Máster en Data Science. URJC
BBDD No convencionales

MongoDB y Neo4j

César Gonźalez Fernández, Carlos Correa García

18 de abril de 2018

Índice

Lista de Figuras

Glosario

BBDD Base de datos.

CSV Comma Separated Value.

JSON JavaScript Object Notation.

SQL Structured Query Language.

XML eXtendible Markup Language.

1. MongoDB

La primera parte de la práctica estará enfocada en la Base de datos (BBDD) no ralacional MongoDB. Esta se trate de la BBDD más importante actualmente. En el momento de escribir este texto, se encuentra en el 5to de BBDD más utilizadas según la web `https://db-engines.com/en/ranking` y solo por detrás de las BBDD relacionales.

MongoDB tiene las siguientes características:

- Es schemas less, es decir, no es preciso definir un esquema de datos. Aunque no por ello, tenemos que dejar de saber como son almacenados los datos.
- Usa el formato BSON para almacenar los datos, haciendo que sea facil su uso desde lenguajes de programación como Python y JavaScript.
- Permite realizar agregaciones. Incluye el framework aggregation el cuál le permite hacer operaciones realmente potentes.
- Podemos crear índices (incluyendo índices parciales) de cualquiera de sus campos, lo que acelera notablemente las busquedas.
- Su característica de *sharding* le permite ser muy escalable.
- Cuenta con una gran comunidad de desarrolladores detrás lo cual es una garantía de futuro.
- y muchas más.

Para la realización de esta parte de la práctica vamos a partir de tres documentos JavaScript Object Notation (JSON), los cuales podemos ver la forma de obtenerlos en el anexo B.

Cada uno de estos documentos contiene la información sobre los tres tipos de publicaciones que vamos a usar:

- Articles.
- Inproceedings.
- Incollections.

En nuestro caso, hemos hecho uso de una base de datos MongoDB encapsulada en un contenedor docker. Es posible realizarlo mediante una instalación nativa sobre linux pero hemos preferido la otra opción. Si se desea utilizar el mismo contenedor que hemos utilizado en nuestro caso, basta con ejecutar el siguiente comando:

```
sudo docker run --name practica-bbdd-mongo -d -p 27017:27017 mongo
```

Una vez lanzado el comando indicado, para volver a arrancar el contenedor basta con ejecutar el comando start con el nombre del contenedor creado:

```
sudo docker start practica-bbdd-mongo
```

1.1. Diseño

Aunque se dice que MongoDB se trata de una BBDD *Schemasless*, eso no quita de definir una correcta estructura para almacenar los datos la cual nos permita que las consultas que se hagan sean tanto más sencillas y más rápidas. También es importante el saber definir correctamente los índices que vamos a aplicar. En este sentido hay que tener en cuenta que el uso de índices aumenta el rendimiento de nuestras consultas de busqueda en la BBDD, pero resulta perjudiciales a la hora de hacer inserciones, además de ocupar espacio en memoria.

También es importante a la hora de definir la forma en la que se guardarán los datos el hecho de como gestionar las relaciones entre documentos. A grandes ragos, y sin entrar en muchos detalles, existen dos estrategias:

- Utilizar referencias entre documentos. Es decir, que uno de los campos de un documento almacene un identificador que sirva para identificar unívocamente a otro documento. Esta relacion puede darse en ambos sentidos. La principal ventaja es el ahorro de espacio al no tener elementos duplicados, así como la facilidad de gestionar una posible actualización de los datos. Por contra, las queries de busqueda serán más complejas al tener que también hacer una busqueda por el identificador del segundo documento.
- El segundo método consite en insertar el segundo documento dentro del primero como si de un campo más se tratase. Esto Facilita las queries de busqueda ya que recuperamos ambos documentos a la vez, pero tiene la pega de que muy probablemente dupliquemos datos lo cuál se traduce en una mayor cantidad de datos a almacenar y hace más complejo la actualización.

El hecho de optar por una u otra estrategia dependerá de las operaciones que vayamos a realizar sobre nuestra BBDD.

Para el caso que nos ocupa, hemos decidido crear una colección por cada tipo de publicación que vamos a trabajar. También vamos a crear una cuarta colección dentro de la cual cada documento representará un autor. Además, cada uno de estos documentos almacenará una información mínima de las publicaciones de dicho autor con el fín de facilitar las queries futuras. Además, se considera que la información de un libro, y la relación de este con sus autores, no debería sufrir cambios (salvo casos excepcionales), por lo que el tener estos datos embebidos dentro de otros no supondrá un gran problema a la hora de gestionar las actualizaciones.

La siguiente tabla muestra los datos de las colleciones con las que vamos a trabajar:

Colección	Información	Indices
Articles	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id
Incollections	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id
Inproceedings	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id
Authors	▪ _id ▪ authors: Array con los nombres de los autores.	▪ _id

1.2. Carga de los datos.

Los datos han sido cargados usando la herramienta **mongoimport**. Esta herramienta se ejecuta desde la línea de comandos y se instala junto con el propio MongoDB. Hay que tener en cuenta que, en caso de no existir la base de datos o la colección donde se indica que se deben cargar los datos, es el propio MongoDB el encargado de crearlas.

Para cargar nuestros datos se han ejecutado los siguientes comandos:

```
mongoimport --db=dblp --collection=articles json/articles.json
```

```
mongoimport --db=dblp --collection=incollections json/incollections.json
```

```
mongoimport --db=dblp --collection=inproceedings json/inproceedings.json
```

NOTA: Para facilitar su ejecución, hemos encapsulado sendas ejecuciones sobre un script: ïmport.sh".

Una vez cargado, para comenzar a ejecutar comandos y queries sobre nuestros datos, debemos indicar la bbdd que vamos a utilizar, en nuestro caso "dblp", que es la que hemos indicado en los comandos de mongoimport. Lo indicamos mediante el comando siguiente:

```
use dblp
```

Como salida, obtendremos:

```
switched to db dblp
```

Al haber cargado los datos a partir del formato indicado en los ficheros json, los *array* de autores han sido cargados como *arrays* de objetos, los cuales tienen un campo llamado **__VALUE** que contiene la cadena de texto. Es por ello que aplicamos un preprocesamiento para convertir estos campos en *arrays* formados por cadenas de texto.

El código utilizado para ello:

```
db.articles.aggregate([{$addFields: {author: '$author._VALUE'}}, {$out: "articles"}])
```

```
db.incollections.aggregate([{$addFields: {author: '$author._VALUE'}}, {$out: "incollections"}])
```

```
db.inproceedings.aggregate([{$addFields: {author: '$author._VALUE'}}, {$out: "inproceedings"}])
```

Además, tambien necesitamos cambiar el tipo del campo año ya que no ha sido reconocido como númerico al realizar la carga. Por ello ejecutamos el siguiente código:

```
db.articles.find().forEach(function(obj){
    db.articles.update(
        {"_id": obj._id, 'year': {$exists : true}},
        {$set: {"year": NumberInt(obj.year)}})
    })
```

```
db.incollections.find().forEach(function(obj){
    db.incollections.update(
        {"_id": obj._id, 'year': {$exists : true}},
        {$set: {"year": NumberInt(obj.year)}})
    })
```

```
db.inproceedings.find().forEach(function(obj){
    db.inproceedings.update(
        {"_id": obj._id, 'year': {$exists : true}},
        {$set: {"year": NumberInt(obj.year)}})
    })
```

