

3

SISTEMAS DE ALMACENAMIENTO

Al inicio del libro caracterizamos el dato de múltiples maneras: tipología, formato, tamaño, latencia, etc. En este capítulo nos basaremos en su estado para abordar los distintos mecanismos de almacenamiento que podemos utilizar para persistirlo y acceder a él. La Figura 3-1 muestra los tres estados fundamentales en los que se puede encontrar un dato. Los **datos en reposo** (*data at rest*) son aquellos que se encuentran fuera del acceso habitual. Son, en principio, inmutables, y típicamente se almacenan en sistemas de cintas (*backup*), redes de almacenamiento local (SAN, *Storage Area Network*) o en la nube. Desde el momento en que su acceso es esporádico, los requerimientos de estos sistemas en términos de número de operaciones por segundo (IOPS, *Input Output Per Second*), rendimiento y latencia no son muy exigentes.

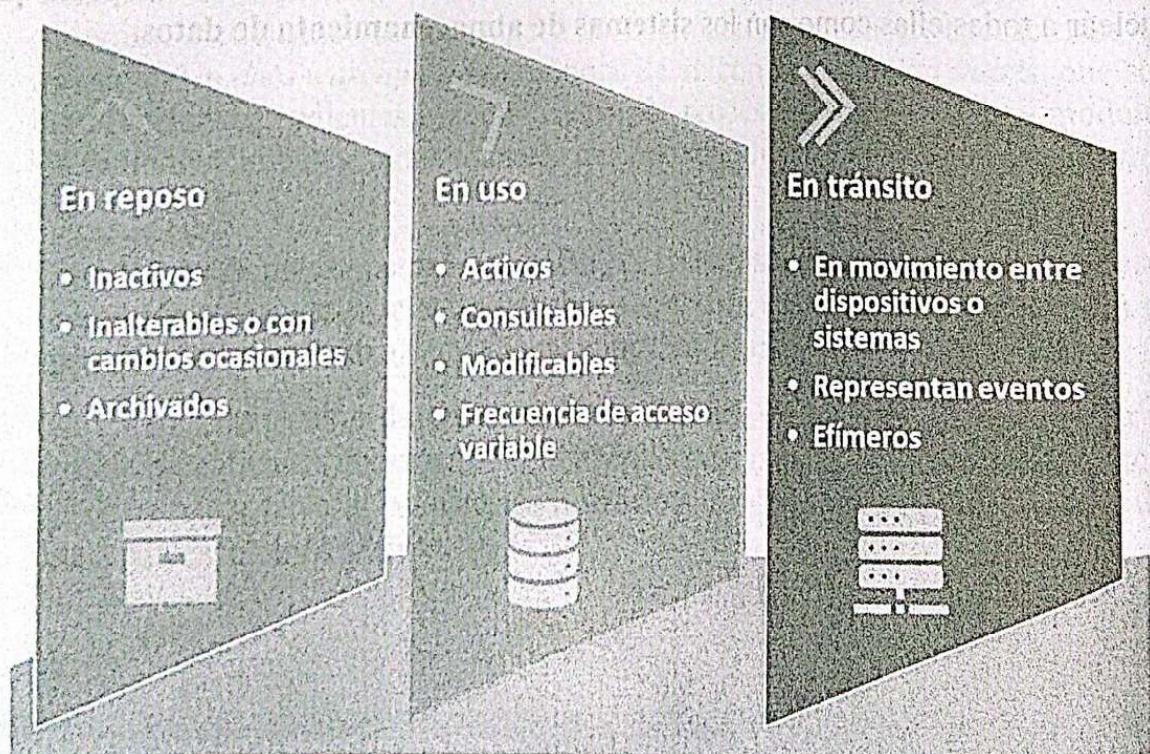


Figura 3-1. Estados del dato.

Estos sistemas se emplean para guardar datos históricos fuera del rango temporal de acceso establecido en la organización, aunque pueden ser consultados bajo solicitud. También contienen datos de acuerdo con lo dictado por los distintos marcos regulatorios que afectan al negocio. Aunque no nos detendremos en este tipo de datos, suelen representar un volumen importante, con unos costes de almacenamiento más bajos que en otros estados.

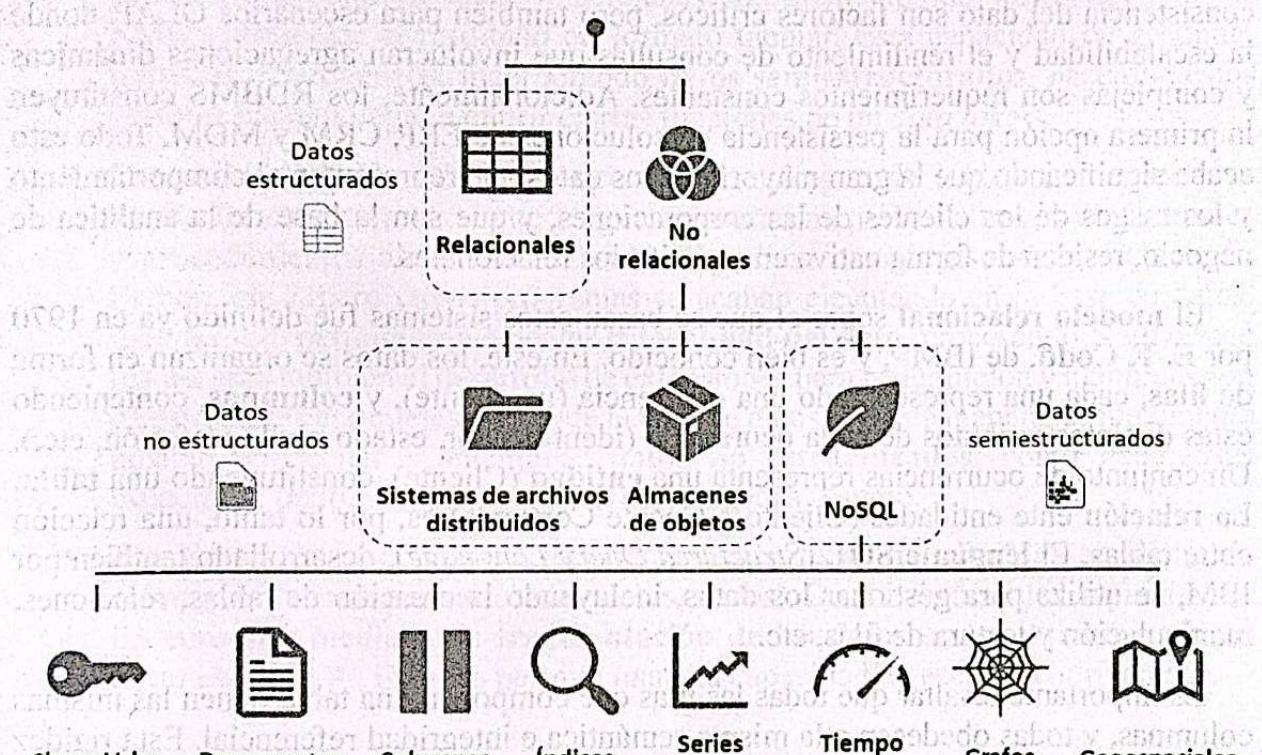


Figura 3-2. Taxonomía de mecanismos de almacenamiento para datos en uso.

Nos vamos a centrar en los **datos en uso** (*data in use*). Estos son aquellos que, residiendo en un tipo de almacenamiento, están siendo usados por los sistemas del negocio. Es decir, son datos activos, que con una frecuencia variable van del disco a la memoria RAM, y de ahí son procesados por la CPU. Dependiendo del tipo de sistema y aplicación que los gestiona, soportarán unos tipos de operaciones (lectura, escritura, actualización, borrado) con más frecuencia que otras. La Figura 3-2 muestra una forma de clasificar los distintos mecanismos de almacenamiento que podemos utilizar para este tipo de datos, y que vamos a tratar a continuación poniendo el foco en el soporte de cargas analíticas. Están basados en diferentes tecnologías, cada una con una serie de características que la hacen más adecuada para una casuística u otra.

Los **datos en tránsito** (*data in motion*), no menos importantes en el contexto de *Big Data*, los abordaremos en un capítulo posterior.

3.1 BASES DE DATOS RELACIONALES

Los sistemas de gestión de bases de datos relacionales (**RDBMS**, *Relational Database Management System*) han estado en el centro de los sistemas de negocio durante décadas, evolucionando con ellos y haciendo evolucionar a la tecnología. Han demostrado ser la mejor opción para las cargas OLTP, donde la disponibilidad y la consistencia del dato son factores críticos, pero también para escenarios OLAP, donde la escalabilidad y el rendimiento de consultas que involucran agregaciones dinámicas y complejas son requerimientos constantes. Adicionalmente, los RDBMS constituyen la primera opción para la persistencia en soluciones de ERP, CRM y MDM. Todo esto acaba significando que la gran mayoría de los datos que representan el comportamiento y los rasgos de los clientes de las corporaciones, y que son la base de la analítica de negocio, residen de forma nativa en repositorios relacionales.

El **modelo relacional** sobre el que se basan estos sistemas fue definido ya en 1970 por **E. F. Codd**, de IBM⁵³, y es bien conocido. En este, los datos se organizan en forma de **filas**, cada una representando una ocurrencia (un cliente), y **columnas**, conteniendo estas distintos atributos de cada ocurrencia (identificador, estado civil, profesión, etc.). Un conjunto de ocurrencias representa una **entidad** (Cliente), constituyendo una **tabla**. La **relación** entre entidades (Cliente – Cuenta Corriente) es, por lo tanto, una relación entre tablas. El **lenguaje SQL** (*Structured Query Language*), desarrollado también por IBM, se utiliza para gestionar los datos, incluyendo la creación de tablas, relaciones, manipulación y lectura de filas, etc.

Es importante resaltar que todas las filas que componen una tabla tienen las mismas columnas, y todas obedecen a la misma semántica e integridad referencial. Esta rigidez del modelo relacional es lo que propició en su momento la aparición de las **bases de datos NoSQL** (*Non-SQL*), mucho más flexibles al permitir una libertad de esquema (*schemaless*) tanto en la estructura de las entidades como en sus relaciones. Sin embargo, esta flexibilidad sólo es posible a costa de relajar ciertas características que pueden resultar fundamentales, como el soporte de transacciones en los RDBMS según las propiedades ACID.

3.1.1 Gestión de cargas analíticas

Los RDBMS han evolucionado mucho desde sus inicios. Además de ser la tecnología de referencia para los sistemas de *data warehouse*, incluyendo los *data marts*, y los ODS, hoy en día incorporan una serie de características muy interesantes, especialmente para la gestión de cargas analíticas. Por ejemplo:

- **Poliglotía.** Un RDBMS moderno es híbrido, en el sentido en que en un único entorno podemos soportar modelos relacionales (tabulares) y otros formatos más flexibles. Aquí se incluye el soporte nativo para **XML** (*eXtensible Markup*

53 <https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf>

Language)⁵⁴, **JSON** (*JavaScript Object Notation*)⁵⁵ o **RDF** (*Resource Description Framework*), aunque este no está estandarizado como en los dos casos anteriores. Esto no solo implica la persistencia de este tipo de documentos, que quedan almacenados e indexados como una columna más en una tabla, sino también el soporte de los lenguajes de interrogación correspondientes (XPath/XQuery, SPARQL, etc.) y su combinación con SQL. Por ejemplo, es posible lanzar una consulta que combine datos relacionales con elementos de varios documentos XML y presentar el resultado en formato tabular. Esta capacidad de gestionar formatos heterogéneos, incorporando **datos semiestructurados**, hace que estos gestores puedan llegar a cubrir ciertas funciones de un *data lake*⁵⁶.

- **Extensibilidad.** Un RDBMS soporta el despliegue de lógica de negocio, ya sea de forma nativa mediante lenguajes procedimentales (a través de funciones y procedimientos almacenados), o bien mediante lenguajes externos (Java, C, Python, etc.), pero cuyos programas se acaban ejecutando en la base de datos. Esto no solo permite implementar la **localidad del dato**, procesándolo allí donde reside, sino también el desarrollo de extensiones para la definición y manipulación de estructuras complejas dentro de la base de datos. De esta manera, es igualmente posible almacenar y manipular **datos no estructurados**, como geometrías espaciales, texto, modelos predictivos, imágenes o audio.
- **Rendimiento.** Los RDBMS soportan **arquitecturas distribuidas** en forma de clúster compuesto por varios servidores (nodos). De esta forma pueden parallelizar las consultas mediante la **fragmentación de los datos** en los diferentes nodos (*data sharding*)⁵⁷. Esto les permite una alta capacidad de escalado horizontal⁵⁸.

