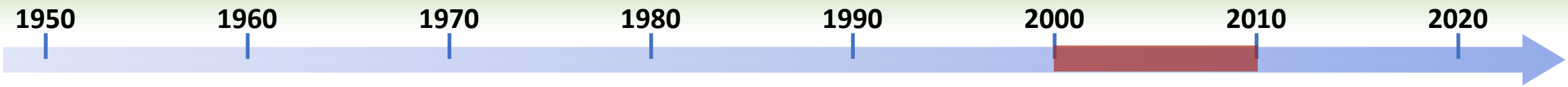
Introducción a los Sistemas NoSQL

Autor: Sergio D'Arrigo

Algo de historia…



En este período aparece la necesidad de manejar tipos de **datos no estructurados**.

- Los RDBMS comienzan a soportar datos semiestructurados (XML y JSON) y datos espaciales.
- Aumenta el uso de RDBMS **Open Source**, principalmente PostgreSQL y My SQL.

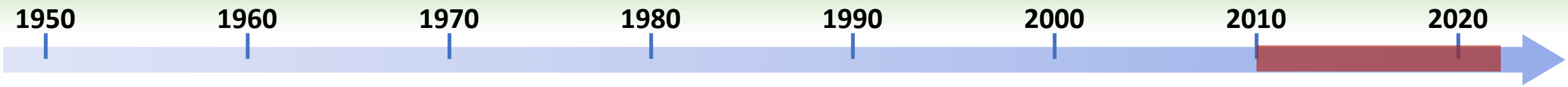
Con el crecimiento de las plataformas de redes sociales, surge la necesidad de representar conexiones entre personas y sus conexiones. Esto motiva el desarrollo de **bases de datos de grafos**.

Las aplicaciones comienzan a requerir un uso más intensivo de datos de tipos diversos, así como facilidades para el desarrollo rápido.
Esto motiva la aparición de las bases de datos NoSQL (Not Only SQL), generalmente muy livianas.

- Puntos débiles en relación a las RDBMS: falta de un lenguaje de consultas declarativo estándar, y falta de consistencia estricta (proveen consistencia eventual)
- A favor: manejan nativamente tipos de datos semiestructurados o no estructurados, escalabilidad y disponibilidad.

En la segunda parte de la década, las organizaciones comenzaron a dedicar esfuerzo al análisis de datos y data mining.
Surge también el framework map-reduce, orientado a facilitar el paralelismo en el procesamiento analítico de los datos.

Algo de historia…



Se profundiza el uso de aplicaciones Cloud. Profundización de almacenamiento y gestión de BBDD en la nube

Las BBDD NoSQL evolucionan para soportar nociones de consistencia más estricta, y para proveer lenguajes de consulta de alto nivel.

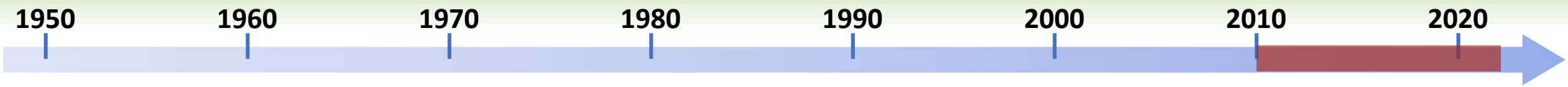
2016: última revisión (a la fecha) del estándar SQL. Actualmente cuenta con soporte para JSON, expresiones regulares, operaciones analíticas y estadísticas.

Surgen las Bases New SQL: aparecen para proveer escalabilidad, performance y cumplimiento de propiedades ACID de las transacciones.

Surgen los Data Lakes, inicialmente muchos on premise, la tendencia actual es la nube.

En relación al procesamiento analítico, aparecen
– Data Lakehouses en nube
- Bases de Datos HTAP : Para intentar resolver sobre una plataforma única el procesamiento Transaccional y Analítico

Algo de historia…



La migración a la nube no es trivial, tiene puntos positivos y desafíos por resolver.

Cuestiones: Bajo costo, alta disponibilidad, desafíos en cuanto a seguridad (sobre todo para uso gubernamental) y protección de datos personales.

La masificación de aplicaciones que hacen uso intensivo de los datos y la analítica, ponen en primer plano un conflicto no resuelto : derecho de la sociedad a saber y conocer vs derecho individual a la privacidad.

Evolución de las tecnologías de BBDD

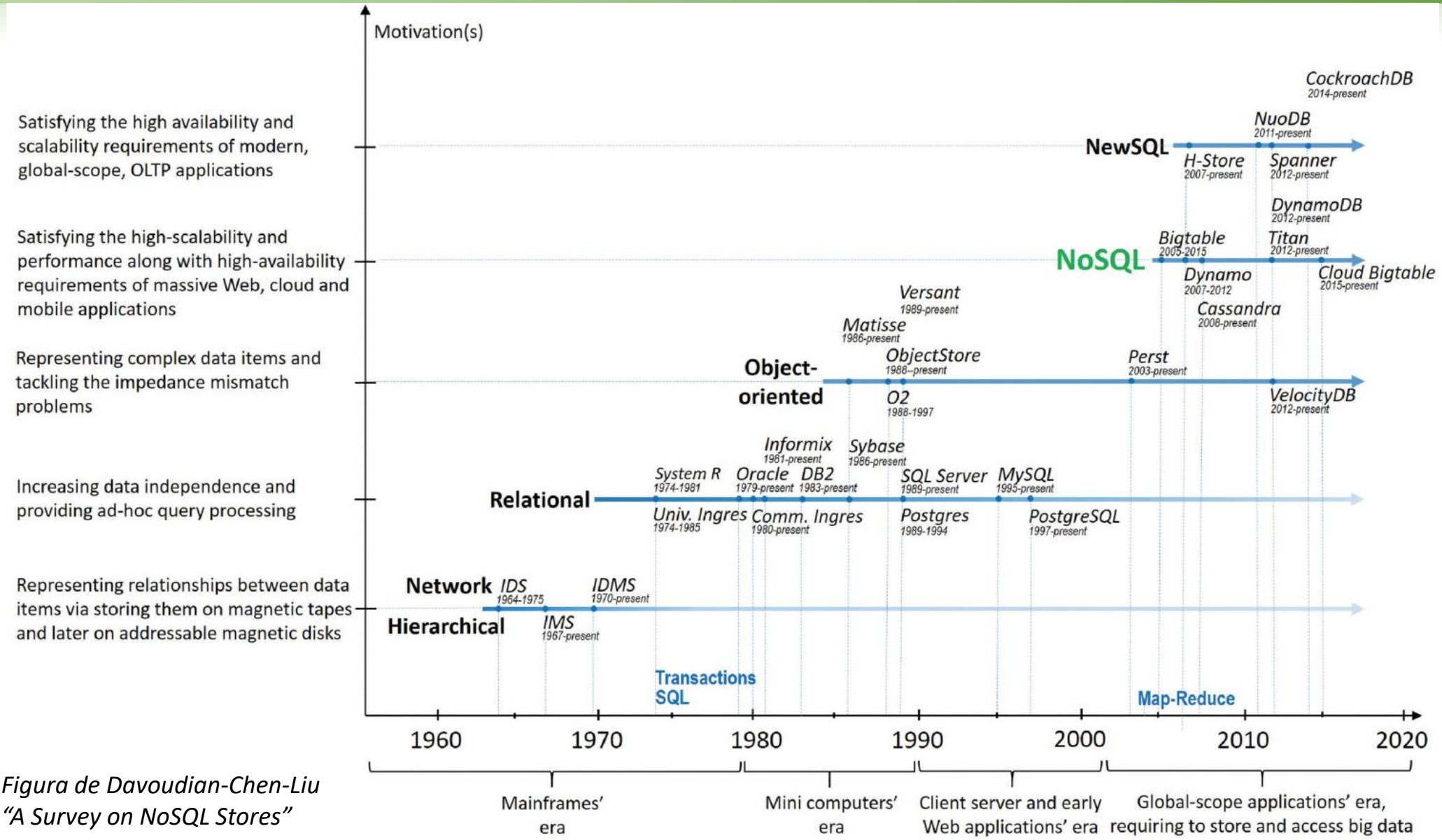


Figura de Davoudian-Chen-Liu "A Survey on NoSQL Stores"

