COMPUTACIÓN DE ALTA PERFORMANCE

Curso 2024

Sergio Nesmachnow (sergion@fing.edu.uy)

Centro de Cálculo

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

TEMA 10: CLOUD COMPUTING

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Cloud Computing

CLOUD COMPUTING: CONCEPTOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cloud Computing es un término genérico que describe un paradigma de computación distribuida sobre Internet.
- Extiende el concepto de *utility computing*, que se basa en el suministro de recursos computacionales (procesamiento y almacenamiento) como un servicio a demanda, similar a otros productos públicos tradicionales (electricidad, agua, gas natural, etc).
- El modelo que cuenta con la ventaja de tener un costo nulo (o muy bajo) de adquisición de hardware; en cambio, los recursos computacionales son esencialmente alquilados y tienen costo.
- Clientes que realizan procesamiento de datos a gran escala o que están frente a un pico de demanda también pueden evitar los atrasos que resultarían de adquirir y ensamblar físicamente una gran cantidad de computadoras.

CLOUD COMPUTING: una “definición”

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

¿Qué es Cloud computing?

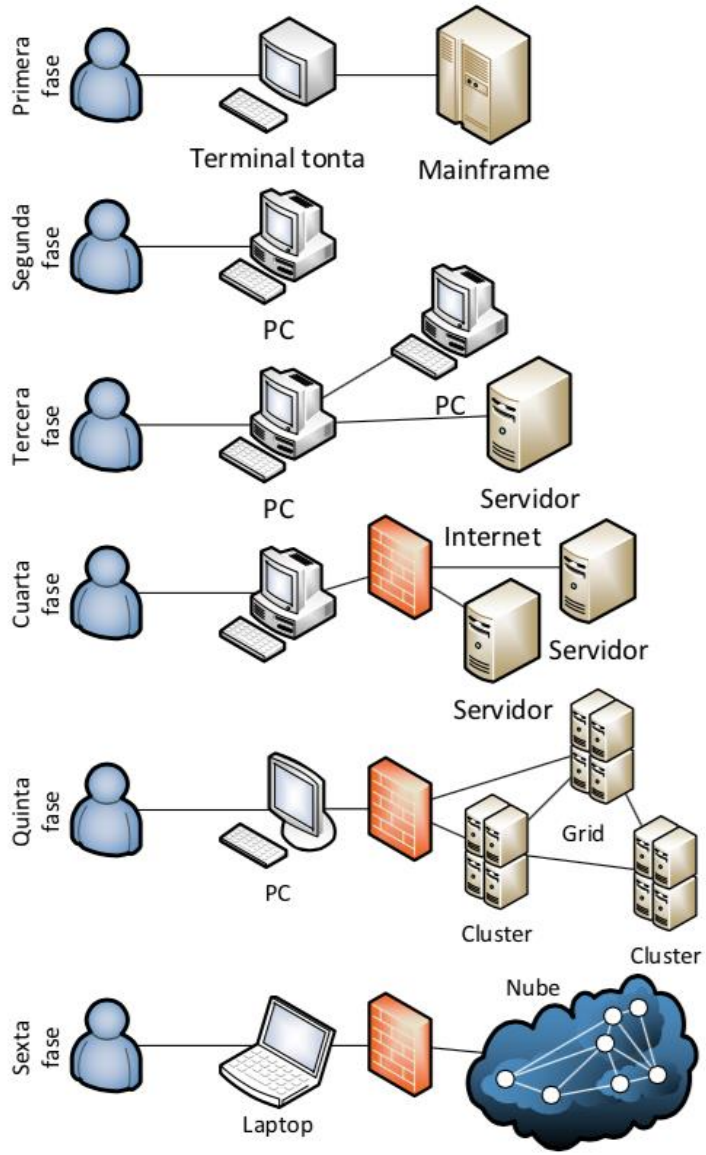
> “Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.”
>
> – National Institute of Standards and Technology, Information Technology Laboratory (2009).

