



Universidad
Rey Juan Carlos

Material docente en abierto de la
Universidad Rey Juan Carlos

Máster Universitario en Ingeniería de Sistemas de Decisión

Análisis de Big Data

2024-2025

Alberto Fernández Isabel
Felipe Ortega Soto



Copyright (c) 2025 Alberto Fernández Isabel, Felipe Ortega Soto.
Esta obra está bajo la licencia CC BY-SA 4.0, [Creative Commons](#)
[Atribución-Compartir Igual 4.0 Internacional](#)

0 - Introducción: Análisis de Big Data

Autor: Felipe Ortega

Análisis de Big Data

Máster Univ. Ingeniería de Sistemas de Decisión

Móstoles, Madrid

17 de febrero de 2025



Contenidos

0.1 Motivación y Objetivos

0.2 Contenidos y Temario

0.3 Criterios de evaluación

Section 1

0.1 Motivación y Objetivos

Motivación

► Importancia estratégica.

- Un gran número de empresas se enfrentan hoy día al reto de analizar conjuntos de datos complejos para extraer conocimiento valioso.
- Gran ventaja competitiva de aquellos profesionales y empresas que sepan cómo sacar el máximo partido del análisis de *big data*.
- Necesidad de constante actualización, incorporando nuevas herramientas tecnológicas y metodológicas, pero teniendo claro el propósito general y buenas prácticas de trabajo a seguir.

► Demanda laboral.

- Existe una intensa demanda de este tipo de profesionales en el mercado actual.
- Generalmente gira en torno a tecnologías (Spark, ecosistema Hadoop, NoSQL, etc.) o lenguajes de programación (Python, R, Scala, etc.).

Perspectivas presentes y futuras

► Principales sectores de aplicación.

- Ciencias de la salud, biotecnología.
- *Retail*, fabricación, logística.
- Servicios de Internet, comercio electrónico, servicios y aplicaciones móviles.
- Servicios inmobiliarios.
- Banca, finanzas e inversiones, seguros.
- Políticas públicas, servicios públicos.
- *Utilities* (energía, agua, comunicaciones...).
- Transportes.
- Márketing, publicidad, imagen de marca, análisis de mercado.
- Seguridad, servicios de inteligencia.

Perspectivas presentes y futuras

- ▶ El volumen de datos de negocio generado por empresas a nivel mundial se dobla aproximadamente cada 1,2 años [2].
- ▶ El big data ha sido explícitamente reconocido como un activo de valor añadido fundamental para la consecución de los objetivos estratégicos enumerados en la [Agenda Digital para Europa](#).
- ▶ The Economist estimó en 2010 [1] que el valor global de la industria del big data sobrepasaría los 100.000 millones de dólares, a un ritmo de crecimiento del 10% anual.
- ▶ Un estudio más reciente publicado por la consultora global IDC [3] predijo que “el mercado de tecnología y servicios de big data crecerá a una tasa CAGR del 27%, hasta los 32.400 millones de dólares en 2017 (6 veces el volumen de crecimiento del mercado TIC en su conjunto).
- ▶ Fundación BBVA estimó que en 2022 el 100% de las empresas comprarían y venderían datos como parte habitual de sus operaciones.

Perspectivas presentes y futuras

► Predicciones [*].

- Importancia de disponer de datos limpios y de calidad.
- Creciente tendencia a utilización de plataformas *cloud computing*.
- El análisis de contenido multimedia (audio, vídeo e imágenes) se triplicará en 2015-2016 (**cumplida**).
- Incremento sustancial de las plataformas de análisis de datos en tiempo real (**cumplida**).
- En 2017 se unificarán tecnologías de análisis de datos, búsquedas y gestión de información (**cumplida**).
- En 2019 el 100% de las empresas comprarán datos externos (**cumplida**).
- Hacia 2022 será posible la personalización en masa de productos y servicios (**cumplida**).

[*] <http://www.bbvaopen4u.com/es/actualidad/infografia-big-data-presente-y-futuro>

Objetivos

- ▶ Adquisición de unos conocimientos básicos generales sobre los principales **métodos, arquitecturas y herramientas tecnológicas** para análisis de *big data*.
- ▶ Apendicular los principales hitos dentro del diseño y realización de **proyectos de ciencia de datos**.
- ▶ Adquirir **conocimientos prácticos** sobre algunas de las principales herramientas y tipos de análisis que podemos utilizar para trabajar con *big data*.
- ▶ Conocer las principales **tendencias de evolución** del sector, así como identificar las estrategias más adecuadas para abordar los principales tipos de problemas a enfrentar en el análisis de *big data*.

Section 2

0.2 Contenidos y Temario

Tecnologías

- ▶ Servicios y repositorios de **acceso y consumo de datos**.
- ▶ **Lenguajes de programación**.
 - ▶ Python, R.
- ▶ **Entornos de análisis**.
 - ▶ Apache Spark.
 - ▶ Otras herramientas del ecosistema Hadoop (opcional).
- ▶ Bases de datos no relacionales (**NoSQL**).
 - ▶ MongoDB.
 - ▶ Redis (opcional).

Siempre que sea posible, se usarán **herramientas multiplataforma** (para Linux/Mac/Windows).

Temario

- ▶ **Módulo I. Conceptos y métodos para análisis de big data.**
 - ▶ Tema 1. Datos estructurados, semi-estructurados y no estructurados.
 - ▶ Tema 2. Computación paralela y distribuida. MapReduce.
 - ▶ Tema 3. Tecnologías cloud computing y almacenamiento de alto rendimiento.
 - ▶ Tema 4. Métodos y arquitecturas de procesamiento de datos.
- ▶ **Módulo II. Procesamiento paralelo y distribuido de datos.**
 - ▶ Tema 5. Análisis de *big data* en R y Python.
 - ▶ Tema 6. Ecosistema Apache Hadoop.
- ▶ **Módulo III. Bases de datos no relacionales (NoSQL).**
 - ▶ Tema 7. Bases de datos no relacionales (NoSQL).

Temario

- ▶ **Módulo IV. Procesamiento de datos en memoria y otras técnicas avanzadas.**
 - ▶ Tema 8. Procesado de datos en memoria.
 - ▶ Tema 9. Tecnologías avanzadas: TokuDB y Aerospike.
- ▶ **Módulo V. Caso práctico.**
 - ▶ Práctica final. Análisis de un conjunto de dato estructurados utilizando Apache Spark.

Section 3

0.3 Criterios de evaluación

Evaluación

- ▶ Los criterios detallados se pueden consultar en la **Guía Docente** de la asignatura, disponible en la página de la asignatura en Aula Virtual.
- ▶ Los alumnos serán evaluados a través de **diferentes elementos**:
 - ▶ **Foros** de pregunta y respuesta (30% de la nota).
 - ▶ Resolución de un **caso práctico** al final del Tema 5 (30% de la nota).
 - ▶ Entrega de una **práctica final** (40% de la nota).
- ▶ Las actividades evaluables se podrán encontrar en el apartado “Evaluación” dentro de la página de la asignatura.
- ▶ La fecha tope de entrega en convocatoria ordinaria para las **preguntas en los foros P&R** y la **práctica intermedia** (Tema 5) es el **16 de marzo de 2025** (a las 23:59).
- ▶ La fecha tope de entrega en convocatoria ordinaria para la **práctica final** es el **11 de abril de 2025** (a las 23:59).

Referencias I

- [1] K Cukier. *The Economist*, *Data, data everywhere: A special report on managing information*, 2010, February 25. 2010.
- [2] James Manyika et al. *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute, 2011.
- [3] D Vesset et al. “Worldwide Big Data Technology and Services 2013-2017 Forecast (IDC# 244979)”. En: *IDC Mark Anal* 34.1 (2013).

1 - Obtención de datos

Autor: Felipe Ortega

Análisis de Big Data

Máster Univ. Ingeniería de Sistemas de Decisión

Móstoles, Madrid

15 de febrero de 2021



Contenidos

1.1 Obtención de datos

1.2 Representación de datos

1.1 Obtención de datos

Obtención de datos

- ▶ Etapa crucial y con frecuencia infravalorada.
- ▶ Con frecuencia, la obtención y preparación de datos consume cerca del **85% del tiempo total** del proyecto de análisis de datos.
- ▶ Diferentes retos.
 - ▶ Multiplicidad de fuentes.
 - ▶ Métodos de obtención de datos (*scrapping*, *streaming*, APIs...).
 - ▶ Diferentes formatos de representación.
 - ▶ Consolidación de datos obtenidos.



Obtención de datos

► Multiplicidad de fuentes

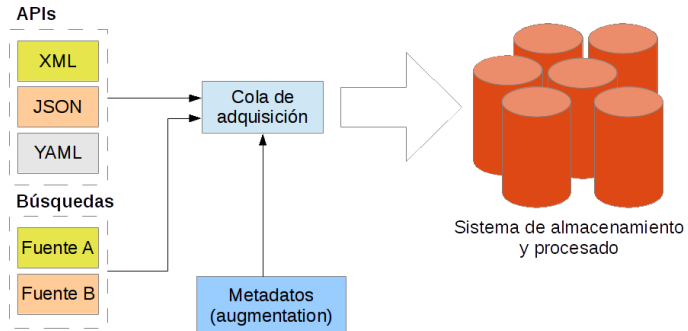


Figura 1: Esquema que representa un flujo típico de ingesta de datos.

Obtención de datos: aspectos de diseño

- ▶ Construir módulos intercambiables para manejar cada tipo de fuente.
- ▶ Misma interfaz de uso, ocultando peculiaridades del manejo de cada fuente o tipo de datos.
- ▶ Considerar diseños basados en colas de elementos (datos o bloques de datos de entrada) que permitan gestionar:
 - ▶ Distintas velocidades de adquisición.
 - ▶ Datos heterogéneos.
 - ▶ Mantenimiento estricto del orden de llegada.

Obtención de datos: aspectos de diseño

- ▶ Nunca debemos asumir que las fuentes nos van a enviar los datos correctamente representados o íntegros.
- ▶ Ejemplo: [Beautiful Soup](#).
 - ▶ Biblioteca Python para adquisición de datos HTML (y XML).
 - ▶ Soporta fallos en sintaxis HTML (o XML) de los documentos de origen.
 - ▶ Proporciona una interfaz unificada para acceso y navegación a los elementos y atributos del documento que parseamos.
 - ▶ Rendimiento depende mucho de la biblioteca de bajo nivel que usemos (e.g. `lxml` proporciona gran rendimiento para XML y HTML al usar código compilado en C, pero es menos robusto frente a fallos de formato).

Obtención de datos: aspectos de diseño

- ▶ Aquí importa (y mucho) la velocidad de ejecución.
 - ▶ En flujos de datos en tiempo real podemos perder datos si no los recuperamos a tiempo.
 - ▶ Los tiempos de espera para tratamiento de fuentes de gran volumen se pueden alargar demasiado (días, semanas).
- ▶ Ejemplos: bibliotecas `lxml`, `UJSON` (ambas para Python).



Obtención de datos: aspectos de diseño

- ▶ Pero también hay que respetar los límites impuestos por determinados sistemas fuente.
 - ▶ En APIs públicas, se suele limitar el número de consultas que pueden realizarse en un cierto intervalo, también la cantidad de datos devueltos por cada consulta o el rango temporal que podemos abarcar.
- ▶ Ejemplos: [Twitter Developers REST API](#), [Facebook](#).
 - ▶ [Tabla de listado de límites de velocidad en la API REST de Twitter](#).



1.2 Representación de datos

Tipos de datos según su estructura

Recordemos:

- ▶ **Datos estructurados:** Tienen una serie de campos con significado predefinido. Cada campo está asociado a un tipo de datos (numérico, textual, doble precisión, objeto serializado...). Ejemplo: bases de datos relacionales (RDBMS), ficheros CSV, *data frames* (en Spark, R, o mediante la biblioteca Pandas de Python científico).
- ▶ **Datos semi-estructurados:** Se representan mediante un formato de codificación que aporta cierta estructura e información sobre los datos (metadatos). Sin embargo, su contenido (número de campos, formato de cada campo, etc.) puede ser muy variado. Ej: documentos XML, JSON, YAML.
- ▶ **Datos no estructurados:** El formato de los datos no está claramente definido de forma previa. Pueden aparecer mezclados datos numéricos, textuales o multimedia, y en un orden imprevisible. Ej: campos de texto libre, comentarios en informes de error.

Tipos de datos según su estructura

- ▶ Los datos **no estructurados** se diferencian claramente del resto por requerir estrategias especiales de análisis.
 - ▶ Por ejemplo, para análisis de datos textuales se suelen aplicar técnicas de Procesamiento de Lenguaje Natural (NLP). Un ejemplo sería la biblioteca **NLTK** (Python).
- ▶ En **datos estructurados**, es obligatorio definir el esquema de almacenamiento de información que vamos a usar.
 - ▶ Generalmente, podemos pensar en ellos como una tabla de una base de datos tradicional.
 - ▶ Siempre que falte un valor en una posición (fila i, columna j), tendremos un nulo (dato faltante).

Tipos de datos según su estructura

- ▶ En **datos semiestructurados** no es necesario definir previamente el esquema de almacenamiento, por lo que este es flexible.
 - ▶ Pueden faltar subelementos o atributos en cualquier elemento y no hay error por ello.
 - ▶ Esos sí, siempre que se defina un esquema formalmente (por ejemplo, mediante una DTD en XML) se tiene que cumplir el esquema y ya no pueden faltar campos o elementos (si no, el parser que sepa acerca de la descripción del esquema devolverá un error).
 - ▶ Los sistemas de almacenamiento y procesamiento de datos semiestructurados suelen permitir realizar búsquedas y navegación directamente sobre el contenido de los archivos, ya que soportan que algunos de los campos puedan no estar presentes en ciertos elementos.

Representación de datos

- ▶ Formatos relacionados con tecnologías web.
 - ▶ HTML, XML, JSON, YAML, etc.
- ▶ Procesamiento de datos.
 - ▶ CSV, HDF5, ff (lenguaje R), otros formatos específicos.
 - ▶ Avro, Parquet (Hadoop, Spark).
- ▶ Metadatos.
 - ▶ RDF (datos enlazados).



Representación de datos

► Ejemplos [3]: JSON

JSON sample [\[edit\]](#)

```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021"
  },
  "phoneNumber": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

Representación de datos

► Ejemplos [3]: YAML

YAML sample [\[edit\]](#)

The above JSON code is also 100% valid **YAML**; however Indents.^[40]

```
---
firstName: John
lastName: Smith
age: 25
address:
  streetAddress: 21 2nd Street
  city: New York
  state: NY
  postalCode: 10021

phoneNumber:
  -
    type: home
    number: 212 555-1234
  -
    type: fax
    number: 646 555-4567
```

Representación de datos

► Ejemplos [3]: XML

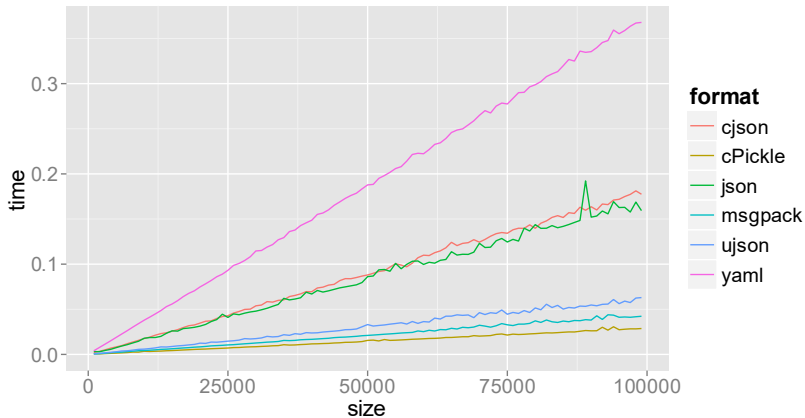
XML samples [\[edit\]](#)

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber type="home">212 555-1234</phoneNumber>
    <phoneNumber type="fax">646 555-4567</phoneNumber>
  </phoneNumbers>
</person>
```

```
<person firstName="John" lastName="Smith" age="25">
  <address streetAddress="21 2nd Street" city="New York" state="NY" postalCode="10021" />
  <phoneNumbers>
    <phoneNumber type="home" number="212 555-1234"/>
    <phoneNumber type="fax" number="646 555-4567"/>
  </phoneNumbers>
</person>
```

Representación de datos

► Benchmark bibliotecas serialización (Python) [4].



Representación de datos: procesamiento

- ▶ Almacenamiento de estructuras de datos de gran tamaño en disco.
- ▶ Estándares
 - ▶ Hierarchical Data Format version 5 (HDF5).
- ▶ Otros formatos específicos (por ejemplo para R).
 - ▶ Paquetes R `ff`, `ffbase` o `bigmemory`.

Representación de datos: HDF5

- ▶ Conjunto de datos jerárquicos, estructurados y autodescriptivos (metadatos).
- ▶ Capaz de escalar con facilidad al nivel de Exabyte (~1000 TB), compresión transparente, ubicación en múltiples dispositivos.
- ▶ Capacidad de indexación y E/S parcial.
 - ▶ Evitamos cargar grandes volúmenes de datos en memoria o búsquedas secuenciales.
- ▶ Bibliotecas disponibles en C, C++, Python, MATLAB, etc.

Representación de datos: HDF5

- ▶ Recomendable cuando los datos sean [5]:
 - ▶ Grandes arrays numéricos.
 - ▶ De tipo homogéneo.
 - ▶ Que se puedan organizar jerárquicamente.
 - ▶ Con metadatos de tipo arbitrario.
- ▶ Para gestión de relaciones entre datos mejor usar bases de datos.
- ▶ Se puede usar también formatos más sencillos (e.g. CSV) para casos simples.

Representación de datos: otros formatos

- ▶ Proyecto `ff` para el lenguaje R.
- ▶ Permite manejar grandes volúmenes de datos en R, sin necesidad de recurrir a clusters o cloud computing.
- ▶ Implementación de estructuras de datos comunes en R (ej. data frames).
- ▶ Implementación en C y C++ a bajo nivel, transparente para el usuario.
- ▶ Soporte para aplicación paralela de operaciones sobre datos en disco.

Representación de datos: RDF

- ▶ Resource Description Framework.
- ▶ Familia de estándares de representación de metadatos promovida por W3C.
- ▶ Tripletas (sujeto-predicado-objeto) definen grafos dirigidos.
- ▶ Ofrecen información sobre ubicación y relaciones entre los datos almacenados (recursos web enlazados).
- ▶ Es posible consultar el grafo mediante el lenguaje `SPARQL`.

Representación de datos

► Ejemplo grafo RDF.



References I



[Kazil, 2016] Kazil, J., Jarmul, K.

Data Wrangling with Python

O'Reilly Media, Inc., febrero, 2016.



[Mitchell, 2015] Mitchell, R.

Web Scraping with Python: Collecting Data from the Modern Web.

O'Reilly Media, Inc., abril, 2015.



[Collette, 2013] Collette, A.

Python and HDF5: Unlocking Scientific Data.

O'Reilly Media, Inc., abril, 2015.



[Grus, 2015] Grus, J.

Data Science from Scratch: First Principles with Python.

O'Reilly Media, Inc., abril, 2015.

Computación Paralela: MapReduce

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

22 de febrero de 2021



Contenidos

2.1 Computación Paralela

2.2 El Paradigma MapReduce

2.1 Computación Paralela

Comp. paralela: objetivos

- ▶ El **objetivo principal** de la computación paralela y distribuida es **incrementar las prestaciones** de una arquitectura computacional, aumentando la **velocidad de ejecución** de los programas.
 - ▶ Para mejorar un sistema monoprocesador, es más rentable añadir otra CPU que desarrollar una CPU el doble de rápida.
 - ▶ Ciertos problemas extremadamente complejos no pueden resolverse utilizando una sola CPU (en general, problemas que involucran *big data*).
- ▶ Por tanto, se desarrollan arquitecturas de computación no convencionales para poder alcanzar la potencia de cómputo necesaria para abordar estos problemas.
- ▶ Existen varias alternativas para construir sistemas de procesamiento paralelo/distribuido.

Cómputo de datos

► Multiprocesador.

- Una sola máquina con mucha memoria y múltiples CPUs (cores).

► Clúster.

- Conjunto de máquinas independientes funcionando de forma solidaria.
- Cada máquina puede tener varias CPUs (cores).
- Coordinación por mensajería (más frecuente) u otros medios.

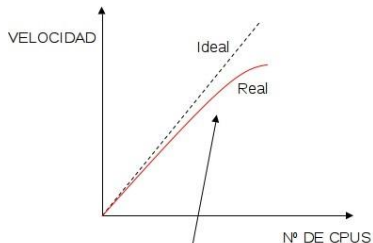
► Sistemas distribuidos.