Requerimientos para aplicaciones globales

- Desde comienzos de los 2000s, los avances tecnológicos (web, redes sociales, dispositivos móviles, IoT) derivaron en una explosión de datos estructurados, semi-estructurados y no estructurados generados por aplicaciones de alcance global.
- Los principales requerimientos de estas aplicaciones para los Sistemas de Bases de Datos son:
  - Escalabilidad
  - Alta disponibilidad
  - Flexibilidad del esquema de base de datos
  - Bajos costos

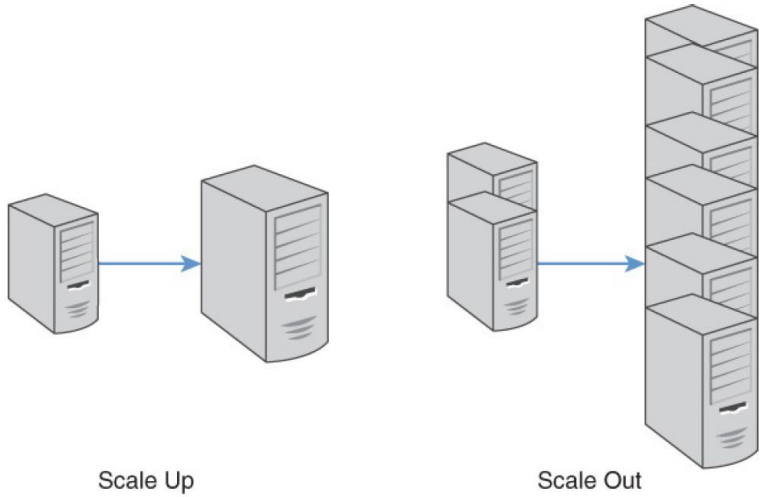


Figure 1.11 *Scaling up versus scaling out.*

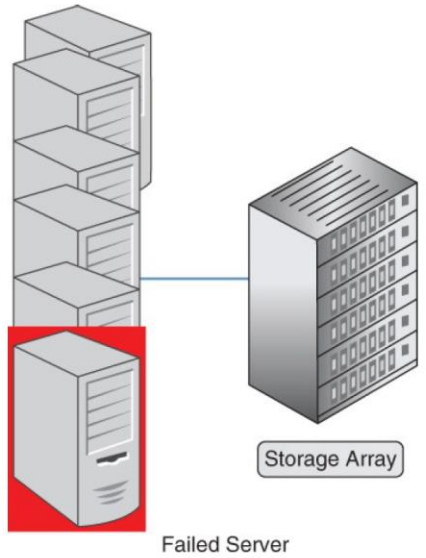


Figure 1.12 *High-availability NoSQL clusters run multiple servers. If one fails, the others can continue to support applications.*

Figuras de *Sullivan, "NoSQL for Mere Mortals"*

Requerimientos para aplicaciones globales

- Los SGBD relacionales no se adaptaban bien a estos requerimientos y necesidades, fundamentalmente por:
  - Ofrecen una cantidad importante de funcionalidades innecesarias
  - El modelo de datos estructurado y con esquema fuerte es muy restrictivo

- Esto dio lugar a **dos tendencias**:
  - Evolución de los SGBD relacionales evoluciones a modelos más flexibles y nuevas funcionalidades
  - Surgimiento de las Bases de Datos NoSQL

- Nos centraremos en la segunda

Requerimientos para aplicaciones globales

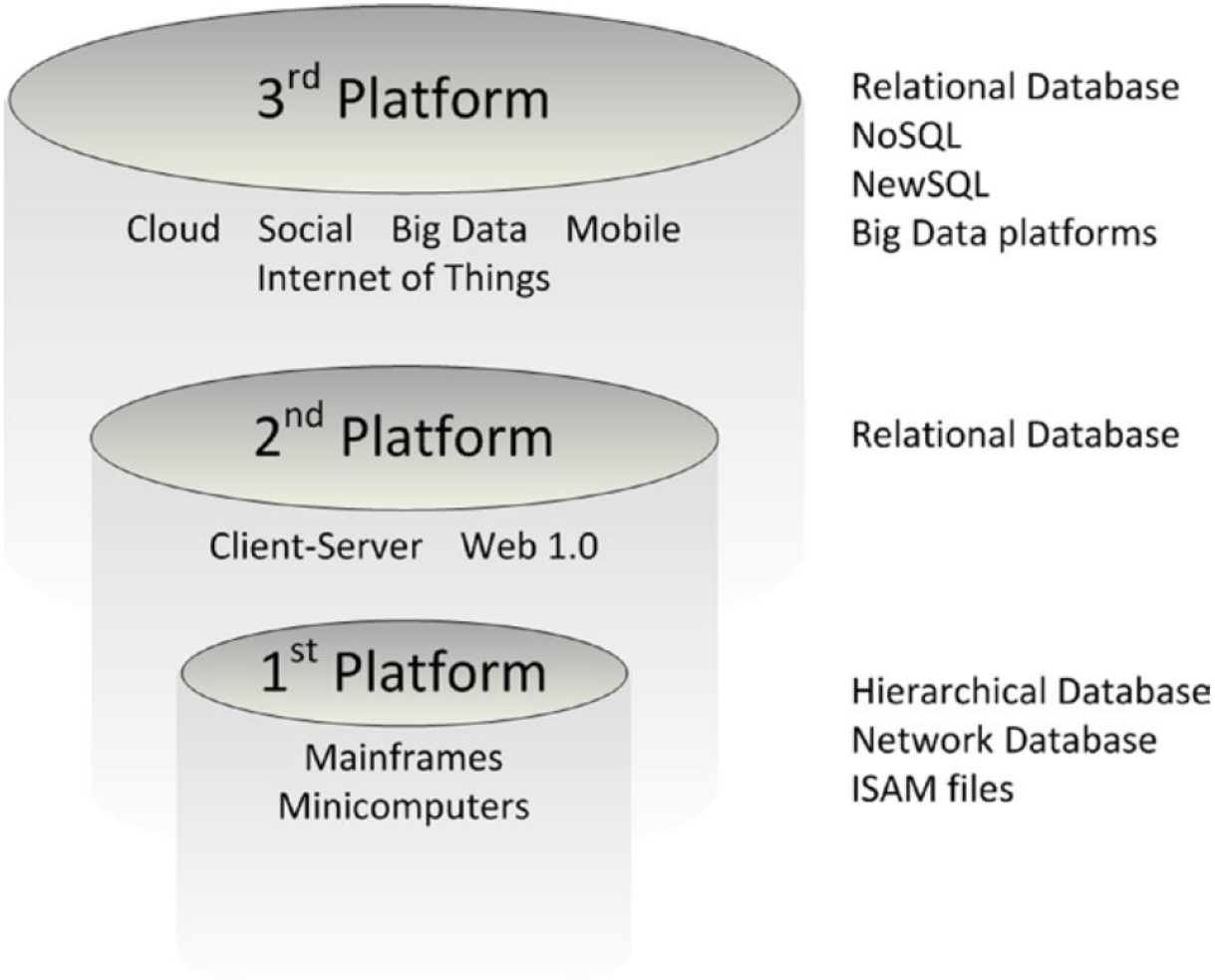


Figure 1-6. *IDC's "three platforms" model corresponds to three waves of database technology*

Figura de *Harrison, "Next Generation Databases "*

Características comunes de las bases NoSQL

Esquemas Flexibles

- Utilizan modelos de datos más flexibles, incluso sin esquema (schema-less)

Arquitectura "Shared Nothing"

- Cada nodo es independiente y autosuficiente, no comparte disco ni memoria ni procesador

Elasticidad

- Capacidad de expansión dinámica, fácil escalamiento horizontal agregando nuevos nodos

Sharding

- Los datos se particionan horizontalmente entre diferentes nodos.

Disponibilidad y Replicación Asincrónica

- Los datos están replicados, con un esquema asincrónico que se completa lo antes posible.