CLOUD COMPUTING: CARACTERÍSTICAS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Existe una colección de recursos de hardware integrados a través de una red, mediante un software específico, y con acceso a través de Internet (este conjunto integrado se denomina *plataforma*).
- Se utiliza Internet para las comunicaciones y para proveer servicios de hardware, software y conectividad (networking) a los clientes.
- Las plataformas ocultan la complejidad y los detalles de la infraestructura subyacente a los usuarios y proveen soporte para el desarrollo y ejecución de aplicaciones mediante simples interfaces gráficas y/o APIs.
- La plataforma provee servicios a demanda, que están siempre disponibles (ubicuos: anywhere, anytime, anyplace).
- Los usuarios pagan por la utilización de los servicios.
- La plataforma puede escalar elásticamente las capacidades y funcionalidades de los servicios provistos.

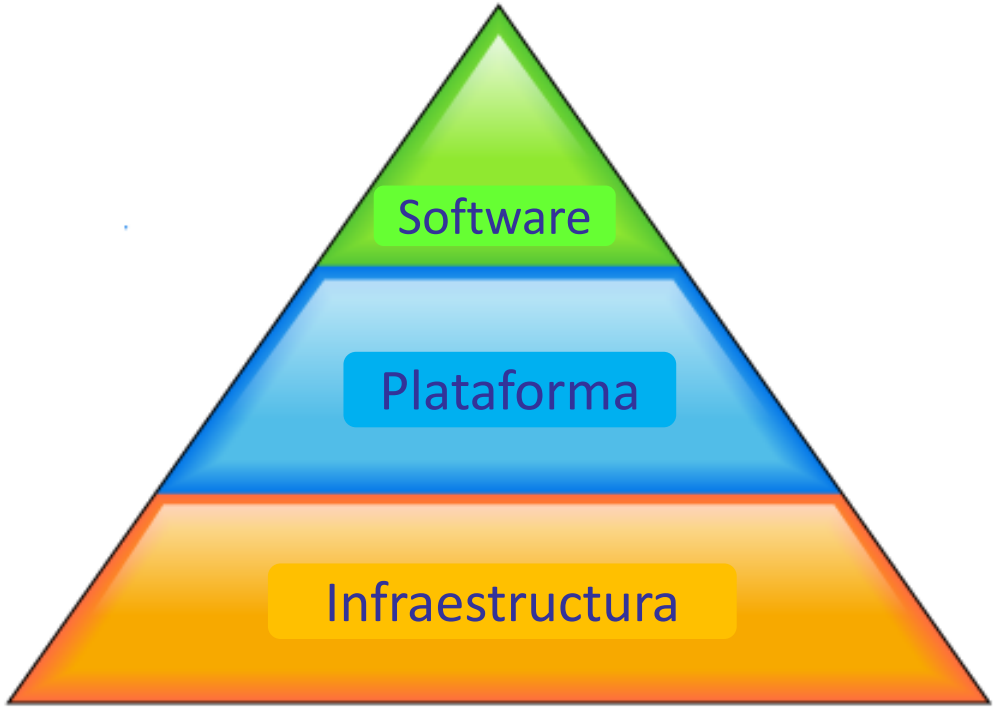
CLOUD COMPUTING: HISTORIA



1. *Primera etapa*: Paradigma de computación centralizada basado en mainframes.
2. *Segunda etapa* : conexión de usuario a PC.
3. *Tercera etapa* : conexiones en red de área local.
4. *Cuarta etapa* : interconexión de computadoras en red a través de internet.
5. *Quinta etapa* : Grid y el uso compartido de procesamiento y cómputo.
6. *Sexta etapa* : Usuarios se conectan a la nube.

CLOUD COMPUTING: CAPAS

- Capas asociadas a los servicios
  - Infraestructura, plataforma y software



Clientes Cloud

Navegadores web, aplicaciones móviles, clientes livianos, emuladores de terminal, etc.

Software

SaaS

E-mail, desktop virtuales, comunicación y redes sociales, juegos, etc.

Plataforma

PaaS

Entorno de ejecución, bases de datos, servidor web, herramientas de desarrollo, etc.

