ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA


Proyecto Fin de Carrera

# Implementación y análisis de prototipos de Sistemas de Procesamiento Distribuido para algortimos tipo *Map&Reduce*

AUTOR: Rubén Conesa Zaplana

DIRECTOR: Javier Garrigós Guerrero

Octubre / 2014


| | |
|---|---|
| **Autor** | Rubén Conesa Zaplana |
| **E-mail del Autor** | rubenconesa@gmail.com |
| **Director del proyecto** | Javier Garrigós Guerrero |
| **E-mail del Director** | |
| **Título del Proyecto** | Implementación y análisis de prototipos de Sistemas de Procesamiento Distribuido para algoritmos tipo *Map&Reduce* |
| **Título del Proyecto en Inglés** | Implementation and analysis of several Distributed Processing System Prototypes to develop Map&Reduce algorithms |
| **Resumen** | Este proyecto tiene como objeto el estudio de los modelos de computación paralela para su aplicación en tareas de análisis de datos que mejoren la inteligencia de negocio en las PYMES.<br><br>En la actualidad, muchas empresas comienzan a tener grandes volúmenes de datos que son difíciles de manejar con las arquitecturas de cómputo convencionales. Sin embargo, los modernos sistemas de procesamiento distribuido, que pueden utilizar de forma transparente incluso cientos de servidores convencionales, las bases de datos distribuidas y redundantes y los algoritmos de computación paralela sobre múltiples datos (SIMD, *Single Instruction Multiple Data*) tipo *Map&Reduce* pueden incrementar la velocidad de cómputo para permitir que volúmenes de varios cientos de Gigabytes de datos puedan ser tratados óptimamente. Diferentes áreas y tareas de negocio en la empresa pueden beneficiarse del procesamiento de estas ingentes cantidades de información para mejorar la inteligencia de negocio. Algunos ejemplos son: el análisis de datos estadísticos sobre pedidos, equipos informáticos, gestión de stocks, etc. |
| **Titulación** | Ingeniería Técnica de Telecomunicación, Especialidad Telemática |
| **Departamento** | Electrónica, Tecnología de Computadoras y Proyectos |
| **Fecha de Presentación** | |

# Índice

# 1. Introducción y Objetivos

En la era de la información se generan grandes volúmenes de datos que toda mediana empresa debería almacenar y procesar, pero que se están dejando escapar en muchas ocasiones por ser difíciles de almacenar y procesar con las arquitecturas de cómputo convencionales.

Por ejemplo, las bases de datos que almacenan datos y metadatos sobre clientes, miembros, ciudadanos o cualquier tipo de usuario abarcan distintas operaciones como la gestión de datos de suscriptores, la gestión de relaciones con los clientes, los perfiles de los usuarios, la biometría, gestión de identidad y acceso y la planificación de recursos empresariales.

Anteriormente se registraba la información básica de los usuarios (nombre, dirección,..) pero en la actualidad tenemos acceso a nuevos tipos de datos (ubicación, preferencias de marca, hilos de twitter,…), por lo tanto se deben modificar las bases de datos para poder capturar y procesar estos nuevos datos, pero las bases de datos relaciones no se prestan a este tipo de desarrollo iteractivo, ya que se diseñaron para soportar estructuras fijas de datos bien estructurados en campos y tablas

Sin embargo, los modernos sistemas de procesamiento distribuido, que pueden utilizar de forma transparente incluso cientos de servidores convencionales, las bases de datos distribuidas y redundantes y los algoritmos de computación paralela sobre múltiples datos tipo *Map&Reduce* pueden incrementar la velocidad de cómputo para permitir que volúmenes de varios cientos de Gigabytes de datos puedan ser tratados óptimamente.

Diferentes áreas y tareas de negocio en la empresa pueden beneficiarse del procesamiento de estas ingentes cantidades de información para mejorar la inteligencia de negocio. Algunos ejemplos son: el análisis de datos estadísticos sobre pedidos, equipos informáticos, gestión de stocks, etc.

## 1.1.Big Data

Una definición razonable es: Conjunto de datos demasiado grandes para ser capturados, gestionados y procesados en un tiempo razonable por superar la capacidad del software y hardware habitual. Es una definición reciente, hasta hace poco años no existía preocupación por los Big Data como se observa en el siguiente gráfico correspondiente a la búsquedas del citado término en el buscador google a lo largo del tiempo.

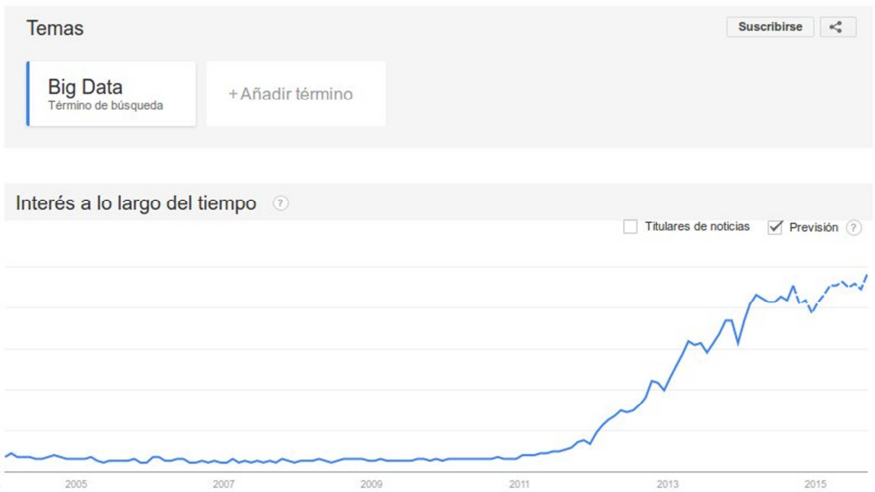


Ilustración 1. Captura de pantalla

Las tres características que definen Big Data se conocen como las tres V: volumen, variedad y velocidad. En 2001, años antes que el marketing acuñara el término Big Data, la empresa analista META Group publicó un informe llamado "3-D DATA Management: Controlling Data Volume, Velocity and Variety", que trataba de los retos del almacenamiento de datos, y el camino para el uso de tecnologías relacionales para superarlos. Aunque las tres V de este documento eran algo distintas, quedó como base para la definición de las tres V del Big Data.

- Volumen. Tamaño de los datos

  Tal vez sea la característica más inmediata, ya que hablamos siempre de grandes cantidades de datos. El tamaño de los datos disponibles ha estado creciendo a un ritmo creciente.

- Variedad. Formato y Origen de los datos

  Los datos actuales tienen formatos y orígenes muy distintos, se trata de textos, imágenes, audio, datos de sensores… que no entran dentro de la tipología que exigen los esquemas relacionales.

- Velocidad. Generación y procesamiento de los datos.

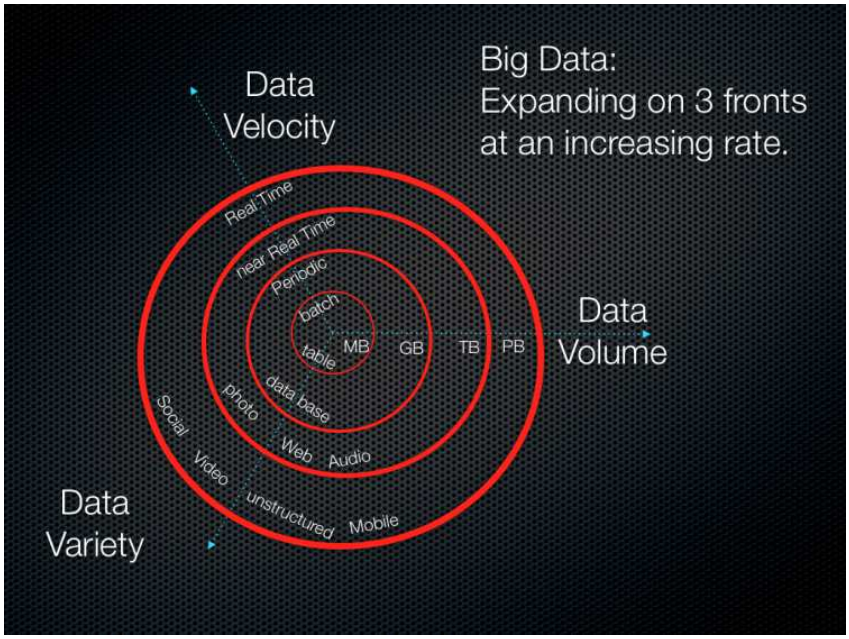


**Ilustración 2. [1]**

Los datos que generamos de forma individual (fotos, videos, llamadas de teléfono, emails, documentos,...) ha aumentado vertiginosamente, pero la evolución de las máquinas ha provocado un aumento de datos generados por ellas incluso mayor que el generado por los humanos, por citar algunos ejemplos: logs, RFID, sensores de red, rutas del GPS, transacciones mercantiles....Los medios de almacenaje han ido aumentando su tamaño para responder al crecimiento de los datos, el problema radica en que no se ha evolucionado en la misma proporción la velocidad de acceso y procesamiento de los datos. Por ejemplo un típico disco rígido de 1990 podía almacenar 1370 megabytes y poseía una velocidad de

transferencia de 4,4 Mb/seg, veinte años después los discos de un terabytes y de 100 Mb/seg son la norma.

La realización de cálculos a gran escala es difícil. Para trabajar con este volumen de datos se requiere la distribución de las partes del problema a varios equipos para procesarlos en paralelo. En un entorno distribuido los fallos parciales son comunes, cuando ocurren estos fallos el resto del entorno debe ser capaz de recuperarse y seguir avanzando la tarea. Por supuesto proporcionar esta resistencia es un gran reto de ingeniería de software. Una solución común es la replicación de datos.

Además de preocuparse por este tipo de errores y problemas, también está el hecho de que el hardware de cómputo tiene recursos limitados. Los principales recursos son: Tiempo de procesador, memoria, espacio en disco duro, ancho de banda de red. Para tener éxito, un sistema distribuido a gran escala debe ser capaz de gestionar los recursos antes mencionados de manera eficiente. Además, se debe asignar algunos de estos recursos a mantener el sistema en su conjunto, mientras dedica tanto tiempo como sea posible para el procesamiento de los datos.

En la segunda jornada del congreso “BDigital Global Congress” celebrado en Barcelona durante el 27, 28 y 29 de mayo de 2014 dejó patente que la tecnología está preparada para afrontar los retos del Big Data, pero el camino para su explotación económica es largo. Para empezar un dato recurrente el 90% de los consumidores espera la personalización de los servicios y productos pero sólo el 32% de las empresas cree estar capacitada para cubrir esta demanda. [2]

## 1.2.Map&Reduce

Map&Reduce es un modelo de programación utilizado por Google para dar soporte a la computación paralela sobre grandes colecciones de datos en grupos de computadoras. El nombre del framework está inspirado en los nombres de dos importantes métodos de programación funcional: *Map* y *Reduce*. Los usuarios especifican una función *Map* donde se procesan pares de clave/valor y se generan pares intermedios de clave/valor. La función *Reduce* fusiona todos los valores intermedios asociados con la misma clave.

Fue desarrollado por Google para la indexación de páginas web, y remplazó sus algoritmos y heurística de indexación originales en 2004.

Durante años se han implementado programas específicos para procesar datos de formatos de muy distintos tipos. Estas implementaciones son conceptualmente sencillas, pero cuando se procesa gran cantidad de datos debemos distribuir el procesamiento por diferentes equipos. Map and Reduce define un sistema distribuido apropiado para esta tarea.

Cualquier implementación del paradigma Map&Reduce, es una gran abstracción para el programador, que a diferencia de antes se centra en cómo resolver un problema utilizando solo funciones *Map* y *Reduce*. Parte de la abstracción incluye la partición de los datos, asignación y monitoreo de tareas, así como la distribución de los datos a las máquinas utilizando un sistema de archivos distribuidos.

Ejemplo sencillo. Se quiere averiguar el número de post que contiene un blog con Map&Reduce, para ello se siguen los siguientes pasos:

- La función Map se encarga de asociar a cada post un 1, crea un par clave/valor donde cada post es la clave y su valor es 1.
- La función Reduce suma todos los 1 para obtener el número de post del blog.

MapReduce es apropiado para problemas que tienen que analizar todo el conjunto de datos, en particular para análisis *ad hoc*, mientras que las bases de datos relacionales (RDBMS) lo es para queries puntuales y actualizaciones, donde el conjunto de datos se ha indexado. En las aplicaciones que implementan Map&Reduce los datos se escriben una vez y se consultan muchas veces, mientras que en RDBMS los datos se actualizan constantemente.

| | RDBMS | MapReduce |
|---|---|---|
| Tamaño datos | Gigabytes | Petabytes |
| Acceso | Interactiva y batch | Batch |
| Actualizaciones | Lectura y escritura muchas veces | Escritura una vez y lectura muchas veces |
| Estructura | Esquema estático | Esquema dinámico |
| Integridad | Alta | Baja |
| Escala | No lineal | Lineal |

### 1.2.1. Ejecución

Cuando se ejecuta un programa sobre Map&Reduce, se dividen los datos de entrada al programa en paquetes de entre 16 y 64 megabytes, este tamaño es elegido por el usuario mediante un parámetro opcional. Estos paquetes son distribuidos junto al programa a ejecutar a los equipos que forman el clúster, menos a un equipo que actúa como maestro para la asignación de las tareas a cada uno de los restantes equipos.

Los equipos asignados para realizar la función Map la ejecutan almacenando los resultados en memoria. Estos resultados, que son los pares intermedios de clave/valor, son enviados de forma periódica al equipo maestro para que pueda transmitirlos a los equipos encargados de la función Reduce.

Cuando el Reduce es notificado con la localización de los pares intermedios, mediante llamadas de proceso remoto lee los datos en los discos locales de los equipos que se encargaron de la función Map y los ordenan. Si la cantidad de datos intermedios es demasiado grande se usa una aplicación externa de ordenación. Se ejecuta el proceso Reduce sobre los datos ordenados obteniendo un fichero que se envía al maestro.[3]

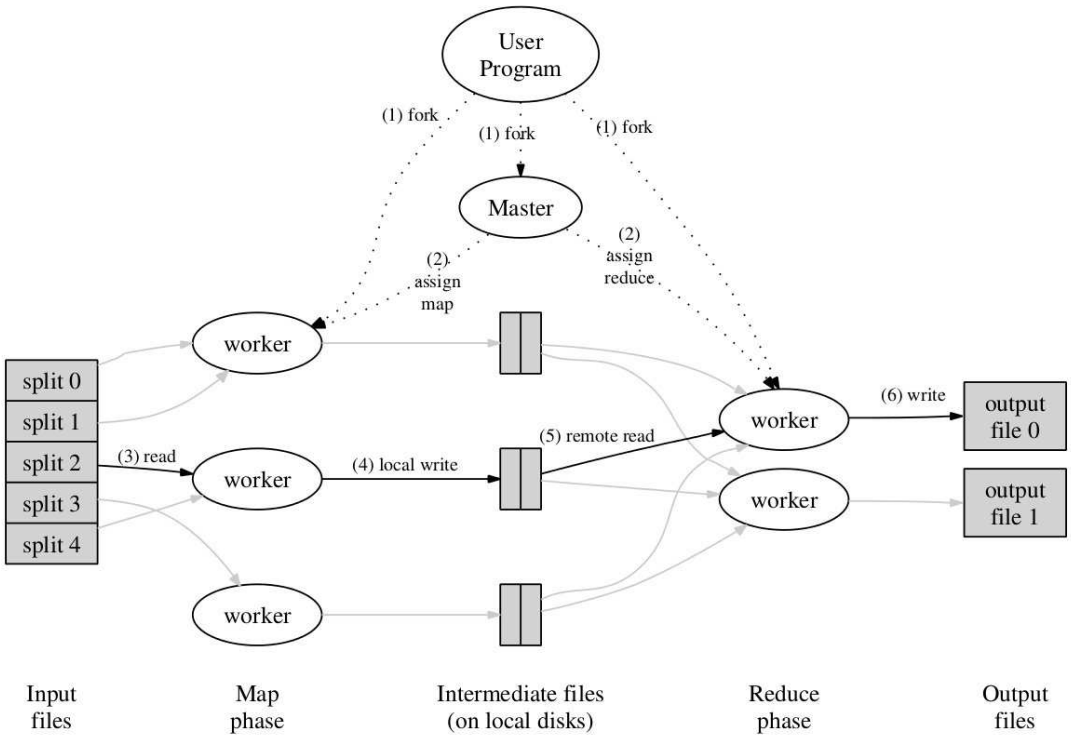


Ilustración 3. [3]

### 1.2.2. Estructura de datos

El equipo maestro mantiene varias estructuras de datos. Por cada tarea Map y tarea Reduce almacena su estado (inactivo, en proceso o terminado) y una entidad por máquina, no por tarea (una máquina puede tener varias tareas). También almacena la localización de los datos intermedios generados por las tareas Map.

Los paquetes de datos de entrada se replican, creando dos copias adicionales de cada paquete. El total de los paquetes, tanto originales como replicas, se distribuyen entre los equipos secundarios.

Se han escrito implementaciones de bibliotecas de Map&Redece en diversos lenguajes de programación como C++, Java y Python, entre otros muchos, lo que facilita enormemente la tarea a los desarrolladores.

## 1.3.Objetivos

El objetivo del presente proyecto es el estudio de los diferentes sistemas de procesamiento distribuido, bases de datos distribuidas y los algoritmos de computación paralela sobre gran cantidad de datos tipo Map&Reduce para su posterior análisis sobre la idoneidad de la aplicación de estos frameworks en las PYMEs. Para ello se implementarán varios prototipos para realizar diferentes pruebas de rendimiento.

# 2. Estado de la técnica

En la actualidad existen gran variedad de sistemas de archivos distribuidos, bases de datos distribuidos y algoritmos de computación paralela. A continuación se hace una breve reseña de algunos de ellos.

## 2.1.Cassandra

Apache Cassandra es una base de datos NoSQL distribuida y basada en un modelo de almacenamiento de «clave-valor», escrita en Java. Permite grandes volúmenes de datos en forma distribuida. Su objetivo principal es la escalabilidad lineal y la disponibilidad. La arquitectura distribuida de Cassandra está basada en una serie de nodos iguales que se comunican con un protocolo P2P con lo que la redundancia es máxima.

Cassandra está desarrollada por Apache Software Foundation.

Cassandra no soporta uniones o subconsultas, salvo para el análisis por lotes a través de Hadoop. [4]

## 2.2.VoltDB

VoltDB implementa el diseño de H-Store.

Es una base datos en memoria diseñada por varios conocidos investigadores de bases de datos como Michael Stonebraker, Sam Madden y Daniel Abadi. Se trata de una base de datos que se rige por las propiedades ACID de las RDBMS, que utiliza una arquitectura de no compartición.

Tiene version empresarial, así como versión dedicada a la comunidad, ésta última es software libre con licencia GNU. [5]

## 2.3.HBase

Es una base de datos distribuida y escalable de Hadoop. Se usa cuando se requiere acceso en tiempo real y aleatoriedad en el acceso a Big Data. Esta base de datos almacena los datos en grandes tablas que se almacenan en conjuntos de equipos sencillos.

Fue modelado después de Google´s Bigtable. Así como Bigtable aprovecha el almacenamiento de datos distribuidos proporcionados por Google File System (GFS), HBase proporciona las características de almacenar en grandes tablas a Hadoop. [6]

Características:

- Lineal y escalabilidad modular.
- Lectura y escritura consistente.
- Configurable y automática fragmentación de las tablas.

- Automática conmutación de error entre RegionServers.
- Uso sencillo de la API de Java para el acceso de clientes.
- Cache de bloques y filtros para queries en tiempo real.

## 2.4. Apache CouchDB

Es una base de datos que abarca completamente la web. Almacena sus datos con documentos JSON y su acceso se realiza a través de http. Indexa, combina y transforma sus documentos con Java Script. CouchDB funciona bien con la web moderna y aplicaciones móviles. Se puede incluso servir aplicaciones web directamente desde CouchDB. Y se puede distribuir sus datos o sus aplicaciones, de manera eficiente mediante la replicación incremental de los CouchDB. CouchDB soporta configuraciones maestro-maestro con la detección automática de conflictos. [7]

## 2.5.OrientDB

Es una base de grafos distribuida de segunda generación con flexibilidad de documentos. La primera generación no añadía las características demandadas por Big Data: multi-servidor, replicación, fragmentación y mayor flexibilidad para los complejos casos actuales.[8]

## 2.6. Redis

Redis es un motor de base de datos en memoria, basado en el almacenamiento en tablas de clave-valor, pero que opcionalmente puede ser usada como base de datos durable o persistente. Está liberado bajo licencia BSD por lo que es considerado software de código abierto.

Por ser un base de datos de en memoria su tamaño está limitado a la memoria RAM del equipo.[9]

## 2.7.Accumulo.

Apache Accumulo es un almacén estructurado altamente escalable basado en BigTable de Google. Accumulo está escrito en Java y funciona en el sistema de archivos distribuidos Hadoop (HDFS), que forma parte del popular proyecto Apache Hadoop. Accumulo admite el almacenamiento y recuperación de datos estructurados, incluyendo consultas de rangos eficiente, y proporciona soporte para el uso de tablas Accumulo como entrada y salida para los trabajos MapReduce.[10]

## 2.8.Hypertable

Hypertable es una base de datos altamente escalable de código libre basada en el diseño de Big Table de Google.[11].

## 2.9.Scalaris

Scalaris es una base de datos escalable de claves-valor. Fue la primera base de datos NoSQL que soportó las propiedades ACID[12].

## 2.10. Aerospike

Aerospike es una base de datos distribuida escalable NoSQL[13].

## 2.11. Comparativa y elección de sistema.

| | LENGUAJE | LICENCIA | PROTOCOLO | FORTALEZA | MEJOR UTILIZACION |
|---|---|---|---|---|---|
| **Redis** | C | BSD | telnet-like, binary safe | Velocidad | Para cuando cambien mucho los datos |
| **MongoDB** | C++ | AGPL | Custom, binary(BSON) | Cercano a SQL | Si necesitan búsquedas dinámicas. Si necesita gran rendimiento en base de datos grandes |
| **Cassandra** | JAVA | Apache | CQL3 Thrift | Almacena grandes conjuntos de datos casi en SQL | Cuando se necesita almacenar gran cantidad de datos en servidor pero se quiere una interfaz amigable |
| **CouchDB** | ERLANG | Apache | http/REST | Base de datos consistente y fácil de usar | Para la acumulación de datos, ocasionalmente su cambio y búsquedas predefinidas |
| **Accumulo** | JAVA and C++ | Apache | Thrift | Gran tabla con seguridad a nivel de celda | Cuando se necesite restringir el acceso a nivel de celda |
| **Hbase** | JAVA | Apache | http/REST | Billones de filas para millones de columnas | Base de datos de Hadoop |
| **Hypertable** | C++ | GPL 2 | C++ library,HQL shell | Más rápido y pequeño Hbase | Si necesitas mejorar Hbase |
| **OrientDB** | JAVA | Apache 2,0 | binary, http rest/json, java api | Base de datos tipo graph | Para estilo graph |

| | | | | | |
|---|---|---|---|---|---|
| **Scalaris** | ERLANG | Apache | Propietario y JSON-RPC | Almacén de claves/valores distribuidos bajo P2P | Si te gusta erlang pero necesitas que los datos sean accesibles desde otros lenguajes |
| **Aerospike** | C | AGPL | Propietario | Velocidad y soporte ssd | Aplicaciones que demanden baja latencia en lectura de datos, soporte de alta concurrencia y alta disponibilidad |
| **VoltDB** | JAVA | GPL 3 | Propietario | Transacciones rápidas y rápida evolución de los datos | Se necesita actuar rápido sobre cantidades masivas de datos de entrada |

[14]

El presente proyecto se centrará en el estudio de Hadoop por ser la primera y más importante implementación de Map&Reduce y Mongodb por sus características y por ser la base de datos NoSQL más utilizada. Grandes organizaciones usan estos dos sistemas

| | |
|---|---|
| Hadoop | Amazon, Adobe, AOL, Alibaba, Ebay, Facebook, Fox, ImageShack, Last.fm, [15] |
| Mongodb | Foursquare, Linkdln, Orange, Telefónica, Cisco, Bosch, Ebay, Expedia, Forbes, IBM, McAfee, TheGuardian, Le Figaro, The New York Times,..[16] |

# 3. Hadoop

Hadoop fue creado por Doug Cutting, mientras estaba trabajando en un buscador web open source llamado Apache Nucth. Este proyecto comenzó el año 2002 pero su arquitectura no podía soportar las búsquedas sobre billones de páginas web. Al año siguiente Google publicó información sobre la arquitectura de su sistema de ficheros distribuidos denominado GFS [17]. En el año 2004 publicaron la adaptación de GFS para Nucth que denominaron NDFS, este mismo año Google publicó una introducción a Map&Reduce [3] que rápidamente fue implementado en Nucth.