Propiedades BASE

- Cumples las propiedades BASE, y relajan las propiedades ACID

Propiedades BASE

BASE es un enfoque que valora la **disponibilidad** por sobre la **consistencia** de las operaciones.

Basically Available

- El sistema siempre parece funcionar
- Garantiza disponibilidad en términos del teorema CAP.
- Los datos se distribuyen en múltiples sistemas de almacenamiento con alto grado de replicación para lograr alto grado de disponibilidad

Soft State

- En un momento determinado las copias de un data ítem podrían no tener el mismo valor en todas las réplicas.
- El estado del sistema puede cambiar a lo largo del tiempo, aún sin recibir nuevos inputs, a causa de la Consistencia Eventual.

Eventual Consistency

- Las actualizaciones de datos se pueden propagar de manera asincrónica.
- Durante un cierto tiempo se pueden tener diferentes versiones de un data ítem, pero en algún momento posterior debería quedar consistente.

Teorema CAP

- Cualquier sistema distribuido con datos replicados, **sólo puede garantizar dos de las tres** siguiente propiedades deseables:

Consistencia

- Entre copias replicadas

Disponibilidad

- Del sistema para operaciones de lectura y escritura

Tolerancia a particiones

- Cuando los nodos están divididos debido a fallas de red

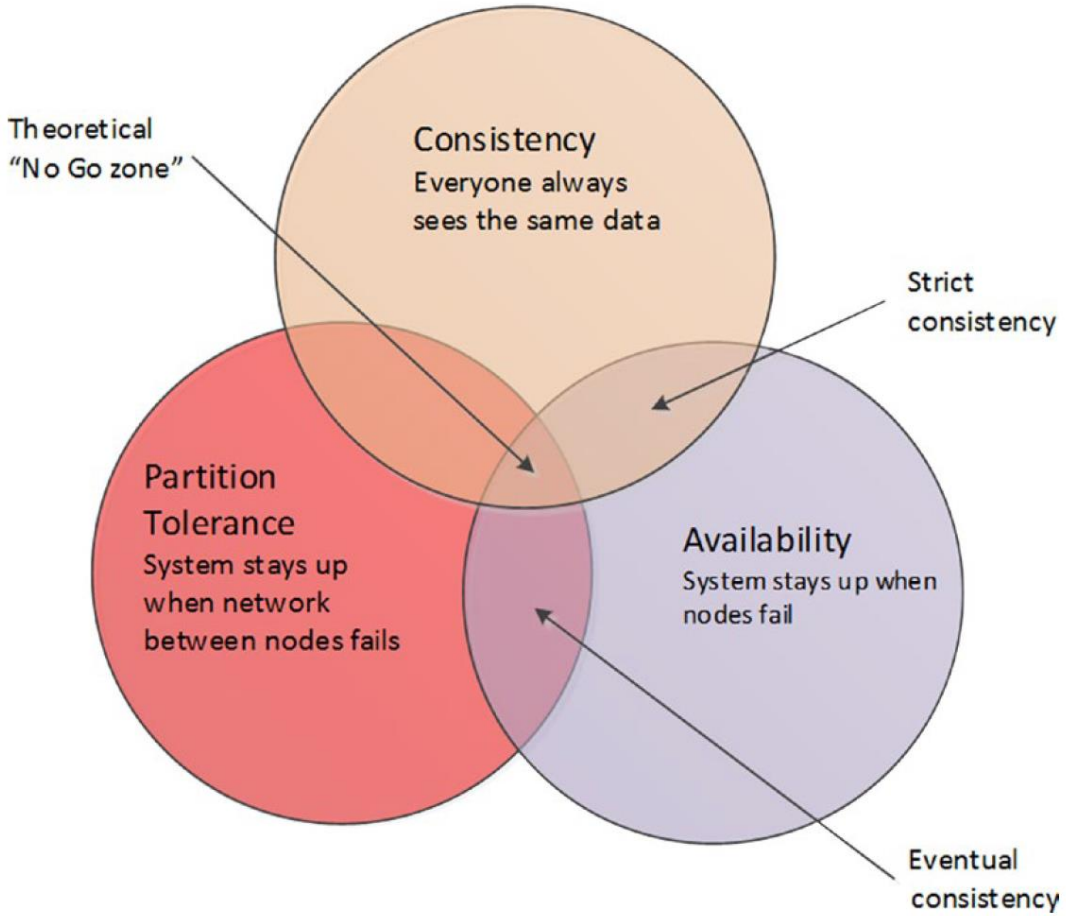


Figure 3-4. *Dynamo and ACID RDBMS mapped to CAP theorem limits*

Figura de *Harrison, "Next Generation Databases "*

Notar que el concepto "consistencia" aquí tiene un significado diferente que en ACID

Teorema CAP

- El gráfico presenta una partición:

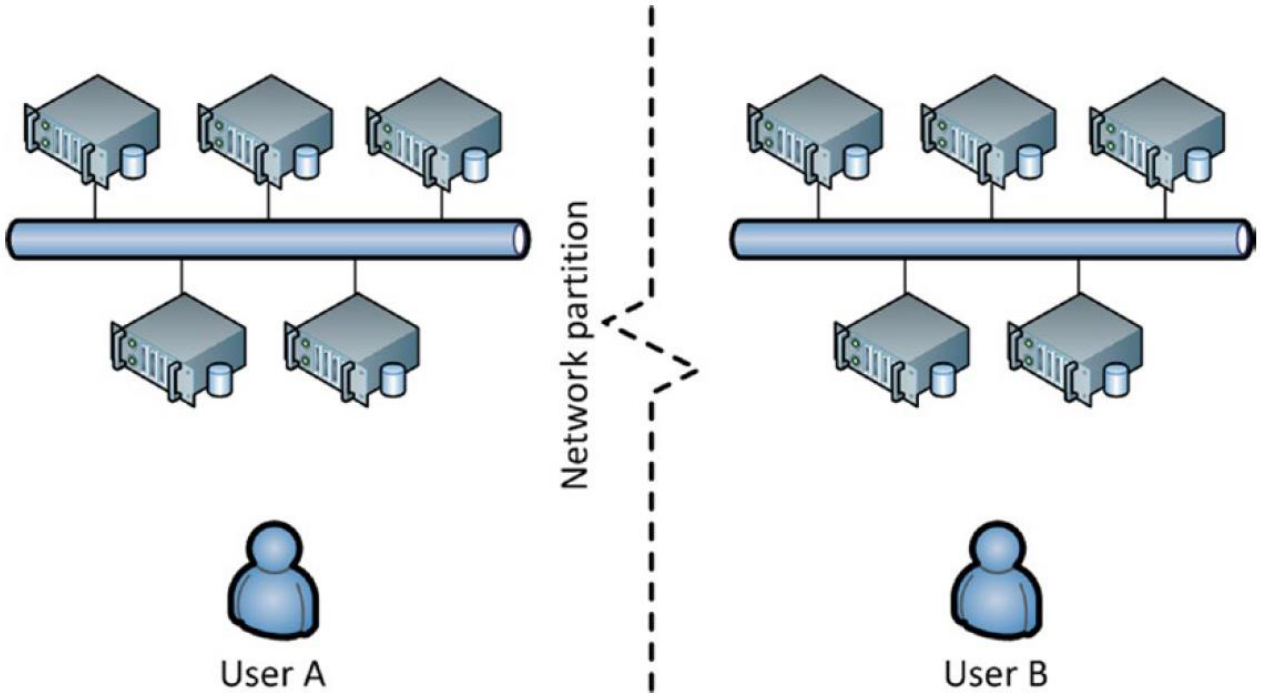


Figure 3-3. *Network partition in a distributed database application*

Figura de *Harrison, "Next Generation Databases"*

Ante una partición, hay dos opciones:

- ❑ Permitir a cada usuario acceder a vistas diferentes de los datos (renunciando a la consistencia)
- ❑ Inhabilitar una de las particiones y desconectar a uno de los usuarios (renunciando a la disponibilidad)

Recién cuando los nodos se comuniquen, podríamos lograr consistencia y disponibilidad (aunque ya en ese caso no tendríamos partición)

Notar que la decisión entre consistencia y disponibilidad sólo se tendría que tomar ante la ocurrencia de una partición.