Infraestructura

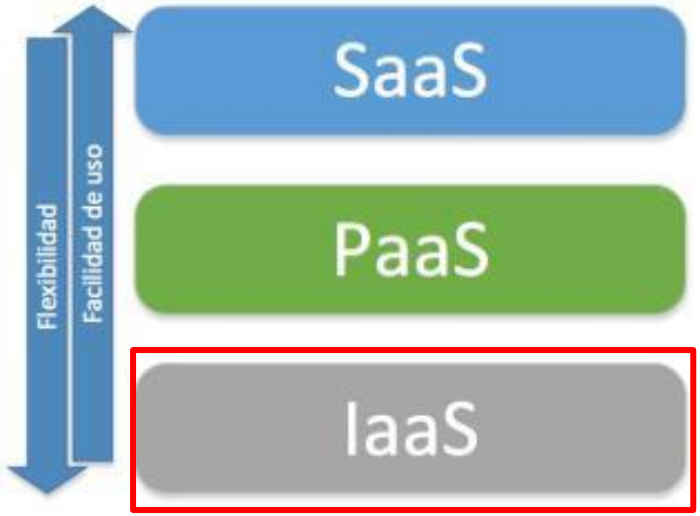
IaaS

Máquinas virtuales, servidores, almacenamiento, balance de cargas, red, etc.

CLOUD COMPUTING: CAPAS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

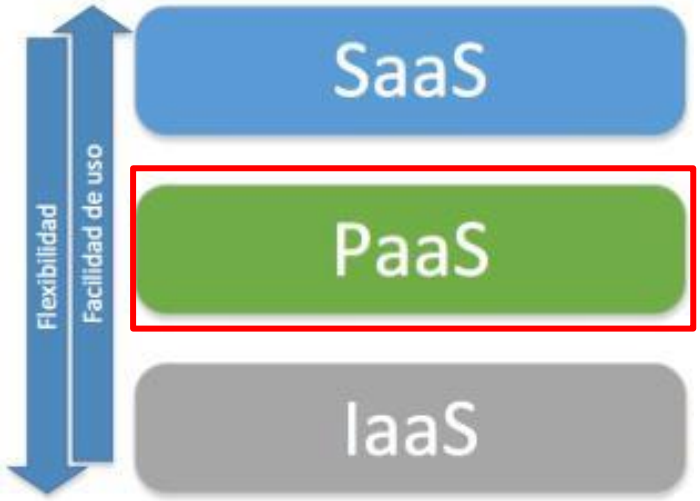
- Infraestructura como servicio (Infrastructure as a Service, IaaS)
  - Capa inferior del cloud, es un medio de entregar almacenamiento y capacidades de cómputo como servicios estandarizados en la red.
  - Servidores, almacenamiento, conexiones, routers, y otros sistemas se concentran (por ejemplo a través de virtualización) para manejar tipos específicos de cargas de trabajo (procesamiento "batch", o aumento de servidor/ almacenamiento durante cargas pico).
  - Ejemplos comerciales de referencia: Amazon Web Services [servicios EC2 (cómputo) y S3 (almacenamiento)]; Azure y Joyent, que ofrecen hardware virtualizado a demanda, altamente escalable para diversas aplicaciones en múltiples lenguajes.



CLOUD COMPUTING: CAPAS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Plataforma como servicio (Platform as a service, PaaS)
  - Encapsula una abstracción de un ambiente de desarrollo y empaqueta una serie de módulos que proporcionan una funcionalidad horizontal (persistencia de datos, autenticación, mensajería, etc.).
  - Ejemplo: un entorno conteniendo sistemas, componentes o APIs preconfiguradas y listas para integrarse sobre una tecnología concreta de desarrollo (servidor web en Linux y ambiente de programación Perl o Ruby).
  - PaaS puede dar servicio a todas las fases del ciclo de desarrollo de software, o especializarse en una particular (administración de contenidos).
  - Ejemplos comerciales: Google App Engine para aplicaciones Google, y Windows Azure para desarrollar aplicaciones en .NET, Java y PHP.
  - Los servicios PaaS otorgan flexibilidad, pero pueden estar restringidos por las capacidades del proveedor.