Para el análisis de los datos a lo largo de toda la práctica, hemos generado una nueva colección **authors**. Esta colección contiene por cada documento, un autor y la documentación que ha generado. Para generar esta nueva colección, hemos obtenido colección por colección e incluido ese título en función del autor:

```
db.incollections.aggregate([
    {$unwind: "$author"},
    {$group:
        { _id: "$author",
          incollections: {$push: {_id: "$_id", title: "$title", year: "$year"}}}}],
    { allowDiskUse: true })
    .forEach(
        function(obj){
            db.authors.update(
                {_id: obj._id},
                {$set: {"incollections": obj.incollections}},
                {upsert: true})})
```

```
db.articles.aggregate([
    {$unwind: "$author"},
    {$group:
        { _id: "$author",
          articles: {$push: {_id: "$_id", title: "$title", year: "$year"}}}}],
     { allowDiskUse: true })
     .forEach(
        function(obj){
            db.authors.update(
                {_id: obj._id},
                {$set: {"articles": obj.articles}},
                {upsert: true})})
```

```
db.inproceedings.aggregate([
    {$unwind: "$author"},
    {$group:
        {_id: "$author",
         inproceedings: {$push: {_id: "$_id", title: "$title", year: "$year"}}}}],
    { allowDiskUse: true })
    .forEach(function(obj){
        db.authors.update(
            {_id: obj._id},
            {$set: {"inproceedings": obj.inproceedings}},
            {upsert: true})})
```

2. Análisis

2.1. Listado de todas las publicaciones de un autor determinado.

Para esta query vamos a hacer uso de la potencia que nos ofrece el *aggregation framework*. Este framework funciona creando un pipeline en la que definiremos *stages*. La salida de un *stage* pasa a ser la entrada del *stage* siguiente. Con esto podemos realizar sobre los datos operaciones realmente potentes.

Para obtener el listado de las publicaciones, la query que debemos ejecutar es sencilla. En primer lugar filtramos por el nombre de autor que del cual queremos hacer la consulta, y a continuación concatenamos los *arrays* con los títulos de las publicaciones.

```
db.authors.aggregate([
  { $match : { _id : "Chin-Wang Tao" } },
  { $project: {publication: {$concatArrays:
    ["$incollections.title", "$articles.title", "$inproceedings.title"]}}
  }
] )
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "Chin-Wang Tao",
    "publication" : [
        "iPhone as Multi-CAM and Multi-viewer.",
        "On the robustness of stability of fuzzy control systems.",
        "Adaptive fuzzy PIMD controller for systems with uncertain deadzones.",
        "Design and analysis of region-wise linear fuzzy controllers.",
```

```
"Adaptive Fuzzy Switched Swing-Up and Sliding Control for the Double-Pendulum-and-Cart
↪  System.",
"An Approximation of Interval Type-2 Fuzzy Controllers Using Fuzzy Ratio Switching
↪  Type-1 Fuzzy Controllers.",
"Adaptive fuzzy terminal sliding mode controller for linear systems with mismatched
↪  time-varying uncertainties.",
"A reduction approach for fuzzy rule bases of fuzzy controllers.",
"Fuzzy Sliding-Mode Formation Control for Multirobot Systems: Design and
↪  Implementation.",
"Segmentation of Psoriasis Vulgaris Images Using Multiresolution-Based Orthogonal
↪  Subspace Techniques.",
"Fuzzy control for linear plants with uncertain output backlashes.",
"Iris Recognition Using Possibilistic Fuzzy Matching on Local Features.",
"Radial Basis Function Networks With Linear Interval Regression Weights for Symbolic
↪  Interval Data.",
"Adaptive fuzzy sliding mode controller for linear systems with mismatched
↪  time-varying uncertainties.",
"Design of fuzzy controllers with adaptive rule insertion.",
"Flexible complexity reduced PID-like fuzzy controllers.",
"Interval competitive agglomeration clustering algorithm.",
"Fuzzy sliding-mode control for ball and beam system with fuzzy ant colony
↪  optimization.",
"Hybrid robust approach for TSK fuzzy modeling with outliers.",
"Design of a Fuzzy Controller With Fuzzy Swing-Up and Parallel Distributed Pole
↪  Assignment Schemes for an Inverted Pendulum and Cart System.",
"Nested design of fuzzy controllers with partial fuzzy rule base.",
"Fuzzy hierarchical swing-up and sliding position controller for the inverted
↪  pendulum-cart system.",
"Design of a parallel distributed fuzzy LQR controller for the twin rotor multi-input
↪  multi-output system.",
"Fuzzy adaptive approach to fuzzy controllers with spacial model.",
"Simplified type-2 fuzzy sliding controller for wing rock system.",
"Unsupervised fuzzy clustering with multi-center clusters.",
"An advanced fuzzy controller.",
"A Novel Approach to Implement Takagi-Sugeno Fuzzy Models.",
"Fuzzy Swing-Up and Fuzzy Sliding-Mode Balance Control for a Planetary-Gear-Type
↪  Inverted Pendulum.",
"A Design of a DC-AC Inverter Using a Modified ZVS-PWM Auxiliary Commutation Pole and
↪  a DSP-Based PID-Like Fuzzy Control.",
"Robust fuzzy control for a plant with fuzzy linear model.",
"Comments on \"Reduction of fuzzy rule base via singular value decomposition\".",
"A Novel Fuzzy-Sliding and Fuzzy-Integral-Sliding Controller for the Twin-Rotor
↪  Multi-Input-Multi-Output System.",
"Robust L",
"A Fuzzy Logic Approach to Target Tracking",
"Index compression for vector quantisation using modified coding tree assignment
↪  scheme.",
"Hybrid SVMR-GPR for modeling of chaotic time series systems with noise and
↪  outliers.",
"Parallel Distributed Fuzzy Sliding Mode Control for Nonlinear Mismatched Uncertain
↪  Systems.",
"An approach for the robustness comparison between piecewise linear ",
"Interval fuzzy sliding-mode formation controller design.",
"Simplification of a fuzzy mechanism with rule combination.",
"Robust control of systems with fuzzy representation of uncertainties.",
```

```
        "Checking identities is computationally intractable NP-hard and therefore human
        ↪  provers will always be needed.",
        "A kernel-based core growing clustering method.",
        "An Alternative Type Reduction Approach Based on Information Combination with Interval
        ↪  Operations for Interval-Valued Fuzzy Sliding Controllers.",
        "Designs and analyses of various fuzzy controllers with region-wise linear PID
        ↪  subcontrollers.",
        "Adaptive Bound Reduced-Form Genetic Algorithms for B-Spline Neural Network
        ↪  Training.",
        "A New Neuro-Fuzzy Classifier with Application to On-Line Face Detection and
        ↪  Recognition.",
        "Statistical and Dempster-Shafer Techniques in Testing Structural Integrity of
        ↪  Aerospace Structures.",
        "Embedded support vector regression on Cerebellar Model Articulation Controller with
        ↪  Gaussian noise.",
        "Iris Recognition Using Gabor Filters and the Fractal Dimension.",
        "A two-stage design of adaptive fuzzy controllers for time-delay systems with unknown
        ↪  models.",
        "Errata: Iris recognition using Gabor filters optimized by the particle swarm
        ↪  algorithm.",
        "Texture classification using a fuzzy texture spectrum and neural networks.",
        "Iris recognition using Gabor filters optimized by the particle swarm algorithm.",
        "Sliding control for linear uncertain systems.",
        "FPGA implementation of improved ant colony optimization algorithm for path
        ↪  planning.",
        "Medical image compression using principal component analysis.",
        "Design of a Kinect Sensor Based Posture Recognition System.",
        "A simplified interval type-2 fuzzy CMAC.",
        "iOS based Multi-Cloud Manage System.",
        "A Novel Approach to Implement Takagi-Sugeno Fuzzy Models.",
        "Path-planning using the behavior cost and the path length with a multi-resolution
        ↪  scheme.",
        "Iris recognition using Gabor filters optimized by the particle swarm technique.",
        "Hybrid robust LS-SVMR with outliers for MIMO system.",
        "Design of A Two-Stage Adaptive Fuzzy Controller.",
        "Face Detection Using Eigenface and Neural Network.",
        "Robust control of the mismatched systems with the fuzzy integral sliding
        ↪  controller.",
        "Support Vector Regression for Controller Approximation.",
        "Robust clustering algorithm for the symbolic interval-values data with outliers."
    ]
}
```