Interpretación CAP

Según el propio Brewer, quien realizó la formulación original:

- CAP prohíbe solo una pequeña parte del espacio de diseño: la disponibilidad y la consistencia perfectas en presencia de particiones, **las cuales son raras**.
- El objetivo moderno de CAP debería ser maximizar las combinaciones de consistencia y disponibilidad que tengan sentido para la aplicación específica.
- Un enfoque de este tipo requiere la gestión de las particiones, mediante 3 pasos básicos:
  - ❑ Detección del inicio de la partición
  - ❑ Operación en "modo partición", posiblemente restringiendo algunas operaciones
  - ❑ Recuperación de la partición, una vez restablecidas las comunicaciones

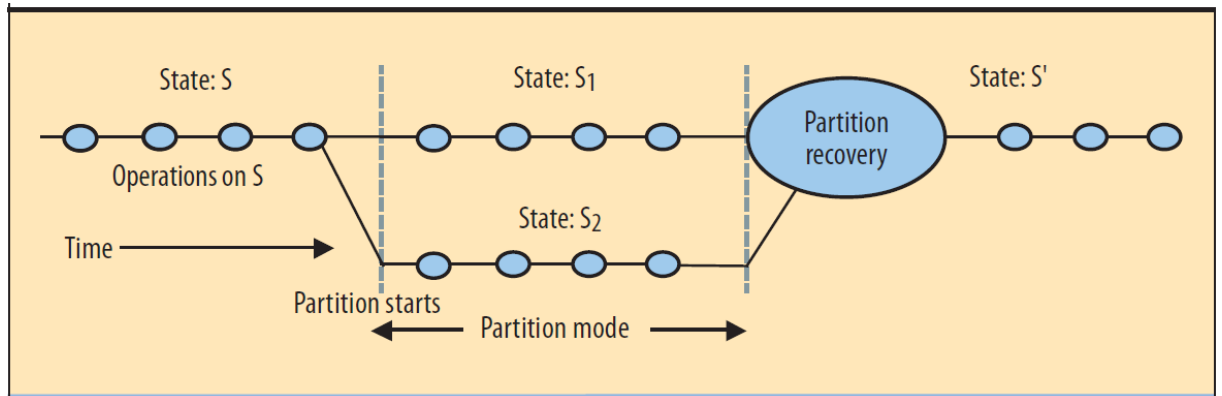


Figure 1. The state starts out consistent and remains so until a partition starts. To stay available, both sides enter partition mode and continue to execute operations, creating concurrent states S_1 and S_2, which are inconsistent. When the partition ends, the truth becomes clear and partition recovery starts. During recovery, the system merges S_1 and S_2 into a consistent state S′ and also compensates for any mistakes made during the partition.

Figuras de *Brewer , "CAP Twelve Years Later: How the 'Rules' Have Changed"*

Relación CAP - Latencia

- En la interpretación clásica del teorema CAP **se ignora la latencia**.
- En la práctica, **latencia y particiones tienen gran relación**.
- La detección de una partición, depende del tiempo de espera para las comunicaciones.

- Reintentos de comunicaciones para lograr consistencia, dilatan la detección, o evitan caer en situación de partición.
- Con reintentos ilimitados ante fallas, tendremos consistencia pero no disponibilidad.

- Los diseñadores deberán establecer límites temporales acordes a los tiempos de respuesta esperados.
  - Con límites muy pequeños, el sistema entrará más frecuentemente en “modo partición”, aún cuando la red esté muy lenta y no particionada.
  - Con límites muy altos, la latencia será mayor.

Relación CAP - Latencia

- Disponibilidad y latencia son dos caras de la misma moneda. Un sistema indisponible provee una latencia extremadamente alta.
- Hay un compromiso entre consistencia y latencia, aún cuando no ocurran particiones.

- El requerimiento de alta disponibilidad implica que el sistema replique datos, para mitigar la posibilidad de ocurrencia de una falla en un componente.
- Hay diversas formas de replicación, cada una de ellas con diferente grado de operaciones de replicación sincrónicas y asincrónicas.

- A **mayor grado de sincronismo en la réplicas** (y por lo tanto, mayor grado de consistencia), tendremos **mayor latencia**.

Teorema PACELC

- Es una reformulación del teorema CAP.
- Surge para incorporar el concepto de latencia

- **PACELC**
  - **P**artition: **A**vailability vs **C**onsistency
  - **E**lse: **L**atency vs **C**onsistency
- Si hay una partición, el sistema debe negociar entre disponibilidad y consistencia
- Si no hay una partición, el sistema debe negociar entre latencia y consistencia

Alternativas para replicación

Alternativas principales

Las actualizaciones se envían **A TODOS LOS NODOS AL MISMO TIEMPO**.

- Hay mecanismos de preprocesamiento o consensos para lograr consistencia

Las actualizaciones de datos se envían **PRIMERO A UN LUGAR ACORDADO**.

- El lugar acordado es el nodo maestro (pueden ser diferentes según cada data-ítem)
- El maestro resuelve la(s) actualización(es), luego lo replica a las otras locaciones
- Opciones:
  - Sincrónico (mayor latencia)
  - Asincrónico (menor latencia)
  - Híbrido Sincrónico - Asincrónico

Las actualizaciones de datos se envían **PRIMERO A UN LUGAR ARBITRARIO**.

- El lugar arbitrario funciona como nodo maestro
- Primero se resuelve en el maestro, luego replica
- La diferencia con el anterior es que el nodo maestro para un data-ítem puede no ser siempre el mismo
- Puede ser sincrónico o asincrónico.
- Impacto en latencia para asegurar consistencia del orden de replicaciones

Tipos de consistencia eventual

Causal

- Asegura que la base de datos refleje el orden de realización de las actualizaciones, y las lecturas también

Leer-tus-escrituras (RYW)

- Una vez que actualizaste un data ítem, todas tus lecturas sobre el data ítem nunca recuperarán un valor previo a tu actualización.

De sesión

- Similar a leer-tus-escrituras, pero con alcance para toda la duración de la sesión

De lectura monótona

- Si un proceso lee un valor para un data ítem, cualquier lectura subsiguiente recuperará esa versión del dato o una posterior.

De escritura monótona

- Si un proceso realiza actualizaciones sucesivas de un data ítem, serán impactadas en el orden en que las realizó el proceso.

La consistencia no es un concepto binario, tiene grados, y un mismo motor puede aplicar diferentes criterios para diferentes aspectos, por ej utilizar consistencia estricta para algunos aspectos críticos y consistencia eventual para otros

ACID vs BASE

- La siguiente tabla resume las diferencias entre sistemas que adhieren a ACID y BASE

Table 2 ACID versus BASE

ACID [C+A]	BASE [A+P]
Strong consistency	Accomplishes Consistency, Atomicity and Partition tolerance "eventually"
Isolation	Availability first
Focus on “commit”	Best effort
Nested transactions	Approximate answers
Pessimistic: Force consistency at the end of Transaction	Optimistic: Accepts temporary database inconsistencies, Eventually Consistent
Difficult evolution	Simpler, Faster, Easier evolution
Suitable for Financial Portals	Suitable for non-financial web-based applications
Safe	Fast
Shared Something (Disk, Memory)	Shared Nothing
Scale UP (limited)	Scale Out (Unlimited)
Simple Code, robust database	Complex code, simple database
Single Machine	A cluster
CA	AP/CA/CP, i.e. any 2 out of 3.
Scale Vertically	Scale Horizontally
SQL	Custom APIs
Full Indexes	Indexing is mostly on Keys

Tabla de Ganesh Chandra
“BASE analysis of NoSQL database”

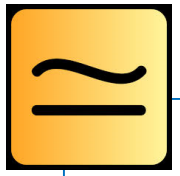
Categorías de bases de datos NoSQL

Orientadas a agregaciones

- Recopilan información de varios nodos en el cluster de red
- Se pueden utilizar Map-Reduce
- Son útiles si las mismas agregaciones se utilizan frecuentemente
- Modelos: Clave-Valor, Familia de columnas, Orientada a documentos

