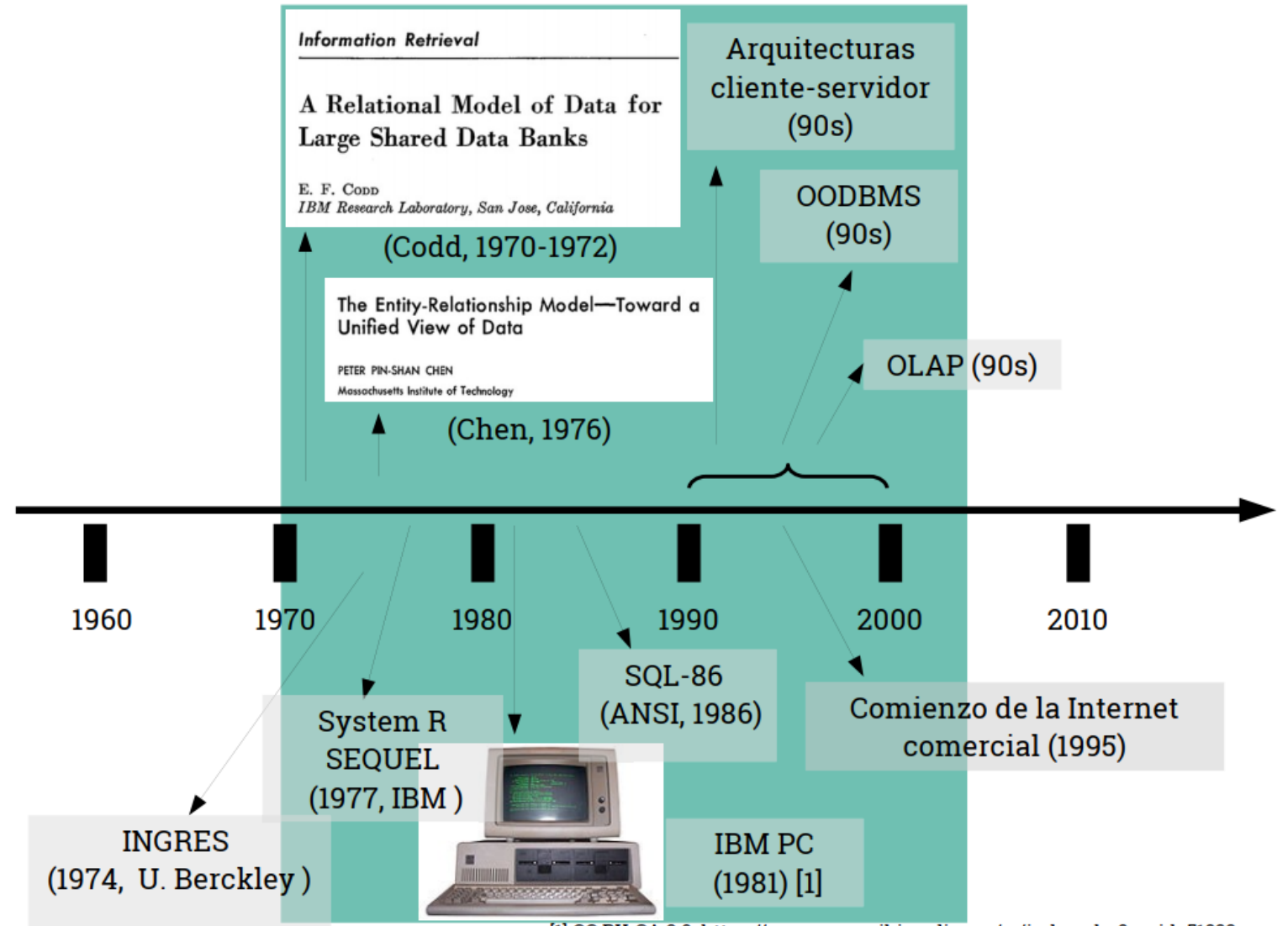


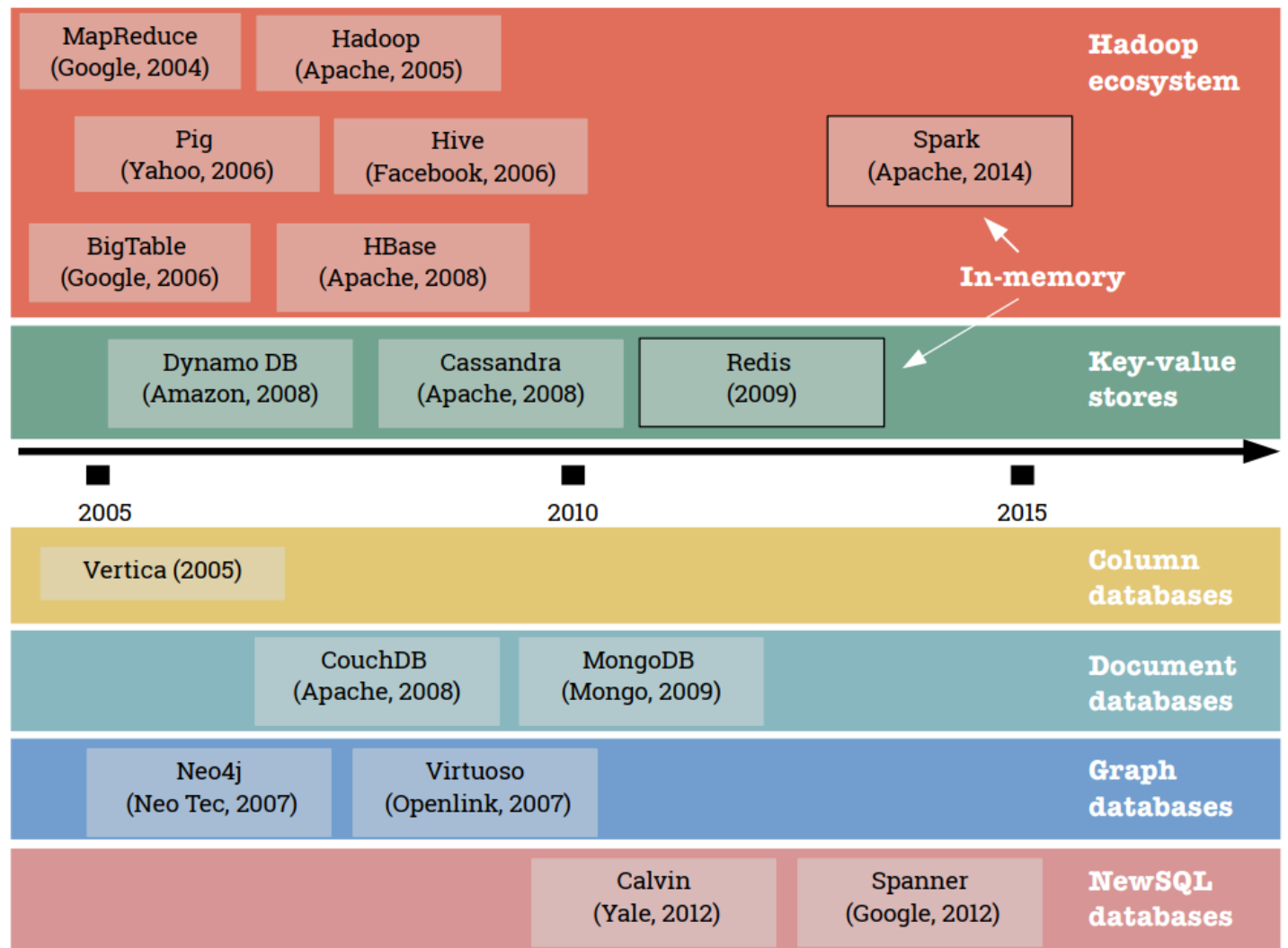
Bases de Datos NoSQL y Big Data

Inicios de las bases de datos computacionales



[1] CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=51833>

Surgimiento de las BDs NoSQL



Introduction of New Database Technologies 1994–2014

Arranged by date of first public release (source: Wikipedia)

~~closed~~ • closed-source • open-source

Server Databases

MySQL

PostgreSQL

MarkLogic

Netezza

Hadoop

CouchDB

Greenplum

Vertica

Neo4J

SimpleDB

Drizzle

Cassandra

Riak

Voldemort

Hypertable

MariaDB

Redis

MongoDB

RethinkDB

OrientDB

Xeround

FlockDB

RavenDB

Clustrix

Membase

Translattice

NimbusDB/NuoDB

Citrusleaf/Aerospike

DynamoDB

Datomic

MemSQL

HyperDex

TokuDB

GenieDB

FoundationDB

Apollo

Cayley

Mobile Databases

SQLite

Realm

NotOnlySQL

NoSQL, Not Only SQL ("no sólo SQL") es una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS) en aspectos importantes, en donde el más destacado es que no usan SQL como el principal lenguaje de consultas.

Los datos almacenados no requieren estructuras fijas como tablas, normalmente no soportan operaciones de reunión (join), ni garantizan completamente las propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad) transaccionales, y habitualmente escalan bien horizontalmente.

Las bases de datos NoSQL están altamente optimizadas para las operaciones recuperar y agregar, y normalmente no ofrecen mucho más que la funcionalidad de almacenar los registros

NoSQL

Se puede decir que la aparición del término NoSQL fue con la llegada de la web 2.0 con aplicaciones como Facebook, Twitter o Youtube, cualquier usuario podía subir contenido, provocando así un crecimiento en las necesidades de almacenamiento.

Las bases de datos NoSQL tienen estructuras que permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales generan ciertos problemas como escalabilidad y rendimiento, en donde se dan cita miles de usuarios concurrentes y con millones de consultas diarias

Las bases de datos NoSQL son sistemas de almacenamiento de información que no cumplen con un esquema entidad relación. Tampoco utilizan una estructura de datos en forma de tabla donde se van almacenando los datos, sino que hacen uso de otros formatos, como clave–valor, familia de columnas o grafos.

Diferencias con SABDR

No utilizan SQL como lenguaje de consultas. La mayoría de las bases de datos NoSQL evitan utilizar este tipo de lenguaje o lo utilizan como un lenguaje de apoyo. Por poner algunos ejemplos, Cassandra utiliza el lenguaje CQL, MongoDB utiliza JSON o BigTable hace uso de GQL.

No utilizan estructuras fijas como tablas para el almacenamiento de los datos. Permiten hacer uso de otros tipos de modelos de almacenamiento de información como sistemas de clave–valor, objetos o grafos.

No suelen permitir operaciones JOIN. Al disponer de un volumen de datos tan extremadamente grande suele resultar deseable evitar las reuniones. Esto se debe a que, cuando la operación no es la búsqueda de una clave, la sobrecarga puede llegar a ser muy costosa. Las soluciones más directas consisten en desnormalizar los datos, o bien realizar el JOIN mediante software, en la capa de aplicación.