- Además de contar con varios nodos de cómputo, cada uno de los cuales puede contener varios procesadores, los nodos pueden estar en diferentes ubicaciones.

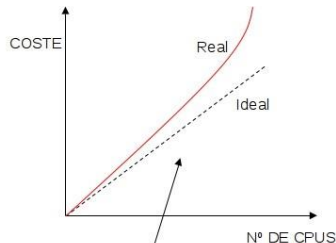


Coste paralelización

- ▶ No obstante, el incremento de prestaciones y coste no es lineal respecto al número de procesadores (y/o nodos) presentes en el sistema.



A partir de un cierto número de CPUs es más difícil sincronizar sus tareas



El coste real aumenta más rápidamente ya que cuantas más CPUs haya más costoso y complicado es el sistema de comunicación y control.

Programación paralela

- ▶ Permite la **división explícita** de tareas, para repartirlas a los nodos de computación.
- ▶ En general, el programador es capaz de asignar una tarea concreta a un nodo o un core específico.
- ▶ Además, se suelen utilizar **hilos de ejecución** para repartir las tareas en varios flujos de ejecución, dentro de un mismo proceso.
- ▶ Se deben emplear lenguajes de programación específicos que permiten repartir las tareas en distintas CPUs (o nodos). Algunos ejemplos: Occam, PVM, OpenMP, etc.
- ▶ **Desventajas:**
 - ▶ Es necesario aprender un nuevo lenguaje de programación.
 - ▶ Comprender los esquemas de paralelización explícita de tareas no es en absoluto trivial, y requiere un importante esfuerzo por parte del programador.

Paralelización: acceso a memoria

- ▶ Tenemos un problema añadido a la hora de paralelizar tareas en sistemas clúster y sistemas de computación distribuidos: el **acceso a datos en memoria**.
- ▶ En un multiprocesador, aunque tengamos varias unidades de cómputo (cores) todas ellas pueden acceder a la totalidad de memoria del sistema. Los programas, por tanto, se pueden hacer considerando que cualquier tarea en cualquier CPU puede acceder a un dato generado por otra tarea que se almacene en RAM (y con los permisos de acceso adecuados).
- ▶ Por el contrario, en un clúster o un sistema de computación distribuido cada nodo solo ve su propia memoria, y no puede acceder directamente a un dato que esté en la memoria de otros nodos.

Paralelización: acceso a memoria

- Arquitecturas que permiten un acceso uniforme a cualquier dato de memoria (como en un multiprocesador) se denominan **UMA (Uniform Memory Access)**

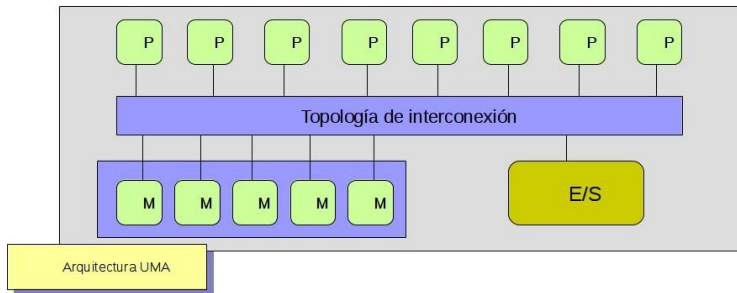


Figura 1: Esquema de una arquitectura UMA.

Paralelización: acceso a memoria

- Arquitecturas que no permiten un acceso uniforme a cualquier dato de memoria, sino que implican latencia (retardo) para acceder a datos de otro nodo (clúster, sist. distribuido), se denominan **NUMA** (*Non-Uniform Memory Access*)

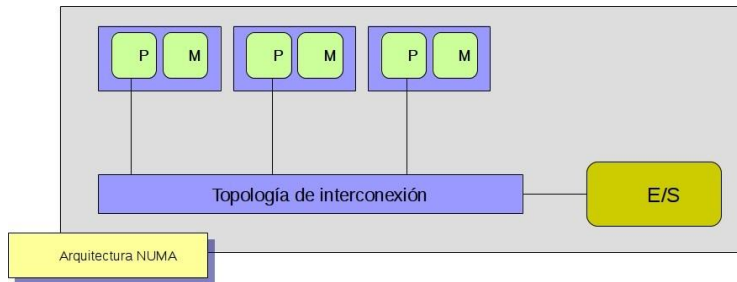


Figura 2: Esquema de una arquitectura NUMA.

Paralelización: acceso a memoria



Cluster de computadores Linux en la Universidad de Purdue (West Lafayette, Indiana, EE.UU).

Cada nodo es un PC de gama estándar para consumidores finales.

La ventaja de los clusters es que son fácilmente escalables y el coste de cada nodo no suele ser elevado.

En ocasiones, cada nodo se presenta en carcasas diseñadas para ocupar poco espacio (*blades*) y se montan en un armario *rack*.

Paralelización: conclusiones

- ▶ En este escenario, emergen los requisitos que debería tener un sistema de paralelización de tareas dirigido a programadores generalistas.
- ▶ Dicho sistema, debería requerir ciertos conocimientos sobre la infraestructura de computación pero no un conocimiento tan profundo de las complejidades inherentes a este tipo de sistemas (paralelización explícita, acceso a memoria, sincronización de tareas, etc.).
- ▶ Como respuesta a estas premisas iniciales surge MapReduce.

2.2 El Paradigma MapReduce

MapReduce: origen

- ▶ En este escenario, emergen los requisitos que debería tener un sistema de paralelización de tareas dirigido a programadores generalistas.
- ▶ Dicho sistema, debería requerir ciertos conocimientos sobre la infraestructura de computación pero no un conocimiento tan profundo de las complejidades inherentes a este tipo de sistemas (paralelización explícita, acceso a memoria, sincronización de tareas, etc.).
- ▶ Como respuesta a estas premisas iniciales surge MapReduce.

Definiendo MapReduce

- ▶ MapReduce es un **modelo de programación y procedimiento de implementación** que permite **generar y procesar grandes volúmenes de datos** en sistemas de computación paralelos y distribuidos.
- ▶ Su origen está en un artículo publicado por Google en diciembre de 2004 ¹, en el que se describen pormenorizadamente todos los detalles del modelo.
- ▶ A su vez, este paradigma de programación necesita un sistema de ficheros distribuido que permita su implementación en plataformas de computación paralelas. El **Google File System** es uno de estos sistemas, que había sido presentado un años antes, en octubre de 2003 ².

¹ <http://research.google.com/archive/mapreduce.html>

² <http://research.google.com/archive/gfs.html>

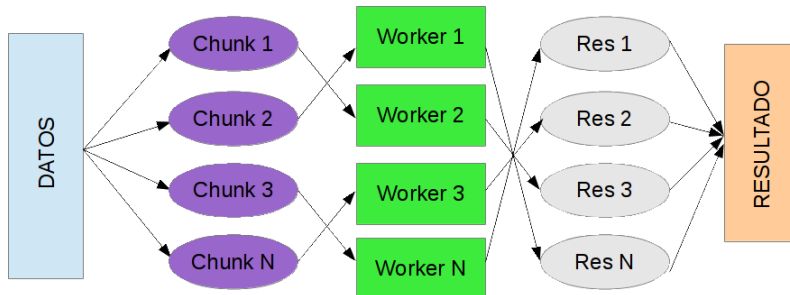
Definiendo MapReduce

- ▶ Sin embargo, ambos sistemas eran tecnología propietaria de Google, por lo que surge la necesidad de una implementación equivalente como proyecto *open source*.
- ▶ La respuesta a esta necesidad vino de un proyecto conjunto de dos investigadores, Doug Cutting y Mike Cafarella³, que trabajaban en un sistema para implementar una nueva generación de motor de búsqueda en la Web.
- ▶ Cutting acabó trabajando en Yahoo!, donde el proyecto creció definitivamente.
 - ▶ El primer objetivo era un sistema para migrar el motor de búsqueda de la compañía, escalando sus operaciones de forma eficiente.
 - ▶ Sin embargo, el proyecto pronto se reveló capaz de aprovechar la ingente infraestructura de computación distribuida en Yahoo! para acelerar el **análisis de conjuntos masivos de datos**.

³ <https://gigaom.com/2013/03/04/the-history-of-hadoop-from-4-nodes-to-the-future-of-data/>

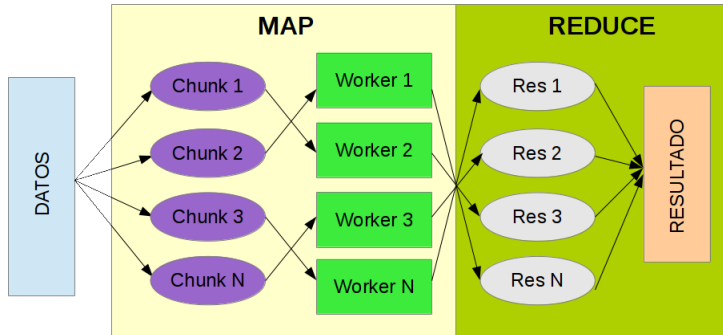
Esquema MapReduce

- El siguiente diagrama muestra una aproximación genérica para la paralelización de algoritmos.



Esquema MapReduce

- El paradigma MapReduce sigue ese esquema de paralelización.



Esquema MapReduce

- ▶ El esquema se compone de dos grandes fases:
 - ▶ **Fase Map:** Se aplica (“mapea”) una función que ejecuta una operación o un conjunto de operaciones sobre cada elemento de los datos de entrada (por ejemplo, construir duplas `(word, 1)` para cada palabra en un documento).
 - ▶ **Fase Reduce:** Se aplica una función resumen a los resultados de la fase anterior (por ejemplo, sumar todos los pesos para duplas de la misma palabra, con lo que obtenemos la frecuencia de aparición de cada palabra).
- ▶ Este paradigma se basa en un modelo previo creado en programación funcional. De hecho, podemos encontrarlo en numerosos lenguajes de programación.
 - ▶ Ejemplo de [implementación MapReduce con multiprocessing](#) en Python 2.7.

Esquema MapReduce

- ▶ En realidad, la verdadera aportación de MapReduce no es su modelo de programación, que ya existía.
- ▶ La innovación de los sistemas open source que surgieron a raíz de MapReduce (como Hadoop) es proporcionar los mecanismos de computación distribuida y tolerancia a fallos necesarios para poder escalar estas aplicaciones a sistemas compuestos por cientos o miles de nodos.
- ▶ Se necesitan por tanto muchos componentes adicionales:
 - ▶ Sistema de ficheros distribuido (e.g. HDFS).
 - ▶ Módulo de gestión de recursos del sistema distribuido (e.g. YARN).
 - ▶ Sistema de monitorización de trabajos en ejecución (e.g. Zookeeper).
 - ▶ Etc.

Esquema MapReduce

- ▶ Para que este modelo de trabajo funcione se deben cumplir ciertas condiciones.
 - ▶ El problema a resolver se debe poder **descomponer en tareas** que pueden **aplicarse sobre los datos de forma casi independiente** las unas de las otras, con muy poca comunicación entre ellas.
 - ▶ En computación distribuida se les conoce como *embarrassingly parallel problems*.
 - ▶ Surgen complicaciones en flujos de trabajo con tareas que precisan resultados intermedios de otras tareas para proseguir su ejecución (por ejemplo, **algoritmos iterativos**, muy frecuentes en data mining/machine learning).
 - ▶ Las operaciones que aplicamos sobre los datos deben ser **asociativas**, lo que garantiza que partes de ellas se puedan distribuir en los nodos del sistema paralelo de computación, para ejecutarse de forma independiente.

MapReduce: conclusiones

- ▶ Es el esquema central que ha permitido popularizar el análisis de conjuntos masivos de datos en muchas empresas.
- ▶ La mayoría de distribuciones software para análisis de big data (Cloudera, Hortonworks, MapR, etc.) nacieron bajo el paraguas del paradigma MapReduce.
- ▶ Sin embargo, en la actualidad MapReduce se ha revelado como un paradigma poco flexible para implementar muchos problemas en análisis de *big data*, principalmente los que involucran cálculos iterativos entre fases de procesamiento de datos.
- ▶ Por ello, aunque sigue siendo un estándar en la industria va progresivamente cayendo en desuso, dando paso a otras soluciones más potentes y avanzadas, como Spark (que veremos más adelante).
 - ▶ Google anuncia en junio 2014 que ya no usa MapReduce.
 - ▶ Cloudera unifica Hadoop y Spark como núcleo de su distribución.

Computación paralela: tendencias

- ▶ Muchas empresas se ven obligadas a invertir en costosos sistemas hardware para implementar el análisis de *big data* dentro de la compañía, frecuentemente por razones legales y de protección de datos.
- ▶ Sin embargo, una tendencia fuerte en la actualidad es aprovechar las capacidades de **computación bajo demanda** que ofrecen plataformas y servicios de **cloud computing**, evitando costes de mantenimiento, despliegue y gestión.
- ▶ Los nodos de computación tienen cada vez más memoria RAM, lo que habilita el **procesamiento de datos en memoria**, evitando el uso del disco convencional (magnético) más que para **almacenamiento persistente** de datos intermedios o resultados finales.
 - ▶ [Amazon anuncia la X1](#), primera instancia de su servicio AWS EC2 con hasta 2TB de RAM.

Computación paralela: tendencias

- ▶ Cada vez más aplicaciones necesitan **analizar datos en tiempo real**, o al menos con un ritmo de obtención de resultados mucho más ágil que el proporcionado por sistemas MapReduce clásicos, como Hadoop.
 - ▶ Aparecen nuevos modelos de procesamiento de datos que combinan procesado *en batch* y procesado *streaming* dentro del mismo sistema.
 - ▶ Aparecen sistemas software específicos para procesado de datos streaming y en memoria: Spark, Storm, Flink, etc.
- ▶ Por último, las nuevas tecnologías de almacenamiento persistente de datos, como los **discos de estado sólido (SSD)**, permiten mucha mayor velocidad de E/S que los discos convencionales, acelerando gran cantidad de aplicaciones de análisis de big data.

Referencias I



[White, 2015] White, T.
Hadoop: The Definitive Guide.
O'Reilly Media, Inc., abril, 2015.



[Marz, 2015] Marz, N.
Big Data: Principles and best practices of scalable realtime data systems
Manning Pub., mayo, 2015.



[Holmes, 2014] Marz, N.
Hadoop in Practice
Manning Pub., octubre, 2014.



[Frampton, 2015] Frampton, M.
Big Data Made Easy: A Working Guide to the Complete Hadoop Toolset.
Apress, diciembre, 2014.

Referencias II



[Eadline, 2015] Eadline, D.

Hadoop 2 Quick-Start Guide.

Addison-Wesley Data & Analytics, noviembre, 2015.

Tecnologías Cloud Computing

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

22 de febrero de 2021



Contenidos

3.1 Computación en la nube

3.2 Almacenamiento de alto rendimiento

3.1 Computación en la nube

Definición cloud computing

- ▶ El NIST define *cloud computing* como un modelo que posibilita acceso **ubicuo, cómodo y bajo demanda**, a través de una red de comunicación, a un conjunto de recursos de computación configurables.
 - ▶ Recursos tales como redes, servidores, almacenamiento, aplicaciones y servicios.
- ▶ Los recursos del cloud pueden **provisionarse rápidamente** y ser **liberados** con **mínimo esfuerzo de gestión** o intervención del proveedor de servicios.

Definición cloud computing: conceptos clave

- ▶ Computación **bajo demanda**: Ya no es preciso una fuerte inversión inicial en costosos equipos físicos que luego hay que mantener y actualizar. Se paga por el acceso a recursos solo durante el tiempo que se utilizan.
- ▶ Computación **elástica**: Si existen picos de demanda de recursos de computación no es un problema. El sistema permite solicitar más recursos **dinámicamente**, y agregarlos a la infraestructura, así como **reasignar** recursos entre diferentes tareas y servicios.
- ▶ **Virtualización** de recursos: La tendencia general gira en torno a **encapsular el acceso** a cualquier recurso de computación físico (servidores, sistemas operativos, software, sistemas de conexión de redes, etc.) a través de objetos virtuales que ofrecen una API para interactuar con ellos.
 - ▶ Facilita el acceso del usuario a los recursos, ofrece una capa uniforme de interacción, ocultando los detalles.
 - ▶ Facilita la gestión de recursos, al uniformizar los elementos de distinto tipo bajo una misma abstracción.

Abstracción cloud computing

- La abstracción permite ocultar los elementos reales del proveedor de servicios a los usuarios, como si se tratase de una nube.

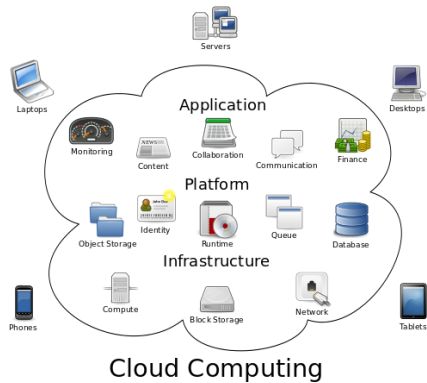


Figura 1: Algunos servicios proporcionados por arquitecturas en la nube.

Tipos de sistemas cloud

- Según su gestión y en qué clase de recursos se basa, distinguimos tres tipos de sistemas cloud.

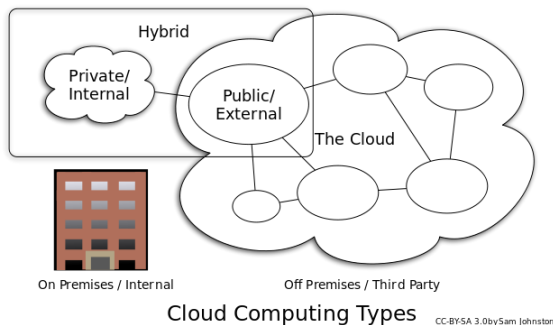


Figura 2: Esquema que muestra diferentes tipos de sistemas cloud, según su gestión e infraestructura.

Tipos de sistemas cloud

- ▶ Cloud **privada**: El acceso a los recursos en la nube es exclusivo del propietario de los servicios y se realiza mediante una conexión privada.
 - ▶ La infraestructura física de la nube suele estar operada de forma privada por una sola organización, independientemente de que la gestión sea interna o a través de un proveedor externo.
 - ▶ En cuanto a su ubicación, puede ser estar tanto en instalaciones de la propia compañía como en un proveedor externo, pero siempre que proporcione los recursos en exclusiva.
 - ▶ Algunos proveedores de servicios de cloud pública como Amazon AWS o Microsoft Azure pueden ofrecer también acceso exclusivo y seguro a los servicios, con un coste adicional.

Tipos de sistemas cloud

- ▶ Cloud **pública**: El acceso a los recursos en la nube no es exclusivo, sino que se ofrece un servicio público para cualquier usuario al que se puede acceder a través de Internet.
 - ▶ Tanto la infraestructura física como la ubicación pueden ser, como en el caso anterior, internas u ofrecidas mediante un proveedor, pero en esta modalidad el acceso a los recursos no es exclusivo, lo que implica importantes diferencias de seguridad.
- ▶ Cloud **híbrida**: En una solución compuesta por recursos de las dos modalidades anteriores, en la que el cliente del servicio puede elegir dónde despliega sus servicios o aplicaciones en función de diversos requisitos como coste o medidas de seguridad necesarias.

Modelos de servicio en cloud

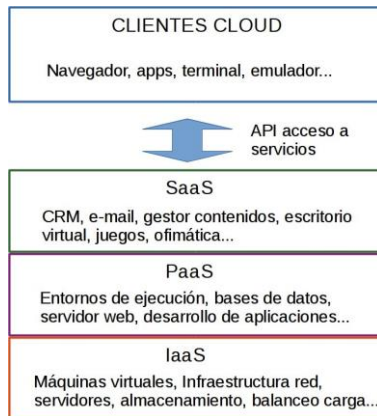


Figura 3: Diferentes niveles de servicio en arquitecturas en la nube.

Modelos de servicio cloud: IaaS

- ▶ Nivel de servicio más básico que se ofrece sobre cloud.
- ▶ El cliente tiene acceso directo a recursos que son **abstracciones de la infraestructura** real que aprovisiona el cloud.
- ▶ Típicamente incluye servicios tales como: virtualización, funciones y servicios de red virtuales (SDN y NFV), almacenamiento en crudo, de archivos y objetos serializados, filtrado de tráfico, balanceo de carga, etc.
- ▶ Elimina labores de mantenimiento y actualización de infraestructura física (consumo energético, actualización hardware, etc.).
- ▶ Sigue precisando conocimientos sobre administración y gestión de los recursos utilizados.