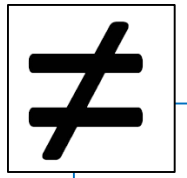
No orientadas a agregaciones

- Utilizan relaciones de manera intensiva
- Son más compatibles con las propiedades ACID
- Son una buena opción para consultas complejas
- Modelos: Grafos



Principal similitud

Ambas categorías son schema-less



Principal diferencia

Las bases de datos orientadas a agregaciones dividen las relaciones

Relación escalabilidad vs complejidad

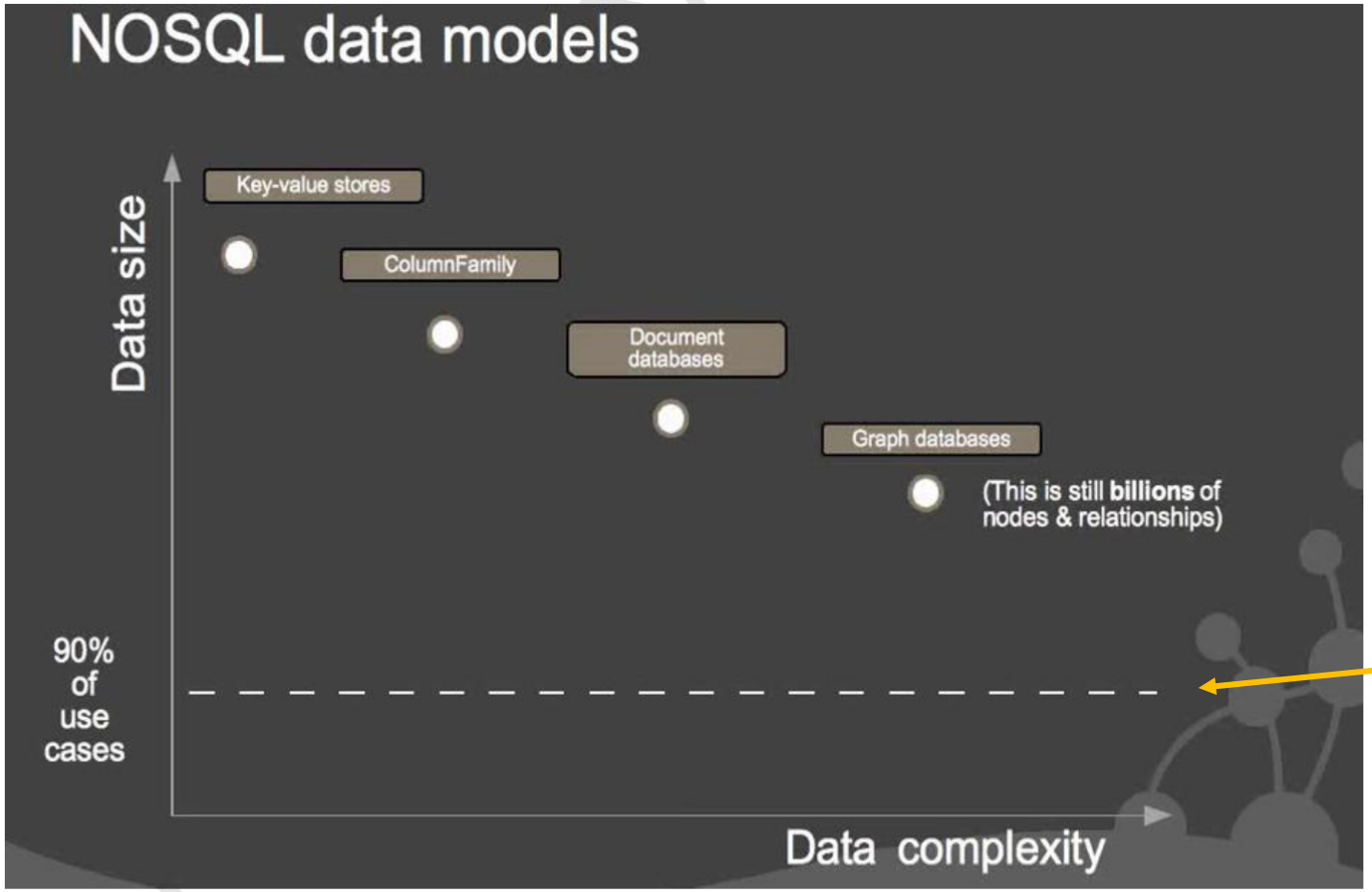


Figure 1. NoSQL databases Data size and Data Complexity Analysis graph

- **Clave-valor** se adapta mejor para aplicaciones de gran volumen y baja complejidad de datos
- **Grafos** se adapta mejor para datos complejos
- **Documentos** da un buen balance entre ambos

Si bien son datos de 2015 y la proporción puede haber variado, se puede apreciar que la gran mayoría de los casos de uso no requieren volumen tan intensivo.

Figura de Ganesh Chandra
"BASE analysis of NoSQL database"

Bases de Datos Clave-Valor

- Es el modelo más simple y popular
- Se enfocan en alta performance, disponibilidad y escalabilidad.
- Los datos se representan como un par Clave – Valor

(a). Using a basic key-value data model for a health management system

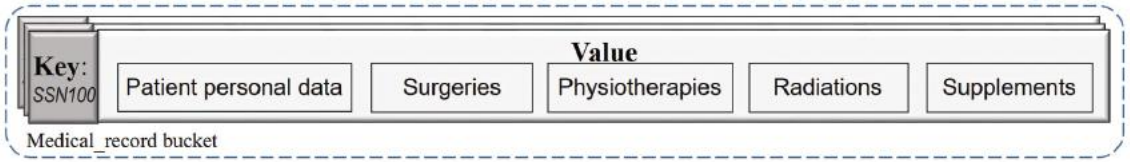


Figura de Davoudian-Chen-Liu "A Survey on NoSQL Stores"

- La clave puede ser simple o estructurada, y puede ser generada por el sistema o por la aplicación.
- No tienen esquema, el valor representa un dato de tipo, estructura y tamaño arbitrarios.
- Es unívocamente identificado por la clave indexada.
- En general, la base de datos no soporta consultas basadas en el contenido de los datos, lo debe implementar la aplicación cliente
- Ejemplos: Redis, Oracle NoSQL, Yahoo Pnuts, Amazon DynamoDB

Bases de Datos de Familia de Columnas

- Los datos están representados de una forma tabular de filas y familias de columnas.
- Una familia de columnas se compone de un arbitrario número de columnas que están relacionadas entre sí y se acceden frecuentemente juntas.
- El esquema es flexible, y se pueden agregar o quitar columnas en tiempo de ejecución.
- Una columna (celda) tiene un nombre, y un valor con una estructura simple o compleja
- Usualmente permiten almacenar **versiones** de los valores de cada celda, con un timestamp. Los valores se recuperan mediante la tupla *<clave-fila, clave-columna, timestamp>*
- Suelen permitir familias de columnas anidadas o supercolumnas

(b). Using the wide-column data model for the Facebook Inbox Search

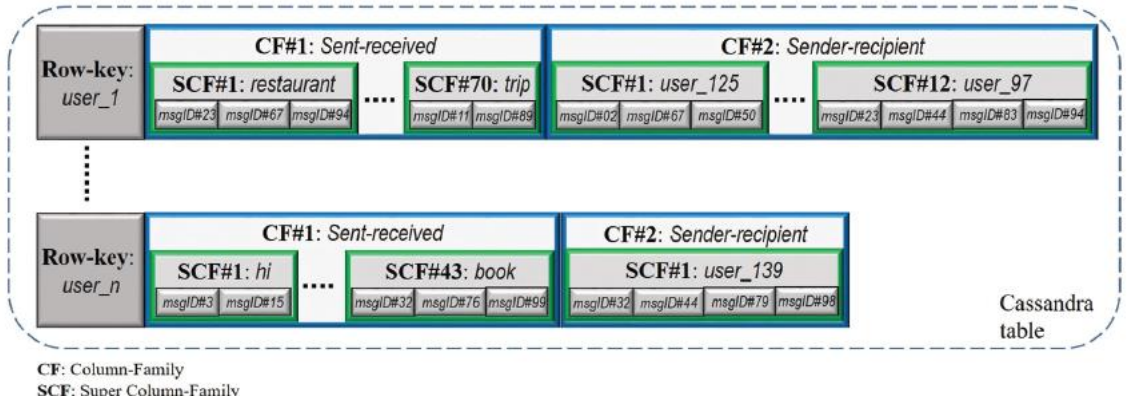


Figura de Davoudian-Chen-Liu "A Survey on NoSQL Stores"

Bases de Datos de Familia de Columnas

- Los datos pueden ser particionados eficientemente por filas y por familias de columnas, de mucha utilidad para datos masivos
- Ejemplos: Apache Cassandra, Apache HBase, Google Cloud Bigtable

Bases de Datos Orientadas a Documentos

- Almacena colecciones de documentos, en algún formato **estándar semiestructurado**: XML, JSON, BSON
- Se podría ver como una extensión el modelo clave-valor.

