Bases de Datos NoSQL: Distribución

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: https://citius.gal/team/jose-ramon-rios-viqueira

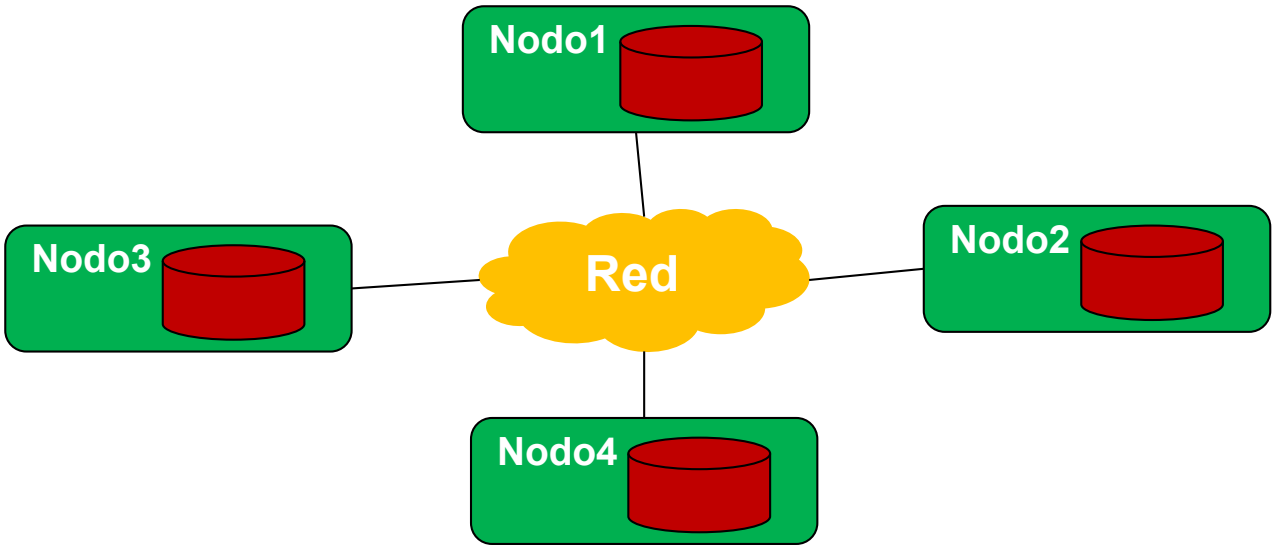
Curso 2023/2024

Guion

- Introducción
- Particionamiento (Sharding)
- Replicación
- Procesamiento (Map-Reduce)

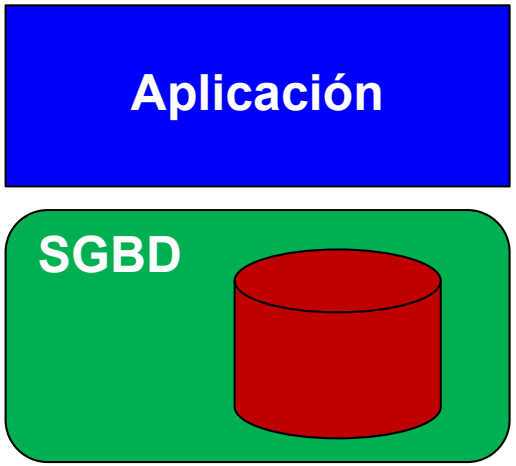
Introducción

- **Principal interés del uso de tecnologías NoSQL**
  - Funcionar aprovechando los recursos de un **cluster** de computación
  - Uso del **agregado** como unidad natural de distribución de datos.
- **Ventajas**
  - Capacidad para gestionar mayor volumen de datos
  - Incrementar el tráfico de operaciones de lectura y escritura (rendimiento)
  - Incrementar la disponibilidad (tolerancia a fallos)
- **Desventaja**
  - Sistema más complejo. Usar solo en caso necesario



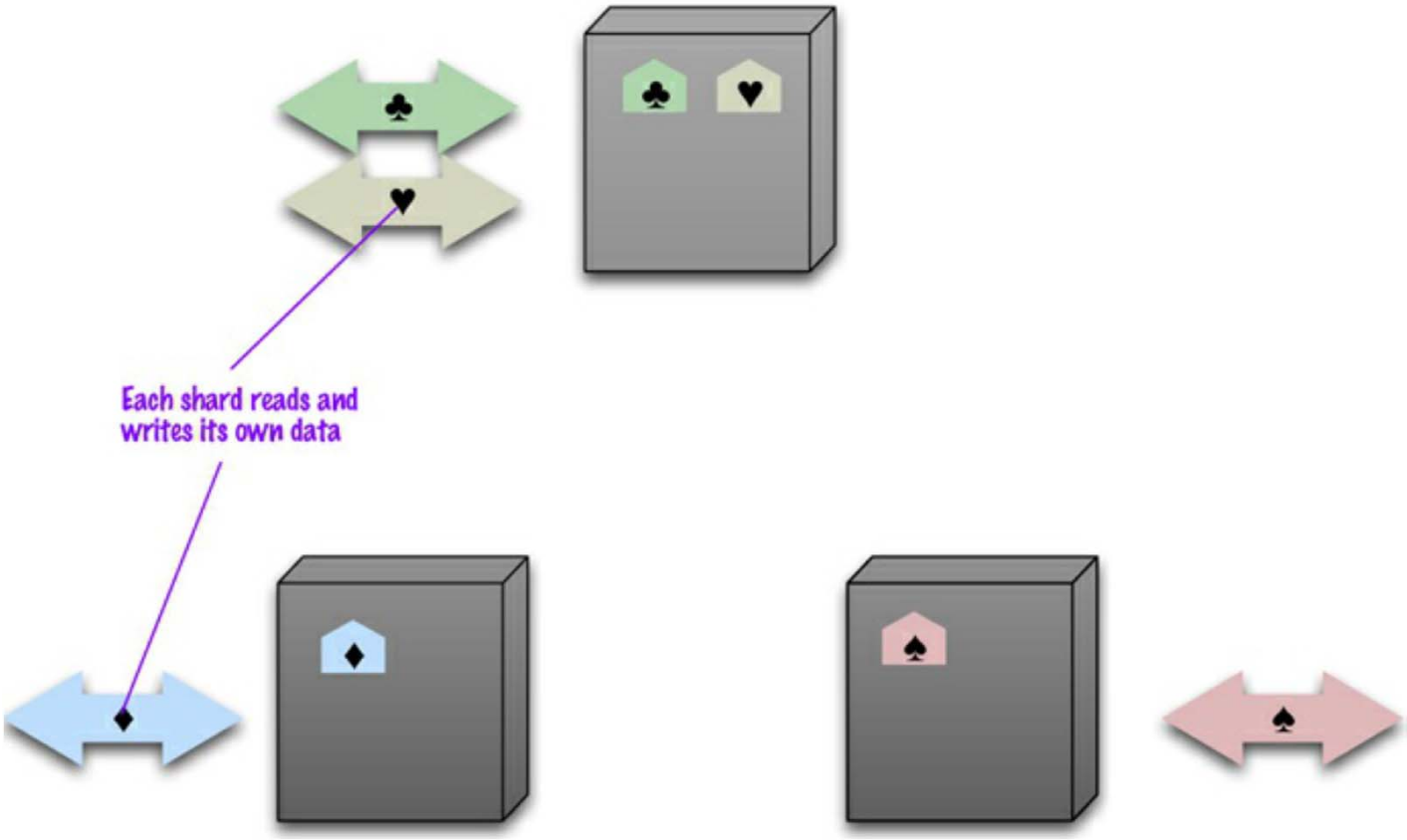
- **Formas de distribución de los datos**
  - **Replicación**
    - Varias copias del mismo dato en distintos nodos
    - Arquitecturas
      - Maestro-esclavo
      - Peer-to-peer
  - **Particionamiento (Sharding)**
    - Repartir los datos entre los nodos
  - Técnicas ortogonales: Se pueden combinar
- **Técnicas** (de más sencillo a más complejo)
  - Un solo servidor
  - Replicación Maestro-Esclavo
  - Particionamiento
  - Replicación peer-to-peer