54 El término *NoSQL* es comúnmente asociado a bases de datos que no utilizan SQL para su manejo. Sin embargo, el uso de SQL en bases de datos NoSQL es cada vez más común, ya que las bases de datos NoSQL están comenzando a adoptar tecnologías de optimización y mejoramiento de rendimiento que permiten la ejecución de consultas SQL. Algunas bases de datos NoSQL, como MongoDB y Couchbase, tienen soporte para consultas SQL.

55 <https://en.wikipedia.org/wiki/JSON>

56 Muchos utilizan también el término NoSQL para referirse a los gestores relativamente modernos, pero ahora con otro significado para las siglas: *Not only SQL*.

57 También se conoce como arquitectura sin compartición (*shared-nothing*).

58 Esto significa añadir más servidores, distribuyéndose la carga de trabajo a lo largo de los diferentes nodos. Hacerlo verticalmente, por el contrario, implica añadir más recursos (CPU, RAM, disco) a un mismo servidor para asumir una carga que va en aumento. El crecimiento horizontal es, en principio, preferible para un entorno de *Big Data*, ya que no está limitado por la capacidad de un único servidor y ofrece una mayor tolerancia a fallos. En cualquier caso, la aplicación sustentada debe estar diseñada de forma acorde.

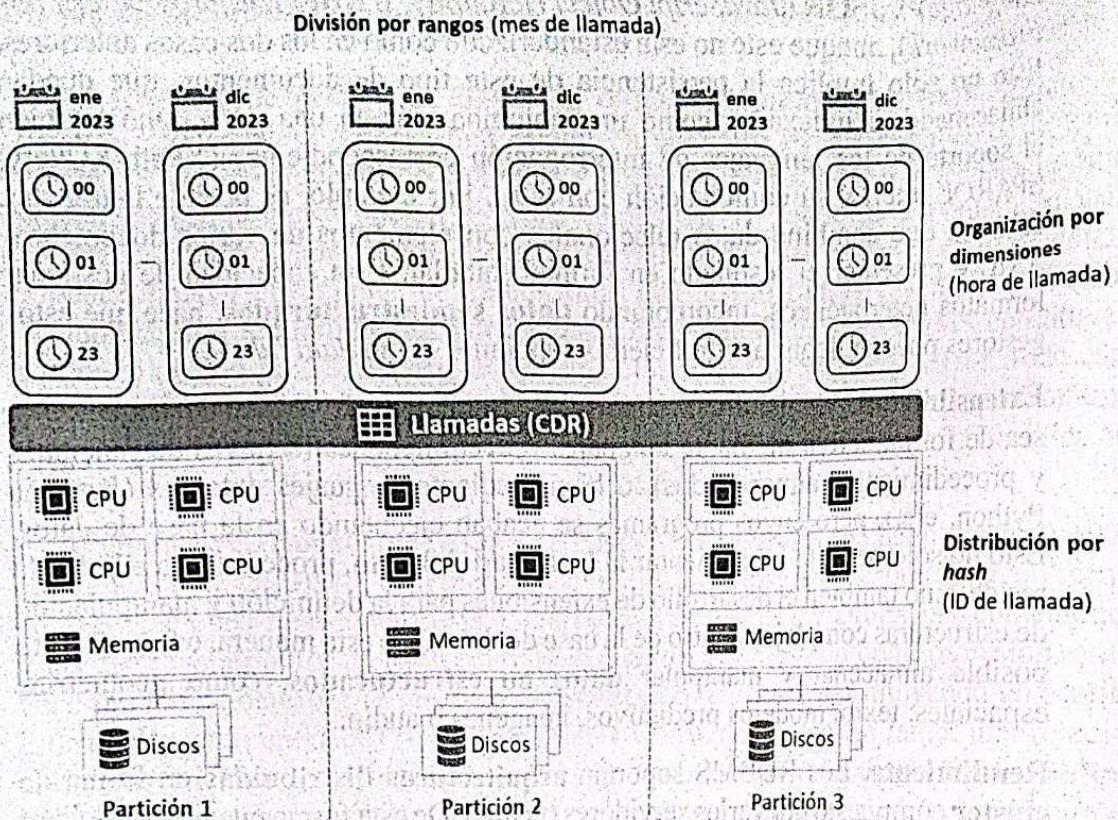


Figura 3-3. Organización de datos a tres niveles en un gestor relacional.

Nota. Extraído de *Five reasons for in-database data mining [Figura]*, por L. Fandiño, V., 2020, Medium (<https://medium.com/towards-data-science/five-reasons-for-in-database-data-mining-84e9cf8e084a>).

La Figura 3-3 muestra un ejemplo de organización de datos a tres niveles. Una única tabla conteniendo atributos de llamadas telefónicas (CDR, *Call Detail Record*) ha sido fragmentada en tres particiones, cada una soportada por un nodo. El criterio para la fragmentación ha sido el identificador de llamada, una clave sin ninguna lógica de negocio que permite una distribución uniforme de los registros, de forma que ninguna partición esté ociosa cuando se accede a los datos. Adicionalmente, en cada partición la tabla ha sido dividida por rangos, utilizando el mes de la llamada para ello. Cada mes se encuentra en un objeto de almacenamiento diferente, de forma que la eliminación y adición de nuevos meses es muy rápida. Finalmente, la tabla se organiza e indexa en bloques por dimensiones, según la hora de la llamada en este caso. Esta organización permite que una misma consulta SQL se subdivida en múltiples partes que se ejecutan de forma paralela, tanto dentro de una misma partición (*intrapartitioning*) como entre particiones (*interpartitioning*). Arquitecturas de este tipo son las que permiten consultar modelos en estrella, donde las tablas de hechos contienen billones de registros y están rodeadas de múltiples tablas de dimensiones, resultando en operaciones de lectura con un gran número de tablas involucradas (*joins*).

- **Organización.** En sistemas OLTP, caracterizados por tener un acceso aleatorio a los registros, recuperándose todas las columnas, estos se organizan en disco a modo de filas consecutivas. Sin embargo, las cargas analíticas recuperan grandes bloques de registros, pero accediendo ahora a menos columnas. Para estos casos, ciertos gestores soportan una **organización en columnas** de los datos. En este caso, los bloques de disco contienen columnas en formato comprimido, con lo que el acceso llega a ser varios órdenes de magnitud más rápido, eliminándose además la necesidad de definir y gestionar índices o vistas materializadas⁵⁹.
- **Federación.** La federación es la base de la virtualización de datos. Esto quiere decir que con una única sentencia SQL, tanto de lectura como de escritura, podemos acceder a datos distribuidos en distintas fuentes heterogéneas (una tabla en otro gestor, un fichero Parquet en Hadoop o S3, una hora de cálculo Excel, un documento JSON en MongoDB, etc.), como si sus datos estuvieran juntos. Esto permite combinar datos estructurados y semiestructurados sin necesidad de movimiento físico, aunque a costa de cierta penalización en los tiempos de acceso.

Otras características, como la gestión y segregación de cargas de trabajo (*workload management*), el control de acceso granular a nivel de registro y columna o la alta disponibilidad con configuraciones **activo-activo** o **activo-pasivo**⁶⁰, son características que hacen de las RDBMS una opción siempre a tener en cuenta.

59 Una vista materializada contiene el resultado de una consulta SQL, normalmente resultado de un proceso de agregación de datos costoso de resolver. Sin embargo, al contrario que en una vista SQL ordinaria, dicho resultado se persiste en disco, con lo que el acceso es mucho más rápido. Este tipo de vistas son transparentes al usuario, que sigue escribiendo las consultas pensando en las tablas base que integran la vista. Además, cada vez que estas varían, el gestor recalcula la vista, manteniendo así la sincronía.

60 Una configuración activo-pasivo significa que todas las operaciones de escritura van dirigidas a un nodo primario, que a su vez va replicando los cambios a una serie de nodos secundarios. En el caso de caída del primario, uno de los secundarios asume su papel. Mientras tanto, los nodos secundarios pueden soportar operaciones solo de lectura. Por el contrario, en una configuración activo-activo todos los nodos soportan tanto escritura como lectura, si bien en este caso hay que tener un control de que no existan conflictos en el acceso, manteniendo las propiedades ACID. Estas configuraciones son la base de la alta disponibilidad y recuperación ante fallos (HADR, *High Availability and Disaster Recovery*).

3.1.2 Escenarios e inconvenientes

La Tabla 3-1 resume las características de las RDBMS para cargas analíticas.

Escenarios	Inconvenientes
<ul style="list-style-type: none"> Los datos se organizan a modo de entidades y relaciones El modelo de datos es consistente, siendo relativamente estático Aunque no mayoritarios, hay que almacenar datos semiestructurados Es necesario asegurar la consistencia del dato (propiedades ACID) La proporción de datos históricos a analizar es mayoritaria La carga es principalmente por lotes El acceso debe ser estándar (ODBC/JDBC, SQL) Hay necesidad de soportar accesos mixtos (OLTP & OLAP) Hay recursos para permitir un crecimiento horizontal 	<ul style="list-style-type: none"> Soporte escaso para tipos de datos y relaciones complejas La modificación del modelo de datos (<i>schema evolution</i>) es complicada Ineficientes cuando dominan datos no estructurados e inconsistentes en cuanto a organización Mayor esfuerzo de mantenimiento (particiones, índices, agregados, etc.) Las restricciones ACID pueden afectar al rendimiento Poco adaptadas para cargas no planificadas y en tiempo real Coste elevado para su crecimiento, ya sea en infraestructura local o como servicio en la nube Percepción de tecnología obsoleta

Tabla 3-1. Ventajas e inconvenientes de las bases de datos relacionales para *Big Data*.

3.1.3 Software y soluciones para *data warehouse*

A la hora de evaluar las distintas ofertas para el almacenamiento en sistemas de *data warehouse*, nos podemos encontrar, por un lado, con los motores tradicionales de bases de datos, con versiones más o menos especializadas para cargas analíticas. **Oracle Database**, **IBM Db2 Warehouse**, **Microsoft SQL Server** o **Teradata Vantage** dominan la porción del mercado con licencias comerciales, mientras que **PostgreSQL**, **Maria DB** o **MySQL** lo hacen desde un planteamiento de código abierto. Todos ellos pueden ser instalados localmente o en infraestructura en la nube (IaaS) a través de distintas opciones, incluyendo el despliegue en contenedores.

Otro grupo interesante es el de los **sistemas de *data warehouse* nativos en la nube**, con un modelo que estaría entre un servicio de plataforma (PaaS) y de *software* (SaaS). La Tabla 3-2 muestra las 4 soluciones más extendidas en esta categoría con algunas de sus características.

	Amazon Redshift	Azure Synapse	Google BigQuery	Snowflake Data Platform
Proveedor	AWS	Azure	Google	AWS, Azure, Google
Arquitectura		Sin compartición		
Motor SQL	PostgreSQL	SQL Server	Dremel	Propietario
Transacciones			ACID	
Organización			Columnas	
Elasticidad (serverless)	Manual y automática	Manual y automática	Automática	Automática
Formatos consultables	CSV, Parquet, JSON, ORC, Avro	CSV, Parquet, JSON	CSV, Parquet, JSON, ORC, Avro	CSV, Parquet, JSON, XML, ORC, Avro

Tabla 3-2. Comparación entre distintas soluciones de *data warehouse* en la nube.

Una característica interesante de estas soluciones es que explotan el modelo *serverless*. Es decir, la asignación de recursos al clúster (nodos, CPU, memoria, disco, etc.) es realizada por el proveedor del servicio, de acuerdo con la carga de trabajo de cada momento, y de forma transparente al usuario. Este autoescalamiento permite que el servicio sea elástico, desatendido y con un coste ajustado al consumo en cada instante, separando el almacenamiento del cómputo. Adicionalmente, se trata de arquitecturas sin compartición, donde cada nodo tiene sus propios recursos y una porción de los datos⁶¹.

Un factor común a todas ellas es que intentan expandir sus capacidades más allá del puro modelo relacional con el fin de asumir algunas funcionalidades propias del *data lake*. Por ejemplo, el soporte de **tablas externas** permite la consulta mediante SQL y el análisis datos existentes en un almacenamiento externo, sin necesidad de cargarlos primero en la base de datos.