(c). Using the document data model for the McGraw-Hill Education

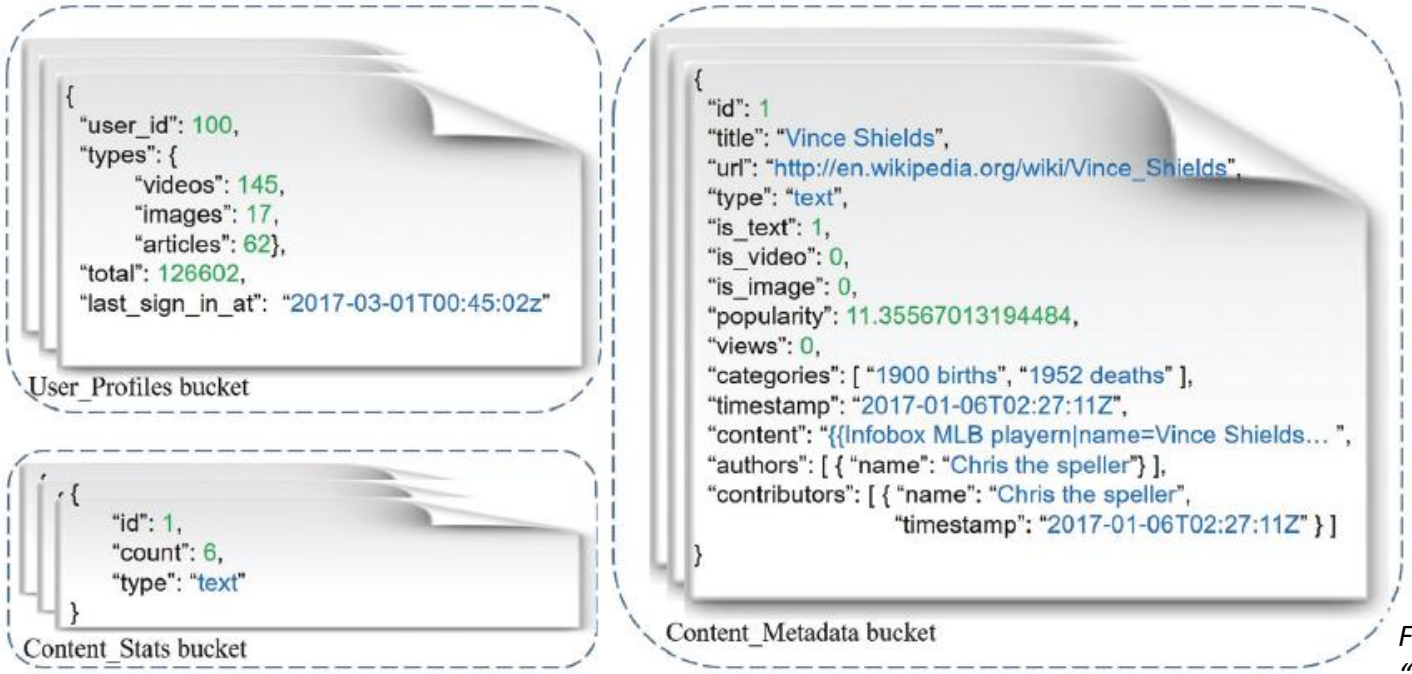


Figura de Davoudian-Chen-Liu "A Survey on NoSQL Stores"

- Soporta índices y funcionalidades de búsqueda basadas en los nombres de los atributos y sus valores
- Tiene esquema flexible, se pueden agregar o quitar atributos a los documentos en tiempo de ejecución.

Bases de Datos Orientadas a Documentos

- Algunos motores permiten buckets o colecciones, que agrupan documentos de la misma categoría de información (cada documento podría tener un esquema diferente)
- Permiten consultas para recuperar datos de un documento sin necesidad de recuperar el documento completo
- Un documento podría embeber una relación uno a muchos (por ej, factura con sus ítems), aunque esto podría implicar duplicación de datos y las anomalías de redundancia conocidas.
- Ejemplos representativos: MongoDB, Amazon DynamoDB, Couchbase

Bases de Datos Orientadas a Documentos

```
{
    "ID": "22222",
    "name": {
        "firstname: "Albert",
        "lastname: "Einstein"
    },
    "deptname": "Physics",
    "children": [
        {"firstname": "Hans", "lastname": "Einstein" },
        {"firstname": "Eduard", "lastname": "Einstein" }
    ]
}
```

Figure 8.1 Example of JSON data.

```
<purchase_order>
    <identifier> P-101 </identifier>
    <purchaser>
        <name> Cray Z. Coyote </name>
        <address> Route 66, Mesa Flats, Arizona 86047, USA </address>
    </purchaser>
    <supplier>
        <name> Acme Supplies </name>
        <address> 1 Broadway, New York, NY, USA </address>
    </supplier>
    <itemlist>
        <item>
            <identifier> RS1 </identifier>
            <description> Atom powered rocket sled </description>
            <quantity> 2 </quantity>
            <price> 199.95 </price>
        </item>
        <item>
            <identifier> SG2 </identifier>
            <description> Superb glue </description>
            <quantity> 1 </quantity>
            <unit-of-measure> liter </unit-of-measure>
            <price> 29.95 </price>
        </item>
    </itemlist>
    <total_cost> 429.85 </total_cost>
    <payment_terms> Cash-on-delivery </payment_terms>
    <shipping_mode> 1-second-delivery </shipping_mode>
</purchase_order>
```

Figure 8.2 XML representation of a purchase order.

Figuras de *Silberschatz-Korth-Sudarshan "Database System Concepts "*

Bases de Datos de Grafos

- A diferencia de las anteriores, se centra en representar **entidades y relaciones**.
- Está basado en el modelo teórico de grafos.
- Los nodos representan entidades y los arcos representan interrelaciones.
- Los nodos y los arcos pueden ser **etiquetados** para indicar el tipo de entidad o interrelación que representan, y también pueden tener **datos asociados** (propiedades).

- Los grafos representados pueden ser dirigidos o no. En este último caso, todas las interrelaciones son simétricas.
- Suelen cumplir ACID
- Ejemplo representativo: Neo4j

(d). Facebook's social network uses the graph data model. As an illustration, suppose David along with his friend Sara visit the Eiffel Tower. David uses his cellphone to record this visit by 'checking in' to the Eiffel Tower and tagging Sara to let other friends know that she is also there. Jack writes a comment on this and John likes it.