2.2. Número de publicaciones de un autor determinado.

Esta consula es similar a la anterior, pero debemos añadir un paso más en nuestro *pipeline* en el cual contaremos el número de puclicaciones.

```
db.authors.aggregate([
  { $match : { _id : "Chin-Wang Tao" } },
  { $project: {publications: {$concatArrays:
    ["$incollections.title", "$articles.title", "$inproceedings.title"]}}
  },
  {$project: {number_of_publications: {$size: "$publications"}}}
] )
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "Chin-Wang Tao",
    "number_of_publications" : 70
}
```

2.3. Número de artículos en revistas para el año 2017.

```
db.articles.find({year: 2017}).count()
```

Aprovechando que esta query es simple, podemos ver la importancia que adquieren los índices en cuanto a tiempo de ejecución. Para ello, generamos el índice sobre el campo year de la siguiente forma:

```
db.articles.createIndex({year:1})
```

```
/* 1 */
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1.0
}
```

Vemos que el resultado es satisfactorio y que a partir de este momento, además del índice originario (sobre _id), tenemos índice sobre year.

La comparativa en tiempos de ejecución es la siguiente:

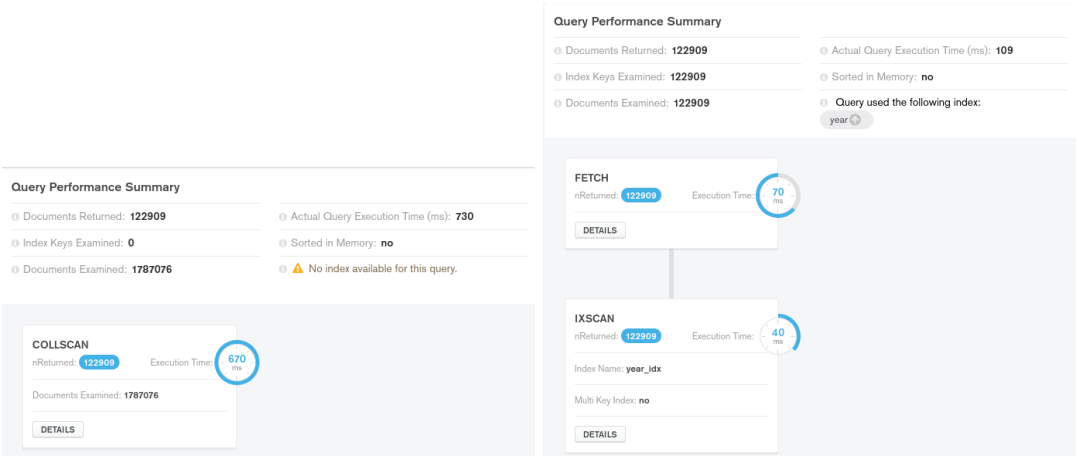

Figura 1: A la izquierda, la query sin índice, a la derecha, aplicando un índice sobre la columna year.

El tiempo de respuesta es casi 7 veces inferior al hacer uso del índice.

Como resultado, independientemente del uso de índice, obtenemos:

```
136121
```

2.4. Número de autores ocasionales (menos de 5 publicaciones).

En esta ocasión no hará falta hacer uso del *aggregation framework*, si no que haremos una búsqueda utilizando la función **find()**.

```
db.authors.find({$expr: {$lt: [{$concatArrays: ["$incollections", "$articles",
  "$inproceedings"]}, 5]}}).count()
```

Como resultado obtenemos:

```
1988889
```

2.5. Número de artículos y de artículos en congresos de los diez autores con más publicaciones en total.

Volvemos a usar *aggregation framework* y construimos un *pipeline* con varios pasos. Primero contamos el número de cada tipo de publicacion de cada uno de los autores. A continuación, creamos un nuevo campo con el con **$addFields** con el total de publicaciones. Con **$sort** y **$limit** ordenamos por el campo recién creado y nos quedamos solo con los 10 primeros, es decir, con los autores que más publicaciones tienen. Por último, y con el *stage* **$project**, nos quedamos solo con los campos que nos interesan (el campo **_id** se incluye por defecto si no se especifica lo contrario).

```
db.authors.aggregate(
  [
    {$project: {
      "total_incollections": {$size: {$ifNull: ['$incollections', []]},},
      "total_inproceedings": {$size: {$ifNull: ['$inproceedings', []]},},
      "total_articles": {$size: {$ifNull: ['$articles', []]},}
    }},
    {$addFields: {
      "total_publications": { $add: [ "$total_incollections", "$total_inproceedings",
                  "$total_articles" ] }
    }},
    {$sort : { total_publications : -1 } },
    {$limit : 10},
    {$project: {
      "total_articles": 1,
      "total_inproceedings": 1
    }}
  ],
  { allowDiskUse: true }
)
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "H. Vincent Poor",
    "total_inproceedings" : 477,
    "total_articles" : 1024
}

/* 2 */
{
    "_id" : "Mohamed-Slim Alouini",
```

```
    "total_inproceedings" : 533,
    "total_articles" : 600
}

/* 3 */
{
    "_id" : "Philip S. Yu",
    "total_inproceedings" : 707,
    "total_articles" : 389
}

/* 4 */
{
    "_id" : "Wen Gao 0001",
    "total_inproceedings" : 734,
    "total_articles" : 332
}

/* 5 */
{
    "_id" : "Wei Wang",
    "total_inproceedings" : 630,
    "total_articles" : 430
}

/* 6 */
{
    "_id" : "Yu Zhang",
    "total_inproceedings" : 616,
    "total_articles" : 429
}

/* 7 */
{
    "_id" : "Lajos Hanzo",
    "total_inproceedings" : 407,
    "total_articles" : 635
}

/* 8 */
{
    "_id" : "Wei Li",
    "total_inproceedings" : 554,
    "total_articles" : 449
}

/* 9 */
{
    "_id" : "Li Zhang",
    "total_inproceedings" : 577,
    "total_articles" : 404
}

/* 10 */
{
```

```
    "_id" : "Yang Liu",
    "total_inproceedings" : 553,
    "total_articles" : 417
}
```

2.6. Número medio de autores por publicación.

En este caso vamos a tener que hacer consulta a varias coleciones y luego hacer calculos con los datos obtenidos en estas consultas.