Arquitectura distribuida. Las bases de datos relacionales suelen estar centralizadas en una única máquina o bien en una estructura maestro–esclavo; sin embargo, en los casos NoSQL la información puede estar compartida en varias máquinas mediante mecanismos de tablas Hash distribuidas.

Consistencia y transacciones en RDBMS



La noción de consistencia estricta implica que toda instancia de la BD satisface una serie de restricciones (ej: restricciones de clave primaria, clave foránea, etc.)



Las operaciones sobre la BD garantizan que van de un estado consistente a otro estado consistente.



Esto se implementa con transacciones (soporte a acceso concurrente)

¿qué pasa en los sistemas noSQL?

- Se define una nueva noción de consistencia.
- Teorema CAP
- Diferentes enfoques plantean estrategias diferentes:
 - Consistencia estricta (típicamente via locks)
 - Eventual consistency
 - Consistencia ajustable (ej: Dynamo DB)
 - En algunos casos puede requerir de mecanismos de resolución de conflictos

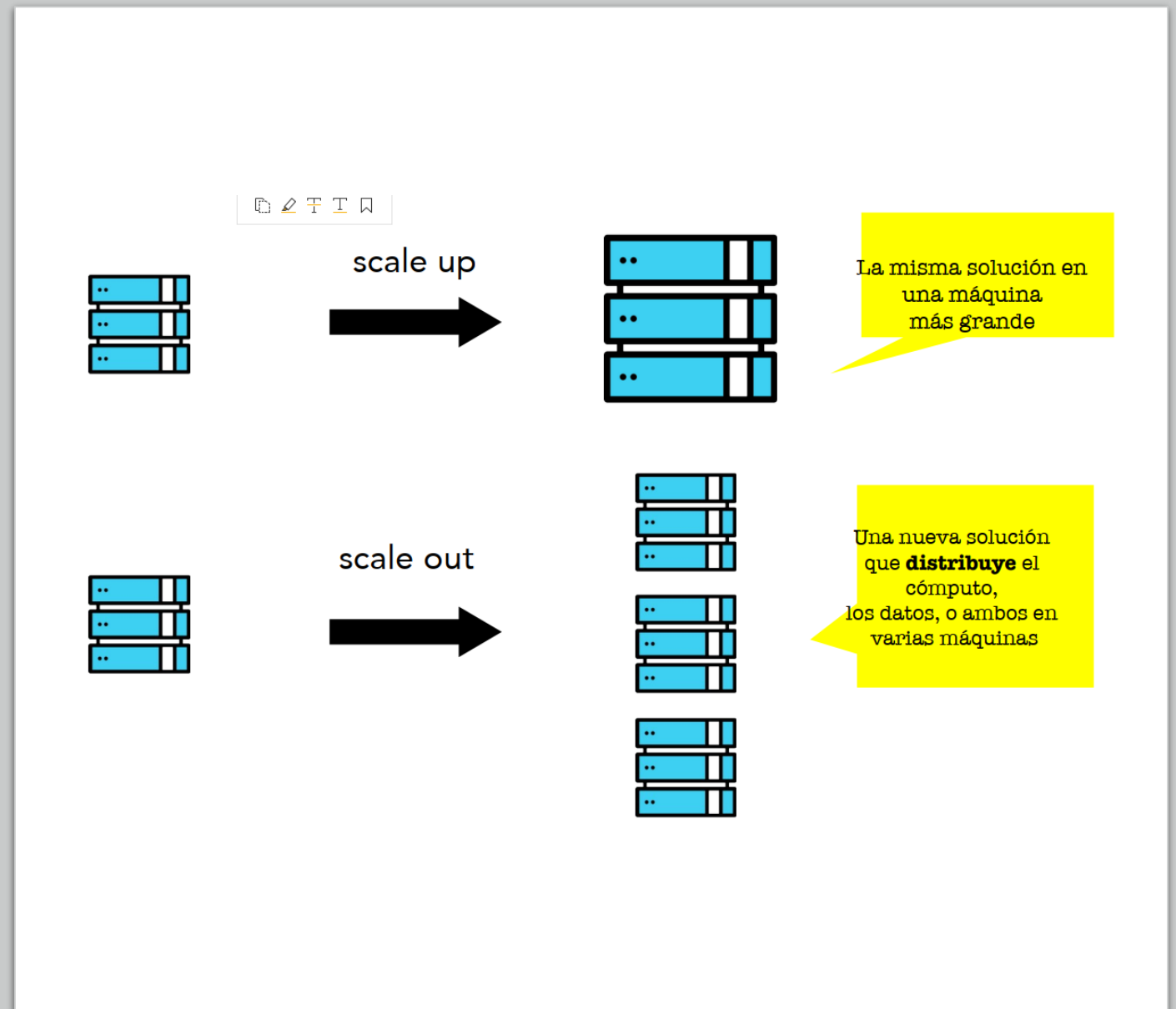


Escalabilidad

- Discutir la escalabilidad significa considerar preguntas como, si el sistema crece de una manera particular:
- ¿Cuáles son nuestras opciones para hacer frente al crecimiento?
- ¿Cómo podemos agregar recursos informáticos para manejar la carga adicional?
- Una aplicación intensiva de datos debería funcionar bien en caso de aumento de la carga de trabajo.
- La carga de trabajo depende de la arquitectura del sistema.
 - Por ejemplo, el número de solicitudes por segundos al servidor web, la relación entre lecturas y escrituras en una base de datos, el número de usuarios activos simultáneamente en una sala de chat o la tasa de visitas en un caché o algo más.

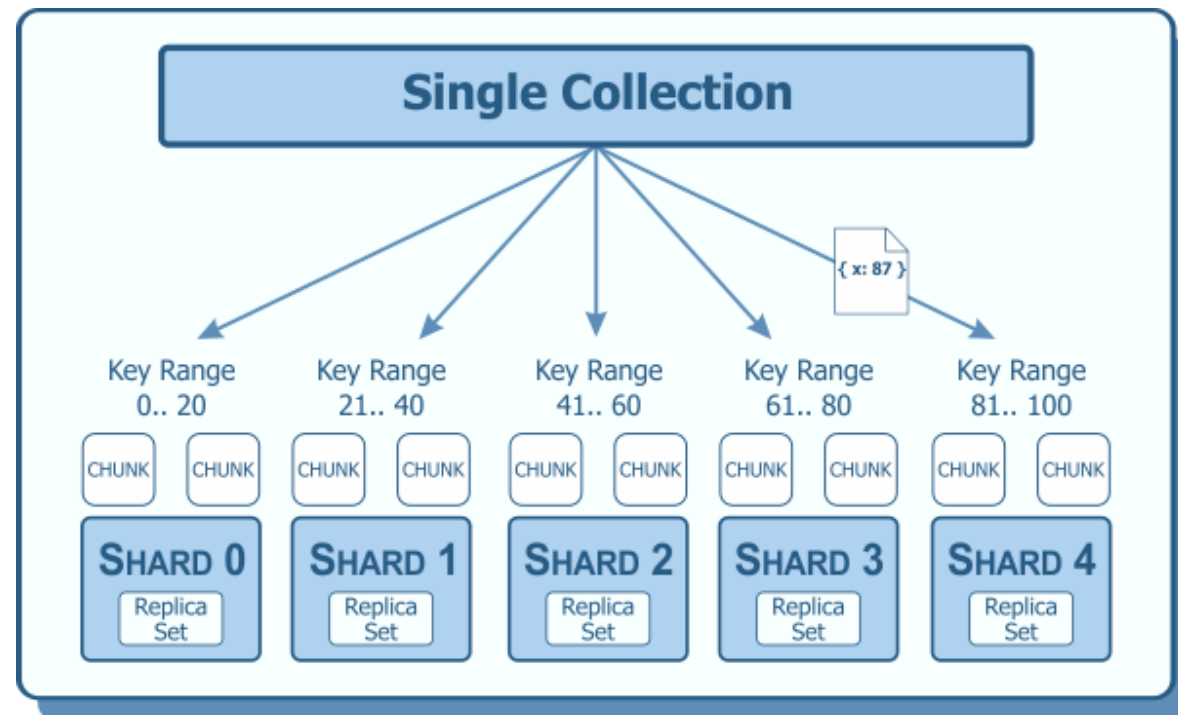
Estrategias para escalar

- **Vertical.-** Se añaden más dispositivos a un sistema de cómputo para mejorar sus capacidades.
- **Horizontal.-** Se forma una malla de computadoras con capacidades de cómputo similares, y mediante algún mecanismo de software se reparte el procesamiento en cada máquina nodo.



Particionamiento

- El *Sharding* se basa en particionar los datos de forma que la información que es frecuentemente accedida a la vez se encuentre en el mismo nodo.
- Además, se deberá tener en cuenta que la carga debe quedar distribuida entre todos los nodos del sistema de forma más o menos homogénea.
- Estos “shards” o fragmentos pueden ser, a su vez, replicados por razones de fiabilidad o balanceo de carga



Estrategias para el particionamiento de datos



Sharding por clave (y variantes)

MongoDB



Un master que sabe dónde colocar los datos

Hadoop, HDFS y Hbase



Distribución vía funciones de hash

Amazon Dynamo

Fiabilidad

Una aplicación de uso intensivo de datos debe ser confiable, escalable y mantenible. Incluso en el caso de un incremento de la carga de trabajo.

La fiabilidad es una característica importante de las aplicaciones de uso intensivo de datos.

La fiabilidad es cuando el sistema debe continuar funcionando correctamente incluso frente a la adversidad.

La aplicación realiza las funciones esperadas por el usuario.

- Puede tolerar errores de usuario o la ejecución de software de maneras inesperadas.
- Su rendimiento es lo suficientemente bueno para el caso de uso requerido bajo cargas esperadas y volumen de datos.
- El sistema evita cualquier acceso no autorizado.
- En el caso de la escalabilidad a medida que un sistema crece en volumen de datos, volumen de tráfico o complejidad, deberían existir formas razonables de hacer frente a ese crecimiento.
- La escalabilidad es la capacidad de un sistema para hacer frente al aumento de la carga.

Modelo de consistencia

- Consistencia estricta
 - Típicamente vía bloqueos (locks)
- Consistencia eventual
- Consistencia ajustable
 - ¡Requiere de mecanismos de resolución de conflictos!