3.2 SISTEMAS DE ARCHIVOS DISTRIBUIDOS

El **sistema de archivos (FS, File System)** es uno de los elementos centrales de cualquier sistema operativo. Es el encargado de almacenar y recuperar los datos en forma de archivos, incluyendo su nomenclatura, organización jerárquica (directorios), así como el manejo de los distintos metadatos (fecha de creación, última modificación, longitud, permisos de acceso, etc.). Hablamos de **sistema de archivos distribuido (DFS, Distributed File System)** cuando esta gestión se realiza a lo largo de múltiples servidores (nodos) conectados por red, permitiendo un acceso transparente, compartido y controlado a los distintos archivos que gestiona. La posibilidad de ir añadiendo nodos

61 Algunas soluciones, como Snowflake, realizan compartición de disco a nivel de todo el clúster, aunque cada nodo acaba almacenando una porción de los datos localmente.

al sistema facilita un escalado horizontal prácticamente ilimitado. Además, como los archivos son distribuidos y replicados entre los distintos nodos⁶², la tolerancia a fallos es muy alta, pudiéndose parallelizar el acceso.

Un sistema de archivos distribuidos y un gestor de bases de datos relacional son, por lo tanto, dos aproximaciones a la hora de almacenar los datos de una forma segura y escalable. Aunque el segundo puede proporcionar de forma inherente muchas funcionalidades, el primero presenta a su vez una serie de ventajas: flexibilidad en cuanto a la estructura de los datos, empleo de formatos no propietarios, sencillez a la hora de programar y manipular los archivos o, especialmente, menores costes en infraestructura, por citar algunas.

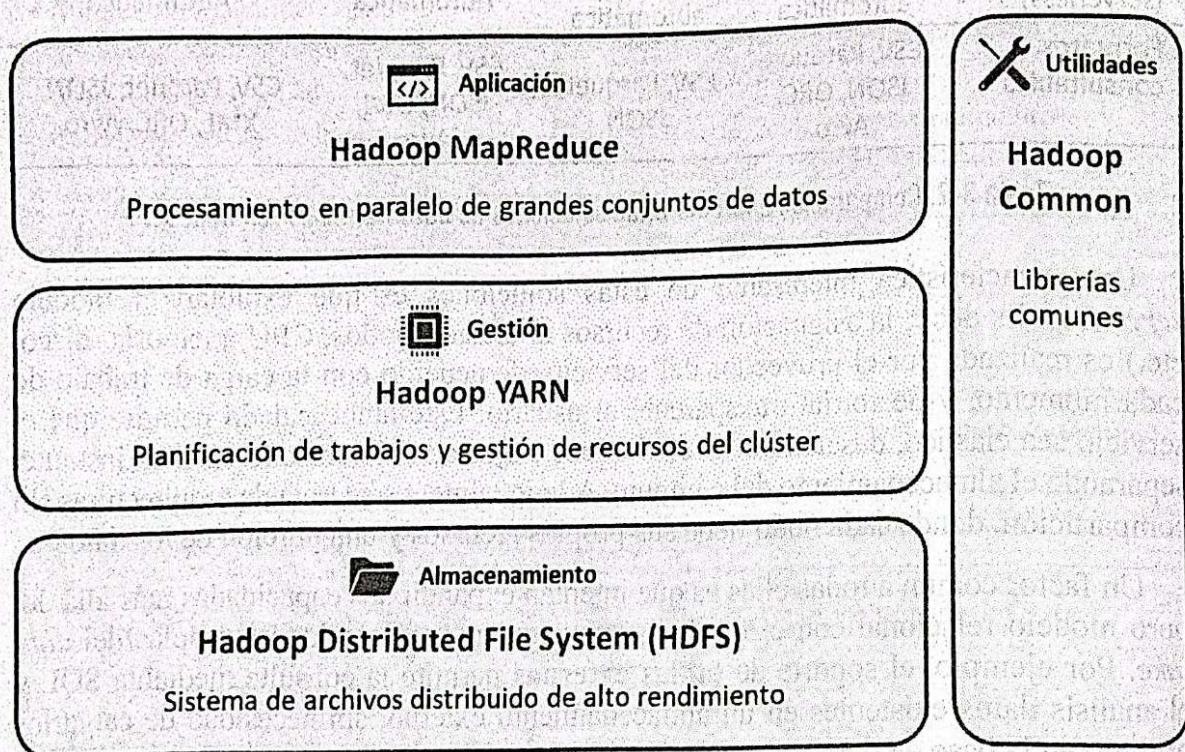


Figura 3-4. Módulos de Apache Hadoop.

3.2.1 Apache Hadoop–HDFS

Apache Hadoop⁶³ es un proyecto de código abierto consistente en una librería de software para el procesamiento distribuido de grandes conjuntos de datos. Programada en Java, la primera versión fue liberada en abril de 2006. La principal motivación detrás de su desarrollo fue la de construir un sistema que se pudiera ejecutar en un clúster de

62 Formalmente, un sistema de archivos distribuido sigue una arquitectura sin compartición (*share nothing*). Es decir, cada nodo tiene su propio almacenamiento, pudiendo existir además replicación de datos o no entre los nodos. Otra opción sería una configuración de varios nodos bajo un modelo de disco compartido con redundancia ante fallos, aunque en este caso la escalabilidad es menor.

63 <https://hadoop.apache.org/>

servidores convencionales y asequibles, proporcionando un entorno altamente escalable y con tolerancia a fallos. Aunque Hadoop ha dado lugar a un amplio ecosistema de herramientas y aplicaciones a su alrededor, el proyecto como tal consta de 4 módulos, mostrados en la Figura 3-4. Esencialmente, Hadoop proporciona un entorno de programación de aplicaciones distribuidas apoyado sobre un gestor de recursos, denominado **YARN** (*Yet Another Resource Manager*). Junto con los módulos comunes de administración (Hadoop Common), YARN es el único elemento imprescindible de Hadoop, ya que tanto la capa de almacenamiento, **HDFS** (*Hadoop Distributed File System*), como la de aplicación, **MapReduce**, pueden ser reemplazadas o complementadas con otras alternativas⁶⁴. Hablaremos de YARN y MapReduce en el siguiente capítulo.

Característica	Descripción
Resiliencia	Se asume que siempre existirán nodos del sistema de archivos no operativos. HDFS es tolerante ante estos fallos, los detecta y se recupera de forma automática. Por ello, HDFS puede ejecutarse en servidores convencionales, sin requerir RAID ⁶⁵ u otras configuraciones específicas y más costosas
Portabilidad	HDFS es fácilmente portable entre plataformas de <i>hardware</i> y <i>software</i> heterogéneas
Escalable	HDFS puede escalar a varios cientos de nodos por clúster, con archivos individuales de tamaño en el orden de los <i>terabytes</i> ⁶⁶ .
Acceso por lotes	HDFS parte de la base que la mayoría de las operaciones sobre un archivo son de lectura, involucrando estas su completo procesado. En este sentido, si bien las operaciones de adición y truncado al final del archivo están soportadas, no así las actualizaciones en un punto concreto. HDFS no es un sistema pensado para un uso interactivo, primando el rendimiento de acceso (<i>throughput</i>) frente a la latencia ⁶⁷
Localidad del dato	Con el fin de maximizar el rendimiento de todo el clúster, incluyendo el acceso a los recursos de red, HDFS permite la ejecución de aplicaciones en los nodos, moviendo el procesado a donde residen los datos

Tabla 3-3. Características de Hadoop HDFS.

64 Es posible un acceso programático desde YARN a otros repositorios de archivos, como Amazon S3 o Azure Blob Storage, entre otros.

65 RAID (*Redundant Array of Independent Disks*) consiste en la virtualización de varios discos con el fin de ofrecer una mayor tolerancia a fallos y/o un mayor rendimiento de acceso, dependiendo del nivel (RAID 0, RAID 1, RAID 5, etc.). Al multiplicar el número de discos, es una solución de redundancia más cara que JBOD (*Just a Bunch Of Drives*), donde esta se consigue a nivel de *software* y replicación de datos, como en el caso de HDFS.

66 <https://cwiki.apache.org/confluence/display/hadoop2/PoweredBy>

67 La latencia mide el tiempo en que tarda en llegar un paquete de datos a su destino, mientras que el rendimiento tiene en cuenta el número de paquetes accedidos en un periodo de tiempo.

HDFS es el sistema distribuido de archivos propio de Hadoop, diseñado originalmente para trabajar de forma efectiva con aplicaciones MapReduce al explotar la localidad del dato. El objetivo y funcionamiento de HDFS se basa en 5 premisas⁶⁸, recogidas en la Tabla 3-3.

Desde un punto de vista arquitectural, HDFS se compone de un **servidor de nombres** (**NameNode**) y de un número variables de **servidores de datos** (**DataNode**). Estos servidores son elementos de *software*, si bien lo habitual es que exista uno por nodo en el clúster. En el caso del NameNode es habitual disponer de un servidor de nombres secundario para evitar un único punto de fallo.

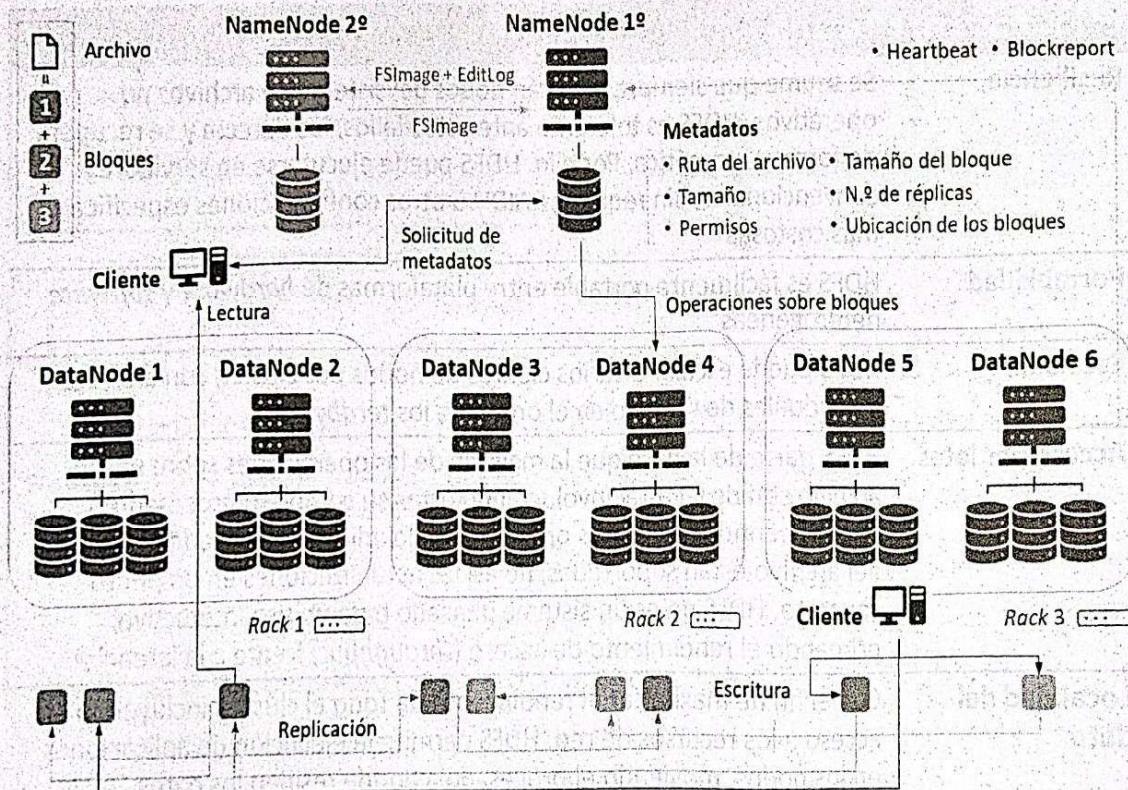


Figura 3-5. Arquitectura de HDFS

La Figura 3-5 muestra los elementos principales de un clúster de HDFS. En este ejemplo podemos identificar tres bastidores (*racks*), cada uno con dos DataNodes⁶⁹. Cada DataNode tiene su propio almacenamiento. Adicionalmente hay un NameNode primario y otro secundario, así como una serie de aplicaciones cliente que hacen uso del sistema de archivos.

El papel de los diferentes elementos es como sigue:

68 <https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>

69 El concepto de bastidor aquí es en sentido amplio, entendido como un conjunto de servidores geográficamente próximos y conectados al mismo *switch* de red, por lo que el ancho de banda entre dos nodos del bastidor es mayor que entre dos nodos en bastidores diferentes.