En primer lugar realizamos consultas en cada una de las colecciones para obtener el número total de autores, incluyendo duplicados. Es decir, si por ejemplo, en un artículo han participado 3 autores, se sumara 3 a la cuenta de autores para esta colección. Estos resultados son sumados y el total lo dividimos por el número total de publicaciones guardadas.

```
(db.articles.aggregate([
  {$match: {author : {$exists: true}}},
  {$project: {num_authors: { $size: '$author' }}},
  {$group: {_id: '', num_authors: {$sum: '$num_authors'}}},
  {$project: {_id: 0, 'num_authors': '$num_authors'}}])
  .next()['num_authors'] +
db.incollections.aggregate([
  {$match: {author : {$exists: true}}},
  {$project: {num_authors: { $size: '$author' }}},
  {$group: {_id: '', num_authors: {$sum: '$num_authors'}}},
  {$project: {_id: 0, 'num_authors': '$num_authors'}}])
  .next()['num_authors'] +
db.inproceedings.aggregate([
  {$match: {author : {$exists: true}}},
  {$project: {num_authors: { $size: '$author' }}},
  {$group: {_id: '', num_authors: {$sum: '$num_authors'}}},
  {$project: {_id: 0, 'num_authors': '$num_authors'}}])
  .next()['num_authors'])/
(db.articles.find().count() +
  db.incollections.find().count() +
  db.inproceedings.find().count())
```

Como resultado obtenemos:

```
2.947463651066849
```

2.7. Lista de coautores de un autor.

Para poder recuperar esta información, será necesario acceder a cada una de las publicaciones de la lista de publicaciones de un autor, "seguir" la referencia para obtener la publicación, y extraer los autores de esta. Para poder "seguir" una referencia a otro documento, *aggregation framework* cuenta con el *stage* **$lookup**, al que le decimos en que colección buscar y que campos tiene que utilizar a la hora de realizar la busqueda.

```
db.authors.aggregate([
  { $match : { _id : "Chin-Wang Tao" } },
  {$lookup:
    {
      from: "articles",
      localField: "articles._id",
      foreignField: "_id",
      as: "articles_detail"
```

```
    }
  },
  {$lookup:
    {
      from: "incollections",
      localField: "incollections._id",
      foreignField: "_id",
      as: "incollections_detail"
    }
  },
  {$lookup:
    {
      from: "inproceedings",
      localField: "inproceedings._id",
      foreignField: "_id",
      as: "inproceedings_detail"
    }
  },
  {$project:{
      _id : 1,
      authors_detail : { $concatArrays: [ "$articles_detail", "$incollections_detail",
                      "$inproceedings_detail" ] }
    }
  },
  {$project: {
    "_id": 0,
    "results": {
      $reduce: {
        input: "$authors_detail.author",
        initialValue: [],
        in: { $concatArrays : ["$$value", "$$this"] }
      }
    }
  }
  },
  { $unwind : "$results" },
  {$group: {
    _id: "$results",
    }
  },
  {$group: {
      _id: null,
      coauthors: { $push: { _id: "$_id"} }
    }
  },
  {$project: {
      _id: 0,
      "coauthors": "$coauthors._id"
    }
  }
])
```

Como resultado obtenemos:

```
/* 1 */
{
```

```
    "coauthors" : [
        "T. H. Su",
        "W. C. Cheng",
        "C. C. Tsai",
        "Meng-Hua Lai",
        "W. Z. Chen",
        "Po-Chun Wang",
        "Song-Shyong Chen",
        "Wen-Rong Xiao",
        "Tzuen Wuu Hsieh",
        "Jin-Tsong Jeng",
        "Hung-Wei Lin",
        "Gwo-Her Lee",
        "Tsu-Tian Lee",
        "Ching-Wen Yang",
        "Chia-Chu Hsu",
        "Ze-Si Huang",
        "Chung-Chih Tsai",
        "Chin-Wang Tao",
        "Min Da Nian",
        "Chien-Chou Chen",
        "Heng-Yi Lin",
        "Mei-Lang Chan",
        "C. L. Tsai",
        "Jin-Shiuh Taur",
        "Jyun-Long Wu",
        "Chia-Wen Chang",
        "Chen-Guan Chang",
        "Chen-Chien J. Hsu",
        "Yeong-Hwa Chang",
        "Chunlin Chen",
        "Ching Ho Chi",
        "Huei-Rong Chen",
        "Chen-Chia Chuang",
        "W. Y. Wang",
        "Shun-Feng Su",
        "Yi-Fang Chen",
        "Rustom Mamlook",
        "Meng-Cheng Yang",
        "J. H. Chang",
        "Y. C. Chen",
        "Chien-Ming Wang",
        "Yung-Chih Liu",
        "U. S. Chen",
        "Ana C. Holguin",
        "Yao-Chu Hsueh",
        "Sun-Yuan Kung",
        "Cheng-Yuan Yang",
        "C. M. Wang",
        "Vladik Kreinovich",
        "Wei-Yen Wang",
        "Chih-Ching Hsiao",
        "Roberto A. Osegueda",
        "Seetharami R. Seelam",
        "Yi-Hsing Chien",
```

```
        "Wiley E. Thompson",
        "Hung T. Nguyen 0002",
        "Ru-Yu Hou"
    ]
}
```

2.8. Edad de los 5 autores con el periodo de publicaciones más largo.

En primer lugar, y usando las facilidades de *aggregation framework*, agrupamos todas las publicaciones de un mismo autor en una lista única. A continuación nos quedamos de estas listas solo con el año de publicación de estas, y filtramos tanto el máximo como el mínimo de de estos años. Ya con esto, podemos calcular la diferencia entre ambos años y realizar una ordenación a partir de este campo para quedarnos con los cinco primeros.

```
db.authors.aggregate(
  [
    {$project: {
      publication: {$concatArrays: [
        {$ifNull: ['$incollections', []]},
        {$ifNull: ['$inproceedings', []]},
        {$ifNull: ['$articles', []]}]}
    }},
    {$addFields: {
      max_publication: { $max: "$publication.year"},
      min_publication: { $min: "$publication.year"}
    }},
    {$addFields: {
      age: { $subtract: [ "$max_publication", "$min_publication"] }
    }},
    {$sort : { age : -1 } },
    {$limit : 5},
    {$project: {
      _id: 1,
      age: 1
    }}
  ],
  { allowDiskUse: true }
)
```

Como resultado obtenemos:

```
/* 1 */
{
    "_id" : "Alan M. Turing",
    "age" : 75
}

/* 2 */
{
    "_id" : "Rudolf Carnap",
    "age" : 71
}

/* 3 */
{
    "_id" : "David Nelson",
```

```
    "age" : 68
}

/* 4 */
{
    "_id" : "Eric Weiss",
    "age" : 64
}

/* 5 */
{
    "_id" : "Bernard Widrow",
    "age" : 64
}
```

2.9. Número de autores novatos.

Caso idéntico al anterior, cambiando las fases finales para quedarnos con aquellos cuya "edad" es inferior a 5 años, y contar el número de elementos que obtenemos.

```
db.authors.aggregate(
  [
    {$project: {
      publication: {$concatArrays: [
        {$ifNull: ['$incollections', []]},
        {$ifNull: ['$inproceedings', []]},
        {$ifNull: ['$articles', []]}]}
    }},
    {$addFields: {
      max_publication: { $max: "$publication.year"},
      min_publication: { $min: "$publication.year"}
    }},
    {$addFields: {
      "age": { $subtract: [ "$max_publication", "$min_publication"] }
    }},
    {'$match':{'age': {'$lt': 5}}}
  ],
  { allowDiskUse: true }
).itcount()
```

Como resultado obtenemos:

```
1503034
```

2.10. Porcentaje de publicaciones en revistas con respecto al total de publicaciones.

Al igual que hicimos cuando calculamos el número medio de autores, que tuvimos que hacer varias queries a cada una de las colecciones, para este caso vamos a volver a operar con diversas queries aplicadas sobre las distintas colecciones. Necesitaremos obtener el número de documentos en cada colección y con estos datos podemos calcular el porcentaje de publicaciones en revistas como número de estas dividido entre el número total de publicaciones.

```
db.articles.find().count()*100/(
  db.articles.find().count() +
  db.incollections.find().count() +
  db.inproceedings.find().count())
```

Como resultado obtenemos:

```
43.67313322575196
```

3. Neo4J

En esta segunda parte vamos a utilizar los mismos datos sobre publicaciones, pero en este caso, vamos a utilizar una BBDD orientada a grafos. Este tipo de BBDD se engloba dentro de las BBDD NoSQL. Este tipo de BBDD son útiles cuando lo que nos importa son la relaciones entre las entidades que guardamos.

