Resumen Interciclo

Capítulo 1

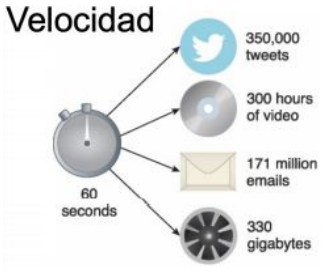
¿Qué es Big Data?

- Big Data se dedica al análisis, procesamiento y almacenamiento de una gran cantidad de datos provenientes de fuentes heterogéneas. Es una actualización de las técnicas tradicionales de análisis, procesamiento y almacenamiento por no ser suficientes.
- Big data consiste en conjuntos de datos que crecen tanto que resulta difícil trabajar con ellos utilizando herramientas de administración de bases de datos disponibles (Wikipedia).
- Big data es cuando el tamaño de los datos en sí se convierte en parte del problema. (Mike Lukides, O'Reilly Radar)
- No son solo sus problemas de "Big Data", se trata de sus GRANDES problemas de "datos" (Alexander Stojanovic, Hadoop Manager on Win Azure)

Campo de aplicación de Big Data

- Optimización de operaciones
- Identificación de nuevos mercados
- Predicción (Clima, desastres, bolsa de valores)
- Detección de fraudes
- Soporte a la toma de decisiones
- Descubrimientos científicos

Las 5 vs de BIG DATA

- **Volumen**: El tamaño de la data
- **Velocidad**: La velocidad con la que la data se genera
  - 

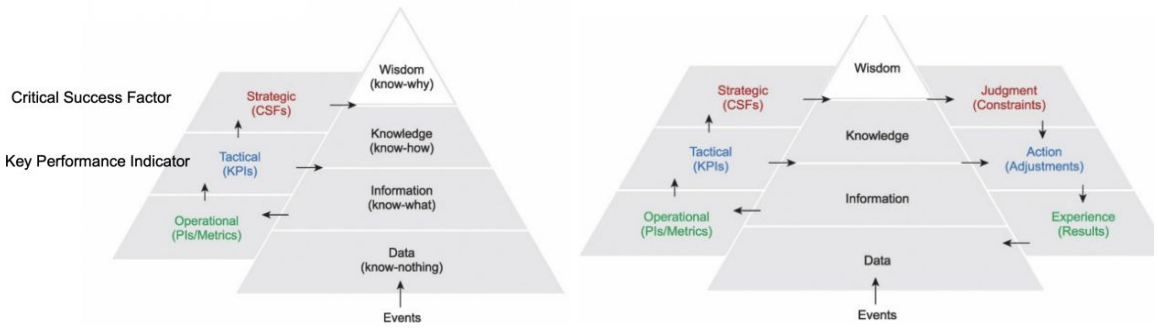
- **Variedad**: Los diferentes tipos de data

- **Veracidad**: La confiabilidad de los datos en términos de precisión

    - Twitter
- **Valor**: Big Data es útil cuando se puede convertir en valor (Lucrar)

Causas para la adopción de Big Data

- Nueva dinámica del mercado
    - Burbuja .com, recesión 2008.
    - Mantener rentabilidad y Reducir costos
    - Man/Tener nuevos - antiguos clientes, mediante nuevo productos/servicios o valor agregado al cliente
- Arquitectura del negocio



- Manejo de procesos de negocios
    - Descripción de cómo se realiza el trabajo.
    - Actividades del negocio y las relaciones con los actores responsables de ejecutarlas.
    - Procesos alineados a los objetivos del negocio
- TICs
    - Análisis de datos y ciencia de datos
    - Digitalización
    - Tecnología asequible y hardware básico
    - Medios de comunicación social
    - Comunidades y dispositivos hiperconectados
        - Relación directa con la Internet de las cosas (IoT)
        - Permitir obtener información de todos los dispositivos posibles.
        - Utilizar como fuente de datos comunidades de información.
    - Computación en la nube
        - Capacidad de utilizar la nube para acceder a los datos
        - Utilizar la capacidad de procesamiento existente en aplicaciones almacenadas en la nube.
- Internet of Everything (IoE)
    - 14 billones => 020: 32 billones

Características deseadas de un sistema de Big Data

- Robustez y tolerancia a fallos
    - *El sistema se comporta correctamente aunque algunos PCs se han caído.*

    - Compleja semántica y consistencia en base de datos distribuidas.
    - Los sistemas deben ser ”human-fault-tolerant”-tolerante a fallo humano
- Latencia baja en lecturas y escrituras
    - *Leer/Escribir mucha información en muy pocos segundos.*
    - Algunas aplicaciones requieren tiempo para propagar las actualizaciones en sus sistemas.
    - Se requiere leer rápidamente información sin comprometer la robustez del sistema
- Escalabilidad
    - *Capacidad de agregar nuevos datos o recursos sin comprometer el desempeño del sistema.*
    - La arquitectura Lambda es horizontalmente escalable a través de cada capa.
    - Se logra al añadir varias computadoras.
- Generalizable
    - *Puede soportar un número grande de aplicaciones.*
    - La arquitectura Lambda esta basada en función de todos los datos.
    - Los datos pueden ser de diferente tipo: financieros, social media, aplicaciones cientıficas.
- Extendible
    - *No reinventar la rueda cada vez que se quiera agregar una característica.*
    - *Agregar una funcionalidad requiere un costo mínimo de esfuerzo.*
    - A veces la inclusión de una nueva funcionalidad requiere de la migración de datos viejos en un nuevo formato.
    - *Capaz de migrar grandes cantidades de datos rápida y fácilmente.*
- Ad hoc queries
    - La posibilidad de *crear consultas específicas* para obtener información interesante.
- Mantenimiento Mínimo
- Depurable

Soluciones de Big Data

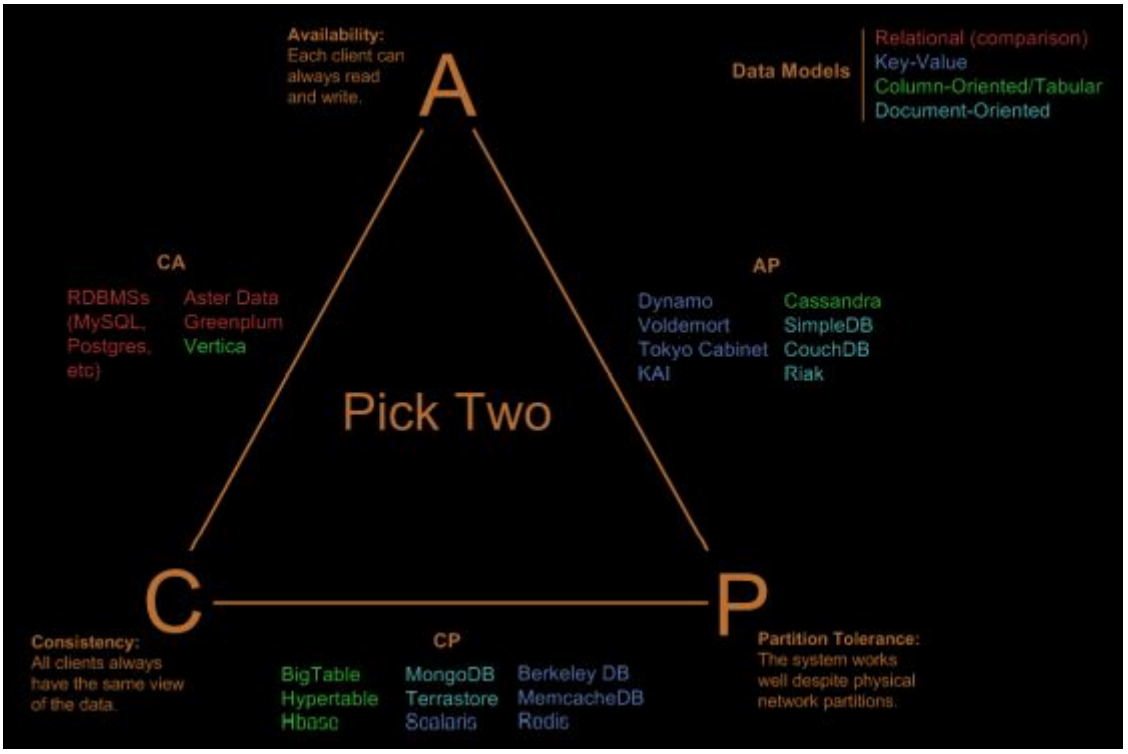
- **MapReduce**: Framework de hardware distribuido (cluster/grids) que divide los problemas en subproblemas (Map) y luego se recopila las mini-respuestas (Reduce) para generar conclusiones. La solución más común es Hadoop. Modelo creado y promovido por Google.
- **NoSQL Database**

  Amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico del sistema de gestión de bases de datos relacionales (RDBMS) en múltiples aspectos:
    - No usan SQL como principal lenguaje de consultas

    - Los datos almacenados *no requieren estructuras fijas como tablas*
    - *No garantizan ACID* (atomicidad, consistencia, aislamiento y durabilidad)
    - *Escalan bien horizontalmente* (ej: MongoDB, Cassandra, BigTable)
- **Algoritmos genéticos**
    - *Un algoritmo es una serie de pasos organizados que describen el proceso que se debe seguir, para dar solución a un problema específico.*
    - Con la inteligencia artificial, surgieron los algoritmos genéticos, inspirados en la evolución biológica
    - Evolucionan sometidos a mutaciones y recombinaciones genéticas.
- **Recaptcha (Google)**
    - reCAPTCHA es una extensión de la prueba CAPTCHA.
    - Reconocer texto presente en imágenes, usado para determinar si el usuario es o no humano.
    - Mejorar la digitalización de textos.
- **Reconocimiento de patrones**
    - *Ciencia que se ocupa de los procesos sobre:*
        - Ingenieria, computacion y matematicas
        - Relacionados con objetos fisicos o abstractos
        - Extracción de información que permita establecer propiedades de entre conjuntos de dichos objetos.
- **NUI (Natural User Interface)**
    - *Interfaz que permite interactuar con un sistema sin utilizar sistemas de mando o dispositivos de entrada de las GUI, usando en su lugar movimientos gestuales*.
    - Ej: Kinect. Reconocimiento de gestos y movimientos.