CLOUD COMPUTING: CAPAS

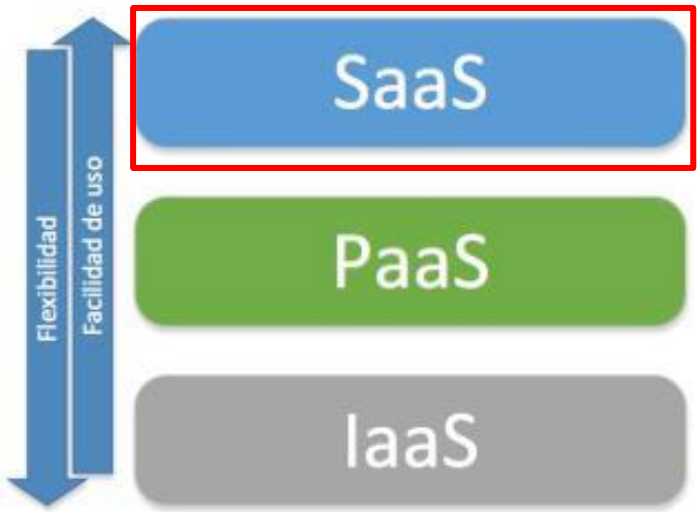
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Plataforma como servicio (Platform as a service, PaaS)
  - Varios tipos de proveedores, pero todos ofrecen hosting, un entorno de desarrollo de aplicaciones y servicios de escalabilidad y mantenimiento.
  - Tipos de servicios PaaS:
    - Desarrollo de adds-on y software personalizado: usando macrolenguajes de personalización de aplicaciones (Lotus Notes, Microsoft Word, etc.)
    - Entornos de desarrollo genéricos y stand-alone: no tienen dependencias técnicas, de licencia, o financieras con aplicaciones SaaS específicas.
    - Entornos de entrega de aplicaciones: no incluyen funcionalidades de desarrollo o testeo, que pueden proveerse fuera de línea. Los servicios en general se enfocan en seguridad y escalabilidad a demanda.
    - Plataforma abierta como servicio: no incluye hosting, pero provee software open source para ejecutar aplicaciones. Ejemplo: AppScale, que permite desplegar aplicaciones desarrolladas en Google App Engine en servidor del usuario. Otras plataformas dan más flexibilidad sobre lenguajes, bases de datos, sistemas operativos, etc.