- **Un solo servidor**
  - No existe la distribución de datos
  - Preferida por ser la más simple
  - El uso de NoSQL se justificaría solo por cuestiones relacionadas con el modelo de datos
    - Ej: Las BD de Grafos se suelen utilizar con esta arquitectura
    - Datos agregados que se procesan en el nivel de aplicación y se recuperan juntos



Particionamiento (Sharding)

- **Varios usuarios accediendo a partes distintas de la base de datos**
  - Colocar partes distintas de los datos en nodos distintos
  - Caso ideal: Cada usuario acaba accediendo a un nodo distinto
  - **Escalabilidad horizontal**



Particionamiento (Sharding)

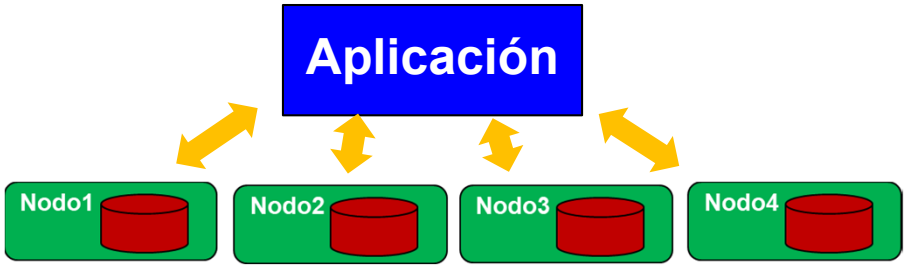
- **Localidad de los datos**
  - Intentar que los datos que se acceden juntos estén almacenados en el mismo nodo y cerca en el disco
  - ¿Cómo?
    - Colocar datos que se acceden juntos dentro del mismo agregado
    - Utilizar el agregado como unidad de datos para la distribución
  - Incluso mantener juntos agregados que suelen accederse juntos
    - Cuando se conoce el orden más típico de acceso

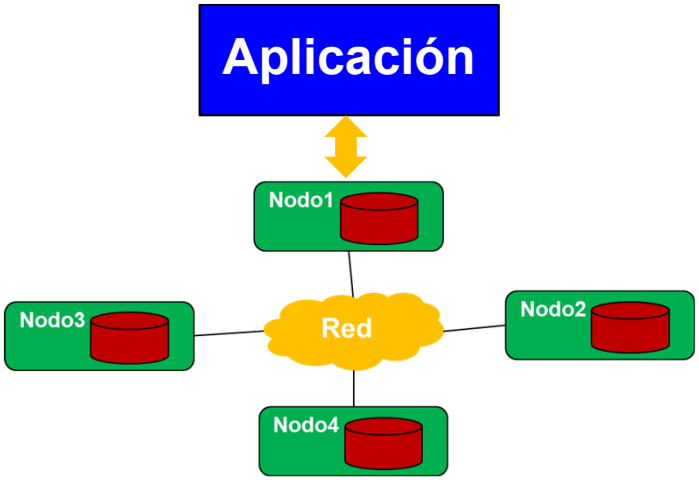
join precalculado se parece mucho a tener un agregado

- Tener los datos lo más cerca posible de donde son utilizados con más frecuencia
  - Ejemplo: Los pedidos de Boston almacenados en el servidor de la costa este
  - **MongoDB**: Uso de Zonas en el Sharding
- Mantener todos los nodos con una carga de datos similar
  - Utilizar distribuciones uniformes de los datos
    - Misma probabilidad para todos los nodos

Particionamiento (Sharding)

- **Implementación del Sharding en el nivel de aplicación**
  - Aplicaciones con código mucho más complejo
  - Necesidad de cambiar la aplicación al reorganizar datos
- Muchas BD NoSQL hacen una gestión automática del Sharding
  - Asumen la responsabilidad del particionamiento
  - Simplifican la labor de la aplicación
- Rendimiento
  - Particionamiento mejora lecturas y escrituras
  - Replicación puede mejorar lecturas, pero puede empeorar escrituras
- Fiabilidad
  - Sharding no mejora la disponibilidad
  - Puede haber fallos parciales
  - Aumenta la probabilidad de fallo.