Teorema CAP

- **Consistency** (Consistencia), **Availability** (Disponibilidad) y **Partition Tolerance** (Tolerancia al Particionamiento)
- Nos dice que en un sistema distribuido de almacenamiento de datos no podemos garantizar consistencia y disponibilidad (para actualizaciones)
- Partición (queda separado en dos o mas islas).
- *Depende de las exigencias del proyecto para saber que atributos de calidad es necesario y elegible.*



El teorema es que solo puedes garantizar dos de estos tres atributos:
CP (Consistencia y Tolerancia al particionamiento):

- No disponibilidad
- No es elegible con clientes que requieren que el sistema esté disponible 100% del tiempo o muy cerca
- Se puede lograr en cierto nivel, pero el **sistema esta enfocado en aplicar los cambios de forma consistente aunque se pierda comunicación con algunos nodos**.

AP (Disponibilidad y Tolerancia al particionamiento):

- No garantiza datos iguales en todos los nodos todo el tiempo
- En este caso el **sistema siempre estará disponible para las peticiones aunque se pierda la comunicación entre los nodos**.

CA (Consistencia y disponibilidad):

- No particionado de los datos, porque se **garantiza que los datos siempre son iguales y el sistema estará disponible respondiendo todas las peticiones**.
- Por ejemplo, los sistemas de bases de datos relacionales (SQL) son CA porque todas las escrituras y lecturas se hacen sobre la misma copia de los datos.

Presentaciones

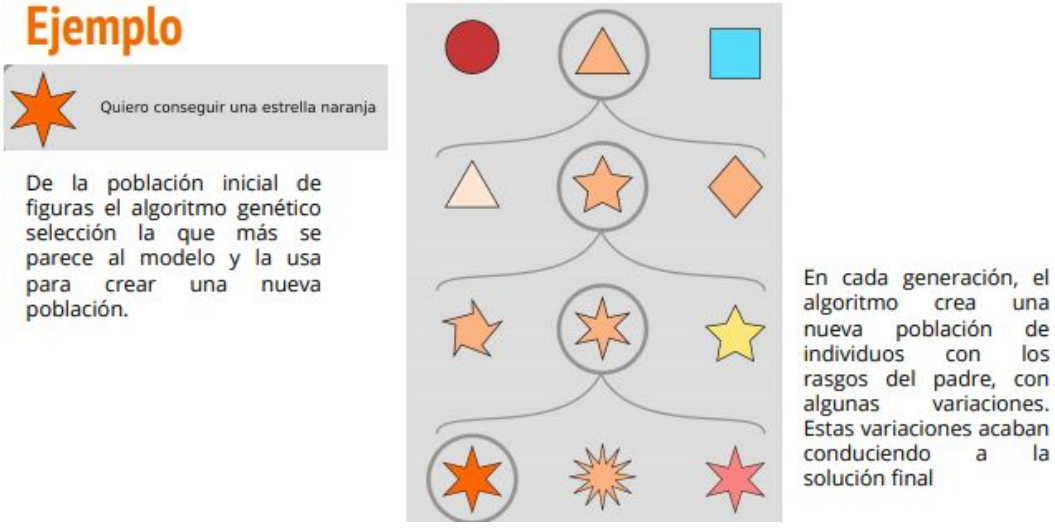
- Cassandra: Es clave valor, CAP es AP
- CouchDB: Es documental, CAP es AP
- Mongo: Es documental, CAP depende contexto, mayoritariamente CP
- Firebase: Es documental organizado en colecciones, CAP es CP
- REDIS: Es clave valor, Redis es CP xq deja de estar disponible en particiones minoritarias

Algoritmo Genético

Inspirado en la evolución natural para solucionar problemas de optimización que de otra forma serían difíciles para un diseñador humano. Se tiene un conjunto de soluciones al problema a resolver. Se llama población al conjunto de soluciones e individuo a cada una de las soluciones. **El algoritmo evalúa cada una de las soluciones y selecciona las que mejor resuelvan el problema**

Algoritmo: Serie de pasos que describen el proceso de búsqueda de una solución a un problema concreto.

Algoritmo genético: Usan mecanismos que simulan los de la evolución de las especies de la biología para formular dichos pasos. Es una **técnica de IA** inspirada en la idea de que el que **sobrevive es el que está mejor adaptado al medio** (teoría evolución de Darwin, combina esa idea de la evolución con la genética). Son **algoritmos de optimización búsqueda y aprendizaje** inspirados en los procesos de Evolución Natural y Genética.

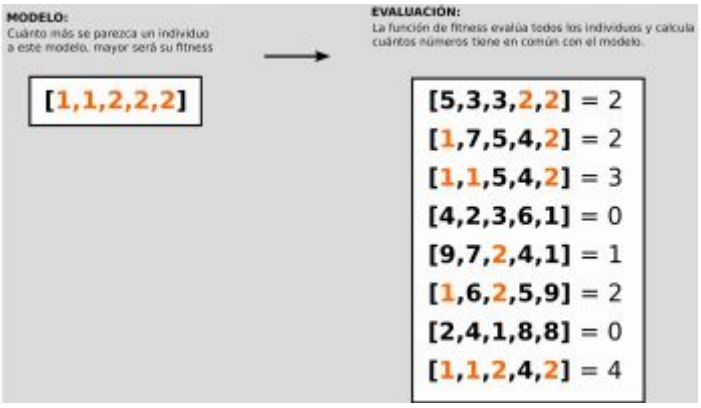


Pasos para construir un Algoritmo genético

- Inicialización

- Evaluación/Función de Fitness
- Selección
- Reproducción
- Crossover
- Mutación

1. Inicialización: La población inicial puede generarse al azar, pero debe ser grande y diversa para que durante la evaluación los individuos muestran un fitness distinto. Si todos los individuos tienen el mismo fitness, el programa no selecciona bien los mejores para su reproducción.
2. Evaluación/Función de Fitness: La Función de Fitness evalúa a cada uno de los individuos de la población y les asigna un valor numérico según su calidad. *Este valor numérico se llama fitness.*



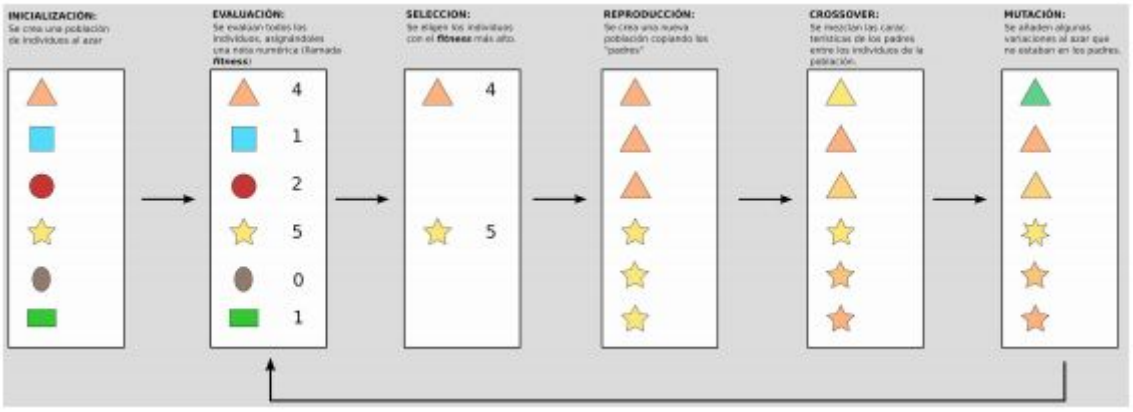
3. Selección/Reproducción: Varias formas de seleccionar los individuos con el fitness más elevado para crear una nueva población. Algunas de las más utilizadas son:

   Selección proporcional: cuanto más alto sea el fitness de un individuo, mayor será la probabilidad de que pase a la siguiente generación.

   Selección por torneo: se eligen varios individuos al azar de la población y se compara su fitness. El individuo con el fitness más alto pasará a la siguiente generación. Hay ocasiones en que es interesante escoger soluciones que no son tan buenas para poder mantener una buena variabilidad genética entre los individuos. Así al mezclar el material genético se exploran soluciones muy distintas.

   Selección por rango: los individuos se ordenan en función de su fitness, y la probabilidad que tienen de reproducirse va según su posición.
4. Crossover/Mutación: Una vez se han seleccionado los mejores individuos, el **Crossover consiste en mezclar el material genético de estos para crear los nuevos individuos.** La forma más sencilla es mediante un One-point Crossover, consiste en elegir un punto al azar de los dos individuos e intercambiar el material genético a partir de esta posición. El crossover no ayuda a encontrar

nuevas soluciones al problema. Es por eso que hay que mutar el material genético de los nuevos individuos, es decir: añadir pequeñas variaciones al azar.



Aplicación

- Diseño automatizado, incluyendo investigación en diseño de materiales y diseño multiobjetivo de componentes automovilísticos: mejor comportamiento ante choques, ahorros de peso, mejora de aerodinámica, etc.
- Diseño automatizado de equipamiento industrial.
- Diseño automatizado de sistemas de comercio en el sector financiero.
- Construcción de árboles filogenéticos.
- Optimización de carga de contenedores.
- Diseño de sistemas de distribución de aguas.
- Diseño de topologías de circuitos impresos.
- Diseño de topologías de redes computacionales.
- En teoría de juegos, resolución de equilibrios.

Relación con Big Data

- El principal problema de Big Data es la pérdida de rendimiento de los algoritmos de aprendizaje automático y minería de datos. Porque se tiene una gran cantidad de características
- Aplicación de AG relacionada con Big Data se da en el procesamiento de grandes cantidades de datos. Combina con técnicas de clustering para realizar el eficiente procesamiento de los datos. O combinación de técnicas de AG con un procesamiento en paralelo
- **Uso de Algoritmos Genéticos** para:
    - Clasificar los datos de manera eficiente.
    - Extraer la información relevante de la gran cantidad de datos como paso previo para luego aplicar el algoritmo de clasificación.

La implementación de un algoritmo genético como paso previo resulta eficiente para resolver el problema de selección de características

Ventajas:

- **Mejor que la IA convencional por ser más robusta**, no se rompen fácilmente incluso si las entradas cambian levemente o en presencia de ruido razonable.

Además, al buscar en un gran espacio de estado multimodal o superficie n-dimensional, un algoritmo genético puede ofrecer beneficios significativos sobre una búsqueda más típica de técnicas de optimización.