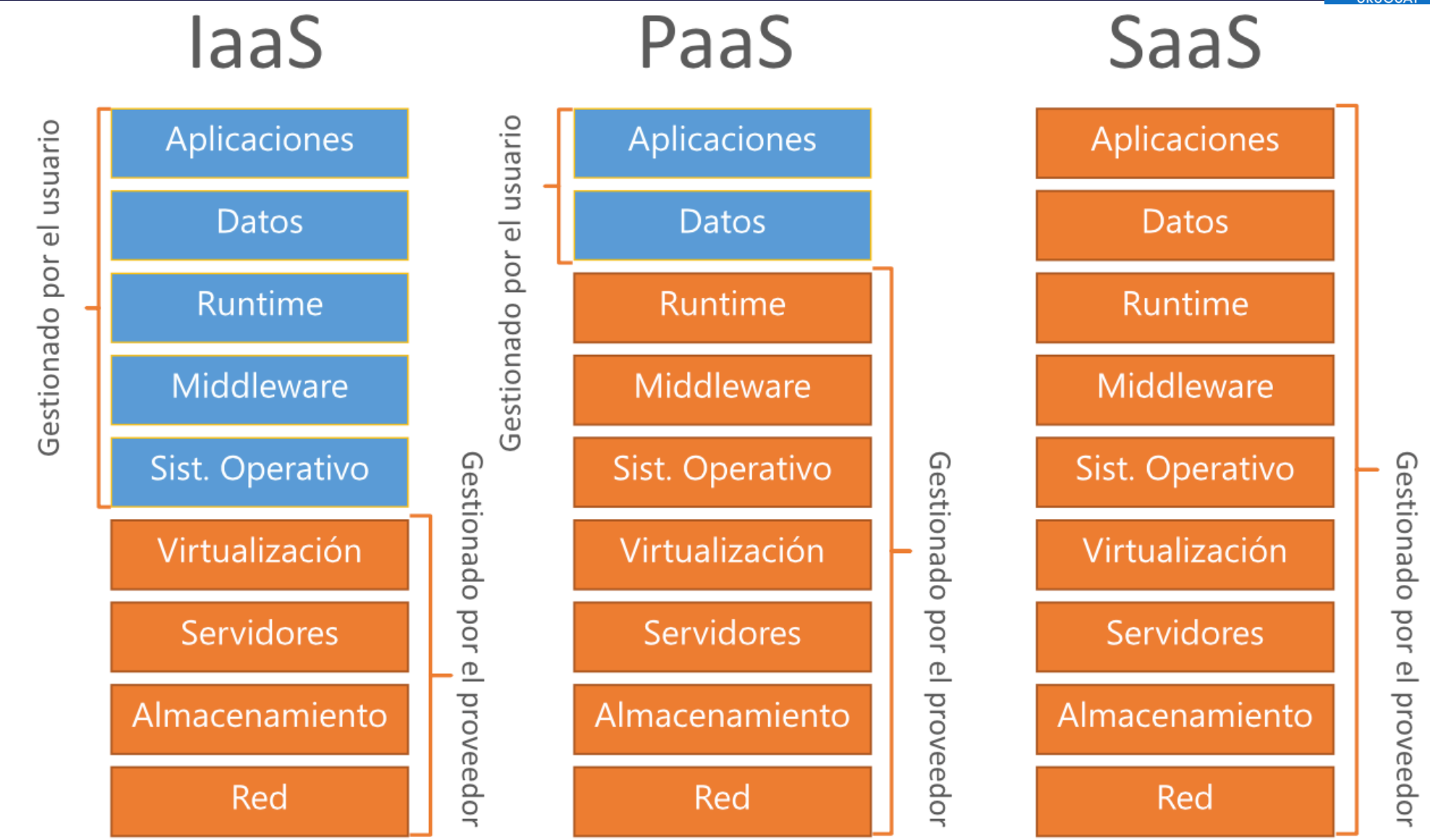
CLOUD COMPUTING: CAPAS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Software como servicio (Software as a Service, SaaS)
  - Aplicaciones completas ofrecidas como servicio a demanda, y en multitenencia: una única instancia del software ejecuta en la infraestructura del proveedor y sirve a múltiples organizaciones de clientes.
  - Ejemplo comercial de referencia: Salesforce.com, uno de los mejores ejemplos de cómputo en cloud durante muchos años, con sus aplicaciones para ventas (Sales Cloud), servicio al cliente (Service Cloud), colaboración empresarial (Chatter).
  - Otros ejemplos: Google Apps que ofrece servicios básicos de negocio (incluyendo e-mail, calendarios, documentos web, etc.), Microsoft Office 365, la plataforma MS Office como servicio, que incluye versiones online de la mayoría de las aplicaciones de la suite ofimática de Microsoft.



CLOUD COMPUTING: RESPONSABILIDADES



CLOUD COMPUTING: BENEFICIOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Integración simple y rápida con otras aplicaciones (tradicionales o cloud), ya sean desarrolladas de manera interna o externa.
- Prestación de servicios con mayores capacidad de adaptación, recuperación de pérdida de datos y reducción de tiempos de inactividad.
- Permite prescindir de la instalación y administración de hardware, requiriendo una menor inversión inicial para un trabajo.
- Implementación rápida y con menos riesgos, incluso al requerir niveles considerables de personalización o integración.
- Actualizaciones automáticas de las aplicaciones, con poca (o ninguna) intervención del usuario.
- Contribuye al uso eficiente de la energía (requerida para el funcionamiento de la infraestructura), ofreciendo grandes reducciones de consumo al compararse con los datacenters tradicionales.

CLOUD COMPUTING: DESVENTAJAS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La centralización de las aplicaciones y el almacenamiento de los datos origina una interdependencia de los proveedores de servicios.
- La disponibilidad de las aplicaciones está sujeta al acceso a Internet.
- Datos "sensibles" residen fuera de sus propietarios; se puede generar un contexto de alta vulnerabilidad para el robo de información.
- La confiabilidad de los servicios depende de la robustez tecnológica y financiera de los proveedores. Servicios altamente especializados podrían tardar meses o años para que sean factibles de ser desplegados en la red.
- Muchas aplicaciones modifican continuamente sus interfaces, por lo cual el aprendizaje y su uso automático por otras aplicaciones es complejo.
- La información debe recorrer diferentes nodos para llegar a su destino; cada uno de ellos es un foco de inseguridad. Si se utilizan protocolos seguros (HTTPS u encriptado), la velocidad de transferencia disminuye debido a la sobrecarga.