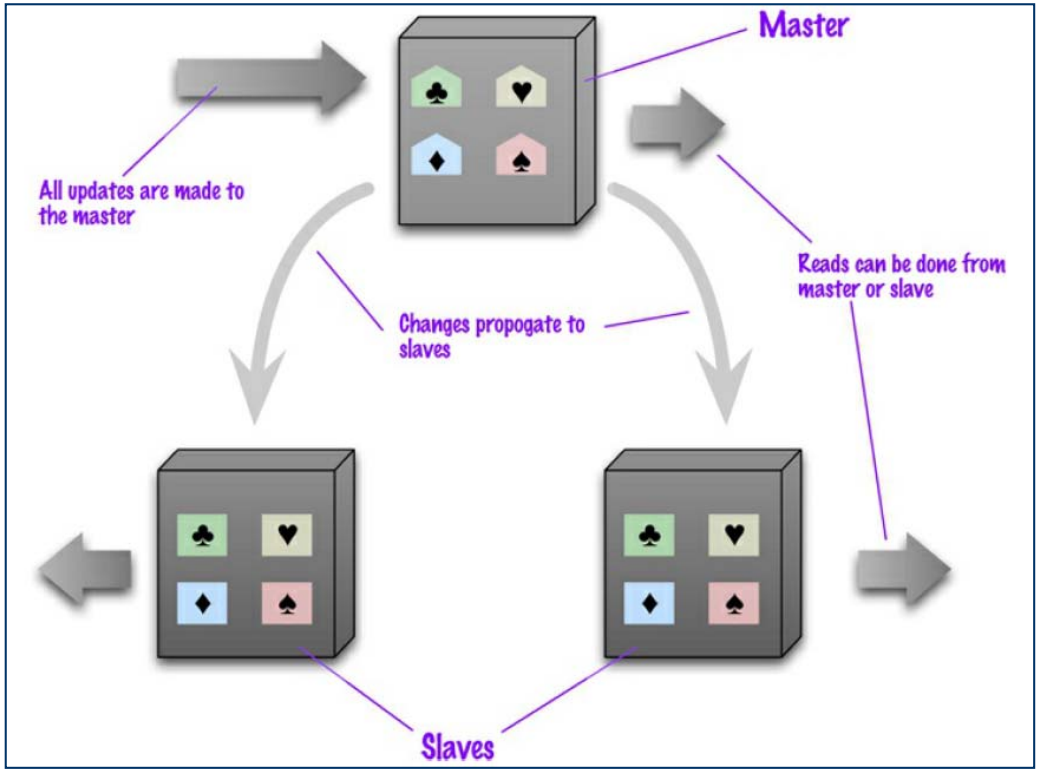




Replicación maestro-esclavo

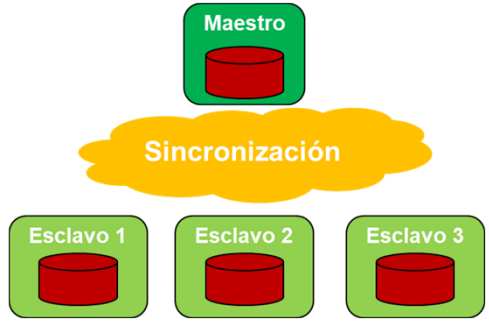
cuanto más sincrono, mas consistencia pero bajando rendimiento y disponibilidad

- Datos replicados en varios nodos.
- Un nodo elegido como maestro (o primario)
  - Autoridad como fuente de los datos y responsable de su actualización
- Resto de nodos: esclavos (o secundarios)
- Proceso de replicación sincroniza los esclavos con el maestro



Replicación

Replicación maestro-esclavo



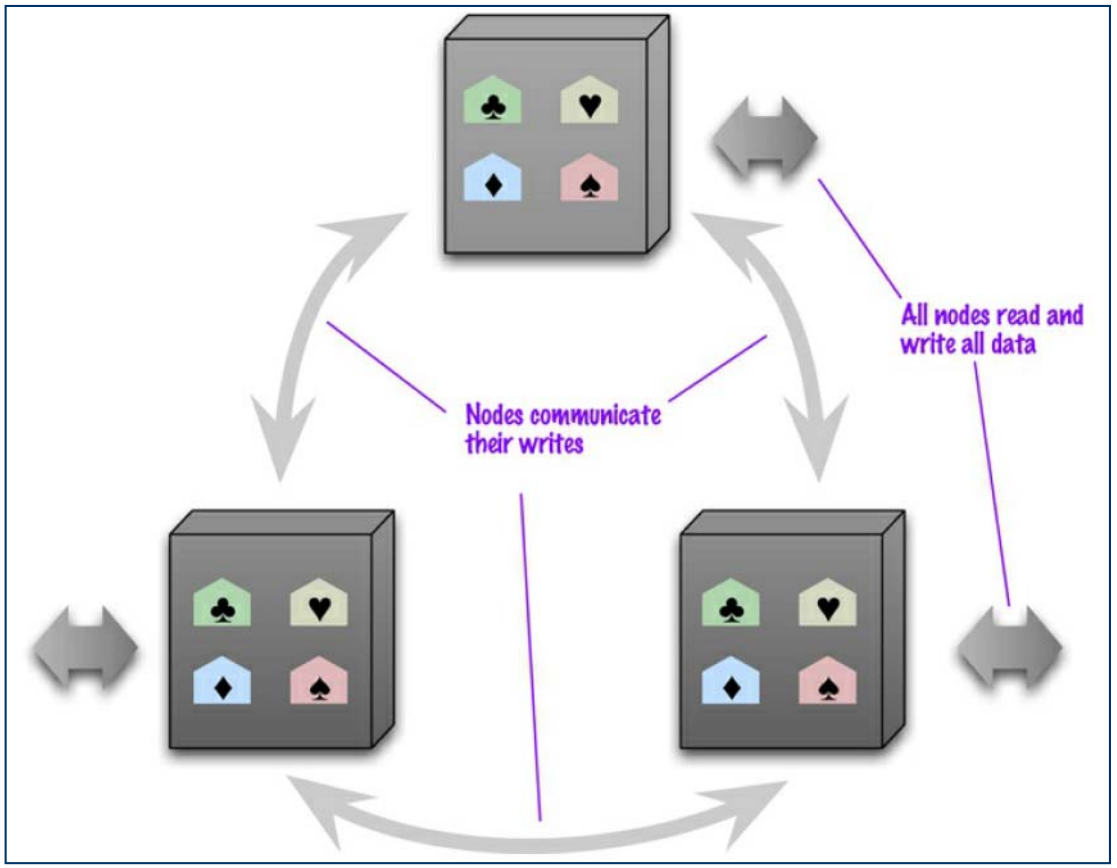
- Buena solución cuando la aplicación es intensiva en lecturas (sincronización)
- **Ventajas**
  - **Alta disponibilidad para lecturas (IMPORTANTE)**
    - Si el maestro falla los esclavos pueden atender peticiones de lectura
  - Sustitución del maestro ante un fallo
    - Si tenemos replicación completa del maestro podemos hacer recuperación en caliente
    - sustitución del maestro manual o automática
- **Problemas**
  - Maestro cuello de botella en modificaciones. Problemas con muchas escrituras
  - Baja disponibilidad en escrituras.
  - Problemas de **Consistencia**
    - Lecturas de esclavos distintos pueden dar resultados distintos para el mismo dato
    - Un cliente podría no conseguir leer una actualización recién escrita por el mismo
      - Escritura del maestro y lectura del esclavo antes de sincronizar.

Replicación

- **Replicación peer-to-peer**
  - **Motivación**: problemas de la arquitectura maestro-esclavo
    - Escalabilidad en escrituras: maestro cuello de botella.
    - Baja disponibilidad: maestro punto único de fallo.