Así como MongoDB es la BBDD NoSQL más importante en cuanto a terminos de uso, podemos decir que **Neo4J** es la principal BBDD orientadas a grafos.

En Neo4J podemos definir:

- Nodos: Serían los elementos que conforman nuestros nuestros datos. Estos elementos pueden tener definidos una serie de atributos los cuales definen ciertas características de estos nodos. Algunos ejemplos de nodos pueden ser personas, empresas, productos, ...
- Relaciones; Como su nombre indica, se trata del enlace, la relación, que unen dos nodos. Al igual que estos, pueden tener una serie de atributos definidos. Un ejemplo de estas relaciones puede ser la amistad entre una persona y otra, o la relación entre cliente y aquel que oferta un servicio. En el caso particular de Neo4J, estas ralaciones se definen de manera unidirecional, esto quiere decir que, el hecho de que el nodo X tenga una relación R con Y, no quiere decir que esta misma relación exista entre Y y X (es importante el orden).

El resultado de unir los elementos anteriores es lo que se denomina: un grafo.

Neo4J nos permite crear lo que se conoce como *LABELS*. Estos *LABLES* son asignados a los distintos nodos y podemos usarlos para crear subgrafos dentro de nuestro grafo. También podemos, como ocurre con otras BBDD, crear índices sobre cualquiera de los atributos que caracterizan a un nodo.

Lo dicho en el parrafo anterior también se aplica a las *relaciones*, pero en este caso, las *LABELS* se les conoce como *TYPES*

Por último señalar que el lenguaje que se utiliza para realizar consultas sobre Neo4J se conoce como **Cypher**. Este lenguaje es muy parecido a Structured Query Language (SQL), lo cual facilita su uso a aquellos acostumbrados a trabajar con BBDD SQL.

3.1. Diseño

Como ya nos ocurría con MongoDB, aunque usa de las características de Neo4J es que se trata de una BBDD *schemaless*, es necesario diseñar la estructura con la que se guardarán los datos.

Como comentabamos en el apartado anterior, en Neo4J podemos definir *nodos* y *relaciones* entre estos nodos, caracterizar a estos mediante atributos y definir subgrados utilizando *:LABELS o :TYPES*.

Para nuestro diseño, vamos a guardar cada publicación en un nodo, al que se le añadirán el resto de datos de la publicación como si fueran atributos. Con los autores haremos exactamente lo mismo. A estos nodos se les añadirán las siguientes *:LABELS* según corresponda:

- Author: A todos los nodos que almacenen la información de un autor.
- Publication: A todos los nodos que representen una publicación sin importar el tipo.
- Article / Inproceeding / Incollection: Estás *LABELS* se asignarán según sea el tipo de publicación.

Solo tendremos un tipo de relación, **:WRITED**, y que se dara entre el nodo de un autor y el nodo de una publicación, en la dirección del primero al segundo.

Por último, comentar los índices que se van a definir. Neo4J permite definir índices sobre los atributos de un nodo, de forma análoga a los índices que podemos definir sobre columnas en otras BBDD. Nosotros vamos a definir un índice sobre el atributo **year** de las publicaciones, lo cuál, acelerará la ejecución de las queries que hagan uso de este atributo. Haremos lo mismo sobre el atributo **name** de los autores.

Otra opción hubiera sido almacenar los años como si fueran nodos y crear una relación entre ellos y las publicaciones. Estos nodos tendrían una gran cantidad de relaciones con nodos de publicaciones asociadas. Para evitar esto, hemos optado por hacer uso de los índices y considerar este año como un atributo de la publicación.

3.2. Carga de los datos.

La carga de datos se ha hecho a partir del mismo fichero eXtendible Markup Language (XML) que contabamos al inicio del capítulo anterior. En este caso, el formato intermedio usado antes de realizar la carga ha sido Comma Separated Value (CSV).

Antes de explicar como se ha realizado esta carga, quisieramos comentar que uno de los problemas con el que nos hemos encontrados, y que más dolores de cabeza nos ha dado ha sido precisamente la carga de los datos. Las herramientas con las que cuenta Neo4J para realizar esta carga tienen ciertas limitaciones que pueden ser un problema, a saber:

- La carga de dato mediante Cypher utilizando el comando *LOAD CSV* es extremadamente lento cuando tratamos varios millones de items, como es nuestro caso.
- La herramienas por línea de comandos *neo4j-admin*, con su opción *import*, nos obliga a borrar, si existe, el grafo donde queramos guardar los datos. Además, es susceptible a errores, y en caso de no poder procesar algun dato, detiene por completo el proceso.

Así pués, para realizar la carga de nuestros datos, realizamos los siguientes pasos:

1. Procesamos el fichero XML para obtener los datos en formato CSV. Para ello ejecutamos el script *parser.py*. Este script realiza las siguientes acciones:
   a) Lee el fichero XML original y crea a partir de él, un fichero temporal con los caracteres conflictivos escapados y correctamente codificados.
   b) A partir de este fichero temporal, genera tres **datasets** dferentes, compuesto cada uno de ellos por multiples ficheros CSV:
      - El primero, *publications*, con la información sobre las publicaciones.
      - Otro, *authors*, con la información sobre los autores.
      - Y el último, *rel*, con las relaciones entre autores y publicaciones.
2. Desde línea de comandos, eliminamos el grafo.
3. Creamos los ficheros con las cabeceras de nuestros CSVs. Estos no son más que ficheros en este mismo formato con una sola línea en la que se especifica la cabecera de nuestros datasets en un formato adecuado.
4. Ejecutamos el siguiente comando y esperamos que acabe el proceso.

```
neo4j-admin import -nodes:Publication "publications.header.csv,publications/part.*"
↪  -nodes:Author "authors.header.csv,authors/part.*" -relationships:Writed
↪  "rels.header.csv,rels/part.*" --ignore-duplicate-nodes=true
```

Una vez concluida la carga, deberiamos se capaces de acceder a la interfaz web que nos ofrece el propio servidor Neo4j desde cualquier navegador (si lo estamos ejecutando en local, por lo general deberá ser accesible `http://localhost:7474/browser/`) y ver una interfaz muy amigable, en la que poder ejecutar diferentes consultas y ver los datos de manera gráfica.