El modelo BASE

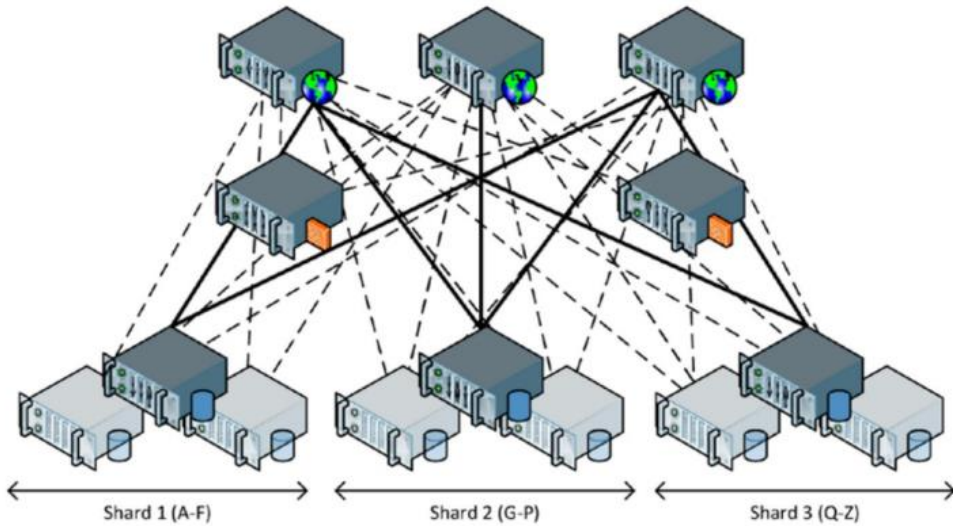
- El modelo BASE toma su nombre de:
 - **Basic Availability.** El sistema funciona incluso cuando alguna parte falla, debido a que el almacenamiento sigue los principios de distribución y replicación.
 - **Soft State.** Los nodos no tienen por qué ser consistentes entre si todo el tiempo.
 - **Eventual Consistency.** La consistencia se produce de forma eventual.
- La siguiente tabla muestra una comparación de las características ACID vs. BASE

ACID	BASE
<ul style="list-style-type: none">- Fuerte coherencia- Aislamiento- Enfocado a commits- Transacciones anidadas- Más conservador- Evolución complicada (esquema)	<ul style="list-style-type: none">- Coherencia débil- Disponibilidad- Mejor esfuerzo- Respuestas aproximadas- Más optimista- Más sencillo y rápido- Evolución más sencilla

	Bases de datos relacionales	Bases de datos NoSQL
Cargas de trabajo óptimas	Las bases de datos relacionales están diseñadas para aplicaciones de procesamiento de transacciones online (OLTP) altamente coherentes y transaccionales, y son buenas para el procesamiento analítico online (OLAP).	Las bases de datos NoSQL están diseñadas para varios patrones de acceso a datos que incluyen aplicaciones de baja latencia. Las bases de datos de búsqueda NoSQL están diseñadas para hacer análisis sobre datos semiestructurados.
Modelo de datos	El modelo relacional normaliza los datos en tablas conformadas por filas y columnas. Un esquema define estrictamente las tablas, las filas, las columnas, los índices, las relaciones entre las tablas y otros elementos de las bases de datos. La base de datos impone la integridad referencial en las relaciones entre tablas.	Las bases de datos NoSQL proporcionan una variedad de modelos de datos, como clave-valor, documentos y grafos, que están optimizados para el rendimiento y la escalabilidad.
Propiedades ACID	Las bases de datos relacionales ofrecen propiedades de atomicidad, coherencia, aislamiento y durabilidad (ACID): <ul style="list-style-type: none">•La atomicidad requiere que una transacción se ejecute por completo o no se ejecute en absoluto.•La coherencia requiere que una vez confirmada una transacción, los datos deban acoplarse al esquema de la base de datos.•El aislamiento requiere que las transacciones simultáneas se ejecuten por separado.•La durabilidad requiere la capacidad de recuperarse de un error inesperado del sistema o de un corte de energía y volver al último estado conocido.	Las bases de datos NoSQL a menudo hacen concesiones al flexibilizar algunas de las propiedades ACID de las bases de datos relacionales para un modelo de datos que puede escalar horizontalmente. Esto hace que las bases de datos NoSQL sean una excelente opción para casos de uso de baja latencia y alto rendimiento que necesitan escalar horizontalmente más allá de las limitaciones de una sola instancia.
Rendimiento	Normalmente, el rendimiento depende del subsistema de disco. Se necesita la optimización de consultas, índices y estructura de tabla para lograr el máximo rendimiento.	El rendimiento por lo general depende del tamaño del clúster de hardware subyacente, la latencia de red y la aplicación que efectúa la llamada.
Escalabilidad	Las bases de datos relacionales generalmente escalan en forma ascendente las capacidades de computación del hardware o la ampliación mediante la adición de réplicas para cargas de trabajo de solo lectura.	Las bases de datos NoSQL normalmente se pueden particionar porque los patrones de acceso son escalables mediante el uso de arquitectura distribuida para aumentar el rendimiento.
API	Solicita almacenar y recuperar datos que están comunicados mediante consultas que se ajustan a un lenguaje de consulta estructurado (SQL). Estas consultas son analizadas y ejecutadas por la base de datos relacional.	Las API basadas en objetos permiten a los desarrolladores almacenar y recuperar fácilmente estructuras de datos. Las claves de partición permiten que las aplicaciones busquen pares de clave-valor, conjuntos de columnas o documentos semiestructurados que contengan atributos y objetos de aplicación serializados.

El teorema CAP

- El teorema CAP o teorema Brewer, dice que en sistemas distribuidos es imposible garantizar a la vez: la consistencia, la disponibilidad y la tolerancia a particiones (Consistency-Availability-Partition Tolerance):
 - **Consistencia:** todos los usuarios de la base de datos ven los mismos datos en cierto instante; al realizar una consulta o inserción siempre se tiene que recibir la misma información, con independencia del nodo o servidor que procese la petición.
 - **Disponibilidad:** ante una falla, la base de datos sigue operacional; que todos los clientes puedan leer y escribir, aunque se haya caído uno de los nodos.
 - **Tolerancia a particiones:** el sistema sigue siendo operacional ante una falla de la red que comunica los segmentos del sistema distribuido (generalmente de forma geográfica). Así que esta condición implica, que el sistema tiene que seguir funcionando, aunque existan fallos o caídas parciales que dividan el sistema.



- Más allá del teorema CAP implementar transacciones ACID en un sistema distribuido es costoso en diferentes sentidos
 - Disponibilidad implica réplicas (típicamente distribuidas físicamente)
 - ¡Para asegurar consistencia estricta cada transacción implica propagar cambios en forma síncrona!

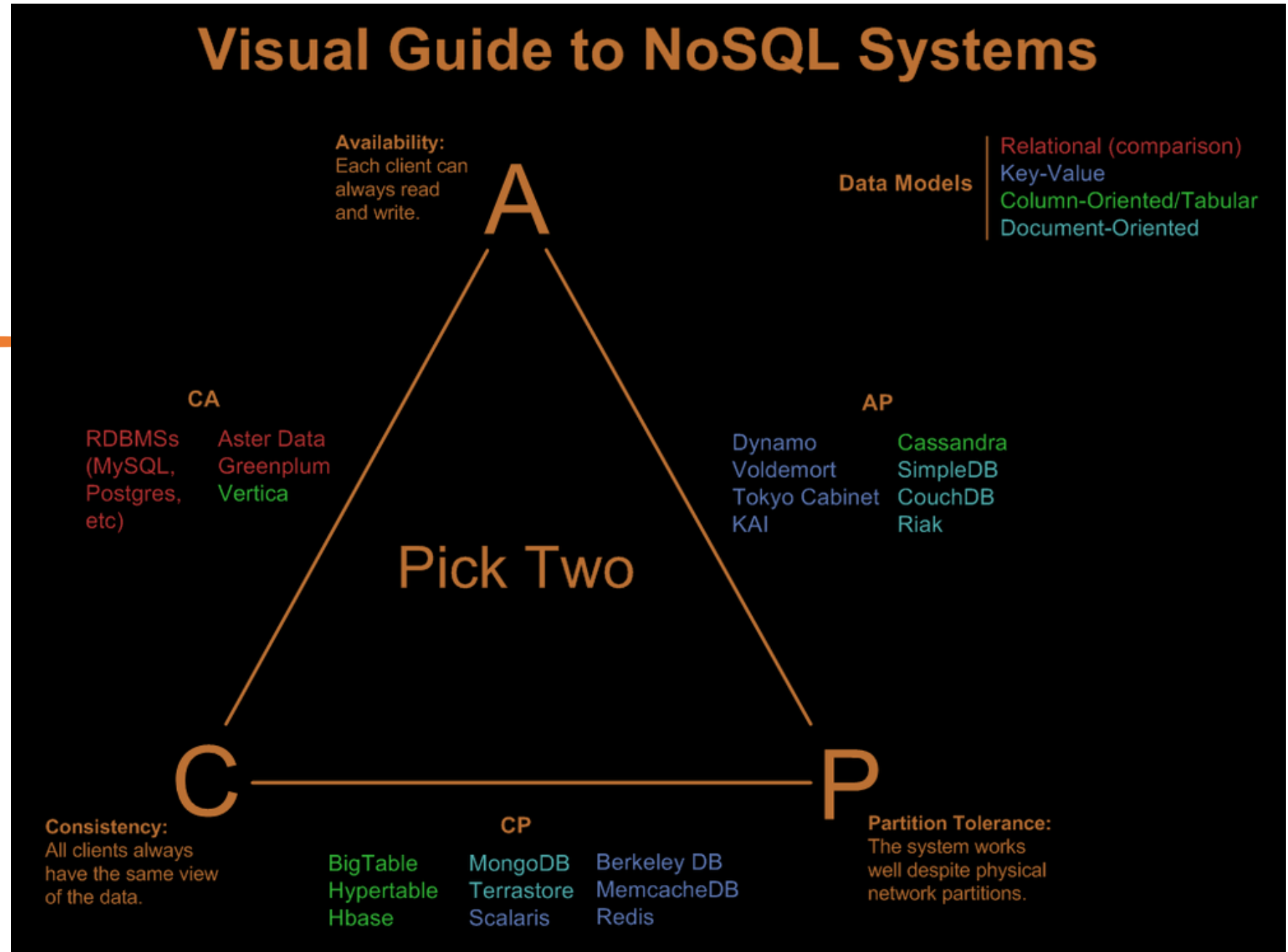
Clasificación de cada base de datos NoSQL según el teorema CAP

Para ser escalables y distribuidas, las bases de datos NoSQL, siguen distintos métodos, por lo que **no todas cumplen los mismos puntos del teorema CAP.**

En la imagen que encabeza el artículo, podemos ver cómo se reparten algunas de las bases de datos según las condiciones que cumplen del teorema CAP.

- **AP:** garantizan disponibilidad y tolerancia a particiones, pero no la consistencia, al menos de forma total. Algunas de ellas consiguen una consistencia parcial a través de la replicación y la verificación.
- **CP:** garantizan consistencia y tolerancia a particiones. Para lograr la consistencia y replicar los datos a través de los nodos, sacrifican la disponibilidad.
- **CA:** garantizan consistencia y disponibilidad, pero tienen problemas con la tolerancia a particiones. Este problema lo suelen gestionar replicando los datos.