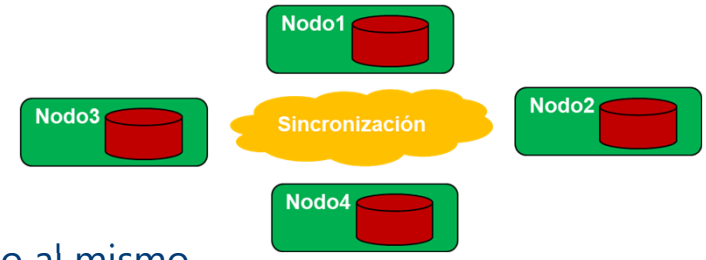
problemas

  - **Solución**: Eliminar la distinción entre maestro y esclavos



Replicación

- **Replicación peer-to-peer**
  - **Problemas**
    - Consistencia
      - Dos usuarios modificando el mismo dato al mismo tiempo en dos nodos distintos (conflicto write-write)
        - **Grave!!**: Las inconsitencias read-write crean problemas transitorios, pero las write-write, crean problemas que perduran.
    - Se verán soluciones más detalladas en la parte de consistencia
  - **Soluciones generales**
    - Coordinar las replicas durante las escrituras
      - Coste de red adicional para la coordinación
      - Con actualizar la mayoría de forma coordinada sería suficiente
        en la maestro-escalvo actualizas un nodo, aqui hay que ir a soluciones como esta
    - Asumir las inconsistencias e intentar arreglarlas combinando réplicas



Compromiso

Consistencia vs Disponibilidad

Replicación

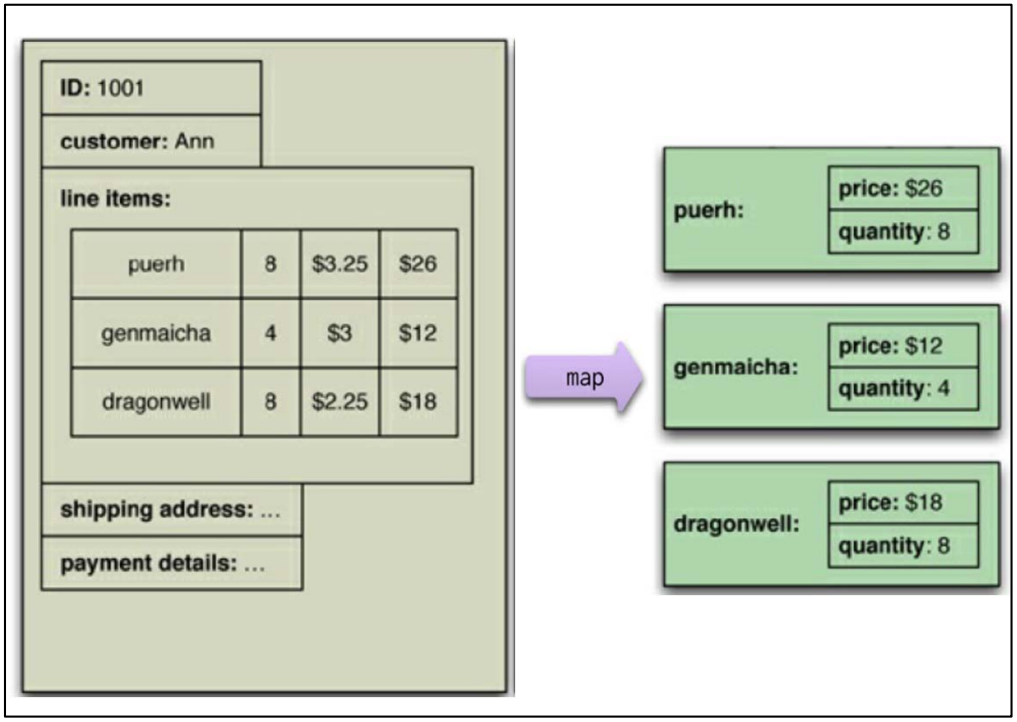
- **Combinación de particionamiento y replicación**
  - Con replicación maestro-esclavo
    - Un maestro único para cada partición
    - Dependiendo de la configuración
      - Elegir maestro y esclavos a nivel de cluster
      - Elegir maestro y esclavos para cada partición
  - Con replicación peer-to-peer
    - Muy común en soluciones NoSQL con modelos de tipo column-family
    - Replicación con factor 3
      - Cada partición tiene tres copias en tres nodos
      - Si un nodo falla, las particiones de ese nodo se distribuyen entre los demás.

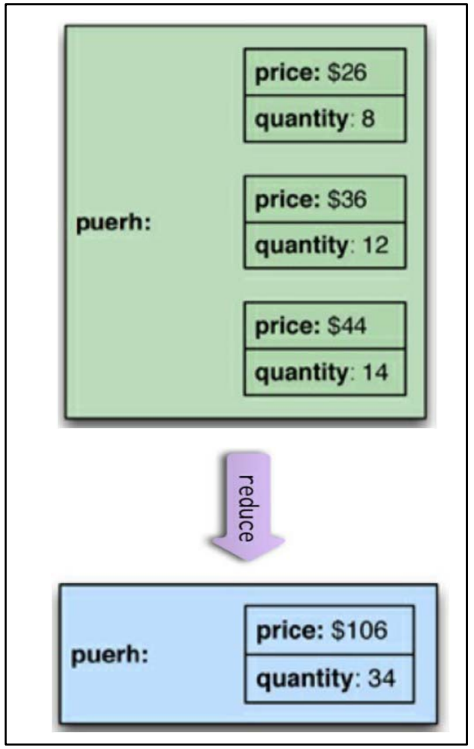
Procesamiento (map-reduce)

- **Cambios en la forma de procesar**
  - Arquitectura centralizada
    - Procesamiento en el cliente
      - (+) Más flexible. (-) Mover datos del servidor al cliente
    - Procesamiento en el servidor
      - (-) Menos cómodo para programar. (+) Más eficiente
  - Arquitectura distribuida
    - Muchas máquinas entre las que repartir la carga de procesamiento
    - Intentar minimizar el tráfico de datos entre nodos
- **Patrón Map-Reduce**
  - Forma de programar el procesamiento para minimizar tráfico entre nodos
  - Ejemplo de plataforma: Apache Hadoop
  - Varias bases de datos NoSQL tienen su propia implementación
  - Basado en operaciones Map y Reduce de lenguajes de programación funcional