- ▀ La función del NameNode primario consiste en mantener el **espacio de nombres** (*namespace*) del sistema de archivos, es decir, toda la estructura jerárquica de elementos (directorios, subdirectorios, archivos y bloques) y su nomenclatura, así como soportar todas las operaciones relacionadas: listado de archivos, creación, borrado, movimiento, control de acceso, etc. Este nodo sólo realiza funciones de gestión, de forma que los datos nunca pasan a través de él⁷⁰.
- ▀ Con el fin de mantener el estado del espacio de nombres y gestionar los cambios, el NameNode primario guarda una foto actual de sus metadatos (FSImage) y un archivo de transacciones (EditLog), conteniendo este último todos los cambios desde la última foto. Para evitar la pérdida del sistema de archivos ante una eventual caída del NameNode primario, existe un NameNode secundario cuya misión es mantener una copia de esos dos ficheros, ir aplicando los cambios y replicar la foto de los metadatos al primario para agilizar su arranque en caso de reinicio.
- ▀ Con el fin de gestionar de forma ágil archivos grandes, HDFS los divide en **bloques**, replicando a su vez cada bloque a lo largo de los diferentes DataNodes del clúster. El tamaño del bloque (*block size*) y el número de réplicas (*replication factor*) es configurable a nivel de archivo, siendo este último modificable a posteriori⁷¹. Toda esta información forma parte de los metadatos gestionados por el NameNode.
- ▀ La ubicación de cada bloque de datos y sus réplicas es decisión del NameNode. Para ello tiene en cuenta el estado de disponibilidad de cada nodo (Heartbeat) y la lista de bloques que maneja (Blockreport), información que le es enviada de forma periódica por cada uno. De igual manera, el NameNode considera la **distribución de nodos entre bastidores** (*rack awareness*) para la ubicación de los bloques⁷². Cuando el número de réplicas es 3, entonces dos copias del bloque se crearán en un mismo bastidor (aunque en nodos distintos) y la tercera en otro diferente⁷³. Esta distribución de las copias permite proporcionar una mayor disponibilidad y fiabilidad de los datos, mejorando al mismo tiempo el rendimiento del clúster y la optimización del ancho de banda de la red.

-
- 70 Si el clúster es muy grande es posible añadir más NameNodes adicionales, gestionando cada uno una porción del sistema de archivos a modo de sistema federado.
- 71 Por defecto, el tamaño de bloque es de 128 MB y el factor de replicación es 3. Ahora bien, si el número de nodos es menor que el factor de replicación, entonces no todas las réplicas podrán ser creadas, marcándose como omitidas en el NameNode.
- 72 Por defecto, se asume que todos los DataNodes pertenecen al mismo bastidor.
- 73 Si el factor de réplica es superior a 3, la cuarta copia y sucesivas son ubicadas de forma aleatoria. Esta política de colocación se puede alterar para hacer el sistema más tolerante a fallos. Por ejemplo, se puede hacer que las tres réplicas se almacenen en bastidores distintos, soportando así el fallo simultáneo de dos de ellos.

- Cuando una aplicación cliente desea leer un archivo envía una petición de metadatos al NameNode. Este le contesta con una lista de los DataNodes donde puede encontrar las réplicas de los bloques que constituyen el archivo. Para elaborar esta lista, el NameNode tiene en cuenta la topología de la red, la proximidad del cliente a cada bastidor, así como la disponibilidad y la carga de trabajo de cada nodo. Si existe una réplica en el bastidor donde se encuentra el cliente⁷⁴, entonces esta será empleada para satisfacer la petición de lectura.
- De forma similar, cuando un cliente quiere escribir un archivo hace también una solicitud al NameNode, indicando el nombre del archivo, su tamaño, etc. El NameNode le responderá indicando la lista de DataNodes para cada bloque de datos. Si el cliente se encuentra en el clúster, entonces el primer bloque se escribirá en su mismo nodo. De lo contrario el nodo es elegido aleatoriamente. Cuando el cliente escribe el primer bloque de datos, el DataNode de destino se encarga de ir replicándolo al segundo nodo y este a su vez al tercero, según la configuración establecida por el NameNode. Este proceso (*replication pipelining*) se va ejecutando para todos los bloques del archivo.

Hadoop está pensado y optimizado para soportar un modelo de una escritura y múltiples lecturas de un mismo archivo (WORM, *Write Once Read Many*), donde el acceso se efectúa sobre todo el contenido, de ahí los tamaños de bloque tan grandes. Una vez creados, los ficheros no pueden ser actualizados, soportando solo el anexo o el truncado de su contenido. Desde el punto de vista del acceso, HDFS proporciona una serie de API (Java, C, REST), una interfaz HTTP y una línea de comandos (FS Shell).

En el siguiente capítulo estudiaremos **Apache Hive**, una solución para la implantación de un *data warehouse* virtual sobre Hadoop empleando SQL.

3.2.2 Formatos de archivos

Debido a sus características de resiliencia, procesamiento por lotes, localidad y, sobre todo, bajo coste, Apache Hadoop se convirtió en su momento en la primera opción para implementar un *data lake* en la mayoría de las organizaciones. Pero otro factor importante era la posibilidad de utilizar distintos formatos de archivo especializados, de naturaleza genérica u orientados a consultas, y de código abierto.

⁷⁴ No hay que perder de vista que una de las características principales de Hadoop es la localidad del dato, de forma que las aplicaciones se pueden ejecutar en los nodos.

	CSV	JSON	SEQ	RCFfile	ORC	Avro	Parquet
Tipo de archivo	Texto	Texto	Binario	Binario	Binario	Binario	Binario
Metadatos	No	No	Sí	No	Sí	Sí	Sí
Acceso aleatorio	No	No	No	No	Si	No	Sí
Orientación del registro	Fila	Fila	Fila	Columna	Columna	Fila	Columna
Esquema incluido	No	No	No	No	Sí	Sí	Sí
Esquema modificable	No	Sí	No	No	Parcial	Sí	Parcial
Tipos complejos	No	Sí	No	No	Sí	Sí	Sí
Adición de datos	Sí	Sí	Sí	Sí	Sí	Sí	No

Tabla 3-4. Formatos de archivo en Hadoop.

La Tabla 3-4 enumera estos formatos. Todos ellos son divisibles en bloques de registros, característica necesaria en HDFS. También admiten compresión a distintos niveles (registro o bloque), siendo este un factor importante de cara a reducir el espacio en disco y aumentar la velocidad de proceso; no hay que perder de vista que la compresión permite ejecutar el paso de disco a memoria (la operación más costosa) de forma más rápida, aún a expensas de un mayor trabajo posterior de la CPU en la tarea de descompresión.

La orientación del registro es otro elemento a tener en cuenta. Al igual que en las bases de datos relacionales, el almacenamiento puede ser en filas o en columnas. En el primer caso cada fila es ubicada de forma contigua en el archivo, mientras que en el segundo son los valores de cada columna para todas las filas los que son almacenados juntos. Por lo tanto, un almacenamiento en columnas será preferible cuando se desee acceder solo a un número reducido de estas, mientras que la orientación en filas dará un mejor rendimiento cuando se necesiten leer un número menor de estas últimas, pero de forma completa.

ORC (*Optimized Row Columnar*) y **Parquet** son dos de los formatos orientados a columnas más empleados cuando el rendimiento de las consultas es un factor importante. Estos incluyen además metadatos sobre la distribución de los valores y su ubicación en el archivo, lo que permite un acceso aleatorio más optimizado a los registros.

Otro factor importante es la inclusión del esquema de datos y su posibilidad de modificación. Si el esquema no está incluido en el fichero, es necesario conocer su estructura de atributos por otra vía para poder consultarlos. Si existe la necesidad de que el esquema pueda variar con el tiempo, un formato como **Avro** puede ser una buena elección.

El formato de secuencia (**SEQ**) es muy empleado en Hadoop como mecanismo de almacenamiento intermedio. Son archivos binarios donde cada fila contiene un par clave-valor. La clave es un identificador del registro y el valor puede ser cualquier cosa. Por este motivo, se suelen usar para condensar múltiples archivos pequeños en uno grande, haciendo su gestión más sencilla. Existen librerías para distintos lenguajes de programación que permiten la creación y manipulación de estos formatos de archivo.

3.2.3 Escenarios e inconvenientes

La Tabla 3-5 resume los puntos fuertes de HDFS y sus deficiencias.

Escenarios	Inconvenientes
<ul style="list-style-type: none"> • Soporte para el almacenamiento de datos masivos, con estructuras variables y cambiantes • Alta tolerancia a fallos • Alto rendimiento en el procesado de grandes archivos • Soporte a una gran variedad de formatos de archivo optimizados • Concebido para explotar la localidad del dato • Soporte a distintos modelos de cómputo y aplicaciones • Hardware convencional 	<ul style="list-style-type: none"> • Mal rendimiento en la gestión de archivos pequeños y en patrones de acceso aleatorio e iterativos • Falta temporal de consistencia en los datos por la replicación de bloques • Modelo de seguridad muy complicado • Excesivo acoplamiento del almacenamiento y el cómputo • Elevada latencia de acceso, no adecuada para procesos en tiempo real • Entorno complejo de gestionar y muy difuso en cuanto a herramientas y proyectos relacionados

Tabla 3-5. Ventajas e inconvenientes de HDFS para *Big Data*.

3.2.4 Software y soluciones para Apache Hadoop

Como plataforma de código abierto que es, Apache Hadoop puede ser descargado e instalado de forma gratuita, estando disponible para sistemas operativos GNU/Linux y Windows⁷⁵. Desde un punto de vista comercial, existen distribuciones de pago que, siguiendo un modelo habitual con el *software* de código abierto, ofrecen soporte alrededor de la plataforma, así como una serie de herramientas y utilidades enfocadas a simplificar su gestión. Este era el caso de compañías como **Cloudera** y **Hortonworks**, pioneras en el desarrollo y mantenimiento de Hadoop, que se fusionaron en 2018 unificando sus plataformas. Actualmente, Cloudera Data Platform (CDP) Private Cloud es de las pocas distribuciones comerciales de Hadoop que pueden emplearse para desplegar un clúster localmente.

75 <https://hadoop.apache.org/releases.html>

Como en el resto de las soluciones de *Big Data*, la principal oferta comercial de Hadoop gira alrededor de los servicios gestionados en la nube, con modelos de pago por uso. Los principales proveedores proporcionan servicios de Hadoop y Spark combinados, junto a otras soluciones del ecosistema de Apache. **Amazon EMR**, **Azure HDInsight** y **Google Dataproc** son algunos ejemplos. De forma similar a las soluciones de *data warehouse*, estos servicios facilitan el autoescalado de la plataforma, permitiendo incluso prescindir temporalmente de HDFS para la persistencia. Esto lo consiguen integrándose con sistemas de almacenamiento de objetos, que veremos en el siguiente apartado. De esta manera, es factible separar el cómputo del almacenamiento, siendo posible eliminar el clúster de Hadoop cuando no se está usando, manteniendo los datos para un uso posterior.

Volveremos a hablar de Hadoop de una forma más integral en el siguiente capítulo, al abordar MapReduce y otras aplicaciones de su ecosistema para el procesamiento de los datos.

3.3 ALMACENES DE OBJETOS

Una de las características fundamentales de HDFS es su modelo sin compartición, especialmente en lo referente al almacenamiento de datos: cada DataNode del clúster tiene sus propios **discos conectados directamente (DAS, Direct-Attached Storage)**, estando los datos divididos en bloques y replicados entre los nodos. De esta manera, un DataNode no tiene conocimiento de lo que es un archivo, almacenando sólo un bloque en su sistema local como si fuera un fichero más. Es el NameNode el que proporciona sentido a todos los bloques y réplicas, dando la consistencia de un sistema de archivos unificado.

Entre otras muchas cosas, HDFS vino a suponer una alternativa a los **sistemas de archivos en red (NFS, Network File System)** convencionales (Figura 3-6). Aunque en HDFS los nodos del clúster están conectados en red, el almacenamiento es local en cada uno. Por el contrario, en un NFS existe un almacenamiento compartido en forma de un **servidor de ficheros conectado en red (NAS, Network-Attached Storage)** con sus propios discos virtualizados (RAID), interfaz de red, sistema operativo y software de gestión. NFS implica un patrón opuesto a la localidad del dato, ya que implica su compartición y desplazamiento hacia los puntos de proceso a los que da servicio⁷⁶. Los servidores NAS se utilizan típicamente como servidores de ficheros.