Ejemplos

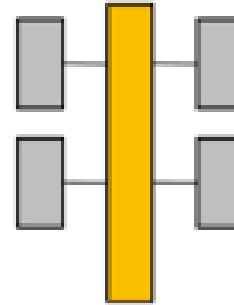


SQL Database

Relational

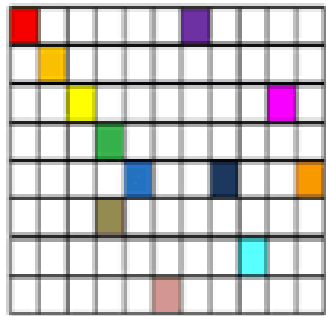


Analytical (OLAP)

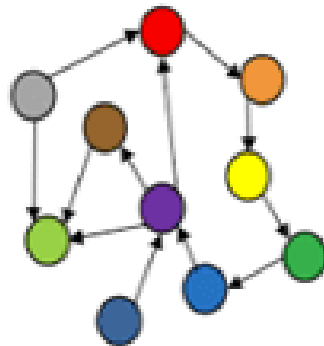


NoSQL Database

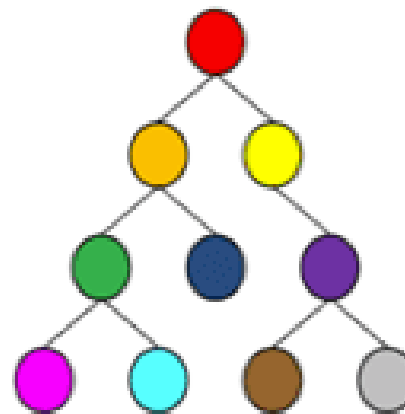
Column-Family



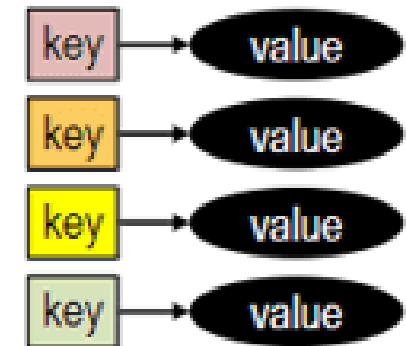
Graph



Document



Key-Value

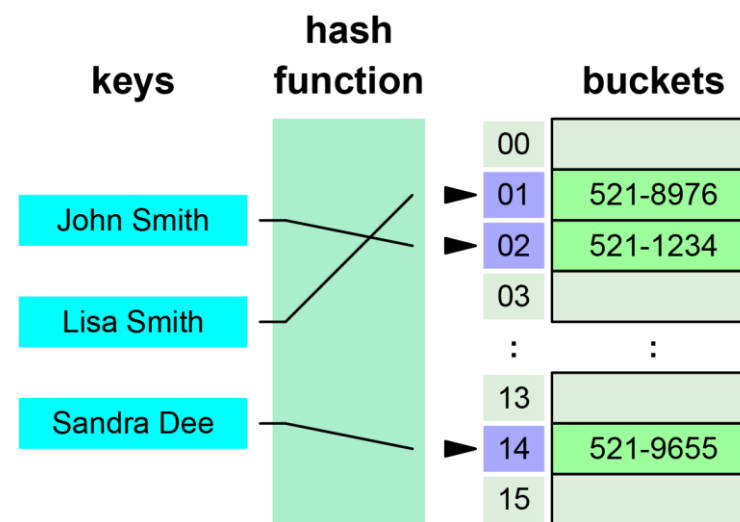
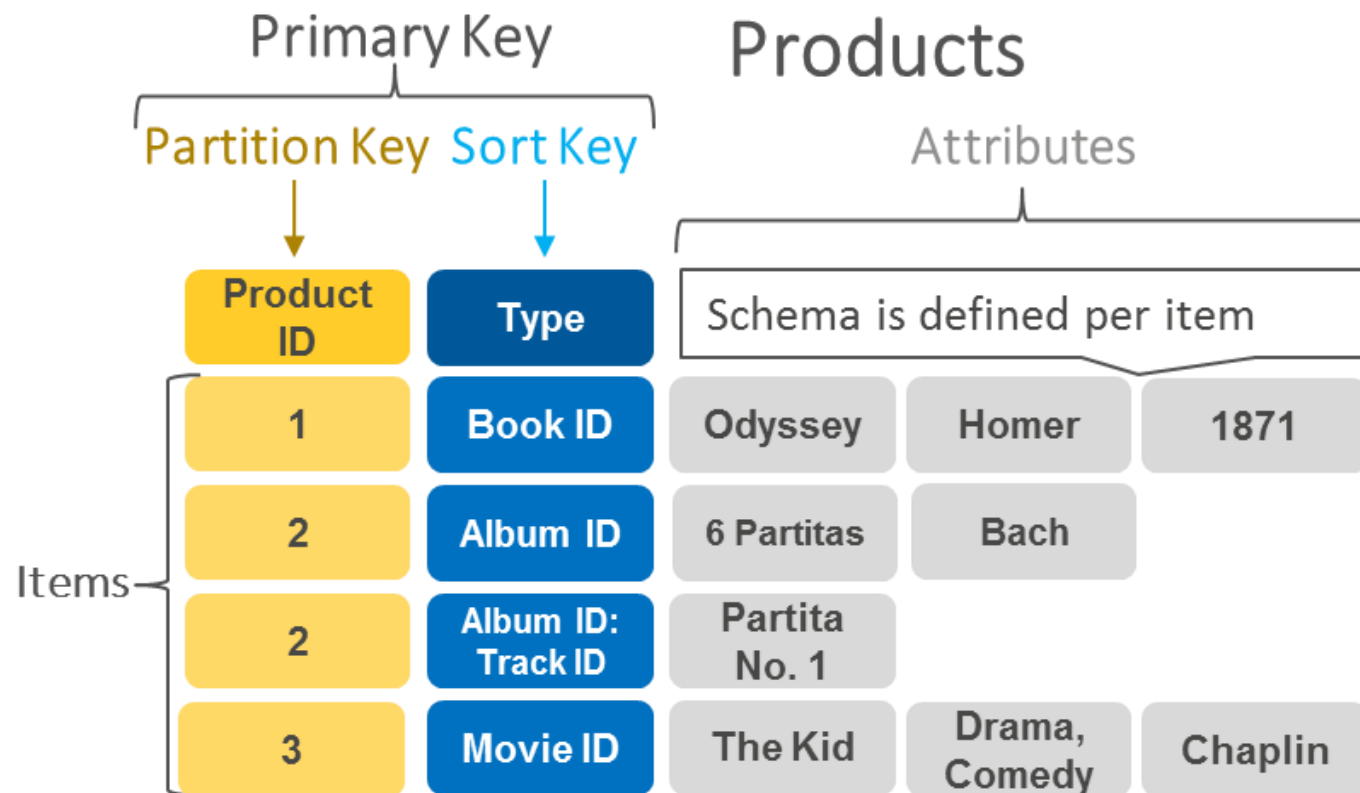


Clasificación de Bases de datos recientes

- Se pueden identificar 4 categorías diferentes, y aunque todas ellas comparten rasgos comunes, cada una se adapta a un uso particular:
 - **Orientadas a documentos.**- son las más versátiles, gestionan datos semi-estructurados o documentos que son almacenados en algún formato estándar como puede ser XML, JSON o BSON. Ejemplos: **MongoDB**, CouchDB, Couchbase Server, MarkLogic.
 - **Orientadas a columnas.**- pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos, funcionan de forma parecida a las bases de datos relacionales, pero almacenando columnas de datos en lugar de registros. Ejemplos: Accumulo, **Cassandra**, HBase, Apache Flink
 - **De clave-valor.**- las más simples de entender, guardan tuplas clave-valor. Ejemplos: Aerospike, Redis, **DynamoDB**, Berkeley DB
 - **En grafo.**- basadas en la teoría de grafos, utilizan nodos y aristas para representar los datos almacenados, y son muy útiles para guardar información en modelos con muchas relaciones, como redes y conexiones sociales. Ejemplos: InfiniteGraph, **Neo4j**
 - **Multimodelo.**- presentan una combinación de dos o más tipos de almacenamiento. Ejemplos ArangoDB, OpenLink Virtuoso, OrientDB, Oracle NoSQL
- <http://nosql-database.org/>
- <https://www.trustradius.com/nosql-databases>

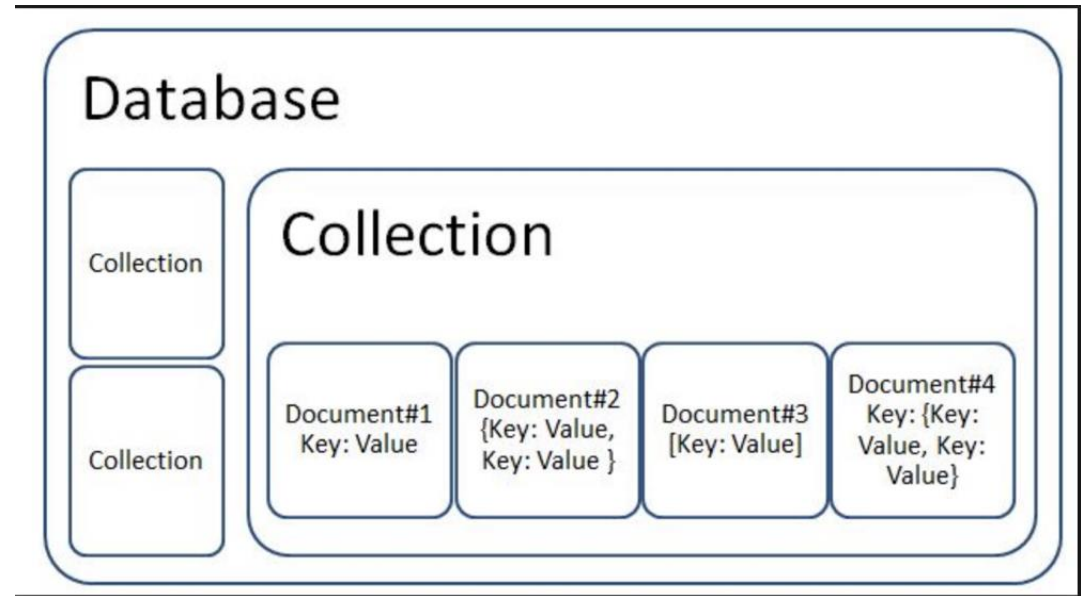
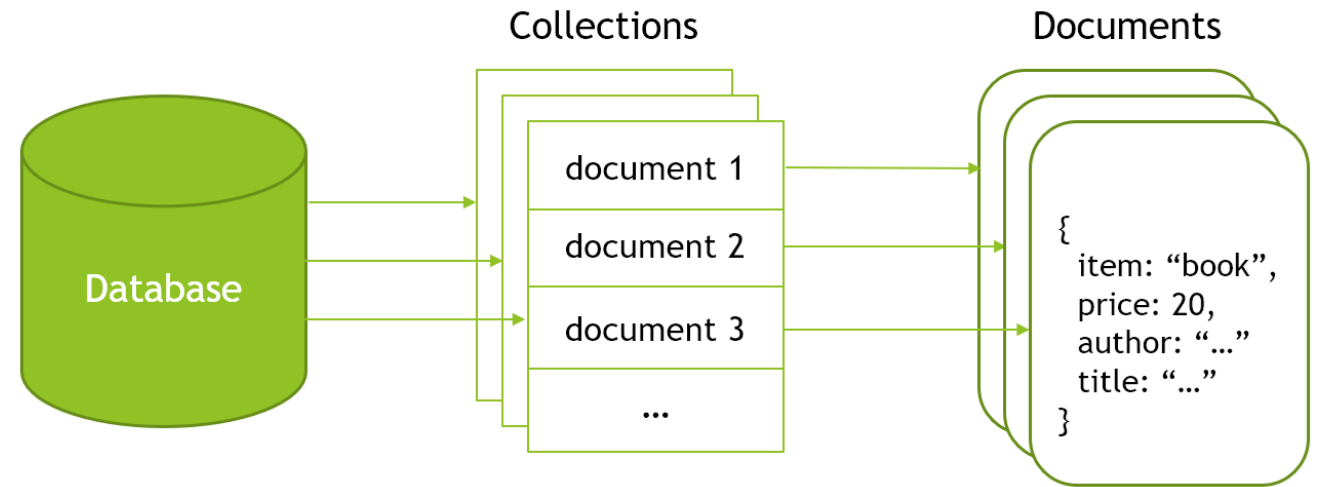
Bases de datos clave – valor