Modelos de servicio cloud: PaaS

- ▶ Típicamente ofrecidos para desarrollo de aplicaciones.
- ▶ El servicio oferta un entorno de trabajo preparado para desarrollo software en una plataforma específica: lenguajes y sistemas de computación (.NET, Java, Python, Ruby, servicios Google, etc.).
- ▶ El desarrollador puede centrarse en la creación de aplicaciones, liberándose de las tareas de gestión y mantenimiento de la plataforma computacional subyacente.
- ▶ Un caso importante es el de **data Platform as a Service (dPaaS)**, donde el proveedor ofrece una plataforma vertical integrada para análisis de datos e integración con servicios basados en datos.
 - ▶ Ej: IBM Bluemix con Spark.

Modelos de servicio cloud: SaaS

- ▶ El usuario obtiene acceso a aplicaciones de alto nivel, dirigidas a usuario/cliente final.
- ▶ Suele requerir el pago por tiempo de uso o por suscripción.
 - ▶ Ejemplo: [Microsoft Office 365](#).
- ▶ El usuario ya solo se ocupa de interactuar con el software, todos los demás detalles sobre plataforma e infraestructura son transparentes.
- ▶ El proveedor de servicios tiene control total sobre la versión de software utilizado, acceso a recursos, etc.

Ejemplo: Databricks



Figura 4: Arquitectura del servicio Databricks.

Ejemplo: databricks

- ▶ Producto proporcionado por la empresa del mismo nombre, que ofrece un servicio de dPaaS.
- ▶ El cliente tiene acceso a varios componentes:
 - ▶ **Notebooks** compatibles con Jupyter, para implementar procesos de análisis de datos y acelerar su replicación y mejora, compartiéndolos con otros usuarios.
 - ▶ **Gestión de plataforma** para controlar la creación asistida de clústers, accediendo a la cuenta del cliente para infraestructura con un proveedor IaaS (actualmente, solo compatible con Amazon AWS).
 - ▶ Gestor de **trabajos**, que permite monitorizar la ejecución de tareas de obtención, procesamiento y almacenamiento de datos en el clúster.
 - ▶ **Dashboards** para la creación de informes visuales a partir de resultados de análisis de datos generados mediante Spark.
 - ▶ Conectores con **aplicaciones de otros proveedores**: Pentaho, Qlik, TIBCO Spotfire, Tableau, Pantera o Zoomdata.

Ejemplo: Databricks

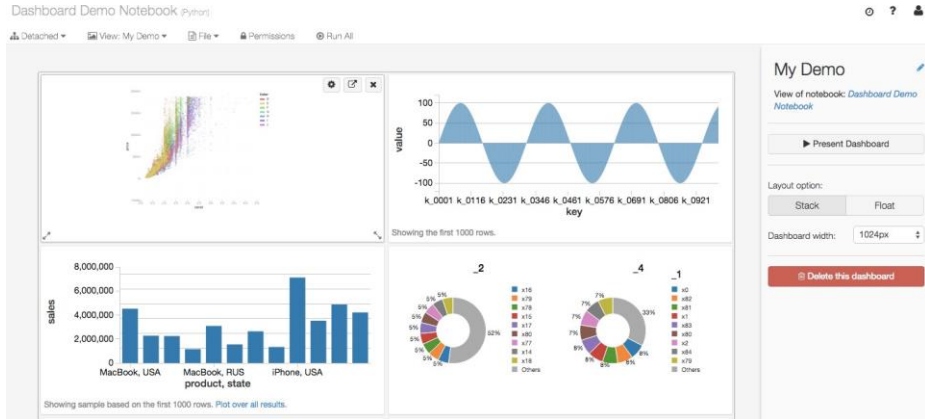


Figura 5: Ejemplo de dashboard en Databricks.

Virtualización

- ▶ Antigua idea (*mainframes*) ahora renovada.
- ▶ Ejecución de múltiples entornos operativos completos en una sola máquina.
- ▶ Ahorro de costes de despliegue de infraestructuras servidor.
- ▶ Múltiples ventajas para la administración de sistemas.
- ▶ Mejor aprovechamiento de consolidación de almacenamiento.

Virtualización de máquina completa

- ▶ Existen varias modalidades. Las siguientes son algunas de las más comunes.
- ▶ **Emulación:** virtualizamos hardware y software, puedo correr un S.O. sin modificar, creado para una plataforma hardware diferente (distinta CPU).
- ▶ **Nativa y completa:** Simulamos un sistema hardware completo (con la misma CPU), de manera que el huésped puede ejecutarse sin ser modificado. Ej: VMWare, VirtualBox, KVM.
- ▶ **Paravirtualización:** El huésped no accede a hardware, sino a una API especial. El anfitrión (hypervisor) controla el acceso a los recursos de la máquina hardware subyacente.
 - ▶ Ejemplo: Sistema de virtualización Xen.

Virtualización del sistema operativo

- ▶ El huésped y el anfitrión comparten el mismo S.O., pero las aplicaciones del huésped ven un sistema autónomo.
- ▶ Ejemplos: Docker, Virtuozzo, Solaris Containers , VMWare ThinApp.
- ▶ Proporciona grandes ventajas para despliegue rápido de entornos PaaS, donde se encapsula un entorno de desarrollo que podemos desplegar, replicar, salvaguardar el entorno, o también pararlo para volver a desplegarlo en otra localización, todo en pocos pasos.
- ▶ Docker se está convirtiendo rápidamente en una de las soluciones más populares, hasta el punto de ofrecerse ya como soporte estándar en muchos proveedores de cloud (e.g. Amazon AWS).

3.2 Almacenamiento de alto rendimiento

Almacenamiento local redundante

- ▶ **Redundant Array of Inexpensive Disks.**
- ▶ Agrupamos varios dispositivos de almacenamiento (discos magnéticos o SSDs) para que trabajen conjuntamente de forma sincronizada.
- ▶ Se definen 6 configuraciones básicas, y también modos combinados (anidados).
 - ▶ El RAID se muestra al sistema como una sola unidad (no vemos los discos individuales).
 - ▶ Algunos niveles incluyen control de errores y duplicación de datos para mejorar disponibilidad.
 - ▶ Objetivos: Mayor capacidad de almacenamiento, mayor ancho de banda, tolerancia a fallos, posibilidad de sustituir unidades con fallos.

Configuraciones RAID

- ▶ **RAID 0.** Entrelazado (striping), sin redundancia.
- ▶ **RAID 1.** Espejo (mirroring), duplicación de datos en discos de reserva.
- ▶ RAID 2. Entrelazado más códigos Hamming de redundancia. *No se usa en la práctica.*
- ▶ RAID 3 y RAID 4. Entrelazado más redundancia (bit o bloque) en un solo disco. *No se usan en la práctica.*
- ▶ **RAID 5.** Entrelazado con sector de paridad, redundancia distribuida entre todos los discos.
- ▶ **RAID 6.** Como el anterior, pero con doble información de redundancia.
- ▶ Listado de niveles RAID (Wikipedia).

Resumen niveles RAID

Nivel	Capacidad útil	Min. núm. disp.	Tolerancia a fallos	Descripción	BW total
RAID 0	N discos	2	Ninguna	Striping, sin redundancia	$\sum BW_i$
RAID 1	1 sólo disco (resto para replicación)	2	N-1 discos	Mirroring (duplicación de discos)	BW_i
RAID 2	$N - \log_2(N-1)$	7	1 disco, supuesta info paridad legible	Paridad de bit con códigos Hamming	$\frac{4}{7} \sum BW_i$
RAID 3	N-1 discos	3	1 disco (no el de paridad)	XOR nivel de bit en un solo disco	$\sum_{i=1}^{n-1} BW_i$
RAID 4	N-1 discos	3	1 disco (no el de paridad)	XOR nivel de bloque en 1 disco	$\sum_{i=1}^{n-1} BW_i$
RAID 5	N-1 discos	3	1 disco cualquiera	XOR bloque distribuido	$\sum_{i=1}^{n-1} BW_i$
RAID 6	N-2 discos	4	2 discos cualesquiera	Doble paridad de bloque distrib.	$\sum_{i=1}^{n-2} BW_i$

Figura 6: Tabla resumen con las características de los principales niveles RAID.

Niveles RAID anidados

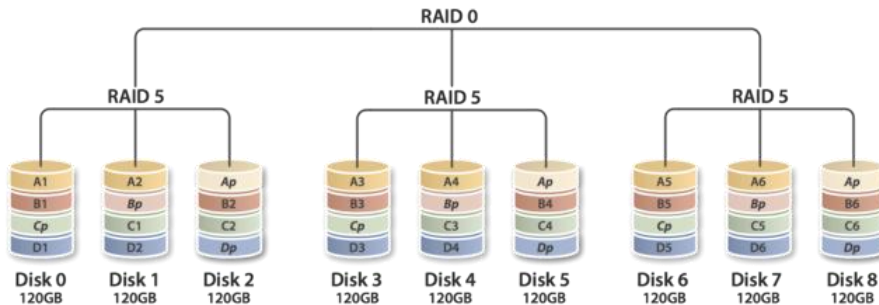


Figura 7: Diagrama de un RAID anidado 50. Existen otras opciones, consulta el [listado de niveles RAID anidados](#).

Tarjeta controladora RAID

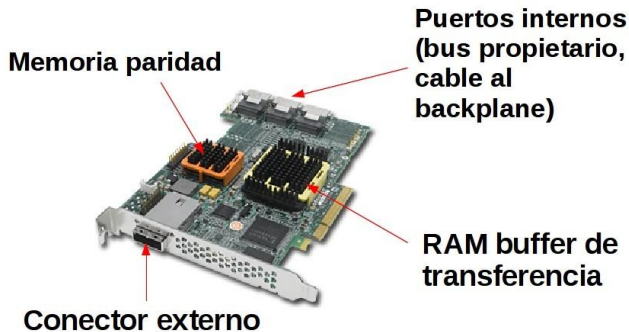


Figura 8: Detalle de componentes de una tarjeta controladora RAID real.

SAN/NAS

- ▶ **Storage Area Network (SAN):** Infraestructura que permite el acceso a dispositivos de almacenamiento conectados en red, de forma que se pueden usar como si estuviesen conectados localmente.
- ▶ **Network Attached Storage (NAS):** En este caso, se hace patente que el dispositivo de almacenamiento es remoto, y se solicitan archivos en lugar de bloques de datos.

Arquitectura SAN/NAS

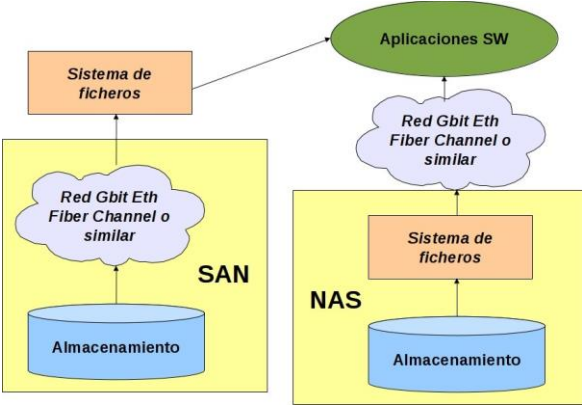


Figura 9: Esquema comparativo de la tecnología SAN vs. NAS.

Cluster local

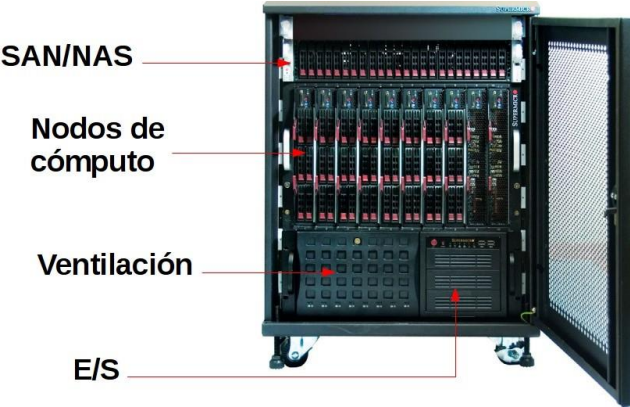


Figura 10: Detalle de componentes de un sistema cluster compacto.

Almacenamiento de datos en cloud

- ▶ Sistemas de almacenamiento nativo en la nube para acceso más eficiente por parte de los recursos de la infraestructura.
- ▶ Ejemplo más importante: **Amazon Simple Storage Service (S3)**.
- ▶ Proporciona sistema de almacenamiento de datos bajo demanda, pensado para acceso frecuente a los mismos.
 - ▶ Para almacenamiento de datos a los que rara vez se accede (por ejemplo, archivos de respaldo de sistemas, etc.) es mejor usar Amazon Glacier.
- ▶ Varias APIs para acceso al servicio: REST, SOAP o BitTorrent.
- ▶ [Página y vídeo introductorio a Amazon S3](#).

Amazon S3

- ▶ El sistema organiza los datos en cubos de información (*buckets*).
- ▶ No existen directorios, así que la organización de datos no es intuitiva al principio para muchos usuarios.
- ▶ Podemos emular un comportamiento parecido a un sistema de ficheros mediante etiquetas que contengan niveles de identificación anidados.
 - ▶ Usamos el mismo prefijo para todos los objetos/datos/archivos relacionados.
 - ▶ Accedemos luego a todos ellos preguntando por los datos cuyo identificador coincide con dicho prefijo (vemos un ejemplo práctico con código más adelante).
- ▶ El sistema está diseñado para proporcionar una **alta disponibilidad (99.99%)**, si bien el contrato de nivel de servicio es flexible en cuanto a la durabilidad de la infraestructura.

Amazon S3: boto

- ▶ **Boto** es el SDK para servicios web con Python que permite acceder a la API de Amazon S3 desde este lenguaje.
- ▶ Permite el acceso a recursos de almacenamiento de datos en Amazon S3 desde cualquier framework de desarrollo en Python:
 - ▶ Aplicaciones Standalone.
 - ▶ Aplicaciones web con Django o frameworks similares.
 - ▶ Aplicaciones Apache Spark que se ejecuten en modo local, en un clúster local o en un clúster de Amazon AWS.
- ▶ **Instalación estándar:** `pip install boto3`.

Amazon S3: boto

► Ejemplo práctico (Python 3).

```
from pyspark import SparkConf
from pyspark import SparkContext
from boto3.session import Session

HDFS_MASTER = 'HEAD_NODE_IP'
AWS_KEY = '****' # Introducir clave de AWS
AWS_SECRET = '****' # Introducir password para la clave AWS

session = Session(aws_access_key_id=AWS_KEY,
                  aws_secret_access_key=AWS_SECRET)
```

Amazon S3: boto

► Ejemplo práctico (Python 3, continuación).

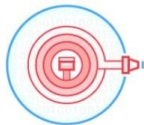
```
# Lectura de todos los directorios almacenados en el
# bucket publico de S3 'blaze-data' (proyecto Blaze)
good_files = []
client = session.client('s3')
emails_meta = client.list_objects(Bucket='blaze-data',
                                   Prefix="enron-email/",
                                   Delimiter="/")
emails_dirs = emails_meta.get('CommonPrefixes')
```

Otros servicios de datos en Amazon AWS

- ▶ **Relational Database Service (RDS):** Servicio de base de datos relacional, accesible mediante una API. Compatible con las principales opciones de motores relacionales: MySQL, Oracle Database, Microsoft SQL Server, PostgreSQL, MariaDB.
 - ▶ En 2014 Amazon lanzó **Aurora**, un servicio enfocado en ofrecer bases de datos MySQL con alta disponibilidad y rendimiento mejorado.
- ▶ **RedShift:** proporciona acceso a un servicio de datawarehouse masivamente paralelo, compatible con el resto de la infraestructura y servicios de AWS. Proporciona una funcionalidad similar a la de las grandes bases de datos columnares para analítica de datos (que veremos en la parte de NoSQL).
- ▶ **DynamoDB:** Otra alternativa de servicio de base de datos no convencional (NoSQL). Ofrece integración con la plataforma Hadoop de Amazon, Elastic MapReduce (EMR).

Sistemas integrados procesamiento de datos

- **Kinesis:** Un servicio integral de procesamiento de flujos de datos en tiempo real (similar a Storm, Flink o Spark Streaming).



Amazon Kinesis Firehose

Cargue fácilmente enormes volúmenes de datos de streaming en AWS

[Más información »](#)



Amazon Kinesis Analytics

Analice fácilmente datos de streaming mediante SQL estándar

[Más información »](#)



Amazon Kinesis Streams

Cree sus propias aplicaciones personalizadas para procesar o analizar datos de streaming

[Más información »](#)

Página con la descripción de Amazon Kinesis.

Referencias I



[Rafaels, 2015] Rafaels, Ray J.

Cloud Computing: From Beginning to End.

CreateSpace Indep. Pub. Platform, abril, 2015.

Arquitecturas de procesamiento de datos

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

22 de febrero de 2021



Contenidos

4.1 Jerarquía de memoria

4.2 Estrategias procesamiento de datos en memoria

4.3 Software para procesamiento de datos en memoria

4.1 Jerarquía de memoria

Jerarquía de memoria en computación

- ▶ Se trata de un modelo de arquitectura computacional que permite conseguir un sistema con gran capacidad para almacenamiento de datos a un coste razonable.
- ▶ Lo ideal sería tener grandes cantidades de memoria RAM para guardar toda la información de nuestras aplicaciones software, accediendo a esos datos de forma directa y rápida. Sin embargo, tradicionalmente existen dos grandes inconvenientes:
 - ▶ La memoria RAM es **volátil**, por lo que cuando el sistema se queda sin energía los datos que almacena se pierden.
 - ▶ El coste de la memoria RAM es mucho más elevado que el de otros sistemas de almacenamiento mucho más lentos pero de gran capacidad, como las unidades de discos magnéticos (HDDs).

Jerarquía de memoria en computación

- ▶ El modelo de **jerarquía de memoria** propone una solución de compromiso para salvar estas limitaciones.
- ▶ **Objetivo:** combinar las tecnologías de almacenamiento para obtener el mejor rendimiento al menor coste posible.
 - ▶ Ponemos **cerca del procesador** la **tecnología más rápida** pero volátil y de mayor coste.
 - ▶ Ubicamos **lejos del procesador** la **tecnología más lenta**, no volátil y de menor coste.
- ▶ Como resultado, para un coste fijo del sistema la combinación nos deja **menos capacidad** de almacenamiento en la **memoria de sistema** (más cerca de la CPU) y mucha **mayor capacidad** de **almacenamiento persistente**, más lejos de la CPU (sistema de E/S).

Modelo de jerarquía de memoria



Figura 1: Modelo de jerarquía de memoria.

Principio de localidad

- ▶ El modelo de jerarquía de memoria funciona bien en la práctica gracias al llamado **principio de localidad**.
- ▶ Estudios empíricos a comienzos de los años 80 constataron que el procesador, al ejecutar un programa software, accede a los datos en memoria siguiendo un patrón muy concreto. Este patrón se denomina **localidad** y se manifiesta en tres dimensiones:
 - ▶ **Localidad espacial:** En cada instante, la CPU tiende a usar los datos almacenados en memoria que están en posiciones cercanas entre sí (e.g. arrays, bucles de ejecución).
 - ▶ **Localidad temporal:** En cada instante, la CPU tiende a volver a usar los datos que acaba de emplear en el pasado cercano, y seguirá empleándolos en el futuro próximo (**ventana temporal de localidad**).
 - ▶ **Localidad secuencial:** La CPU tiende a acceder a las instrucciones de forma secuencial o repetitiva, por ejemplo en la ejecución de bucles.

Principio de localidad

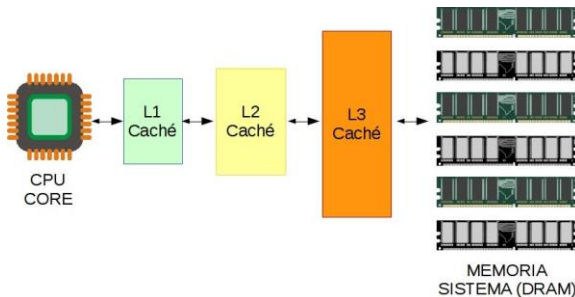
- ▶ Consecuencia para jerarquía de memoria: mantenemos en los niveles más cercanos a la CPU (menor tiempo de acceso) un subconjunto de los datos, con los que trabajará más frecuentemente al menos durante un tiempo (localidad *temporal*).
 - ▶ Cuando el dato no se encuentra en un nivel superior de la jerarquía, entonces bajamos al nivel inferior a buscarlo y subir una copia al nivel superior. De es forma, se acelerarán los siguientes accesos a ese dato.
 - ▶ En lugar de subir un dato individual, subimos un **conjunto de datos adyacentes**, esperando que eso ayude a reducir el número de fallos en futuras peticiones (localidad *espacial*).
- ▶ Esta estrategia permite que el rendimiento de la jerarquía de memoria para acceso a los datos sea bueno a un coste razonable y (globalmente) con una gran capacidad de almacenamiento de información.