Otra de las aproximaciones para la compartición de datos es a través de una **red de dispositivos de almacenamiento (SAN, Storage Area Network)**, dedicada y separada de la red de servidores a los que da servicio. Estas redes no comparten un sistema de

76 Las tasas de transferencia de red actuales hacen que esto no sea necesariamente un problema. Incluso existen soluciones que permiten combinar la tecnología NAS con los protocolos de HDFS, evitando la replicación de bloques.

archivos, sino que exponen directamente discos para el almacenamiento de bloques, empleando distintos protocolos (Fibre Channel). Ahora bien, un servidor puede montar estos discos y formatearlos con un sistema de archivos⁷⁷. Este almacenamiento de bloques permite una transferencia muy rápida de grandes cantidades de datos con un rendimiento muy consistente. Se acostumbra a utilizar para la persistencia de contenedores, máquinas virtuales y espacios de tablas para bases de datos relacionales, donde el acceso directo a disco (*raw I/O*) proporciona un mejor resultado.

Almacenamiento de archivos	Almacenamiento de bloques	Almacenamiento de objetos
Organización jerárquica de datos en archivos, carpetas, subdirectorios y directorios	División de datos en bloques y su almacenamiento como piezas separadas	Gestión de los datos como un objeto autocontenido bajo una estructura plana
<ul style="list-style-type: none"> Efectivo con un número reducido de archivos estructurados Los archivos se pueden compartir fácilmente con protocolos comunes Guardan metadatos Escalabilidad limitada y navegación compleja Tecnología NAS o en la nube 	<ul style="list-style-type: none"> Los bloques se guardan de forma eficiente con un identificador único El dato se ensamblan y recuperan muy rápido Puede ser integrado con diferentes sistemas operativos No guardan metadatos Tecnología SAN o en la nube 	<ul style="list-style-type: none"> Permite una localización rápida del dato Guarda metadatos sobre la estructura, permisos, políticas, etc. Ideal para datos estáticos no estructurados Acceso sin sistema operativo (HTTP, API REST) Tecnología en la nube
Colaboración, archivado, copias de seguridad	Contenedores, máquinas virtuales, bases de datos	Big Data, IoT, contenido multimedia

Figura 3-6. Tecnologías de almacenamiento compartido en red.

El tercer mecanismo de compartición es el **almacenamiento de objetos (object storage)**. En este caso, el elemento de gestión es un objeto, conformado por los datos en sí, un identificador único y un conjunto de metadatos que lo caracterizan. Los objetos son persistidos de forma plana, sin ningún tipo de organización jerárquica en directorios. Esta simplificación permite el crecimiento de los recursos de almacenamiento según necesidades, sin limitar el tamaño que pueden llegar a alcanzar los objetos, y facilitando y agilizando su acceso al poderse referenciar directamente mediante su identificador único.

Dadas sus características, el almacenamiento de objetos permite la gestión de grandes volúmenes de datos, de estructuras variables y con un patrón de acceso en el que el objeto se escribe una vez (la actualización de los datos implica la rescritura de todo el objeto) y se lee de forma repetida. Otra de sus grandes ventajas es la de ofrecer un acceso

77 Es decir, una unidad NAS se mostraría a un servidor en forma de unidad de red, mientras que una SAN aparecería como un disco sin formato.

programático: un almacén de objetos no se monta a nivel de sistema operativo, sino que se manipula mediante distintas interfaces y protocolos (HTTP, REST, etc.), a través de las cuales los objetos son creados, accedidos, movidos, etc.

Otro rasgo que aporta mucho valor es la riqueza de metadatos que pueden acompañar al objeto. Este se puede etiquetar de múltiples maneras, facilitando su búsqueda, consulta y gestión. Estos metadatos, definidos en forma de clave-valor, se pueden clasificar como fijos, donde la clave no es editable pero sí el valor, y a medida, establecidos enteramente por el usuario. Con los primeros se puede establecer el control de acceso al objeto, el tipo de contenido o su forma de presentación.

Si bien existen soluciones de almacenamiento de objetos en local, su popularidad y uso está ligada al desarrollo de la computación en la nube, especialmente al de aplicaciones nativas en este entorno. Los modelos de facturación por tipo y volumen de almacenamiento, así como por el número de veces que se accede a los objetos son muy atractivos desde el punto de vista económico. Los servicios en la nube son, además, muy elásticos al acomodar recursos de forma dinámica y transparente para el usuario.

Aunque hemos comentado que la estructura de estos almacenes es plana, los objetos se agrupan en **contenedores** (*buckets*), que pueden ser creados y modificados a voluntad. Además de proporcionar una mínima organización⁷⁸, los contenedores permiten establecer un control de acceso a los objetos que contienen. Otra de sus características es la implementación de **clases de almacenamiento** (*storage classes*). Los proveedores de almacenes de objetos en la nube ofrecen una serie de clases predefinidas sobre las que crear los contenedores⁷⁹. Estas clases definen como queremos que sea el almacenamiento: tamaño, redundancia a lo largo de distintas regiones geográficas, latencia de acceso, disponibilidad, etc. Esto es interesante para segregar los objetos según la frecuencia de uso y su temperatura, pagando de acuerdo a esto.

3.3.1 Catálogos de tablas

El tamaño máximo que puede tener un objeto en un almacén en la nube es del orden de varios *terabytes*⁸⁰. Para trabajar con conjuntos de datos tabulares de estos tamaños o superiores, es habitual segregarlos en varios objetos organizados por columnas, Parquet u ORC, que pueden ser consultados después de forma conjunta. Sin embargo, esta división no es del todo efectiva cuando se trata de volúmenes muy grandes, dando lugar también a problemas cuando se modifican los datos. Esto último es consecuencia de la falta de características ACID de estos formatos y almacenes, lo que permite la aparición de transacciones incompletas entre particiones e inconsistencias en las operaciones de lectura.

Como vimos en el capítulo anterior, la idea del *data lakehouse* está enfocada a solventar estos problemas dentro del *data lake*, con independencia de que este resida en un sistema de archivos distribuido o en un almacén de objetos. Es decir, asumir el

78 En cualquier caso, no es posible anidar contenedores a modo de subdirectorios.

79 También suele ser posible especificar la clase directamente a nivel de objeto.

80 5 TB en el caso de Amazon S3 (<https://aws.amazon.com/s3/faqs/>).

procesamiento por lotes y en tiempo real en una única plataforma basada en formatos abiertos, con una capa de gestión de metadatos que de consistencia al acceso y a la modificación de los datos.

Existen en la actualidad tres proyectos de código abierto que van en este sentido: **Apache Iceberg** (2017), **Apache Hudi** (2017) y **Delta Lake** (2019). Iceberg nació como un proyecto interno de Netflix, mientras que Hudi lo fue de Uber. Ambos acabaron bajo el paraguas de la Apache Software Foundation. Delta Lake fue desarrollado por Databricks, y posteriormente donado a la Linux Foundation. Los tres giran alrededor de la definición de un formato de tabla que conceptualiza y unifica un conjunto de archivos comunes (ORC, Parquet, etc.) permitiendo a las aplicaciones interactuar con él como si se tratara de una base de datos: acceso SQL, características ACID, evolución del esquema, versionado de datos, etc.

En definitiva, estas capas semánticas permiten la creación de un catálogo de tablas sobre un almacén de objetos, agnóstico tanto del formato físico de los datos como del motor de consulta, exponiendo su contenido mediante API y otras interfaces y protocolos.

3.3.2 Escenarios e inconvenientes

El almacenamiento de objetos ha ido ganado mucha popularidad frente a los sistemas de archivos distribuidos como HDFS para la implantación de *data lakes*, especialmente debido al movimiento de cargas de trabajo a la nube. La Tabla 3-6 contiene sus principales ventajas e inconvenientes.

Escenarios	Inconvenientes
<ul style="list-style-type: none"> Necesidad de interoperabilidad, separando el almacenamiento del cómputo, dando acceso a los datos desde cualquier sistema Gran libertad de etiquetado de los objetos, permitiendo una búsqueda y localización rápida Alta disponibilidad y resiliencia, con replicación a nivel geográfico Gran elasticidad, con un modelo de costes muy económico, flexible y predecible Optimización del almacenamiento, moviendo de forma automática objetos a contenedores más económicos en función de su uso Soporte a una variedad de formatos de archivo optimizados Acceso programático a los datos a través de diferentes protocolos Diversas opciones de encriptación y modelo de control de acceso integrado con el resto de los recursos en la nube 	<ul style="list-style-type: none"> Los objetos son inmutables. Cualquier alteración implica su rescritura No hay soporte transaccional nativo mediante mecanismos de bloqueo y compartición de objetos Variabilidad en los tiempos de acceso, impactando en el rendimiento de las aplicaciones que usan los datos Organización de objetos excesivamente plana Mayor latencia de acceso, significativa en el caso de objetos pequeños Mal rendimiento en cargas de trabajo iterativas Dependencia de los servicios de procesamiento del proveedor para una explotación ágil de los datos

Tabla 3-6. Ventajas e inconvenientes del almacenamiento de objetos en la nube para Big Data.

El poder desacoplar el almacenamiento del cómputo es uno de los requerimientos más demandados, no sólo por facilitar el escalado de los sistemas, sino por aligerar los costes que supone tener ligado un recurso muy caro, como la capacidad de proceso, con el almacenamiento, mucho más barato. Ahora bien, esta separación implica la pérdida de la localidad del dato, una característica importante en un sistema como Hadoop con YARN, ya que el acceso implica una transferencia a través de la red. En cualquier caso, la combinación de ambos tipos de almacenamiento permite descargar datos poco usados de HDFS a un almacén de objetos, aligerando el clúster de Hadoop. Es más, permite no consumir y facturar por recursos de cómputo mientras el clúster no está en uso, creando una instancia de forma dinámica cuando se necesite y recuperando los datos en local para su uso.

3.3.3 Servicios para el almacenamiento de objetos

Cada proveedor de servicios en la nube tiene su propia oferta de almacenamiento de objetos, más o menos parecida: **Amazon S3**, **Google Cloud Storage**, **Azure Blob Storage** o **IBM Cloud Object Storage** son algunas de ellas. Amazon S3 se puede considerar el pionero de este tipo de almacenamiento en la nube, y su API de acceso es prácticamente un estándar, ya que muchos de sus competidores la implementan para acceder a su propio servicio⁸¹.

Adicionalmente al propio almacenamiento, es habitual que los proveedores monten alguna capa de consulta sobre los objetos, en línea con la idea de virtualizar el *data warehouse* a través de un *data lake* que veíamos en el capítulo anterior⁸². Mediante estos servicios se pueden lanzar consultas empleando el lenguaje SQL sobre archivos CSV, JSON, Parquet, ORC, etc., guardando los resultados en el propio almacén de objetos o en una base de datos relacional. El esquema y el formato de los datos se puede especificar en la propia consulta (*schema-on-read*), o bien guardar la definición en un catálogo para un uso repetitivo posterior. Veremos estos motores de consulta en el siguiente capítulo. También existen soluciones que sobre el almacén de objetos implementan un sistema de archivos jerárquico, compatible con Apache Hadoop⁸³.

3.4 BASES DE DATOS NOSQL

Si bien el abanico de bases de datos NoSQL es amplio y heterogéneo, podríamos citar una serie de rasgos comunes entre todas ellas:

- Almacenan los datos en un formato no tabular, sin seguir un modelo de entidades al anidar las relaciones dentro de una estructura única, más fácil de consultar al eliminar la necesidad de enlaces (*joins*).

⁸¹ Bien directamente (IBM), parcialmente (Google), o bien a través de terceros (Microsoft), como S3Proxy (<https://github.com/gaul/s3proxy>).

⁸² IBM Cloud Data Engine o Amazon Athena son dos ejemplos.

⁸³ Por ejemplo, Azure Data Lake Storage.

- ▼ Son muy flexibles en cuanto al esquema de los datos y su evolución, pudiendo albergar tanto datos estructurados como semiestructurados, aunque su foco está en estos últimos.
- ▼ El crecimiento de la base de datos es fundamentalmente horizontal, aumentando o reduciendo el número de nodos según necesidades.
- ▼ Salvo excepciones, no implementan transacciones ACID (tampoco es su negocio, como ahora veremos).
- ▼ El desarrollo de soluciones sobre ellas es más directo y próximo al programador, ya que las estructuras sobre las que se codifica coinciden con las que se almacenan⁸⁴.

Desde el momento en que las bases de datos NoSQL son muy específicas de casos de uso concretos, la elección entre estas y una base de datos relacional no siempre es fácil. El posicionamiento del modelo BASE como alternativa a las propiedades ACID y teorema CAP pueden ayudar en este sentido.