- Son el modelo de base de datos NoSQL más popular, además de ser la más sencilla en cuanto a funcionalidad. En este tipo de sistema, cada elemento está identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario (BLOB). Se caracterizan por ser muy eficientes tanto para las lecturas como para las escrituras.
- Su API es bastante simple y no es más que una variación de 3 operaciones: Put, Get, Delete.
- Su limitante está en que no permiten realmente un modelo de datos, todo lo que guardan es un valor binario.
- Algunos ejemplos de este tipo son Cassandra, BigTable o HBase.



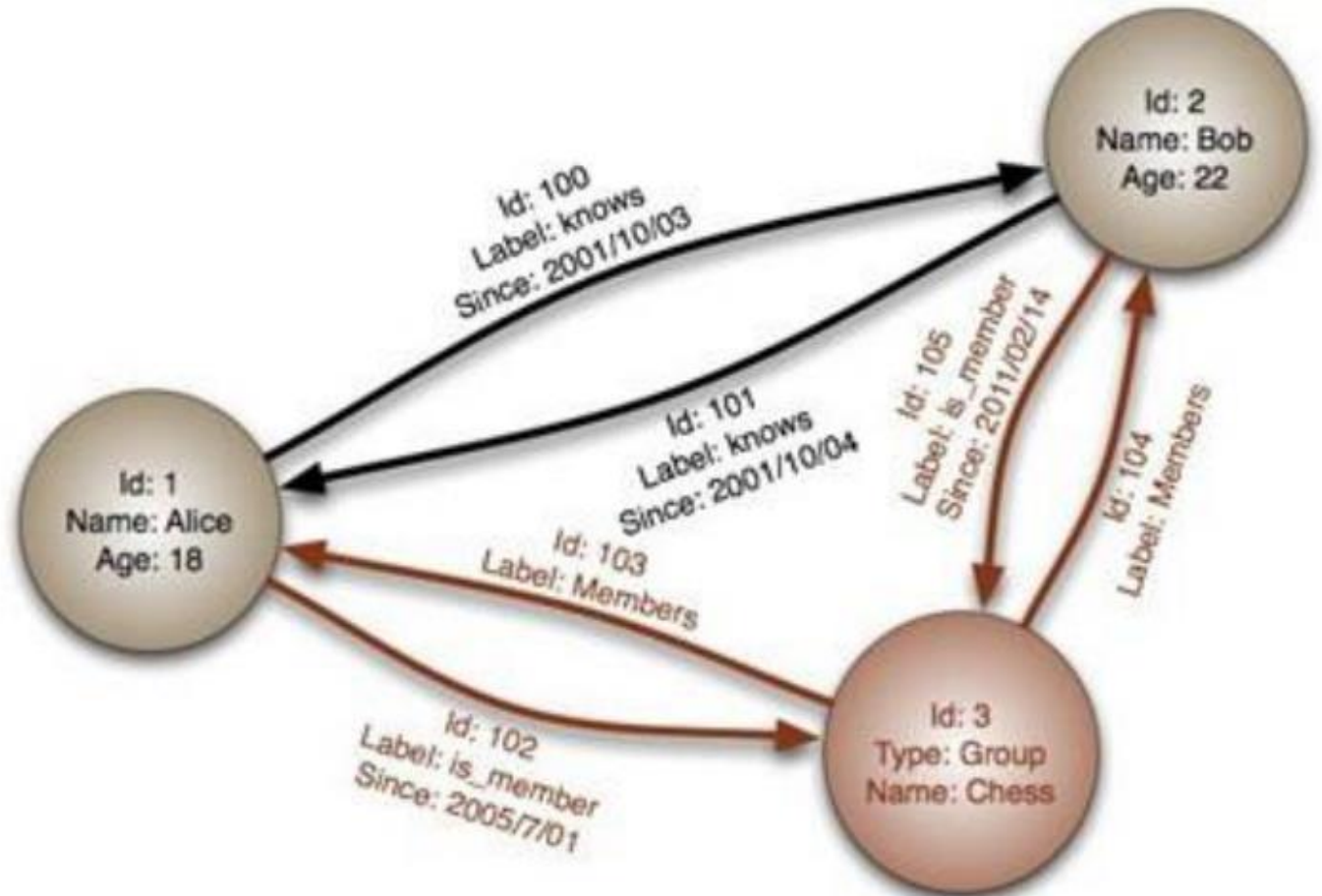
Bases de datos documentales

- Este tipo almacena la información como un documento, generalmente utilizando para ello una estructura simple como JSON o XML y donde se utiliza una clave única para cada registro. Este tipo de implementación permite, además de realizar búsquedas por clave-valor, realizar consultas más avanzadas sobre el contenido del documento.
- Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales.
- Algunos ejemplos de este tipo son MongoDB o CouchDB.



Bases de datos en grafo

- En este tipo de bases de datos, la información se representa como nodos de un grafo y sus relaciones con las aristas del mismo, de manera que se puede hacer uso de la teoría de grafos para recorrerla. Para sacar el máximo rendimiento a este tipo de bases de datos, su estructura debe estar totalmente normalizada, de forma que cada tabla tenga una sola columna y cada relación dos.
- Este tipo de bases de datos ofrece una navegación más eficiente entre relaciones que en un modelo relacional.
- Algunos ejemplos de este tipo son Neo4j, Infinity Graph



Bases de datos columnares

- Como su nombre lo indica, guardan los datos en columnas en lugar de renglones. Se gana mucha velocidad en lecturas, ya que, si se quiere consultar un número reducido de columnas, es muy rápido hacerlo.
- Por otro lado, este paradigma no es muy eficiente para realizar escrituras. Por ello este tipo de soluciones es usado en aplicaciones con un índice bajo de escrituras, pero muchas lecturas.
- Típicamente se usan en ambientes de Data Warehouses y sistemas de Business Intelligence, donde además resultan ideales para calcular datos agregados.

Row-oriented

ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented

Name	ID	Grade	ID	GPA	ID
John	001	Senior	001	4.00	001
Karen	002	Freshman	002	3.67	002
Bill	003	Junior	003	3.33	003

Comparativa entre modelos

- El siguiente es un cuadro comparativo entre los principales tipos de bases de datos NoSQL, atendiendo a las principales características funcionales como puede ser la escalabilidad o el rendimiento.
- El objetivo de la comparativa no es otro que el de encontrar el tipo de base de datos adecuada a cada momento en función de las necesidades del proyecto (Strauch 2011).

Modelo de Datos	Rendimiento	Escalabilidad	Flexibilidad	Complejidad	Funcionalidad
Clave Valor	Alto	Alto	Alto	Bajo	Variable
Columnar	Alto	Alto	Moderado	Bajo	Mínimo
Documental	Alto	Variable	Alto	Bajo	Variable
Grafo	Variable	Variable	Alto	Alto	Teoría de grafos

Comparativa entre modelos

Key Value Stores

Key	Value
CGHScore
RPForm	...

Document Stores

Key	Value
CGHScore	{ "Title": "Chris Gayle Innings..." "UserId": "nrules" "Tags": ["IPL", "Cricket", "RCB"] "Body": "CG at his best..." }
RPForm	{ "Title": "Ricky P Innings..." "UserId": "nrules" "Tags": ["IPL", "Cricket", "MI"] "Body": "RP not his best..." } ...

Column Family Stores

Key	Value
CGHScore
RPOOForm	...

Comments	UserID	Title
WOW Awe...		

Comments	UserID	Title
Improvise...		

Ventajas de NoSQL

Se ejecutan en máquinas con pocos recursos: Estos sistemas, a diferencia de los sistemas basados en SQL, no requieren de mucha computación, por lo que se pueden montar en máquinas de costo reducido.

Escalabilidad horizontal: Para mejorar el rendimiento de estos sistemas, se añaden más nodos, con la única operación de indicar al sistema cuáles son los nodos que están disponibles.

Pueden manejar gran cantidad de datos: Esto es debido a que utiliza una estructura distribuida, en muchos casos mediante tablas Hash.

Desventajas

No están lo suficientemente maduros para algunas empresas

Limitaciones de Inteligencia de Negocios

La falta de experiencia

Problemas de compatibilidad

THE FUTURE OF THE DATABASE

1960s

First Computerized Database Models

Hierarchical Model (IMS)



Network Model (CODASYL)



1970s

The Dawn of the Database

- The relational model and its language SQL emerge
- The disruptive model causes the demise of other models

1970 E.F. Codd Writes a Paper on the Relational Database Model



Single Instance Relational Database

System R by IBM (SQL)
Ingres (QUEL)



Entity Relational Database

ORACLE

1st commercially available RDBMS

IBM DB2

SAP Sybase
Informix



Object Oriented Database



Distributed SQL Data Warehouse

Teradata



Dawn of the Internet

1990s

Technology Shifts

- Data explodes with the Internet age
- Single server SQL databases run into resource problems
- Business Intelligence and Analytics move out of transactional database

BUSINESS INTELLIGENCE

ANALYTICS



Application Layer



New Distributed SQL Data Warehouse

2000s

2000s

New Players Emerge

- Data variety, velocity and volume increase
- New analytics SQL databases are introduced
- NoSQL databases fill the gap for processing unstructured data
- Hadoop gains traction for analyzing petabytes of data

Today

Databases Adapt and Evolve

- Businesses require real-time analytics on operational data
- Scale-up SQL proves too costly, but scale-out removes resource constraint
- Scale-out provides real time analytics with high volume transactions
- Google and Clustrix are pioneers in this space

The Future

Businesses Advance with Database Innovations

- Single node SQL gets replaced by scale-out SQL
- Data warehouse type analytics will become available in real-time database
- Businesses gain a significant edge and increased agility

Winning Database Platforms

NOSQL DATABASE

DISTRIBUTED SQL

HADOOP

NOSQL
Google BigTable
CouchDB
MongoDB
Cassandra
Redis

Distributed SQL
Clustrix
NuoDB
VoltDB
Google Spanner

MPP
+MASSIVELY PARALLEL PROCESSING
Clustrix adds MPP
SAP HANA In-memory

REAL-TIME ANALYTICS

SQL HIVE

HADOOP
Netezza
Microsoft
Aster
Oracle
SAP
IBM
Vertica

+ COLUMNAR
REPLACED?