Desventajas

- Función evaluación costosa en tiempo y recursos en problemas de alta complejidad.
- Algoritmo podría no converger en una solución óptima o terminar en una convergencia prematura con resultados no satisfactorios
- Mala escalabilidad con la complejidad.
- La "mejor" solución es solo en comparación a otras soluciones. No se tiene un criterio claro de cuándo detenerse porque no se cuenta con una solución específica.
- No es recomendable usarlos para problemas que convergen en soluciones simples como Correcto/Incorrecto ya que el algoritmo difícilmente convergerá y el resultado será tan válido como escogerlo al azar.
- El diseño, la creación de la función de aptitud y la selección de los criterios de mutación entre otros, necesitan de cierta pericia y conocimiento del problema para obtener buenos resultados.

CASSANDRA

- Base de datos NoSQL distribuida que maneja grandes volúmenes de datos.
- El objetivo principal es la **escalabilidad lineal y la disponibilidad**.
- Soporte robusto para múltiples centros de datos, con la replicación asincrónica sin necesidad de un servidor maestro, que permiten operaciones de baja latencia para todos los clientes.

Características

- *Arquitectura Estable:* Diseño masterless, donde todos los nodos son iguales. Ofrece simplicidad operativa y fácil escalabilidad horizontal.
- *Diseño activo de principio a fin:* Escribir y leer en todos los nodos.
- Rendimiento a escala lineal: Puede añadir nodos sin frenar el ritmo, produce aumentos en el rendimiento.
- *Disponibilidad continua:* elimina los puntos únicos de fallo y proporciona un tiempo de actividad constante.
- *Detección de fallos y recuperación transparente:* fácil restauración en eliminación de nodos.
- *Modelo de datos flexible y dinámico:* soporta tipos de datos modernos para lectura y escritura rápida.
- *Protección de datos sólida:* diseño de registro de confirmación evita la pérdida de datos y construye copias de seguridad.
- *Consistencia de los datos sintonizables:* consistencia de los datos en un clúster ampliamente distribuido.

- *Replicación de datos multi-centro:* centro de datos transversal (diferentes zonas geográficas) que recibe el el apoyo de múltiples zonas de disponibilidad en la nube tanto para escritura como para lectura.
- *Compresión de datos:* garantiza la comprensión de datos hasta un 80% sin que ello suponga un gasto de recursos.
- *CQL (Lenguaje de Consulta Cassandra):* similar a SQL que consigue que la transición desde una base de datos relacional sea muy sencilla.
- En Cassandra los datos están desnormalizados de manera que el **concepto de joins o subqueries no existe.** Interacción mediante shell con cqlshell, o GUI como DevCenter o por drivers soportados para múltiples lenguajes de programación.

Cuando elegir Cassandra.

- Cuando se quiere obtener un escalamiento lineal, un nivel alto de disponibilidad y cortos tiempos de respuesta.
- Debido a las limitaciones que tiene en filtros y ordenaciones, el conocimiento de las consultas a realizar sobre la base de datos tiene un gran impacto en el modelo a definir.

Modelo de Datos.

- Combina propiedades de una base de datos clave-valor y una orientada a columnas.
- Importante definir clave de partición de data, ya que Cassandra usa esta para distribuir los datos a lo largo del cluster.
- Guiarse en diseño por el patrón de acceso a los datos, hay que hacer un análisis de las queries a ejecutar para diseñar un modelo eficiente.

Cassandra da prioridad a la escalabilidad, velocidad y alta disponibilidad por encima de la consistencia.

COUCHDB

Gestor de base de datos NoSQL de código abierto. Desarrollado por Apache Software Foundation en 2005.

AP: DISPONIBLE Y PARTICIONABLE

Compatible con los sistemas operativos Linux, Unix, MacOS y Windows. Escrito en el lenguaje de programación Erlang.

JSON – almacenar datos. API JSON. JavaScript – lenguaje de consulta mediante MapReduce.

HTTP – solicitudes que se emiten desde el navegador.

CouchDB guarda los datos en forma de documentos (almacenados en JSON).

- **_id:** Usado para que CouchDB lo distinga de otros documentos y pueda recuperarlos.
- **_rev:** Controlador de versiones.

- **Otros campos**: Usan expresiones JSON válidas como arrays de strings.

MONGO

Tipos de NOSQL: Bases de datos Documentales, Orientadas a grafos, Bases de datos Clave/Valor, Bases de datos Multivalor, Bases de datos Tabulares, Bases de datos de Arrays.

Sistema de base de datos NoSQL **orientado a documentos (**no guardar datos en registros sino en documentos**),** almacenados en BSON, de código abierto y escrito en C++, multiplataforma.

Si se necesita realizar consultas de agregación MongoDB tiene un Framework para realizar consultas llamado Aggregation Framework.

Algunas aplicaciones que usan MongoDB: WindyGrid, Bosch, AIR FRANCE.

- garantiza operaciones atómicas a nivel de documento, no usan joins
- Balance entre rendimiento y funcionalidad
- Alta velocidad
- *Consultas ad hoc*: se puede realizar todo tipo de consultas, p.e. hacer búsqueda por campos, consultas de rangos y expresiones regulares.
- *Indexación*: similar a bd relacionales, con la diferencia de que cualquier campo documentado puede ser indexado y añadir múltiples índices secundarios.
- *Replicación*: soporta replicación primario-secundario
- *Balanceo de carga*: ejecutarse de manera simultánea en múltiples servidores
- *Ejecución de JavaScript del lado del servidor*: consultas utilizando JavaScript, son enviadas directamente a la base de datos para ser ejecutadas.

CAP:

C: Muy consistente cuando usa una sola conexión o el nivel de preocupación de escritura / lectura correcto (cuesta velocidad de ejecución).

A: Alta disponibilidad a través de Replica-Sets. Tan pronto como la primaria deje de funcionar o deje de estar disponible, las secundarias determinarán una nueva primaria para volver a estar disponible.

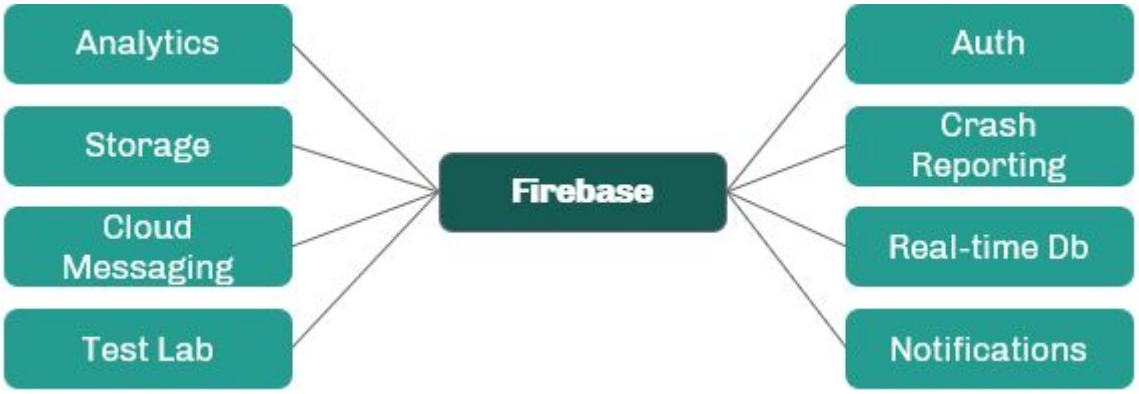
P: tolerancia de partición: siempre que más de la mitad de los servidores de un conjunto de réplicas están conectados entre sí, se puede elegir un nuevo primario.

No se puede simplemente decir que MongoDB es CP / AP / CA, porque en realidad es una compensación entre C, A y P, dependiendo de la configuración de la base de datos / controlador y el tipo de criterio.

Para un sistema distribuido como Mongodb siempre se construye con tolerancia a particiones, por lo tanto se puede decir que si hay una partición de red y si se desea que su sistema siga funcionando, puede proporcionar Disponibilidad o Consistencia y no ambas.

Firebase

- Es todo un backend, actualmente tiene 8 tecnologías que mejora la experiencia de desarrollo de aplicaciones.



- FireStore es una base de datos NoSql de tipo documentos de Firebase.
- Online y Offline - Proporciona sincronización entre dispositivos conectados y está disponible cuando no hay conectividad de red.
- **En CAP es CP**
- Cloud Firestore es una base de datos NoSQL alojada en la nube, flexible y escalable para la programación en servidores, dispositivos móviles y la web desde Firebase y Google Cloud Platform. Almacenar los datos en documentos que contiene campos que se asignan a valores. Los documentos se organizan en colecciones, que son contenedores para los documentos y se pueden usar para organizar los datos y compilar consultas.
- Las consultas son expresivas, eficientes y flexibles.
- Los documentos admiten diferentes tipos de datos, desde strings hasta objetos anidados complejos o de subcolecciones dentro de documentos.
- Firebase Realtime Database, mantiene la sincronización de los datos entre apps cliente a través de agentes de escucha en tiempo real y ofrece asistencia sin conexión.
- Recomendable en proyectos con tiempos de desarrollo limitados.
- Recomendable cuando se requiera construir aplicaciones online por sobre los offline.
- Recomendable en proyectos con manejo de dispositivos inteligentes con mínimos requerimientos de uso.
- Recomendable en proyectos con correcta planificación de costes..

REDIS

Motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes (clave/valor) pero que opcionalmente puede ser usada como una base de datos durable o persistente.

Redis es CP porque deja de estar disponible en particiones minoritarias. Tenga en cuenta que seguirá estando disponible en la partición mayoritaria.