Memoria caché (CPU)

- ▶ Si obviamos los registros del propio procesador (que almacenan datos e instrucciones), el primer nivel de la jerarquía de memoria es la **memoria caché**.
- ▶ Hoy en día es parte del procesador que adquirimos a los principales fabricantes (Intel, AMD). Dentro del encapsulado vienen núcleos de ejecución (*cores*) y una memoria caché para almacenar datos a los que accederá frecuentemente durante cierto tiempo.
- ▶ Cuando el programa avance y se empiece a acceder a otros datos nuevos se generarán *fallos en caché*. Entonces, se traen nuevos datos de niveles inferiores (memoria, disco) para almacenarlos en caché, sobrescribiendo los anteriores. Así se vuelve a acelerar el acceso a los nuevos datos durante otro intervalo.

Memoria caché (CPU)

- ▶ En realidad, los procesadores actuales no contienen una única memoria caché, sino que ésta a su vez **está también dividida en niveles** (a veces hasta 3 niveles anidados).
- ▶ La idea es la misma: cerca de la CPU se coloca caché más cara y rápida (menos espacio al mismo coste), y lejos caché más lenta pero más barata (más espacio de almacenamiento).



Memoria caché (CPU)

- ▶ El rendimiento de la caché es extremadamente importante en procesamiento de datos. También lo es la diferencia de coste entre procesadores similares pero con mayor capacidad de memoria caché.
 - ▶ Una diferencia de tan solo 5MB de almacenamiento de caché L3 puede significar aproximadamente 300€ de coste adicional en un procesador.
- ▶ Sin embargo, para sacar todo el rendimiento a una caché más grande nuestros **programas** y el **entorno software** en el que se ejecutan deben conocer esta ventaja y saber explotarla.
 - ▶ Cualquier programa (bien o mal implementado) ejecutará más rápido con una CPU más rápida (mayor frecuencia de reloj).
 - ▶ Solo los programas preparados para paralelizar su ejecución aprovecharán una CPU con más núcleos (*cores*) de ejecución.
 - ▶ Solo el software que realice una buena gestión de datos en memoria sacará el máximo partido de la jerarquía de memoria.

4.2 Estrategias procesamiento de datos en memoria

Procesamiento de datos: Hadoop vs. Spark

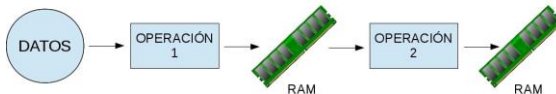
- ▶ Entornos como Apache Hadoop han permitido popularizar la paralelización masiva de tareas de procesamiento de datos sobre grandes clusteres de computación (cientos o miles de nodos).
- ▶ Sin embargo, Hadoop presenta dos graves **inconvenientes**:
 - ▶ Cada vez que concluye una fase (operación) de procesado de datos suele ser necesario almacenar el resultado en disco para su lectura posterior en la siguiente fase. Con la velocidad de acceso actual de los discos magnéticos convencionales (~5 mseg.), esto puede demorar considerablemente la ejecución de la tarea.
 - ▶ En muchos algoritmos de data mining/machine learning, aparecen patrones iterativos de acceso a datos (el resultado de la iteración $n - 1$ se usa en los cálculos de la iteración n). Esto implica constantes accesos a disco, que ralentizan la ejecución.
- ▶ Apache Spark es un entorno alternativo de computación distribuida de datos diseñado para resolver estos problemas.

Procesamiento de datos en memoria

HADOOP



SPARK #1



SPARK #2

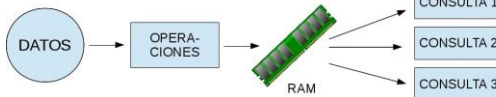


Figura 2: Diferentes opciones de aceleración de procesamiento de datos en memoria con Spark. (Hadoop) Guarda todos los resultados intermedios en memoria; (Spark #1) cachea resultados intermedios en memoria; (Spark #2) cachea resultado final para servirlo en múltiples consultas de los clientes.

Procesamiento de datos en memoria

- ▶ Si podemos almacenar resultados intermedios o datos ya procesados en memoria puedo acelerar mucho (x10 - x100) accesos posteriores.
- ▶ Paradigma: **programación funcional**.
 - ▶ Definir operaciones (en una función).
 - ▶ Aplicarlas en todos los datos (en varios cores o varios nodos).
- ▶ **IMPORTANTE:** debemos garantizar que hay suficiente espacio en memoria para almacenar los datos en cada nodo. De lo contrario...

Memory errors

```
Current usage: 4.3 GB of 4.1 GB physical memory used [...] Killing container.  
java.lang.OutOfMemoryError:...
```

Procesamiento de datos en memoria

- ▶ Podemos ver cómo Spark aprovecha la existencia de la jerarquía de memoria y el principio de localidad en computación para acelerar el procesamiento distribuido de datos en el cluster.
 - ▶ Una estrategia consiste en almacenar en memoria los resultados de la iteración anterior, de modo que la siguiente fase ya no tiene que acudir a disco a buscarlos sino que lo lee de memoria principal (mucho más rápida).
 - ▶ Otra estrategia consiste en almacenar en memoria resultados parciales importantes, que permitan acelerar las consultas que usuarios finales envían al sistema, evitando accesos a disco innecesarios para recuperar esos datos frecuentemente solicitados.

4.3 Software para procesamiento de datos en memoria

Procesamiento de datos en memoria

- ▶ **Apache Spark:** Entorno alternativo de procesamiento de datos para el ecosistema Hadoop. Combina almacenamiento de datos en memoria y en disco para ofrecer persistencia, así como múltiples módulos para diferentes propósitos (consultas SQL, procesamiento *streaming*, análisis de grafos, etc.) en varios lenguajes de programación (Scala, Java, Python).
- ▶ **Apache Geode:** Proyecto en estado de incubación, que ofrece una plataforma de gestión de datos para aplicaciones de consumo intensivo de información en tiempo real que funcionen sobre arquitecturas cloud ampliamente distribuidas (geográficamente). Su motor de procesamiento de datos en memoria ofrece notificaciones asíncronas de eventos y distribución garantizada de mensajes a cualquier nodo del cluster.

Procesamiento de datos en memoria

- ▶ **Aerospike:** Una de las bases de datos con motor de consultas en memoria más potentes de la actualidad. Permite sacar el máximo partido de la jerarquía de memoria, así como del almacenamiento de datos persistente de nueva generación (discos de estado sólido).
 - ▶ [Kayak](#), el conocido servicio online de búsqueda y personalización de viajes utiliza Aerospike para reducir drásticamente el número de nodos de computación necesarios para escalar internacionalmente.

Introducción tecnológica a Aerospike.

Presto (Facebook)

- ▶ Otro ejemplo de sistema que aprovecha el procesamiento de datos en memoria es Presto.
- ▶ Facebook posee uno de los almacenes de datos de mayor tamaño del mundo (+300 Petabytes).
- ▶ Necesidades: Análisis de grafos, machine learning y análisis interactivo.
- ▶ Crearon y liberaron (como software libre) Presto: motor de consultas SQL interactivo, enfocado en minimizar el tiempo de respuesta.



Presto (Facebook)

- Pequeña demo en línea (vídeo).

```
-> order by 2 desc
-> limit 5;

n_name | customers
-----+-----
VIETNAM | 6003717
MOROCCO | 6003115
ROMANIA | 6002183
CHINA   | 6001991
IRAN    | 6001889
(5 rows)

Query 20131105_005539_00082_ee7y3, FINISHED, 13 nodes
Splits: 187 total, 187 done (100.00%)
0:07 [150M rows, 22.4GB] [22.2M rows/s, 3.32GB/s]

presto:default> exit

$ exit
```

Figura 3: Captura del video demostración de Presto.

References I



[Hennessy & Patterson 2011] Hennessy, J. L., Patterson, D. A.
Computer Architecture: A Quantitative Approach (5 ed.)
Morgan Kaufmann, septiembre, 2011.



[Karau, 2015] Karau, H., Konwinski, A., Wendell, P. & Zaharia, M.
Learning Spark: Lightning-Fast Big Data Analysis
O'Reilly Media, Inc., febrero, 2015.

Análisis de Big Data en Python y R

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

22 de febrero de 2021



Contenidos

5.1 Programación científica

5.2 Programación científica en Python

5.3 Programación científica en R

5.4 Integración de Python y R

5.1 Programación científica

Definición: programación científica

- ▶ Creación de **modelos matemáticos**, **estadísticos** y aplicación de **técnicas numéricas** mediante lenguajes de programación (*software*) ejecutados sobre arquitecturas de computación (*hardware*).
- ▶ Plataformas *hardware*:
 - ▶ Computadores multiprocesador.
 - ▶ Plataformas de computación distribuida (e.g. cluster Hadoop).
 - ▶ Infraestructuras de computación en la nube.
 - ▶ Supercomputadores (arquitecturas especiales).

Ejemplo supercomputador: Mare Nostrum



Figura 1: Fotografía del «BSC-MareNostrum-E», el supercomputador más potente de nuestro país, en el BSNC (Barcelona).

Autor: Vcarceler - Trabajo propio. CC BY-SA 3.0 vía [Wikimedia Commons](#).

Lenguajes de programación científica

Pregunta:

¿Cuáles son los lenguajes de programación más populares en la industria?

Lenguajes de programación científica

Situación en 2015 (IEEE Spectrum)

1. Java.
2. C.
3. C++.
4. Python.
5. C#.
6. R.

<http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>.

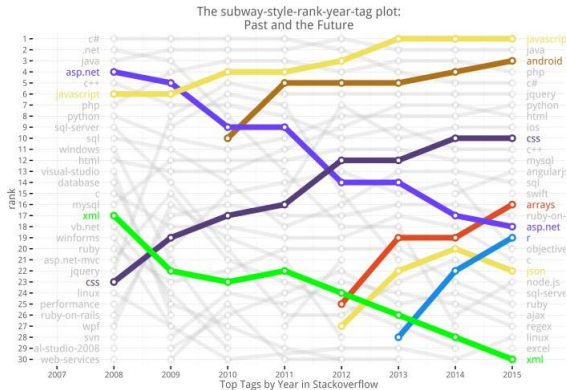
Situación en 2021 (TIOBE Index)

1. C
2. Java
3. Python
4. C++
5. C#
- [...]
11. R.

<https://www.tiobe.com/tiobe-index/>.

<https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved>.

Tags más frecuentes Stack Exchange (2015)



<http://jkunst.com/r/what-do-we-ask-in-stackoverflow/>

Lenguajes de programación científica

Conclusiones sobre popularidad de los lenguajes:

- ▶ Lenguajes de propósito general pero aplicables a sistemas integrados y computación científica en puestos de cabeza.
- ▶ R es uno de los lenguajes que ha experimentado un mayor crecimiento en popularidad durante los 8 últimos años.
- ▶ SQL es el tercer lenguaje que completa el conjunto de herramientas básicas para computación científica.
- ▶ No hay que perder de vista otros lenguajes emergentes, como [Julia](#) o [Go](#).

[Most popular programming languages to learn in 2021 \(Northeastern Univ.\).](#)

Retos en computación científica

1. Es muy difícil crear un solo lenguaje de programación que satisfaga los múltiples requisitos de computación científica.

- ▶ Intuitivo, sencillo de utilizar y comprender.
- ▶ Que soporte distintos paradigmas (programación imperativa, orientación a objetos, programación funcional).
- ▶ Que integre gran cantidad de bibliotecas para facilitar las labores más usuales en múltiples entornos (programación web, multiprocesado, estadística, análisis numérico, visualización de datos).

Python consigue cumplir con la mayoría de estos requisitos, pero no todos.

Retos en computación científica

2. No es inmediato extraer todo el rendimiento de sistemas de computación de altas prestaciones.

- ▶ Multiproceso, multihilo.
- ▶ Computación distribuida.
- ▶ Sistemas cloud.

Necesitamos plataformas adicionales que faciliten el procesamiento de grandes cantidades de datos.

- ▶ Por ejemplo clusters con *stack* tecnológico Hadoop.
- ▶ O sistemas ligeros y escalables de paralelización como [Dask](#) en Python.

Retos en computación científica

3. Conexión de componentes escritos en distintos lenguajes y ejecutándose en plataformas diferentes.

- ▶ Plataformas que integran varios lenguajes de programación (e.g. Spark: Python, Java, Scala, SQL, R...).
- ▶ Sistemas de comunicación avanzados multiplataforma y multilenguaje (e.g. ZeroMQ, ActiveMQ, Apache Kafka, Apache Samza).
- ▶ Distribuciones que faciliten la instalación y administración de bibliotecas y entornos de programación (Anaconda en Python, RStudio en R).

5.2 Programación científica en Python

Computación científica en Python

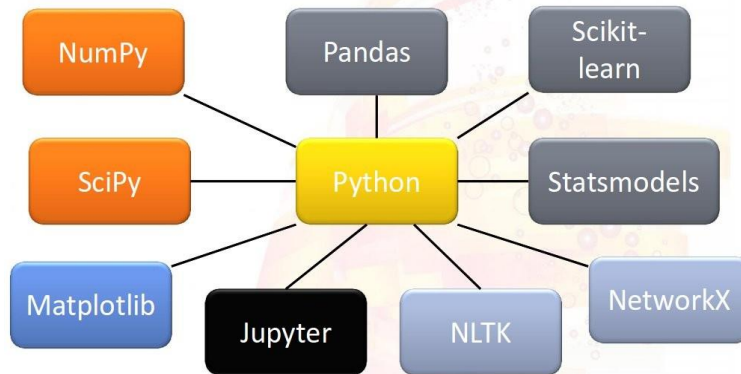


Figura 2: Algunos de los principales paquetes del ecosistema de computación científica en Python.

Computación científica en Python

Bibliotecas científicas básicas:

- ▶ **NumPy**: Cálculo numérico.
- ▶ **SciPy**: Cálculo científico avanzado.
- ▶ **Matplotlib**: representación gráfica.

Alternativas para visualización:

- ▶ **Bokeh**: Incluida en la distribución Anaconda.
 - ▶ Gráficos dinámicos, preparados para la web.
- ▶ **Seaborn**: Proyecto Univ. Stanford.

Computación científica en Python

Bibliotecas estadísticas:

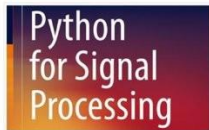
- ▶ **Pandas:** Análisis de datos más allá de NumPy y SciPy.
 - ▶ DataFrames, Series.
 - ▶ Semejante (con salvedades) al trabajo con R.
- ▶ **Scikit-learn:** Proyecto patrocinado por Google (entre otros).
 - ▶ Centrado en *machine learning*.
 - ▶ Pre-proceso de datos, selección del modelo, validación, estimación de errores.
 - ▶ Aprendizaje supervisado y no supervisado.
- ▶ **Statsmodels:** Ampliación de funciones estadísticas.

Proyecto Jupyter

<http://nbviewer.jupyter.org/>

Books

Python for Signal Processing



O'Reilly Book



Probabilistic Programming



Misc

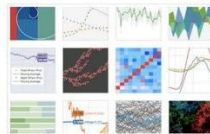
Data Visualization with Lightning



Interactive data visualization with Bokeh



Interactive plots with Plotly



Computación científica en Python

Otras aplicaciones.

- ▶ Comunicación con BB.DD.
- ▶ Trabajo con HTML / XML.
- ▶ Desarrollo web: Django, Flask.
- ▶ Procesado de Lenguaje Natural: NLTK.
- ▶ Análisis de grafos: NetworkX.

5.3 Programación científica en R

¿Qué es R?

- ▶ Originalmente creado por Ross Ihaka y Robert Gentleman, continuando trabajo de Bell Labs a partir del lenguaje S.
- ▶ Es un entorno completo para computación estadística: software y bibliotecas para análisis de datos, junto con un lenguaje de programación.
- ▶ Ampliamente utilizado por multitud de empresas y organizaciones, universidades y centros de investigación.

[Web del proyecto R].

CRAN

- ▶ Comprehensive R Archive Network.
- ▶ Descarga del entorno de programación y computación de base.
- ▶ Descarga de paquetes (bibliotecas) contribuidos por la comunidad de R, disponibles desde una red de mirrors por todo el mundo.
 - ▶ 7.345 paquetes (16/06/2015).
 - ▶ 17.156 paquetes (19/02/2021).
 - ▶ +10.000 paquetes nuevos publicados en 6 años (!!).
 - ▶ El ritmo no parece decaer a medio plazo.

[Web CRAN].

CRAN

- ▶ [Task views en CRAN].
- ▶ Permiten organizar mínimamente la gran cantidad de bibliotecas y recursos disponibles en CRAN, desglosados por temáticas o áreas de aplicación lógicas.
 - ▶ Finanzas, genética, psicométrica, estudios clínicos...
 - ▶ Enfoques bayesianos, gráficos, meta-análisis, análisis multivariante, etc.
- ▶ Otros repositorios populares: Bioconductor, R-Forge, más recientemente GitHub (con `devtools` y otros paquetes posteriores).

Sostenibilidad

- ▶ R Foundation.
- ▶ R Core Team.
- ▶ R Contributors.
- ▶ Institute for Statistics and Mathematics of WU (Viena, Austria).

En España: [\[Comunidad R Hispano\]](#).

Interfaz básico vs. RStudio

- ▶ R es un lenguaje de programación interpretado, enfocado al análisis interactivo de datos.
- ▶ La instalación básica cuenta con una GUI muy sencilla, al estilo de la IDLE de Python.
- ▶ Sin embargo, resulta muy limitada para la mayoría de flujos de trabajo cotidianos en análisis de datos.
- ▶ Alternativas: `Rcmdr` (más bien para fines didácticos), [RStudio](#) (IDE más utilizada actualmente para R.).
- ▶ RStudio desarrolla versiones de pago con soporte y otros productos adicionales (e.g. Shiny, para desarrollo de aplicaciones web).

Paseo inicial por R y RStudio

- ▶ Manual básico de iniciación a R.
 “R desde el principio: Curso cero de R”.
 Emilio López Cano, Javier M. Moguerza.
 Ediciones del Orto, Madrid.
 ISBN 978-84-7923-526-0
 [\[Página web\]](#) [\[PDF\]](#).

Big Data con R

Estrategias para abordar problemas de análisis de *big data* con R.

- ▶ **Muestreo de datos.**

- ▶ En ocasiones no necesitamos realmente analizar todos los datos, basta con generar un modelo sobre una muestra grande y representativa de los datos.
- ▶ Muestras abordables en R pueden llegar a $N \geq 10^9$ muestras o casos.
- ▶ Es importante garantizar que la muestra no esté sesgada, y que sea verdaderamente representativa del conjunto de datos original.

- ▶ Utilizar hardware con **más memoria principal (RAM).**

- ▶ Amazon lanza X1, primera instancia EC2 con hasta 2TB de memoria.
- ▶ Recordemos que si R se ejecuta en un sistema operativo de 64 bits, es capaz de direccionar hasta 8TB de memoria.
- ▶ Más que suficiente para muchos modelos de análisis de datos.

Big Data con R

- ▶ Utilización de paquetes especiales para desbordar objetos en disco.
 - ▶ `biglmm`: Ajuste de modelos `lm()` y `glm()` con conjuntos de datos almacenados fuera de memoria.
 - ▶ `ff` y `ffbase` proporcionan un método para almacenamiento y cómputo incremental con `data.frames` almacenados en disco.
 - ▶ `bigmemory`: almacenamiento de grandes objetos (como matrices) que desbordan fuera de memoria.
- ▶ Entornos de programación paralela de alto rendimiento con R (o interfaces externas).
 - ▶ `sparklyr`: Interfaz entre R y Apache Spark [[Página web](#)].
 - ▶ Revolution Analytics: Adquirido por Microsoft e integrado por completo en [HDI Insight](#).
 - ▶ `bigr`: paquete que proporciona integración total con IBM InfoSphere BigInsights.
 - ▶ `PivotalR`: proporciona integración con productos de Pivotal como Greenplum Database o HAWQ.