Map-Reduce básico

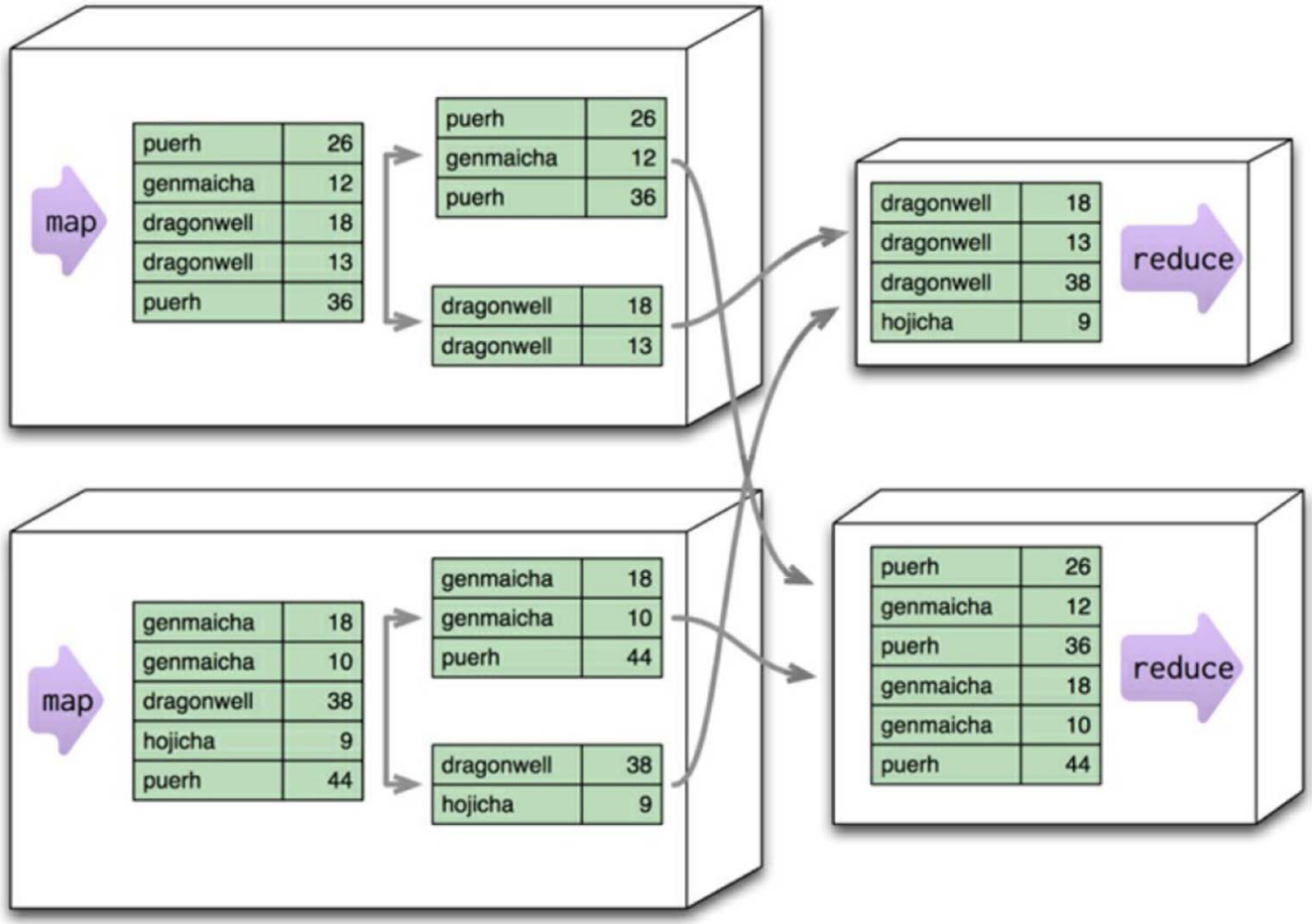
- Ejemplo: Obtener los ingresos por cada producto en los últimos 7 días.
  - Necesitamos procesar en todos los nodos
- **Map**: Para cada agregado genera un conjunto de pares clave-valor.
- **Reduce**: Sobre el resultado del Map, agrega valores de elementos con la misma clave.





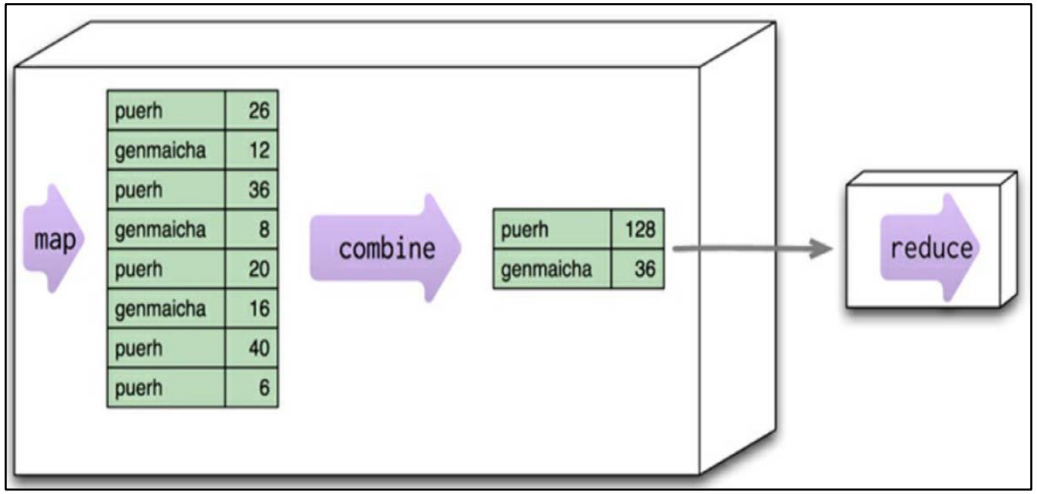
Procesamiento (map-reduce)

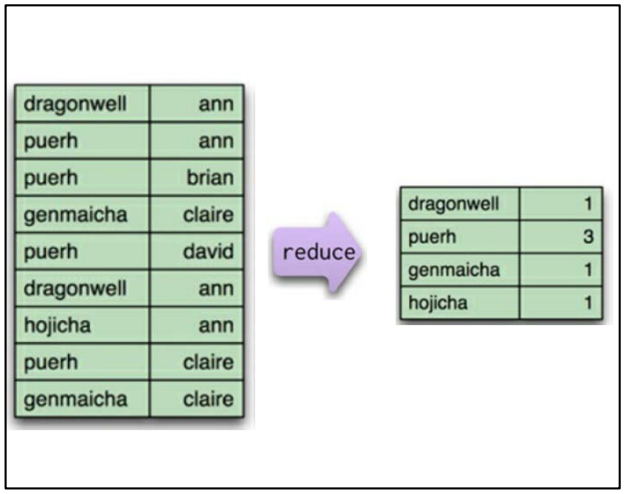
- **Paralelización de la ejecución de la operación reduce**
  - Las operaciones Map en cada agregado son independientes con lo que se pueden paralelizar fácilmente
  - Paralelizar reduce necesita reparticionar la salida del map (**Shuffling**)



Procesamiento (map-reduce)

- **Paralelización de la ejecución de la operación reduce**
  - Minimizar el movimiento de datos entre el map y el reduce
  - Reducer combinable
    - Misma forma en la entrada que en la salida
    - Se puede aplicar en cada nodo antes de enviar los datos.
    - Se envían datos ya parcialmente agregados
    - Se podría empezar el reduce incluso antes de terminar el map
  - No todas las operaciones reduce son combinables
    - Ejemplo: Una función que cuenta. Combina sumando y no contando
  - Algunos sistemas solo admiten reducers combinables
    - Si no lo son hay que separar el procesamiento en pasos (pipelines)