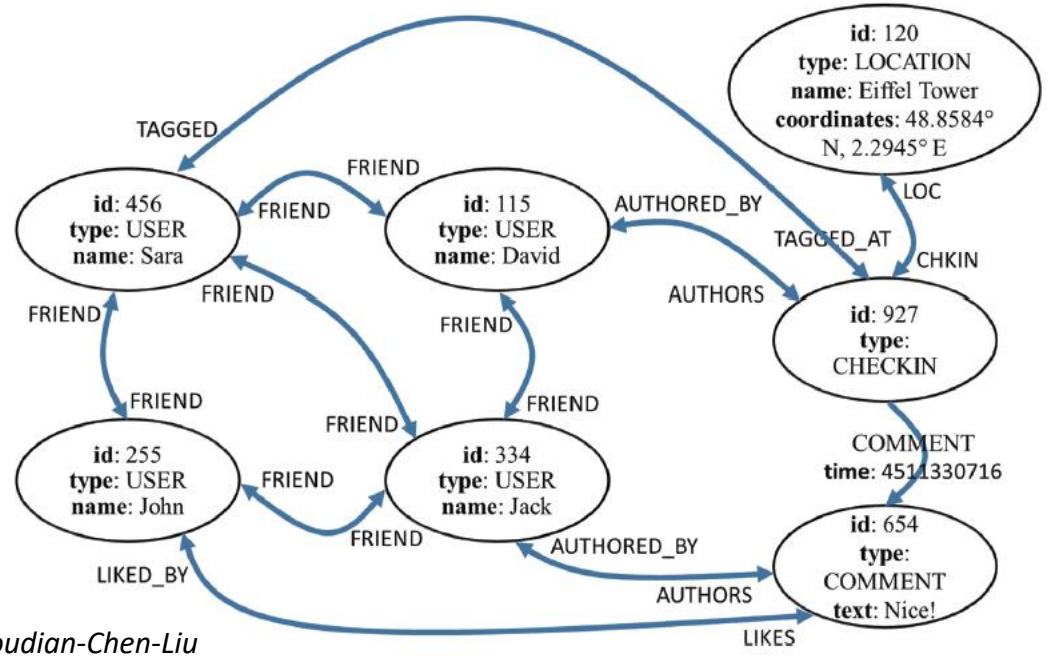
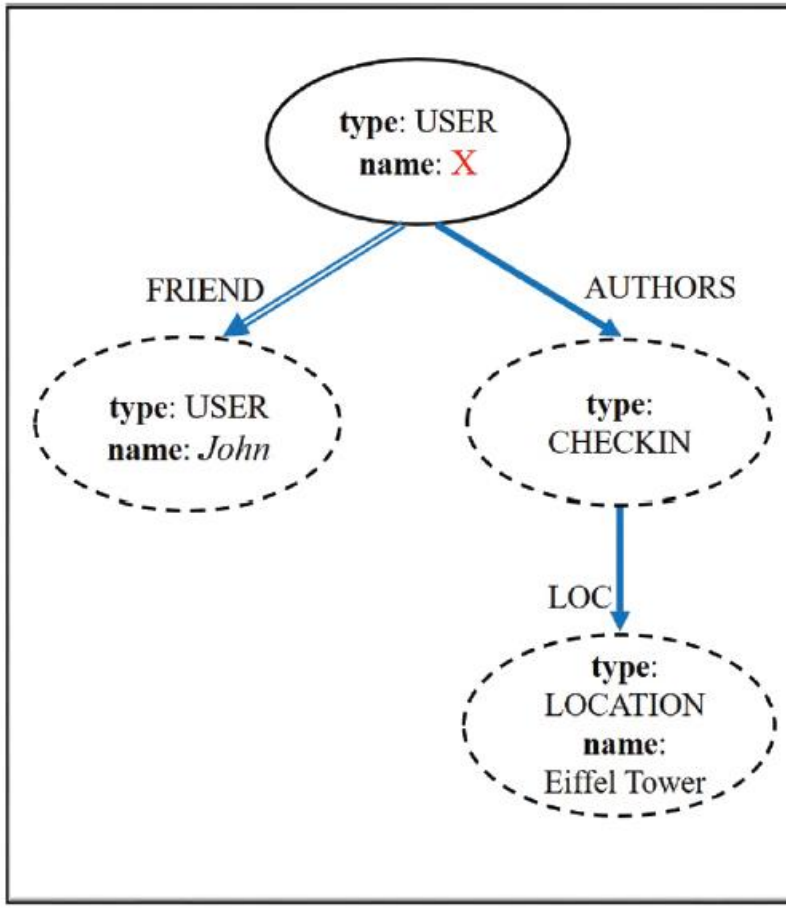


Figura de Davoudian-Chen-Liu "A Survey on NoSQL Stores"

Bases de Datos de Grafos

- Implementan **operaciones que facilitan la navegación** en el grafo.
- Un ejemplo de una consulta



(a) Tree-shaped pattern

- **MATCH** (:USER {name = "John"}) <- [:FRIEND*] - (x:USER) - [:AUTHORS] -> (:CHECKIN) – [:LOC] -> (:LOCATION {name = "Eiffel Tower"})
- **RETURN** x.name;

The **MATCH** clause specifies the graph pattern, and the **RETURN** clause indicates the output variables. Nodes are written inside **"( )"** parenthesis, symmetric edges inside "-[]-", outgoing edge of the LHS node inside "-[]->", incoming edge of the LHS node inside "<-[]-", and labels in nodes/edges are after the separation symbol **":"**. Accordingly, **(x:USER) - [:AUTHORS]** -> represents a node x with the label "USER" and one of its outgoing edges with label "AUTHORS".

Filter properties can be specified after nodes/edges within **"{}"**. Accordingly, **(:USER {name="John"})** represents a node with label "USER" with a property "name" that has a value "John". This query starts from a user node and then matches against all outgoing friend edges in the network of friends to any depth by using **"*"** symbol in **[:FRIEND*]**.

(b) Corresponding pattern query in Cypher

Figura de Davoudian-Chen-Liu "A Survey on NoSQL Stores"

Bases de datos multimodelo

- En una misma plataforma implementan varios de los modelos NoSQL vistos.
- Ejemplos representativos: ArangoDB, OrientDB

Resumen comparativo de los modelos

Table 2. Comparison of NoSQL Data Models

	Fit scenario(s)	Strength(s)	Limitation(s)
Key-value	Objects are only accessed via a single *Key*, object caching, and where objects are not related.	Scalable, a very fast random access via *Key*, and ease of data partitioning	The responsibility of applications for the modeling of *values*, the indexing and querying of objects just by their *Keys*, and the user needs to have the key of an object in order to query it.
Wide-column	Batch-oriented parallel processing of large aggregated datasets	With regard to query workload, a hierarchy of aggregates, such as column-families, are designed that, in turn, increase the performance of queries. A suitable model for storing huge amounts of data, as it can be efficiently partitioned horizontally (by rows) and vertically (by column-families).	Limited ad-hoc querying as any change in the application-specific access patterns will impact the design to a large degree; the predefined set of column-families makes it difficult to use wide-column stores for applications with evolving schemas.
Document	Data can be easily interpreted as documents and are constantly evolving.	A rich data model to store data with arbitrary complexity, such as nested structures, arrays, and scalar values; each component of a document can be accessed via secondary indices.	There is no standard API or query languages.
Graph	There is the need to traverse several levels of relationships among intensely related data.	Fast and simple querying of linked datasets, and easy mapping of entity-relationship diagrams	There is no standard API or query languages. Partitioning of large graphs reduces the performance owing to high amount of internode communications.