3.4.1 El modelo BASE y el teorema CAP

De forma simplificada, podríamos decir que el **modelo BASE** (*Basically Available, Soft state, Eventual consistency*) es a las bases de datos NoSQL lo que el modelo ACID es a las transaccionales⁸⁵. La Tabla 3-7 describe sus características, que giran alrededor de relajar la consistencia de los datos en un sistema formado por varios nodos, en los que los datos se distribuyen y replican siguiendo un escalado horizontal. Por este motivo se plantean de forma inversa a como aparecen en el acrónimo.

Característica	Descripción
Consistencia eventual (<i>Eventual consistency</i>)	No se asegura una consistencia inmediata una vez finalizada una operación que modifica los datos, ya que esta debe propagarse a los diferentes nodos para asegurar la disponibilidad. Mientras esta consistencia se alcanza, el sistema reflejará un estado temporal de los datos
Estado transitorio (<i>Soft state</i>)	El estado de los datos puede variar entre accesos sin que medie una interacción que los modifique, y hasta que todos los nodos del sistema converjan.
Disponibilidad (<i>Basically available</i>)	El sistema estará siempre disponible, tolerando situaciones de fallo, aunque permitiendo situaciones de inconsistencia en los datos, como lecturas fantasma o no repetibles.

Tabla 3-7. Propiedades BASE en una base de datos NoSQL.

84 Con las bases de datos SQL es habitual trabajar con conversores que hacen de puente entre los objetos manejados a nivel de código y las tablas relacionales (ORM, *Object Relational Mapping*).

85 Además del significado de sus siglas, el nombre del acrónimo está inspirado en la contraposición que en química existe en el comportamiento de sustancias ácidas y básicas (<https://www.dataversity.net/acid-vs-base-the-shifting-ph-of-database-transaction-processing/>).

En definitiva, en las bases de datos que siguen el modelo BASE prima la disponibilidad del sistema y su escalado horizontal, sacrificando cualquier garantía alrededor de la consistencia de los datos, cuyo aseguramiento se delega en las aplicaciones y sus programadores. No existe, por lo tanto, el aislamiento transaccional, pudiéndose dar situaciones como lecturas corruptas, no repetibles o fantasmas⁸⁶.

Para contextualizar juntos los modelos ACID y BASE y posicionar las bases de datos SQL frente a las NoSQL, es necesario simplificarlos mediante el **teorema CAP** (*Consistency, Availability, Partition tolerance*)⁸⁷. Este establece que en un sistema de base de datos no se pueden garantizar simultáneamente la **consistencia**, la **disponibilidad** y la **distribución efectiva** de los datos en diferentes nodos, siendo estas tres características mutuamente excluyentes. En definitiva, y lo que es más importante, es necesario renunciar a una de estas en beneficio de las otras dos.

La Figura 3-7 ilustra esta idea, con las tres posibles combinaciones tras tomar las características de dos en dos. Queda claro que en ausencia de división de los datos en diferentes nodos, el sistema solo puede escalar verticalmente, pudiéndose satisfacer tanto la consistencia como la disponibilidad a la vez.

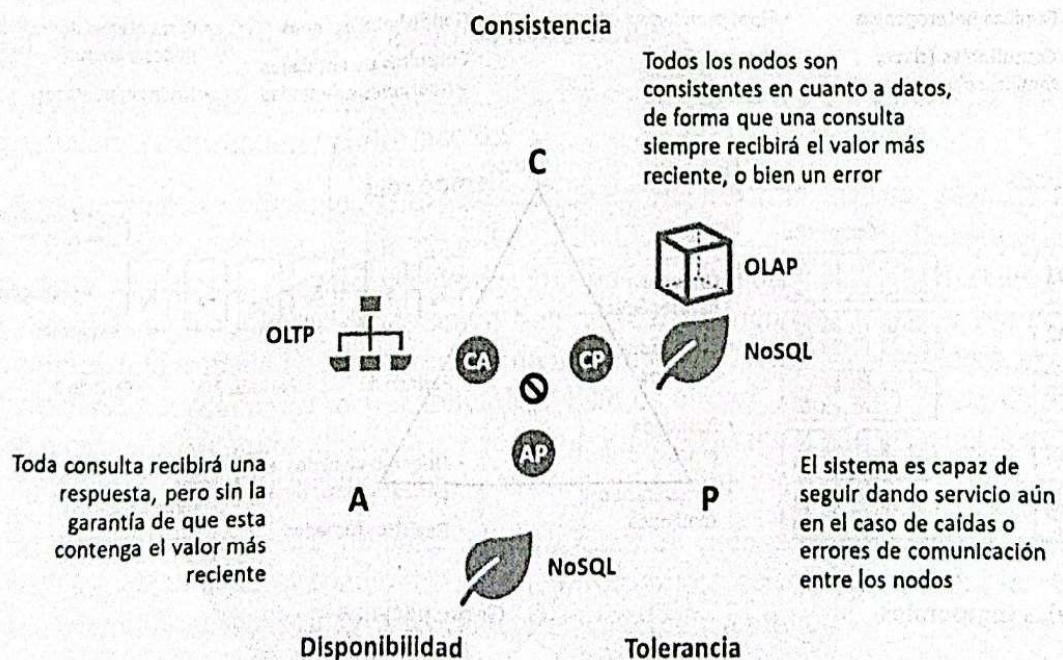


Figura 3-7. Las tres propiedades del Teorema CAP.

86 [https://es.wikipedia.org/wiki/Aislamiento_\(ACID\)](https://es.wikipedia.org/wiki/Aislamiento_(ACID))

87 Formulado por Eric Brewer, un científico computacional, en 1999.

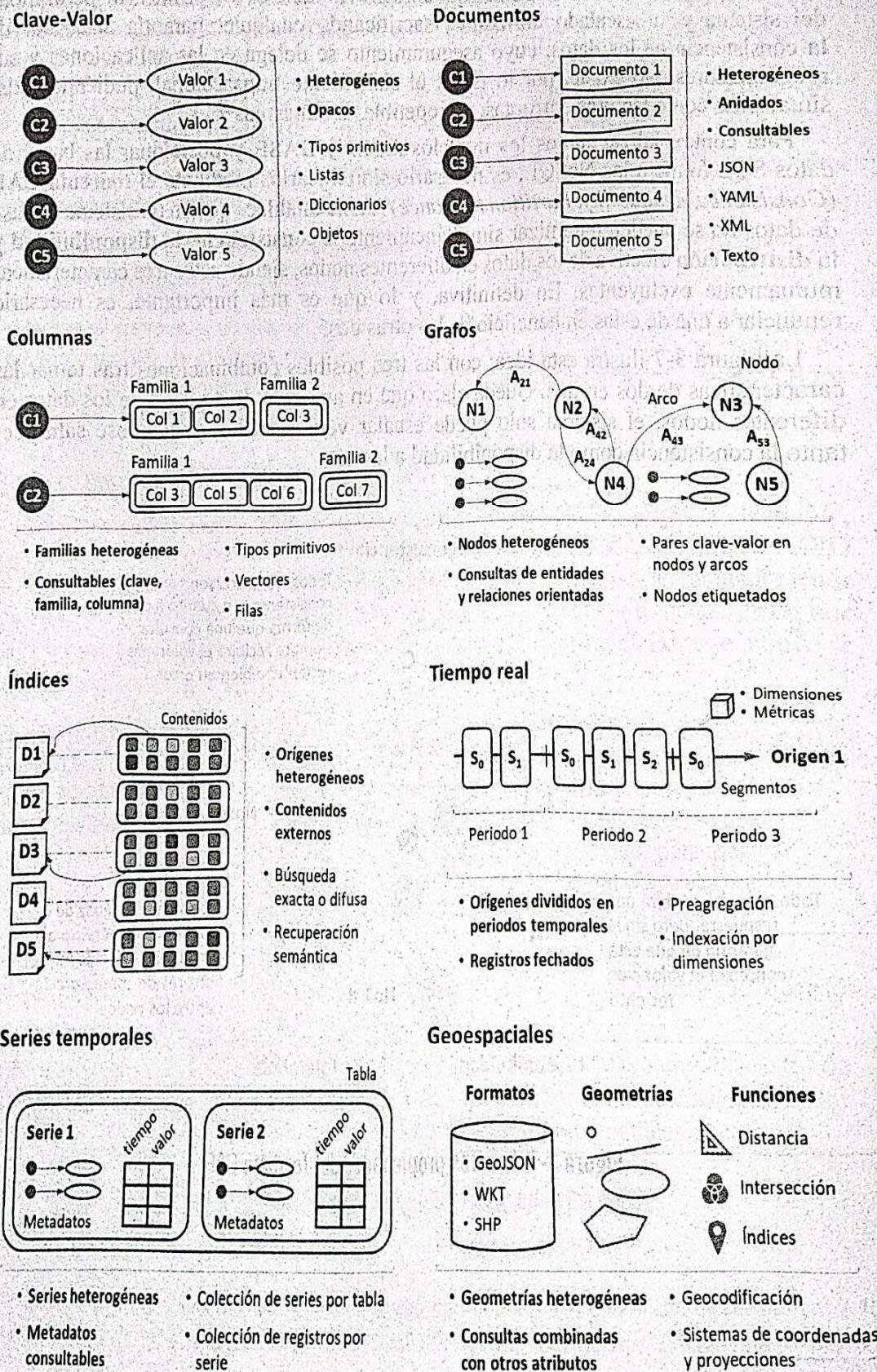


Figura 3-8. Categorías de bases de datos NoSQL.

En el mundo SQL, Un sistema OLTP debe ubicarse necesariamente en la intersección CA, ya que debe proporcionar siempre una respuesta, siendo esta además la más reciente. Por este motivo, la distribución de los datos entre nodos no es posible⁸⁸. Por el contrario, un sistema de *data warehouse* (OLAP), en el que la disponibilidad pudiera no ser tan crítica, podría situarse en el lado CP, garantizando el servicio y asegurando la consistencia del dato. Esta debiera primarse siempre por encima de la disponibilidad, ya que la toma de decisiones no puede basarse en datos inconsistentes.

Las bases de datos NoSQL, siguiendo el modelo BASE y exhibiendo la facilidad de crecimiento horizontal como una de sus principales características, priorizan la disponibilidad frente a la consistencia (AP), aunque hay otras posibles combinaciones. En cualquier caso, existe el consenso generalizado de que no hay una solución única para todas las necesidades de almacenamiento (*one size fit's it all*), requiriéndose un conjunto de ellas para realizar una gestión integral del dato dentro de la organización.

Hay distintas formas de catalogar las bases de datos NoSQL, pero lo habitual es hacerlo en base al modelo de datos. Este marcará los posibles campos de aplicación, señalando también hacia un lateral u otro del teorema CAP.

3.4.2 Gestores NoSQL según el modelo de datos

La Figura 3-2 incluye ocho categorías de bases de datos NoSQL en las que varía la forma en que se estructura la información.

Aunque existen variaciones dentro de cada categoría, la forma en la que se gestiona el dato es común en cada una. La Figura 3-8 esquematiza el modelo para cada categoría. Es importante resaltar que una misma base de datos puede pertenecer a más de una categoría. Por ejemplo, algunas bases de datos en tiempo real dividen los datos en segmentos temporales, almacenando cada uno en forma de columnas. De la misma manera, existen solapamientos en cuanto a la funcionalidad. Este es el caso de las bases de datos de documentos y las de índices, o las de series temporales y las que están orientadas a datos en tiempo real, estas últimas muy similares en cuanto a funcionalidad.

La Tabla 3-8 resume las ventajas y limitaciones de cada gestor, indicando también su posicionamiento en el triángulo CAP. Este último es orientativo, ya que en cada implementación pueden primar unas características frente a otras.

Las **bases de datos clave-valor** son las más sencillas de todas. Los datos son almacenados a modo de pares, identificados por una clave única y un contenido (valor) que puede ser cualquier cosa. Este contenido es opaco a la base de datos, por lo que no se puede indexar ni consultar. La idea es que la base de datos sea muy ágil a la hora de almacenar y recuperar estos pares, delegando en la aplicación que explota los datos la

⁸⁸ No es posible en un sistema sin compartición (*shared nothing*), pero sí en sistemas formados por nodos (miembros) que comparten el almacenamiento y, por lo tanto, pueden asegurar la consistencia de los datos. Este es el caso de Oracle RAC o IBM Db2 pureScale, donde un clúster puede constar de decenas de miembros.

gestión e interrogación de los valores. Este modelo hace que su crecimiento horizontal sea rápido y sencillo. Existen motores que pueden almacenar los datos en memoria a modo de caché, agilizando todavía más el acceso.