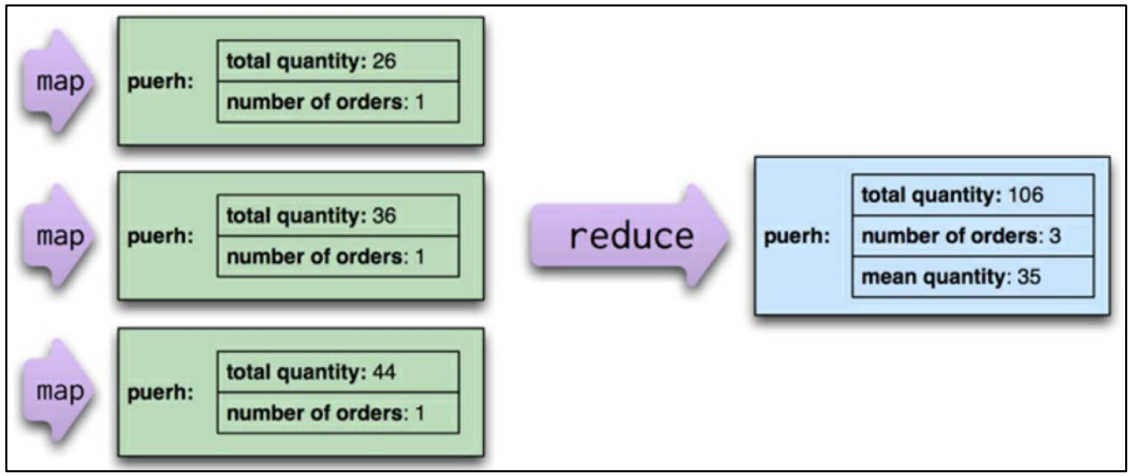


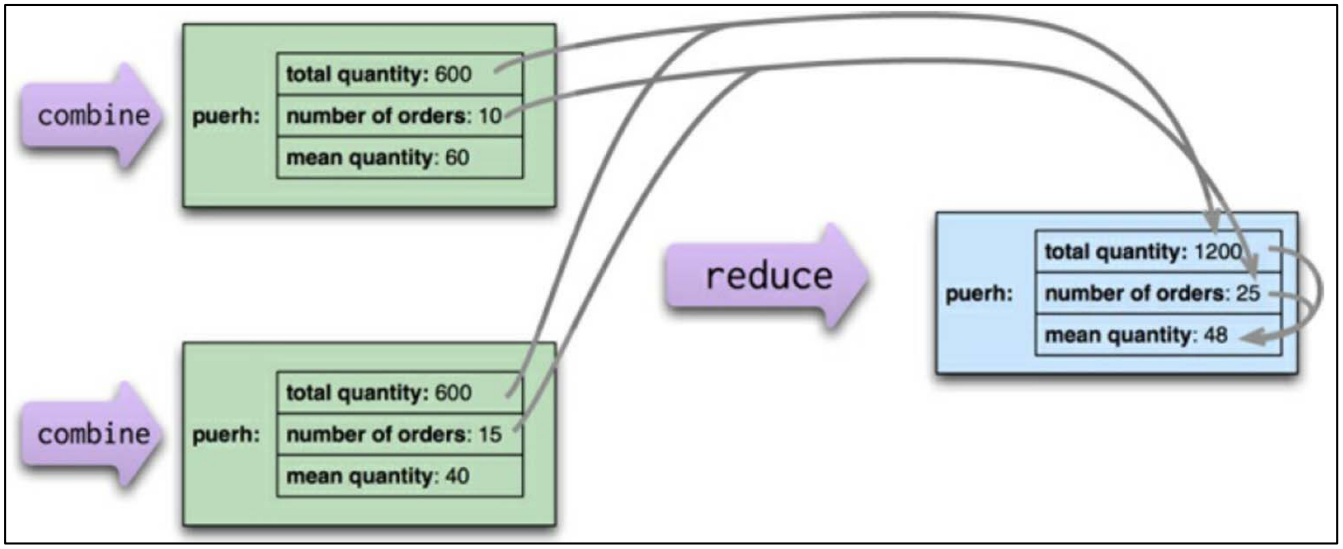


Procesamiento (map-reduce)

Componiendo cálculos Map-Reduce

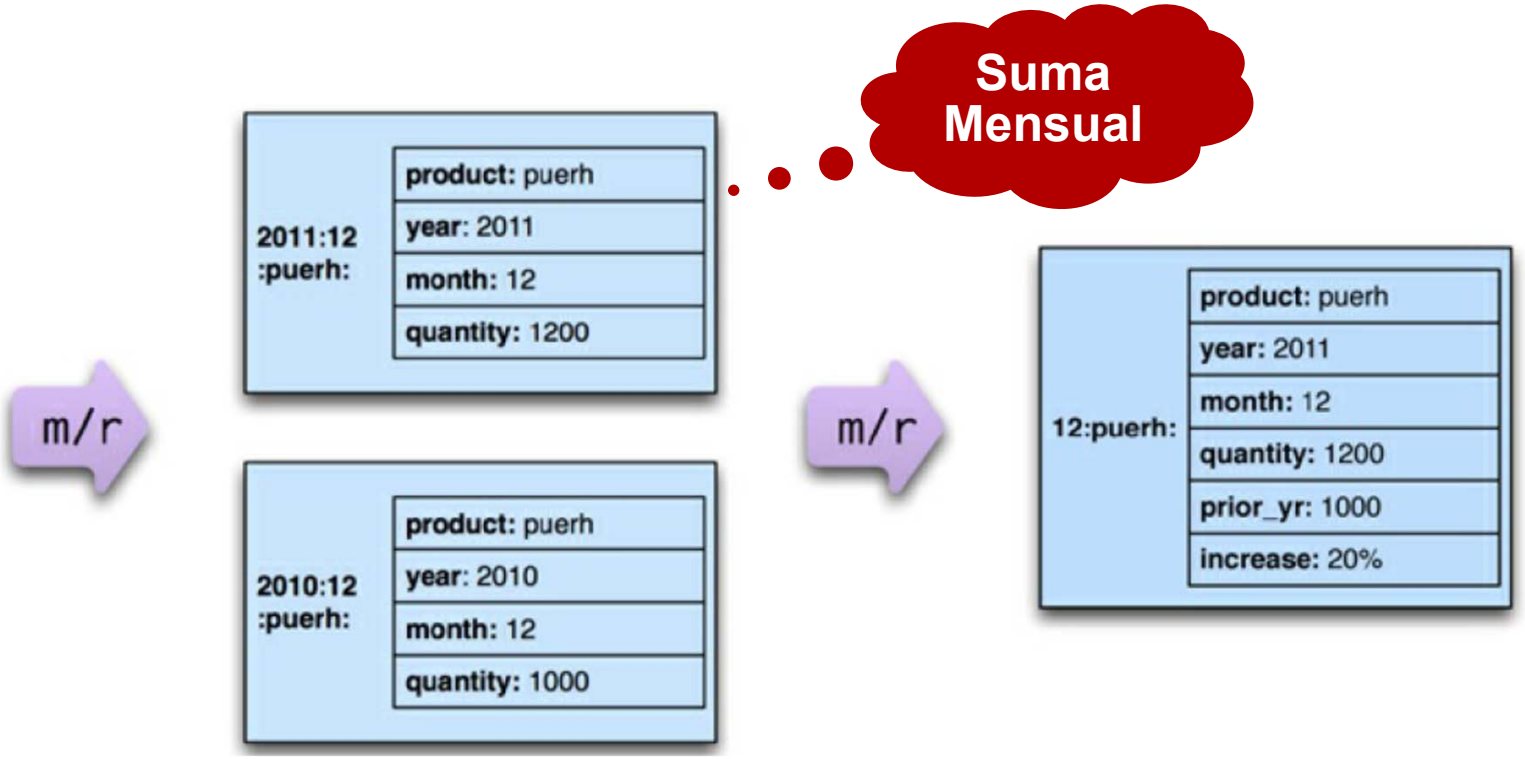
- Restricciones en los cálculos
  - Tarea map opera solo sobre un agregado
  - Tarea reduce opera solo sobre una clave
- Ejemplo: No podemos combinar dos medias parciales para obtener la media total
  - Necesitamos calcular una suma y una cuenta





