor. La arquitectura bajo la que trabaja Hadoop se muestra en la figura 3. Los elementos que conforman la arquitectura en Hadoop son los siguientes [4]:

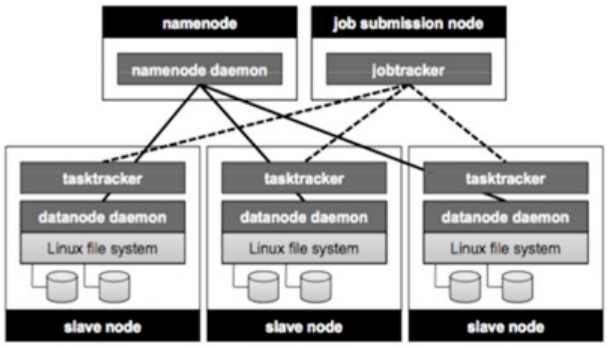

Figura 3. Arquitectura de Hadoop

JobTracker. Es un proceso que recibe los programas MapReduce del usuario, crea y asigna las tareas Map y las reduce a los procesos TaskTracker. Posteriormente, mantiene comunicación con dichos procesos para dar seguimiento al avance de ejecución de cada una de las tareas Map y Reduce. Si el proceso JobTracker falla, el ambiente MapReduce se rompe y no se pueden ejecutar sus programas.

TaskTracker. Son los procesos que se encargan de ejecutar las tareas Map y Reduce que han sido asignadas por JobTracker, y reportan su avance de ejecución al mismo. Aunque sólo se ejecuta TaskTracker por nodo cliente, cada TaskTracker genera múltiples máquinas virtuales de Java para ejecutar una tarea Map o una Reduce de manera paralela.

NameNode: Este proceso mantiene el árbol de directorios y archivos del sistema de archivos HDFS. Ubica en qué nodos se encuentran todos los *splits* de cada archivo y demás metadatos relacionados. Esta información no es persistente, se construye con ayuda de los DataNodes cuando inicia el sistema. También particiona los archivos en *splits* (por defecto con un tamaño de 64MB cada uno) y los distribuye en los DataNodes, a los que ordena la replicación correspondiente. Si un nodo que ejecuta un proceso DataNode falla, ordena a los otros DataNodes que realicen la réplica de los *splits* que se ubican en dicho nodo para mantener el factor de replicación. Si el NameNode falla, se corrompe el sistema de archivos HDFS.

DataNode. Son procesos encargados de realizar las operaciones de entrada/salida en el sistema HDFS. Mantienen comunicación con el NameNode para reportar donde se localizan los *splits* y recibir información para crear, mover, leer o eliminarlos. Por otra parte, se comunican entre ellos para realizar la réplica de los datos. JobTracker y NameNode son los procesos servidores, mientras que TaskTracker y DataNode son los procesos clientes. Hadoop permite a los usuarios especificar los nodos servidores y clientes.

4 OPERADOR *JOIN*

Las consultas en múltiples tablas o *join*, también denominadas combinaciones o composiciones, permiten recuperar datos de dos tablas o más, según las relaciones lógicas entre ellas. Las combinaciones indican cómo debería utilizar el SGBD los datos de una tabla para seleccionar los de otra.

Una condición de combinación define la forma en la que dos tablas se relacionan en una consulta:

- Especificar la columna de cada tabla que debe usarse para la combinación. Una condición de combinación específica a una clave externa de una tabla y su clave asociada en otra tabla.
- Especificar un operador lógico (=, <>, etc.) para usarlo en los valores de comparación de las columnas.

```
foreach tuple r ∈ R do
    foreach tuple s ∈ S do
        if ri==sj then add<r,s> to result
```

Figura 4. Algoritmo de *join* simple

```
Map (K: null, V: a record from a split of either R or L)
    join_key ← extract the join column from V
    tagged_record ← add a tag of either R or L to V
    emit (join_key, tagged_record)


Reduce ( K': a join key,
    LIST_ V': records from R and L with join key K' )
  create buffers BR and BL for R and L, respectively
  for each record t in LIST_ V' do
    append t to one of the buffers according to its tag
  for each pair of records (r,l) in BR X BL do
    emit ( null, new_record (r,l))
```

Figura 5. Fases Map y Reduce del algoritmo *repartition join* [6]

La forma más simple de construir un *join* es, por cada tupla en la primera relación R (ciclo externo), recorrer enteramente la segunda relación S (ciclo interno, ver figura 4). Como podrá verse, este algoritmo es de orden *nxm*; por lo tanto, es caro de implementar con relaciones muy grandes.

```
<row Id="1" Reputation="101" CreationDate="2012-02-08T19:45:13.447" DisplayName="Geoff Dalgas"
LastAccessDate="2013-06-04T21:38:25.000" WebsiteUrl="http://stackoverflow.com" Location="Corvallis, OR"
AboutMe="&lt;p&gt;Developer on the StackOverflow team.  Find me on&lt;/p&gt;&#xA;&#xA;&lt;p&gt;&lt;a
href=&quot;http://www.twitter.com/SuperDalgas&quot; rel=&quot;nofollow&quot;&gt;Twitter&lt;/a&gt;&#xA;&lt;
br&gt;&lt;br&gt;&#xA;&lt;a href=&quot;http://blog.stackoverflow.com/2009/05/welcome-stack-overflow
&quot;&gt;Stack Overflow Valued Associate #00003&lt;/a&gt;&lt;/p&gt;&#xA;" Views="15" UpVotes="0"
DownVotes="0" Age="37" AccountId="2" />
```