Ventajas:

Una velocidad muy alta, gracias a su almacenamiento en memoria

- Posibilidad de persistir datos en disco para recuperación ante fallas
- Fácil configuración
- Alta disponibilidad
- Curva de aprendizaje baja
- Una variedad de tipos de datos

Desventajas:

- El método de persistencia RDB consume mucho I/O (escritura en disco)
- Todos los datos trabajados deben encajar en la memoria (en caso de no usar persistencia física)

Resumen Cap 2

Data Model

- El modelo con el cual la base de datos organiza su información Modelo de Datos Relacional
    - Conjunto de tablas
        - Filas que representan alguna entidad
        - Columnas: describen a la entidad (cada columna un solo valor).
        - Cada columna puede apuntar a otra fila en la misma o en otra tabla, lo que representa una relación entre esas dos entidades

Modelos de Datos NoSQL

Alejado del Modelo Relacional, cada solución NoSQL tiene su propio modelo

Existen 4 categorías

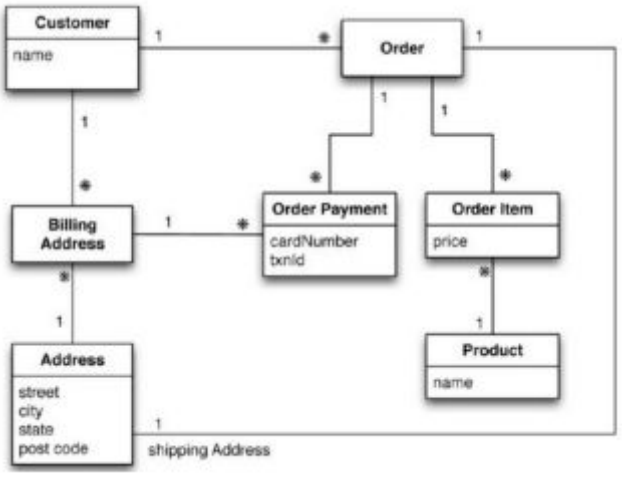
1. **Clave - Valor** = Orientadas a la Agregación
2. **Documentales** = Orientadas a la Agregación
3. **Orientadas a columnas** = Orientadas a la Agregación
4. **Grafos**

Agregados

- Modelo Relacional
    - Tuplas (Estructura de datos limitada)
        - No hay como anidar una tupla dentro de otra
        - Peor aún poner una lista de valores o tuplas dentro de otra
        - Esta simplicidad permite realizar todos los tipos de operaciones conocidas sobre las tuplas.
- Normalmente se necesita trabajar con datos en unidades que tienen una estructura más compleja que un conjunto de tuplas.
- Se puede pensar en un registro complejo que permita anidar listas y otras estructuras dentro de él.

- DBs Clave-Valor, Documentales y Orientadas a Columnas usan esta clase de “registros complejos”
- Un agregado es una colección de datos relacionados que nosotros queremos tratar como una unidad.
- **Unidad** para la manipulación de datos y para el manejo de la consistencia.
- Actualizar agregados con operaciones atómicas y comunicarse con la base de datos en términos de agregados.
- Facilitan la ejecución en un cluster de computadores, ya que un agregado sería la unidad para replicación y “sharding” o fragmentación
- Facilitan el trabajo a los desarrolladores

Ejemplo: Relaciones y Agregados



Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

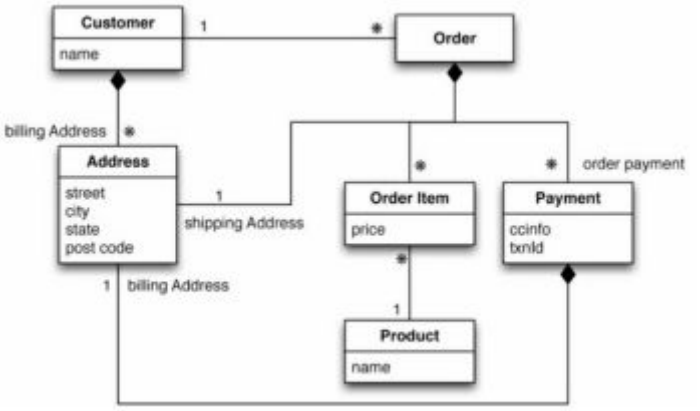
Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abeli879rft




```
// in customers
{
    "id":1,
    "name":"Martin",
    "billingAddress":[{"city":"Chicago"}]
}
```

```
// in orders
{
    "id":99,
    "customerId":1,
    "orderItems":[
        {
            "productId":27,
            "price": 32.45,
            "productName": "NoSQL    Distilled"
        }
    ],
    "shippingAddress":[{"city":"Chicago"}]
    "orderPayment":[
        {
            "ccinfo":"1000-1000-1000-1000",
            "txnId":"abelif879rft",
            "billingAddress": {"city": "Chicago"}
        }
    ],
}
```

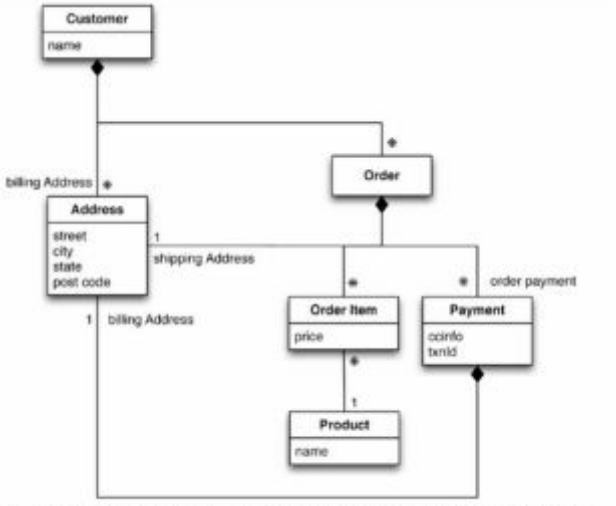


Figure 2.4. Embed all the objects for customer and the customer's orders

```
// in customers
{
   "customer": {
      "id": 1,
      "name": "Martin",
      "billingAddress": [{"city": "Chicago"}],
      "orders": [
      {
         "id":99,
         "orderItems":[
         {
            "productId":27,
            "price": 32.45,
            "productName": "NoSQL Distilled"
         }],
         "shippingAddress":[{"city":"Chicago"}]
         "orderPayment":[
         {
            "ccinfo":"1000-1000-1000-1000",
            "txnId":"abelif879rft",
            "billingAddress": {"city": "Chicago"}
         }],
      }]
   }
}
```

Consecuencias de la Agregación

- El modelo relacional captura los datos y sus relaciones muy bien, pero sin ninguna noción de una unidad de agregación.
- Las técnicas propuestas por las tecnologías NoSQL nos permiten crear agregaciones o estructuras compuestas. Pero....
    - Los que modelan no proporcionan ninguna información para distinguir una agregación de otra
    - Si la hay, esta varıa dependiendo de cada situación
- *Cuando se trabaja con BDs orientadas a agregación, lo único claro es que la agregación es la unidad de interacción con la base de datos.*
- Pero todo depende de cómo los datos son usados dentro de la aplicación. (Esto a veces se sale de las manos de la modelación)
- RDBMS y de Grafos (Aggregate-ignorant) → (No conocen la agregación)
    - NO es el fin del mundo
- Es difícil definir los lımites de las agregaciones correctamente, especialmente si la misma data es usada en diferentes contextos
    - Por ejemplo: Agregación Orden o Producto?
- La agregación facilita la ejecución en clusters.
- Al definir agregaciones le decimos a la base de datos que bits van a
- ser manipulados juntos
- Y por lo tanto deberían residir en el mismo nodo.
- RDBMS → ACID (Tablas, Filas)
- NoSQL → BASE (Una agregación a la vez)
    - Varias Agregaciones de manera atómica

DBs Clave-Valor y Documentales

- Ambas son fuertemente orientadas a agregación
- Ambas bases consisten de varias agregaciones donde cada agregación tiene una clave o ID que es usada para recuperar los datos
- Diferencia Clave-Valor:

    - La agregación es desconocida para la base de datos
    - Blob (Binary Large Objects) de bits sin ningún significado
    - Se puede almacenar lo que sea. (Lımite en espacio)
- Documentales:
    - Es capaz de distinguir la estructura dentro de las agregaciones.
    - Ciertos lımites en lo que se puede almacenar
        - Estructuras y tipos predefinidos
        - Flexibilidad en el acceso
- Diferencia
    - **Clave-Valor**: Permite buscar agregaciones por Clave o ID
    - **Documentales**: Queries basados en la estructura interna de la agregación. Puede ser la clave, pero por lo general es otro valor.

BDs Orientadas a Columnas

- RDBMS
    - Filas como unidad de almacenamiento
        - Ayuda a mejorar el rendimiento en las escrituras
    - Hay muchos escenarios donde las escrituras son esporádicas:
        - Se necesita leer algunas columnas de varias filas a la vez.
        - Grupos de columnas para todas las filas como unidad de almacenamiento.

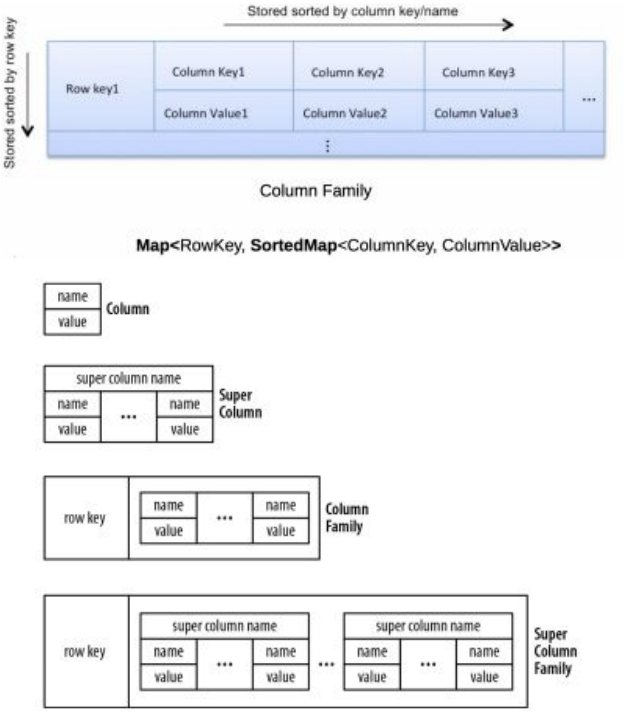
De-Moneda	A-Moneda	Precio venta	Precio Compra	Banco	Fecha
USD	EURO	1.2300	1.2850	Austro	18/09/2014
USD	PLN	1.3400	1.3050	Austro	18/09/2014
USD	AZN	1.2700	1.5150	Austro	18/09/2014
USD	EGP	1.7600	1.5800	Austro	18/09/2014
USD	EEK	1.4000	1.4213	Austro	18/09/2014
USD	FJD	1.4425	1.4553	Machala	18/09/2014
USD	GIP	1.4681	1.4929	Machala	19/09/2014
USD	DKK	1.5177	1.4874	Machala	19/09/2014
EURO	USD	1.4571	1.4642	Machala	19/09/2014
EURO	FJD	1.4713	1.4749	Pichincha	19/09/2014
EURO	GIP	1.4785	1.4799	Pichincha	19/09/2014
EURO	EEK	1.4812	1.4766	Pichincha	19/09/2014