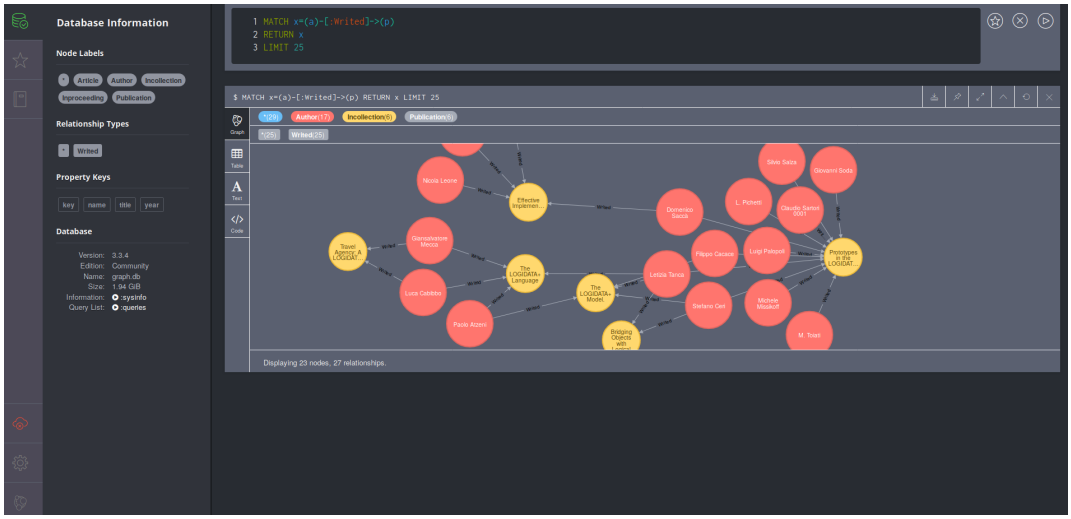

Figura 2: Interfaz web que nos proporciona Neo4j. Una interfaz sencilla desde la cual poder realizar consultas y ver los datos de manera gráfica.

Ya solo nos quedaría crear los índices que habiamos comentados. Las consultas necesarias para crearlos pueden ser ejecutadas desde la misma intefaz web:

```
CREATE INDEX ON :Author(name)
CREATE INDEX ON :Publication(year)
```

4. Análisis

Para este análisis vamos a realizar alguna de las consultas que ya hicimos en MongoDB y así podremos comparar ambas BBDD.

4.1. Listado de todas las publicaciones de un autor determinado.

En primer lugar, vamos a hacer una busqueda de las publicaciones realizadas por un autor en concreto. Para esta busqueda, haremos un filtrado de los nodos que pertenezcan al subgrafo de autores, *:Author*, y sobre estos filtraremos utilizando el atributo *name* de estos nodos. Con esto tendremos el nodo del autor. A partir de este nodo, y siguiendo las relaciones de tipo *:Writed* que tiene definidas como si de un camino se tratase, llegaremos a las publicaciones de este autor.

```
MATCH (:Author {name: 'Chin-Wang Tao'})-[:Writed]->(p:Publication)
RETURN p.title
```

Las publicaciones pertenecientes a este autor son 71, de las cuales podemos ver a continuación una muestra en formato JSON:

```
[
  {
    "keys": [
      "p.title"
    ],
    "length": 1,
    "_fields": [
      "Iris recognition using Gabor filters optimized by the particle swarm algorithm."
    ],
```

```
    "_fieldLookup": {
      "p.title": 0
    }
  },
  {
    "keys": [
      "p.title"
    ],
    "length": 1,
    "_fields": [
      "Texture classification using a fuzzy texture spectrum and neural networks."
    ],
    "_fieldLookup": {
      "p.title": 0
    }
  },
  {
    "keys": [
      "p.title"
    ],
    "length": 1,
    "_fields": [
      "Errata: Iris recognition using Gabor filters optimized by the particle swarm algorithm."
    ],
    "_fieldLookup": {
      "p.title": 0
    }
  },
  ...
  {
    "keys": [
      "p.title"
    ],
    "length": 1,
    "_fields": [
      "iPhone as Multi-CAM and Multi-viewer."
    ],
    "_fieldLookup": {
      "p.title": 0
    }
  }
]
```


Podemos comparar el efecto de aplicar un índice sobre este campo:

- Sin índice: 1620 ms
- Con índice: 80 ms

4.2. Número de artículos en revistas para el año 2017.

Para este caso necesitaremos hacer una agregación. En el caso de Cypher, no contamos con un framework de agregación tan flexible como el que nos ofrece MongoDB, pero aun así podemos realizar ciertas agregaciones. Para poder contar el número de articulos publicados en 2017, filtraremos los nodos pertenecientes al subgrafo *:Article* utilizando el atributo *year*. A continuación utilizamos la funcion **count()** para contar el número de estos.

```
MATCH (p:Article {year: 2017})
RETURN count(p)
```

Y tenemos que el número total de articulos publicados en 2017 es de 136943:


```
[
  {
    "keys": [
      "count(p)"
    ],
    "length": 1,
    "_fields": [
      {
        "low": 136943,
        "high": 0
      }
    ],
    "_fieldLookup": {
      "count(p)": 0
    }
  }
]
```


Como en el caso anterior, comparamos los tiempos de carga con y sin índice:

- Sin índice: 1160 ms
- Con índice: 970 ms

En este caso el índice apenas tiene efecto ya que el número de posibles valores es muy pequeño en comparación al número de elementos.

4.3. Lista de coautores de un autor.

En esta consulta aprovecharemos la potencia de las BBDD orientadas a grafos, las relaciones. Mientras que en MongoDB la query necesaria incluía realizar (en caso de no querer tener demasiados elementos anidados) múltiples operaciones con el framework de agregaciones, en Neo4j podemos obtener la misma información con una query mínima.

```
MATCH (:Author {name: "Chin-Wang Tao"})-[:Writed]->(:Publication)<-[:Writed]-(c:Author)
RETURN c.name
```

La lista de coautores es la siguiente (solo se muestran algunos de los 150 coautores devueltos):


```
[
  {
    "keys": [
      "c.name"
    ],
    "length": 1,
    "_fields": [
      "Jin-Shiuh Taur"
    ],
    "_fieldLookup": {
```

```
      "c.name": 0
    }
  },
  {
    "keys": [
      "c.name"
    ],
    "length": 1,
    "_fields": [
      "Chung-Chih Tsai"
    ],
    "_fieldLookup": {
      "c.name": 0
    }
  },
  ...
  {
    "keys": [
      "c.name"
    ],
    "length": 1,
    "_fields": [
      "Chen-Chia Chuang"
    ],
    "_fieldLookup": {
      "c.name": 0
    }
  }
]
```


En este caso, la ejecución en Neo4j tardó 7 ms, mientras que en MongoDB, el tiempo de ejecución de la consulta similar fue de 5 ms. Se puede entender que el tiempo sea prácticamente idéntico, puesto que este caso de uso auna en una misma consulta relaciones y atributos de las entidades; por lo que el tiempo de ejecucion en MongoDB será muy bueno gracias a la buena gestión de atributos de documentos, mientras que Neo4J toma ventaja de las relaciones entre entidades, y en ambos casos, el uso de índices.

4.4. Edad de los 5 autores con el periodo de publicaciones más largo.

Ahora vamos a intentar realizar una consulta menos amistosa para nuestra BBDD.

```
MATCH (a:Author)-[:Writed]->(p:Publication)
WITH a, max(p.year) - min(p.year) as age
RETURN a.name, age
ORDER BY age DESC LIMIT 5
```

Aunque la consulta a priori parece secilla, incluso más que la que usamos en MongoDB, el tiempo de ejecución y la carga que le supone a la máquina es significativamente mayor, llegando incluso a desbordar en memoria.

Conclusiones

MongoDB y Neo4j son dos BBDD diferentes en su concepción. Mientras que MongoDB está orientada a documentos, Neo4j está orientada a grafos.