<https://cran.r-project.org/web/views/HighPerformanceComputing.html>

Big Data con R

- ▶ Integración con lenguajes de programación científica de mayor rendimiento (trabajo a más bajo nivel).
 - ▶ RCPP: Integración con C++.
 - ▶ `gputools`: Interfaz de acceso a funciones comunes implementadas con el toolkit Nvidia CUDA.
- ▶ En general, no hay una solución claramente dominante para resolver este problema. En muchos casos, las empresas optan por trabajar con un proveedor (IBM, TIBCO, Microsoft, etc.) que ofrezca una solución vertical integrada.

5.4 Integración de Python y R

Anaconda: Distribución computación científica

- ▶ Originalmente concebida para Python (multiplataforma).
- ▶ Desde enero 2015 disponible también para R.
- ▶ Desde septiembre 2015 se puede utilizar también R en los *notebooks* de Jupyter (pero no mezclar R y Python en el mismo *notebook*).

<https://www.continuum.io/downloads>.

<https://www.continuum.io/blog/developer/jupyter-and-conda-r>.

reticulate: Combinando R con Python

- ▶ Interfaz para Python desde el lenguaje R.
- ▶ Permite **combinar instrucciones en ambos lenguajes en el mismo archivo**.
- ▶ Se puede elegir el intérprete de Python que ejecutará el entorno en ese lenguaje (por ejemplo, un entorno virtual concreto para un proyecto).
- ▶ Además, permite transferir variables del entorno de ejecución R al entorno Python y viceversa **durante la ejecución**.
- ▶ Hay que tener en cuenta algunos detalles sobre formatos de representación de datos en ambos entornos para hacer bien el trasvase en algunos casos.

<https://rstudio.github.io/reticulate/>.

References I



[McKinney 2012] McKinney, W.

Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython

O'Reilly Media, Inc., noviembre, 2012.



[Zumel 2014] Zumel, N., Mount, J., Porzak, J.

Practical Data Science with R

Manning Publications, abril, 2014.



[Nolan 2015] Nolan, D., Temple Lang, D.

Data Science in R: A Case Studies Approach to Computational Reasoning and Problem Solving

Chapman & Hall/CRC Press, abril, 2015.

Ecosistema Hadoop

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

22 de febrero de 2021



Contenidos

6.1 Arquitectura Apache Hadoop

6.2 Componentes del Ecosistema Hadoop

4.2.4 Apache Kafka

6.1 Arquitectura Apache Hadoop

Ecosistema Apache Hadoop



Figura 1: Ecosistema original Apache Hadoop (v1).

Proveedores Apache Hadoop

- ▶ La infraestructura de análisis de datos de Hadoop no consta de un único componente, sino que requiere instalar y configurar múltiples componentes en diferentes capas:
 - ▶ Sistema de ficheros distribuido: HDFS.
 - ▶ Además, formatos de serialización de datos como Avro, Parquet, etc.
 - ▶ Bases de datos columnares: HBase, Cassandra.
 - ▶ Núcleo o motor de procesamiento de datos: Hadoop, Spark.
 - ▶ Sistema de gestión de recursos: YARN, Mesos.
 - ▶ Sistema de diseño y monitorización de tareas: Zookeeper, Oozie.
 - ▶ APIs de análisis de datos: Hive, Pig, Mahout, APIs de Spark.
- ▶ [The Hadoop Ecosystem Table](#) (completa descripción de componentes del stack tecnológico Hadoop).
- ▶ Existen **distribuciones Hadoop** creadas por empresas, con estos componentes y otros propios. Además de ofrecer servicio de soporte, mantenimiento, configuración y personalización.

Who has the most connections/partners?

Company	Employees (approx.)
CloudEra	100
Horizonworks	95
geni	90
BMC	85
HP	80
Tecadita	75
MapIt	70
Paracell	65
Jaspersoft	60
Microsoft	55
Oracle	50
Amazon	45
Informatica	40
MicroStrategy	35
SAP	30
Delameter	25
Perfata	20
Veeva	15
Talend	10
Talend	10
Talend	10
NetApp	5
Actuate	5
Orill	5
Cisco	5
Intel	5
Karmaphers	5
Qidtech	5
Revolution Analytics	5
Perseus	5
Fuzzy Logic	5
DataDix	5
Compacare Software	5
Udian	5
Ativia	5
StackIQ	5
RelayrnetDB	5
Snapsight	5
Digital Reasoning	5
Zetaweb	5
Kongitro	5
Concurrent	5
Zemanta	5
Bint	5
AMP	5
Splunk	5
Alpine Data	5
SAS	5
MetaScale	5
LucidWorks	5
SGI	5
Sun Microsystems	5
Ganzing	5
Fujitsu	5
Univa	5
Couchbase	5
Handlert	5

Proveedores Apache Hadoop

- ▶ Actualmente, **Cloudera** sigue siendo el **principal proveedor** de Hadoop a nivel mundial, con una cuota de mercado estimada en más del 50%. En enero de 2019 **se fusionó con Hortonworks, su principal competidor**.
 - ▶ Su política de negocio a día de hoy pasa por ofrecer un servicio de soporte integral, pero sobre ciertos componentes propietarios exclusivos de su plataforma (e.g. Cloudera Manager).
 - ▶ Esto crea importantes focos de incompatibilidad con otras distribuciones.
- ▶ Otras distribuciones como MapR siguen una estrategia de negocio similar.
- ▶ No obstante, tanto estos dos proveedores como la mayoría de las restantes empresas de este sector contribuyen muy activamente **al desarrollo** de los principales **componentes open source** del ecosistema.
 - ▶ Se trata de un esquema típico de **colaboración-competición** (*coopetition*). Las empresas colaboran en los componentes comunes para afianzar y mejorar la plataforma global, concentrándose luego en ofertar productos y servicios de valor añadido sobre la plataforma.

LF AI & Data

- ▶ En 2015 surge un proyecto de Linux Foundation llamado **Open Data Platform Initiative (ODPI)**.
- ▶ Pretende mitigar la **fragmentación tecnológica** que vive el ecosistema Hadoop, con proyectos similares que ofrecen funcionalidad similar pero incompatibles entre sí.
- ▶ El objetivo es proporcionar una **plataforma open source uniforme** para Hadoop, que garantice la interoperabilidad de los productos y acelere la implementación de **casos de uso**.
- ▶ En otoño de 2020 se fusionó con LF IA de Linux Foundation, para dar lugar al proyecto **LF AI & Data**, cuya misión es crear una comunidad abierta alrededor de proyectos de inteligencia artificial, *machine learning* y relacionados con datos.



6.2 Componentes del Ecosistema Hadoop

HDFS

- ▶ Modelado a partir del Google File System [2].
- ▶ Optimizado para maximizar el throughput, mejor cuanto más grandes sean los archivos.
- ▶ Tamaño de bloques grande, intenta optimizar distribución de datos en nodos siguiendo principios de localidad espacial y temporal (similar a jerarquía de memoria).
- ▶ Las primeras versiones utilizaban un nodo para metadatos (*NameNode*) y varios nodos para almacenar y trabajar con los datos (*DataNodes*). En las versiones más modernas, datos y metadatos están distribuidos.

Google File System

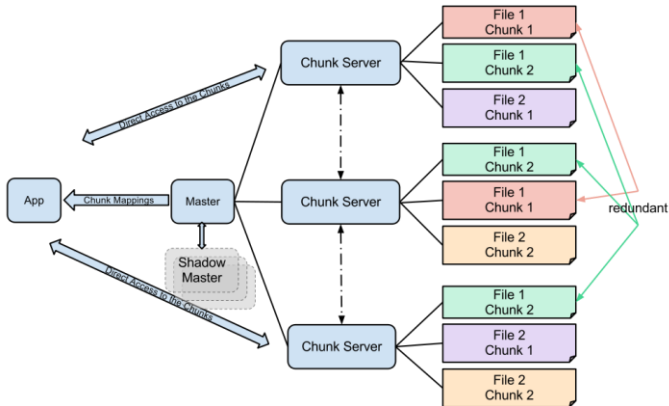


Figura 2: Esquema de organización de componentes de Google File System.

Almacenamiento en la nube

- ▶ Es posible poder ejecutar instancias Hadoop sobre servicios de computación en la nube.
- ▶ Ejemplo: servicios web de Amazon.
 - ▶ Podemos utilizar instancias de cómputo y almacenamiento de datos en Amazon EC2 para ejecutar nuestras tareas desde Hadoop (importación de recursos).
 - ▶ Adicionalmente, Amazon también proporciona Amazon Elastic MapReduce (EMR), un entorno de gestión alternativo similar a Hadoop, que permite gestionar las tareas y recursos de cluster.
 - ▶ EMR también puede albergar otros entornos para procesado de flujos de datos, como Apache Spark o Presto.
- ▶ Completan el [top-10 de proveedores cloud](#): Microsoft Azure, Google Cloud Platform, Alibaba Cloud, IBM Cloud (incluyendo [adquisición de Red Hat en 2019](#)), Oracle, SAP, Salesforce, Rackspace Cloud, VMWare.

Proyectos Hadoop: Hadoop (motor procesamiento)

- ▶ El núcleo de todo el ecosistema de aplicaciones.
- ▶ Hadoop Common Package (abstracciones) + HDFS (sistema de ficheros distribuido) + YARN (MapReduce engine).
- ▶ Se aplica el paradigma MapReduce a datos almacenados en múltiples nodos.
- ▶ Arquitectura Maestro-Esclavo (en su primera versión).
- ▶ Todavía presente en muchas arquitecturas, pero progresivamente sustituido por tecnologías más modernas, orientadas a *streaming data* (por ejemplo, Apache Spark, que veremos en el Tema 8).

Proyectos Hadoop: Hadoop

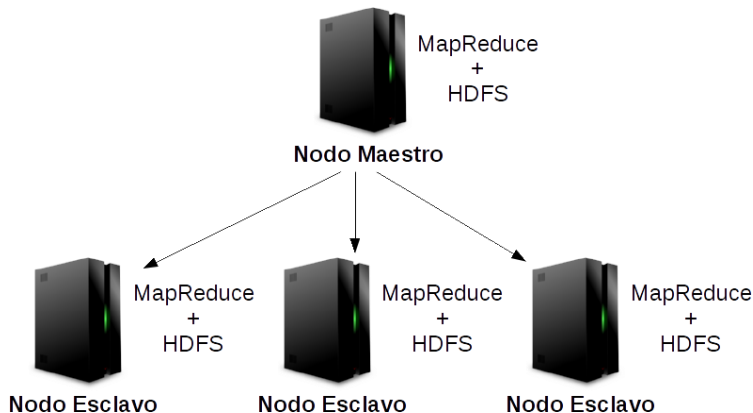


Figura 3: Infraestructura para Hadoop (primera versión, MR1).

Proyectos Hadoop: Hadoop

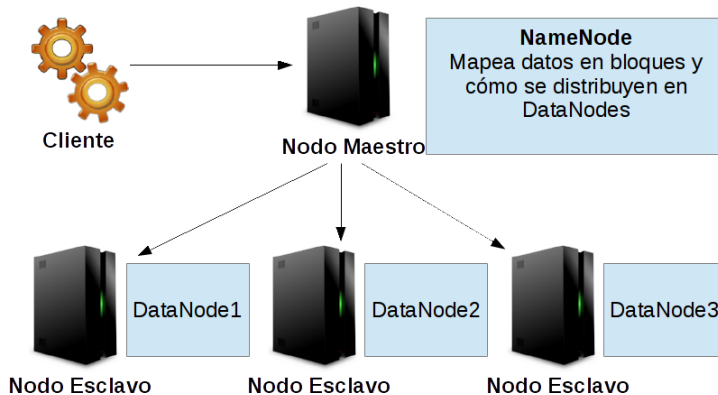


Figura 4: Gestión de datos en nodos en la primera versión, Hadoop MR1.

Proyectos Hadoop: Hadoop

- ▶ Gestión de tareas MapReduce (MR1).
- ▶ El JobTracker mantiene en memoria del nodo maestro información sobre todas las tareas planificadas y en ejecución.
- ▶ Se gestionan tanto las tareas de tipo `map` como las de tipo `reduce`, asociadas a trabajos de alto nivel enviados por el cliente.
- ▶ Límites versión 1 (MapReduce/MR1).
 - ▶ ~5.000 NODOS, 40.000 tareas concurrentes.
 - ▶ Distribución fija de recursos entre procesos `map` y `reduce`.
 - ▶ Fallos acaban con trabajos en ejecución y encolados (catastrófico).

Proyectos Hadoop: Hadoop

- ▶ Las limitaciones de MR1 obligaron a un rediseño completo del sistema de gestión y planificación → Nueva versión (MR2) del corazón de gestión de Hadoop: **YARN (Yet Another Resource Negotiator)**.
- ▶ Principales diferencias de YARN respecto a MapReduce (MR1).
 - ▶ Soporte para múltiples estrategias (*batch*, flujo de datos, interactivo).
 - ▶ Data Operating System (un solo conjunto de datos, múltiples instancias).
 - ▶ Gestión de metadatos y trabajos distribuido.
 - ▶ Mejor aprovechamiento de los recursos de computación y almacenamiento.
 - ▶ Introducción de aspectos de seguridad y autenticación en el diseño.
 - ▶ Compatibilidad hacia atrás (evitar grandes cambios para Hive, Pig, etc.).

Proyectos Hadoop: Hadoop

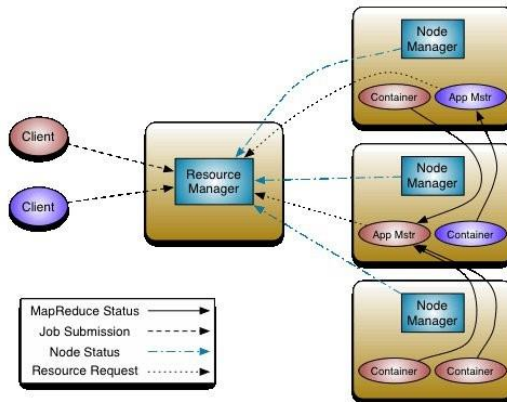


Figura 5: Representación esquemática de la organización y funciones de nodos en YARN.

Proyectos Hadoop: Hadoop

- ▶ Principales elementos de YARN (MR2).
- ▶ **Aplicación.**
 - ▶ Representación a alto nivel de un trabajo de procesamiento de datos.
 - ▶ Puede ser un trabajo MapReduce o un script de shell.
- ▶ **Contenedor.**
 - ▶ Unidades de particionado de los recursos hardware subyacentes.
 - ▶ Permiten la distribución de los recursos computacionales entre diferentes aplicaciones en ejecución, intentando maximizar el aprovechamiento de los mismos.

Proyectos Hadoop: Hadoop

- ▶ Principales componentes de YARN (MR2).
- ▶ **Resource Manager.**
 - ▶ Gestión de tareas de alto nivel.
 - ▶ Gestión de colas jerárquicas de trabajos.
- ▶ **Application Master.**
 - ▶ Uno por cada instancia a nivel de aplicación.
 - ▶ Gestiona recursos, progreso de tareas, planificación, etc.
- ▶ **Node Manager.**
 - ▶ Agentes encargados de la gestión y monitorización de contenedores en cada nodo del cluster.

Proyectos Hadoop: Hadoop

- ▶ Últimos avances en YARN (MR2).
- ▶ El Resource Manager continúa siendo un punto de fallo singular. Genera graves trastornos en caso de pérdida de información.
- ▶ Solución: intentar reconstruir la información que contiene en caso de fallo.
- ▶ Propuesta: Zookeeper (u otro meta-gestor de recursos) debe monitorizar las instancias de Resource Managers y actuar en caso de fallo.

Proyectos Hadoop: Apache Hbase

- ▶ Base de datos NoSQL, diseñada a imagen de Google BigTable, escrita en Java.
- ▶ Orientada a partición por columnas.
- ▶ Tolerancia a fallos.
- ▶ Compresión de datos, operaciones en memoria, filtros Bloom.
- ▶ Funciona sobre HDFS.



Proyectos Hadoop: Apache Cassandra

- ▶ Base de datos NoSQL liberada por Facebook en 2009.
- ▶ Especialmente pensada para requisitos de alta disponibilidad.
- ▶ Replicación en múltiples nodos (incluso alejados geográficamente).
- ▶ Diferentes niveles de consistencia de datos entre réplicas (configurable).
- ▶ No admite operaciones como JOIN ni subconsultas.



Proyectos Hadoop: Apache Hive

- ▶ Sistema *datawarehouse* que ejecuta sobre Hadoop.
- ▶ Programar operaciones para análisis de datos directamente sobre Hadoop puede llegar a ser muy tedioso.
- ▶ Hive proporciona un lenguaje de abstracción similar a las consultas SQL.
- ▶ Procesado de logs (tráfico web, sistemas), minería de texto, indexación de documentos, inteligencia de negocio, predicciones y contraste de hipótesis.



Proyectos Hadoop: Apache Mahout

- ▶ Construcción de bibliotecas de machine learning sobre Hadoop.
- ▶ Clustering (K-means, K-means con lógica difusa).
- ▶ Sistemas de recomendación.
- ▶ Múltiples clasificadores:
 - ▶ Regresión logística.
 - ▶ Naive Bayes.
 - ▶ Árboles de decisión.
 - ▶ Random forest, etc.



Proyectos Hadoop: Flujos de datos

► Apache **Flume**.

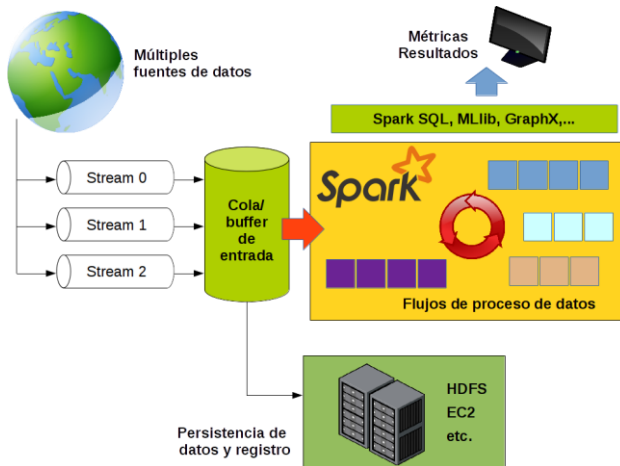
- Servicio distribuido de recolección, agregación y transmisión de colas de mensajes, típicamente registros (logs) de sistemas.
- Generalizable a cualquier otra fuente de datos en streaming (correo electrónico, redes sociales, etc.).
- Los “agentes” de Flume abstraen las labores de adquisición de datos de una fuente y almacenamiento de esos datos, hasta que son consumidos (por ejemplo, reenviándose a otro agente, o guardándolos de forma persistente en un sistema HDFS).

Otros proyectos: Procesamiento *streaming*

- ▶ Apache **Samza**:
 - ▶ Framework para procesamiento de flujos de datos, concebido para su utilización junto a Apache Kafka y YARN.
 - ▶ Combina un sistema de streaming de datos (Apache Kafka) con un gestor de recursos (que puede ser YARN) y una API que ofrece primitivas y rutinas de procesamiento de flujos de datos.
- ▶ Apache **Kafka**.
 - ▶ Sistema *message broker* para gestionar flujos de datos de alta velocidad y elevado throughput.
 - ▶ Fue creado originalmente para tratamiento de logs de eventos en la plataforma LinkedIn, pero su uso se puede generalizar a otros tipos de flujos de datos.
 - ▶ Puede funcionar en combinación con Samza, YARN, Spark, etc.
 - ▶ Optimizado para procesamiento de grandes volúmenes de mensajes minimizando las probabilidades de pérdida de información.
- ▶ Apache **Spark** (descrito en el Tema 8).

4.2.4 Apache Kafka

Procesamiento *streaming*



Introducción Apache Kafka

- ▶ Ante todo, Apache Kafka es un sistema de mensajería:
 - ▶ Distribuido (funciona en un nodo o un cluster).
 - ▶ Particionado (permite repartir los datos entre los diferentes nodos).
 - ▶ Replicado (redundancia para evitar pérdida de información).
- ▶ Está basado en mensajes “commit-log”, por lo que puede mantener una estructura temporal estricta de los datos de entrada.
- ▶ Puede actuar como una cola de mensajes estándar, o como un sistema *publish-subscribe*.