CLOUD COMPUTING: DESVENTAJAS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La escalabilidad a largo plazo es una incógnita. A medida que más usuarios empiecen a compartir la infraestructura de cloud, la sobrecarga en los servidores de los proveedores aumentará, y si la empresa no posee un esquema de crecimiento óptimo puede llevar a degradaciones en el servicio.
- Dependencia de proveedores para almacenamiento de datos y ejecución de aplicaciones.
  - Richard Stallman (2008): “cloud computing is simply a trap aimed at forcing more people to buy into locked, proprietary systems that would cost them more and more over time.” (“cloud computing es simplemente una trampa destinada a obligar a más gente a adquirir sistemas propietarios, bloqueados, que les costarán más y más conforme pase el tiempo.”)

CLOUD COMPUTING y HPC

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cloud computing no fue concebido para brindar soporte a aplicaciones científicas que utilicen HPC.
- El paradigma de cloud computing se enfoca principalmente en soportar aplicaciones y servicios de propósito general.
- La infraestructura de hardware y la plataforma están optimizadas para aplicaciones basadas en transacciones de corta duración, e.g. peticiones de aplicaciones web.
- Este tipo de aplicaciones no requieren comunicación entre procesos.
- No se manejan servidores dedicados, los recursos de cómputo son instancias de cómputo virtualizadas.