En 2006, NDFS y Map&Reduce implementados dentro del proyecto Nucth pasan a ser un proyecto independiente denominado Hadoop. Este nombre proviene del elefante de juguete de su hijo, por ello también el emblema de la aplicación es un elefante.

Hadoop es un framework de código abierto de almacenamiento de datos en clúster de hardware sencillo y de ejecución de aplicaciones sobre dichos datos. Implementa el paradigma computacional Map&Reduce donde la aplicación se divide en muchos pequeños fragmentos de trabajo, cada uno de los cuales se pueden ejecutar en cualquier nodo del clúster y cuantas veces se quiera. Además, proporciona un sistema de archivos distribuido que almacena los datos en los nodos de computación, que se denomina HDFS.

Hadoop dentro del ámbito de la empresa es visto de forma distinta por cada uno de los roles del personal. Para los ejecutivos es un proyecto de software de código abierto de Apache para obtener valor del volumen de datos acerca de su organización. Para los gerentes técnicos es una suite de código abierto de software que extrae los BigData estructurados y no estructurados acerca de su compañía. Para el Departamento legal es una suite de código abierto de software que cuenta con soporte de múltiples proveedores. Para la ingeniería es un entorno de ejecución masivamente paralelo, con resistencia a fallos, basado en Java Map&Reduce. Y para el departamento de seguridad es una suite de software con seguridad Kerberos.

Las principales características de Hadoop:

1. Escalable: Se pueden añadir nuevos nodos cuando sea necesario, sin tener que cambiar formatos de datos, cómo se cargan éstos, o las aplicaciones de las capas superiores.
2. Rentable: Hadoop trae la computación masivamente paralela para servidores básicos. El resultado es una disminución de los costes de almacenamiento.
3. Flexible. Permite cualquier dato independientemente del origen y formato del mismo.
4. Tolerante a los fallos. Cuando se pierde un nodo, el sistema redirige el trabajo a otro nodo donde estuvieran los mismo datos.

## 3.1. HDFS

El sistema de archivos distribuido Hadoop (HDFS) es un sistema de archivos diseñado para almacenar, acceder y ejecutarse en clústeres de ordenadores comunes. La página oficial del proyecto HDFS es http://hadoop.apache.org/hdfs

HDFS es altamente tolerante a fallos y está diseñado para ser implantado en equipos de bajo coste. Proporciona un alto rendimiento en el acceso a los datos y es adecuado para aplicaciones sobre grandes conjuntos de datos.

El error de hardware es la norma y no la excepción. El hecho de que hay un gran número de componentes y que cada componente tiene de modo individual probabilidad de fallar, implica que casi siempre algún componente del HDFS no sea funcional. Por lo tanto, la detección de fallos y la rápida recuperación automática de ellos es un objetivo central de la arquitectura de HDFS.

HDFS está diseñado más para el procesamiento de lotes de datos que para la utilización interactiva de los usuarios. El énfasis se encuentra en conseguir un alto rendimiento en el acceso de los datos en vez de una baja latencia de acceso a los mismos.

Las aplicaciones que se ejecutan en HDFS tienen grandes conjuntos de datos. Un archivo típico en HDFS es de un gigabyte e incluso de un terabyte. Se debe proporcionar un gran ancho de banda para la comunicación entre los cientos de nodos del clúster y dar soporte a la partición de estos grandes ficheros en muchos fragmentos.

HDFS ha sido diseñada para hacer fácil el paso de una plataforma a otra. Y como cualquier sistema de almacenamiento existe un tamaño mínimo de bloque, para HDFS este tamaño es de 64 megabytes.

HDFS tiene una arquitectura maestro/esclavo. Un clúster HDFS consta de un único NameNode y un conjunto de DataNodes. El NameNode actúa como equipo maestro. Mantiene el árbol del sistema de archivos, los metadatos de todos los ficheros y directorios del sistema de archivos. Esta información se almacena de forma persistente en un disco local del nodo en forma de dos ficheros: uno, con una imagen del namespace, y otro, en forma de un log. El DataNode almacena y recupera los bloques, y reporta periódicamente al NameNode una lista con los bloques que tiene almacenados.

El NameNode y DataNode son piezas de software diseñadas para ejecutarse en máquinas de hardware básico. HDFS está construido utilizando el lenguaje Java. El uso del lenguaje Java altamente portátil significa que HDFS se puede desplegar en una amplia gama de máquinas. Una implementación típica tiene un equipo dedicado únicamente al software NameNode, y cada una de las restantes máquinas que componen el clúster ejecuta un DataNode. Aunque la arquitectura no se opone a la ejecución de múltiples DataNodes en la misma máquina, rara vez se encuentra esta configuración en una implementación real.

HDFS es compatible con una organización jerárquica de archivos tradicional. La jerarquía es similar a la mayoría de los otros sistemas de archivos existentes, se puede crear y eliminar archivos, mover un archivo de un directorio a

otro, o cambiar el nombre del archivo, esto mismo lo podemos aplicar a los directorios.

## 3.2.Ecosistema de Hadoop

En Hadoop tenemos un ecosistema muy diverso, que crece día tras día, por lo que es difícil saber todos los proyectos que pueden interactuar con Hadoop de algún modo. A continuación se muestran algunos. [18]

### 3.2.1. Flume

Apache Flume es un sistema distribuido para capturar de forma eficiente, agregar y mover grandes cantidades de datos log de diferentes orígenes a un repositorio central, simplificando el proceso de recolectar estos datos para almacenarlos en Hadoop.

Página web oficial: http://flume.apache.org

### 3.2.2. Hive

Facilita la consulta y gestión de conjuntos de datos almacenados en sistemas distribuidos mediante un lenguaje parecido al SQL, llamado HiveQL.

Página web oficial: http://hive.apache.org

### 3.2.3. HBase

Como se cita anteriormente en el Capítulo 2 del presente Proyecto, HBase es una base de datos de Hadoop. Es el componente de Hadoop a usar cuando se requiere escrituras/lecturas en tiempo real y acceso aleatorio para grandes conjuntos de datos. Es una base de datos que no es relacional y por lo tanto no admite SQL.

Página web oficial: http://hbase.apache.org

### 3.2.4. Mahout

Es un proyecto para crear aprendizaje automático y data mining usando Hadoop. Se usa para análisis predictivos y otros análisis avanzados. Posee algoritmos de recomendación, clustering y clasificación.

Página web oficial: http://mahout.apache.org

### 3.2.5. Sqoop

Apache Sqoop (“Sql-to-Hadoop”) es una herramienta para transferir eficientemente datos entre Hadoop y sistemas de almacenamiento con datos estructurados, así como también con bases de datos relacionales. Algunas de sus características son: permite importar tablas individuales o bases de datos enteras a HDFS, genera clases Java que permiten interactuar con los datos importados y además, permite importar las bases de datos SQL a Hive.

Página web oficial: http://sqoop.apache.org

### 3.2.6. Ambari

Herramientas de administración para la instalación, monitorización y mantenimiento del clúster de Hadoop. También incluye las herramientas para para crear o eliminar nodos esclavos.

Página web oficial: http://ambari.apache.org/

### 3.2.7. Avro

Es un sistema de serialización de datos. Esta serialización puede ser en texto en plano, JSON, en formato binario. Con Avro podemos almacenar y leer datos fácilmente desde diferentes lenguajes de programación. Está diseñado para minimizar el espacio necesario para el almacenaje de los datos.

Página web oficial: http://avro.apache.org

### 3.2.8. Fuse

Permite que el sistema HDFS se pueda manejar como un sistema de archivos normal, dando acceso a comandos como ls, rm, cd y otros en HDFS.

Página web oficial: http://fuse.sourceforge.net/

### 3.2.9. HCatalog

Proporciona una visión relacional de los datos.

Página web oficial: http://hortonworks.com/hadoop/hcatalog/

### 3.2.10. Hue

Interfaz gráfica basada en navegador web para el análisis de los datos en Hadoop.

Página web oficial: http://gethue.com/

### 3.2.11. Oozie

Permite la planificación de los trabajos en Hadoop

Página web oficial: http://oozie.apache.org

### 3.2.12. Pig

Es una plataforma de análisis de grandes datos y programación de alto nivel de sentencias de Map&Reduce. Permite a los usuarios de Hadoop centrarse más en el análisis de los datos y menos en la creación de programas Map&Reduce.

Usa un lenguaje propio denominado PigLatin.

Página web oficial: http://pig.apache.org

### 3.2.13. ZooKeeper

Es un proyecto que proporciona una infraestructura centralizada y de servicios que permiten la sincronización del clúster.

Página web Oficial: http://zookeeper.apache.org

## 3.3.Cloudera

[19]

Cloudera Hadoop Apache (CDH) es la distribución oficial de Hadoop, que contiene el núcleo y los siguientes proyectos relacionados: Apache Avro , DataFu , Apache Flume , fusible-DFS , Apache HBase , Apache Hive , Hue , Apache Mahout , Apache MRv1 , Apache oozie , Apache Pig , Apache Spark , Apache Sqoop , Apache Runrunear , y Apache ZooKeeper. CDH es, de código abierto libre 100%, y está disponible bajo la licencia Apache 2.0.

Tiene dos ediciones disponibles, Cloudera Enterprise que es la versión completa bajo pago de licencia, y Cloudera Express que es la versión gratuita pero no completa. Ésta última versión en su instalación permite durante 60 días acceder a las características de la versión completa.

Cloudera Enterprise

- Basic Edition: ofrece una distribución para la empresa de Apache Hadoop en conjunto con el Administrador de Cloudera y otras herramientas de gestión avanzada y soporte técnico.
- Flex Edition: ofrece lo mismo que la edición básica pero además una extensión de soporte premium opcional para entornos críticos, y la elección de uno de los siguientes componentes avanzados: Cloudera Impala para consultas interactivas SQL, Cloudera Search para búsquedas interactivas; Cloudera Navigator para la gestión de datos, incluyendo la auditoría de datos, el linaje, y el descubrimiento, Apache Spark para análisis interactivos y procesamiento de flujos, o Apache HBase para almacenamiento en línea y aplicaciones NoSQL.
- Data Hub Edition ofrece lo mismo que la edición Flex, pero incluye todo los componentes avanzados.

Cloudera Express

Una descarga gratuita de CDH , que incluye una distribución lista para la empresa de Apache Hadoop , Apache HBase , Cloudera Impala , Cloudera Search , Apache Spark , y el Cloudera Manager , que ofrece capacidades de administración de clústeres robustos como el despliegue automatizado, administración centralizada, monitoreo y herramientas de diagnóstico. Cloudera Express permite a las empresas evaluar Apache Hadoop.

### 3.3.1. Utilización de Cloudera

[20]

En el presente proyecto para evaluar la utilización de Cloudera se usará un sistema virtualizado de Linux con Cloudera integrado, para lo cual se usa la máquina virtual Oracle VirtualBox, que se puede obtener de la siguiente página web:

https://www.virtualbox.org/wiki/Downloads

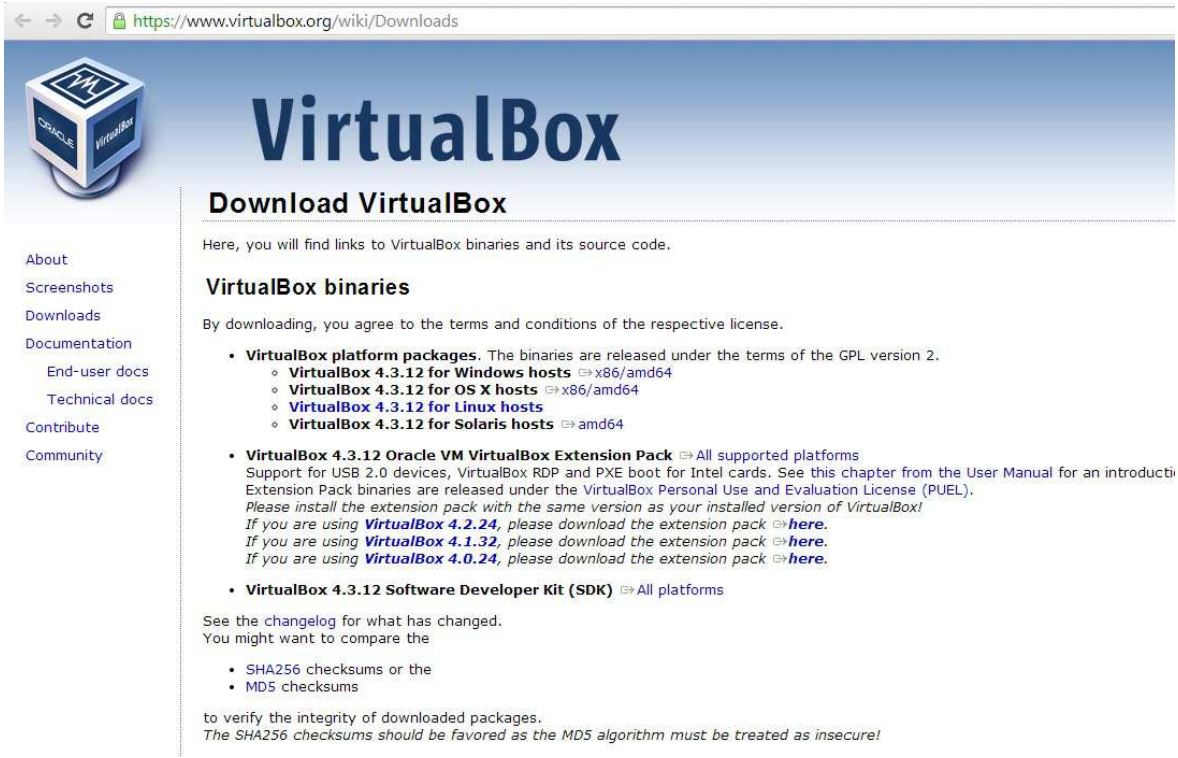


**Imagen Hadoop 1. Captura de pantalla.**

Una vez en esta página se elige según el sistema operativo donde se quiera instalar, se descarga e instala siguiendo los procedimientos de cualquier instalación en los respectivos sistemas operativos.

En el momento que se inicia por primera vez habrá que crear una nueva máquina virtual. Para ello seguimos los siguientes pasos:

- Crear nueva máquina virtual pulsando el botón new.


Imagen Hadoop 2. Captura de pantalla.

- Elegir el nombre, tipo y version. En type elegimos Linux y en version Ubuntu. Después le damos a continuar

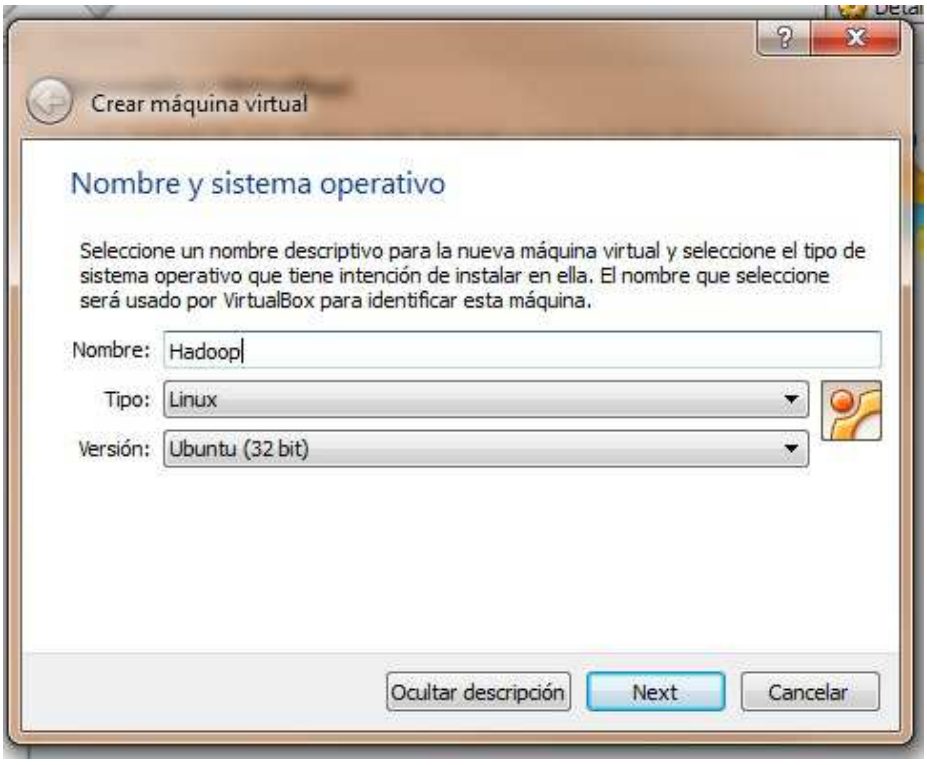


Imagen Hadoop 3. Captura de pantalla.

- Se selecciona el tamaño de memoria de la máquina virtual. Realizada esta operación se pulsa sobre next para continuar.

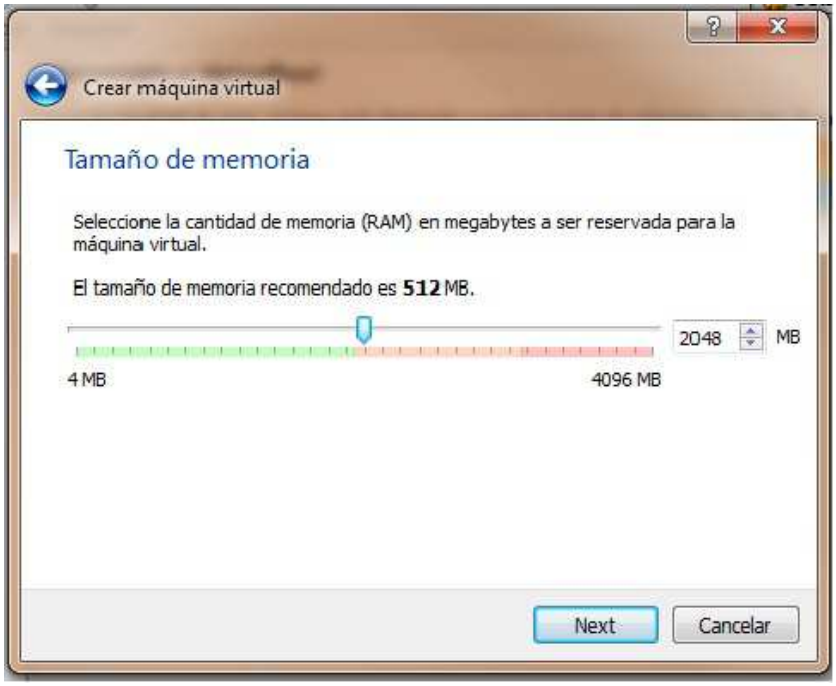


Imagen Hadoop 4. Captura de pantalla.

- Se selecciona "Use an existing virtual hard drive file", haciendo clic en el icono de la carpeta buscamos la imagen del disco virtual, esta imagen se obtiene de la siguiente página web:

  http://content.udacity-data.com/courses/ud617/Cloudera-Udacity-Training-VM-4.1.1.c.zip

  Este fichero imagen ocupa 1,7 Gigabytes antes de descomprimir y 4,2 Gigabytes después de descomprimir.

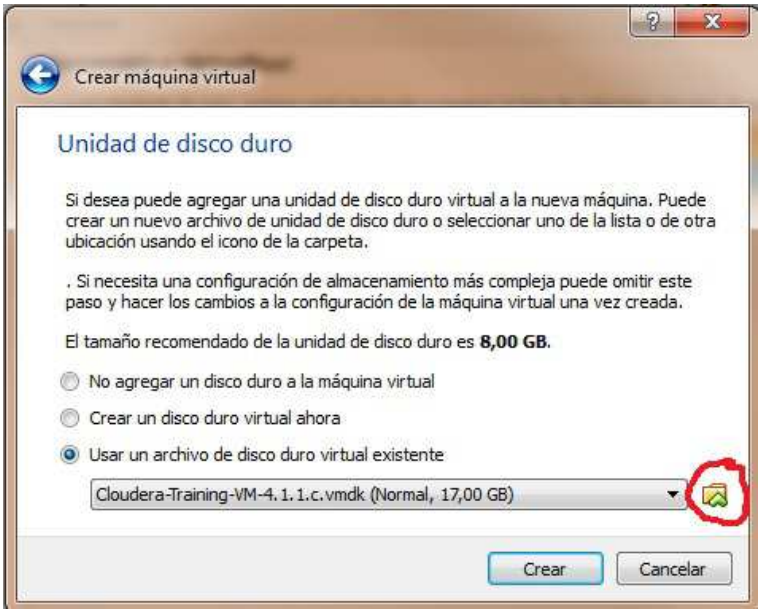


Imagen Hadoop 5. Captura de pantalla.

- Se inicia la máquina virtual.

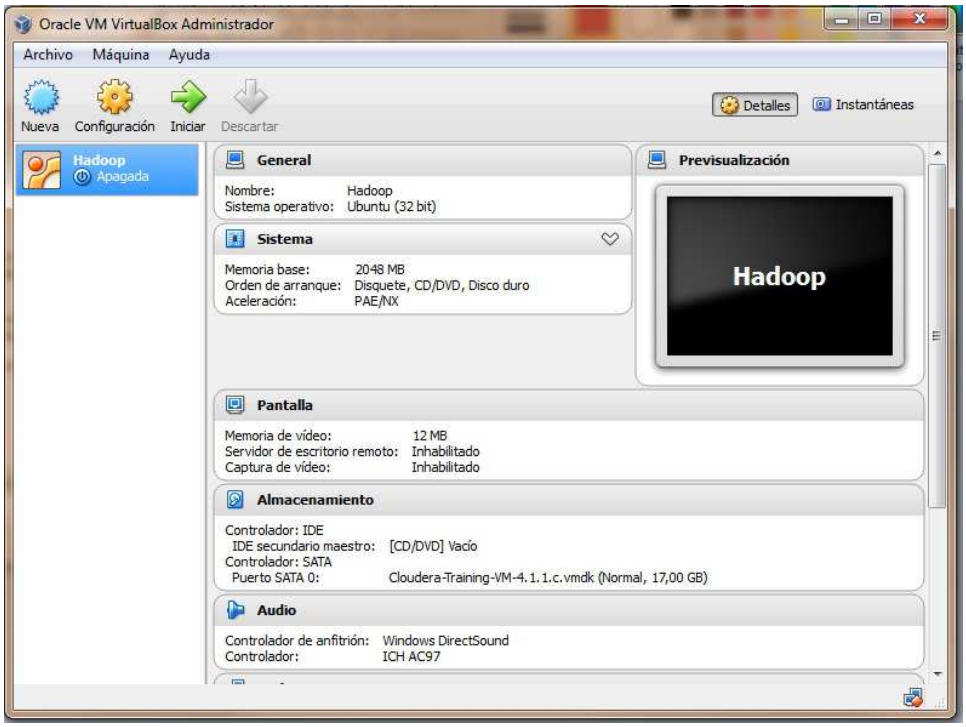


Imagen Hadoop 6. Captura de pantalla.