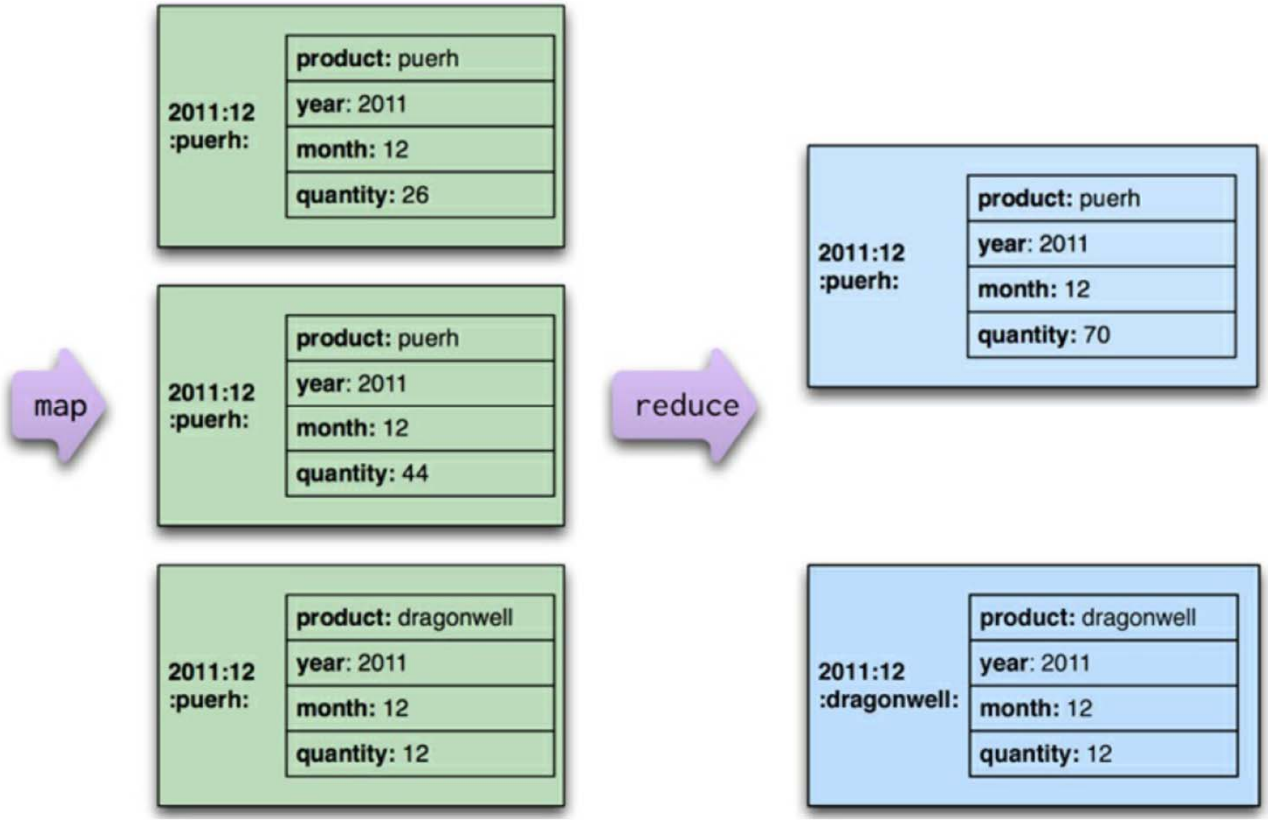
Componiendo cálculos Map-Reduce

- En casos más complejos necesitaremos descomponer el cálculo en varias etapas Map-Reduce
- Ejemplo
  - Comparar las ventas de cada mes de 2011 con el mismo mes del año anterior