Figura 6. Registro de la tabla de usuarios

```
<row Id="19" PostId="11" Score="0" Text="you are correct - I meant in the context of
the traditional goalie being in the goalbox and the defensive player being further
up the field. I guess if the goalie ever came out far enough to not be the last defender
 than you would need to take the second-last defender rule into account.&#xA;&#xA;
And yes, that *is* the rule - which is why I usually chose to call it that way rather
than what I was told I should do. This was just youth soccer games though - I think
you guys are talking more about the official FIFA rules."
CreationDate="2012-02-08T20:29:05.383" UserId="18" />
```

Figura 7. Registro de la tabla de comentarios

4.1 *Repartition join*

El algoritmo r*epartition join* utiliza dos conjuntos de datos 'L' y 'R' con un campo clave en común y realiza dos fases. El pseudocódigo se observa en la figura 5 y las características principales son las siguientes [5]:

Fase Map:

- Cada tarea Map funciona en cada *split* tanto de `R` como de `L`
- Cada tarea Map etiqueta los registros de acuerdo a su tabla original
- Las salidas que resultan de la unión son etiquetadas como (clave-valor)
- Los resultados entonces son particionados, ordenados y agrupados

Fase *reducer*:

- Todos los registros de cada *join* son agrupados y eventualmente reducidos
- Para cada *join*, la función *reducer* separa los registros de entrada en dos conjuntos de acuerdo con la etiqueta de su tabla de origen
- Realiza un producto cruzado entre los registros de los conjuntos anteriores

El problema con esta versión del algoritmo es que todos los registros generados a partir del campo clave, tanto de `L` como de `R`, tienen que ser almacenados. Por lo tanto, se puede causar desbordamiento de memoria.

5 IMPLEMENTACIÓN

Hasta la etapa actual, se configuraron tres equipos IMac, para trabajar de manera distribuida en un clúster, con las siguientes características en el equipo de cómputo:

- Sistema Operativo Mac OS X Lion versión 10.7.5
- Procesador Intel Core 2 Duo, 2.16 GHz
- Memoria RAM 4GB DDR2 a 667 MHz
- Hadoop versión 1.0.4
- Oracle Java SRE 1.6

El conjunto de datos procesados se obtuvo del sitio *stackoverflow*. Los datos son tablas de usuarios y comentarios que realizan los usuarios registrados en un sitio *web*. Los archivos están estructurados en formato XML, como se observa en las figuras 6 y 7.

5.1 Código de las fases Map y Reduce

en la figura 8 se muestra el código para realizar la fase Map sobre los archivos que contienen los datos de los usuarios y los comentarios. En la figura 9 se muestra la fase Reduce, sobre la cual se aplica el algoritmo descrito en la sección anterior.

```
public static class UserMapper extends Mapper<Object, Text, Text, Text> {

        private Text clave = new Text();
        private Text valor = new Text();
        public void map (Object claves, Text valores, Context contexto) throws
        IOException, InterruptedException {
        Map<String, String> tabla = DatosXML.Xml2Map(valores.toString());
        String IdUsuario = tabla.get("Id");
        if (IdUsuario == null) {
                return;
        }
        clave.set(IdUsuario);
        valor.set("A" + valores.toString());
        contexto.write(clave, valor);
        }
}
```

Figura 8. Fase Map sobre el archivo Users.xml

```
public static class UserMapper extends Mapper<Object, Text, Text, Text> {

        private Text clave = new Text();
        private Text valor = new Text();
        public void map (Object claves, Text valores, Context contexto) throws
        IOException, InterruptedException {
        Map<String, String> tabla = DatosXML.Xml2Map(valores.toString());
        String IdUsuario = tabla.get("Id");
        if (IdUsuario == null) {
                return;
        }
        clave.set(IdUsuario);
        valor.set("A" + valores.toString());
        contexto.write(clave, valor);
        }
}
```

Figura 9. Fase Map sobre el archivo Users.xml

6 RESULTADOS Y PRUEBAS

En el cuadro 1 se muestran los resultados obtenidos hasta el momento en la fase de implementación. Las variables tomadas en cuenta en la evaluación de los datos son el tamaño de los archivos y el tiempo en que fueron procesados. La principal ventaja que aporta este algoritmo de acuerdo con las pruebas realizadas es la reducción del tiempo de ejecución del *join*, al obtener tiempos razonables con respecto a la cantidad de información procesada.