Figura de Davoudian-Chen-Liu
"A Survey on NoSQL Stores"

Resumen comparativo de los modelos

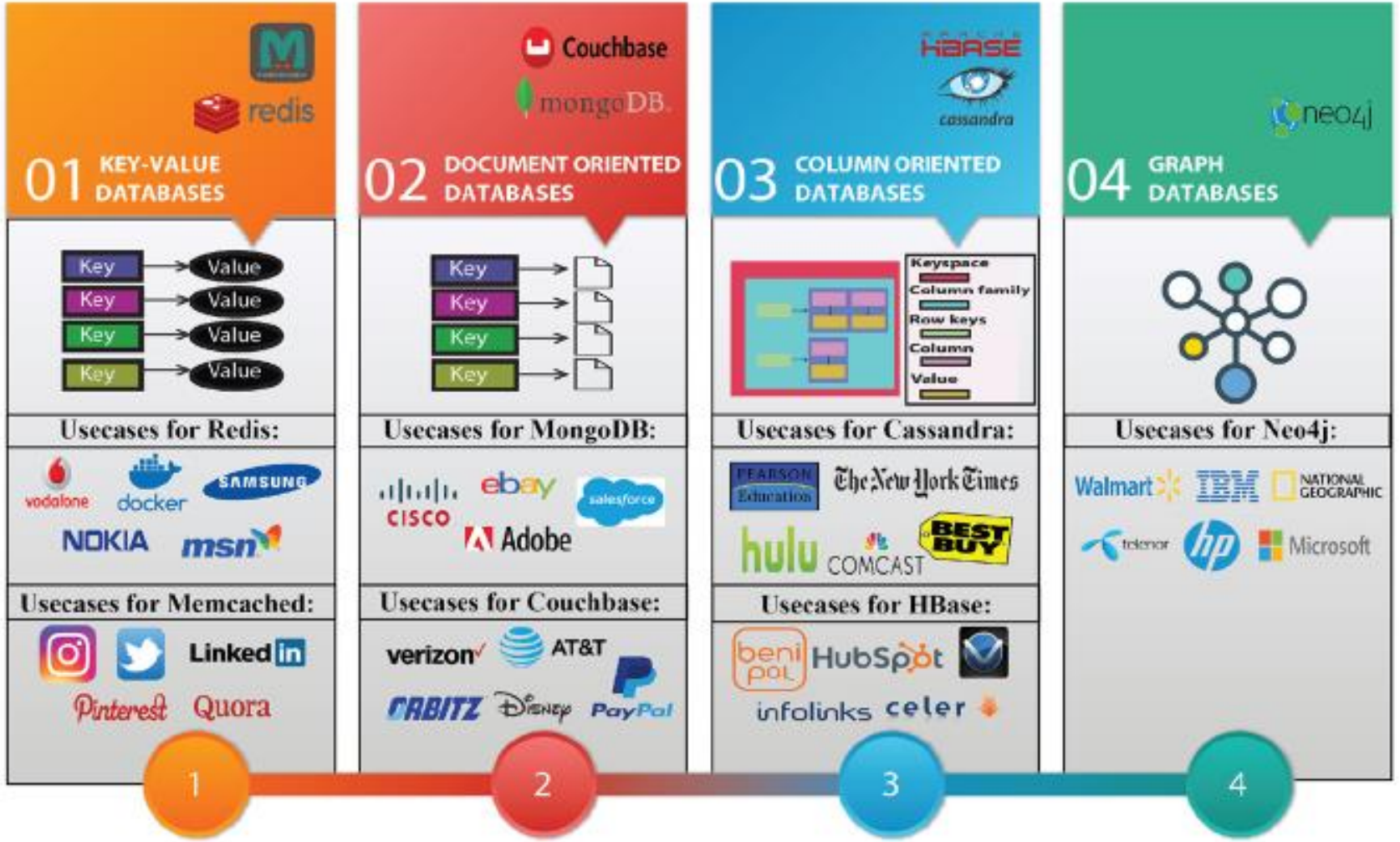


Fig. 1 Categories of NoSQL systems

Figura de Chaudhry-Yousaf, "Architectural assessment of NoSQL and NewSQL systems"

¿Cuando usar Relacional o NoSQL?

BBDD relacional es apropiada:

- Si tenemos datos muy interrelacionados
- Si necesitamos consistencia fuerte e integridad de datos
- Si tenemos datos estructurados con esquemas estables
- Si queremos facilidad para recuperar datos
- Si queremos facilidad para realizar consultas con lenguaje estándar

BBDD NoSQL es apropiada:

- Si tenemos datos poco interrelacionados
- Si priorizamos disponibilidad sobre consistencia fuerte
- Si los datos son no estructurados, semiestructurados, con esquemas flexibles o sin esquemas.
- Si queremos facilidad para escalar horizontalmente a bajo costo
- Si tenemos volúmenes realmente enormes

Persistencia Políglota

- La existencia de distintas tecnologías de almacenamiento, nos presenta la oportunidad de aprovechar las fortalezas de cada una para fines específicos
- **Persistencia políglota**: Uso de múltiples tecnologías de almacenamiento de datos basadas en las características de los datos y la forma que son utilizados por aplicaciones individuales, y aún por una misma aplicación compleja

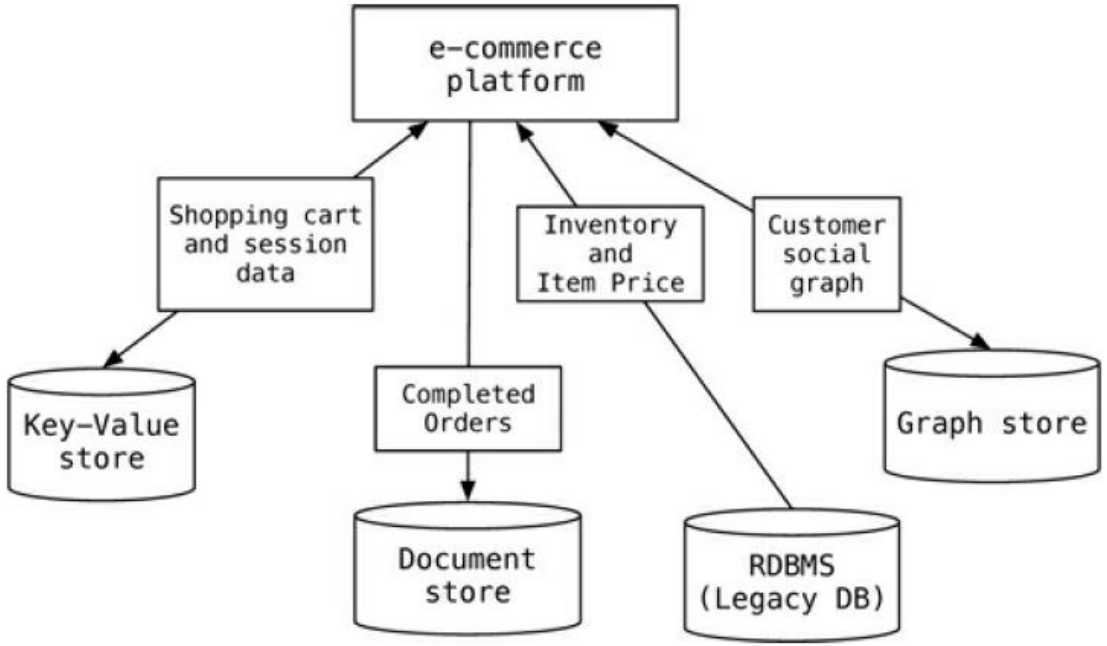


Figure 13.3. Example implementation of polyglot persistence

Figura de Sadalage-Fowler" NoSQL Distilled"

Bibliografía

- *Elmasri R. & Navathe S. (2016) Fundamentals of Database Systems (7ma. Ed.) Pearson, Cap. 24*
- *Silberschatz A., Korth H. & Sudarshan S.(2020) Database System Concepts (7ma. Ed.), Mc.Graw Hill, Cap. 10*
- *Davoudian A., Chen L. & Liu M. (2018) A Survey on NoSQL Stores.* ACM Comput. Surv. 51, 2, Article 40 (April 2018), 43 pages.
- D. Ganesh Chandra, BASE analysis of NoSQL database, Future Generation Computer Systems (2015)
- *Brewer E. (2012) CAP Twelve Years Later: How the "Rules" Have Changed.* Computer (Volume: 45, Issue: 2, February 2012)
- Abadi, D. (2012) *Consistency Tradeoffs in Modern Distributed Database System Design.* Computer (Volume: 45, Issue: 2, February 2012)
- *Harrison, G. (2015) Next Generation Databases (1ra. Ed.) Apress*
- *Sullivan Dan (2015) NoSQL for Mere Mortals (1ra. Ed.) Addison-Wesley*
- *Chaudhry, N., Yousaf, M.M. Architectural assessment of NoSQL and NewSQL systems. Distrib Parallel Databases 38, 881–926 (2020)*
- *Sadalage, P. & Fowler M. (2023) NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence (1ra. Ed.) Addison-Wesley*

Dudas

Muchas gracias!