- Una vez arrancado tendremos un aspecto de escritorio en una ventana

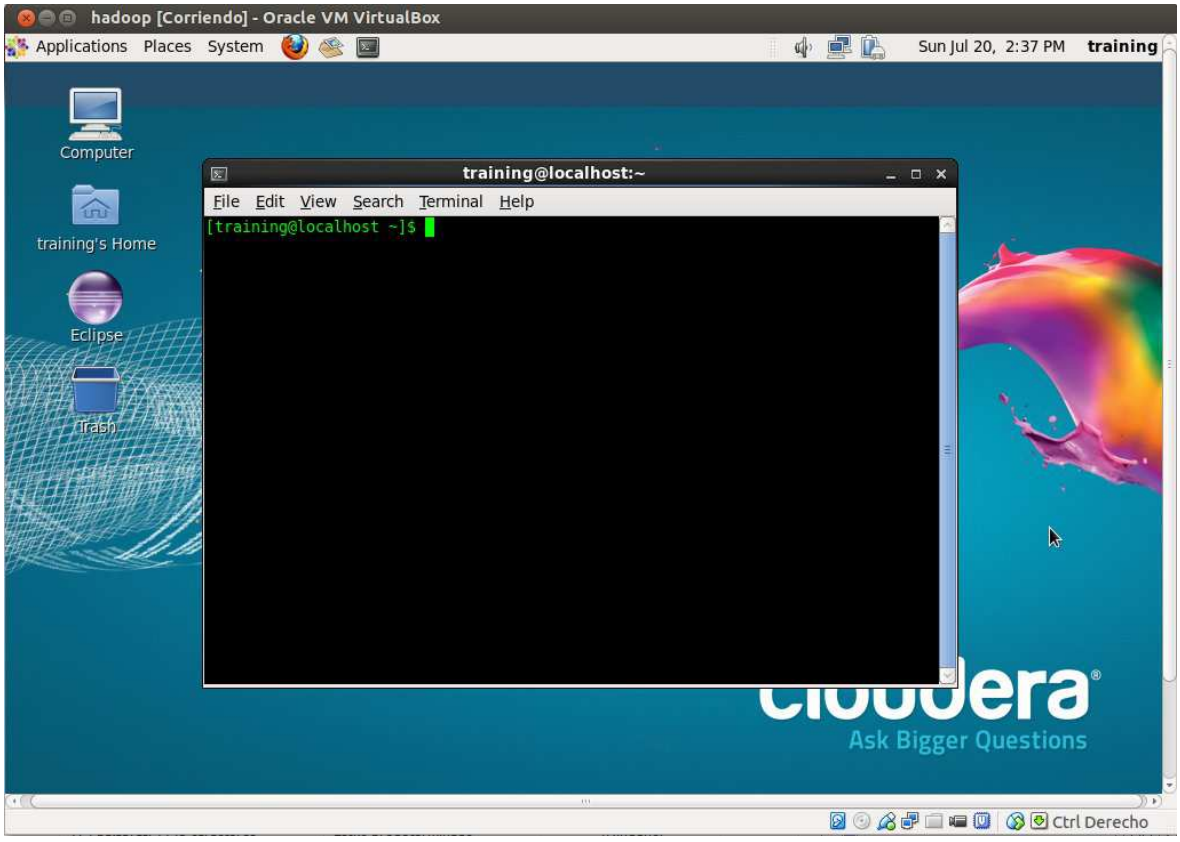


Imagen Hadoop 7. Captura de pantalla.

#### 3.3.1.1. Comandos Básicos

Desde la terminal vamos a trabajar con los siguientes comandos.

*hadoop fs -ls* comando con el que comprobamos los ficheros guardados en el HDFS

*hadoop fs -put fichero* comando para pasar el fichero a formato HDFS

*hadoop fs -tail fichero* comando para observar por pantalla el final del arhivo

hadoop fs -mv fichero fichero_nuevo instrucción con la se cambia el nombre al fichero.

*hadoop fs -rm fichero_nuevo* se elimina el fichero llamado fichero_nuevo.

*hadoop fs -mkdir carpeta* se crea el directorio carpeta dentro del HDFS.

*hadoop fs -put fichero carpeta* comando que introduce el fichero dentro del directorio denominado carpeta del sistema de archivos HDFS.

*hadoop jar /usr/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoop-streaming-2.6.0-mr1-cdh4.1.1.jar mapper fichero_map.py -reducer fichero_reduce.py -file fichero_map.py -file fichero_reduce.py -input directorio_entrada -output directorio_salida* instrucción en el que se realiza el Map&Reduce. Se debe indicar el directorio donde está java, los ficheros map y reduce, a continuación se indicamos el directorio donde está almacenado el fichero al que se aplica el mapreduce y por último el directorio de salida donde se almacenan los resultados en tres ficheros: success, log y part_0000. Este último fichero es realmente donde están almacenados los resultados del mapreduce.

*hadoop fs -cat directorio_salida/part_0000 |less* comando que muestra por pantalla el resultado de la función Map&Reduce.

*hadoop fs -get directorio_salida/part_0000 nombre_fichero* realiza una copia del fichero almacenado en HDFS en nuestro disco local.

## 3.4. Ejemplo básico.

Se considerarán los datos de ventas de un grupo de tiendas una misma empresa con el siguiente formato:

2012-01-01 12:01 San Jose Music 12.99 Visa

Son campos separados por tabulaciones correspondiendo cada uno por este orden a fecha, hora, nombre tienda, producto, precio y método de pago.

El objetivo de este ejemplo es obtener el número total de ventas de cada tienda. Para ello se pasan los datos por los siguientes procesos Map y Reduce, implementados en phyton.

```
def mapper()
      for line in sys.stdin:
            data=line.strip().split("\t")
            fecha,hora,tienda,objeto,precio,pago=data
            print"{0}\t{1}".format(tienda,precio)
```

En este mapper se realiza un bucle que recorre todo el fichero línea a línea, para obtener los datos separados por tabuladores que existentes en cada línea y le asigna las claves. La última línea de código devuelve por pantalla los valores intermedios. Antes de realizar el proceso reduce se ordenan según el nombre de la tienda. Quedando un nuevo fichero cuyas filas son cada una de las ventas realizadas en la empresa, y estas filas están formadas por dos columnas. La primera indica el nombre de la tienda y la segunda el precio esa venta.

| | |
|---|---|
| Miami | 12.34 |
| Miami | 99.07 |
| Miami | 3.14 |
| Miami | 4.95 |
| NYC | 99.77 |
| NYC | 88.99 |
| NYC | 99.99 |
| … | … |

Ahora se ejecuta el proceso reduce.

```
def reducer():
      ventasTotal=0
      oldKey=None
      for line in systdin
            data=line.strip().split("\t")
            if len(data)!=2;
                  continue
            thisKey,thisSale=data
            if oldKey and oldKey!=thisKey
                  print "{0}\t{1}".format(oldKey,ventasTotal)
                  ventasTotal=0
            oldKey=thisKey
            ventasTotal+=float(thisSales)
```

Después de la ejecución del proceso reducer se obtiene el resultado final consistente en pares clave-valor. Donde, la clave será el nombre de la tienda, y el valor es la suma de los precios de todas sus ventas.

| | |
|---|---|
| Miami | 119.5 |
| NYC | 288.75 |
| … | … |

# 4. Mongodb

MongoDB es una poderosa, flexible y escalable base de datos orientada a documentos.

Una base de datos orientada a documentos reemplaza el concepto de fila por un modelo más flexible, el documento. Hace posible representar relaciones jerárquicas complejas en un simple apunte. Estos documentos son almacenados en BSON, que es una representación binaria de JSON.

Ejemplo de base de datos relacional.

| Nombre_alumno | Apellidos_Alumno | Asignatura | Nota |
|---|---|---|---|
| alumno1 | Apellido1 apellido2 | Asignatura1 | 5 |
| Alumno1 | Apellido1 apellido2 | Asignatura2 | 6 |
| Alumno2 | Apellido3 apellido4 | Asignatura1 | 5 |

Estos mismos datos almacenados como datos

```
{
        “Nombre_Alumno”:”alumno1”,
        “Apellidos_Alumno”:”Apellido 1 apellido 2”,
        “Asignatura”:[
                {
                        “nombre”:”Asignatura1”,
                        “nota”:5
                },
                {
                        “nombre”:”Asignatura2”,
                        “nota”:6
                }
        ]
}
```

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema, provocando que la adición o eliminación de campos sea sencilla.

MongoDB está escrito en C++, aunque las consultas se hacen pasando objetos JSON como parámetro. Es algo bastante lógico, dado que los propios documentos se almacenan en BSON. Por ejemplo:

db.Clientes.find({"Nombre":"Pedro"}); Buscará todos los clientes cuyo nombre sea Pedro.

MongoDB viene de serie con una consola desde la que podemos ejecutar los distintos comandos. Esta consola está construida sobre JavaScript, por lo que las consultas se realizan utilizando ese lenguaje. Además de las funciones de MongoDB, podemos utilizar muchas de las funciones propias de JavaScript. En la consola también podemos definir variables, funciones o utilizar bucles.

Si queremos usar nuestro lenguaje de programación favorito, existen drivers para un gran número de ellos. Hay drivers oficiales para C#, Java, Node.js, PHP, Python, Ruby, C, C++, Perl o Scala.

## 4.1.Herramientas

- Mongo

  Shell de MongoDB. Provee una consola para actuar sobre las bases de datos del sistema MongoDB.[21]

- Mongod

  Proceso primario de un sistema MongoDB. Maneja las solicitudes de datos, formato de los datos y realiza las operaciones de gestión.[22]

General options

| | | |
|---|---|---|
| -h | --help | Muestra información de ayuda |
| --version | | Muerta información de la version de mongo |
| -f arg | --config arg | Se especifica un fichero de configuración |
| -v arg | --verbose arg | Aumenta el nivel de detalle mostrado en la shell durante el proceso de conexión |
| --quiet | | Desactiva la información a mostrar en la Shell durante el proceso de conexión |
| --port arg | | Indicamos el puerto de escucha del servidor. Por defecto 27017 |
| --bind_ip args | | Direcciones ip separadas por comas por donde escucha el servidor. Por defecto escucha por todas las ip de la máquina local |
| --maxConns arg | | Número máximo de conexiones simultaneas. Por defecto 1000000 |
| --logpath arg | | fichero log donde se almacena la salida de stdout |
| --httpinterface | | Habilita interfaz http |
| --clusterAuthMode arg | | Tipo de autentificación en el clúster. Opciones disponibles: keyFile, sendKeyFile, sendX509, x509. |
| --auth | | Arranca con seguridad |
| --noauth | | Arranca sin seguridad |
| --username arg | -u arg | Usuario para autentificación |
| --password arg | -p arg | Contraseña para autentificación |
| --ipv6 | | Soporta ip version 6. Por defecto no la soporta |
| --cpu | | Muestra periódicamente el uso de la CPU |
| --sysinfo | | Muestra un diagnóstico del sistema |
| --dbpath arg | | Directorio de la bases de datos. Por defecto /data/db/ |
| --nssize arg (=16) | | Tamaño del fichero .ns para nuevas bases de datos |
| --quota | | Límite de un cierto número ficheros por base de datos. Por defecto 8 |
| --quotaFiles arg | | Número permitido de ficheros por base de datos |
| --smallfiles | | Usa tamaño pequeño de ficheros |
| --syncdelay arg (=60) | | Segundos entre sincronía de disco (indicando 0 no se sincroniza, no es recomendable) |

| | | |
|---|---|---|
| --upgrade | | Actualiza la base de datos si es necesario |
| --repair | | Ejecuta reparación en todas las bases de datos |
| --repairpath arg | | Directorio para la reparación de ficheros. Por defecto es el indicado con dbpath |
| --noscripting | | Deshabilita los script |
| --shutdown | | Para un servidor activo |

Master/slave options (old; use replica sets instead):

| | |
|---|---|
| --replSet arg | Nombre del sistema de replica |
| --oplogSize arg | Tamaño, en megabytes, para el fichero de operaciones. El tamaño por defecto es el 5% del disco |

Sharding options:

| | |
|---|---|
| --configsvr | Declara el mongod como servidor de configuración del clúster, puerto por defecto 27019; directorio por defecto /data/configdb |
| --shardsvr | Declara el mongod como shard del clúster, puerto por defecto 27018 |

- Mongos

Servicio de enrutamiento cuando fragmentamos la base de datos.[23]

| OPCION | OPCION | DESCRIPCION |
|---|---|---|
| -h | --help | Muestra la ayuda de mongos |
| --version | | Muestra la version |
| -f arg | --config arg | Configuración indica en fichero pasado como argumento |
| -v | --verbose | Aumenta el nivel de detalle mostrado en la shell durante el proceso de conexión |
| --quiet | | Desactiva la información a mostrar en la Shell durante el proceso de conexión |
| --port arg | | Puerto por el recibe las conexiones |
| --bind_ip arg | | Dirección receptora de conexiones |
| --maxConns arg | | Máximo número de conexiones simultaneas. Valor por defecto 100000 |
| --logpath arg | | Fichero log donde se almacenan los resultados mostrados por pantalla. |
| --keyFile arg | | Clave privada para la identificación del clúster |
| --setParameter arg | | Modifica el parámetro de configuración |
| --httpinterface | | Habilita la conexión por interfaz http. |
| --clusterAuthMode arg | | Modo de autenticación para el clúster. Las posibilidades son (keyFile\|sendKeyFile\|sendX509\|x509) |
| --nounixsocket | | Desactiva la escucha de socket Unix |
| – fork | | El proceso servidor es independiente de la consola en el que se arranca. |

Sharding options:

| | |
|---|---|
| --configdb arg | 1 o tres configuradores de servicios separados por comas |
| --localThreshold arg | Tiempo de ping en milisegundos para que un nodo se considere local, por defecto 15ms |
| --test | Solo arranca la unidad de test |
| --upgrade | Actualiza los metadatos |
| --chunkSize arg | Tamaño máximo de datos por chunk |
| --jsonp | Permite JSOMP acceso vía http |
| --noscripting | Deshabilita script |

- Mongodump

  Crea un fichero binario de los datos de la base de datos. Se considera una parte fundamental para la estrategia de copia de seguridad junto a la herramienta mongorestore se usa para la restauración de las bases de datos.[24]

  Copiar de la instancia mongod la colección de la base de datos localizada en el equipo indicado por direccion_IP y el puerto por num_puerto

  ```
  mongodump --host dirección_IP --port num_puerto --db base_datos --collection  colección
  ```

  También podemos crear estas copias de seguridad sin que la instancia mongod esté en uso

  ```
  mongodump --dbpath directorio_base_datos
  ```

- Mongorestore

  Graba los datos del archivo binario creado por mongodump en la base de datos. Se puede crear una base de datos nueva o grabar los datos en una existente.[25]

  ```
  mongorestore --collection colección --db base_datos dump/base_datos/collecion.bson
  ```

- Bsondump

  Convierte los documentos BSON en formatos leíbles por personas, incluido el JSON. Es muy útil para la lectura de ficheros binarios creados por mongodump.[26]

  `bsondump fichero.bson` devuelve fichero.json
  `bsondump --type fichero.bson` devuelve una depuración del fichero fichero.bson

- Mongooplog

  Es una sencilla herramienta que sondea las operaciones almacenadas en el fichero oplog servidor de un sistema de replicación, y las aplica al servidor local. Esta capacidad es compatible con ciertas clases de migraciones en tiempo real que requieren que el servidor de origen se mantenga en línea y en funcionamiento en todo el proceso de migración.[27]

  ```
  mongooplog --from equipo1 --host equipo2. Copia el fichero oplog del equipo1 en el equip2.
  ```

- Mongoexport

  Herramienta para la creación de ficheros JSON o CSV a partir de los datos almacenados n la base de datos.[28]

  ```
  mongoexport --host equipo1 --port num_puerto -d base_datos -c coleccion1 -csv –out fichero
  ```

  Exporta la coleccion1 de la base de datos almacenada en el host equipo1 y puerto num_puerto a formato csv a un fichero. Si no se indica el comando -csv el fichero resultante será de tipo json, si no indica el fichero de salida se toma la estándar.

- Mongoimport

  Al contrario que mongoexport transforma ficheros JSON o CSV a ficheros de la base de datos Mongodb.[29]

  ```
  mongoimport --host equipo1 --port puerto –d base_datos –c coleccion1 --type csv --file fichero.csv --headerline --stopOnError --drop
  ```

  Importa el fichero tipo csv tomando como claves de los valores la primera lía del fichero en la coleccion1 de la base de datos base_datos almacenada en la instancia mongodb del equipo y puerto indicados. Si tenemos el comando --drop se borra todos los documentos anteriormente almacenados y el comando stopOnError interrumpe la importación si ocurre algo error.

- Mongostat

  Proporciona una vista del estado de mongod o mongos que están ejecutándose.[30]

- Mongotop

  Método para contabilizar el tiempo de lectura y escritura en el sistema MongoDB[31]

- Mongosniff

  Proporciona operaciones de bajo nivel sobre la actividad en tiempo real de la base de datos.[32]

- Mongoperf

  Utilidad para la comprobación de entrada y salida del discos independientemente del sistema MongoDB.[33]

- Mongofiles

  Utilidad para la manipulación de ficheros almacenados en tu base de datos MongoDB en GridFS.

  Se tiene que tener en cuenta que en sistemas de replicación sólo tendrá acceso a los datos de las instancias primarias.

  Tanto mongofiles como mongodump, mongoexport, mongoimport y mongorestore tienen acceso a los datos almacenados en MongoDb sin que se esté ejecutando ninguna instancia mongod. [34]

## 4.2.Instalación

Primero debemos ir a la página oficial de MongoDB: www.mongodb.org

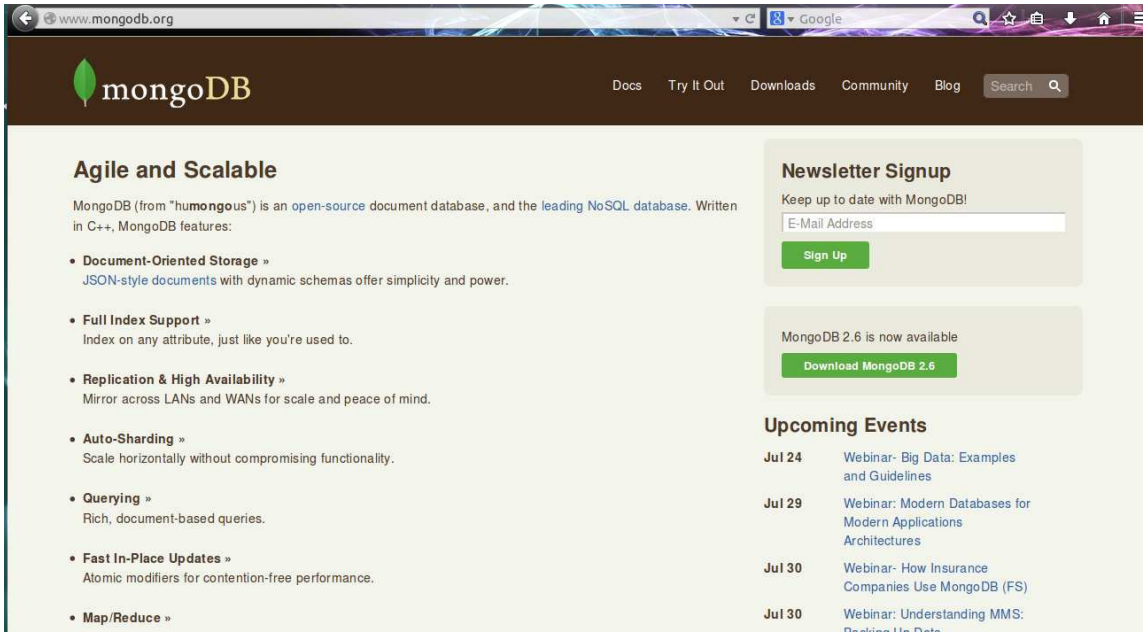


**Imagen mongodb 1. Captura de pantalla**

Entramos en la sección Downloads. Y elegimos según el sistema operativo de nuestro equipo.

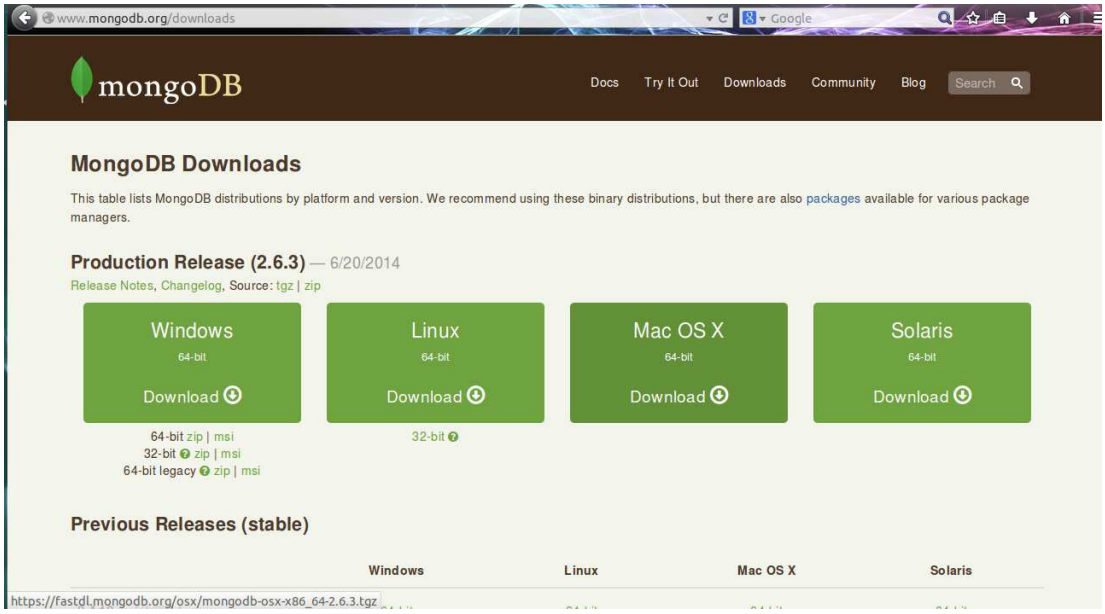


Imagen mongodb 2. Captura de pantalla

Cuando elegimos la versión que necesitamos se descarga un archivo comprimido y nos presenta la siguiente página web donde tenemos la opción de introducir nuestra dirección e-mail para mantenernos actualizados respecto la información sobre MongoDB.

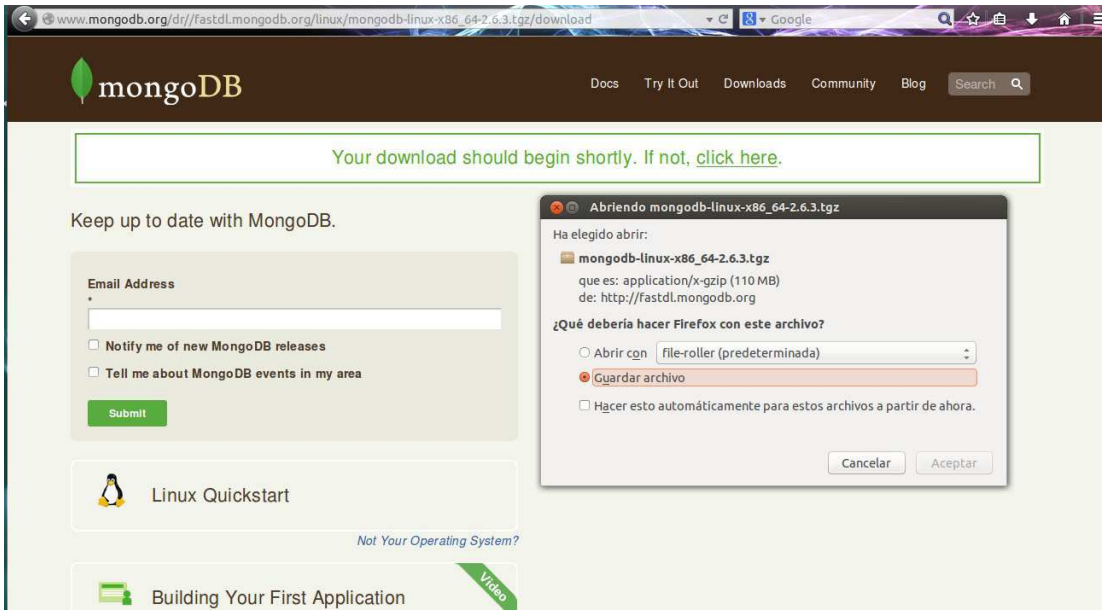


Imagen mongodb 3. Captura de pantalla

Para instalar únicamente se descomprime el fichero descargado.