1B, 100Bytes = 100GB x (3/6); 100MB/sec = 500sec

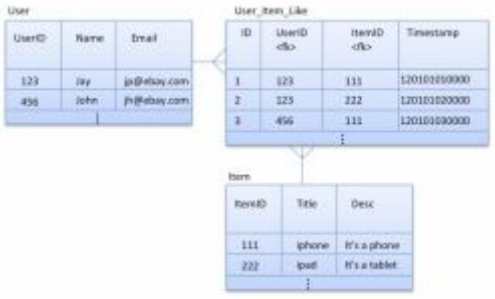
    - Almacenamiento más eficiente debido a la compresión de datos USDx8, EUROx4 Funciones de Agregación Max, Min, Sum, Avg

De-Moneda
USD
USD
USD
USD
USD
USD
USD
USD
EURO
EURO
EURO
EURO

- Estructura de agregación a dos niveles



- Modelar de acuerdo a los patrones de búsqueda
    - Pensar en patrones de búsqueda por adelantado
        - Identificar los queries mas frecuentes
        - Detectar cuales pueden ser lentos
        - Asegurarse que el modelo satisface los queries más frecuentes y críticos
    - Recordar que una “Familia de Columnas” es un SortedMap
        - Búsqueda, Ordenamiento, Agrupamiento, Filtrado, Agregaciones, etc.
- De-Normalizar y Duplicar por lecturas eficientes
    - No de normalizar si no se necesita, encontrar un balance
- Normalización
    - Pros: No hay duplicación de información, menos errores por modificación de datos, conceptualmente mas claro, etc...
    - Cons: Queries demasiado lentos si hay varios joins y demasiados datos (Big Data)
- Likes: Relación entre usuarios e ítems

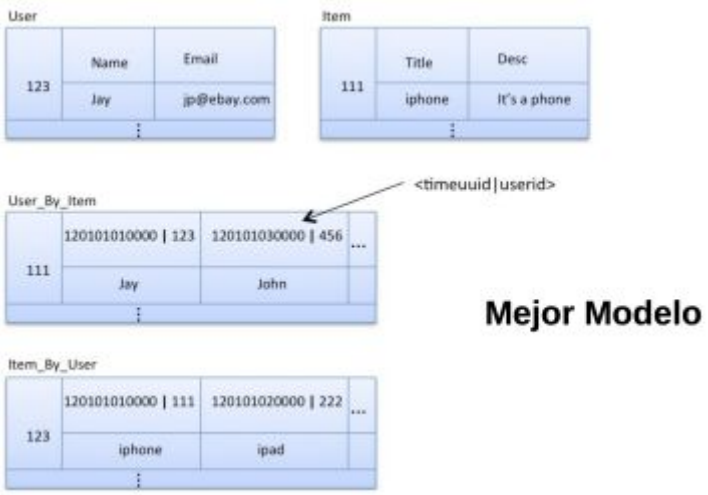


- Usuarios x UserID
- Items x ID
- Todos los items que le gustan a un usuario en particular
- Todos los usuarios a los que les gusta un ítem en particular
- Réplica del modelo relacional

  - Usuarios x UserID
  - Items x ID
- Entidades Normalizadas con ındices personalizados

- Agregar Título de Item
- Agregar Nombre de Usuario

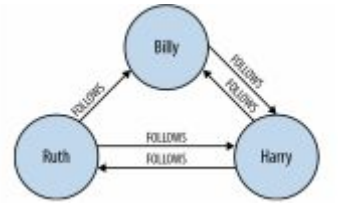
  - Usuarios x UserID
  - Items x ID
  - Todos los items que le gustan a un usuario en particular
  - Todos los usuarios a los que les gusta un ítem en particular
- Entidades Normalizadas con de-normalización en ındices personalizados



- **Resumen**
  - Un agregado es una colección de datos relacionados que queremos tratar como unidad de almacenamiento
  - BDs Clave-Valor, Documentales, Orientadas a Columnas son orientadas a la agregación
  - Agregados permiten la ejecución en clusters de servidores
  - Las BDs orientadas a agregación funcionan muy bien cuando las interacciones con los datos son hechas con la misma agregación.
  - Aggregate-Ignorant BDs son mejores cuando las interacciones con la base usan datos que están organizados de múltiples formas.

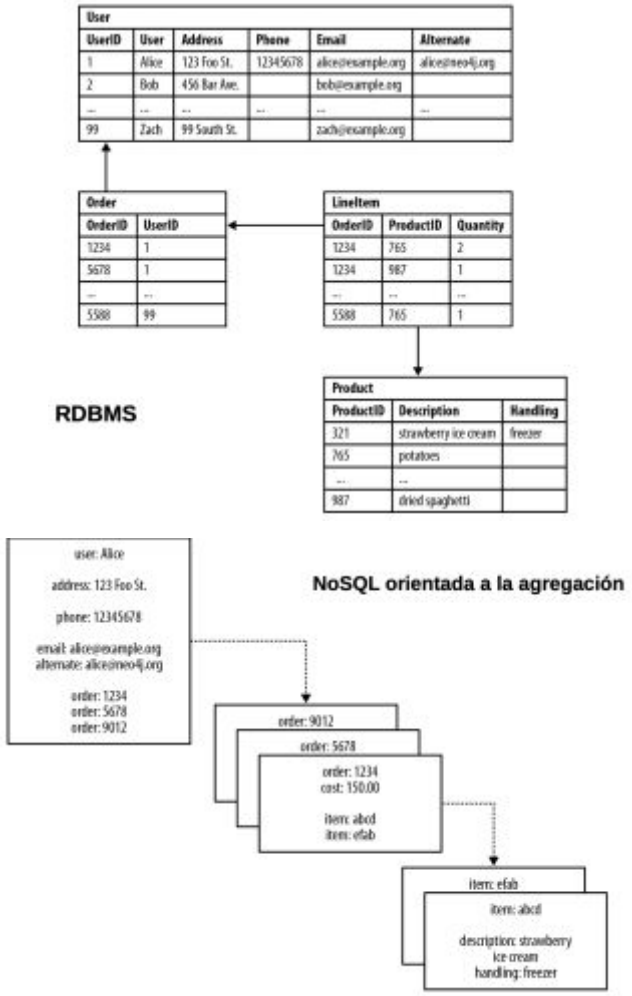
Bases de datos de Grafos

- Cómo manejar pequẽnos registros que tienen conexiones complejas entre śı?
- Modelo de datos de Grafos
  - Nodos y Relaciones Nodos tienen propiedades
  - Las relaciones son nombradas y directas y siempre tienen principio y fin
  - Las relaciones también pueden tener propiedades

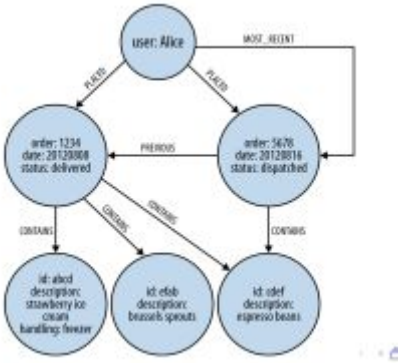


- Base de datos cuya estructura de almacenamiento sigue el modelo de datos de grafos
- Sirven para capturar datos que consisten de relaciones bastante complejas:
  - Redes sociales
  - Preferencias de productos
  - Control de Tráfico
  - Detección de fraude
- Ejemplos:

    - Deme los restaurantes que han sido recomendados por mis amigos
    - Cual es el camino más corto para ir de Cuenca a Machala?
    - Cuales son los productos que a John y Ana les gustan.
- Por que son útiles
    - RDBMS y NoSQL orientadas a agregación no permite capturar relaciones complejas en sus modelos de datos.



    - Recomendaciones
        - Quien compra helado de fresa también compra Cafe
    - Enriquecer nuestros datos
        - Unirlo a grafos de otros dominios (Geo y Social)
    - Todos los sabores de helado que le gusta a la gente que vive cerca de Juan y a las que les gusta el espresso pero no les gusta las coles de bruselas



Bases de datos sin esquema

- RDBMS
    - Debe definir un esquema de antemano
    - Una estructura definida que nos diga que tablas y columnas existen y que tipo de datos va a tener cada columna.
- **NoSQL (el almacenamiento es más casual)**
    - Clave-Valor: permite almacenar cualquier tipo de datos
    - Documentales: No hay restricciones en la estructura de los documentos
    - Columnas: permiten almacenar cualquier tipo de datos en cualquier columna que se desee.
    - Grafos: permite agregar libremente nodos y aristas y propiedades a los nodos y las aristas.
- *Con un esquema se tiene que definir por adelantado lo que se necesita* almacenar. **Sin un esquema que nos ate, se puede almacenar fácilmente lo que sea que se necesite**. Esto permite **cambiar fácilmente** nuestra base de datos a medida que se conoce más el proyecto. Se puede fácilmente agregar nuevas cosas a medida que se las va descubriendo. *Si no se necesita algo, simplemente se deja de almacenarlo.*
- Así como permitir cambios, una **BD sin esquema permite manejar fácilmente datos no-uniformes**.
    - Datos donde cada registro tiene diferentes campos.
- Un esquema pone a todas las filas en una “camisa de fuerza”.
- Que pasa si hay diferentes tipos de datos en cada fila?
    - Se termina con columnas en valor null
    - O columnas sin sentido (Columna 4)
- NoSQL evita esto permitiendo que cada registro tenga lo que necesita. Ni más ni menos. Las BDs sin esquema pueden evitarnos muchos problemas que existen con las bases que tienen un esquema fijo.
- Sin embargo,estas traen algunos problemas consigo.
    - Los programas necesitan saber que la dirección de facturación es llamada direccionDeFacturacion y no direccionParaFacturacion
    - Y que la cantidad es un entero 5 y no cinco

- El hecho es que cuanto escribimos un programa que accede a ciertos datos, este siempre depende implícitamente de un esquema.
- Un programa asume
    - Que ciertos nombres de campos se encuentran presentes y que estos hacen a referencia a datos con cierto significado
    - Algo acerca del tipo de datos almacenado en ese campo.
- Los programas no son humanos
    - No deducir que Nro = Numero, a menos que se los programe para hacer eso.
- **Así tengamos una BD sin esquema, siempre está presente el esquema implícito.**