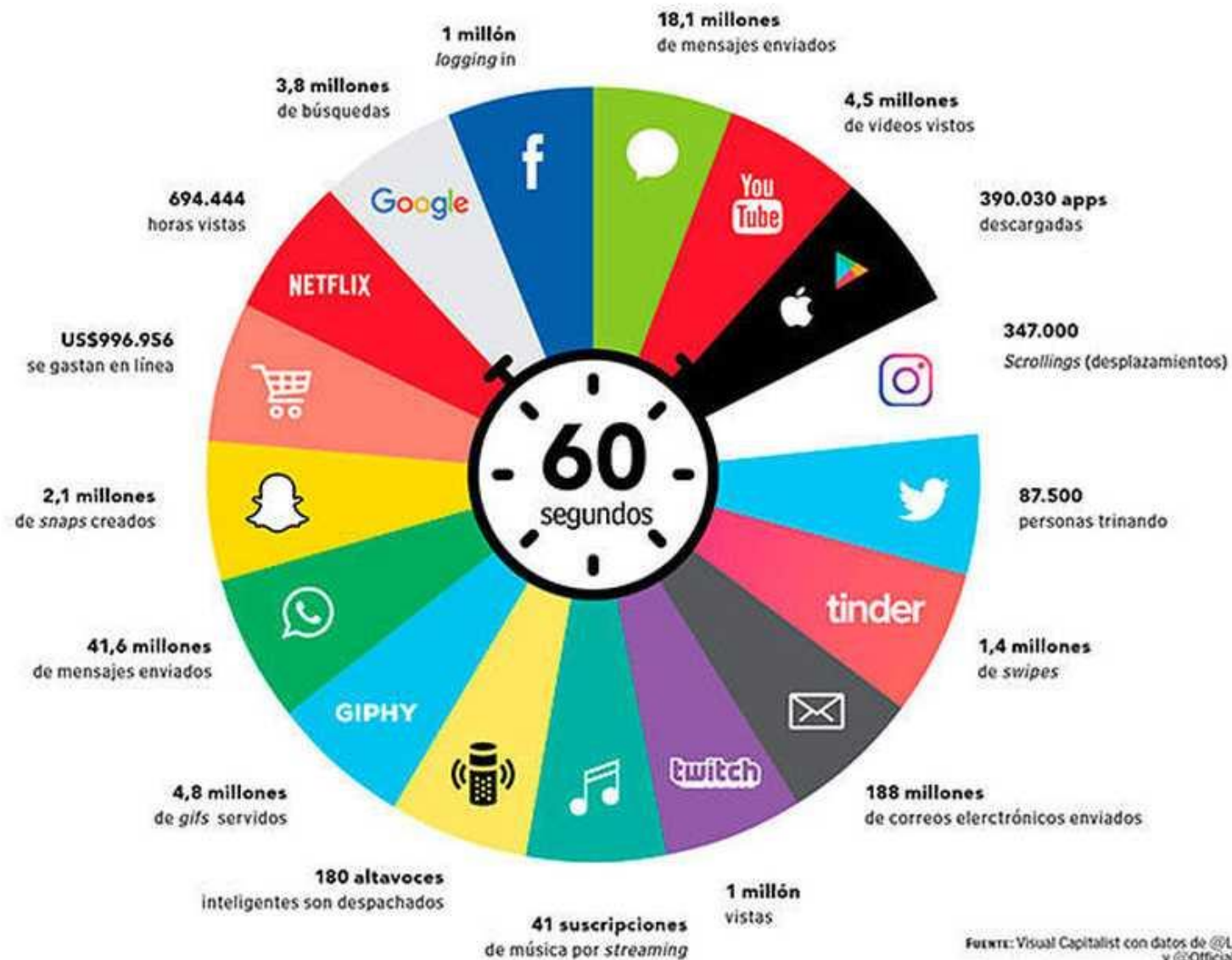
ALL ANALYTICS?

Big Data

Big Data tiene como objetivo resolver problemas antiguos y nuevos de manera más eficiente pero aplicado a extensos volúmenes de datos no usados hasta el momento

Genera valor a un negocio a partir del almacenamiento y procesamiento de cantidades muy grandes de información digital que no puede ser analizada con técnicas tradicionales de computación

En internet, en un minuto pasa esto:



FUENTE: Visual Capitalist con datos de @LoriLewis y @OfficiallyChadd

Las tres 'v' del Big Data

Volumen: Se considera un volumen grande a partir de Petabytes (1.000.000 GB)

Velocidad: Frecuencia a la que se genera los datos / Tiempo de análisis de los datos

Variedad: Datos estructurados, semi-estructurados, no estructurados
¿Otras Vs?: ¿Veracidad? ¿Valor?

El valor de los datos

El uso intensivo de los datos ha pasado a ser el petróleo de muchas compañías

El nuevo enfoque es almacenar cualquier tipo de dato, por irrelevante que pueda parecer, para su posterior análisis

- Clics de ratón en la página web del negocio
- Vibración del motor del coche
- Movimiento del acelerómetro del smartphone

Permite crear modelos para responder preguntas complejas, mostrar percepciones contraintuitivas y aprender resultados únicos

El valor de los datos

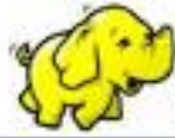


Challenged by: Data Volume, Velocity, Variety

Componentes de una plataforma Big Data

Las organizaciones han atacado esta problemática desde diferentes ángulos. Todas esas montañas de información han generado un costo potencial al no descubrir el gran valor asociado. Desde luego, el ángulo correcto que actualmente tiene el liderazgo en términos de popularidad para analizar enormes cantidades de información es la plataforma de código abierto Hadoop.

Hadoop está inspirado en el proyecto de Google File System(GFS) y en el paradigma de programación MapReduce, el cual consiste en dividir en dos tareas (mapper – reducer) para manipular los datos distribuidos a nodos de un clúster logrando un alto paralelismo en el procesamiento. Hadoop está compuesto de tres piezas: Hadoop Distributed File System (HDFS), Hadoop MapReduce y Hadoop Common.