Características Apache Kafka

- ▶ Mensajería persistente de grandes volúmenes de datos. Puede almacenar TB o PB de información.
- ▶ Diseñado para proporcionar rendimiento de tiempo constante en acceso a cualquier dato en disco. Tiene también un *throughput* muy elevado (lecturas y escrituras simultáneas).
- ▶ Múltiples clientes, incluyendo: Java, Ruby, Python, Scala; bases de datos NoSQL; Hadoop y Spark; Storm, etc.
- ▶ Cumple requisitos de *real-time*: los mensajes de entrada estarán disponibles de inmediato para los consumidores.
- ▶ Permite carga de datos en paralelo para Hadoop y Spark.

Arquitectura Apache Kafka

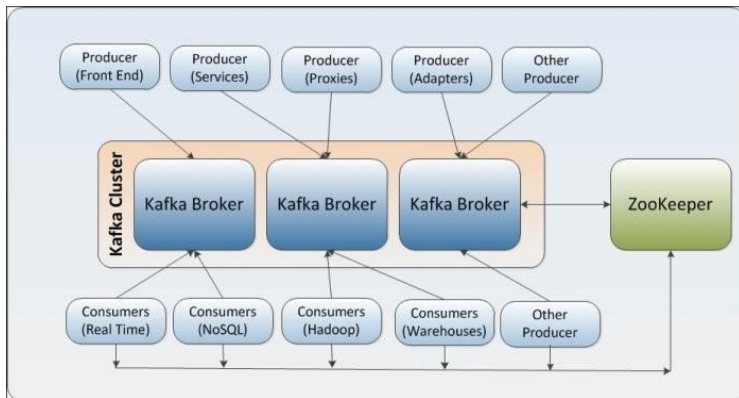


Figura 6: Arquitectura de Apache Kafka. *Learning Apache Kafka, 2nd Ed.*

Ejemplos de aplicación típicos

- ▶ Sistemas de búsqueda en Internet, recomendación personalizada y publicidad.
- ▶ Sistema continuo de análisis de sentimiento y opinión (redes sociales).
- ▶ Seguridad, monitorización de tráfico y usuarios, gestión de infraestructuras.
- ▶ IoT, smart cities: sensores y otros dispositivos que generan flujos continuos de datos.

Ejemplos de aplicación típicos

- ▶ Agregación de logs: centralización de los logs de múltiples sistemas en un data pool.
- ▶ Procesado de datos en streaming (Spark Streaming, Storm, Flink).
- ▶ Cache para logs de commits en sistemas heterogéneos y distribuidos, de forma que ante un fallo de uno de los nodos este pueda recuperar su información de estado antes de reanudar.
- ▶ Mensajería: Aplicación tradicional (como ActiveMQ o RabbitMQ). Desacoplamos productores y consumidores al actuar Kafka como un dbus, con un protocolo de comunicación universal para todos ellos.

Componentes clave

- ▶ **Topic:** Etiqueta que indica la categoría o nombre del *feed* al que se debe entregar el mensaje.
- ▶ **Broker:** Cada nodo de un cluster de Kafka tiene uno o varios procesos que se denominan brokers y se encargan de gestionar las colas de mensajes. Los *topics* se crean dentro de estos procesos *broker*.
- ▶ **Zookeeper:** Componente de monitorización y gestión del ecosistema Hadoop. Supervisa la coordinación en la interfaz entre un broker de Kafka y los consumidores conectados al mismo.
- ▶ **Productores:** Publican datos a los *topics*, donde pueden elegir una partición para escribirlos si lo desean.
- ▶ **Consumidores:** Consumen datos de Kafka, bien extrayendo datos de una cola de mensajes (*message queueing*) o bien suscribiéndose a uno o varios topics (*publish-suscribe*).

Particiones

- ▶ Los topics en Kafka están divididos en un cierto número de *particiones*. Estas permiten paralelizar los datos de un topic, distribuyéndolos a través de múltiples brokers.
- ▶ Aunque no es obligatorio, es habitual que cada partición esté en un nodo diferente para paralelizar la lectura de los datos por parte de los consumidores.
 - ▶ Varios consumidores leen del mismo topic a la vez.
 - ▶ Un consumidor puede leer de múltiples particiones de un mismo topic, paralelizando así el acceso a los datos.

Topics y particiones

Anatomy of a Topic

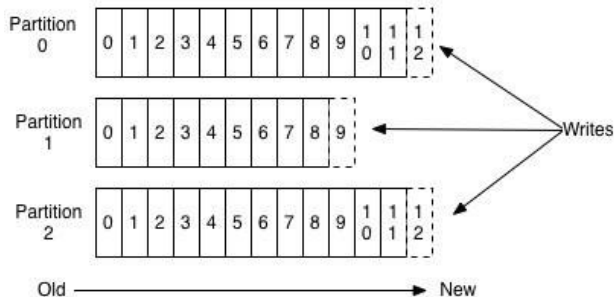


Figura 7: Distribución lógica de particiones en un topic.

[Kafka in a nutshell]

Topics y particiones

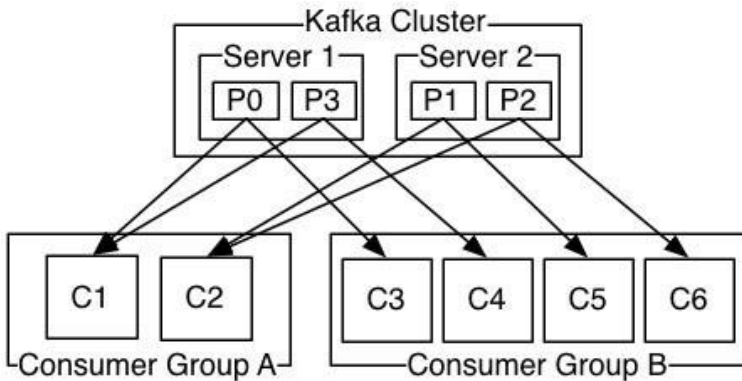


Figura 8: Dos grupos de consumidores leyendo de varias particiones a la vez.

Offset de lectura

- ▶ Desde un punto de vista lógico, Kafka se comporta como un gran log de mensajes del que los consumidores pueden ir leyendo información.
- ▶ Cada consumidor mantiene un **offset** de lectura, que indica hasta donde ha leído dentro de la cola de mensajes correspondiente a un topic.
- ▶ Si el cliente falla, una vez que reanude su funcionamiento es capaz de volver a empezar a leer desde el último mensaje que procesó.
- ▶ En ocasiones, sin embargo, dependiendo de la política de funcionamiento que siga el cliente se puede dar el caso de pérdida o duplicidad de mensajes, salvo que decida utilizar un sistema transaccional que garantice que cada mensaje se procesó correctamente solo una vez.

Topics y particiones

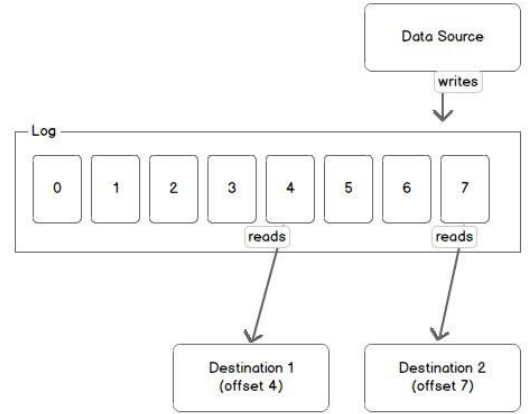


Figura 9: Comportamiento lógico de Kafka con los consumidores.

References I

-  [White 2015] White, T.
Hadoop: The Definitive Guide, 4th Ed.
O'Reilly Media, Inc., marzo, 2015.
-  [Sitto & Presser, 2015] Sitto, K., Presser, M.
Field Guide to Hadoop
O'Reilly Media, Inc., marzo, 2015.
-  [Holmes, 2014] Holmes, A.
Hadoop in Practice, 2nd Ed.
Manning Publications, septiembre, 2014.
-  [Eadline, 2015] Eadline, D.
Hadoop 2 Quick-Start Guide: Learn the Essentials of Big Data Computing in the Apache Hadoop 2 Ecosystem.
Addison-Wesley Professional, noviembre, 2015.

Bases de Datos No Convencionales (NoSQL)

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

20 de marzo de 2018



Contenidos

7.1 Clasificación de Bases de Datos

7.2 Bases de Datos Clave-Valor

7.3 Bases de Datos Documentales

7.4 Bases de Datos Columnares

7.5 Bases de Datos para Grafos

7.1 Clasificación de Bases de Datos

Modelo de datos

- ▶ Define el diseño y la implementación del sistema que almacena y gestiona los datos.
- ▶ La tecnología de almacenamiento de datos que utilizemos debe proporcionar soporte para guardar, consultar, actualizar y/o exportar grandes volúmenes de datos, asumiendo el modelo de datos.
- ▶ **Datos estructurados.**
 - ▶ Podemos almacenar sus valores en **campos predefinidos**, con una clase o **tipo de datos fijo asociado**.
 - ▶ Ejemplo: sistemas de **bases de datos relacionales** (RDBMS).

Modelo de datos

- ▶ **Datos semi-estructurados o no estructurados.**
- ▶ No podemos predefinir de antemano qué campos tendrán, ni el tipo de datos de cada campo, por lo que necesitamos un modelo de datos más flexible para su gestión.
- ▶ Relaciones complejas entre los diferentes elementos de datos.
- ▶ Ejemplo: Sistemas **NoSQL** (*Not Just o Not Only SQL*).
- ▶ Diferentes alternativas: particionado por columnas, documentos, grafos, clave-valor, etc.).

Retos sistemas NoSQL

- ▶ Los nuevos sistemas NoSQL surgen para responder a varios retos:
 - ▶ Necesidad de **escalabilidad**, absorbiendo picos de demanda muy acusados.
 - ▶ Costes: Debemos considerar tanto el coste económico del propio software (por ejemplo, si es un producto propietario pero ofrece funciones mucho más avanzadas) como el coste de la infraestructura necesaria para su funcionamiento (normalmente, *cloud computing*).
 - ▶ Disponibilidad: Muchas tecnologías NoSQL funcionan por defecto sobre un sistema distribuido, que proporciona redundancia y mejora la disponibilidad del servicio.
 - ▶ Flexibilidad: Aunque se centren en un paradigma de trabajo con datos, las bases de datos NoSQL ofrecen mayor flexibilidad a la hora de consultar y recuperar información.

Bases de datos relacionales

- ▶ Numerosas opciones en el mercado, propietarias o software libre.
- ▶ Larga trayectoria, tecnología muy madura y consolidada, permite predecir hasta cierto punto rendimientos esperados.
 - ▶ Oracle, MySQL, MariaDB, PostgreSQL, SQLite, etc.
- ▶ Gran variabilidad en cuanto a soporte para big data.
 - ▶ Tipos de datos nativos.
 - ▶ Particionado de tablas.
 - ▶ Clustering, alta disponibilidad.
- ▶ Object-Relational Mapping (ORM).
 - ▶ Ejemplo: [SQLAlchemy](#) (Python).



Rendimiento RDBMS

- ▶ Es conveniente tener en cuenta algunas premisas importantes para analizar datos utilizando RDBMS.
- ▶ Utilizar motores *ACID-compliant* únicamente cuando sea imprescindible.
 - ▶ Si estamos trabajando con datos localmente, a los que solo accede uno o varios analistas, podemos usar otras opciones más rápidas.
 - ▶ Ejemplos: MyISAM (MySQL), ARIA (MariaDB).
- ▶ Desactivar claves primarias/foráneas en carga de datos. Después, utilizar solo cuando sea imprescindible. Activar particionado de tablas en diferentes ficheros (e.g. InnoDB).
- ▶ Cuidado con la codificación (asegurar UTF-8 siempre que sea posible).



Rendimiento RDBMS

- ▶ La configuración por defecto de cualquier base de datos relacional no suele ser adecuada para el análisis de datos.
 - ▶ Incrementar tamaño de los ficheros de claves (búsquedas, ordenación).
 - ▶ Configurar adecuadamente el sistema de logging (errores, consultas lentas, etc.) ya que puede consumir rápidamente espacio en disco.
- ▶ Formular las consultas basándonos siempre en **conjuntos de datos** y operaciones sobre los mismos (**JOIN**, **LEFT JOIN**, etc.).
 - ▶ Evitar en lo posible cláusulas del tipo **WHERE** $\times$ **IN** (...), ya que suelen involucrar búcles de búsqueda lentos (*full scan*).



NoSQL

- ▶ NoSQL = Not Only SQL.
- ▶ Escalabilidad y alto rendimiento para big data (en especial, tratamiento de tipos de datos muy heterogéneos o información textual).
- ▶ **Esquemas clave-valor.**
 - ▶ Almacen de datos en pares clave-valor, no precisan esquema (Riak, Redis, Voldemort, etc.).
- ▶ **Almacen de columnas.**
 - ▶ Particionan datos por columnas, de forma que podemos paralelizar consultas sobre subconjuntos de datos muy grandes (HBase, Cassandra).

Almacenamiento de datos: NoSQL

► Orientadas a documentos.

- Cada clave está asociada a un documento, codificado según algún estándar de representación de datos (JSON, XML, YAML, etc.).
- Los documentos pueden contener muchos pares clave-valor, clave-array (para listas de datos) u otros documentos.
- Ejemplo: MongoDB.

► Grafos.

- Almacenan explícitamente información sobre nodos y sus relaciones, optimizando consultas que recorren los grafos (Neo4J).
- A cambio, tenemos que aprender un lenguaje de consultas muy diferente.

Ventajas NoSQL

- ▶ Mejor **escalado horizontal**, permite procesamiento paralelo.
 - ▶ Consideran la agregación de nuevos recursos de computación “en caliente” (autosharding, replicación).
- ▶ **No es necesario definir esquemas** (tipos de datos), se pueden mezclar dinámicamente nuevos datos con los ya existentes (con limitaciones).
- ▶ Mejor integración con metodologías ágiles de desarrollo de software.
 - ▶ Sprints cortos, prototipado rápido (vs. esquemas predefinidos).
 - ▶ Dificultad para establecer a priori esquemas fijos de estructuras de datos.
- ▶ Posibilidad de optimizar la recuperación de información mediante indexación (múltiple, dispersa, geoespacial, etc.).

Inconvenientes NoSQL

- ▶ Requieren **importantes conocimientos técnicos** para su instalación, correcta configuración y administración (recordemos importancia del rendimiento).
- ▶ Todavía escasa madurez en comparación con RDBMS. En problemas difíciles no está claro el rendimiento *a priori*.
- ▶ Múltiples estándares de programación y APIs, incompatibles entre sí en muchos casos (necesidad de soporte de comunicación).
- ▶ Problema sistemas distribuidos: **teorema CAP** (*consistencia, disponibilidad, particionado*).

7.2 Bases de Datos Clave-Valor

BBDD por pares clave-valor

- ▶ Son la forma más simple de implementación de bases de datos NoSQL.
- ▶ Los datos están almacenados con una **clave unívoca** asociada a **cada valor**.
- ▶ Es imprescindible que cada valor de clave no se repita. Por lo tanto, los sistemas y aplicaciones software que usen BBDD de este tipo deben establecer una **nomenclatura** o convenio de reglas para crear nombres de claves:
 - ▶ Por ejemplo, el nombre del cliente con ID = 1 puede tener asignada la clave `cust1.name` en una aplicación.
 - ▶ Sin embargo, el cliente con ID = 1 en un sistema de datawarehouse que use la misma base de datos puede tener asignada la clave `wrhs1.name`

BBDD por pares clave-valor

- ▶ Es común seguir un convenio de *namespaces* (nombres de dominio). Se definen los prefijos que usaran todas las claves creadas por cada sistema o aplicación software que comparta la BBDD.
 - ▶ `aap1.nombre_campo`
 - ▶ `aap2.nombre_campo`
 - ▶ `sist1.nombre_campo`
 - ▶ `sist1.nombre_campo`
- ▶ Esto garantiza que no se confundan claves con el mismo nombre de campo que provengan de distintas aplicaciones.

BBDD por pares clave-valor

- ▶ Algunas de estas BBDD admiten varios *namespaces* en el mismo sistema. Comúnmente, se suele llamar a estas estructuras **buckets**. Las claves no se pueden repetir dentro del mismo bucket, pero si se pueden repetir en diferentes buckets (no se ven entre sí).
- ▶ Para acceder a los valores, es esencial conocer la nomenclatura de claves y namespaces empleada.
 - ▶ Ej: en una BD relacional, podemos acceder al dato dirección del cliente con ID = 123 indicando en la consulta SQL el nombre de la tabla, el ID (clave primaria) y el campo a recuperar (*address*).
 - ▶ En una BD clave-valor simplemente se recupera el dato de la clave `cust123.address`.

BBDD por pares clave-valor

- ▶ El ejemplo de uso más clásico de este tipo de BBDD es para crear **cachés de datos locales** para nuestros sistemas o aplicaciones, que además pueden ser accesibles por otros sistemas o aplicaciones que accedan a los datos (dentro de unos ciertos límites de consistencia de valores).
- ▶ Objetivo: recuperar valores asociados a una clave. Normalmente, los datos se guardan en memoria RAM del sistema, aunque también se puede añadir persistencia en disco.
- ▶ Sin embargo, no podremos realizar consultas complejas o en base a nombres de columnas como en SQL.

Ejemplo: Redis

- ▶ Redis es un buen ejemplo de sistema de almacenamiento de datos NoSQL.
- ▶ Tremendamente popular, puesto que ofrece un buen rendimiento tanto con datos en memoria como con datos en disco.
- ▶ Buen enlace con Python: *redis-py*
 - ▶ `sudo pip install redis`
- ▶ Limitación: soporte oficial solo para plataformas GNU/Linux, *BSD u OS X, aunque existe una versión desarrollada por Microsoft para portarlo a Windows (64 bits).

7.3 Bases de Datos Documentales

BBDD documentales

- ▶ Se pueden considerar, en cierto modo, como una evolución de las BBDD clave-valor.
- ▶ Están orientadas a almacenar documentos con datos semi-estructurados: XML, HTML, JSON, etc.
- ▶ La forma de manejarlos es considerar cada documento como una **colección de pares clave valor**.
 - ▶ Cada “etiqueta” con el nombre de un elemento es la clave.
 - ▶ El valor es el contenido de la etiqueta.
 - ▶ Además, muchas de estas BBDD también son capaces de guardar y explotar metadatos sobre los elementos (e.g. atributos).

BBDD documentales

- ▶ Las BBDD documentales sí que nos permiten realizar consultas para filtrar por el valor de un campo.
 - ▶ Por ejemplo, si tenemos una colección de documentos con las transacciones realizadas por los clientes de una tienda virtual, cada una almacenada como un objeto JSON, podemos recuperar todas las transacciones realizadas durante el último mes.
- ▶ Otra ventaja muy importante es que los documentos no tienen por qué tener los mismos atributos.
 - ▶ No vale la pena almacenar un montón de elementos con valor NULL en documentos que no tienen valor para ese campo, o en los que no tiene ningún significado.

Ej. documento JSON

```
[  
  {  
    "customer_id":187693,  
    "name":"Kiera Brown",  
    "address": {  
      "street": "1232 Sandy Blvd.",  
      "city": "Vancouver",  
      "state": "WA",  
      "zip": "99121",  
    },  
    "first_order": "2013/01/15",  
    "last_order": "2014/06/29"  
  },  
  {  
    "customer_id":187699,  
    "name":"Magda Brown",  
  }  
]
```

BBDD documentales: diseño

- ▶ En el diseño de este tipo de bases de datos, es importante intentar no mezclar tipos de documentos diferentes en la misma colección.
 - ▶ Aunque sería posible seguir haciendo filtrado y búsquedas, se hacen mucho más lentas puesto que no compartirán muchas claves.
- ▶ Por tanto, aunque existe flexibilidad para que los campos no tengan que ser los mismos en cada documento, es importante que compartan al menos muchos de los campos (típicamente, al menos un 80%).
- ▶ Otra pista de un mal diseño ocurre cuando en nuestro código de aplicación surgen funciones o bloques de código que manipulan tipos de documentos diferentes (e.g. muchos `if` seguidos). En ese caso, también es recomendable separar documentos distintos en colecciones diferentes.

BBDD documentales: diseño

- ▶ Otro aspecto crucial para el diseño de este tipo de BBDD es el **particionado** de datos en disco, o en diferentes servidores.
- ▶ Por ejemplo, si sabemos de antemano que las consultas se harán sobre los documentos registrados mes a mes, o cada trimestre, podemos particionar los datos en disco siguiendo ese intervalo.
- ▶ La ventaja es que cuando aparezca una consulta que solo involucre a un único intervalo (o a un conjunto pequeño) la BBDD no debe escanear toda la colección de documentos, sino solo la(s) particiones involucradas en la consulta.