  Esquema Implícito

  Es un conjunto de suposiciones acerca de la estructura de los datos en el código que manipula esos datos, presenta algunos problemas:
    - Para entender que datos se están manejando hay que escarbar dentro del código.
    - Si el código es bien estructurado se puede definir fácilmente el esquema, pero nadie nos garantiza eso
    - La BD no puede usar el esquema para recuperar o almacenar los datos más eficientemente.
    - La BD no puede validar los datos para evitar inconsistencias.
- Las BDs sin esquema mueven el esquema al código de la aplicación
    - Que pasa si diferentes aplicaciones hechas por diferentes personas acceden a la misma base de datos?
- Para reducir los problemas
    - Encapsular la interacción con la BD dentro de una sola aplicación que se integra con otras usando servicios web.
    - Delimitar diferentes partes de un agregado para las diferentes aplicaciones
        - Diferentes secciones de un documento
        - Diferentes “Familia de columnas”
- Usar una base de datos sin esquema no es la panacea
- No uniformidad en los datos es una buena razón para usar una BD sin esquema

Vistas materializadas

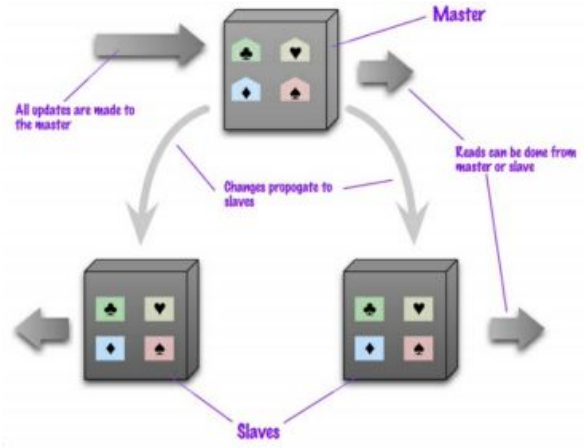
- Ventajas de los modelos orientados a agregación
    - Si se quiere acceder a órdenes, es conveniente tener todos los datos de una orden en un agregado que será almacenado y leído como una unidad
- ¿Hay desventajas?
    - Que pasa si queremos saber cuánto se ha vendido de un producto en las últimas semanas?

    - La agregación juega en contra. Hay que obligatoriamente leer cada orden para responder a esa pregunta
- RDBMS tienen la ventaja que permiten acceder a los datos de diferente manera, además proporcionan un mecanismo que permite observar los datos de manera distinta a la que están almacenados: Vistas
- **Una vista es como una tabla pero es calculada a partir de las tablas base**
- Cuando se accede a la vista la BD calcula los datos a mostrarse en la vista
- Hay algunas vistas que son costosas de calcular
  Solucion: **Vistas Materializadas (VM)**
    - Son vistas precalculadas y son almacenadas en caché en disco.
    - Son efectivas para datos que son de lectura-intensa pero que pueden estar algo desactualizados.
- **NoSQL no tiene vistas materializadas** pero pueden tener **queries precalculados** en una caché en disco.
- Las VM son un aspecto central en la BDs orientas a la agregación, tal vez mas que las RDBMS
    - Las aplicaciones tienen que ejecutar queries que no se adaptan al modelo de agregación
    - Es usual crear vistas materializadas usando **Map-Reduce**
- Dos estrategias para crear vistas materializadas
    - Actualizar la VM al mismo tiempo que se actualiza los datos base.
    - Ejecutar procesos en batch que actualizan la VM en intervalos regulares
- Puede calcular externamente, leyendo los datos calculando la vista y grabando de nuevo en la base de datos
- Es importante conocer el modelo de negocios para saber cuan desactualizada puede estar una VM

Modelos de Distribución

- NoSQL DBs se ejecutan en un cluster de servidores.
    - + volumen de datos
    - + complejidad en escalar verticalmente
    - NoSQL permite escalar horizontalmente.
    - La agregación es la unidad de distribución
- Dependiendo del modelo de distribución:
    - Manejar volúmenes de datos mas grandes
    - Procesar mayor numero de reads o writes
    - Mayor disponibilidad en caso de retardos en la red o particionamiento de red.
- Estos beneficios traen un costo (complejidad) Se debe usar un cluster cuando los beneficios son evidentes.
- Replicación

    - Hace copias exactas de los datos en diferentes servidores
- Sharding
    - Diferente data en diferentes nodos
- Se puede usar cada una por separado o los dos a la vez
- **Modelos de Distribución**
  Único Servidor , Maestro-Esclavo (Replicación), Sharding, Peer-to-Peer (Replicación)
- **Unico Servidor**
    - Sin distribución de datos
    - Ejecutar la BD en una sola máquina que maneja todos las lecturas y escrituras
    - BDs NoSQL fueron diseñadas con la idea de ejecutarse en clusters también se lo puede hacer en un único servidor (BDs Grafos)
    - Si la interacción con los datos es en su mayoría mediante el uso de agregaciones:
        - BDs Único Servidor Documentales o Clave-Valor
    - No hay que dejarse sorprender por la palabra Big Data.
    - Si se puede manejar los datos sin distribución, optar siempre por la opción mas simple:
        - Único Servidor.
- **Maestro-Esclavo (Replicación)**



    - Mas útil para datasets de lectura intensa y pocas escrituras
    - **Garantiza las lecturas (read resilience)**
        - Si el master falla, los esclavos pueden seguir manejando peticiones de lectura (si la mayoría de accesos son de lectura)
    - **Si falla el master no hay escrituras**
        - Hasta que el master se recupere se designa un nuevo máster
    - Facilidad de reemplazar al master por un esclavo (incluso sin necesidad de escalar)
    - Es como un único-servidor con un respaldo en caliente (hot backup).

        - Único servidor con mayor tolerancia a fallos
        - Para garantizar lecturas, se debe asegurar que los paths de lecturas y escrituras sean diferentes
    - El master puede ser seleccionado manual o automáticamente (Leader Election).
    - Replicación tiene ventajas pero tiene su inevitable lado negativo: inconsistencia
        - Diferentes clientes leyendo diferentes esclavos pueden ver datos diferentes
        - Algunos clientes no ven los ultimos cambios del sistema
        - Si falla el master, todos los datos no sincronizados se pierden
- **Sharding**

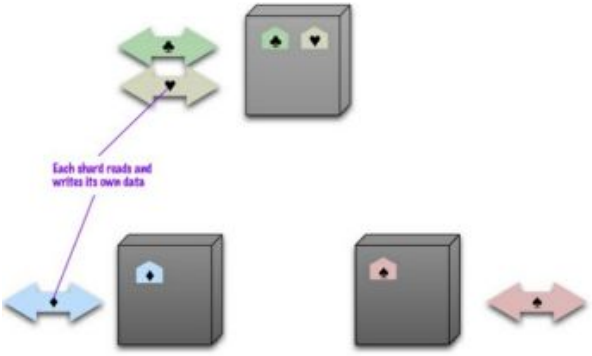


Figure 4.1. Sharding puts different data on separate nodes, each of which does its own reads and writes.

    - Normalmente una BD esta ocupada porque diferentes usuarios están accediendo a diferentes partes de los datos
    - **Se puede soportar escalamiento horizontal poniendo diferentes partes de la data en diferente servidores** (Sharding)
    - En el caso ideal cada usuario se comunica con un solo servidor, obteniendo respuestas rápidas de ese servidor. (10 servidores, 10% c/u)
    - Para lograr algo cercano al caso ideal
        - Los datos que son accedidos simultáneamente sean almacenados juntos en el mismo servidor
            - Agregaciones: unidad de distribución
    - Factores para mejorar el rendimiento
        - Si el acceso esta basado en ubicación física, se puede poner los datos lo mas cerca posible al lugar desde donde son accedidos.
        - Tratar de balancear la carga entre los servidores
        - Poner diferentes agregados juntos si se sabe que se van a leer en secuencia
    - Sharding puede ser **manual** (lógica de la aplicación) o **automática**.
    - Permite escalar horizontalmente lectura y escrituras

        - Sharding no ayuda a mejorar la capacidad de recuperación ante fallos
        - Ante fallos solo los usuarios que acceden al shard sufrirán las consecuencias
    - Sharding es mas fácil de lograr con las agregaciones, pero no se debe tomar a la ligera
    - Algunas bases son diseñadas para usar sharding, en este caso usar sharding desde el principio
    - Otras bases no fueron diseñadas para eso, pero permiten pasar de único-servidor a sharding
    - *Saber identificar cuando se requiere usar sharding y usarlo mucho antes de que realmente se necesita*

- **Peer-to-Peer (Replicacion)**

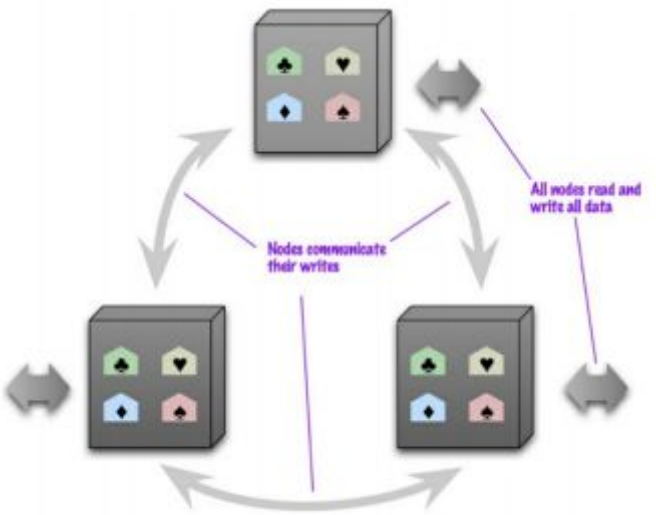


Figure 4.3. Peer-to-peer replication has all nodes applying reads and writes to all the data.