Categoría	Ventajas	Limitaciones	Lados CAP
Clave-Valor	<ul style="list-style-type: none"> Búsquedas simples, recuperando todo el valor Muy escalables Pueden funcionar en memoria 	<ul style="list-style-type: none"> Opacidad de los valores Consultas complejas involucrando varias claves Actualización de datos 	AP CP
Documentos	<ul style="list-style-type: none"> Estructura del dato (JSON, YAML) familiar al desarrollador Gran rapidez de consulta y escalado Diferentes opciones de indexación 	<ul style="list-style-type: none"> Dificultad para crear relaciones complejas Gran consumo de memoria Tamaño del documento y anidación limitada 	AP CP
Columnas	<ul style="list-style-type: none"> Facilidad de adición de columnas de forma dinámica Gran rendimiento en escritura y actualización Recuperación muy rápida en consultas simples 	<ul style="list-style-type: none"> Modelado de datos complejo Patrones de consulta variables (<i>ad-hoc</i>), cambiantes y con agregaciones complejas Soporte de relaciones (<i>joins</i>) 	AP CP
Grafos	<ul style="list-style-type: none"> Consultas y relaciones complejas Fácil evolución del esquema Propiedades ACID (según caso) 	<ul style="list-style-type: none"> Dificultad de escalado horizontal Soporte de grandes volúmenes de datos Usos operacionales (no analíticos) 	CA
Índices	<ul style="list-style-type: none"> Indexación de grandes volúmenes de datos Búsqueda semántica y difusa (sinónimos, raíz, familia) Agregaciones anidadas 	<ul style="list-style-type: none"> Indexación lenta en ocasiones Actualizaciones costosas Soporte de relaciones complejas 	AP

Tabla 3-8. Características de las bases de datos NoSQL.

Categoría	Ventajas	Limitaciones	Lados CAP
Tiempo real 	<ul style="list-style-type: none"> • Alta velocidad de almacenamiento y proceso • Soporte de análisis multidimensional • Capacidad de agregación 	<ul style="list-style-type: none"> • Coste elevado en términos de <i>hardware</i> • Capacidades de consulta limitadas • Almacenamiento no primario ni definitivo 	AP
Series 	<ul style="list-style-type: none"> • Alta capacidad de compresión por la uniformidad del dato • Agregaciones rápidas • Expiración y archivado automático de datos 	<ul style="list-style-type: none"> • Posible pérdida de eventos • Complejidad en la gestión • Indexación costosa en términos de recursos 	CP
Geoespaciales 	<ul style="list-style-type: none"> • Indexación por características espaciales • Consultas espaciales • Representación de topologías arbitrarias 	<ul style="list-style-type: none"> • Alto consumo de recursos en cálculos espaciales • Escalado • Interoperabilidad de formatos 	CA

Tabla 3-8 (continuación). Características de las bases de datos NoSQL.

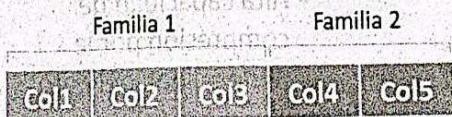
Las **bases de datos de documentos** parten de la misma idea que las de clave-valor, pero ahora el valor es un documento con una estructura a la que el gestor tiene acceso, como XML, JSON o YAML. Este esquema puede variar entre registros y su contenido puede ser indexado, de forma que es consultable, normalmente a través de lenguajes propietarios. El modelo de datos suele estar altamente desnormalizado, incluyendo unas estructuras anidadas dentro de otras (por ejemplo, un documento conteniendo los datos de un cliente y todo el detalle de las compras que realizó en el último mes). Esto es debido a que la relación de documentos (*joins*) es bastante compleja, por lo que se opta por incluir mucha información en un mismo documento, lo que por otro lado acelera las consultas. Al igual que las bases de datos clave-valor, estos gestores no son una buena opción para la implementación de transacciones complejas involucrando múltiples documentos.

De forma muy similar al almacenamiento en columnas en las bases de datos relacionales o en formatos de archivo como ORC y Parquet, las **bases de datos de columnas** agrupan atributos en familias y almacenan estas de forma contigua, pudiendo variar entre registros (Figura 3-9). Por ejemplo, una **familia de columnas** puede contener información sociodemográfica de un cliente (estado civil, profesión, ciudad, etc.) y otra familia de columnas agregados de operaciones (saldo medio en el último mes, número de operaciones a crédito, etc.). A la hora de almacenar en disco

estos datos, las columnas que forman parte de una familia estarán juntas en un mismo archivo, siendo el número de registros variable. Si hay clientes de los que todavía no tenemos operaciones, el registro correspondiente no tiene porqué implementar esa familia. Además de reducir el espacio necesario en disco, esta segregación permite acelerar los tiempos de acceso, ya que solo se leen las columnas involucradas en cada consulta.

Vista lógica

RID	Col1	Col2	Col3	Col4	Col5
001	V ₁₁	V ₁₂	V ₁₃	V ₁₄	V ₁₅
002	V ₂₁	V ₂₂	V ₂₃	V ₂₄	V ₂₅
003	V ₃₁	V ₃₂	V ₃₃	V ₃₄	V ₃₅



Almacenamiento en filas

RID001	RID002	RID003
V ₁₁ V ₁₂ V ₁₃ V ₁₄ V ₁₅	V ₂₁ V ₂₂ V ₂₃ V ₂₄ V ₂₅	V ₃₁ V ₃₂ V ₃₃ V ₃₄ V ₃₅

Archivo 1

Almacenamiento en columnas

Col1	Col2	Col3	Col4	Col5
V ₁₁ :001	V ₂₁ :002	V ₃₁ :003	V ₁₂ :001	V ₂₂ :002
V ₃₂ :003	V ₁₃ :001	V ₂₃ :002	V ₃₃ :003	

Col4	Col5
V ₁₄ :001	V ₂₄ :002
V ₃₄ :003	V ₁₅ :001
V ₂₅ :002	V ₃₅ :003

Archivo 1

Archivo 2

Figura 3-9. Almacenamiento en filas y en columnas.

Estas bases de datos tienden a tener un mejor rendimiento que las bases de datos clave-valor o de documentos, ya que las posibilidades de indexación son mayores al poder acceder por registro, familia o columna. Presentan además muy buenas propiedades de escalado horizontal y una alta flexibilidad en el esquema de los datos. Ahora bien, si el patrón de consulta tiende a recuperar todas las columnas para un grupo de registros, entonces su uso frente a las bases de datos relacionales es contraproducente. Como en otros gestores, la relación de registros debe ser resuelta a nivel de aplicación.

El cuarto tipo de base de datos NoSQL son los almacenes de grafos. Mediante estos es posible representar múltiples relaciones complejas atravesando una estructura de entidades en forma de red. Un grafo se compone de un conjunto de nodos conectados por arcos. Tanto nodos como arcos pueden presentar múltiples etiquetas y propiedades, estas últimas en forma de clave-valor. Dos nodos pueden estar relacionados por más de un arco, estando estos dirigidos. La Figura 3-10 representa un grafo donde distintas personas se compinchan para cometer fraude en el seguro del coche, simulando lesiones en accidentes de tráfico amañados. Como se puede observar, la trama se detecta al revelarse las relaciones existentes entre los diversos actores del montaje.

Estas mismas estructuras se pueden emplear para detectar grupos de influencia en redes sociales o para diseñar recomendaciones de compra. Existen lenguajes de interrogación propios para estos sistemas como Gremlin, parte del proyecto Apache TinkerPop.

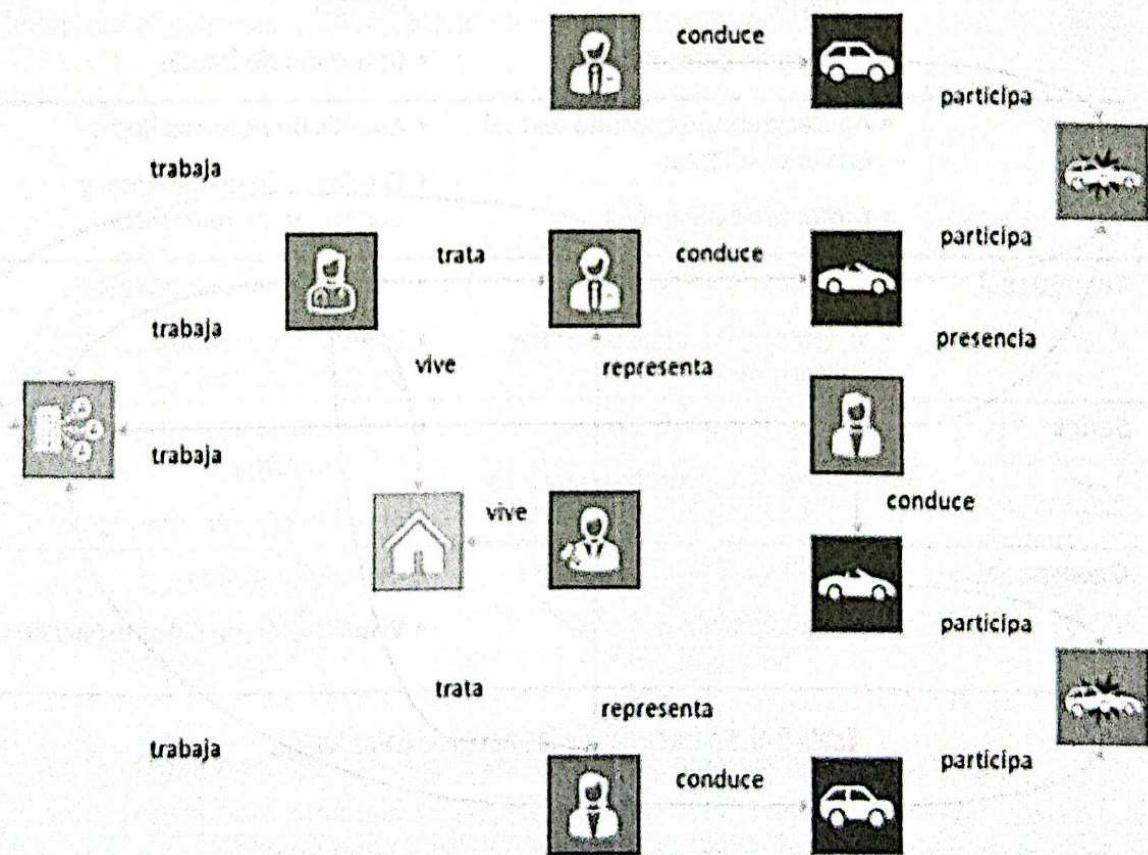


Figura 3-10. Grafo de un caso de fraude involucrando conductores, testigos, médicos y abogados.

Categoría	Casos de uso
Clave-Valor 	<ul style="list-style-type: none"> Gestión de sesiones Configuraciones y preferencias de usuario Almacenamiento de datos en memoria caché
Documentos 	<ul style="list-style-type: none"> Gestión de contenidos Intercambio de datos mediante JSON Análisis de datos de uso en la web (<i>clickstream analysis</i>) Análisis de redes sociales
Columnas 	<ul style="list-style-type: none"> Agregación de datos (IoT, logs, etc.) Sistemas de recomendación Historial de eventos Registro de actividades
Grafos 	<ul style="list-style-type: none"> Análisis de dependencia e impacto Gestión de datos maestros Análisis de redes sociales y clientes Detección de fraude
Índices 	<ul style="list-style-type: none"> Aplicaciones de consulta textual en varios idiomas Motores de búsqueda Análisis de registros (<i>logs</i>) Catalogación documental y asignación de relevancia
Tiempo real 	<ul style="list-style-type: none"> Sistemas de alerta Sincronización de datos entre aplicaciones Procesado en tiempo real Inferencia de modelos predictivos en tiempo real
Series 	<ul style="list-style-type: none"> Monitorización de eventos Control de mercados financieros Observancia de métricas (<i>observability</i>) Almacenamiento intermedio
Geoespaciales 	<ul style="list-style-type: none"> Cálculo de rutas Geolocalización Geosegmentación Visualización de datos espaciales

Tabla 3-9. Aplicaciones de las bases de datos NoSQL.

Las bases de datos de índices tienen la capacidad de hacer consultas en otros almacenes, indexando su contenido. Se utilizan típicamente para realizar búsquedas textuales basadas en palabras clave, obteniéndose una relación de documentos ordenada según relevancia. Sus orígenes pueden ser sistemas de archivos o bases de datos clave-valor, expandiendo en este último caso las capacidades de consulta. Es habitual que incorporen capacidades de **procesamiento del lenguaje natural (NLP, Natural Language Processing)**, permitiendo búsqueda por aproximación del término, sentimiento, conceptos, etc.