Ejemplo: MongoDB

- ▶ Una de las soluciones más populares para bases de datos documentales.
- ▶ Permite consultas por campos, rangos y también utilización de expresiones regulares.
- ▶ Desde el punto de vista de infraestructura, proporciona indexación (acelera consultas sobre campos muy utilizados); replicación (escalabilidad); balanceo de carga y funciones de agregación (similares a la cláusula **GROUP BY** en SQL).
- ▶ Dispone de clientes en numerosos lenguajes de programación, incluidos C, C++, Python, Ruby y Scala.
- ▶ Su principal inconveniente radica en que no proporciona consistencia de datos (ACID), por lo que su utilización en entornos con gran volumen de actualización de datos queda en entredicho.

Ejemplo: PostgreSQL

- ▶ Una de las bases de datos relacionales de más larga trayectoria y más popular.
- ▶ Sin embargo, no tanta gente sabe que también soporta desde su versión 9.3 almacenamiento de documentos en formatos JSON Y RJSON.
- ▶ La [comparativa MongoDB vs. PostgreSQL](#) es bastante elocuente: PostgreSQL mejora el rendimiento de MongoDB en todos los aspectos.
 - ▶ Mayor velocidad de adquisición de datos.
 - ▶ Menor consumo de espacio en disco.
 - ▶ Mayor velocidad de inserción de nuevos datos.
 - ▶ Menor tiempo de ejecución de consultas.

7.4 Bases de Datos Columnares

BBDD columnares

- ▶ La precursora de esta familia fue la BD BigTable, descrita por Google en 2006.
- ▶ Esta familia toma también otros nombres, como “wide column databases”, para poner énfasis en que pueden manejar decenas de miles de columnas.
- ▶ Principales características:
 - ▶ Las columnas se pueden alterar dinámicamente.
 - ▶ Los datos se indexan por identificador de fila, nombre de columna y timestamp.
 - ▶ Los diseñadores y administradores tienen control sobre la ubicación de los datos.
 - ▶ Las lecturas y escrituras en una fila son atómicas.
 - ▶ Las filas se mantienen siempre ordenadas (con frecuencia, por *timestamp*).

BBDD columnares: diseño

- ▶ Las columnas se mantienen agrupadas en familias de columnas (con una etiqueta identificadora), que mantienen cerca (en disco) los datos relacionados entre sí que se suelen acceder conjuntamente.
- ▶ Sin embargo, las columnas de una misma fila pueden o no estar almacenadas en bloques de disco consecutivos.
- ▶ Ejemplo: una empresa con negocio en España.
 - ▶ El diseñador abre la familia de columnas `dirección` sin especificar qué columnas hay dentro.
 - ▶ El desarrollador puede entonces crear una columna `comunidad autónoma` dentro de esta familia.
 - ▶ Si más adelante se abre negocio en Alemania, el desarrollador puede crear directamente otra columna llamada `región`.

BBDD columnares

- ▶ Con diferencia, la propiedad más interesante de este tipo de BBDD es que mantiene siempre los datos ordenados dentro de las columnas.
- ▶ Esto las convierte en la solución ideal para aplicaciones de Internet de las Cosas: datos de sensores, vigilancia, redes, etc.
- ▶ El principal inconveniente de estas bases de datos es que no entienden el concepto de tipo de datos de una columna. Los datos en cada columna se almacenan simplemente como bytes, y por tanto los tiene que interpretar la aplicación cliente, no la BD.
- ▶ Como consecuencia de esta limitación, tampoco se pueden hacer operaciones de tipo `JOIN`.

BBDD columnares: ejemplos

- ▶ **HBase:** Modelada a partir del producto BigTable de Google, como alternativa open source dentro del ecosistema Hadoop.
 - ▶ Se basa en el sistema de ficheros HDFS, con arquitectura master/slave.
 - ▶ Zookeeper mantiene la coordinación entre los diferentes nodos que contienen los datos.
 - ▶ Los datos se almacenan en **regiones**, compuestas por varios nodos que operan de forma solidaria.
- ▶ **Cassandra:** Creada en el seno de Facebook, utiliza un sistema de distribución de datos *peer-to-peer* entre los nodos para pasar información, mantener la consistencia y actualizar datos.

7.5 Bases de Datos para Grafos

BBDD para grafos

- ▶ Son el último tipo surgido dentro de las BBDD NoSQL.
- ▶ Almacenan información en la BD en forma de grafos.
 - ▶ Nodos de la red.
 - ▶ Enlaces entre los nodos.
 - ▶ Propiedades que describen a los nodos y los enlaces (opcional).
- ▶ Permiten ejecutar consultas semánticas a través del grafo de datos conectados.
- ▶ Muchas de ellas incorporan algoritmos distribuidos y escalables para analizar el grafo y obtener métricas, así como para facilitar las labores de visualización de datos de la red.

BBDD para grafos: ejemplos

- ▶ **Neo4J**: Una de las más populares en este ámbito.
- ▶ Proporciona un rendimiento razonable, aunque todavía tiene que mejorar su eficiencia en procesamiento de grafos de gran tamaño.
- ▶ Por otro lado, es necesario aprender su propio lenguaje de programación (*Cypher Query Language*) para poder manejarla.
- ▶ **Titan**: Otra alternativa muy popular, para manejo de grafos de gran tamaño ($+120 * 10^9$ nodos).

References I



[Sullivan 2015] Sullivan, D.
NoSQL for Mere Mortals
Pearson, abril, 2015.



[Harrison 2015] Harrison, G.
Next Generation Databases: NoSQL and Big Data
APress, diciembre, 2015.

Procesado de datos en Memoria (técnicas avanzadas)

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

22 de febrero de 2021



Contenidos

8.1 Procesamiento en Memoria con Bases de Datos

8.2 Apache Spark

8.3 El stack tecnológico de Spark

8.4 Procesamiento de Datos con SSDs

8.1 Procesamiento en Memoria con Bases de Datos

Tablas en memoria

- ▶ Al presentar las técnicas de trabajo con datos en bases de datos (relacionales o NoSQL), en muchas ocasiones solo se hace referencia a los motores de consulta que gestionan los datos en disco.
- ▶ En estos casos, si contamos con discos magnéticos convencionales (más adelante tratamos los SSDs), normalmente las operaciones de consulta tardarán bastante tiempo.
 - ▶ Consideremos por ejemplo una operación **JOIN** entre dos tablas, t_1 y t_2 .
 - ▶ Si los datos que recuperamos de cada tabla no caben en la memoria asignada para el servidor de BBDD, entonces se deben escribir en disco para efectuar el **JOIN**.

Tablas en memoria

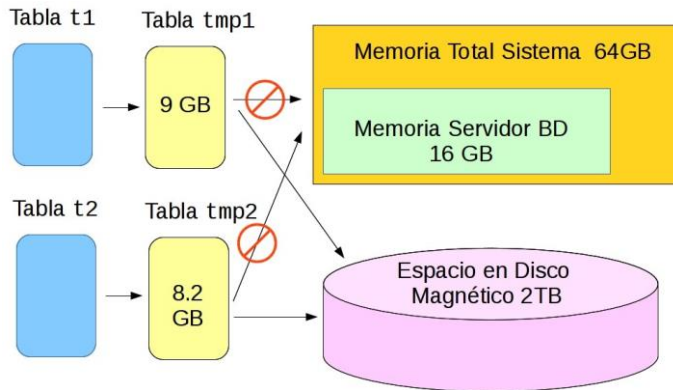


Figura 1: Operación **JOIN** entre dos tablas en BD.

Tablas en memoria

- ▶ Puesto que la tabla total a crear excede la memoria del servidor asignada a la BD, tenemos que escribir los archivos temporales en el espacio temporal de disco.
- ▶ Debido a la velocidad de acceso y actualización de datos en discos magnéticos, esta operación será varios órdenes de magnitud más lenta que su variante en memoria.
- ▶ Si realmente no tenemos espacio en memoria del sistema, no hay otra opción (salvo cambiar de tecnología de almacenamiento).
- ▶ No obstante, **esto no ocurre en muchos casos**, y a pesar de ellos los clientes de la BD siguen lanzando consultas contra disco.

Tablas en memoria

- ▶ Ejemplo: engine `MEMORY` en MySQL (también en MariaDB, Percona Server y otros servidores que reemplazan MySQL).
- ▶ Se trata de un tipo especial de motor que crea las tablas en memoria del sistema.
- ▶ Siempre que se garantice que las consultas no exceden la memoria asignada al servidor de BD, podemos acelerar su ejecución en varios órdenes de magnitud.
- ▶ Por lo tanto, conviene recordar siempre esta técnica, siempre que el cliente tenga los permisos adecuados para crear este tipo de tablas en la BD.
- ▶ Una vez finalizado el proceso, se puede guardar el resultado y descartar las tablas temporales para liberar de nuevo memoria del sistema.

8.2 Apache Spark

Origen de Spark

► AMPLab Berkeley.

- Insatisfechos con Hadoop (MapReduce).
- Mejor soporte para procesamiento de datos iterativo.
- Soporte para procesamiento de datos en memoria.
- Mejora del procesamiento de datos *streaming*.
- Permitir análisis interactivo.

► Empresas colaboradoras: Databricks, Yahoo!, Intel, etc.

► Compañías que integran Spark en sus productos y servicios: Cloudera, Hortonworks, MapR, Amazon EC2, Microsoft Azure, etc.

► Proyecto más activo de Apache, ~ 500 sistemas en producción.

Escenario: análisis de big data

► Analytics in the lab.

- Análisis exploratorio de datos (interactivo).
- Múltiples pruebas (algoritmos, modelos, parámetros, etc.).
- Ajuste fino de las mejores alternativas.
- Generación de informes preliminares.

► Analytics in production.

- Servicios y productos basados en datos.
- Alto nivel de exigencia (múltiples usuarios reales).
- Sistemas en producción (delicados, procesos controlados de despliegue y actualización).
- Monitorización del rendimiento.

Problemas a resolver

- ▶ **Memoria compartida vs. memoria distribuida.**
 - ▶ Permitir escalado de operaciones en miles de nodos sin perder eficiencia.
- ▶ Único paradigma que existía previamente: MapReduce.
 - ▶ Poco flexible (e.g. procesos iterativos).
 - ▶ Código demasiado complejo.
 - ▶ Muy lento (muchas operaciones intermedias con E/S en disco).
- ▶ Mejor tolerancia a fallos, transparente.
- ▶ Gestión más flexible de recursos.

Mejoras en Spark

- ▶ Aumenta **rendimiento en cálculos iterativos**.
 - ▶ Podemos almacenar resultados intermedios en memoria (caché).
 - ▶ Evita constante trasiego de datos al disco.
- ▶ Mayor **variedad de operaciones** para transformar y analizar datos.
- ▶ Análisis y control del flujo de operaciones (*lineage*).
 - ▶ **Directed Acyclic Graphs**.
- ▶ Mantiene integración con Hadoop.
 - ▶ Input: Avro, Parquet, S3, CSV etc.
 - ▶ NoSQL: Cassandra, HBase...
 - ▶ Spark Streaming: Kafka, Flume...
 - ▶ Spark SQL: Hive metastore...
 - ▶ Gestión de recursos: YARN...

Otras mejoras en Spark

- ▶ **Patrones de programación** genéricos.
 - ▶ Mismo motor funciona en múltiples plataformas: multi-core o cluster.
 - ▶ Mismas abstracciones de datos (RDDs, DataFrames) optimizadas en múltiples lenguajes: Scala, Python, R...
- ▶ **Lazyevaluation** del grafo de operaciones (*lineage*).
 - ▶ El grafo solo se evalúa cuando se ejecuta una operación que produce un resultado final, no para transformaciones intermedias.
- ▶ Creación ligera de procesos (no consume muchos recursos).
- ▶ Reparto de datos entre nodos más eficiente.
 - ▶ Podremos evitar excesivo intercambio de datos entre nodos al realizar ciertas operaciones.

Hadoop vs. Spark

	Hadoop	Spark
Paradigma	Batch (MapReduce)	Batch, streaming, interactivo
Operaciones	Map, Reduce	Más de 30 operaciones
Lenguajes	Java	Scala (nativo), Java, Python, R, SQL
Almacenamiento	Disco	Memoria, disco

Tabla 1: Comparación de las principales características de Hadoop y Spark.

Entornos de programación en Spark

- ▶ Spark soporta diferentes lenguajes de programación (aunque no se pueden combinar en un mismo archivo).
- ▶ Scala (lenguaje nativo de desarrollo de Spark).
- ▶ Java (opción natural, base para Scala).
- ▶ SQL (opción muy popular).
- ▶ Python (**PySpark**, más utilizado en porcentaje de usuarios, junto con Scala).
- ▶ R (usando `sparklyr`, paquete de RStudio).
- ▶ Con la arquitectura actual, es más fácil poder integrar nuevos lenguajes de programación en el futuro.
 - ▶ ¿Futuro soporte para Julia en Spark?
 - ▶ <https://juliacomputing.com/blog/2020/06/julia-spark/>.

Inconvenientes y advertencias

- ▶ Es un *framework* **muy dinámico**, sufre cambios importantes y drásticos con frecuencia.
 - ▶ [Página novedades Apache Spark](#) (fechas *releases*).
 - ▶ Spark 1.0.0 (30-05-2014) → Spark 2.0.0 (26-07-2016) → Spark 3.0.0 (18-06-2020).
 - ▶ Suele existir un compromiso de compatibilidad hacia atrás (ej. v2.4.7 lanzada el 12-09-2020).
 - ▶ Sin embargo, extraer toda la potencia de nuevas funcionalidades nos obliga a actualizar nuestro código.
- ▶ Muchas funciones disponibles **solo en Scala**. Es la mejor opción para optimizar rendimiento en Spark.
- ▶ **Integración con otros lenguajes y herramientas** requiere bastante esfuerzo de configuración, desarrollo y mantenimiento.
 - ▶ Spark + R → `sparklyr` puede no integrar todo el catálogo de operaciones en versiones recientes de Spark. Depende de las que estén soportadas por este paquete.
 - ▶ Integración con H2O (framework ML) → [H2O Sparkling Water](#). Precisa instalación de dependencias en todos los nodos del cluster.

8.3 El stack tecnológico de Spark

Visión simplificada

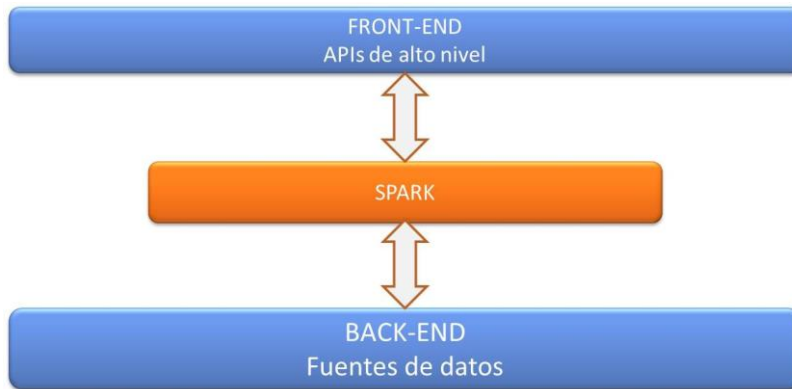


Figura 2: Spark actúa como motor de procesamiento de datos entre las APIs de programación a alto nivel (usuario) y las fuentes de datos.

Visión detallada

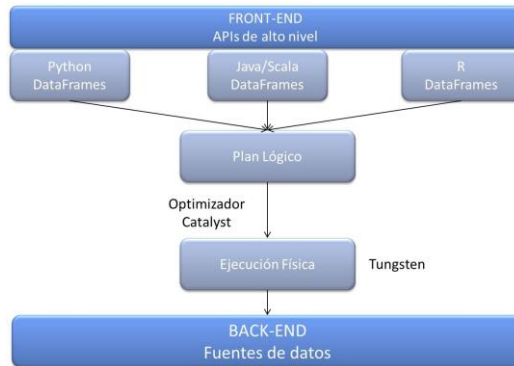


Figura 3: El contrato de Spark garantiza que, sea cual sea el lenguaje de programación utilizado, si usamos las abstracciones para procesamiento de datos estructurados el rendimiento será uniforme.

Conectores para Spark

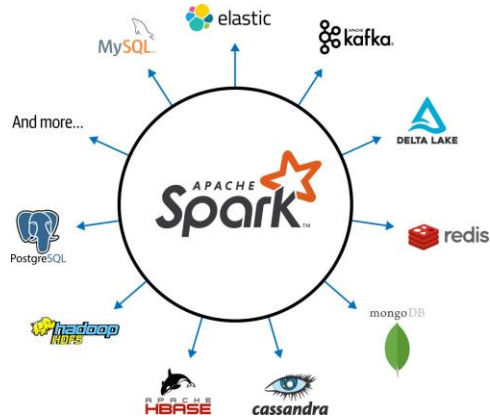


Figura 4: Ecosistema de conectores entre Apache Spark y servicios populares. Figura 1-2 de [1].

Componentes

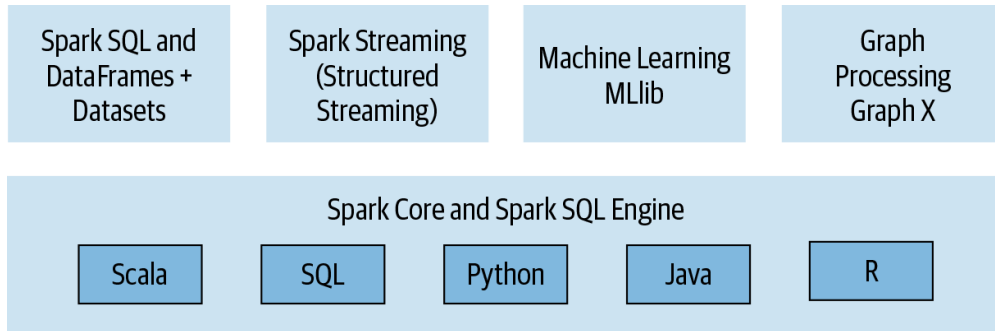


Figura 5: Componentes y API stack de Spark. Figura 1-3 de [1].

Spark Core

- ▶ Núcleo (**engine**) de procesamiento de datos para todo el stack.
- ▶ **Reemplaza a Hadoop** como “data operating system” del stack.
- ▶ Incluye las funciones básicas para procesamiento distribuido de datos.
- ▶ Abstracción de datos básica:
 - ▶ **Resilient Distributed Datasets (RDDs)**.
- ▶ Interfaz común para conectar con múltiples **front-end APIs**.
- ▶ **Gestión optimizada** de tareas y memoria (a nivel lógico y físico)
- ▶ **Recuperación de fallos** en caso de fallo en los nodos.
- ▶ Múltiples interfaces para interactuar con **sistemas back-end** (fuentes de datos).

Spark Core: RDDs

- ▶ Principal abstracción (bajo nivel), que permite distribuir conjuntos de datos masivos entre los nodos de un cluster, particionando el conjunto original en bloques enviados a cada nodo de computación.
- ▶ Los datos se replican para soportar recuperación de fallos.
- ▶ Los RDDs son **inmutables** una vez creados. Esto simplifica la forma en que definimos el gráfico de transformaciones que aplicamos a nuestros datos.
- ▶ Es el nivel más bajo de la torre Spark en procesamiento de datos.
- ▶ Contro manual fino de operaciones, gestión de memoria, define cómo se particionan y distribuyen los datos entre los nodos.
- ▶ No se beneficia de optimización automática de operaciones (a nivel lógico y físico) puesto que ni el esquema ni los tipos de datos de cada campo están fijados inicialmente.

Spark SQL

- ▶ Iniciada como una API para ejecutar sentencias SQL sobre datos estructurados en Spark.
- ▶ Soporta un subconjunto (amplio) del estándar SQL y Hive Query Language.
- ▶ Soporta tablas Hive y diferentes formatos de archivos: Parquet, JSON; CSV o XML (con librerías adicionales).
- ▶ Permite mezclar consultas SQL con operaciones sobre datasets programadas en Spark (Scala, Java, Python or R).
- ▶ Actualmente, es el núcleo de implementación de `DataFrame` la abstracción, principal para **datasets estructurados**.

DataFrame

- ▶ Abstracción introducida en Spark 1.3.
- ▶ Desde Spark 1.6 se convierte en **abstracción de datos principal** para toda operación con **structured datasets**, en cualquier front-end.
- ▶ Optimización a nivel lógico: (**Catalyst**).
- ▶ Optimización de operaciones a bajo nivel (nivel físico), next to hardware: (**Tungsten**).
- ▶ En teoría, debe garantizar un rendimiento uniforme en cualquier lenguaje/entorno de programación soportado en Spark.