- **Combinar Sharding y Replicación**

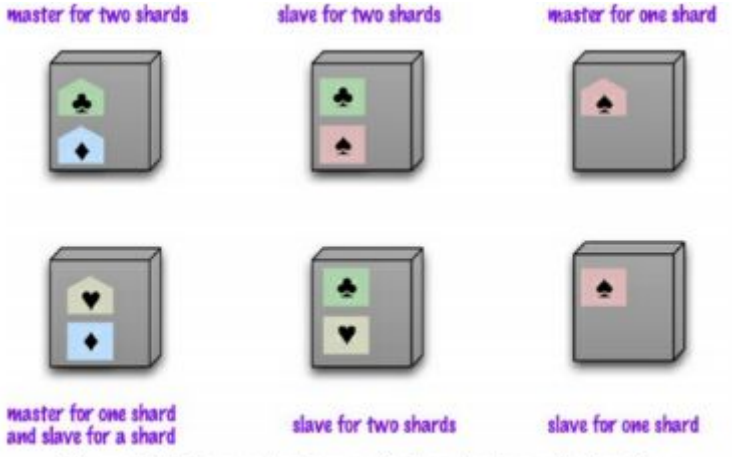


Figure 4.4. Using master-slave replication together with sharding

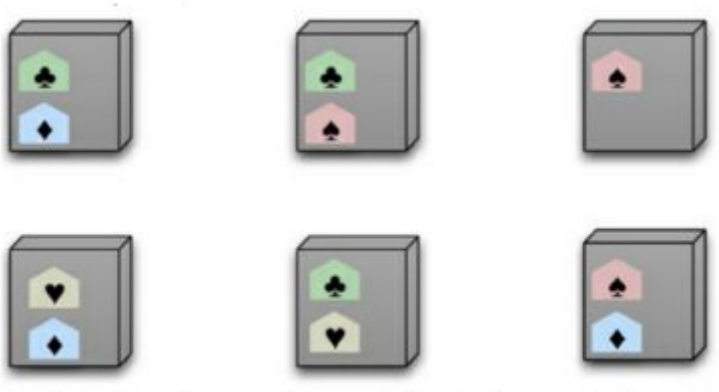
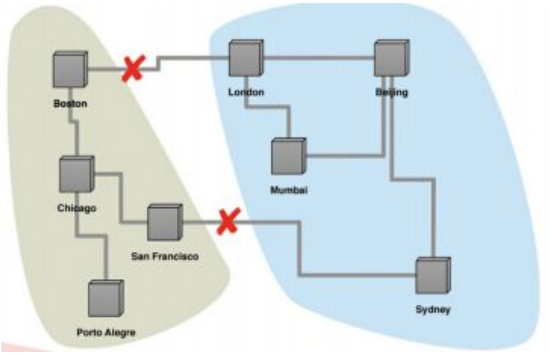


Figure 4.5. Using peer-to-peer replication together with sharding

- **Peer to peer**
    - Todas la réplicas tienen el mismo peso, todas aceptan escrituras y la pérdida de una de ellas no afecta la lectura de los datos
    - Complicación: **Consistencia**
        - Dos clientes tratando de escribir el mismo registro al mismo tiempo (conflicto de escritura)
        - Solución: Coordinar las escrituras.
            - No se necesita que todas se pongan de acuerdo pero si la mayoría

Teorema CAP

- Dr. Eric Brewer, 2000: Un sistema distribuido con datos compartidos puede tener a lo mucho dos de las siguientes propiedades:
    - Consistencia (Consistency)
    - Disponibilidad (Availability)
    - Tolerancia a Particionamientos de red (Partition tolerance).
- **Consistencia**: Después de una escritura exitosa, las lecturas posteriores siempre incluyen dicha escritura.
- **Disponibilidad**: Siempre se puede leer y escribir en el sistema. Toda petición (lectura/escritura) recibida por un nodo que se encuentra operativo debe proporcionar una respuesta.
- **Tolerancia a Particionamientos de Red**: La red podrá perder arbitrariamente varios mensajes enviados de un nodo a otro en presencia de particiones (Gilbert and Lynch)



- Particionamiento: No solo paquetes perdidos
    - Servidor caido (Una partición)
        - Todos los paquetes enviados hacia el se pierden.
        - La falla mas sencilla de manejar (Hay seguridad que el servidor no da respuestas erróneas)
- CA Systems
    - **Consistencia, Disponibilidad – No Particionamiento**
        - Único Servidor: Sistema Monolítico (No red, no particion y CA garantizadas)
    - **Sistemas Distribuidos que sea CA (Multi Nodo)**
        - Los mensajes en la red nunca se pierden o retrasan
        - Los servidores nunca se caen
    - Sistemas asi NO EXISTEN
    - Ante la presencia de particiones de red (Errores), ¿qué se sacrifica, Consistencia o Disponibilidad?
        - La posibilidad de particiones siempre esta presente
    - La decisión no es mutuamente exclusiva
        - El sistema no será completamente consistente ni completamente disponible
        - Combinación de las dos que se adapta a las necesidades.
- **Consistencia sobre Disponibilidad**
    - Garantiza la atomicidad de lecturas y escrituras rechazando algunas peticiones.
        - Bajar el sistema por completo
        - Rechazar escrituras (Two-Phase Commit)
        - Lecturas y escrituras en las particiones cuyo master esta en esa partición.
- **Disponibilidad sobre Consistencia**
    - Responderá a todas las peticiones
        - Lecturas desactualizadas
        - Aceptando conflictos de escritura → Mecanismos para resolver inconsistencias (vector clocks)

        - Hay varios sistemas donde es posible manejar resolución conflictos y en los cuales lecturas desactualizadas son aceptables.
- **Disponibilidad o Consistencia nunca ambas**
    - Sistema que dice ser CA : Servidores A, B y C

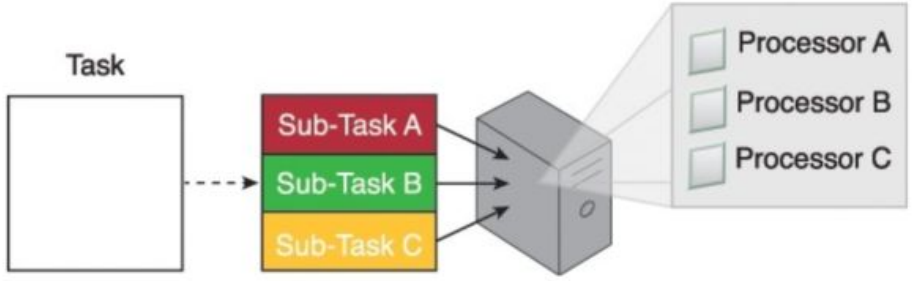
      {A,B} {C}

      Petición de escritura a C para actualizar un dato
        - 1. Aceptar la escritura sabiendo que ni A ni B sabrán de ella (hasta que la red vuelva a la normalidad)
        - 2.Rechazar la escritura, sabiendo que el cliente no podrá contactar A o B (hasta que la red vuelva a la normalidad)
    - Se debe escoger Disponibilidad (opcion 1) o Consistencia (opción 2)
    - En lugar de pensar cual de las dos propiedades nuestro sistema requiere (CA), pensar hasta donde puedo sacrificar cada una, antes que mi sistema empiece a funcionar mal.
    - No importa lo que hagamos siempre habrá fallas en el sistema.
        - Dejar de responder peticiones (lectura/escritura)
        - Dar respuestas basadas en información incompleta

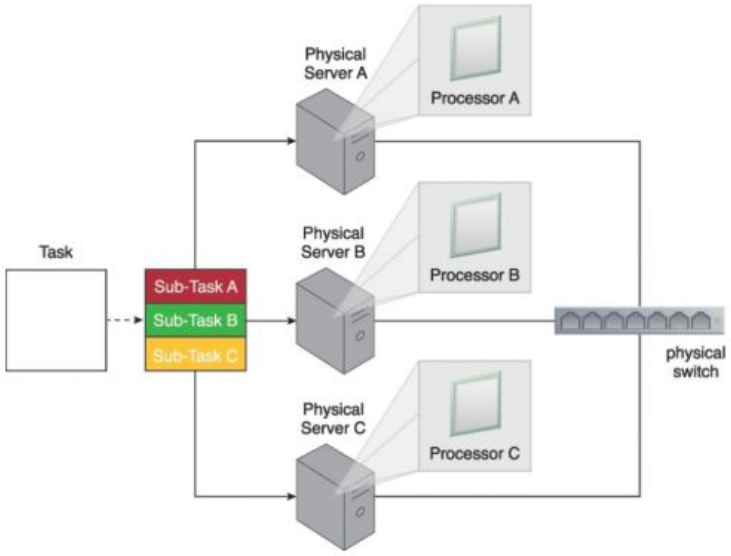
Capítulo 3

Conceptos y Técnicas de Procesamiento de Big Data

Procesamiento Paralelo



Procesamiento Distribuido



Actualización síncrona

- Con actualizaciones sincrónicas, los clientes se comunican directamente con la base de datos y bloquean hasta que se completa la actualización

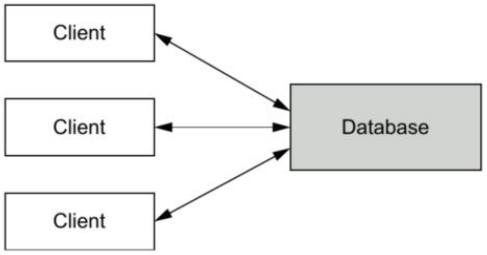


Fig: Arquitectura de capa de velocidad simple que utiliza actualizaciones sincrónicas

- Con actualizaciones síncronas, los clientes envían las actualizaciones a la cola e inmediatamente proceden con otras tareas.
- Después de un tiempo, el stream processor lee un lote de mensajes de la cola y aplica las actualizaciones

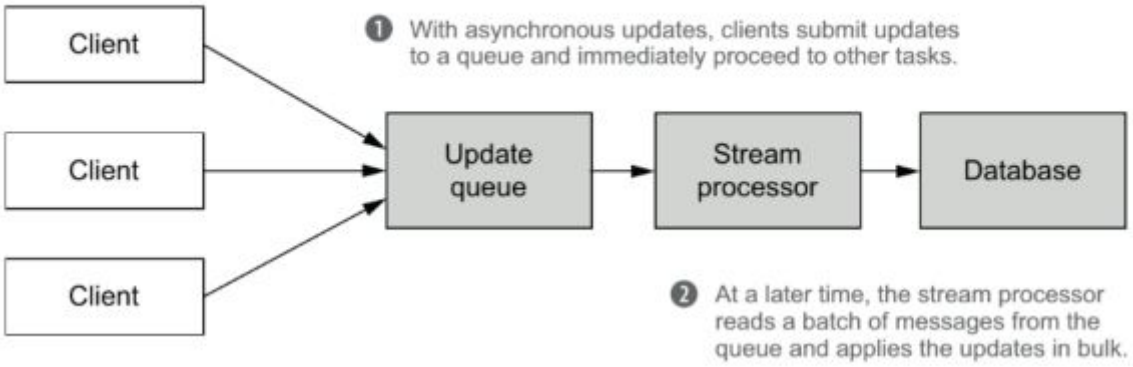


Fig x: Actualizaciones asincrónicas proporcionan mayor rendimiento y manejan fácilmente cargas variables

Colas y Procesamiento de Streams

- Arquitecturas Asíncronas
    - Colas
    - Stream Processing
- Procesamiento sin colas persistentes
    - Dispara y Olvida (Fire and Forget)
    - Tráfico?