Apache Hadoop Ecosystem



Ambari

Provisioning, Managing and Monitoring Hadoop Clusters



Sqoop
Data Exchange



Flume
Log Collector



Zookeeper
Coordination



Oozie
Workflow



Pig
Scripting



Mahout
Machine Learning

R Connectors
Statistics



Hive
SQL Query



Hbase
Columnar Store



HDFS

Hadoop Distributed File System

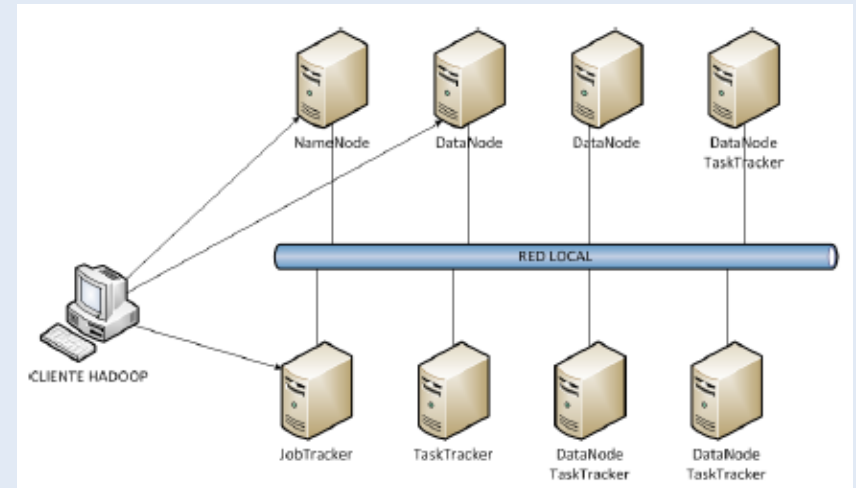
YARN Map Reduce v2

Distributed Processing Framework



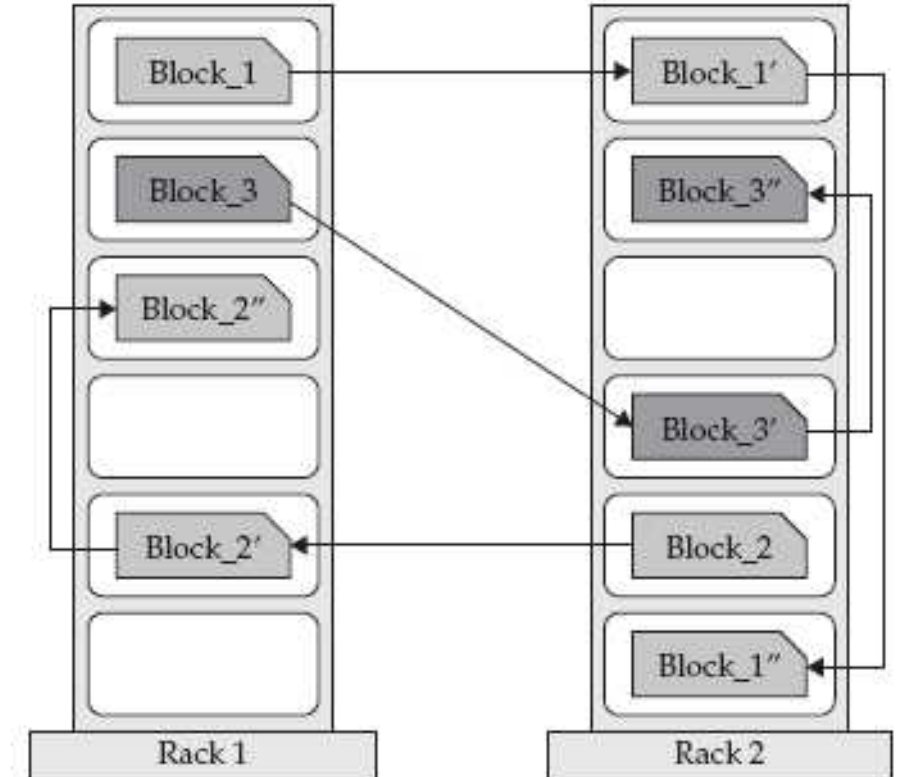
Hadoop

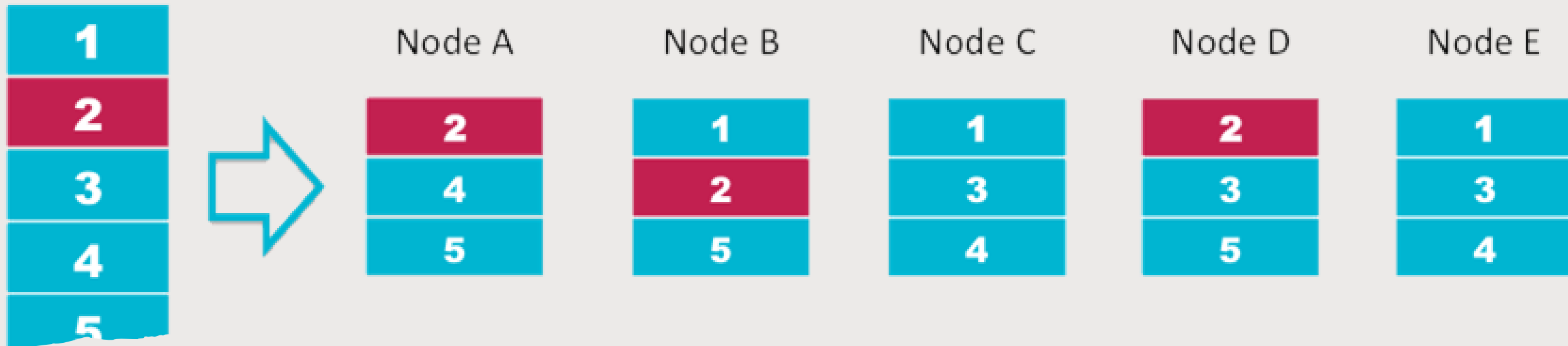
- La idea de contar con un gran número de nodos y cada uno con posibilidad de fallas, significa que en todo momento algún componente siempre no será funcional.
- Por lo tanto, la detección de fallas y rápida recuperación automática de un sistema distribuido es un objetivo central de la arquitectura de HDFS



Hadoop Distributed File System(HDFS)

- Los datos en el clúster de Hadoop son divididos en pequeñas piezas llamadas bloques y distribuidas a través del clúster; de esta manera, las funciones map y reduce pueden ser ejecutadas en pequeños subconjuntos y esto provee de la escalabilidad necesaria para el procesamiento de grandes volúmenes.
- La siguiente figura ejemplifica como los bloques de datos son escritos hacia HDFS. Cada bloque es almacenado tres veces y al menos un bloque se almacena en un diferente rack para lograr redundancia.
- Un tamaño de bloque típico utilizado por HDFS es de 64 MB. Por lo tanto, un archivo de HDFS está segmentado en trozos de 64 MB, y tal vez, cada trozo va a residir en un DataNode diferente.



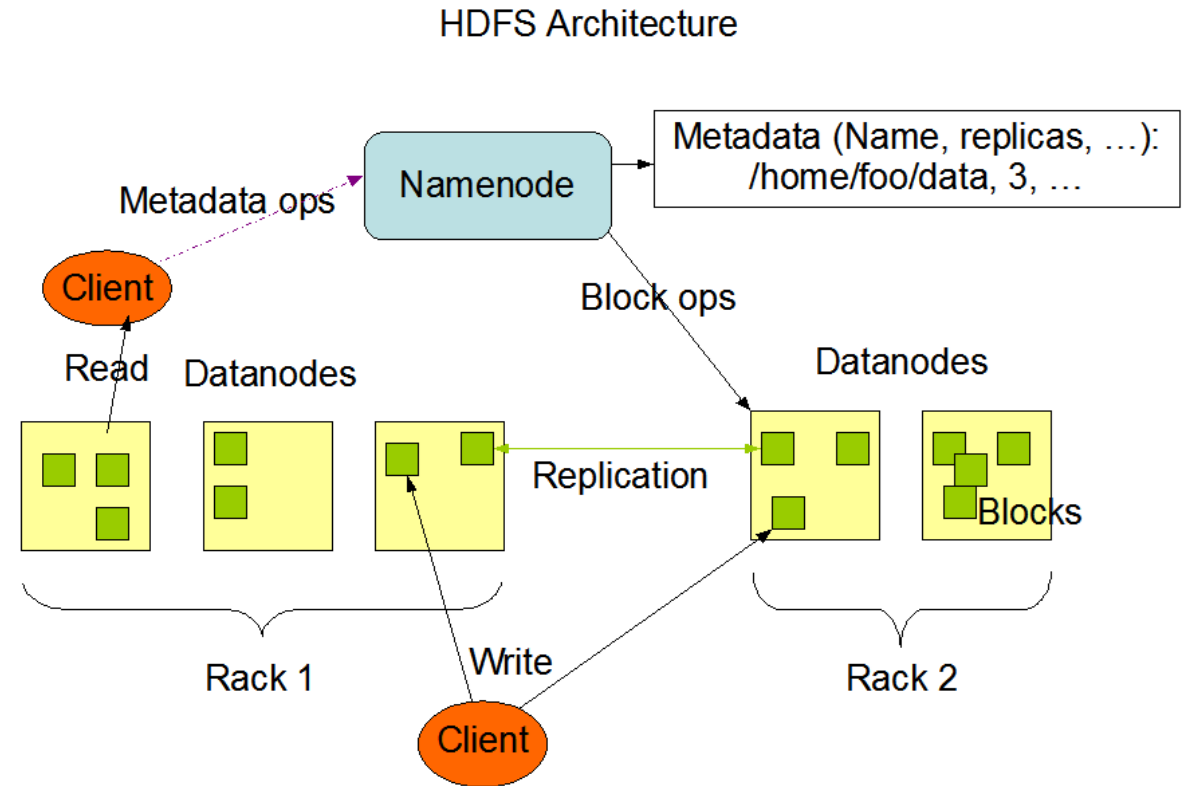


La replicación de datos

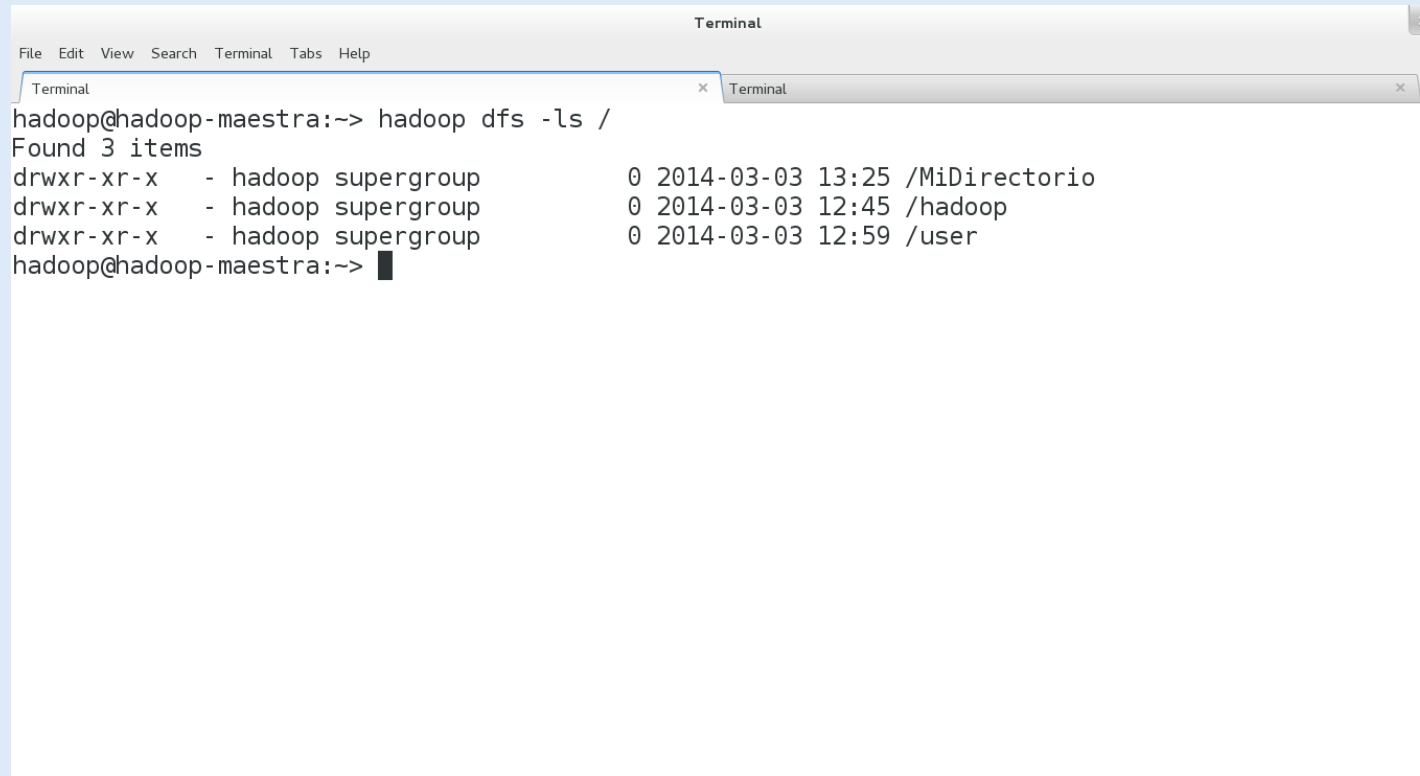
- HDFS está diseñado para almacenar de forma fiable archivos muy grandes a través de las máquinas en un clúster grande.
- Almacena cada archivo como una secuencia de bloques; Todos los bloques de un archivo, excepto el último bloque son del mismo tamaño. Los bloques de un archivo se replican para tolerancia a fallos. El tamaño del bloque y el factor de replicación son configurables por archivo. Una aplicación puede especificar el número de réplicas de un archivo. El factor de replicación se puede especificar en el momento de creación de archivos y se puede cambiar posteriormente.