Antes de iniciar se debe crear el directorio en el equipo local donde estará almacenada la base de datos. Por defecto cuando se inicie una instancia de mongodb, ésta buscará la base de datos en el directorio /data/db almacenado en el directorio raíz del equipo. Si no se desea almacenar la base de datos en el directorio por defecto se puede guardar en otro directorio pero en el inicio de la instancia se debe indicar el directorio elegido.

Para iniciar la instancia mongod, que es el servidor de la base de datos, dependerá del sistema operativo en el que ejecutemos el mongod. A continuación se explica su realización en Windows y en Linux.

Como se cita anteriormente se debe crear los directorios donde se va almacenar la base de datos. En el supuesto de querer usar el directorio por defecto en Linux éste será /data/db. Accedemos a la consola de Linux y se sitúa dentro del directorio que contiene los archivos descomprimidos del paquete de mongodb, y se ejecuta el proceso mongod.

```
./mongod --dbpath directorio_base_datos --port num_puerto
```

Inicia el servidor en el puerto indicado por num_puerto y almacena las bases de datos en el directorio indicado por directorio_base_datos. Si no se indica el parámetro dbpath se almacenan las bases de datos en el directorio por defecto y si no se indica el parámetro port entonces el servidor recibe las peticiones por el puerto por defecto, 27017, que es en todos los sistemas operativos el mismo.

En el caso de ejecución en sistemas operativos Windows el directorio por defecto es c:\data\db. Se ejecutará mongod.exe desde el intérprete de comandos cmd teniendo los mismos parámetros que en los sistemas Linux. Mongodb se puede instalar como un servicio de Windows mediante el siguiente comando:

```
dir_contenedor_mongod/mongod.exe --dbpath directorio_base_datos --
install
```

Con estos sencillos pasos se tiene un servidor mongodb en ejecución, ahora se procede a explicar cómo establecer conexión con el mismo para la manipulación de la base de datos, que se realiza por medio de la herramienta mongo incluida en el paquete descomprimido de mongodb.

- Linux.

```
./mongo --host ip_servidor --port num_puerto
```

- Windows.

```
mongo.exe --host ip_servidor  --port num_puerto
```

Se conecta al servidor mongod, que se encuentra a la escucha de peticiones en el equipo y puerto indicados. Si no se indica el host por defecto es local host (127.0.0.1) y el puerto como se indica anteriormente el 27017. Si se ejecuta dos instancias mongod en el mismo equipo nunca podrán compartir el directorio de almacenaje de bases de datos.

## 4.3.Operaciones básicas

En este apartado se estudia diferentes operaciones básicas de manipulación de los datos almacenados en la base de datos mongodb.

### 4.3.1. db

Orden que muestra el nombre de la base de datos en que se está trabajando.

### 4.3.2. Use

Orden para elegir la base de datos sobre la que se desea actuar. En caso de la no existencia previa se crea automáticamente y se pasa a trabajar sobre ella.

### 4.3.3. show dbs

Muestra una lista con los nombres de las distintas bases de datos almacenadas en el servidor mongod.

### 4.3.4. Creación de registros

Las operaciones son como funciones de JavaScript, así que se llamará al objeto base de datos db y creamos una nueva propiedad llamada alumno y le asociamos su valor correspondiente.

```
db.alumnos.insert({“nombre”:”alumno1”,”nota”:5})
```

Incluso es posible declarar el documento como un objeto, almacenarlo en una variable y posteriormente insertarlo del siguiente modo:

```
var estudiante={“nombre”:”alumno2”,”nota”:6”}

db.alumnos.insert(estudiante)
```

### 4.3.5. Búsqueda de registros. find()

Para realizar una búsqueda de todos los documentos de una colección se realiza con el siguiente comando.

```
db.coleccion.find()
```

Por el contrario si se quiere sólo el primer elemento se usa findOne en vez de find. También se realizan búsquedas filtradas según un único parámetro que se tenga que cumplir.

```
db.coleccion.find({parámetro:valor})
```

Varios parámetros que se cumplan a la vez

```
db.coleccion.find({parametro1:valor1,parametro2:valor2})
```

Que se cumpla al menos uno de los parámetros indicado

```
db.coleccion.find({$or[parametro1:valor1},{parametro2:valor2}]})
```

Si se quiere limitar el número de resultados a un número máximo únicamente hay que agregar *`.limit(num_max)`* al final del comando *`find()`*.

Y en el caso de querer ordenar los resultados por un campo en particular añadimos *`sort({"campo":1)`*

### 4.3.6. Eliminación de registros

Existen tres posibilidades para eliminación de registros: Eliminar los documentos de una colección que cumplan una condición, eliminar todos los documentos y la eliminación de la colección.

Para la eliminación de documentos que cumplan una condición se usa el siguiente comando: *`db.coleccion1.remove({"elemento1":"valor1"})`*. Elimina todos los documentos de la coleccion1 que posean el elemento1 con el valor1.

Para la eliminación de todos los documentos que componen una colección se realiza con la siguiente orden *`db.coleccion1.remove()`*

Y para eliminar la colección entera se usa *`db.coleccion1.drop()`*

## 4.4.Replicación

El principal propósito de la implementación de estrategias de replicación de datos es incrementar la redundancia de los mismos, esto permite tener una base de datos de alta disponibilidad e integridad. Una base de datos al tener varias copias exactas en diferentes infraestructuras separadas asegura que si falla, ya sea a nivel de hardware o situaciones diversas que pudiesen corromper o evitar el acceso a los datos, el sistema de dicha base de

datos no se vea afectado ya que existen otras copias que tomarán el lugar de la original mediante un proceso transparente para los usuarios finales.

En MongoDB, al grupo de instancias de mongod que poseen la misma información se les denomina replica set o grupo de replicación.

Un replica set en MongoDB está compuesto por dos tipos de miembros: instancias primarias y secundarias, teniendo una única instancia primaria la cual aceptará todas las operaciones de escritura/lectura provenientes de los sistemas cliente, aunque es posible configurar la lectura a instancias secundarias.

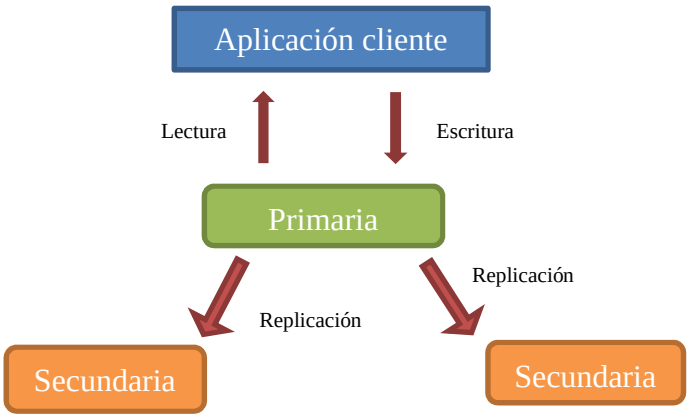


Imagen mongodb 4

Estas operaciones que alteran los datos de la bases de datos se almacenan en un archivo llamado oplog, los miembros secundarios replican este archivo del miembro primario y ejecutan las mismas operaciones sobre su conjunto de datos.

El permitir la lectura en las instancias secundarias tiene como objetivo liberar de carga a la instancia primaria, pero el procedimiento de replicación se realiza de manera asíncrona, por lo que es posible que los clientes que consulten directamente a miembros secundarios no obtengan la información actualizada.

Existen cuatro configuraciones especiales para los miembros secundarios.

- Miembro Prioridad 0

  Esto evitará que este miembro pueda ser convertido en miembro primario en caso de fallo de este.

- Miembro Escondido

  Son miembros secundarios de prioridad 0 pero además no son accesibles para lectura por parte de los clientes.

- Miembro retrasado

  También son miembros secundarios de prioridad 0 y poseen la particularidad de mantener un estado retrasado de la base de datos, se usa como unidades de respaldo. Debido a su retraso deben ser también miembros escondidos.

- Miembro árbitro.

  Son miembros secundarios que no almacenan copia de la base de datos, y por lo tanto nunca podrá ser elegido como primaria. Son instancias que votan en las elecciones a primaria.

En caso de fallo de la instancia primaria se realiza de forma transparente al usuario una votación entre las instancias secundarias sin prioridad cero, para facilitar esta elección se suelen crear sistemas de réplicas con un número impar de instancias secundarias o que una de las secundarias sea árbitro. Un sistema de réplica puede tener como máximo doce miembros, y de los cuáles con derecho a voto sólo siete.

Métodos de replicación en el mongo Shell

| | |
|---|---|
| rs.add() | Añade un miembro al sistema replica |
| rs.addArb() | Añade un miembro arbitro al sistema replica |
| rs.conf() | Devuelve el fichero de configuración del sistema de replica |
| rs.freeze() | Provoca que el miembro secundario no pueda pasar a primario durante un tiempo. |
| rs.help() | Muestra una ayuda básica de las funciones replica set |
| rs.initiate() | Inicia una replica set |
| rs.printReplicationInfo() | Imprime informe del sistema replica desde el punto de vista de los servidores primarios. |
| rs.printSlaveReplicationInfo() | Imprime informe del sistema replica desde el punto de vista de los servidores secundarios. |
| rs.reconfig() | Reconfigura la replica set |
| rs.remove() | Elimina un miembro del sistema de replica |
| rs.slaveOK() | Asigna a la instancia indicada que sea secundaria. |
| rs.status() | Retorna un documento con información del estado del sistema de réplica. |
| rs.stepDown() | Convierte al miembro primario en secundario, obligando a unas elecciones. |
| rs.SyncFrom() | Asigna al miembro indicado como el que los demás miembros del sistema replica se conectarán para la sincronización |

### 4.4.1. Replicación en el mismo equipo

Primero se crean los directorios necesarios, ya que mongodb no permite la compartición de directorios entre varios procesos mongod. Para continuar se arrancan los mongod indicando para cada uno de ellos su directorio y puerto, además de añadir el nombre del sistema replica.

```
mongod --dbpath directorio1 --port puerto1 --replSet “rs1”

mongod --dbpath directorio2 --port puerto2 --replSet “rs1”

mongod --dbpath directorio3 --port puerto3 --replSet “rs1”

mongod --dbpath directorio4 --port puerto4 --replSet “rs1”
```

Una vez se han creado todos los procesos mongod se conecta por medio de la shell de mongo al proceso mongod que se quiera usar como instancia primaria del sistema de réplica. Y se inicia el sistema de réplica con el comando rs.initiate().

Ahora sólo queda añadir a cada una de las instancias mongod que conforman el sistema replica. Para nuestro ejemplo se supone que la instancia primaria es la que tiene como puerto de entrada el puerto1.

```
rs.add(localhost:puerto2)

rs.add(localhost:puerto3)

rs.add(localhost:puerto4)
```

Para comprobar si está todo correcto se comprueba por medio del comando rs.status(), que devuelve información sobre los nodos que forman el sistema de réplica.

### 4.4.2. Replicación en distintos equipos

Para la realización de un sistema de réplica distribuido por diferentes máquinas se siguen los siguientes pasos:

- Creación de los diferentes mongod indicando que conforman un sistema de réplica.

- Desde el mongo shell conectado al mongod que se quiere que actúe como servidor primario se crea el documento de configuración del sistema de replicación.

```
rsconf={
        _id:”nombre sistema replica”,
        members:[
                {_id:0
                host:”dirección_ip:puerto”}
        ]
}
```

- Iniciamos el sistema de replicación con el documento de configuración del anterior punto

```
rs.initiate(rsconf)
```

- Añadimos los miembros secundarios

```
rs.add("dirección_ip:puerto")
```

- Se comprueba el sistema de replicación

```
rs.status()
```

### 4.4.3. Administración de sistemas de replicación

Una gran cantidad de tareas de mantenimiento no se pueden realizar en instancias secundarias, por implicar escritura, y tampoco se deben realizar en las primarias. Por lo que se realiza es el reinicio de la instancia primaria para que sea un ser servidor independiente, no forme parte del sistema replica.

Para realizar el mantenimiento se reinicia sin la opción replSet, como se quiere que exista comunicación entre este servidor y los demás equipos que forman el sistema de replica se cambia el puerto elegido, pero mantenemos el mismo directorio.

#### *4.4.3.1. Configuración del sistema de réplica*

La configuración de un sistema de réplica siempre está almacenada en un documento de la colección local.system.replset.

Este documento es el mismo para todos los miembros del sistema de replicación. Nunca se actualiza este documento por medio de la función update, siempre se usará el comando específico para esta acción `replSetReconfig`.

##### 4.4.3.1.1. Cambios de los miembros

Se puede añadir nuevos miembros al sistema de replicación. Estos nuevos miembros pueden tener los directorios vacíos y al conectarse se sincronizan, o previamente contener una copia de los datos de algún miembro del sistema.

Al igual que se permite la adición de miembros se pude eliminar miembros del sistema de replicación.

Cuando se cambian las configuraciones de los miembros para una reconfiguración se tienen ciertas restricciones:

- No cambiar la identifacación.
- No se puede hacer la reconfiguración desde un miembro con prioridad 0, generalmente se realiza desde el miembro primario.
- No se puede convertir un miembro arbitro en no arbitro y el viceversa tampoco

- No se puede cambiar un miembro con el elemento "buildIndexes" : false a otro miembro con el elemento tenga valor true.

La estrategia que se sigue para cambiar cualquier elemento es buscar el elemento con rs.config() modificar las partes deseadas y reconfigurar enviando esta nueva configuración con rs.reconfig()

#### 4.4.3.1.2. Ampliación del sistema de réplica

Los conjuntos de réplicas están limitados a 12 miembros, y sólo 7 miembros de ellos con derecho de voto. Esto es para reducir la cantidad de tráfico de red necesario para los ping entre miembros para la comprobación estado y para limitar la cantidad de tiempo que duran las elecciones.

#### 4.4.3.1.3. Forzar reconfiguración

Cuando se pierde permanentemente la mayoría de los miembros del sistema, es posible que desee volver a configurar el conjunto, por lo general usted debe enviar el reconfig a la primaria. Si no tiene una primaria en este caso, puede forzar reconfigurar del conjunto mediante el envío de un comando reconfig a un secundario y pasarle un reconfig con la opción de forzar: `rs.reconfig(config, {"force" : true})`

### *4.4.3.2. Manipulación de los estados de los miembros de un sistema de réplica*

Hay varias maneras de cambiar manualmente el estado de un miembro para el mantenimiento o en respuesta de la carga. Tenga en cuenta que no hay forma de obligar a un miembro a convertirse primario que no sea configurando el valor apropiado.

#### 4.4.3.2.1. Cambio de primario a secundario

Puede cambiar un miembro primario para que sea un secundario utilizando la función: `rs.stepDown ()`

Esto hace que el primario pase a estado secundario durante 60 segundos. Si durante este periodo no ha sido elegido otro miembro para ser primario, podrá intentar una reelección. Si quisiera seguir siendo un secundario por un tiempo distinto, ya sea más largo o más corto, se puede especificar esta cantidad en segundos. Ejemplo: `rs.stepDown (600)`, la instancia quedará como secundaría un mínimo de 10 minutos.

#### 4.4.3.2.2. Prevención de elecciones

Si necesita hacer mantenimiento en un miembro primario, pero no desea que cualquiera de los otros miembros elegibles se convierta en primarios, se puede forzar que permanezcan como secundarios mediante la ejecución del método freeze en cada uno de ellos. Cuando haya terminado el mantenimiento solo tiene que usar el mismo método indicando cero segundos.

### *4.4.3.3. Monitorear el sistema de replicación*

Es importante ser capaz de controlar el estado de un conjunto: no sólo que todo los miembros han terminado, sino los estados que se encuentran en cada momento y cómo va la replicación. Hay varios comandos que puede utilizar para ver la información del conjunto de réplicas.

A menudo, los problemas con la replicación son transitorios. La forma más fácil de ver los problemas es buscar en los registros. Asegurarse de saber dónde se almacenan los registros y que puedan acceder a ellos.

#### 4.4.3.3.1. Obtención del estado

Uno de los comandos más útiles es `rs.status()`, que devuelve la información corriente de cada miembro del sistema de replicación.

Los campos más importantes que muestra este comando son:

- self
  Este campo sólo está presente en el miembro desde que se hace la llamada a rs.status()
- stateStr
  Un Sting que describe el estado del servidor
  - STARTUP
    Estado para el inicio por primera vez de un miembro
  - STARTUP2
    Estado durante la sincronización
  - RECOVERING
    Indica que el miembro es operativo pero no está habilitado para la lectura
  - ARBITER
    Indica que el miembro es un árbitro
  - DOWN
    Indica que ha sido miembro del sistema pero ahora no es alcanzable por los demás miembros

- UNKNOWN
  Cuando el miembro no ha sido capaz de comunicar con el resto de miembros
- REMOVED
  Indica que ha sido eliminado del sistema.

- ROLLBACK
  Cuando es un miembro retrasado en la actualización de los datos.
- FATAL
  Cuando algún error no se ha podido corregir y el usuario ha renunciado a tratar de funcionar correctamente

- uptime
  Número de segundo que el miembro ha sido accesible
- optimeDate
  El tiempo de la última operación indicada en el fichero oplog
- lastHeartbeat
  El tiempo de la última recepción de un ping desde un miembro secundario al primario
- pingMs
  Promedio de tiempo de ping entre cada miembro secundario con el primario.
- errmsg
  Cualquier mensaje de estado elegido por el miembro secundario para devolver al primario cuando éste realizar ping. Son mensajes meramente informativos y no mensajes de error.

#### 4.4.3.3.2. Bucles en el sistema de replicación

Un bucle de replicación es cuando los miembros terminan replicando el uno del otro, por ejemplo A sincroniza desde B que sincroniza desde C que sincroniza desde A.

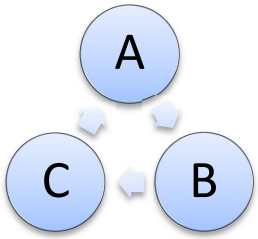


Imagen mongodb 5

Como ninguno de los miembros del bucle es primario no reciben operaciones de replicación y se atrasan. Lo positivo es que

no es posible la existencia de bucles cuando se elige la sincronización automática en el sistema de replicación. Por lo que sólo se puede dar el caso de bucles si manualmente cambiamos el sistema de replicación.

#### 4.4.3.3.3. Encadenamiento

El encadenamiento es cuando una instancia secundaria se sincroniza desde otra secundaria.

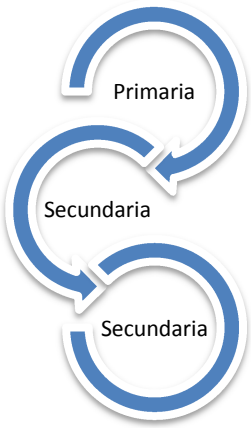


Imagen mongodb 6

#### 4.4.3.3.4. Calculo del lag

Uno de los indicadores más importantes para realizar un seguimiento de la replicación es velocidad de sincronización de las instancias secundarias respecto a las primarias.

El lag es el tiempo transcurrido entre que la instancia primaria ejecuta una operación y que la instancia secundaria ejecuta la misma operación.

Puede usar `rs.status()` para ver el estado de la replicación pero también puede obtener un resumen rápido ejecutando `db.printReplicationInfo()` en instancias primarias y `db.printSlaveReplicationInfo()` en las secundarias.

`db.printReplicationInfo` muestra un resumen del fichero oplog de una instancia primaria:

```
configured oplog size: 10.48576MB
log length start to end: 34secs (0.01hrs)
oplog first event time: Tue Mar 30 2010 16:42:57 GMT-0400 (EDT)
oplog last event time: Tue Mar 30 2010 16:43:31 GMT-0400 (EDT)
now: Tue Mar 30 2010 16:43:37 GMT-0400 (EDT)
```

#### 4.4.3.3.5. Modificación del fichero oplog

Si el equipo que actúa como principal tiene un fichero oplog de un tamaño que permite almacenar operaciones ocurridas desde cierto tiempo, los equipos secundarios tendrán ese mismo tiempo para actualizarse sin perder datos. En el caso de no hacerlo tendrían que realizar una sincronización desde cero.

Por lo general se debe tener un tamaño de fichero de oplog que permita almacenar las operaciones ocurridas desde hace dos días. Desafortunadamente, no hay manera fácil de saber cuándo se va a llenar y tampoco hay manera de modificar su tamaño con el servidor en funcionamiento.

#### 4.4.3.3.6. Restauración desde una secundaria retrasada

Por alguna operación se elimina la base de datos de la primaria, por suerte tenemos una copia de la base de datos en una secundaria con sincronización retrasada, con lo cual se puede restaurar la base de datos previa a la operación que causó el borrado.

Se puede realizar de varios modos, pero el más sencillos es: apagar todos los miembros, eliminar los datos de sus directorios, asegurándose que todos los miembros incluso los secundarios tienen los directorios vacíos y por último restaurar todos los miembros con los datos almacenados en la instancia secundaria retrasada.

### *4.4.3.4. Maestro-Esclavo*

MongoDB originalmente soportaba la configuración más tradicional de maestro y esclavo.

Existen dos motivos para la elección de la configuración de maestro-esclavo de en vez del sistema de réplica. Una de ellas es que se necesiten más de 11 esclavos y la otra la necesidad de replicar bases de datos individuales.

Esta configuración es obsoleta pero de todos modos cuando se cumplan alguno de los dos motivos anteriores es posible su utilización. Para su implementación únicamente hay crear el proceso mongod maestro con el parámetro --master y para el proceso mongod esclavo tenemos dos opciones de parámetros --slave o --source donde se indica el host maestro.

## 4.5.Sharding

La fragmentación de datos permite separar las colecciones por conjuntos de documentos en diferentes instancias o fragmentos. Esta estrategia te permite escalar tu base de datos horizontalmente y aumentar el rendimiento al agregar más equipos para repartir la información en lugar de obligar a mejorar el que tienes. La mayoría de las veces resulta más costoso tener un único computador de altas capacidades que varios de gama inferior.

Sin embargo para sistemas de menos de tres shard se pierde rendimiento respecto a un sistema sin fragmentar debido al aumento de la información en movimiento, mantenimiento del metadata y el enrutamiento. A partir de sistemas mayores el rendimiento aumentara de forma lineal.

Se puede realizar de manera manual creando diferentes bases de datos para diferentes datos, cada vez que queramos tratar un dato deberemos conectarnos a la base de datos correcta. Pero esta forma de actuar dificulta el añadir o eliminar nodos al clúster y el acceso a los datos, debido a que siempre el usuario es el responsable de saber dónde está el dato o la base de datos correcta.

MongoDB soporta autosharding, mongodb automatiza el balance de datos a través de los shards y hace más fácil el añadir y remover capacidad. Uno de los objetivos del sharding en mongodb es hacer que un clúster de 5,10 o 100 máquinas parezca a la vista del cliente como una única máquina.

En MongoDB la unidad de base de datos que se fragmenta son las colecciones. Por lo tanto una colección que sea declarada como fragmentada podría poseer distintos documentos en los fragmentos del clúster.