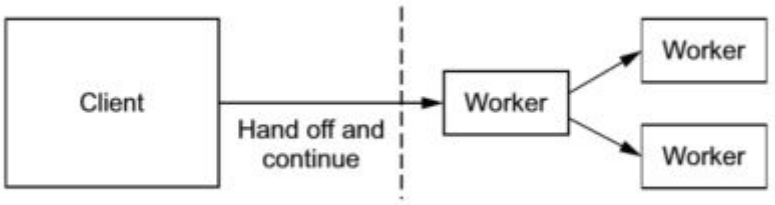


Fig: Para implementar procesamiento asincrónico sin colas, un cliente envía un evento sin monitorear si su procesamiento es exitoso.

Servidor de colas de único consumidor - Single-consumer queue server

- Los mensajes se eliminan de la cola cuando son confirmados
- Múltiples aplicaciones consumiendo los mismos eventos?
- El problema es que la cola controla que fue consumido y que no

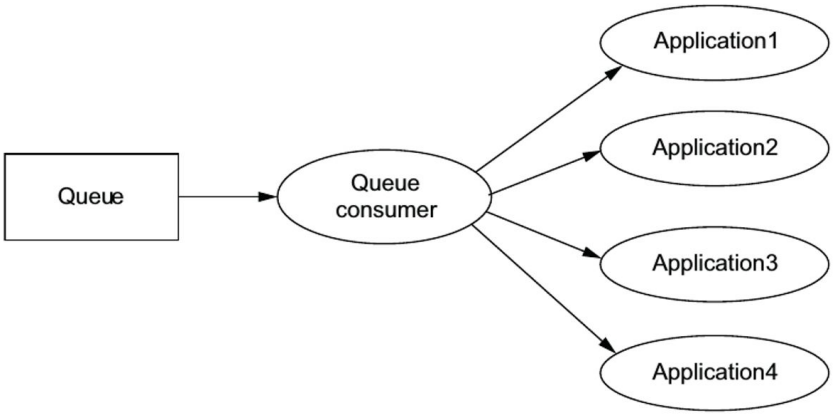


Figure 14.2 Multiple applications sharing a single queue consumer

Multi-consumer queues

Pasar el control de los eventos consumidos a la aplicación. Con cola multiconsumidor, las aplicaciones requieren ítems específicos de la cola y es responsable del tracking de los procesos exitosos de cada evento

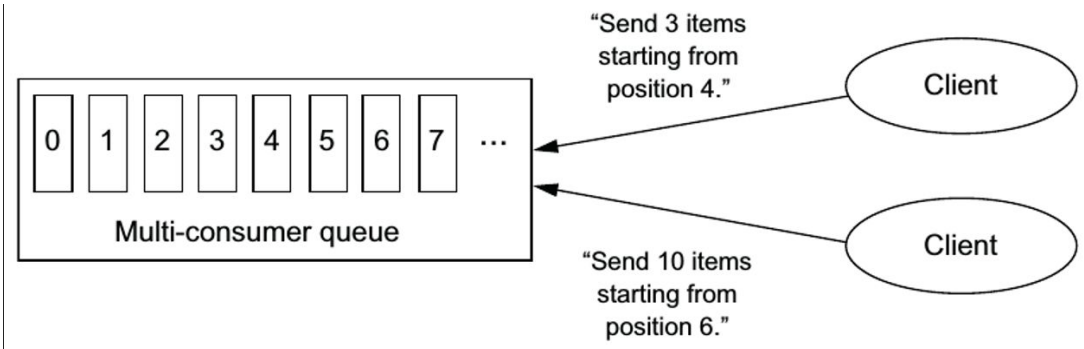


Figure 14.3 With a multi-consumer queue, applications request specific items from the queue and are responsible for tracking the successful processing of each event.

Stream processing

- Procesar los eventos y actualizar las vistas en tiempo real
    - Uno a la vez
    - Micro Batches

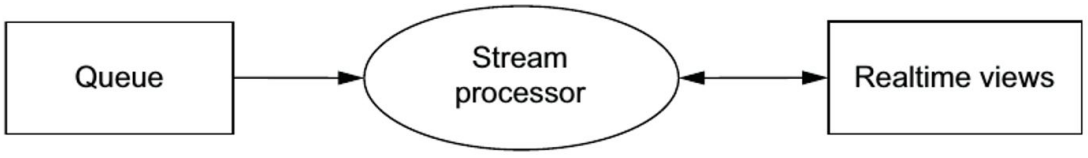


Figure 14.4 Stream processing

	One-at-a-time Uno a la vez	**Micro-batched Micro-por lotes**
Baja latencia	✔	
Alto Rendimiento		✔
Semántica al menos una vez	✔	✔
Semántica exactamente una vez	en algunos casos	✔
Modelo de programación simple	✔	

Colas y Procesadores(Workers)

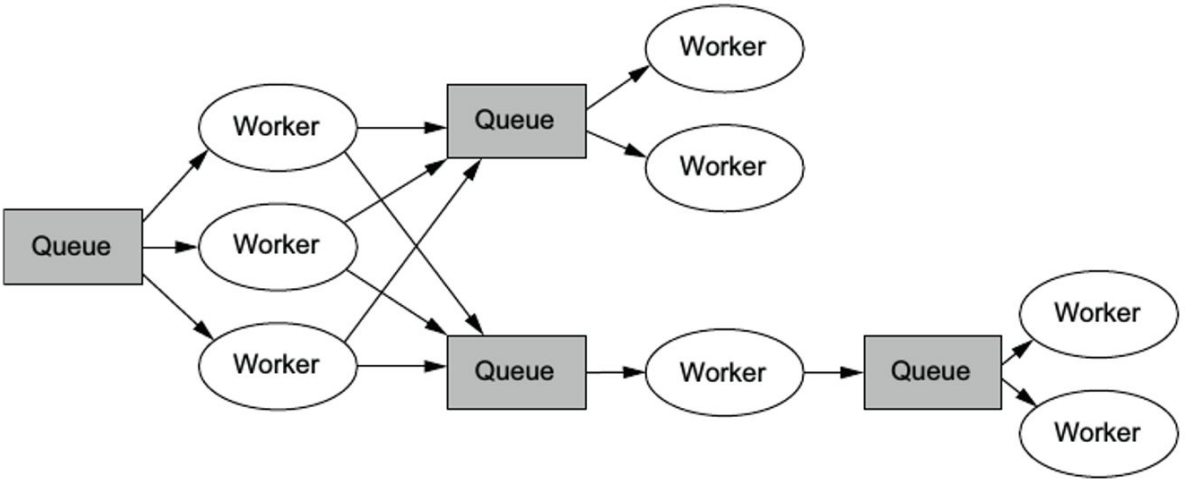


Figure 14.6 **A representative system using a queues-and-workers architecture. The queues in the diagram could potentially be distributed queues as well.**

Un sistema representativo usado en arquitectura de colas y trabajadores. Las colas en el diagrama pueden potencialmente ser colas distribuidas

Pregunta	Respuesta
Señale lo correcto respecto al modelo de distribución Sharding. Seleccione una o más de una: ☐ a. Por si solo ayuda de manera notable a mejorar la capacidad de recuperación ante fallos ☐ b. Jamás se puede usar en combinación con otro modelo de distribución. ☐ c. Diferentes partes de la data se almacena en diferente servidores ☐ d. Facilita el escalamiento horizontal de lecturas y escrituras	c, d
Referente al teorema CAP, señale lo correcto. Seleccione una o más de una: ☑ a. Hablar de Consistencia sobre Disponibilidad significa garantizar la atomicidad de lecturas y escrituras rechazando algunas peticiones. ☐ b. Es posible garantizar sistemas distribuidos que sean 100% consistentes y 100%disponibles. ☑ c. Los particionamientos en la red es algo que se puede ignorar. ☑ d. La consistencia significa que ante una escritura exitosa, las lecturas posteriores siempre incluirán dicha escritura.	a, d
Las bases de datos documentales permiten buscar únicamente por la clave Seleccione una: ◉ Verdadero ○ Falso	Falso
Sharding se refiere a hacer copias exactas de los datos en diferentes servidores. Seleccione una: ◉ Verdadero ○ Falso	Falso

Señales lo correcto respecto a la replicación en mongoDB Seleccione una o más de una: ☐ a. Los árbitros jamás pueden covertirse en masters si el máster actual falla. ☑ b. El máster y los esclavos pueden atender peticiones de escritura ☐ c. Los árbitros a más de almacenar una copia de los datos, votan en la elección de un nuevo master si el máster actual falla. ☑ d. En un replicaSet pueden haber esclavos que no participen en una elección.	a, d
La consistencia de sesión garantiza que durante una sesión un usuario puede leer sus propias escrituras. Seleccione una: ◉ Verdadero ○ Falso	V
Cuáles de los siguientes tipos de bases de datos NoSQL son orientados a la agregación Seleccione una o más de una: ☐ a. Clave-Valor ☐ b. Bases de datos orientas a columnas ☑ c. Bases de datos de Grafos ☐ d. Bases de Datos Documentales	a,b,d
Al hablar de distribución de datos se puede utilizar replicación o sharding pero nunca ambas simultáneamente. Seleccione una: ◉ Verdadero ○ Falso	F
Enlace según corresponda Permiten ejecutar consultas basadas en la estructura interna de los agregados — BDs Clave-Valor Son útiles para almacenar datos que representan relaciones complejas como redes sociales, preferencias de usuarios, etc. — BDs de grafos No usan filas como unidad de almacenamiento y se usan en escenarios donde se necesita leer simultáneamente unas cuantas columnas de varias filas — BDs orientadas a columnas Colección de objetos relacionados que queremos tratar como unidad de manipulación de datos y consistencia — Agregación Generalmente no permiten ejecutar consultas basadas en los campos de una agregación — BDs de grafos	Documentales Grafos Columnas Agregación Clave Valor
El teorema CAP hace referencia a ciertas características que deben cumplir los sistemas distribuidos. Señale cuáles son dichas características Seleccione una o más de una: ☐ a. Rendimiento ☑ b. Tolerancia a particionamientos en la red ☐ c. Tolerancia a fallos ☑ d. Consistencia ☐ e. Usabilidad ☑ f. Disponibilidad	b, d, f