Dataset

- ▶ Nueva generación de abstracción para operar con datos estructurados en Spark.
- ▶ RDDs no tienen información sobre el tipo de dato de cada campo, y no pueden aprovecharla para optimizar operaciones.
- ▶ DataFrames proporcionan optimización de operaciones para la interfaz SQL. Sin embargo, no pueden detectar tipos de datos de columnas en tiempo de compilación.
- ▶ Datasets proporcionan una interfaz **type-safe** de **programación orientada a objetos** que combina los beneficios de ambos mundos, incluyendo optimización lógica y física.
- ▶ Puesto que tenemos información detallada sobre los tipos de cada columna, podemos aplicar operaciones especiales para ahorrar memoria y acelerar el procesado.

GraphFrame

- ▶ Soporte para procesado de grafos, siguiendo las características incluidas en GraphX.
- ▶ Creado sobre DataFrames, para que:
 - ▶ Soporte los principales lenguajes en Spark (Scala, Java, Python).
 - ▶ Podemos lanzar consultas sobre datos de grafos directamente, usando la misma aproximación que en Spark SQL y DataFrames (sin lenguajes especiales).
 - ▶ Lectura y escritura de datos en formatos estándar (Parquet, JSON or CSV).

Spark Streaming

- ▶ Análisis de datos *streaming*.
- ▶ Admite diferentes fuentes de datos, como Kafka, Redshift, etc.
- ▶ Dos posibles modalidades:
 - ▶ Programación a bajo nivel: DStream.
 - ▶ Structured Streaming: DataFrames dinámicas, que siguen recibiendo datos y actualizan los resultados.
- ▶ Operaciones similares a Spark core (RDDs), ya que sigue el modelo *micro-batch* (no es estrictamente un motor streaming).
- ▶ Misma tolerancia a fallos, fiabilidad y escalabilidad que en Spark Core.

MLlib

- ▶ Biblioteca de aprendizaje máquina (algoritmos y procesos) en Spark.
- ▶ Algunas aplicaciones para regresión, clasificación, clustering, filtrado colaborativo, escalado multidimensional, optimización, etc.
- ▶ Todavía muy lejos del amplio catálogo de algoritmos en R (~ 10.000).
- ▶ Actualmente soporta trabajo con RDDs y DataFrames para procesamiento distribuido.
 - ▶ Pero no soporta `dplyr`, `tidyr`, `ff`, etc. de R, directamente.
 - ▶ Toda interacción con clases y paquetes R debe pasar por `sparklyr`.

<https://databricks.com/blog/2015/01/28/introducing-streaming-k-means-in-spark-1-2.html>

GraphX

- ▶ Biblioteca para representar grafos e implementar computación paralela en grafos.
- ▶ Extiende la abstracción RDD introduciendo **Graph**: un multigrafo dirigido en el que se pueden asignar propiedades a cada vértice o enlace.
- ▶ Define un conjunto de operaciones primitivas para trabajo con grafos (e.g. encontrar subgrupos).
- ▶ Incluye un variante optimizada de la API Pregel de Google.
- ▶ Soporte completo para el lenguaje Scala, pero la mayoría de los métodos ya están portados a Python desde Spark 2.0.0.

8.4 Procesamiento de Datos con SSDs

Tecnología de estado sólido

- ▶ Los Discos o Unidades de Estado Sólido (Solid-State Drives) son una alternativa tecnológica a los discos magnéticos convencionales para almacenamiento persistente de datos.
- ▶ En sus inicios, era una tecnología joven, por lo que aún existían dudas sobre su fiabilidad a medio y largo plazo.
- ▶ Sin embargo, en la actualidad ya se cuenta con suficiente trayectoria de uso como para garantizar su aplicación en análisis de datos.
- ▶ Puede utilizarse conjuntamente con discos magnéticos convencionales (como si proporcionase una caché interna al dispositivo). Estos discos se denominan unidades híbridas o SSHD.

Ej: SSHD Western Digital



Figura 6: Esta unidad híbrida de Western Digital combina un disco magnético de 1TB con una unidad SSD interna de 8GB con tecnología NAND.

Tecnología de estado sólido

- ▶ Los discos SSD puros ofrecen un excelente rendimiento con velocidades de acceso varios órdenes de magnitud más rápidas que los discos magnéticos convencionales.
- ▶ Inconvenientes:
 - ▶ Hasta hace poco tiempo, tenían mucha **menos capacidad** de almacenamiento de datos que la tecnología magnética tradicional. Sin embargo, ya podemos encontrar unidades SSD de varios TB.
 - ▶ La **lectura es mucho más eficiente** que las operaciones de escritura. Esto se debe a que las células de estado sólido se estropean al escribir repetidamente en ellas. Por eso, los controladores (hardware) de estas unidades intentan repartir las escrituras por todo el dispositivo, mediante el *wear leveling*, lo que provoca un fenómeno llamado *write amplification*.

Write amplification

- ▶ Cuando escribimos por primera vez en una celda de un SSD no hay problema, se ponen los datos en celdas de memoria consecutivas.
- ▶ Sin embargo, cuando actualizamos un dato, en lugar de volver a escribir sobre la celda se toman N posiciones consecutivas de datos y se escriben en una nueva ubicación (incluido el nuevo dato), descartando el puntero a los datos anteriores.
- ▶ Esto se hace para intentar distribuir la carga de escritura por todo el dispositivo, evitando sobrecargar en exceso algunas celdas con muchas escrituras (*write leveling* o balanceo de escritura).
- ▶ El grave inconveniente es que se escriben más datos que los indicados en la instrucción original: el dato nuevo más todos los datos adyacentes que se mueven a la nueva posición → *write amplification*.

SSD and write amplification (MIT Lincoln Labs).

Procesado de datos con SSDs

- ▶ Podemos aprovechar arrays de SSDs (en alguna configuración RAID) para ampliar capacidad de almacenamiento y a la vez ganar velocidad de transferencia de datos.
- ▶ **Cuidado:** es muy importante utilizar solo **controladoras RAID** (hardware) **pensadas específicamente para SSDs**.
 - ▶ Utilizar una controladora de discos convencionales puede perjudicar seriamente el rendimiento del dispositivo SSD. Por ejemplo, las controladoras RAID convencionales realizan una primera pasada de configuración escribiendo datos en toda la unidad a modo de inicialización. Esto baja el rendimiento de escritura de la unidad SSD (por los motivos antes descritos del *write amplification*).

Procesado de datos con SSDs

- ▶ En sistemas operativos que permiten configurar el espacio de **memoria virtual en disco** (swap), podemos configurar una zona de intercambio muy amplia en un disco SSD o un RAID de discos SSD.
- ▶ Con discos magnéticos, al acabarse la memoria principal comienza el *swapping* en la zona de memoria virtual; al tener mucha menor velocidad de acceso el sistema se ralentiza enormemente.
- ▶ Sin embargo, con un SSD la velocidad de acceso no es tan inferior → Podemos expandir la memoria del sistema enormemente, de forma mucho más económica que comprando RAM.
- ▶ Sería una buena forma de montar un equipo para análisis de big data en memoria con un método *low cost*.

References I



[Wenig, 2020] Brooke Wenig and Jules S. Damji and Tathagata Das and Denny Lee.
Learning Spark, 2nd Ed.
O'Reilly Media, Inc., julio, 2020.



[Zecevic, 2016] Petar Zečević and Marko Bonaći.
Spark in Action
Manning Publishing, noviembre, 2016.



[Ryza, 2015] Ryza, S., Laserson, U., Owen, S. & Wills, J.
Advanced Analytics with Spark: Patterns for Learning from Data at Scale.
O'Reilly Media, Inc., abril, 2015.

TokuDB y Aerospike

Felipe Ortega

Análisis de Big Data.

Máster Universitario Ingeniería de Sistemas de Decisión.

Madrid

22 de marzo de 2018



Introducción

- ▶ Para concluir la asignatura, repasamos dos casos paradigmáticos aunque normalmente no muy conocidos de productos que combinan técnicas innovadoras para análisis de *big data* con los últimos avances tecnológicos.
- ▶ **TokuDB** surgió como un motor de base de datos alternativo para MySQL y sus equivalentes (e.g. MariaDB), creado por la empresa Tokutek. Su revolucionaria tecnología permite realizar **consultas sobre datos** que se almacenan **comprimidos en disco**, reduciendo así enormemente las necesidades de espacio en el sistema.
- ▶ **Aerospike** es una base de datos **híbrida**, que encarna la acepción *Not only SQL* del acrónimo NoSQL. Incluye tanto un motor relacional como un motor que soporta datos semi-estructurados. Además, es capaz de explotar toda la rapidez de la tecnología SSD trabajando **directamente con datos en bruto** guardados en la unidad, sin pasar a través del sistema de ficheros del sistema operativo.

TokuDB: gestión de datos comprimidos

Introducción

- ▶ Se trata de un motor de base de datos para MySQL o equivalentes (Percona Server, MariaDB, etc.).
- ▶ Fue creado originalmente por la empresa Tokutek, que también generó un motor con características parecidas para base de datos documental (MongoDB), llamado TokuMX.
- ▶ Tokutek fue posteriormente adquirida por Percona, que ahora incluye TokuDB por defecto en su producto, Percona Server.
- ▶ MariaDB, aunque no trae este motor activado por defecto tras la instalación, permite activarlo de forma muy sencilla.

Enabling TokuDB in MariaDB.

Listado motores MariaDB (clon de MySQL)

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 45
Server version: 10.0.24-MariaDB-1-trusty mariadb.org binary distribution

Copyright (c) 2000, 2016, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW ENGINES;
```

Engine	Support	Comment	Transactions	XA	Savepoints
MEMORY	YES	Hash based, stored in memory, useful for temporary tables	NO	NO	NO
CSV	YES	CSV storage engine	NO	NO	NO
PERFORMANCE_SCHEMA	YES	Performance Schema	NO	NO	NO
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)	NO	NO	NO
MyISAM	YES	MyISAM storage engine	NO	NO	NO
TokuDB	YES	Percona TokuDB Storage Engine with Fractal Tree(tm) Technology	YES	YES	YES
MRG MyISAM	YES	Collection of identical MyISAM tables	NO	NO	NO
ARCHIVE	YES	Archive storage engine	NO	NO	NO
FEDERATED	YES	FederatedX pluggable storage engine	YES	NO	YES
InnoDB	DEFAULT	Percona-XtraDB, Supports transactions, row-level locking, and foreign keys	YES	YES	YES
Aria	YES	Crash-safe tables with MyISAM heritage	NO	NO	NO

```
11 rows in set (0.00 sec)
```

Figura 1: Lista de motores habilitados en MariaDB, que incluye TokuDB

Características

- ▶ La principal característica que distingue a TokuDB del resto de motores para MySQL es la posibilidad de **almacenar los datos comprimidos en disco**, posibilitando la realización de consultas sobre dichos datos comprimidos.
- ▶ Actualmente soporta al menos los algoritmos de compresión `zlib` (archivos GZIP, opción por defecto), así como `quicklz` y `lzma`, que proporciona la mayor capacidad de compresión de espacio en disco, a costa de una menor velocidad de ejecución de consultas.
- ▶ Como resulta evidente, el ahorro de espacio en disco puede suponer grandes ventajas en el análisis de big data.

Ventajas compresión de datos

- ▶ Si podemos almacenar la misma cantidad de datos en menos espacio en disco, podemos por ejemplo usar unidades SSD, de menor capacidad pero mucho mayor rendimiento de E/S, para acelerar todavía más las consultas.
- ▶ Se necesita menor inversión en hardware o menor gasto en recursos computacionales cloud para poner en marcha sistemas de este tipo que puedan procesar la misma cantidad de datos.
- ▶ El innovador sistema de almacenamiento e indexación de información de TokuDB permite mantener un rendimiento mucho mayor (como máximo 20 veces mayor) que otros motores como InnoDB.

Fractal trees

- ▶ El sistema de indexación (patentado por Tokutek) que emplea TokuDB se denomina **Fractal Tree**.
- ▶ Este sistema de indexación de información en disco proporciona varias ventajas:
 - ▶ Los árboles tradicionales (BTrees) trabajan por páginas (bloques) de datos. Aunque deseemos leer o escribir menos datos que el tamaño de página, el mínimo bloque de datos con el que podemos trabajar es del tamaño de la página.
 - ▶ Por el contrario, un Fractal Tree usa *buffers* a diferentes niveles del árbol para acelerar las inserciones. Cuando un buffer de nivel superior se llena, se “empuja” los datos al buffer del nivel inferior, hasta que en última instancia alcanzan las hojas del árbol.



Inserción datos B-Tree (tradicional)

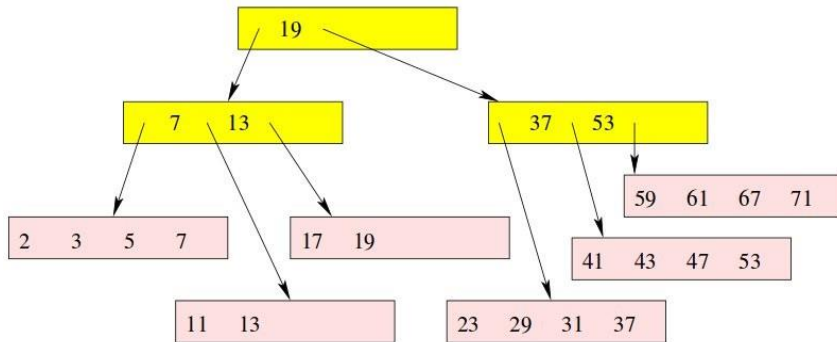


Figura 2: Ejemplo de indexación de datos con B-Tree. Los datos se almacenan en bloques en las hojas (abajo), organizados según el valor de los datos en los nodos raíz (parte superior).

Inserción de datos Fractal Tree (TokuDB)

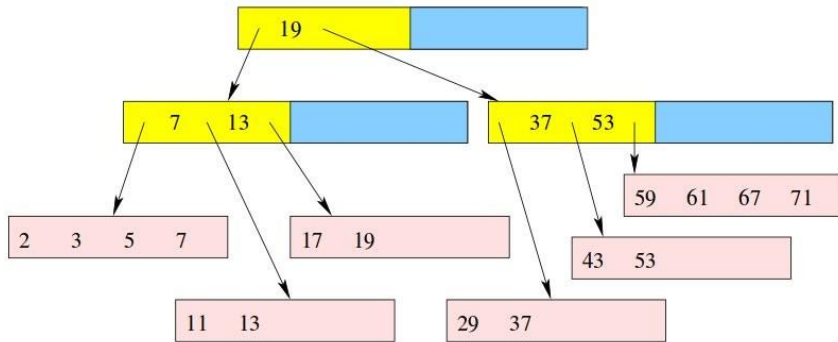


Figura 3: En un Fractal Tree, los nodos superiores tienen un buffer de inserción de nuevos datos, por lo que no hay que recorrer todo el camino hasta las hojas para introducir nueva información.

Inserción de datos Fractal Tree (TokuDB)

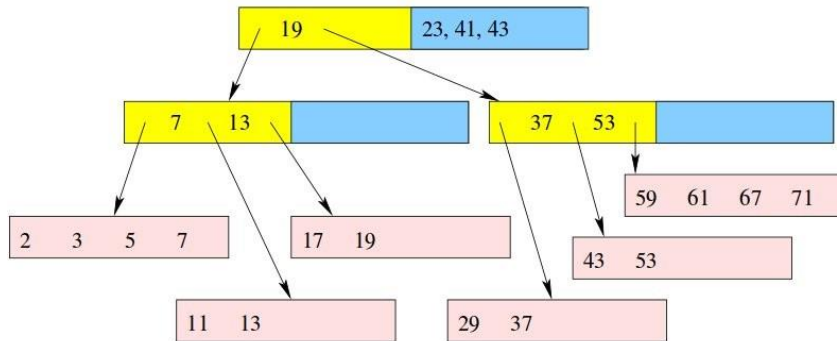


Figura 4: Al insertar nuevos datos, se quedan en el buffer de entrada del nivel raíz (marcados en azul celeste).

Inserción de datos Fractal Tree (TokuDB)

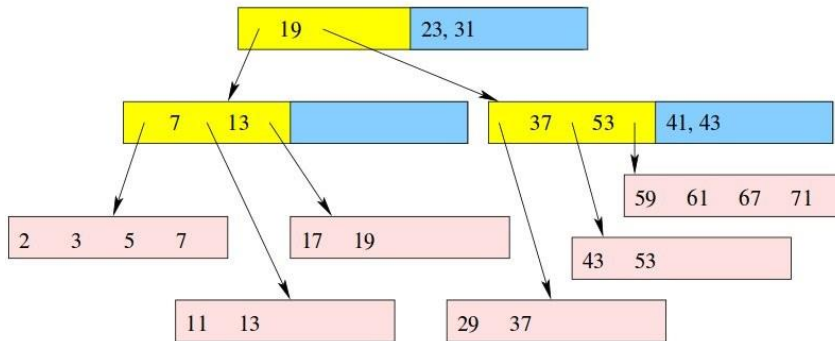


Figura 5: Conforme se insertan más datos, se llena el buffer superior y comienzan a pasarse datos a los buffers de niveles inferiores, hasta que en algún momento llegan a las hojas del árbol (bloques en rosa). [\[Enlace white paper\]](#)

Aerospike: BD híbrida para tecnología SSD

Introducción

- ▶ En 2012 la compañía Aerospike lanza la base de datos del mismo nombre, sucesora de un producto anterior llamado Citrusleaf.
- ▶ Aerospike incluye una idea revolucionaria para la época: **optimizar** al máximo el rendimiento de **unidades SSD, saltándose la capa de gestión de ficheros** ofrecida por el sistema operativo.
- ▶ De este modo, los datos se almacenan “en crudo” en el dispositivo SSD, y se pueden leer o escribir sin más capas intermedias que ralenticen este proceso.
- ▶ Puesto que la tecnología SSD todavía era relativamente joven, una estrategia tan agresiva generaba ciertas dudas, aunque a la postre se ha confirmado como un verdadero éxito, visionario e innovador.

Características

- ▶ Está escrita directamente en lenguaje C, para un control muy fino de la gestión del almacenamiento de datos en disco y memoria.
- ▶ Su arquitectura cuenta con 3 capas principales:
 - ▶ Una capa de bajo nivel, que gestiona el almacenamiento de datos en unidades SSD mediante un formato nativo, sin sistema de ficheros tradicionales.
 - ▶ Una capa autogestionada de distribución de datos entre los nodos del clúster.
 - ▶ Una capa cliente, que proporciona APIs de acceso en varios lenguajes y permite recuperar los datos del cluster de máquinas que funciona con Aerospike.
- ▶ En el *whitepaper* incluido en la sección *Referencias* de este tema en Aula Virtual, se pueden ampliar más detalles.

Características

- ▶ El principal objetivo de Aerospike es acelerar al máximo la inserción y el acceso a la información. Está pensado para sistemas transaccionales que precisen tiempos de respuesta realmente rápidos (e.g. metabuscadores de Internet).
- ▶ Usa una doble estrategia de indexación de datos:
 - ▶ Los índices se guardan en memoria RAM, utilizando cachés de acceso a los índices más utilizados.
 - ▶ Los bloques de datos se guardan en unidades SSD, aunque también admite combinación con discos convencionales.
- ▶ Adicionalmente, emplea una estrategia para anticipar y evitar la fragmentación de datos (en la medida en que sea posible), intentando guardar la información en bloques contiguos (memoria o disco). En particular, esto reduce el fenómeno de *write amplification* que ya conocemos en los SSDs.

Gestión de datos

- ▶ Dos componentes adicionales de Aerospike se encargan de gestionar los espacios de almacenamiento y que los datos se guarden en posiciones contiguas.
- ▶ **Defragmentor** es un módulo que monitoriza el acceso a bloques de datos en disco, y reclama aquellos espacios con datos menos utilizados (principio de localidad).
- ▶ **Evictor** monitoriza el uso de índices en memoria RAM del sistema. Cuando los índices apuntan a datos cuya validez ha expirado, o cuando se sobrepasa un límite indicado por el administrador del sistema, el componente libera índices de la memoria RAM.
- ▶ Los dos trabajan de forma solidaria como sistema avanzado de “recolección de basura”, garantizando un rendimiento óptimo del sistema en la mayoría de situaciones.

©2025 Autores Felipe Ortega Soto, Alberto Fernández Isabel

Algunos derechos reservados

Este documento se distribuye bajo la licencia

“Atribución 4.0 Internacional” de Creative Commons, disponible en

<https://creativecommons.org/licenses/by/4.0/deed.es>