Las bases de datos en tiempo real y de series temporales comparten una serie de características comunes. Ambas trabajan con información fechada, tratando secuencias de eventos, lo que les permite crecer fácilmente de forma horizontal. En el caso de las series temporales, la ingestión suele ser a intervalos de tiempo regulares, siendo el objetivo el análisis retrospectivo y el pronóstico, aunque no necesariamente en tiempo real. Las bases de datos en tiempo real sí requieren una alta velocidad de procesamiento, incluyendo cargas más complejas en vivo, donde el foco no está necesariamente en la componente temporal. Permiten realizar análisis multidimensionales, con una alta capacidad de agregación dinámica de métricas.

Por último estarían las bases de datos geoespaciales. Más que constituir una tecnología diferente, suelen ser extensiones de bases de datos relacionales o de documentos, equipadas con tipos geométricos, funciones para la manipulación de estos, e índices para soportar consultas espaciales. Aplicaciones como el cálculo de rutas, o consultas que tienen en cuenta la adyacencia, el solapamiento o la distancia entre geometrías pueden ser implementadas mediante estos gestores.

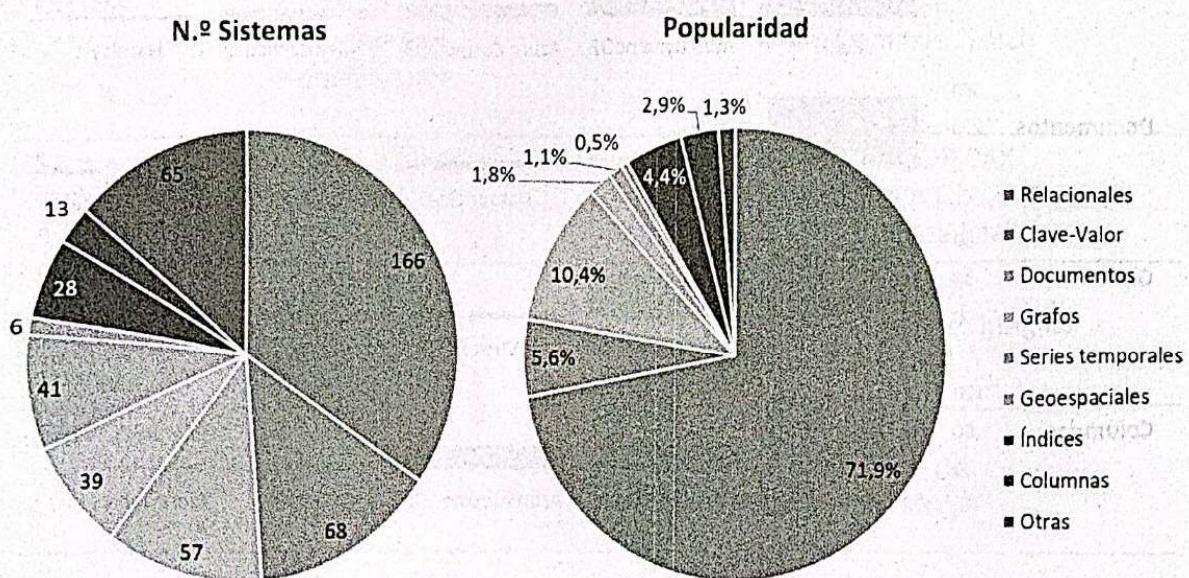


Figura 3-11. Distribución de bases de datos por categoría en febrero de 2023.

Nota. Datos extraídos de *DBMS popularity broken down by database model*, DB-Engines, 2023, Solid IT (https://db-engines.com/en/ranking_categories).

La Tabla 3-9 contiene los casos de uso típicos para cada categoría de base de datos. Tan solo insistir en la idea de escoger la mejor opción para cada tipo de problema, huyendo de soluciones únicas para todo.

3.4.3 Software y servicios de bases de datos NoSQL

El mercado de las bases de datos sigue dominado a día de hoy por los gestores relacionales. La Figura 3-11 muestra la distribución del número de sistemas existentes en el mercado de cada categoría, incluyendo aquellos de código libre y de licencia comercial, así como su popularidad según sus menciones, comentarios y ofertas de trabajos en redes sociales a nivel mundial para febrero de 2023. Si bien a partir de estos datos no es posible inferir el posicionamiento en cuanto a ventas o base instalada, es esperable cierta correlación.

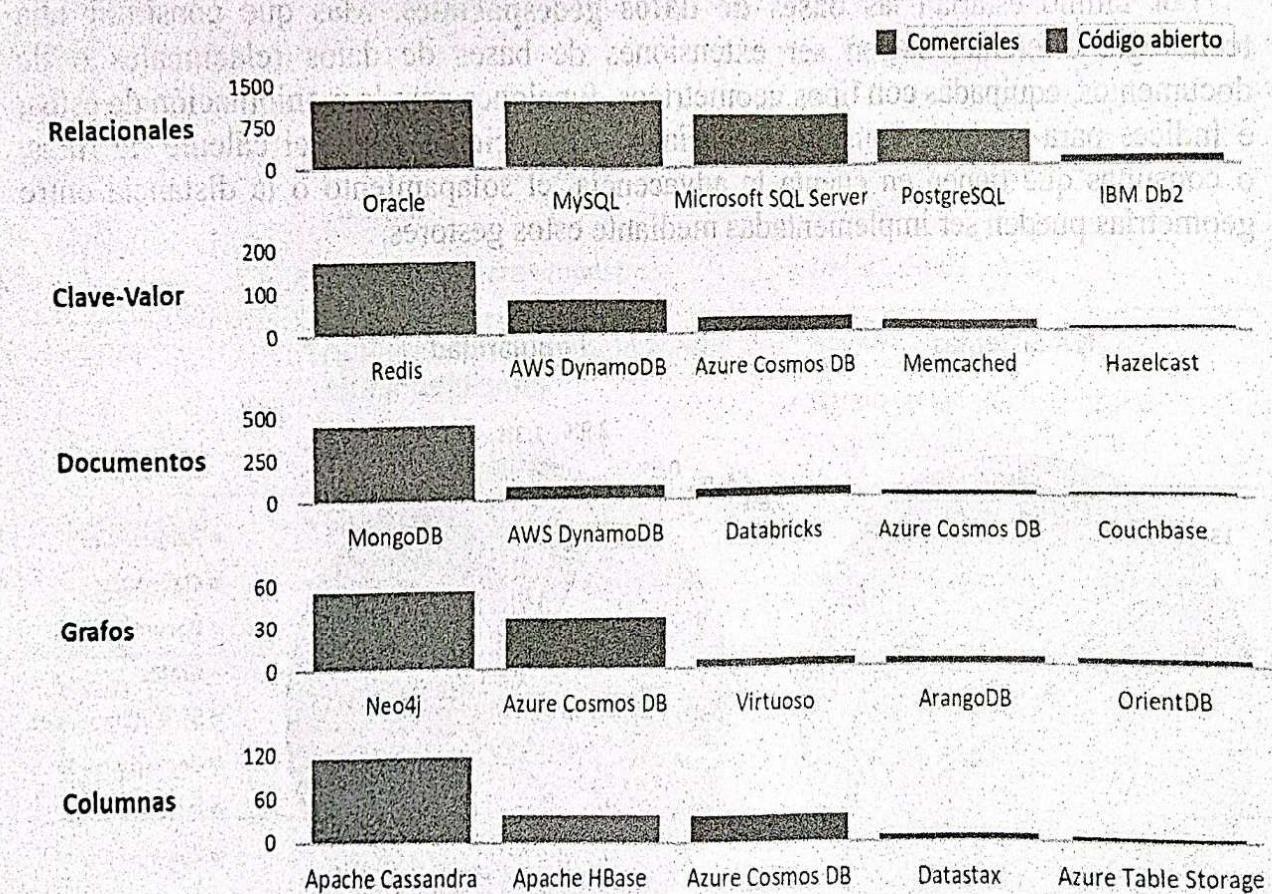


Figura 3-12. Popularidad de los sistemas de bases de datos por categoría en febrero de 2023.

Nota. Datos extraídos de *DB-Engines ranking*, DB-Engines, 2023, Solid IT (<https://db-engines.com/en/ranking>).

La Tabla 3-10 enumera el *software* y los principales servicios en la nube de para los gestores NoSQL. En el caso de los servicios, hay que destacar que los principales proveedores tienden a ofrecer gestores que soportan diferentes modelos de almacenamiento y compatibilidad con otros gestores, como es el caso de **Google Bigtable** o **Azure Cosmos DB**.

Tipo	Software	Servicios en la nube
Clave-Valor 	<ul style="list-style-type: none"> Redis etcd Apache Accumulo 	<ul style="list-style-type: none"> AWS DynamoDB Azure Cosmos DB Google Bigtable
Documentos 	<ul style="list-style-type: none"> MongoDB Apache CouchDB Couchbase 	<ul style="list-style-type: none"> AWS DocumentDB Azure Cosmos DB Google Firestore
Columnas 	<ul style="list-style-type: none"> Apache HBase Apache Cassandra ScyllaDB 	<ul style="list-style-type: none"> AWS Keyspaces Azure Cosmos DB Google Bigtable
Grafos 	<ul style="list-style-type: none"> Neo4J JanusGraph AllegroGraph 	<ul style="list-style-type: none"> AWS Neptune Azure Cosmos DB
Índices 	<ul style="list-style-type: none"> Apache Lucene Apache Solr Elasticsearch 	<ul style="list-style-type: none"> AWS CloudSearch Azure Cognitive Search
Tiempo real 	<ul style="list-style-type: none"> Apache Druid Apache Pinot RethinkDB 	<ul style="list-style-type: none"> AWS DynamoDB DAX Azure Cosmos DB Google Firebase Realtime
Series 	<ul style="list-style-type: none"> InfluxDB Prometheus Graphite 	<ul style="list-style-type: none"> AWS Timestream Azure Time Series Insights Google Bigtable
Geoespaciales 	<ul style="list-style-type: none"> PostGIS⁸⁹ MongoDB Couchbase 	<ul style="list-style-type: none"> AWS Aurora³⁷ Azure Cosmos DB Google BigQuery³⁷

Tabla 3-10. Software y servicios en la nube de bases de datos NoSQL.

Respecto a los sistemas de código abierto, nos encontramos con soluciones con mucha solera en el mercado. **MongoDB**, cuya primera versión es de 2009, es probablemente la referencia en cuanto a bases de datos de documentos. Lo mismo ocurre con **Redis** y **Apache Cassandra** para las de tipo clave-valor y columnas, respectivamente. Es estas últimas destaca también **Apache HBase**, un motor con capacidades en tiempo real que se ejecuta en Hadoop sobre HDFS.

La Figura 3-12 ordena los principales sistemas por popularidad para cada una de las categorías de bases de datos.

3.5 RESUMEN DEL CAPÍTULO

En este capítulo hemos planteado los principales mecanismos de almacenamiento que nos podemos encontrar en aplicaciones de *Big Data*, exponiendo sus diferencias en cuanto a modelo, despliegue y campos de aplicación.

- ▀ Las **bases de datos relacionales** constituyen una de las principales opciones a la hora de almacenar y gestionar información estructurada y tabular, siendo la plataforma empleada en los sistemas de *data warehouse*.
- ▀ Los **sistemas de archivos distribuidos**, como **Apache Hadoop**, permiten un gran crecimiento horizontal, empleando *hardware* convencional o servicios en la nube. Se caracterizan por llevar el procesamiento a donde reside el dato, soportando distintos formatos de archivo especializados. Aunque en declive, siguen siendo una opción a tener en cuenta para la implementación de *data lakes*.
- ▀ Los **almacenes de objetos en la nube** se han posicionado en los últimos años como la base de los sistemas de *data lake* y *data lakehouse*. Su bajo coste, resiliencia, acceso universal y nulo mantenimiento permiten almacenar todo tipo de datos. Sobre estos repositorios se están desarrollando capas de metadatos que permiten la optimización y la estandarización del acceso por parte de las aplicaciones.
- ▀ Las **bases de datos NoSQL** permiten utilizar repositorios especializados en función de la tipología del dato, tanto en lo referente a su generación y estructura como a la forma de consulta.
- ▀ Partimos de la base de que no existe una única solución para todas las necesidades de almacenamiento. Un entorno de *Big Data* moderno se compone de distintos gestores, especializados en modelos de datos y cargas concretas.

Una vez que somos capaces de almacenar la información, empleando para ello el sistema más adecuado, es hora de integrarla y procesarla. En el siguiente capítulo estudiaremos como podemos hacer esto, accediendo a los orígenes de los datos y transformándolos según nuestras necesidades.