NameNode y DataNodes

- HDFS tiene una arquitectura maestro / esclavo.
- Un clúster HDFS consta de una solo Nodo de Nombres (NameNode) o servidor maestro que gestiona el espacio de nombres del sistema de archivos y que regula el acceso a los archivos.
- Además, puede constar de una serie de Nodos de Datos (DataNodes), donde se almacenan los archivos y llevan a cabo los procesamientos.
- Internamente, un archivo se divide en uno o más bloques y estos bloques se almacenan en un conjunto de DataNodes. Los DataNodes también llevan a cabo la creación de bloques, eliminación, y la replicación según las instrucciones del NameNode o máquina maestra.
- El NameNode ejecuta operaciones de espacio de nombres del sistema de archivos como abrir, cerrar y cambiar el nombre de archivos y directorios, los cuales se ejecutan en los nodos de datos



El espacio de nombres de archivos del HDFS

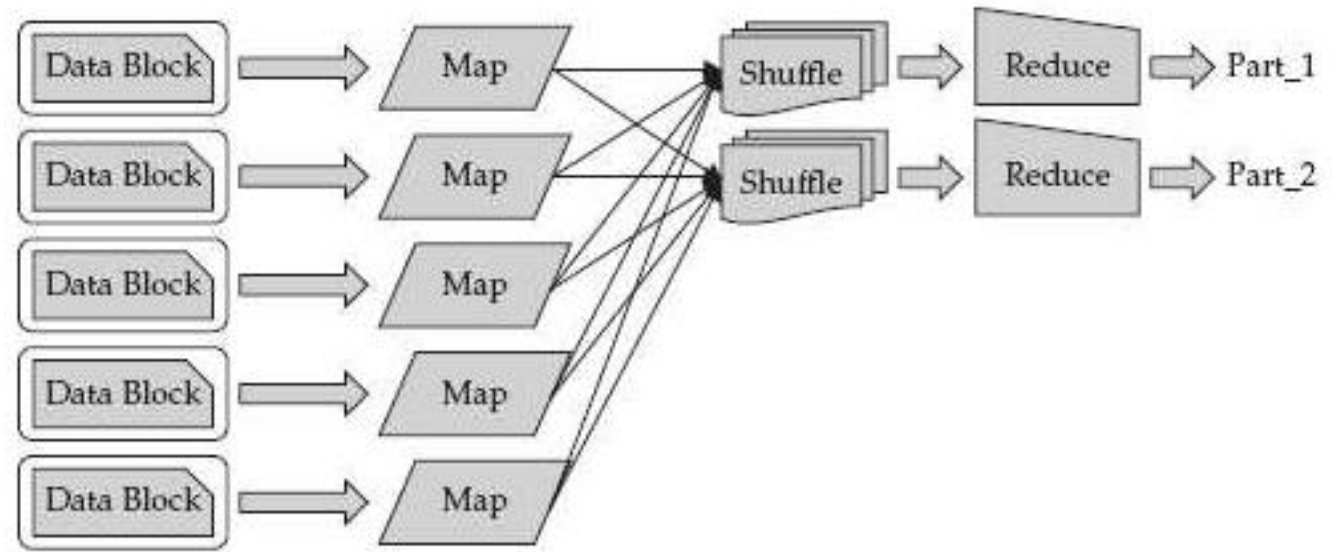
A screenshot of a terminal window titled "Terminal" with a menu bar (File, Edit, View, Search, Terminal, Tabs, Help). The terminal shows the command "hadoop dfs -ls /" being executed. The output indicates "Found 3 items" and lists three entries: "/MiDirectorio", "/hadoop", and "/user", each with permissions "drwxr-xr-x", owner "hadoop", group "supergroup", size "0", and timestamps from 2014-03-03. The prompt "hadoop@hadoop-maestra:~>" is visible at the bottom.

```
hadoop@hadoop-maestra:~> hadoop dfs -ls /
Found 3 items
drwxr-xr-x  - hadoop supergroup      0 2014-03-03 13:25 /MiDirectorio
drwxr-xr-x  - hadoop supergroup      0 2014-03-03 12:45 /hadoop
drwxr-xr-x  - hadoop supergroup      0 2014-03-03 12:59 /user
hadoop@hadoop-maestra:~>
```

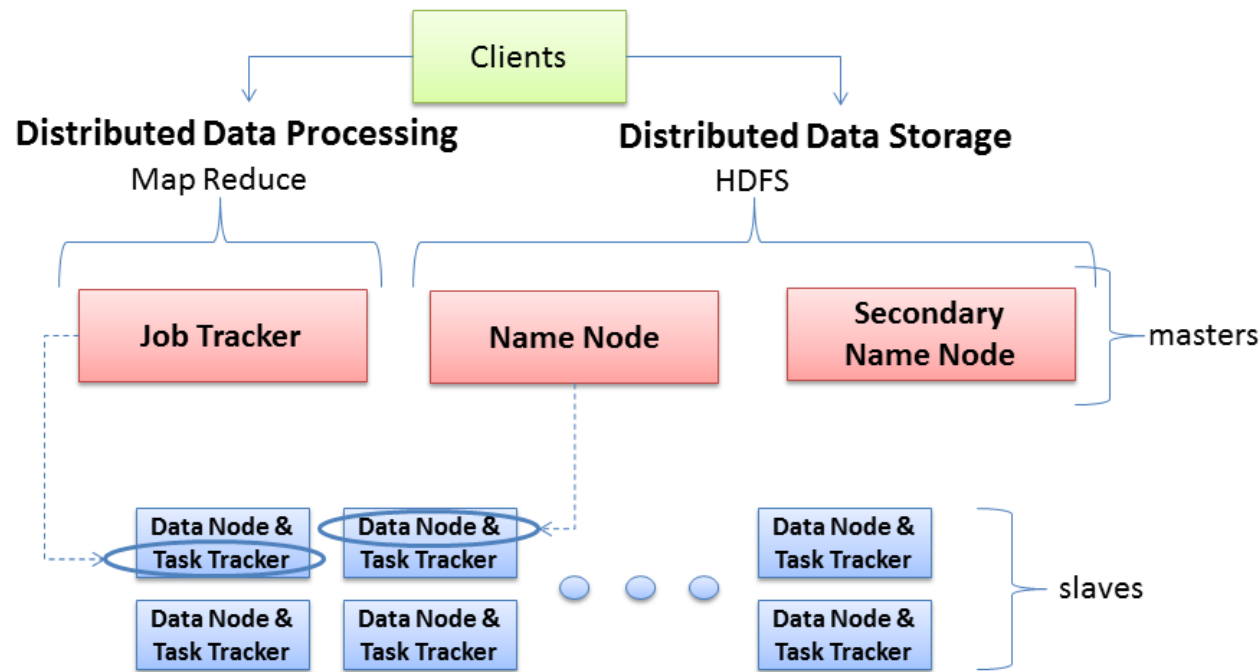
- HDFS ofrece una organización de archivos jerárquico tradicional.
- Un usuario o una aplicación pueden crear directorios y archivos de almacenamiento dentro de estos directorios.
- La jerarquía del espacio de nombres del sistema de archivos es similar a la mayoría de los otros sistemas de archivos existentes; uno puede crear y eliminar archivos, mover un archivo de un directorio a otro, o cambiar el nombre de un archivo.

Hadoop MapReduce

- **MapReduce** es el núcleo de Hadoop. El término MapReduce en realidad se refiere a dos procesos separados que Hadoop ejecuta.
- El primer proceso es **map**, el cual toma un conjunto de datos y lo convierte en otro conjunto, donde los elementos individuales son separados en tuplas (pares de llave/valor).
- El proceso **reduce** obtiene la salida de **map** como datos de entrada y combina las tuplas en un conjunto más pequeño de las mismas.
- Una fase intermedia es la denominada **Shuffle** la cual obtiene las tuplas del proceso **map** y determina que nodo procesará estos datos dirigiendo la salida a una tarea **reduce** en específico.



Job Tracker

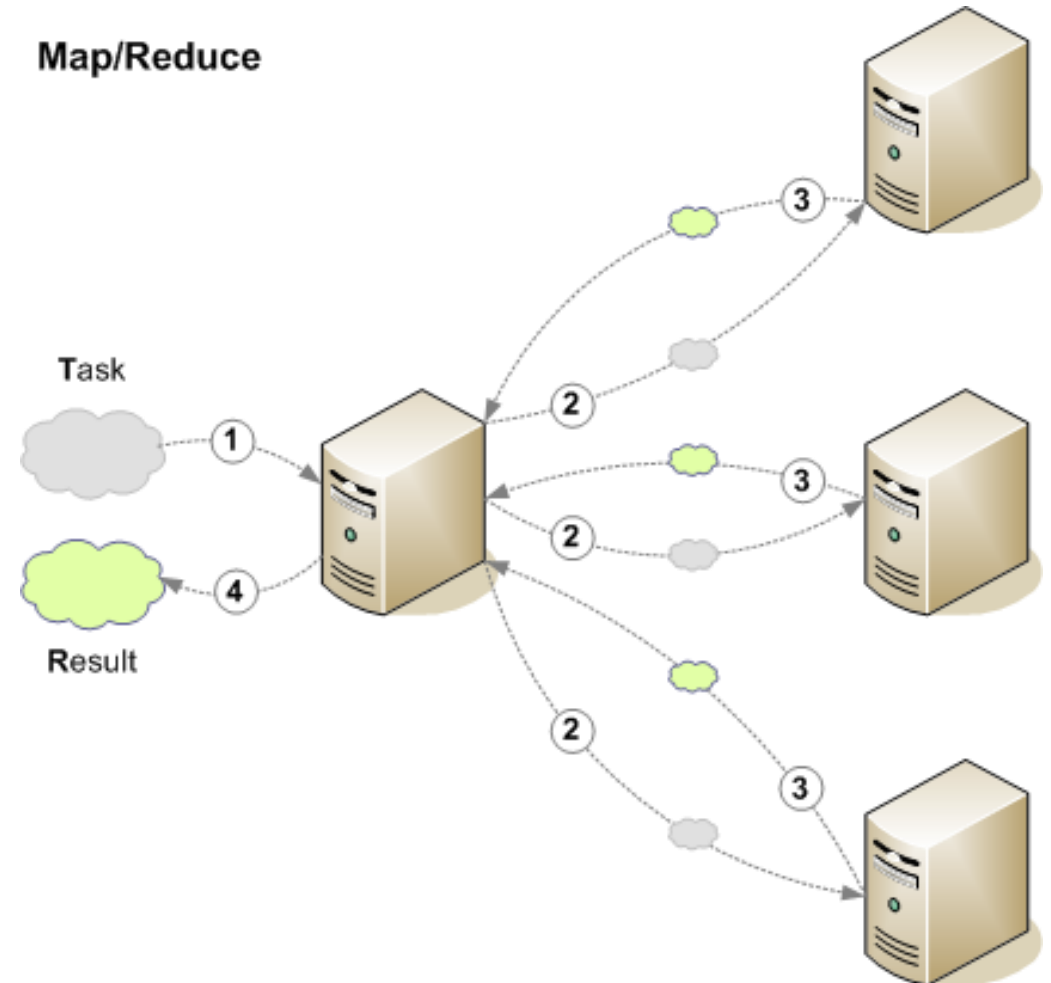


- El marco de referencia MapReduce consta de un solo monitor de trabajo JobTracker maestro y múltiples monitores de tareas un nodo esclavo TaskTracker. El maestro es el responsable de la programación de tareas, se compone de puestos de trabajo en los nodos de datos o esclavos, el seguimiento de ellos y la reejecución de las tareas fallidas. Los esclavos ejecutan las tareas según las instrucciones del maestro.

Ejemplo WordCount

- Ejemplo de aplicación del marco de referencia MapReduce para generación de un análisis de frecuencia de palabras en un texto (WordCount)
- WordCount es una sencilla aplicación que cuenta el número de apariciones de cada palabra en un conjunto de datos de entrada. Es decir, se realiza un análisis de frecuencia de cada palabra en el texto.

Map/Reduce



The Overall MapReduce Word Count Process