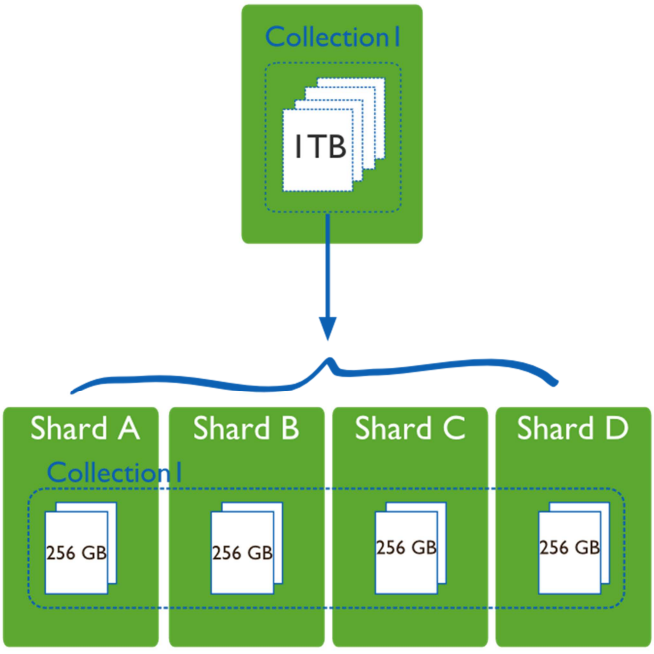


**Imagen mongodb 7. [35]**

Un único documento nunca estará repartido entre fragmentos. Un documento puede tener un tamaño máximo de 16MB, en caso de necesitar mayor tamaño para un documento se necesitaría implementar la solución de GridFS el cual separa el documento en varios trozos o chunks.

Un clúster de fragmentación suele tener la siguiente estructura:

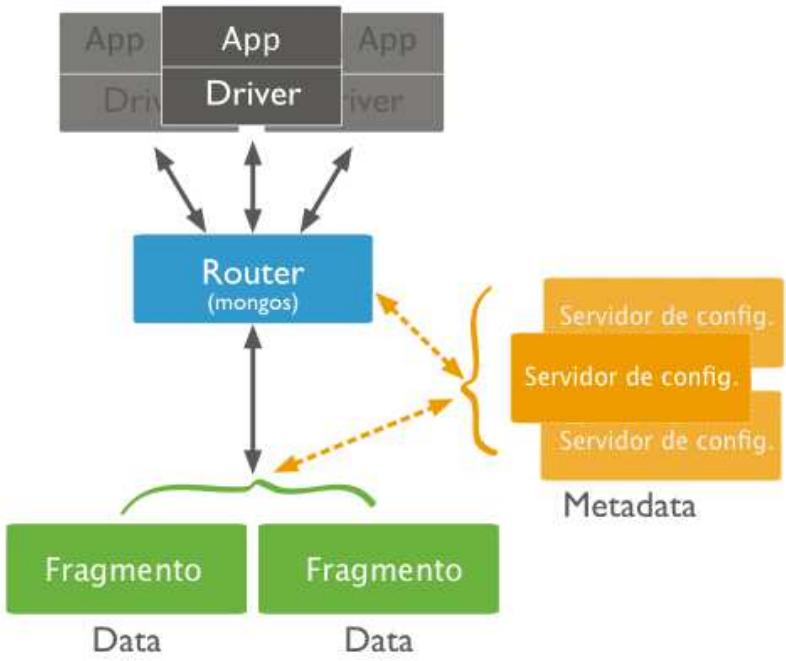


**Imagen mongodb 8 [36]**

Como se observa en la figura anterior, existen cuatro elementos claves: Aplicación y drivers, Fragmento, Router y Servidores de configuración.

- Aplicación y drivers

  Las aplicaciones cuando necesitan comunicarse con la base de datos de MongoDB lo hacen a través de un *driver*, estos tienen implementados los métodos y protocolos necesarios para comunicarse correctamente con la base de datos encapsulando la complejidad del proceso al desarrollador.

- Fragmento

  Un fragmento o *shard* es aquel miembro del clúster que posee los datos fragmentados de las colecciones que componen la base de datos. Éste a su vez suele ser un sistema de replica; sin embargo en ambientes de desarrollo podría ser una única instancia por fragmento.

- Router

  A causa de ver el clúster como un todo, el router es el encargado de recibir las peticiones y de dirigir las operaciones a los fragmentos correspondientes.

En ambientes de producción es habitual la existencia de varios routers para balancear la carga de los mismos.

- Servidores de configuración.

Este tipo de miembros se encargan del almacenamiento del metadata del clúster de fragmentación, es decir, contiene la información relativa a donde se encuentra almacenado cada fragmento. Esta información es almacenada en caché por el router para lograr un óptimo tiempo de procesamiento.

### 4.5.1. Sharding en único equipo

En este apartado se estudia la fragmentación teniendo todos los componentes en un mismo equipo físico. Como se observa en la imagen se tiene un usuario mongo que se conecta a un enrutador mongos que se encarga de gestionar las peticiones sobre la base de datos que está fragmentada en los shard que son procesos mongod.

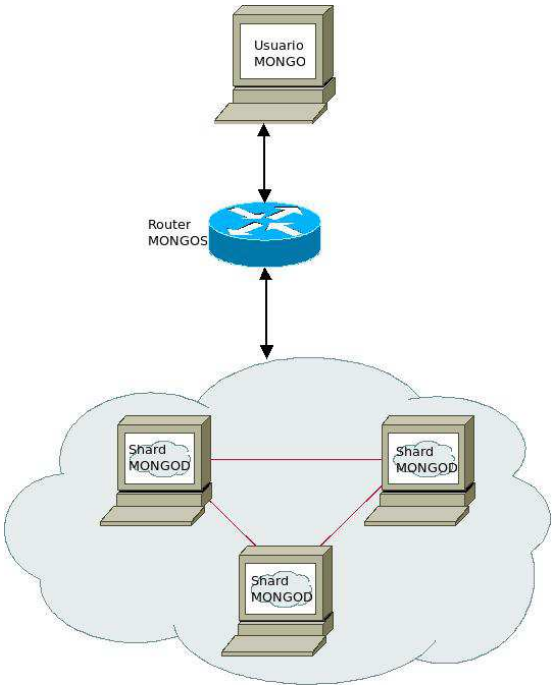


**Imagen mongodb 9**

Para esta implementación abrimos una consola mongo sin conexión a ninguna base de datos.

```
mongo –nodb
```

Desde esta misma consola se crea un clúster formado por tres shard. Ejecutando el siguiente comando se crea un clúster con tres shard, procesos mongod, y en enrutador, proceso mongos. El comando por defecto asigna los puertos a los shard de forma correlativa, siendo el primero el 30000. Y al enrutador le asigna el puerto número 30999 también por defecto.

```
cluster = new ShardingTest({“shards”:3,”chunksize”:1})
```

Ahora es el momento de conectarse al enrutador. Desde otra consola ejecutamos el siguiente comando

```
mongo  --host localhost --port 30999
```

Una vez conectados se puede observar el estado de nuestro sistema de fragmentación por medio del comando `sh.status()`, que imprime por pantalla algo parecido a la siguiente imagen:


**Imagen mongodb 10**

Se crea una colección users dentro de la base de datos test

```
for(var i=0;i<10000;i++){
db.users.insert({“username”:”user”+i,”created_at”:new Date()});
}
```

Si se vuelve a solicitar el estado del sistema de fragmentación, se puede observar que la colección recientemente creada está almacenada sin fragmentar en un único shard. Este shard ha sido elegido por el enrutador de forma aleatoria. Para poder fragmentar se debe activar sobre la base de datos:

```
sh.enableSharding(“base_datos”)
```

Cuando se divide una colección se tiene que elegir una llave, shard key, que se usa como referencia para la partición de la colección. Es un elemento del documento para dividir la colección según el rango de valores que tome en los distintos documentos que forman la colección.

```
db.users.ensureIndex({“key_elegida”:1})
```

Si se desea comprobar cómo queda dividida la colección solo tendremos que comprobar el estado del sistema de fragmentación. `sh.status()`.

```
shards:
	{  "_id" : "shard0000",  "host" : "localhost:30000" }
	{  "_id" : "shard0001",  "host" : "localhost:30001" }
	{  "_id" : "shard0002",  "host" : "localhost:30002" }
databases:
	{  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
	{  "_id" : "test",  "partitioned" : true,  "primary" : "shard0001" }
		test.users
			shard key: { "username" : 1 }
			chunks:
				shard0000	1
				shard0002	1
				shard0001	1
			{ "username" : { "$minKey" : 1 } } -->> { "username" : "user5210" } on : shard0000 Timestamp(2, 0)
			{ "username" : "user5210" } -->> { "username" : "user9425" } on : shard0002 Timestamp(3, 0)
			{ "username" : "user9425" } -->> { "username" : { "$maxKey" : 1 } } on : shard0001 Timestamp(3, 1)
```

Imagen mongodb 11

Como se observa en la imagen cada shard tiene un fragmento. El shard0000 contiene desde el valor más pequeño hasta el valor user5210, el shard0001 contiene desde user5210 sin contenerlo hasta el user9425 y por último el shard0002 contiene desde el último elemento del anterior shard hasta el máximo valor que toma el elemento.

### 4.5.2. Sharding en varios equipos

Cuando se realiza el sharding repartiendo los fragmentos por distintos equipos es obligatorio el uso de uno o tres, nunca 2, configuradores de servicio que mantienen la información sobre la localización de cada fragmento que componen las bases de datos. Tanto los shard como los configuradores de servicio son procesos mongod.

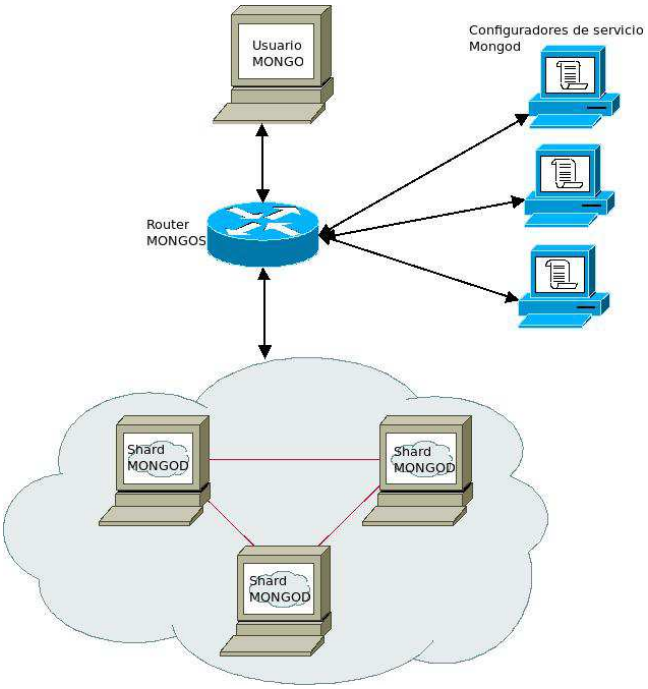


Imagen mongodb 12

Para explicar mejor todos los pasos a seguir se utiliza un ejemplo con una infraestructura de 7 equipos conectados en una misma LAN, que vamos a denominar equipo seguido de un número del 1 al 7. Las direcciones IP serán 192.168.1.x, donde la x se corresponde con el número de equipo.

El equipo1 contiene el enrutador y el cliente. Los equipos del 2 al 4 tendrán los configuradores de servicio. Y los tres restantes equipos serán los shard del sistema de fragmentación.

| Equipo | IP | Puerto por defecto | Directorios necesario | Tipo de procesos |
|---|---|---|---|---|
| 1 | 192.168.1.1 | 27017 | Ninguno | mongo y mongos |
| 2 | 192.168.1.2 | 27019 | configsvr1 | mongod |
| 3 | 192.168.1.3 | 27019 | configsvr2 | mongod |
| 4 | 192.168.1.4 | 27019 | configsvr3 | mongod |
| 5 | 192.168.1.5 | 27017 | shard1 | mongod |
| 6 | 192.168.1.6 | 27017 | shard2 | mongod |
| 7 | 192.168.1.7 | 27017 | shard3 | mongod |

Se crean los directorios para cada uno de los procesos de tipo mongod, los procesos mongos como no almacenan datos no es necesario crear directorios para ellos.

Se arranca cada proceso mongod

Equipo 2 `mongod --configsvr --dbpath raíz/data/configsvr1`

Equipo 3 `mongod --configsvr --dbpath raíz/data/configsvr2`

Equipo 4 `mongod --configsvr --dbpath raíz/data/configsvr3`

Equipo 5 `mongod --dbpath raiz/data/shard1`

Equipo 6 `mongod --dbpath raiz/data/shard2`

Equipo 7 `mongod --dbpath raiz/data/shard3`

Se inicia el proceso enrutador, mongos, indicándole donde se sitúan los servidores de configuración. Y desde el shell de mongo conectado al mongos añadimos los shard al sistema.

```
mongos --configdb “192.168.1.2:27019”, “192.168.1.3:27019”, “192.168.1.4:27019”

sh.addShard(“192.168.1.5:27017”)

sh.addShard(“192.168.1.6:27017”)

sh.addShard(“192.168.1.7:27017”)
```

Una vez realizado se debe habilitar la fragmentación y elección de la shard key como se explicó en el anterior apartado de este proyecto.

### 4.5.3. Sharding y replicación.

En este apartado se trata que los equipos que actúan como shard sean a su vez el servidor primario de un sistema de réplica. Se sigue el esquema anterior, y los equipos que funcionan como shard ahora además actuarán como replicas secundarias de los otros shard.

| Equipo | IP | Puerto por defecto | Directorios necesario | Tipo de proceso | Sistema de replica |
|---|---|---|---|---|---|
| 1 | 192.168.1.1 | 27017 | Ninguno | mongos | - |
| 1 | 192.168.1.1 | - | Ninguno | mongo | - |
| 2 | 192.168.1.2 | 27019 | configsvr1 | mongod | - |
| 3 | 192.168.1.3 | 27019 | configsvr2 | mongod | - |
| 4 | 192.168.1.4 | 27019 | configsvr3 | mongod | - |
| 5 | 192.168.1.5 | 28001 | shard1 | mongod | rs1 |
| 5 | 192.168.1.5 | 28002 | rs2 | mongod | rs2 |
| 5 | 192.168.1.5 | 28003 | rs3 | mongod | rs3 |
| 6 | 192.168.1.6 | 28002 | shard2 | mongod | rs2 |
| 6 | 192.168.1.6 | 28003 | rs3 | mongod | rs3 |
| 6 | 192.168.1.6 | 28001 | rs1 | mongod | rs1 |
| 7 | 192.168.1.7 | 28003 | shard3 | mongod | rs3 |
| 7 | 192.168.1.7 | 28001 | rs1 | mongod | rs1 |
| 7 | 192.168.1.7 | 28002 | rs2 | mongod | rs2 |

Se crean los directorios para cada uno de los procesos de tipo mongod, los procesos mongos como no almacenan datos no es necesario crear directorios para ellos.

Se arranca cada proceso mongod

Equipo 2 `mongod --configsvr --dbpath raíz/data/configsvr1`

Equipo 3 `mongod --configsvr --dbpath raíz/data/configsvr2`

Equipo 4 `mongod --configsvr --dbpath raíz/data/configsvr3`

Equipo 5
```
mongod --dbpath raiz/data/shard1 --port 28001 --replSet “rs1”
mongod --dbpath raiz/data/rs2 --port 28002 --replSet “rs2”
mongod --dbpath raiz/data/rs3 --port 28003 --replSet “rs3”
```

Equipo 6
```
mongod --dbpath raiz/data/shard2 --port 28002 --replSet “rs2”
mongod --dbpath raiz/data/rs1 --port 28001 --replSet “rs1”
mongod --dbpath raiz/data/rs3 --port 28003 --replSet “rs3”
```

Equipo 7
```
mongod --dbpath raiz/data/shard3 --port 28002 --replSet “rs3”
mongod --dbpath raiz/data/rs2 --port 28002 --replSet “rs2”
mongod --dbpath raiz/data/rs1 --port 28001 --replSet “rs1”
```

Lo siguiente a realizar es activar los sistemas de réplica, para lo cual se conecta a cada uno de los procesos shard.

Sistema replica del shard1:

```
mongo --host 192.168.1.5 --port 28001

rs.initiate()

rs.add(“192.168.1.6:28001”)

rs.add(“192.168.1.7:28001”)
```

Sistema replica del shard2:

```
mongo --host 192.168.1.6 --port 28002

rs.initiate()

rs.add(“192.168.1.5:28002”)

rs.add(“192.168.1.7:28002”)
```

Sistema replica del shard3:

```
mongo --host 192.168.1.7 --port 28003

rs.initiate()

rs.add(“192.168.1.6:28003”)

rs.add(“192.168.1.5:28003”)
```

Se inicia el proceso enrutador, mongos, indicándole donde se sitúan los servidores de configuración. Y desde el shell de mongo conectado al mongos añadimos los shard al sistema indicando que son parte de un sistema de réplica.

```
mongos --configdb “192.168.1.2:27019”, “192.168.1.3:27019”, “192.168.1.4:27019”

sh.addShard(“rs1/192.168.1.6:28001,192.168.1.7:28001”)

sh.addShard(“rs2/192.168.1.5:28002,192.168.1.7:28002”)

sh.addShard(“rs3/192.168.1.6:28003,192.168.1.5:28003”)
```

Una vez seguidos estos pasos se habilita la fragmentación y se elige la keys shard como se indica anteriormente en el punto de sharding en un único equipo.

### 4.5.4. Shard key

Las shard keys determina la distribución de los documentos entre los equipos que forman el clúster de fragmentación. La clave es un campo indexado o un campo existente en todos los documentos que forman la colección.

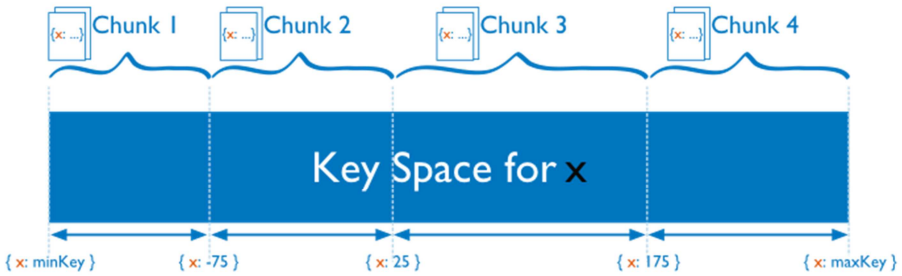


**Imagen mongodb 13 [37]**

Mongodb particiona sus datos usando rangos de valores de la clave. Cada fragmento contiene los documentos cuyo campo que se corresponde con la clave está dentro del rango asignado al fragmento. Si en algún instante un fragmento tiene demasiados documentos mongodb modifica los rangos de los fragmentos para que todos tengan el mismo número de documentos.

Se usan tres tipos de elección de shard keys: clave ascendente, aleatoria y basada en localización. Clave ascendente se elige un campo que se incrementa en la colección, el ejemplo más típico es elegir el identificador que todo documento tiene. En aleatoria se elige un campo al azar y en localización se eligen campos que contienen direcciones ip, latitud y longitud o direcciones.

La elección de la key shard afecta a las operaciones del clúster.

Algunas podrán aumentar la capacidad de escritura del clúster y otras por el contrario no. Una key shard de tipo ascendente no obstaculiza el rendimiento si se tiene una tasa de inserción muy baja, o si la mayor parte de sus operaciones de escritura son actualizaciones distribuidas a través de todo el conjunto de datos. Por lo general esta elección distribuye las operaciones de escritura en todo el clúster.

También podrá afectar en la consultas. Las consultas más rápidas serán aquellas que se hagan usando el campo elegido para la shard key, para las consultas que no usen el citado campo se tendrá que recorrer toda la colección. [37]

### 4.5.5. Administración Sharding

Un clúster de fragmentación es el parte más difícil de administrar en un sistema mongodb. En este apartado se estudia:

- El estado del clúster: miembros que lo componen, donde se almacenan los datos y las conexiones abiertas.
- Como añadir, remover o cambiar miembros del clúster
- Administración de los movimientos de datos, ya sean estos movimientos manuales como automáticos.

#### *4.5.5.1. Información de configuración*

Toda la información relativa a la configuración del clúster se encuentra en las colecciones de la base de datos config de los servidores de configuración. Se puede acceder directamente a estas colecciones, pero no se debe para evitar la modificación o incluso la eliminación de algún servidor de configuración por error. Por este motivo se accede desde el mongos, ya que éste se encarga de que los servidores de configuración estén sincronizados y previene de varias acciones que pueden provocar la eliminación de alguno de ellos.

Por lo general no se debe modificar ningún dato de la base de datos config. Si se realiza algún cambio para que éste surta efecto se tiene que reiniciar todos los mongos

##### 4.5.5.1.1. config.shards

La colección shards almacena los datos sobre los shards que forman el sistema de fragmentación. Cada documento shard contiene un identificador, host y posibles etiquetas.

##### 4.5.5.1.2. config.database

Esta colección guarda los datos relativos a las base de datos almacenadas en el sistema de fragmentación.

Los campos de los documentos de esta colección son un identificador que corresponde con el nombre de la base de datos, un campo booleano llamado partitioned donde se indica si la base de datos está particionada o no, y campo primary que indica el shard principal donde se almacena la base de datos.

#### 4.5.5.1.3. config.collections

Esta colección almacena los datos sobre las colecciones. Los campos más importantes de los documentos de esta colección son:

- _id: nombre de la colección
- key: shard key elegida para la fragmentación de la colección
- uique: indica si la shard key es la única indexación.

#### 4.5.5.1.4. config.chunks

Guarda información de cada uno de los fragmentos o chunks en lo que se divide la colección. Los campos a destacar de estos documentos son:

- _id: identificador del chunk
- ns: colección de la que forma parte
- in: el valor mínimo del rango del chunk
- max: el valor máximo del rango del chunk
- shard: shard en el que se encuentra el fragmento

#### 4.5.5.1.5. config.changelog

Almacena los cambios que se realizan en el sistema de fragmentación. Registra todas las divisiones y migraciones que suceden

#### 4.5.5.1.6. tags

Esta colección se crea si y solo si se asigna etiquetas a los shards

#### 4.5.5.1.7. config.settings

Esta colección contiene los documentos relativos al balanceador y el tamaño del chunk. Cambiando los documentos de esta colección se puede activar y desactivar el balanceador así como modificar el tamaño de los fragmentos. Hay que tener en cuenta que para la modificación de estos documentos no se debe conectar directamente a los servidores de configuración.

#### 4.5.5.2. Conexiones

Existen muchas conexiones entre los componentes de un clúster, en este apartado se estudian solo las relativos a los sistemas de fragmentación.

##### 4.5.5.2.1. Estadísticas de conexión

Hay un comando, connPoolStats, que muestra la información relativa de las conexiones de mongos y mongod.

```
db.adminCommand({“connPoolStats”:1})
```

##### 4.5.5.2.2. Limitar el número de conexiones

Cuando un cliente se conecta a mongos, éste crea una conexión con al menos un shard.

Si se posee muchos mongos, éstos pueden crear más conexiones de las que pueden manejar. Cada mongos y cada mongod soportan como máximo 20000 conexiones. Para prevenir que se sobrepase este valor se usa el argumento maxConns en la línea de configuración del mongos. Este número de se calcula mediante la siguiente fórmula:

$$maxConns = 20000 - (numeroMongoS\ x\ 3) - (cantidadMiembrosReplicaSet\ x\ 3) - (\frac{otros}{numeroMongoS})$$

Cada mongos crea 3 conexiones, una con el cliente otra para el seguimiento de errores y otra para monitorear el estado del shard. Cada shard que sea sistema de replicación tiene otras 3 conexiones uno de la instancia primaria a cada una de las secundarias y otras dos desde las secundarias a la primaria. Y por último se resta el cociente entre otro tipo de conexiones entre el número de mongos del sistema.