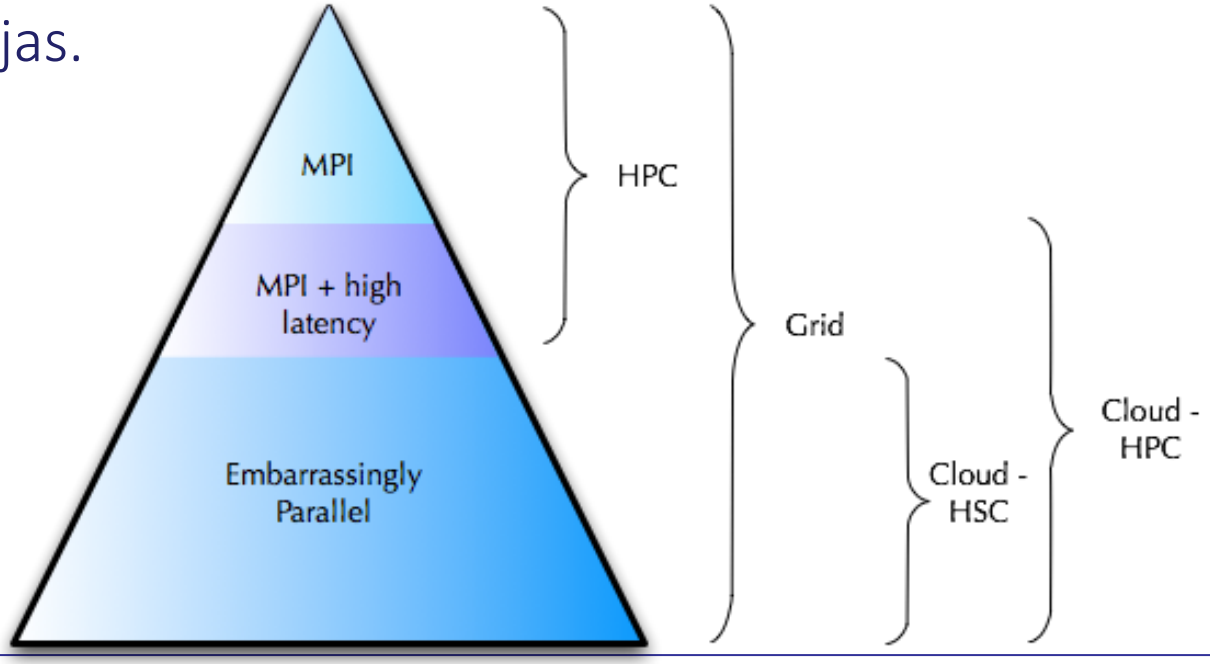
CLOUD COMPUTING y HPC

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Aunque cloud computing no es ni ha sido concebida para dar soporte a aplicaciones de HPC …
- Actualmente, el paradigma puede proveer el soporte para HPC.
- En general, sobre cloud pueden ejecutarse aplicaciones paralelas con esquema de particionamiento sencillo, y sin comunicaciones (embarrassingly parallel applications).
- Cloud computing provee un paradigma de computación diferente, denominado High Scalability Computing (HSC).
- Muchísimas (del orden de varios millones) unidades de procesamiento (virtuales), con poca capacidad, pero disponibles para usar a demanda.
- Por ejemplo: la unidad básica de Amazon EC2 (Amazon EC2 unit), tiene capacidad equivalente a una CPU correspondiente a un procesador Opteron o Xeon de 1.0-1.2 GHz, del año 2007.

CLOUD COMPUTING y HPC

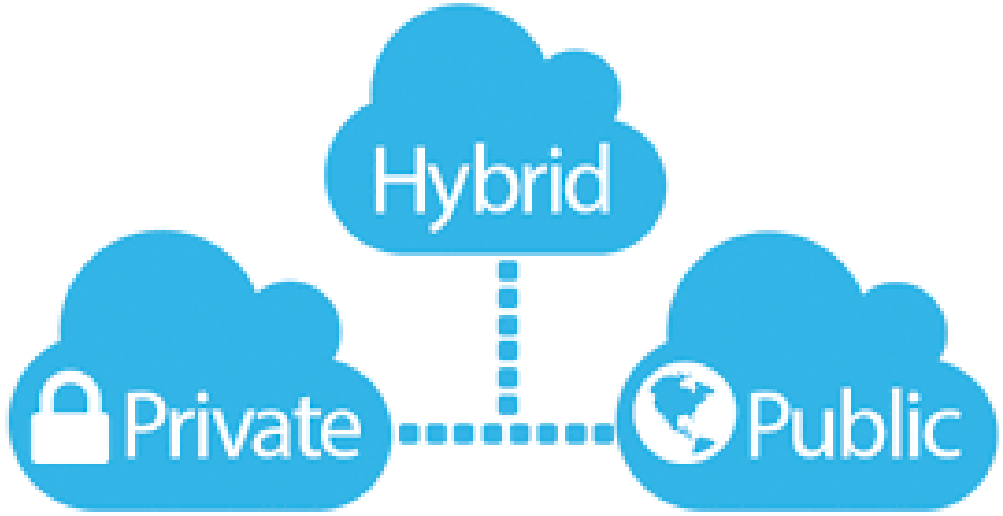
- Algunas infraestructuras cloud pueden dar soporte a aplicaciones previamente diseñadas para ejecutar en grid, o inclusive a programas MPI con requerimientos de comunicación bajos (altas latencias).
- Amazon EC2 ha mejorado su tecnología de comunicaciones, permitiendo la ejecución de aplicaciones basadas en pasaje de mensajes.
- Clouds con GPU o XeonPhi proveen poder de cómputo adicional para aplicaciones más complejas.



CLOUD COMPUTING y HPC

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Amazon Elastic Compute Cloud (EC2) ofrece instancias de cloud computing (IaaS) específicas para aplicaciones HPC y aplicaciones que requieren comunicación entre procesos.
- Características:
  - Desde 23 GB a 68 GB de memoria.
  - Desde 33.5 a 88 unidades de cómputo EC2. El desempeño de una unidad de cómputo EC2 equivale a un CPU Opteron o Xeon del 2007 con 1.0-1.2GHz.
  - SO Linux de 64bits y red 10 Gigabit Ethernet.
  - Desde 840 GB a 4 × 840 GB de almacenamiento.
  - Dos GPU NVIDIA Tesla M2050.
- Desarrollado por Amazon en conjunto con investigadores del Lawrence Berkeley National Laboratory.
- Costo a demanda: USD 2.10–2.40 hora (Linux, USA) y 2.60–3.00 (Windows, USA), o por reserva USD 2000 anuales (sin GPU).