Durante la realización de esta práctica hemos podido sacar algunas conclusiones respecto al uso de ambas BBDD:

- La carga de los datos en Neo4j ha sido de largo, más complicada que en MongoDB. Esto se debe a las herramientas disponibles para realizar esta carga de datos, al menos, la primera carga, donde el uso de *LOAD CSV* era inviable por el tiempo de carga. En el caso de hacer uso de *neo4j-admin*, es necesario eliminar el grafo en caso de existir, además de ser sensible a errores (en versiones anteriores existía una opción que permitia definir un número máximo de errores permitidos y así poder tratarlos de manera individual a posteriori).
- La sintaxis que ofrece *Cypher* resulta más sencilla, en especial a aquellas personas que venga del mundo SQL.
- Neo4j demuestra su fortaleza a la hora de realizar consultas sobre relaciones entre nodos, pero palidece en aquellas consulta en las que hay que realizar operaciones con los atributos de estos. El uso de índices resulta útil en aquellos atributos que puedan tomar un gran número de posibles valores.
- MongoDB dispone de un framework para hacer agregaciones muy potente, aunque su curva de aprendizaje es muy empinada al principio.
- En la actualidad, MongoDB está presentando actualizaciones constantemente, ampliando a través de nuevas funcionalidades que los usuarios que venían del mundo SQL parecían echar de menos. Esto está consiguiendo que la expansión de utilización de MongoDB sea a un ritmo frenético.

ANEXOS

A. Creación de vistas

De forma similar que en bases de datos relaciones, es posible crear vistas. Las vistas son consultas sobre diferentes tablas a través de los campos que designemos. Una vez creada una vista, las consultas se realizan de la misma forma que si fuese una colección, pudiendo filtrar por alguno de sus campos.

En nuestro caso, hemos creado la vista *publications_extended*. Esta vista, a partir de la colección *authors* que contiene un documento por cada autor y algunos campos básicos de cada tipo de documento, incluyendo el campo _id, cruzando con el resto de colecciones con diferentes publicaciones, conseguimos unificar toda la información extendida sobre una vista. Al incluir nuevos documentos en las colecciones de origen, la vista automáticamente devuelve dichos registros. La definición es la siguiente:

```
db.createView (
  "publications_extended",
  "authors",
  [
    { $lookup: {
      from: "articles", localField: "articles._id",
    foreignField: "_id", as: "articles_extended" } },
    { $lookup: {
      from: "incollections", localField: "incollections._id",
    foreignField: "_id", as: "incollections_extended" } },
    { $lookup: {
      from: "inproceedings", localField: "inproceedings._id",
    foreignField: "_id", as: "inproceedings_extended" } },
    { $project: {
      articles_extended: 1, incollections_extended: 1,
    inproceedings_extended: 1}}
  ]
)
```

Una vez más, comprobamos que se ha ejecutado el comando correctamente:

```
/* 1 */
{
    "ok" : 1.0
}
```

A partir de este momento, ya se puede ejecutar cualquier consulta a la vista como si de una tabla se tratase. Por ejemplo, podemos ejecutar la consulta del ejercicio 1, *Listado de todas las publicaciones de un autor determinado*:

```
db.publications_extended.aggregate([
    { $match : { _id : "Chin-Wang Tao" } },
    { $project: {publication: {$concatArrays:
        ["$articles_extended.title", "$incollections_extended.title",
         "$inproceedings_extended.title"]}}
    }
])
```

Como resultado, obtenemos el mismo resultado (aunque el orden puede variar):

```
/* 1 */
{
```

```
    "_id" : "Chin-Wang Tao",
    "publication" : [
        "On the robustness of stability of fuzzy control systems.",
        "Adaptive fuzzy PIMD controller for systems with uncertain deadzones.",
        "Design and analysis of region-wise linear fuzzy controllers.",
        "Adaptive Fuzzy Switched Swing-Up and Sliding Control for the Double-Pendulum-and-Cart
        ↪  System.",
        "An Approximation of Interval Type-2 Fuzzy Controllers Using Fuzzy Ratio Switching
        ↪  Type-1 Fuzzy Controllers.",
        "Adaptive fuzzy terminal sliding mode controller for linear systems with mismatched
        ↪  time-varying uncertainties.",
        "A reduction approach for fuzzy rule bases of fuzzy controllers.",
        "Fuzzy Sliding-Mode Formation Control for Multirobot Systems: Design and
        ↪  Implementation.",
        "Segmentation of Psoriasis Vulgaris Images Using Multiresolution-Based Orthogonal
        ↪  Subspace Techniques.",
        "Fuzzy control for linear plants with uncertain output backlashes.",
        "Iris Recognition Using Possibilistic Fuzzy Matching on Local Features.",
        "Radial Basis Function Networks With Linear Interval Regression Weights for Symbolic
        ↪  Interval Data.",
        "Adaptive fuzzy sliding mode controller for linear systems with mismatched
        ↪  time-varying uncertainties.",
        "Design of fuzzy controllers with adaptive rule insertion.",
        "Flexible complexity reduced PID-like fuzzy controllers.",
        "Interval competitive agglomeration clustering algorithm.",
        "Fuzzy sliding-mode control for ball and beam system with fuzzy ant colony
        ↪  optimization.",
        "Hybrid robust approach for TSK fuzzy modeling with outliers.",
        "Design of a Fuzzy Controller With Fuzzy Swing-Up and Parallel Distributed Pole
        ↪  Assignment Schemes for an Inverted Pendulum and Cart System.",
        "Nested design of fuzzy controllers with partial fuzzy rule base.",
        "Fuzzy hierarchical swing-up and sliding position controller for the inverted
        ↪  pendulum-cart system.",
        "Design of a parallel distributed fuzzy LQR controller for the twin rotor multi-input
        ↪  multi-output system.",
        "Fuzzy adaptive approach to fuzzy controllers with spacial model.",
        "Simplified type-2 fuzzy sliding controller for wing rock system.",
        "Unsupervised fuzzy clustering with multi-center clusters.",
        "An advanced fuzzy controller.",
        "A Novel Approach to Implement Takagi-Sugeno Fuzzy Models.",
        "Fuzzy Swing-Up and Fuzzy Sliding-Mode Balance Control for a Planetary-Gear-Type
        ↪  Inverted Pendulum.",
        "A Design of a DC-AC Inverter Using a Modified ZVS-PWM Auxiliary Commutation Pole and
        ↪  a DSP-Based PID-Like Fuzzy Control.",
        "Robust fuzzy control for a plant with fuzzy linear model.",
        "Comments on \"Reduction of fuzzy rule base via singular value decomposition\".",
        "A Novel Fuzzy-Sliding and Fuzzy-Integral-Sliding Controller for the Twin-Rotor
        ↪  Multi-Input-Multi-Output System.",
        "Robust L",
        "A Fuzzy Logic Approach to Target Tracking",
        "Index compression for vector quantisation using modified coding tree assignment
        ↪  scheme.",
        "Hybrid SVMR-GPR for modeling of chaotic time series systems with noise and
        ↪  outliers.",
```

```
        "Parallel Distributed Fuzzy Sliding Mode Control for Nonlinear Mismatched Uncertain
        ↪  Systems.",
        "An approach for the robustness comparison between piecewise linear ",
        "Interval fuzzy sliding-mode formation controller design.",
        "Simplification of a fuzzy mechanism with rule combination.",
        "Robust control of systems with fuzzy representation of uncertainties.",
        "Checking identities is computationally intractable NP-hard and therefore human
        ↪  provers will always be needed.",
        "A kernel-based core growing clustering method.",
        "An Alternative Type Reduction Approach Based on Information Combination with Interval
        ↪  Operations for Interval-Valued Fuzzy Sliding Controllers.",
        "Designs and analyses of various fuzzy controllers with region-wise linear PID
        ↪  subcontrollers.",
        "Adaptive Bound Reduced-Form Genetic Algorithms for B-Spline Neural Network
        ↪  Training.",
        "A New Neuro-Fuzzy Classifier with Application to On-Line Face Detection and
        ↪  Recognition.",
        "Statistical and Dempster-Shafer Techniques in Testing Structural Integrity of
        ↪  Aerospace Structures.",
        "Embedded support vector regression on Cerebellar Model Articulation Controller with
        ↪  Gaussian noise.",
        "Iris Recognition Using Gabor Filters and the Fractal Dimension.",
        "A two-stage design of adaptive fuzzy controllers for time-delay systems with unknown
        ↪  models.",
        "Errata: Iris recognition using Gabor filters optimized by the particle swarm
        ↪  algorithm.",
        "Texture classification using a fuzzy texture spectrum and neural networks.",
        "Iris recognition using Gabor filters optimized by the particle swarm algorithm.",
        "iPhone as Multi-CAM and Multi-viewer.",
        "Sliding control for linear uncertain systems.",
        "FPGA implementation of improved ant colony optimization algorithm for path
        ↪  planning.",
        "Medical image compression using principal component analysis.",
        "Design of a Kinect Sensor Based Posture Recognition System.",
        "A simplified interval type-2 fuzzy CMAC.",
        "iOS based Multi-Cloud Manage System.",
        "A Novel Approach to Implement Takagi-Sugeno Fuzzy Models.",
        "Path-planning using the behavior cost and the path length with a multi-resolution
        ↪  scheme.",
        "Iris recognition using Gabor filters optimized by the particle swarm technique.",
        "Hybrid robust LS-SVMR with outliers for MIMO system.",
        "Design of A Two-Stage Adaptive Fuzzy Controller.",
        "Face Detection Using Eigenface and Neural Network.",
        "Robust control of the mismatched systems with the fuzzy integral sliding
        ↪  controller.",
        "Support Vector Regression for Controller Approximation.",
        "Robust clustering algorithm for the symbolic interval-values data with outliers."
    ]
}
```