#### 4.5.5.3. Administración de servidores

Cuando el clúster crece es necesario aumentar la capacidad o cambiar la configuración. Este apartado versa sobre creación, modificación y eliminación de servidores del clúster. Se pueden añadir nuevos procesos mongos en cualquier instante.

##### 4.5.5.3.1. Modificación de shards

A medida del uso del sistema de fragmentación es posible que se quiera cambiar algún shard. Para este cambio se conecta directamente a la instancia mongod principal del shard. Una vez

realizado el cambio se emite una reconfiguración del sistema de replica que es el shard, los servidores de configuración recogen los cambios y actualizan los documentos necesarios de la colección config.shrads automáticamente, no se debe modificar manualmente esta colección.

La única excepción es que el sistema de fragmentación se haya iniciado con shards simples, sin que estos formen parte un sistema de replicación.

#### 4.5.5.3.2. Eliminación de shards

Por lo general no se deben eliminar shards del clúster. Además la adición y eliminación regular crea un innecesario stress al sistema.

Si se han creado demasiados shards es mejor esperar el crecimiento del sistema clúster hasta que sean necesarios los shards sobrantes, que eliminarlos y después volver a crearlos según la necesidad. De todos modos si fuera necesario se pueden eliminar.

#### 4.5.5.3.3. Cambio de los configuradores de servicios

La modificación de cualquier elemento de los configuradores de servicios es difícil, peligroso y por norma implica un tiempo de inactividad. Antes de realizar modificaciones se debe realizar una copia de seguridad del sistema.

Todos los mongos que forman parte del sistema deben tener los mismos valores en el argumento –configdb mientras se están ejecutando. Para cualquier cambio de los servidores de configuración se deben apagar los mongos y arrancar posteriormente con los nuevos valores.

### *4.5.5.4. Balanceo de datos*

Por lo general, MongoDB realiza el balance de datos de modo automático. En este apartado se explica cómo activar y desactivar el balance automático, así como intervenir en los procesos de balanceo de datos.

#### 4.5.5.4.1. El balanceador

El apagar el balanceador es un requisito previo para casi cualquier actividad administrativa. Para este propósito se usa el siguiente comando

```
sh.setBalancerState(false)
```

A pesar de emitir la orden de detención del balanceo si existe alguna operación de balanceo en ejecución, ésta no se detendrá hasta acabar. Para comprobar si hay alguna operación del tipo de balance de datos se ejecuta el comando `db.locks.find({“_id”:”balancer”})`. En el documento que devuelve el anterior comando existe un campo “state” que indica el estado del balanceador. Sus posibles valores son 0 para cuando está desactivado, 2 para cuando está activado y 1 cuando se está intentando desactivar.

La migración de datos crea carga en el sistema. El shard destinatario deba consultar en el shard originario los todos los documentos a trasladar, realizar una copia de los mismos en el shard destinatario y eliminarlos en el shard originario.

Existen dos situaciones donde se puede crear problemas por la cantidad de migraciones. Cuando se elige una shard keys que provoca migraciones constantes y al añadir un nuevo shard al sistema.

Si las migraciones de datos afectan a las aplicaciones que actúan sobre las bases de datos se pueden programar ventanas temporales donde se realicen las citadas migraciones.

```
db.settings.update({"_id": "balancer"},{"$set":{"activeWindow":
{"start" : "13:00", “stop" : "16:00"}}},true)
```

#### 4.5.5.4.2. Cambio de tamaño del fragmento

Los tamaños pequeños de fragmentos provocan migraciones constantes y por el contrario si se tienen un tamaño grande de fragmento se provoca que la migración sea lenta. Se debe buscar un valor intermedio. Por defecto el tamaño de los fragmentos en un sistema de fragmentación es de 64 megabytes.

Para la modificación de este tamaño se debe realizar por medio de mongos y cambiar la colección config.settings. Se ejecuta el siguiente comando.

```
db.settings.save({“_id”:”chunksize”,”value”:nuevo
_valor})
```

#### 4.5.5.4.3. Movimiento de fragmentos

Como se menciona anteriormente el balanceo de los fragmentos se realiza de modo automático en mongodb. Pero si se desea trasladar fragmentos de forma manual se realiza por medio del comando `sh.moveChunk()`.

#### 4.5.5.4.4. Fragmentos enormes

La elección de determinadas shard key puede tener como consecuencia el crecimiento desigual de los fragmentos. Aunque un fragmento tome un valor mayor al máximo permitido por la configuración almacenada en la colección config.settings no se podrá dividir.

Uno de los indicadores que avisan de la presencia de este problema es el crecimiento a más velocidad de un fragmento respecto a la velocidad de los demás. Para su comprobación basta observar los fragmentos marcados como jumbo en la información mostrada por el comando `sh.status()`.

Si el sistema presenta varios fragmentos enormes se deben distribuir de modo manual en distintos shard del clúster.

## 4.6.Robomongo

Mongodb no incluye interfaz gráfica para su administración, ésta se realiza mediante la Shell de mongo. Sin embargo hay disponibles proyectos independientes centrados en la administración y en la visualización de bases de datos de Mongodb por medio de interfaz gráfica.

En el apartado de documentación de la página web oficial del proyecto mongo encontramos un listado de los diferentes proyectos de interfaz gráfica con una pequeña reseña de cada uno de los mismos. [38]

Robomongo es una interfaz gráfica multiplaforma de código abierto para la administración y visualización de bases de datos Mongodb. Robomongo incorpora el mismo motor de JavaScript que la Shell propia de Mongodb, por lo tanto cualquier instrucción que se ejecute en la Shell también podrá ser ejecutada en Robomongo.

### 4.6.1. Características

- Mongo Shell

  Como anteriormente explicamos permite realizar las mismas tareas que la Shell propia de Mongodb.

- Múltiples Shell

  Permite abrir tantas Shell como sean necesarias. Cada pestaña en Robomongo es un Shell independiente.

- Múltiples Resultados

  Robomongo ejecuta el código orden a orden, lo que implica que devuelve tantos resultados como ordenes ejecute.

- Autocompletación

  Robomongo proporciona Autocompletación para todos los objetos y funciones de la base de datos.

- Sintaxis Resaltada

  Robomongo resalta con diferentes colores los diferentes componentes de una orden.

### 4.6.2. Instalación

Para obtener la aplicación Robomongo accedemos a la página oficial del proyecto www.robomongo.org que tiene el siguiente aspecto:

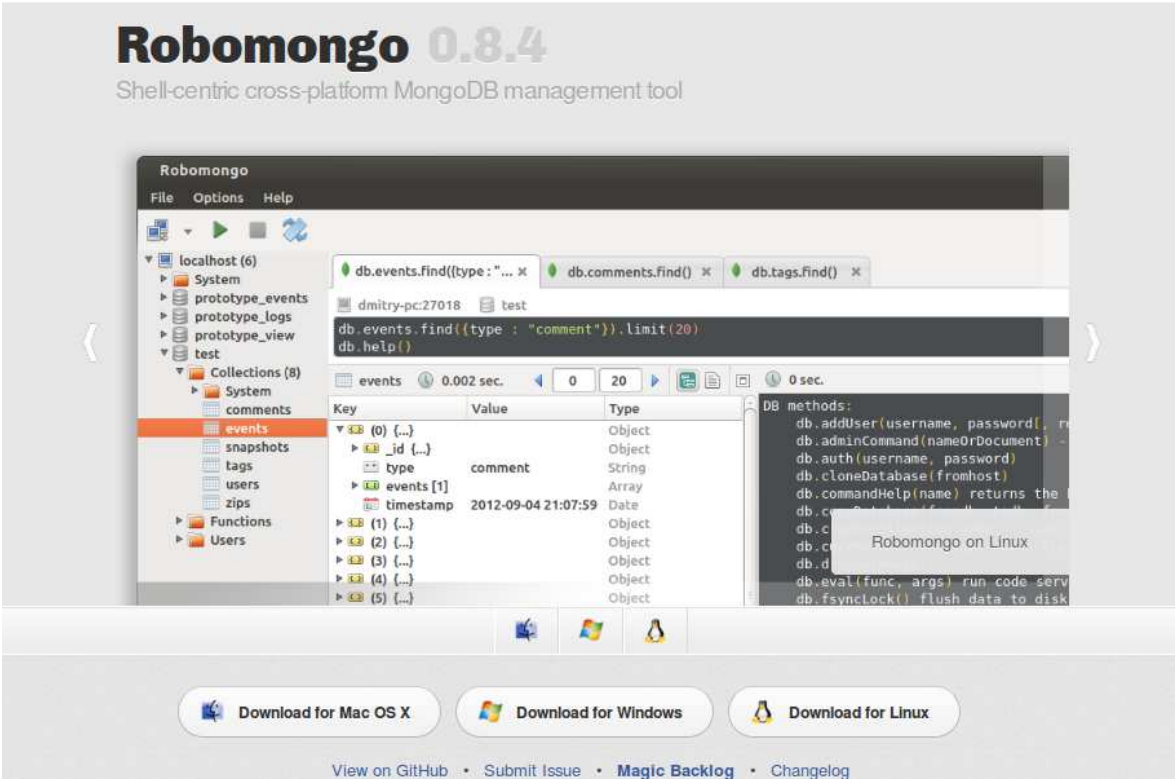


Imagen mongodb 14

Existen tres posibles descargas dependiendo de la plataforma en la que se ejecutará: Linux, Windows y Mac. Al pulsar sobre cada una de los botones de descarga se despliega las opciones relativas a cada plataforma.

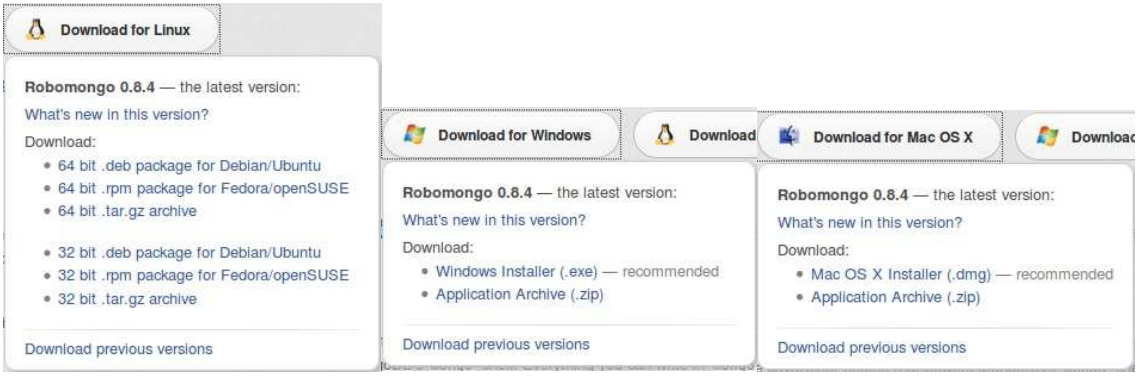


Imagen mongodb 15

Una vez descargado el instalador propio del sistema elegido únicamente habrá que instalar siguiendo las instrucciones.

### 4.6.3. Manejo

Cuando se inicia la aplicación Robomongo se muestra una ventana con las conexiones a bases de datos Mongodb. En esta misma ventana se accede a crear, editar, eliminar y clonar conexiones.

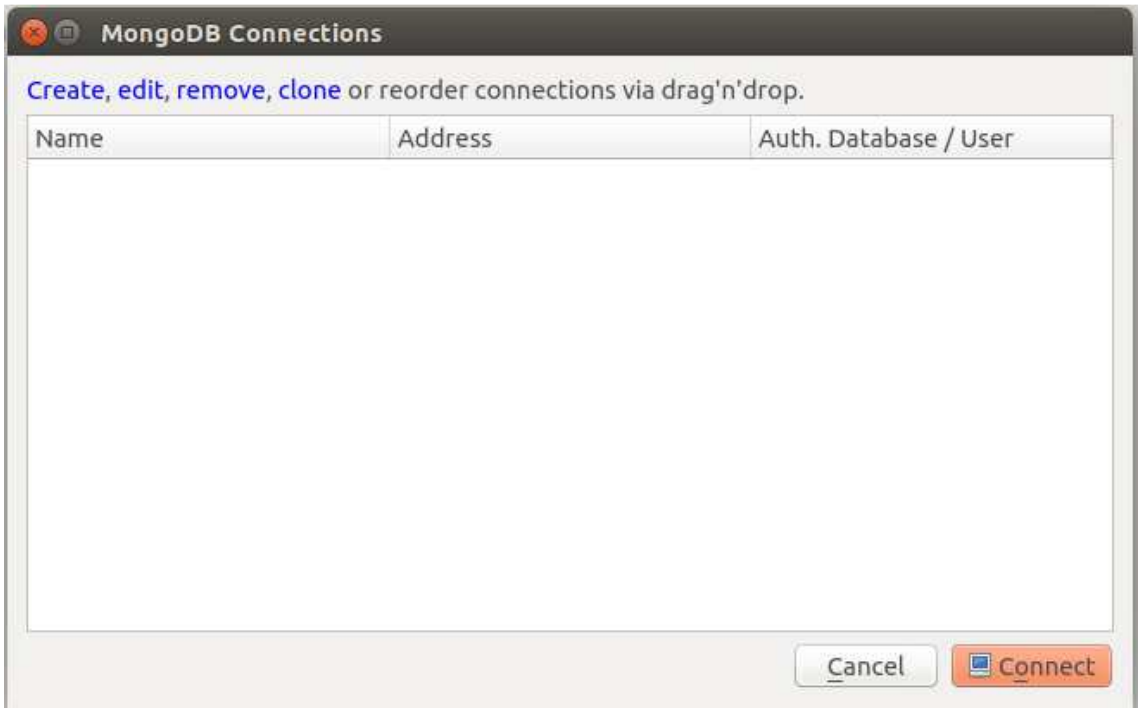


Imagen mongodb 16

Al pulsar en create se abre una ventana con cinco pestañas donde se configuran distintas características de la conexión.

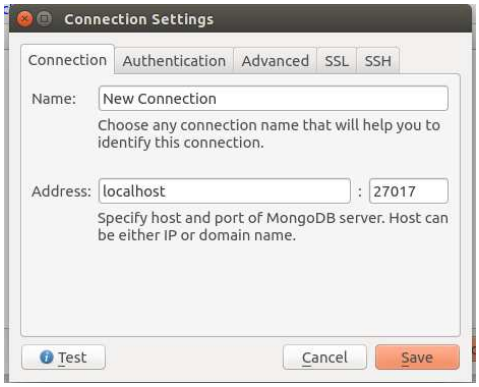


**Imagen mongodb 17**

- Connection
  - Name: Nombre para identificar la conexión
  - Address: Dirección IP y puerto de la máquina donde está la base de datos.

- Authentication
  - Database. Nombre de la base de datos. Si nos autentificamos en admin tendremos acceso a todas la bases de datos almacenadas en el Mongodb.
  - Username. Nombre del usuario
  - Password. Contraseña

- Advanced. Para elegir la base de datos a conectarse.

- SSL. Protocolo de seguridad para conexiones de red

- SSH. Protocolo de acceso seguro a máquinas remotas

Después de conectar se abre la ventana principal de la aplicación, que se encuentra dividida en tres partes: Parte superior encontramos los menús e iconos de acceso directo a algunas funciones del menú. En la parte inferior izquierda encontramos una estructura en forma de árbol con las bases de datos disponibles, y por último en la sección inferior derecha nos encontramos a la Shell junto a los resultados a mostrar.

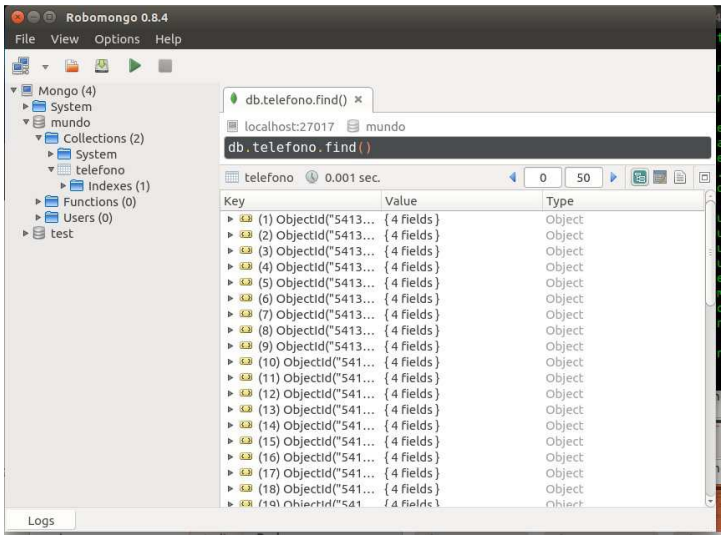


Imagen mongodb 18

Como se observa en la siguiente imagen los resultados se pueden mostrar de tres modos distintos: árbol, tabla y texto.

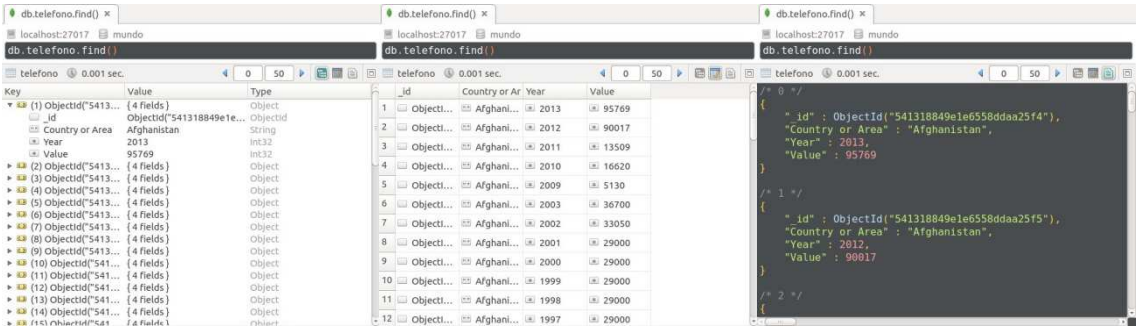


Imagen mongodb 19

# 5. R y RStudio

## 5.1.R

R es un proyecto de software libre de GNU y se podría definir desde dos puntos de vista, por una parte es un lenguaje de programación y por otra un entorno de trabajo, estando ambos orientados al cálculo estadístico y a la generación de gráficas.

Como lenguaje de programación proporciona una amplia variedad de técnicas y recursos para el trabajo con gráficas y análisis estadístico y, a su vez, es altamente ampliable. Cuenta con una comunidad extensa de desarrolladores, investigadores y usuarios. Se distribuye con licencia GNU GPL v2 y está disponible para distintos sistemas operativos de tipo Unix y similares (FreeBSD y Linux), Windows y Mac OS.

Como entorno de trabajo se entiende como un sistema totalmente planificado y coherente y no una acumulación incremental de herramientas muy específicas y poco flexibles, como es frecuentemente el caso con otro software de análisis de datos. En este caso el entorno de trabajo R nos proporciona una serie de utilidades para manipulación de datos, cálculo y representación gráfica.

El lenguaje R es "case sensitive", es decir, distingue entre mayúsculas y minúsculas. Es un lenguaje robusto intenta no dar mensajes de error, pero es posible que al ejecutar un comando no se obtenga ningún error y sin embargo R no esté haciendo lo que se pretendía.

Una característica de R es que se maneja a través de consola en la que se introduce código de programación para obtener los resultados deseados. Pero existen varias IDE gráficas que facilitan el desarrollo de análisis complejos. Para este trabajo, se ha seleccionado RStudio, por ser uno de los entornos más robustos y extendidos.

### 5.1.1. Comandos básicos

Para obtener ayuda acerca de, por ejemplo la función plot, se podrían utilizar los siguientes comandos `help(plot)` y `help.search(plot)`. Él último permite búsquedas más avanzadas.

El comando `source("comandos.R")` permite introducir comandos procedentes de archivos. Útil para la carga de funciones elaboradas por el usuario o bien para la ejecución de macros y scripts.

El comando `sink("archivo")`, nos permite que la salida de los comandos se almacene en archivo. Para devolver la salida a la pantalla se usa nuevamente el comando `sink()`.

Para consultar el código de una función en R, ya sea una función propia de R o bien una del usuario, basta con teclear el nombre en la línea de comandos.

El comando *getwd()* nos devuelve el directorio de trabajo y el comando *setwd()* nos permite modificarlo. Al iniciar el programa R se abre automáticamente un espacio de trabajo, en él se almacenan todos los datos, funciones... usadas durante esa sesion. En cualquier momento se pueden guardar todos los objetos del espacio de trabajo como un archivo con extensión ".RData". También al iniciar el programa R se cargan por defecto unas librerías básicas. A veces es necesario cargar otras librerías se realiza con el comando *library(nombre)*.

El comando *ls()* proporciona un listado de los objetos que hay actualmente en el espacio de trabajo. Para hacer búsquedas de objetos en el espacio de trabajo se pueden utilizar los comandos *apropos()* y find(). Además, el comando *search()* nos da una lista de las librerías cargadas.

Borrado: Para eliminar uno o varios objetos del espacio de trabajo se usa el comando *rm(objectos)*.

Mediante el teclado de flechas se puede acceder a los últimos comandos ejecutados. Además, el comando *history()* nos devuelve un archivo de texto con los últimos comandos ejecutados.[39]

### 5.1.2. Operaciones básicas

| | | |
|---|---|---|
| + | | Suma |
| - | | Resta |
| * | | Producto |
| / | | Cociente |
| ^ | | Potencia |
| log | | Logaritmo |
| exp | | Exponencial |
| sin | | Seno |
| cos | | Coseno |
| tan | | Tangente |
| sqrt | | Cuadrado |
| min | | Valor mínimo de un vector |
| max | | Valor máximo de un vector |
| range | | Vector formado por el mínimo y el máximo de un vector |
| sum | | Suma de todos los elementos de un vector |
| pro | | Producto de todos los elementos de un vector |
| length | | Tamaño de un vector |
| mean(x) | sum(x)/length(x) | Calcula la media de los valores de un vector |
| var(x) | sum(x-mean(x))^2/(length(x)-1) | Cuasi-varianza |
| sort(x) | | Ordena los valores del vector x de forma ascendente |
| seq(a,b,c,d) | | Crea una secuencia de longitud d, |

| | | |
|---|---|---|
| | | que inicia en a y termina en b dando saltos de tamaño c |
| plot(x,y) | | Gráfica de x sobre y |

[40]

### 5.1.3. Listas.

En R, una lista es un objeto consistente en una colección ordenada de objetos, conocidos como componentes.

No es necesario que los componentes sean del mismo tipo, así una lista puede estar compuesta de, por ejemplo, un vector numérico, un valor lógico, una matriz y una función.

El siguiente es un ejemplo de una lista:

```
Lst<-list(nombre="Pedro",esposa="María",no.hijos=3,edad.hijos=c(4,7,9))
```

Los componentes siempre están numerados y pueden ser referidos por dicho número. En este ejemplo, Lst es el nombre de una lista con cuatro componentes, cada uno de los cuales puede ser referido, respectivamente, por `Lst[[1]]`, `Lst[[2]]`, `Lst[[3]]` y `Lst[[4]]`. Como, además, `Lst[[4]]` es un vector, `Lst[[4]][1]` se refiere a su primer elemento.

La función *`length()`* aplicada a una lista devuelve el número de componentes de la lista.[40]

## 5.2. RStudio

RStudio es un entorno de desarrollo integrado (IDE) para R. Es software libre con licencia GPLv3 y se puede ejecutar sobre distintas plataformas (Windows, Mac, o Linux) o incluso desde la web usando RStudio Server. [41]

Es una IDE gratuita y código abierto para el lenguaje R escrito en C++ y usa Qt para la interfaz gráfica de usuario. Existen dos versiones Rstudio Desktop, que se ejecuta de forma local, y RStudio Server que se ejecuta en un servidor al que se accede por navegador web.