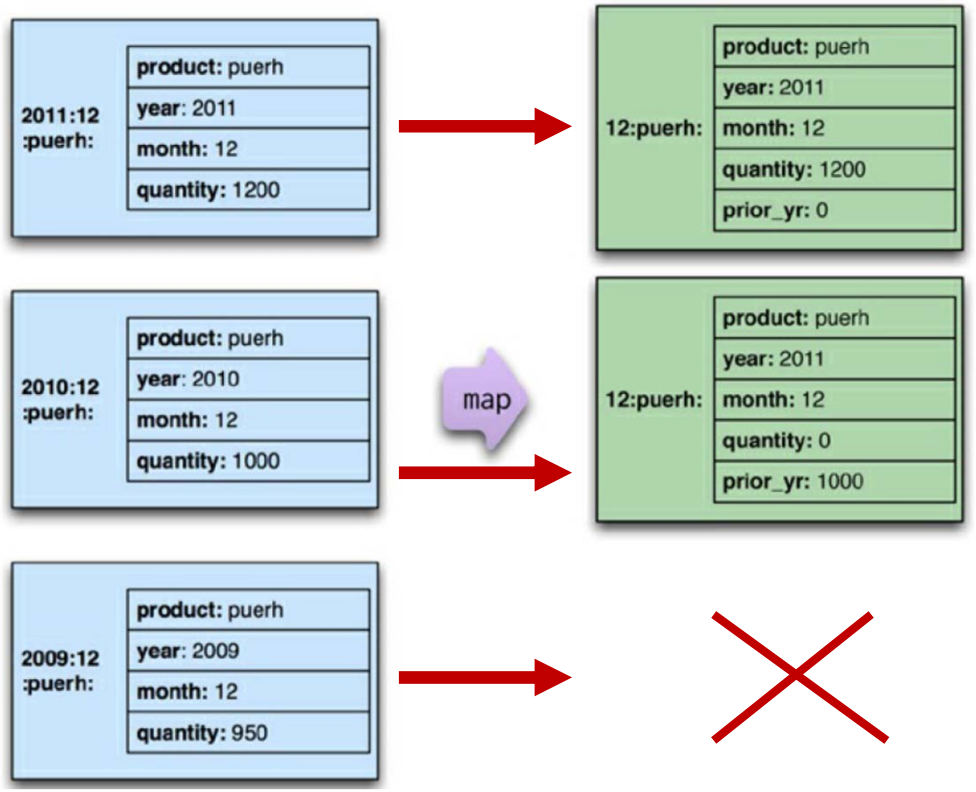
Componiendo cálculos Map-Reduce

- En casos más complejos necesitaremos descomponer el cálculo en varias etapas Map-Reduce
- Ejemplo
  - Comparar las ventas de cada mes de 2011 con el mismo mes del año anterior
    - Calculo de la suma mensual (**Map-Reduce**)



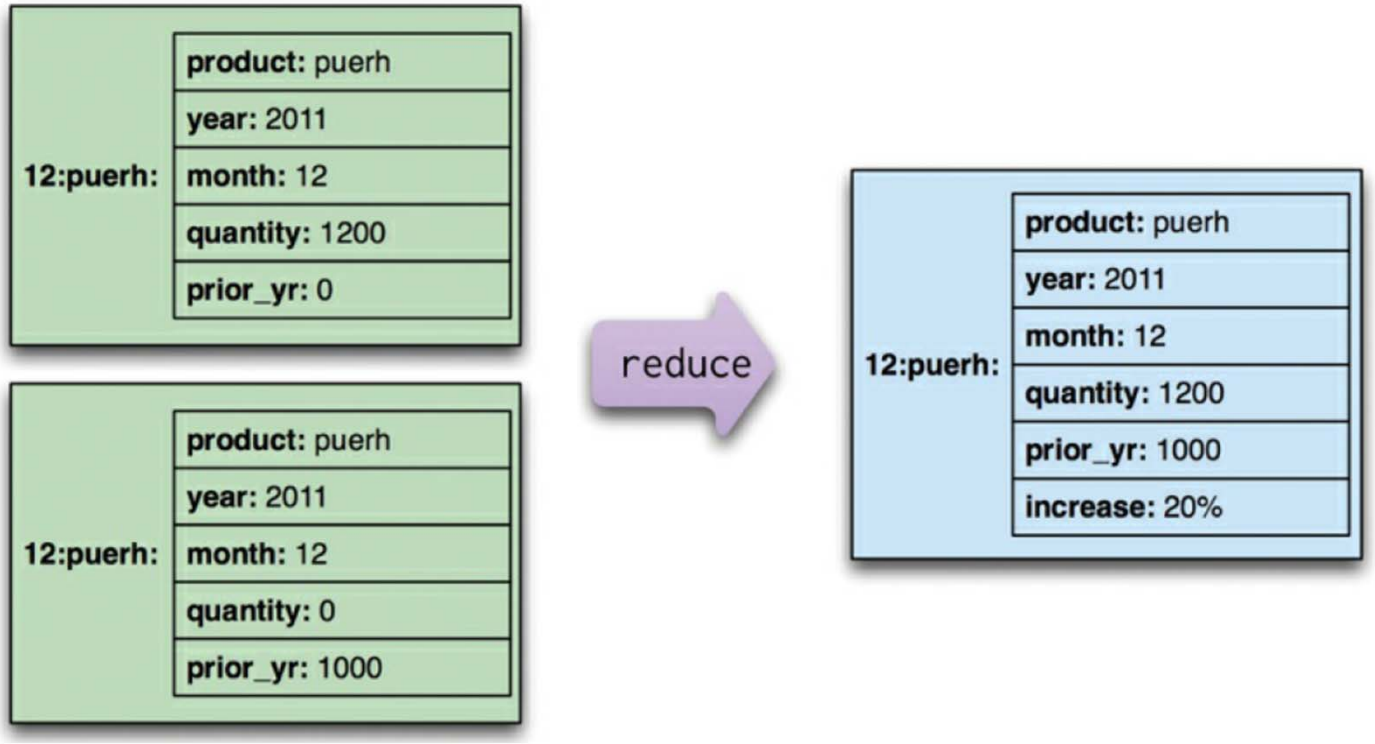
Componiendo cálculos Map-Reduce

- En casos más complejos necesitaremos descomponer el cálculo en varias etapas Map-Reduce
- Ejemplo
  - Comparar las ventas de cada mes de 2011 con el mismo mes del año anterior
    - Cálculo de cantidad de año 2011 y cantidad del año previo (**Map**)



Componiendo cálculos Map-Reduce

- En casos más complejos necesitaremos descomponer el cálculo en varias etapas Map-Reduce
- Ejemplo
  - Comparar las ventas de cada mes de 2011 con el mismo mes del año anterior
    - Resultado final (**Reduce**)



Procesamiento (map-reduce)

Componiendo cálculos Map-Reduce

- Lenguajes especializados en este tipo de programación (ecosistema Hadoop)
  - Pig
  - Hive (Sintaxis similar a SQL)
- Map-Reduce es un paradigma importante fuera de NoSQL, cuando trabajamos con archivos en sistemas distribuidos

Map-Reduce Incremental

- A veces los datos llegan a través de un flujo continuo
  - Empezar cada cálculo desde el principio con todos los datos puede no ser viable
- Operación Map es fácil de ejecutar de forma incremental
  - Cada tarea Map es independiente de las demás
- Tareas Reduce dependen de muchos resultados de diferentes Map.
  - Tareas reduce que se aplican a resultado Map que no cambiaron no necesitan ejecutarse de nuevo
  - Reducer combinable y cambios aditivos: solo hay que aplicar reduce sobre cambios.
    - Caso contrario: Usar redes de dependencias para recalcular solo las partes necesarias, dependientes de los cambios en la entrada.

Bases de Datos a Gran Escala
Master Universitario en Tecnologías de Análisis de da Datos Masivos: Big Data
Escola Técnica Superior de Enxeñaría (ETSE)
Universidade de Santiago de Compostela (USC)

Bases de Datos NoSQL: Distribución

José R.R. Viqueira

Centro Singular de Investigación en Tecnoloxías Intelixentes (CITIUS)
Rúa de Jenaro de la Fuente Domínguez,
15782 - Santiago de Compostela.
Despacho: 209
Telf: 881816463
Mail: jrr.viqueira@usc.es
Skype: jrviqueira
URL: https://citius.gal/team/jose-ramon-rios-viqueira

Curso 2023/2024