Cuadro 1. Resultados de la fase de implementación

Archivo "Users.xml"	Archivo "Comments.xml"	Tiempo de ejecución
1.1 Mb	1.2 Mb	34 s
1.1 Mb	2.9 Mb	1 m 5 s
98.8 Mb	104.9 Mb	3 m 6 s
809,7 Mb	956.6 Mb	13 m 24 s
809,7 Mb	4,3 Gb	26 m 37 s

7 CONCLUSIONES Y TRABAJO A FUTURO

Map Reduce es un modelo de programación que ha sido aceptado ampliamente para trabajar en los sistemas distribuidos para procesar grandes cantidades de datos. La razón de esta investigación va de la mano con el hecho de que el modelo y el algoritmo presentado puede ser de gran utilidad dentro de estos sistemas. Mediante este trabajo se ha podido demostrar el procesamiento práctico de *big data* en Hadoop de manera eficiente. Aún queda como tarea futura evaluar otros tipos de archivos y añadir más nodos en el clúster, y evaluar las mismas variables, tamaño de los archivos procesados y tiempo de ejecución del algoritmo como comparación. Se debe considerar como factor importante la cantidad de nodos disponibles, ya que entre más nodos se integren, existe la posibilidad de que el procesamiento de ejecución sea más rápido. Además nos planteamos estudiar a detalle la implementación de los algoritmos para proponer mejoras que se reflejen en su desempeño.

REFERENCIAS

1. Gantz, J. y Reinsel, D. The Digital Universe in 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East, EMC, Diciembre 2012, Disponible en: https://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf
2. Goicochea, A. Big Data es la necesidad, In memory computing es la solución (volumen y variedad con velocidad). Tecnologías de la Información y Estrategia. Blog personal, Noviembre de 2013. Disponible en: http://anibalgoicochea.com/2012/11/07/big-data-es-la-necesidad-in-memorycomputing-es-la-solucion/
3. Lin, J. y Dyer, C. Data-Intensive Text Processing. Manuscrito preparado, University of Maryland, College Park, abril de 2010.
4. Lam, C. Hadoop in Action. Stanford: Manning Publications, 2010.
5. Blanas, S., Patel, J.M., Ercegovac, V., Rao, J., Shekita, E.J., y Tian, Y. A comparison of join algorithms for log processing in MapReduce. Proceedings of the 2010 international conference on Management of data.
6. Manning, Ch. D., Raghavan, P. y Schütze, H. An Introduction to Information Retrieval, Cambridge: Cambridge University Press, 2009.

Acerca de los autores

Néstor Iván Escalante Fol. Egresado de la Licenciatura en Ingeniería en Computación de la Universidad Autónoma de Tlaxcala.

Genoveva Vargas-Solar. She is senior researcher of the French Council of Scientific Research (CNRS) and deputy-director the Franco-Mexican Laboratory of Informatics and Automatic Control (LAFMIA, UMI 3175). She is also member of the HADAS group of the Informatics Laboratory of Grenoble (France) and invited research fellow of the Data and Knowledge Management Group of the Research Centre of Information and Automation Technologies at Universidad de las Américas Puebla. She was elected president of the Mexican Society of Computer Science (2007-2009). She is senior member of the scientific council of the Mexican Network on Information and Communication Technologies which is a national program of the Mexican Council of Science and Technology.

Marva Angélica Mora Lumbreras. Profesora de tiempo completo del Posgrado en Computación y Electrónica de la Universidad Autónoma de Tlaxcala. Es Doctora en Ciencias de la Computación Magna Cum Laude por la Fundación Universidad de las Américas-Puebla, México con Perfil PROMEP de la SEP y miembro de la Red de TIC's del CONACyT y Editora de la Revista Iztatl Computación de la FCBIyT-UATx. Ha publicado artículos arbitrados a nivel nacional e internacional en el área de graficación y realidad virtual.

Alberto Portilla Flores. Profesor de tiempo completo y Coordinador del Posgrado en Computación y Electrónica de la Universidad Autónoma de Tlaxcala. Es Doctor en Informática por Universidad de Grenoble, Francia, Doctor en Ciencias de la Computación Cum Laude por la Fundación Universidad de las Américas-Puebla, México y PosDoc obtenido en el French Mexican Laboratory of Informatics and Automatic Control (LAFMIA UMI-3175). Es evaluador acreditado de los Comités de Acreditación de Evaluadores del Sistema Nacional de Evaluación Científica y Tecnológica y miembro de la Red de TIC's del CONACyT. Ha publicado artículos arbitrados a nivel nacional e internacional en el área de cómputo en la nube, sistemas transaccionales y computo orientado a servicios.

Carolina Rocío Sánchez Pérez. Es profesora de tiempo completo de la Licenciatura en Ingeniería en Computación de la Universidad Autónoma de Tlaxcala. Es Maestra en Ciencias de la Computación por Instituto Nacional de Astrofísica, Optica y Electrónica, México con Perfil PROMEP de la SEP. Ha publicado artículos arbitrados a nivel nacional e internacional.