### 5.2.1. Instalación de RStudio

Accedemos a la página web http://www.rstudio.com/

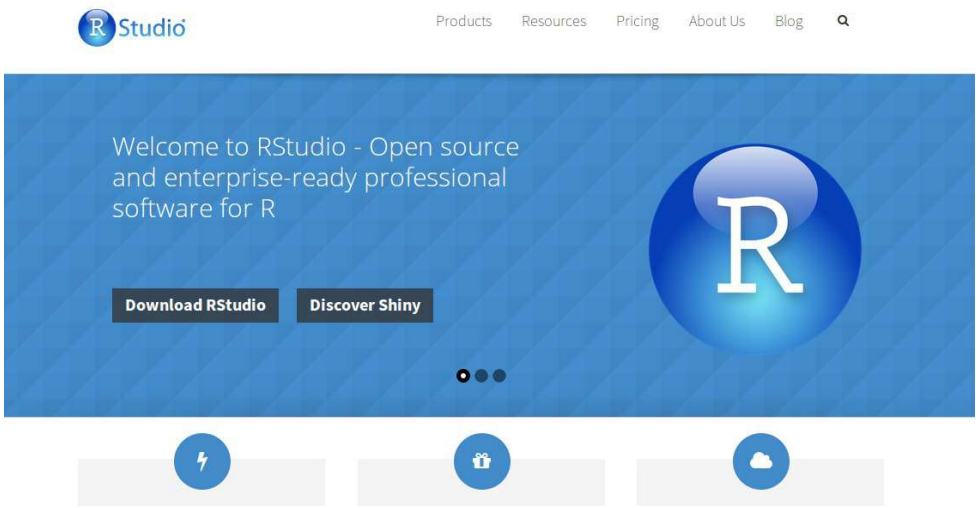


**Imagen R 1. Captura de pantalla**

Haciendo click en el botón Download RStudio, se accede a la página que mostramos a continuación.

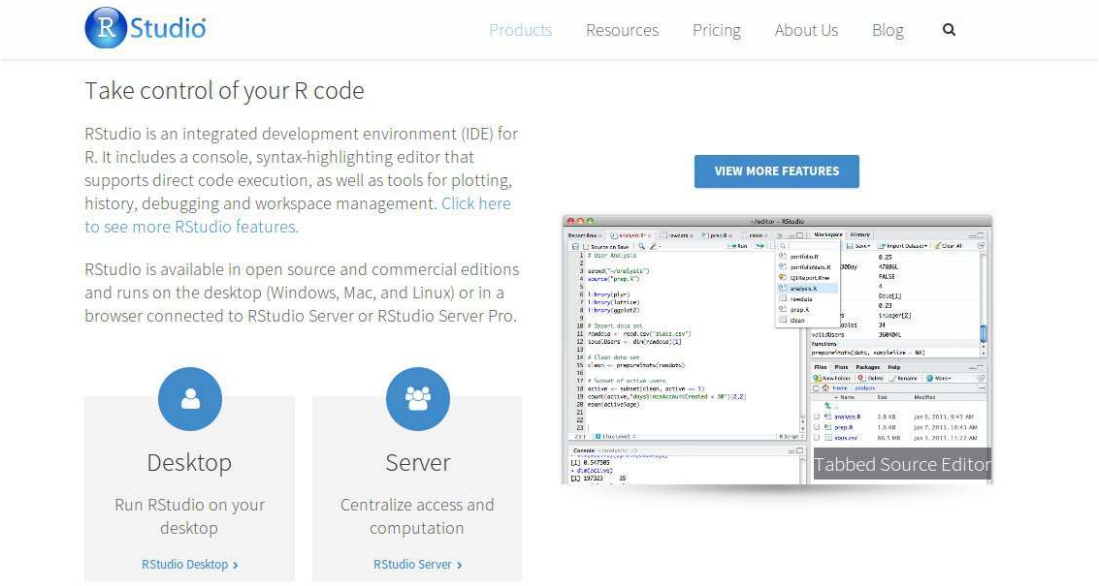


**Imagen R 2. Captura de pantalla**

En las siguientes imágenes se observa las características de las dos versiones existentes de RStudio

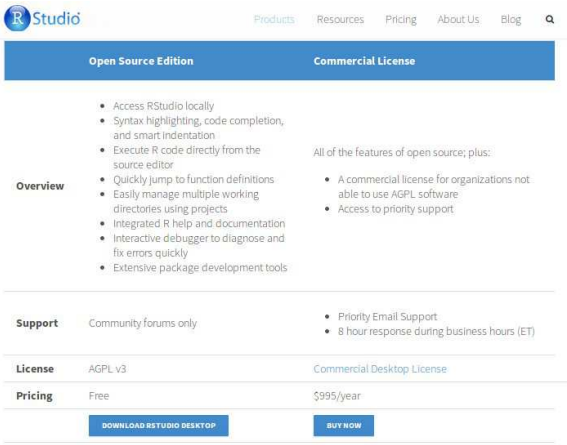


**Imagen R 3. Captura de pantalla**

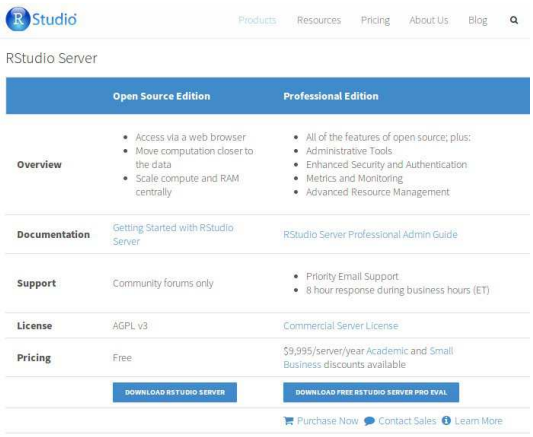


**Imagen R 4. Captura de pantalla**

Se elige la desktop para poder ejecutarla de forma local, apareciendo entonces las diferentes versiones de desktop según la plataforma donde vayamos a instalarla.

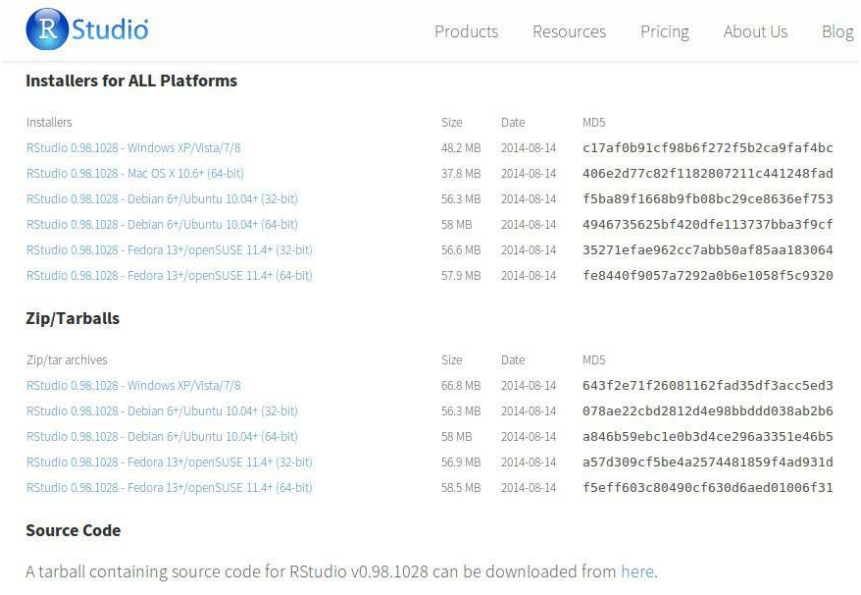


**Imagen R 5. Captura de pantalla**

### 5.2.2. Utilización de RStudio

RStudio está organizado en tres zonas de trabajo distintas, como se aprecia en la siguiente figura:

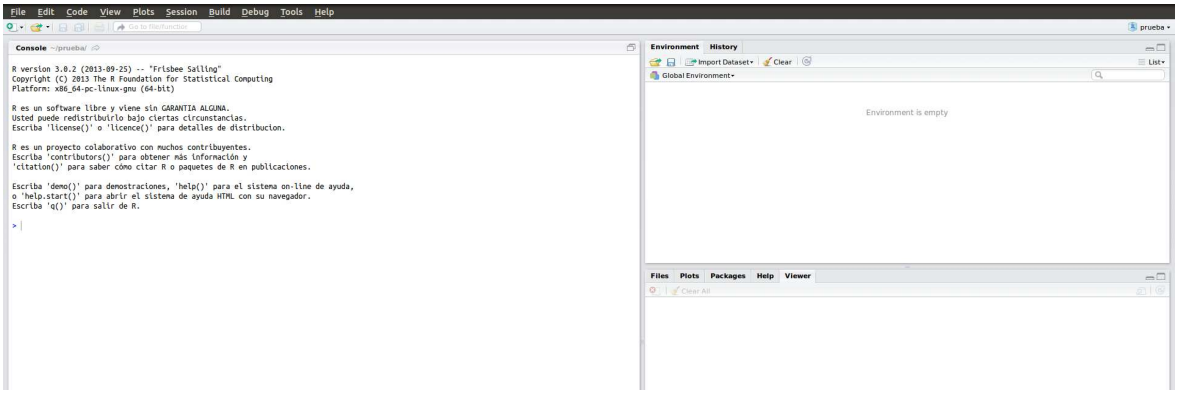


Imagen R 6. Captura de pantalla

- En la zona izquierda hay una consola de R en la que pueden ejecutarse comandos de R.
- La zona superior derecha tiene dos pestañas:
  - *Workspace*, donde aparece la lista de los objetos creados en memoria.
  - *History*, que contiene el histórico de las líneas de código ejecutadas en R
- La zona inferior derecha dispone de cinco pestañas:
  - *Files*, que da acceso al árbol de directorios y ficheros del disco duro.
  - *Plots*, donde aparecen los gráficos creados en la consola.
  - *Packages*, que facilita la administración de los paquetes de R instalados en la máquina.
  - *Help*, en el que se abren las páginas de ayuda.
  - Viewer.

### 5.2.3. Instalación de rmongo y rmongodb.

Rmongo y rmongodb son librerías que permiten el acceso del lenguaje R a las bases de datos de Mongodb.

Rmongodb es el driver oficial de MongoDB, es una herramienta realmente poderosa para el estudio estadístico de Big Data, pero su uso es complicado.

Rmongo es driver no oficial. Su uso es más sencillo que rmongodb al estar limitadas sus funciones. Usa Java como lenguaje con lo cual si queremos usarlo tendremos que tener instalado Java en nuestro equipo. No puede ser usado en equipos con sistemas operativos Mac.[42]

Para instalar estos dos drivers en Rstudio se usa la utilidad  Install Packages que se sitúa en el menú tools

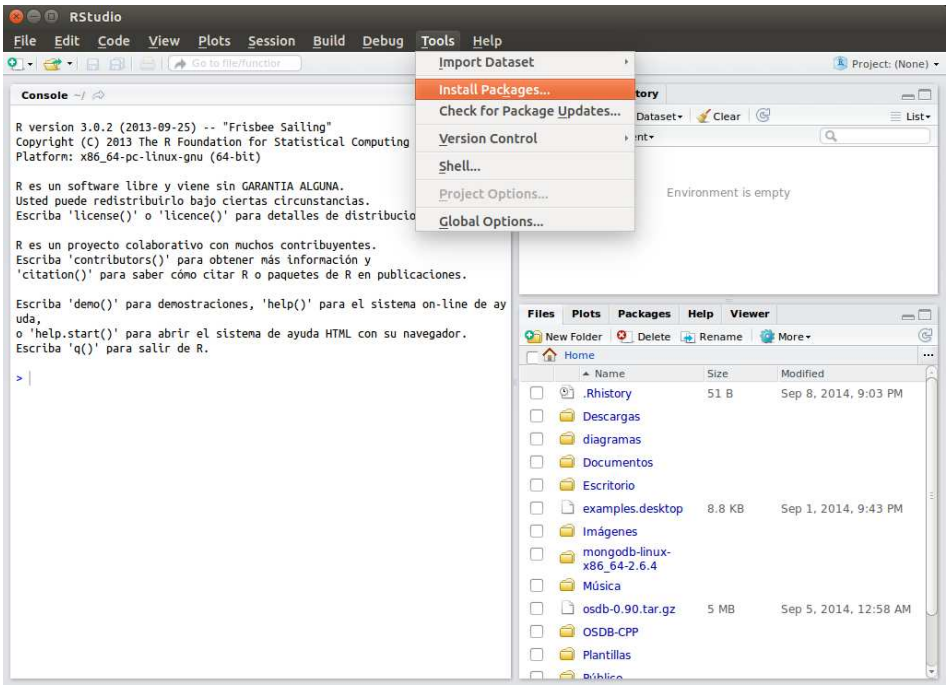


Imagen R 7.Captura pantalla

Al usar esta utilidad se permite el uso de repositorios, con la comodidad que esto conlleva.

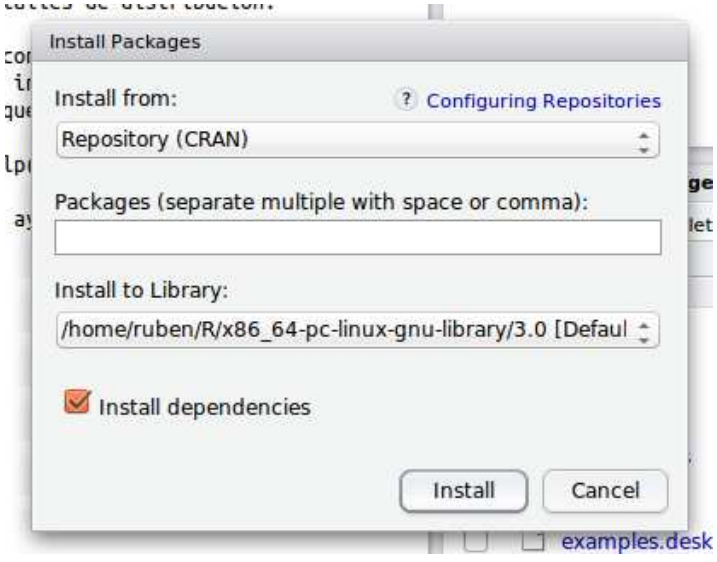


Imagen R 8. Captura de pantalla.

Al empezar a escribir en la sección packages se sugieren los paquetes que empiezan con las mismas letras que las escritas. Se elige el paquete rmongodb e instalamos, después repite el proceso pero esta vez se elige el paquete rmongo.

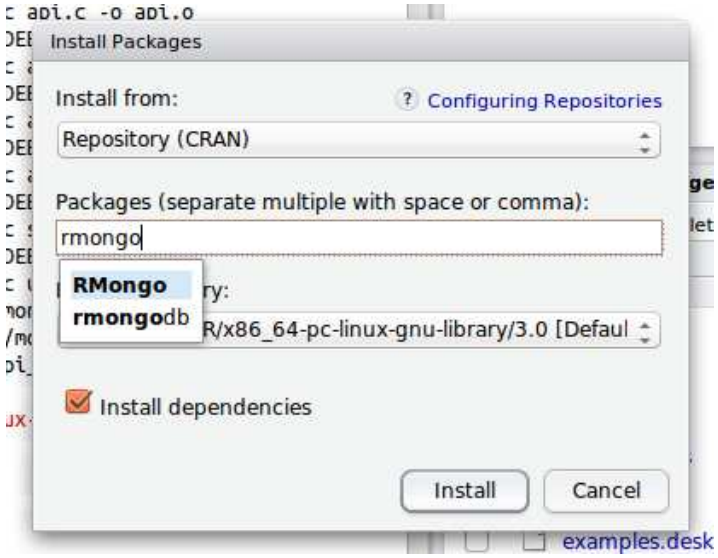


Imagen R 9. Captura de pantalla

El paquete de rmongo para RStudio, por dependencia, pedirá la instalación del paquete de java para RStudio. El citado paquete se llama rjava.

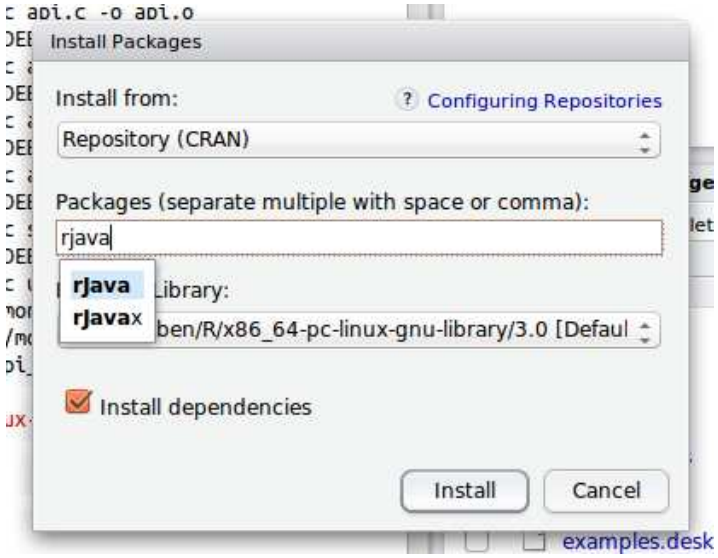


Imagen R 10. Captura de pantalla

### 5.2.4. Uso de Rmongo y Rmongodb.

Para la comprobación de la diferencia en dificultad en el uso de rmongo y rmongodb, a continuación se realiza el mismo ejemplo, en el cual el objetivo es obtener la media de las líneas telefónicas existentes en España por año desde los últimos 45 años.

Los datos se han obtenido, en forma de archivo csv, de las estadísticas oficial y de libre acceso de la ONU. Están formados por el número de líneas telefónicas existentes por año en cada país.[43]

Los citados datos se han importado a una base de datos MongoDb, denominada por el usuario como “onu”, en la colección también denominada por el usuario como “teléfono”.

```
library(RMongo)
mongo1<-mongoDbConnect('onu')
print(dbShowCollections(mongo1))
query<-dbGetQuery(mongo1,'telefono','{"Country":"Spain"}')
for (i in 1:length(query[[2]]))
  x[i]<-query[[2]][i]
mean(x)
```

El anterior código se corresponde a la realización en rmongo. Los pasos que sigue el código: importa el paquete RMongo, crea la conexión con la base de datos y comprueba las colecciones existentes en la misma, realiza la query para obtener sólo los datos relativos a España, como los resultados de la query se devuelve con un vector de listas se transforma la lista correspondiente a los valores anuales de líneas telefónicas en un vector al que posteriormente se le aplica la función *mean()* para obtener la media.

```
library(rmongodb)
mg2 <- mongo.create()
print(mongo.get.databases(mg2))
print(mongo.get.database.collections(mg2, 'onu'))
buf <- mongo.bson.buffer.create()
mongo.bson.buffer.start.object(buf, 'Country')
mongo.bson.buffer.append(buf, 'Country', 'Spain')
mongo.bson.buffer.finish.object(buf)
query <- mongo.bson.from.buffer(buf)
cur <- mongo.find(mg2, 'onu.telefono', query)
country <- NULL
while (mongo.cursor.next(cur)){
      value <- mongo.cursor.value(cur)
     country<-rbind(country,mongo.bson.value(value,'Country'))
  }
mongo.destroy(mg2)
data2 <- data.frame(Country = country)
mean(data2)
```

Como se observa en el anterior código es más complicado realizar el citado ejemplo haciendo uso del paquete rmongodb, y que para la realización de la query es necesario el uso de buffer y para recorrer la misma es necesario un cursor para la obtención de los datos a los que se realiza la función mean()

# 6. Resultado

## 6.1.Implementación de prototipo

Para evaluar la plataforma, en primer lugar se monta un prototipo de sistema MongoDB con replicación y fragmentación para el almacenamiento de una base de datos importada.

| EQUIPO | PROCESADOR | MEMORIA RAM | SIST.OPERATIVO | CONEXION | IP |
|---|---|---|---|---|---|
| 1 | I5 m340 | 4 Gb | Ubuntu 14.10 | Wifi | 192.168.1.4 |
| 2 | QuadCore 2.4 Ghz | 2 Gb | Windows 7 | Lan | 192.168.1.5 |
| 3 | DualCore 1.83 Ghz | 2 Gb | Windows 7 | Wifi | 192.168.1.2 |

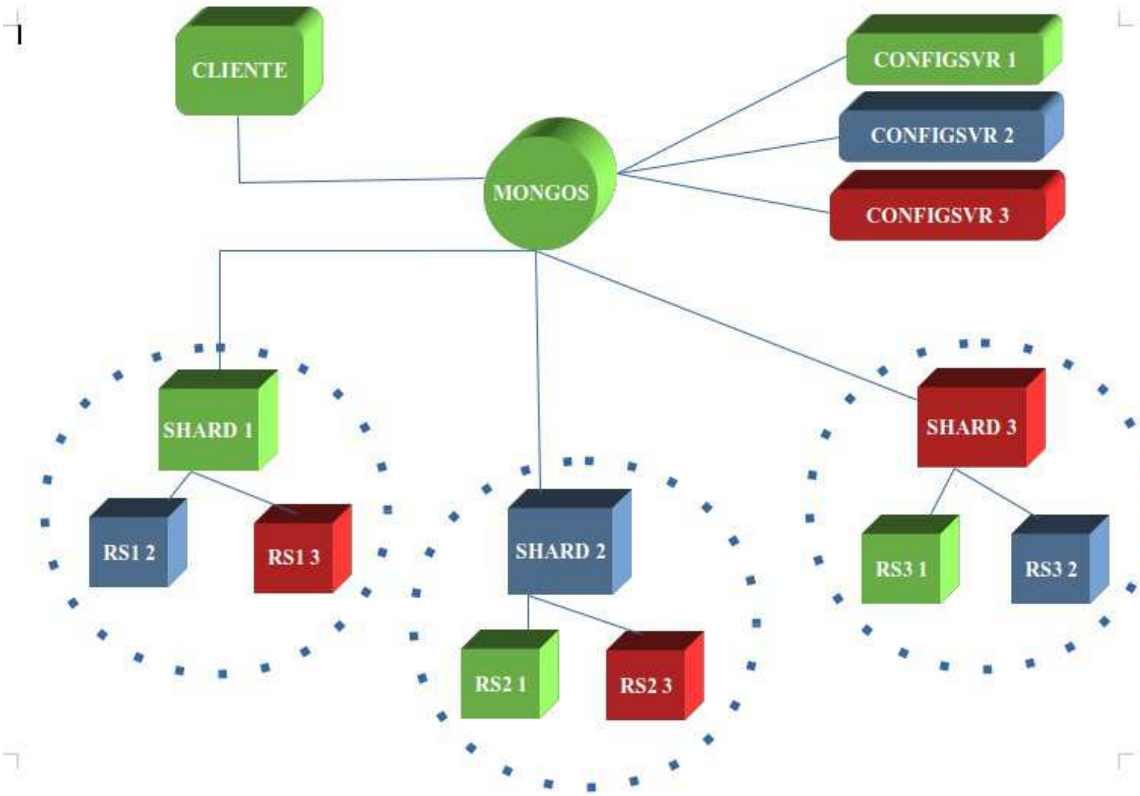


Imagen Resultado 1

En la anterior imagen cada color corresponde con un equipo, siendo el color verde para el equipo 1, el azul para el equipo 3 y por último el rojo para el equipo 2. En la siguiente tabla de cinco columnas la primera indica el equipo, la segunda es el nombre que se le da a cada proceso en la anterior imagen, la tercera, una pequeña descripción, la cuarta indica el tipo de proceso y por último se indica el puerto en el que se ejecuta. La elección de los puertos son por defecto para el proceso mongos y para los servidores de configuración, para los sistemas réplicas se decide usar el 28001 para el sistema de réplica uno, el 28002 para el sistema dos y el 28003 para el tres.