B. Parseador de XML a JSON y CSV

Para realizar el parseo de los datos hemos decidido hacer uso de la biblioteca pyspark sobre Python 3 para aprovechar la gestión de la memoria que hace. ¿Por qué de esta decisión?. Intentar procesar el fichero XML

directamente con un script de Python sin uso de la gestión de memoria de las bibliotecas nativas de Spark provocaba que las máquinas donde se ejecuba acabaran bloqueandose debido a que se llegaba al límite de la capacidad de la memoria, y por lo que se ha podido comprobar, los paquetes encargados de este parseo no gestionan correctamente estos escenarios.

Otra alternativa era dividir el fichero XML en bloques y procesarlos de manera individual. Así se reduce la carga de datos en memoria y se consigue no llegar al límite de la máquina. Esta técnica puede se más compleja ya que requiere leer el fichero línea a línea y procesarla para saber si podemos "cortar" sobre ella no separar un elemento valido (un *articles* en dos bloques.

Debido a la facilidad de uso que nos ofrece PySpark, y aprovechando que es contenido de otra materia de este master, lo hemos visto como la opción más recomendable.

B.1. Pasos

Para leer el XML simplemente debemos ejecutar el script adjunto parser.py. En primer lugar, hemos definido las bibliotecas necesarias a utilizar:

```
import os
import pyspark
from tempfile import NamedTemporaryFile
import html

from pyspark.sql import SparkSession, functions
from pyspark.sql.types import StringType, StructType, StructField, ArrayType
from pyspark.sql.functions import explode, concat_ws, lit, translate
```

En segundo lugar, hemos definido el uso de la biblioteca externa de parseo de xml. Para ello, debemos modificar la variable de entorno PYSPARK_SUBMIT_ARGS:

```
packages = "com.databricks:spark-xml_2.11:0.4.1"

os.environ["PYSPARK_SUBMIT_ARGS"] = (
    "--packages {0} pyspark-shell".format(packages)
)
```

Una vez definida la variable de entorno, inicializamos la variable sc, que contendrá de aquí en adelante el SparkContext, objeto necesario para gestionar las llamadas en Spark. Además, definimos el esquema que nos vamos a encontrar mediante la tipología de datos de cada campo:

```
spark = (SparkSession.builder
    .master("local[4]")
    .config("spark.driver.cores", 1)
    .appName("Getting graph information")
    .getOrCreate() )

sc = spark.sparkContext
```

Al no tener un esquema definido, es el propio paquete de **Spark** el que intenta inferirlo por si mismo. Esto causa problemas al intentar parsear algunos campos. Por ello lo mejor es definir nuestro propio esquema definiendo todos los campos como si se tratasen de "Strings". Ya que solo utilizamos este dataframe para transformar el *XML* a *JSON* y *CSV*, y no vamos a realizar ningún tipo de validación u operación con el **Datafame** que se crea, tratar todos los campos como "Strings" es más que suficiente.

```
schema = StructType([
  StructField("_key", StringType()),
  StructField("title", StringType()),
```

```
  StructField("year", StringType()),
  StructField("author", ArrayType(
      StructType([
          StructField("_VALUE", StringType())
      ])
   ))
])
```

Una vez definido, procedemos a abrir el fichero dblp.xml y almacenamos eliminando caracteres de escape en html cada línea en un fichero temporal.

```
with open('./dblp.xml') as source, NamedTemporaryFile('w') as unescaped_src:
    for line in source:
        unescaped_src.write(html.unescape(line))
```

Posteriormente, recuperamos cada tipo de ejecución mediante el siguiente mandato:

```
incollections_df = spark.read.format('com.databricks.spark.xml').option(
        "rowTag", "incollection").option('charset', "UTF-8").schema(
        schema).load(unescaped_src.name)
inproceedings_df = spark.read.format('com.databricks.spark.xml').option(
        "rowTag", "inproceedings").option('charset',
        "UTF-8").schema(schema).load(unescaped_src.name)
articles_df = spark.read.format('com.databricks.spark.xml').option("rowTag",
        "article").option('charset', "UTF-8").schema(schema).load(
        unescaped_src.name)
```

Una vez cargados los DataFrames en función de la tipología de publicación, volcamos cada tipo en 3 ficheros json para la carga desde mongoimport:

```
incollections_df.write.option("charset", "UTF-8").json('./json/incollections')
inproceedings_df.write.option("charset", "UTF-8").json('./json/inproceedings')
articles_df.write.option("charset", "UTF-8").json('./json/articles')
```

Por último, para la carga de datos en Neo4j vamos a generar 3 ficheros CSV, el primero de ellos con todas las publicaciones, el segundo con todos los autores y el tercero con todas las relaciones entre ellos:

```
publications_df =   incollections_df.withColumn('LABEL',lit('Incollection')).union(
                    inproceedings_df.withColumn('LABEL',lit('Inproceeding'))).union(
                    articles_df.withColumn('LABEL', lit('Article')))

publications_df = publications_df.filter(publications_df._key.isNotNull())

publications_df.withColumn('id', translate('_key', '/', '_')).select('id',
    'title', 'year', 'LABEL').write.option('escape', '"').csv(
    './csv/publications')

publications_df.withColumn('_author', explode('author._VALUE')).select(
    '_author').write.option('escape', '"').csv('./csv/authors')

publications_df.withColumn('start', explode('author._VALUE')).withColumn(
    'end', translate('_key', '/', '_')).select('start', 'end').write.option(
        'escape', '"').csv('./csv/rels')
```

Hemos encapsulado la ejecución de este script mediante simplemente la ejecución del script adjunto "parse.sh"mediante su ejecución:

```
./parse.sh
```

Ese fichero eliminará los directorios donde se almacenarán los ficheros de salida, ejecutará el script y agrupará los ficheros temporales *json* en un fichero completo por cada tipología. Para el caso de *CSV*, no ha sido necesario agrupar puesto que en la importación a neo4j indicamos que se recuperen todos los ficheros part-* (ficheros generados por spark).