| | | | | |
|---|---|---|---|---|
| Equipo 1 | | | | |
| | Cliente | | mongo | |
| | MONGOS | Enrutador | mongos | 27017 |
| | CONFIGSVR1 | Servidor de configuración 1 | mongod | 27019 |
| | SHARD1 | Instancia primaria RS1 | mongod. | 28001 |
| | RS2 1 | Instancia secundaria RS2 | mongod | 28002 |
| | RS3 1 | Instancia secundaria RS3 | mongod | 28003 |
| Equipo 2 | | | | |
| | CONFIGSVR2 | Servidor de configuración 2 | mongod | 27019 |
| | SHARD2 | Instancia primaria RS2 | mongod. | 28002 |
| | RS1 2 | Instancia secundaria RS1 | mongod | 28001 |
| | RS3 2 | Instancia secundaria RS3 | mongod | 28003 |
| Equipo 3 | | | | |
| | CONFIGSVR3 | Servidor de configuración 3 | mongod | 27019 |
| | SHARD3 | Instancia primaria RS3 | mongod. | 28003 |
| | RS2 3 | Instancia secundaria RS2 | mongod | 28002 |
| | RS1 3 | Instancia secundaria RS1 | mongod | 28001 |

Como se indica en el apartado correspondiente a MongoDB de este proyecto el primer paso a realizar es crear los directorios para cada uno de los procesos mongod y mongos. En la siguiente tabla se observa los directorios a realizar en cada equipo y su relación con el proceso.

| | | |
|---|---|---|
| Equipo 1 | | |
| | MONGOS | db |
| | CONFIGSVR3 | configsvr1 |
| | SHARD1 | shard1 |
| | RS2 1 | rs2 |
| | RS3 1 | rs3 |

| | | |
|---|---|---|
| Equipo 2 | | |
| | CONFIGSVR2 | configsvr2 |
| | SHARD2 | shard2 |
| | RS1 2 | rs1 |
| | RS3 2 | rs3 |

| | | |
|---|---|---|
| Equipo 3 | | |
| | CONFIGSVR3 | configsvr3 |
| | SHARD3 | shard3 |
| | RS2 3 | rs2 |
| | RS1 3 | rs1 |

Se inician los procesos de los configuradores de servicio:

Equipo 1: `./mongod --configsvr --dbpath  /data/configsvr1`

Equipo 2: `mongod --configsvr --dbpath C:\data\configsvr2`

Equipo 3: `mongod --configsvr --dbpath C:\data\configsvr3`

Se inician los procesos shard con sus réplicas:

Equipo 1:

```
./mongod --dbpath /data/shard1 --port 28001 --replSet “rs1”
./mongod --dbpath  /data/rs2 --port 28002 --replSet “rs2”
./mongod --dbpath  /data/rs3 --port 28002 --replSet “rs3”
```

Equipo 2:

```
mongod --dbpath  C:\data\shard2 --port 28002 --replSet “rs2”
mongod --dbpath  C:\data\rs1 --port 28001 --replSet “rs1”
mongod --dbpath  C:\data\rs3 --port 28003 --replSet “rs3”
```

Equipo 3:

```
mongod --dbpath  C:\data\shard3 --port 28003 --replSet “rs3”
mongod --dbpath  C:\data\rs2 --port 28002 --replSet “rs2”
mongod --dbpath  C:\data\rs1 --port 28001 --replSet “rs1”
```

Ahora se inician los sistemas replicas, para lo cual nos conectamos por medio del cliente a cada una de las instancias primarias, se inicia el sistema réplica y se añaden las secundarias. Todos estos pasos se pueden realizar en una misma consola:

```
./mongo --host 192.168.1.4 --port 28001
rs.initiate()
rs.add(“192.168.1.2:28001”)
rs.add(“192.168.1.5:28001”)
exit
./mongo --host 192.168.1.2 --port 28002
rs.initiate()
rs.add(“192.168.1.4:28002”)
rs.add(“192.168.1.5:28002”)
exit
./mongo --host 192.168.1.5 --port 28003
```

```
rs.initiate()

rs.add(“192.168.1.2:28003”)

rs.add(“192.168.1.4:28003”)

exit
```

Una vez iniciados los sistemas de réplicas es el turno de iniciar el enrutador indicándole los servidores de configuración.

```
./mongos --configdb “192.168.1.4:27019”,“192.168.1.2:27019”, “192.168.1.5:27019”
```

El siguiente paso a realizar, es conectarse al recién creado mongos por medio de la shell de mongo. Como la shell y el mongos están situados en el mismo equipo no hace falta indicar el host, y el puerto que se ha usado es el asignado por defecto tampoco es necesario su indicación en la llamada al proceso mongo, por tanto basta con ejecutar:

```
./mongo
```

Ahora es el turno de añadir los shard, como son parte de un sistema de replicación se debe indicar su nombre y los mongod que actúen como instancia secundarias:

```
sh.addShard(“rs1/192.168.1.2:28001,192.168.1.5:28001”)

sh.addShard(“rs2/192.168.1.4:28002,192.168.1.5:28002”)

sh.addShard(“rs3/192.168.1.4:28003,192.168.1.2:28003”)
```

A continuación se introducen los datos, para este prototipo se usan los datos de los vuelos internos que llegan a su hora programada en Estados Unidos durante el año 2013[44]. Los datos originales son un fichero, para cada mes del año 2013, con formato csv. Se indica como ejemplo el mes de enero, para los demás meses se realiza la importación con el mismo comando cambiando el fichero origen:

```
./mongoimport –d “aviones” –c “vuelos” --stopOnError --type csv --file directorio/enero2013.csv --headerline
```

Se importan los datos del fichero enero2013.csv a la colección “vuelos” de la base de datos “aviones”, se interrumpe la operación en caso de error y se toma como claves la primera línea del citado fichero.

Aunque estamos introduciendo los datos en un sistema de fragmentación, al no estar activa, se comprueba que el proceso mongos ha añadido todos los datos en un mismo shard elegido al azar de los tres disponibles, mediante el comando `sh.status()`.

Se activa por tanto la fragmentación en la base de datos y se elige la shard key en la que se va a dividir la colección, en este caso se ha elegido el número de vuelo:

```
sh.enableSharding(“aviones”)

db.vuelos.ensureIndex({“FL_NUM:1”})
```

Se comparan los tiempos de importación de datos y de búsqueda de todos los vuelos de la base de datos que tienen como aeropuerto de origen JFK:

```
db.vuelos.find({“ORIGIN”:”JFK”})
```

La siguiente tabla muestra los resultados obtenidos para la importación a un sistema de mongodb sencillo sin replicación ni fragmentación en un solo equipo:

| Fichero | Líneas | Líneas totales | Tiempo de importación | Tiempo de búsqueda |
|---|---|---|---|---|
| Enero2013 | 509520 | 509520 | 19,090 seg | 0.003 seg |
| Febrero2013 | 469747 | 979267 | 17,901 seg | 0.003 seg |
| Marzo2013 | 552313 | 1531580 | 20,881 seg | 0.004 seg |
| Abril2013 | 536393 | 2067973 | 19,958 seg | 0.005 seg |
| Mayo2013 | 548643 | 2616613 | 20,603 seg | 0.006 seg |
| Junio2013 | 552142 | 3168758 | 20,348 seg | 0.006 seg |
| Julio2013 | 571624 | 3740382 | 22,496 seg | 0.007 seg |
| Agosto2013 | 562922 | 4303304 | 21,344 seg | 0.007 seg |
| Septiembre2013 | 510807 | 4814111 | 19,668 seg | 0.008 seg |
| Octubre2013 | 535345 | 5349456 | 19,591 seg | 0.008 seg |
| Noviembre2013 | 503297 | 5852753 | 19.655 seg | 0.008 seg |
| Diciembre2013 | 516739 | 6369492 | 19,581 seg | 0.009 seg |

A continuación se muestran los resultados para la importación a un sistema de mongodb con replicación y fragmentación activa:

| Fichero | Líneas | Líneas totales | Tiempo de importación | Tiempo de búsqueda |
|---|---|---|---|---|
| Enero2013 | 509520 | 509520 | 25:43 min | 0.012 seg |
| Febrero2013 | 469747 | 979267 | 22:30 min | 0.011 seg |
| Marzo2013 | 552313 | 1531580 | 30:06 min | 0.011 seg |
| Abril2013 | 536393 | 2067973 | 23:54 min | 0.012 seg |
| Mayo2013 | 548643 | 2616613 | 24:12 min | 0.012 seg |
| Junio2013 | 552142 | 3168758 | 29:58 min | 0.013 seg |
| Julio2013 | 571624 | 3740382 | 32:09 min | 0.012 seg |
| Agosto2013 | 562922 | 4303304 | 31:13 min | 0.014 seg |
| Septiembre2013 | 510807 | 4814111 | 25:59 min | 0.013 seg |
| Octubre2013 | 535345 | 5349456 | 23:48 min | 0.013 seg |
| Noviembre2013 | 503297 | 5852753 | 25:22 min | 0.014 seg |
| Diciembre2013 | 516739 | 6369492 | 26:30 min | 0.012 seg |

Como se puede observar los tiempos de importación de datos son mayores en el sistema con fragmentación y replicación debido a que además de importar los datos tiene que repartir los datos por los tres equipos, copiando por tanto los datos tres veces, y todo ello mientras los equipos además de los datos crean tráfico para su sincronización. El beneficio de este sistema radica en el tiempo de búsqueda ya que mientras en el sistema sencillo a más datos el tiempo de búsqueda aumenta linealmente, en el sistema con fragmentación y replicación apenas varía.

## 6.2. Tolerancia a fallos

Para comprobar el correcto funcionamiento del sistema distribuido y las ventajas de la fragmentación y replicación de la base de datos, se realiza la prueba de que el proceso shard3 pierde la comunicación con lo cual una de las réplicas pasa a ser primaría. Esto implica que uno de los equipos pasa a tener dos procesos mongod shard primario sobre cargando el citado equipo. Esto se traduce en un aumento en tiempos de importación, casi se dobla, y de búsqueda en la base de datos pasa de estar alrededor de 0.013 seg a 0.040 seg, ya que el sistema se reagrupa pero funciona más lento.

## 6.3.Análisis de datos con R sobre el sistema distribuido

Una vez realizadas las pruebas de rendimiento, se comprueba la posibilidad de utilizar R para realizar el tratamiento estadístico de los datos sobre el sistema distribuido, en este caso utilizando el driver RMongo, por su mayor sencillez. En el siguiente ejemplo se obtiene una gráfica que representa los vuelos que llegan a su hora durante el año 2013 con origen el aeropuerto JFK de New York respecto varios destinos:

```
library(RMongo)

mongo<-mongoDbConnect('aviones')

print(dbShowCollections(mongo))

dallas<-dbGetQuery(mongo,'vuelos','{"ORIGIN":"JFK","DEST":"DFW"}')

boston<-dbGetQuery(mongo,'vuelos','{"ORIGIN":"JFK","DEST":"BOS"}')

indian<-dbGetQuery(mongo,'vuelos','{"ORIGIN":"JFK","DEST":"IND"}')

chicago<-dbGetQuery(mongo,'vuelos','{"ORIGIN":"JFK","DEST":"ORD"}')

x<-c(length(dallas[[1]]),length(boston[[1]]),length(chicago[[1]]),length(indiana[[1]]))

barplot(x,main="Vuelos_con_origen_JFK",ylab="Cantidad_vuelos",names.arg=c("Dallas","Boston","
Chicago","Indianapolis"))
```

Y la gráfica obtenida sería la siguiente:

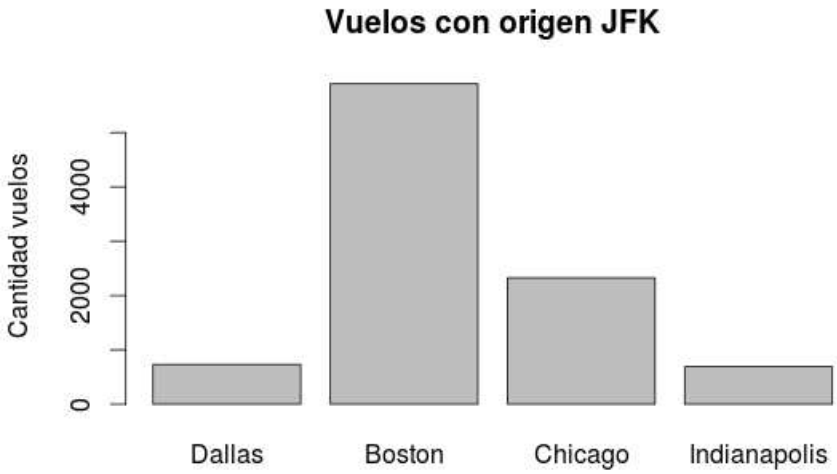


**Imagen Resultado 2**

# 7. Conclusiones y futuras aportaciones.

En este trabajo se han implementado varios prototipos de sistemas de cómputo distribuidos orientados a Big Data, Hadoop y MongoDB, con el fin de evaluar sus características y prestaciones, así como su adecuación al entorno de las pequeñas y medianas empresas, Hadoop es un sistema que da un excelente resultado en Map&Reduce sobre Big Data y su posterior análisis. Sin embargo, por su complejidad y coste (licencia) se descarta el uso directo en PYME´s, ya que la implementación de un sistema real en producción requiere de una inversión considerable tanto en recursos humanos como en licencias (si se desean obtener las utilidades más avanzadas).

MongoDB, es más sencillo de emplear, utiliza una tecnología C++ de código nativo. Se suele elegir para el trabajo con sistemas de gran volumen y sets de datos de tamaño moderado. Como se ha citado es una base de datos NoSQL, pero mantiene un parecido uso a las bases de datos SQL, conocidas desde 1970, con lo que su adaptación es más sencilla.

MongoDB ha demostrado ser una base de datos distribuida, redundante, sencilla y barata que permite el almacenamiento y la computación paralela de grandes conjuntos de datos.

Ejemplo una posible utilidad: en la actualidad muchas tiendas ofrecen acceso gratuito a internet por medio de wifi, permitiendo la conexión de los smartphones de los clientes. Si se almacenan en una base de datos MongoDB las posiciones de los clientes dentro de la tienda, por medio de triangulación, se podrá obtener el recorrido realizado dentro de la tienda, y mediante herramientas estadísticas, como RStudio, averiguar cuáles son los recorridos más habituales, para situar en ellos los productos que nos interese vender en mayor volumen.

El sistema de MongoDB que se implemente en una PYME debería constar como mínimo de tres equipos sencillos. Con este número o mayor de equipos la velocidad del procesamiento compensa el aumento de tráfico en el sistema. Aunque es recomendable que esté formado por más equipos, ya que a causa de un posible fallo de un equipo, el sistema, aunque sin pérdida de datos debido a la replicación, tendría una gran bajada de rendimiento. A mayor sistema menor será la incidencia de los errores en el rendimiento.

## 7.1.Futuras aportaciones

En el presente Proyecto se ha elegido entre MongoDB y Hadoop, pero ambos pueden encajar sin problemas en una típica pila de Big Data. Dependiendo de las características del proyecto que se vaya a llevar a cabo. El modo de realización es empleando MongoDB como almacén de datos operativos en tiempo real y Hadoop para el procesamiento y análisis de datos.

En un escenario típico de producción, los datos que proceden de una aplicación se pueden guardar en múltiples almacenes de datos, cada uno con su propio lenguaje de consulta y funcionalidad. Para reducir la complejidad en estos escenarios, Hadoop puede ser utilizado como un almacén de datos y actuar como un depósito centralizado para los datos de las diversas fuentes. En esta situación, podrían llevarse a cabo

trabajos MapReduce periódicos para la carga de datos de MongoDB en Hadoop. Una vez que los datos de MongoDB, así como los de otras fuentes, están disponibles dentro de Hadoop, los analistas de datos tienen la opción de utilizar MapReduce o Pig para procesar los datos. El resultado puede enviarse de nuevo a MongoDB, asegurando su disponibilidad para posteriores consultas y análisis. [45]

Ingenieros de la Universidad de Toronto, en 2012, realizaron un análisis exhaustivo de distintas bases de datos NoSQL [46], obteniendo una visión del rendimiento bajo diferentes cargas de trabajo. Se identificó a Apache Cassandra como la base de datos NoSQL que daba mejor rendimiento. En este estudio no se incluyó MongoDB, por tanto sería de interés el estudio comparativo de Apache Cassandra y MongoDB.

Se podría implementar un sistema colaborativo de procesamientos de datos de investigaciones llevadas a cabo por la Universidad. Consistente en un sistema Hadoop o MongoDB para el almacenamiento de los datos en equipos de la Universidad, estos datos se distribuyen por equipo de personas voluntarias para ser procesados y devolver los resultados al sistema.

Aunque se descarta la utilización directa de Hadoop en una PYME típica, por las complejidades de su implantación, mantenimiento y coste, existe la posibilidad de contratar sistemas Hadoop en la nube a grandes empresas (Google, Amazon,IBM,…) que se encargan de su implementación y mantenimiento, con lo que la administración del sistema se simplifica, aunque permanece el inconveniente del coste. Por supuesto, al igual que para Hadoop, esta posibilidad está disponible también para sistemas que utilicen el framework de MongoDB.

# 8. Anexos

## 8.1.Bibliografía

### 8.1.1. Bibliografía. Introducción y Objetivos


- [1] Diya Soubra. The 3V´s that define Big Data
  http://www.datasciencecentral.com/forum/topics/the-3vs-that-define-big-data
- [2] Pelayo Herrera, Andrea.
  Las tres V para triunfar en el Big Data: Volumen, Velocidad y Variación.
  http://www.elmundo.es/economia/2014/05/29/53860ac0e2704e361f8b4583.html
- [3] Jeffrey Dean and Sanjay Ghemawat
  MapReduce: Simplified Data Processing on Large Clusters
  Documento de Google publicado en 2004 de MapReduce
  http://research.google.com/archive/mapreduce.html


### 8.1.2. Bibliografía. Estado de la técnica


- [4] Página oficial de Casandra
  http://cassandra.apache.org/
- [5] Página oficial de VoltDB
  https://voltdb.com
- [6] Página oficial de HBase
  hbase.apache.org
- [7] Página oficial de Apache CouchDB
  couchdb.apache.org
- [8] Página oficial de OrientDB
  http://www.orientechnologies.com/
- [9] Página oficial de Redis
  http://redis.io/
- [10] Página oficial de Accumulo
  https://accumulo.apache.org/index.html
- [11] Página oficial de hypertable
  http://hypertable.org/
- [12] Página oficial de Scalaris
  https://code.google.com/p/scalaris/
- [13] Página oficial de Aerospike
  http://www.aerospike.com/
- [14] Kristof Kovacs
  Cassandra vs MongoDB vs CouchDB vs Redis vs Riak vs HBase vs Couchbase vs OrientDB vs Aerospike vs Neo4j vs Hypertable vs ElasticSearch vs Accumulo vs VoltDB vs Scalaris comparison
  http://kkovacs.eu/cassandra-vs-mongodb-vs-couchdb-vs-redis
- [15]Listado de empresas que usan Hadoop
  https://wiki.apache.org/hadoop/PoweredBy
- [16]Listado de empresas que usan MongoDB
  http://www.mongodb.com/who-uses-mongodb

### 8.1.3. Bibliografía. Hadoop


- Tom White
  Hadoop: The Definitive Guide. Third Edition
  O´Reilly
  ISBN: 978-1-449-31152-0
- Página oficial de Hadoop
  http://hadoop.apache.org
- Página oficial de HDFS
  http://hadoop.apache.org/hdfs
- Documentación oficial de HDFS
  http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- [17] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
  The Google File System
  Documento de Google publicado 2003 sobre su sistema de archivos
  http://research.google.com/archive/gfs.html
- [18]Introducción a Hadoop y su ecosistema. jcasanella
  http://www.ticout.com/blog/2013/04/02/introduccion-a-hadoop-y-su-ecosistema
- [19]Página oficial de Cloudera
  http://www.cloudera.com/content/cloudera/en/home.html
- [20]Curso MOOC “Intro to Hadoop and MapReduce”
  Plataforma Udacity
  https://www.udacity.com/


### 8.1.4. Bibliografía. Mongodb


- Kristina Chodorow
  MongoDB. The Definitive Guide
  O´Reilly
  ISBN: 978-1-449-34468-9
- Página oficial de MongoDB
  http://www.mongodb.org/
- [21] Documentación oficial herramienta mongo
  http://docs.mongodb.org/manual/reference/program/mongo/
- [22] Documentación oficial herramienta mongod
  http://docs.mongodb.org/manual/reference/program/mongod/
- [23] Documentación oficial herramienta mongos
  http://docs.mongodb.org/manual/reference/program/mongos/
- [24] Documentación oficial herramienta mongodump
  http://docs.mongodb.org/manual/reference/program/mongodump/
- [25] Documentación oficial herramienta mongorestore
  http://docs.mongodb.org/manual/reference/program/mongorestore/
- [26] Documentación oficial herramienta bsondump
  http://docs.mongodb.org/manual/reference/program/bsondump/
- [27] Documentación oficial herramienta mongooplog
  http://docs.mongodb.org/manual/reference/program/mongooplog/
- [28] Documentación oficial herramienta mongoexport
  http://docs.mongodb.org/manual/reference/program/mongoexport/

- [29] Documentación oficial herramienta mongoimport
  http://docs.mongodb.org/manual/reference/program/mongoimport/
- [30] Documentación oficial herramienta mongostat
  http://docs.mongodb.org/manual/reference/program/mongostat/
- [31] Documentación oficial herramienta mongotop
  http://docs.mongodb.org/manual/reference/program/mongotop/
- [32] Documentación oficial herramienta mongosniff
  http://docs.mongodb.org/manual/reference/program/mongosniff/
- [33] Documentación oficial herramienta mongoperf
  http://docs.mongodb.org/manual/reference/program/mongoperf/
- [34] Documentación oficial herramienta mongofiles
  http://docs.mongodb.org/manual/reference/program/mongofiles/
- [35] http://docs.mongodb.org/manual/core/sharding-introduction/
- [36] Jonathan Wiesel
  MongoDb desde Cero: Fragmentación- Parte I
  http://codehero.co/mongodb-desde-cero-fragmentacion-parte-i/
- [37] http://docs.mongodb.org/manual/core/sharding-shard-key/
- [38] docs.mongodb.org/ecosystem/tolos/administration-interfaz


### 8.1.5. Bibliografía. R y RStudio


- [39] Manuel Febrero Bande, Pedro Galeano San Miguel, Julio González Díaz, Beatriz Pateiro López.
  Prácticas de Estadística en R. Ingeniería Técnica en Informática de Sistemas. Universidad de Santiago de Compostela.
  http://eio.usc.es/pub/pateiro/files/pubdocentepracticasestadistica.pdf
- [40] R Development Core Team
  Introducción a R. Notas sobre R: Un entorno de programación para Análisis de Datos y Gráficos.
  http://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf
- [41] R y RStudio, instalación y primeros pasos
  http://blog.urcera.com/wordpress/?p=242
- [42]R and MongoDB
  http://www.r-bloggers.com/r-and-mongodb/
- [43]Base de datos ONU
  http://data.un.org/Data.aspx?q=telephone&d=MDG&f=seriesRowID%3a779#MDG


### 8.1.6. Bibliografía. Resultados


- [44]Research and Innovative Technology Administration
  Bureau of Transportation Statistics
  Página de estadística de vuelos internos de USA en hora
  http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

### 8.1.7. Bibliografía. Conclusiones y futuras aportaciones


- [45] Como elegir entre MongoDB y Hadoop para tu proyecto Big Data http://blog.powerdata.es/el-valor-de-la-gestion-de-datos/bid/381761/C%C3%B3mo-elegir-entre-MongoDB-y-Hadoop-para-tu-proyecto-Big-Data

- [46]Solving Big Data Challenges for Enterprise Application Performance Management. http://vldb.org/pvldb/vol5/p1724_tilmannrabl_vldb2012.pdf

## 8.2.Índice de ilustraciones