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

MODELOS para DESPLEGAR SISTEMAS CLOUD

CLOUD COMPUTING: MODELOS

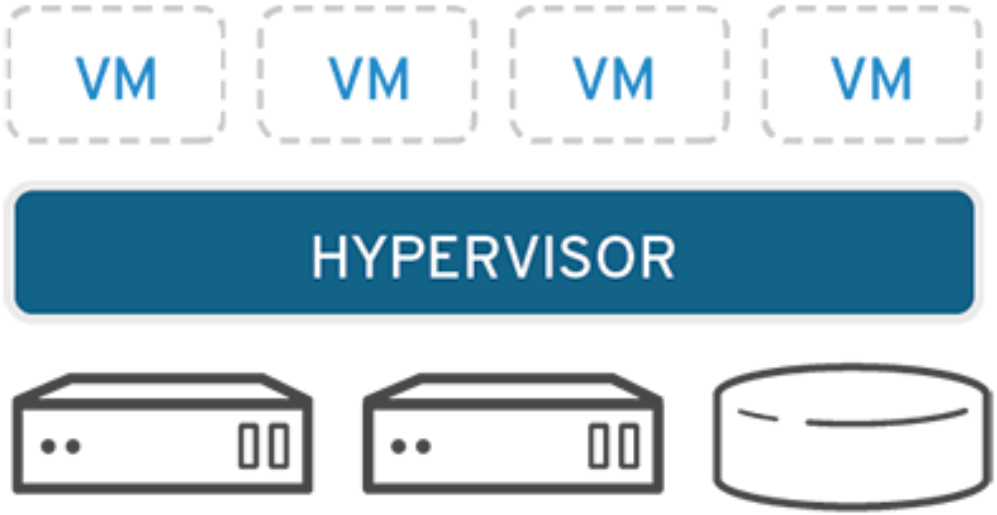
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cloud privada: infraestructura operada por una única organización.
  - Mayor control sobre la infraestructura.
  - Mayor seguridad sobre los datos, el software y las comunicaciones.
  - Requiere inversión en recursos y esfuerzos de implementación.
  - Costos elevados, requerimientos de espacio, refrigeración, mantenimiento, actualización, etc. No se benefician de los aspectos más provechosos del modelo de cloud … "still have to buy, build, and manage, thus do not benefit from less hands-on management", … "[lacking] the economic model that makes cloud computing such an intriguing concept" (Gordon, 2009).
- Cloud pública: servicios ofrecidos a través de una red pública
  - Mayor escalabilidad y flexibilidad, usa datacenters accesibles via Internet
  - Aprovecha los beneficios del modelo de gestión no local.
  - Bajo (o nulo) control sobre la infraestructura.
  - Menor nivel de seguridad (datos, comunicaciones, etc.)

CLOUD COMPUTIN: MODELOS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cloud híbrida: combina infraestructuras de cloud pública y privada. Intenta capturar las ventajas de ambos modelos
  - Los componentes permanecen como entidades separadas, pero conectadas para ofrecer diferentes opciones para desplegar aplicaciones.
  - Permite extender las capacidades de servicios cloud mediante agregación, integración o personalización con otros proveedores de servicios.
- Casos típicos de nubes híbridas:
  - Almacenar datos sensibles en nube local y utilizar servicios (inteligencia computacional, inteligencia de negocios) de la nube pública en modelo SaaS.
  - Cloud bursting: técnica para situaciones temporales como resolver picos de carga. Elasticidad y escalabilidad a demanda. El usuario/organización solamente paga por recursos adicionales cuando los necesita.
  - Se aplica sobre un modelo multiplatforma (cross-platform) con diferentes arquitecturas (x86-64, ARM, etc.) de modo transparente. Potenciados por sistemas en chip (system-on-chip) para servidores.



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

TECNOLOGÍAS de VIRTUALIZACIÓN y SOFTWARE

TECNOLOGÍAS DE VIRTUALIZACIÓN Y SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El paradigma de grid/cloud computing ha sido posible por la convergencia tecnológica en cuatro áreas:

  (1) la virtualización de hardware y los sistemas multinúcleo;

  (2) el desarrollo de los paradigmas de computación utilitaria (incluyendo modelos previos en cluster y grid);

  (3) el desarrollo de sistemas para programar aplicaciones distribuidas; y

  (4) la automatización de los datacenters.

TECNOLOGÍAS DE VIRTUALIZACIÓN Y SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Han posibilitado la construcción de plataformas para computación paralela y distribuida.

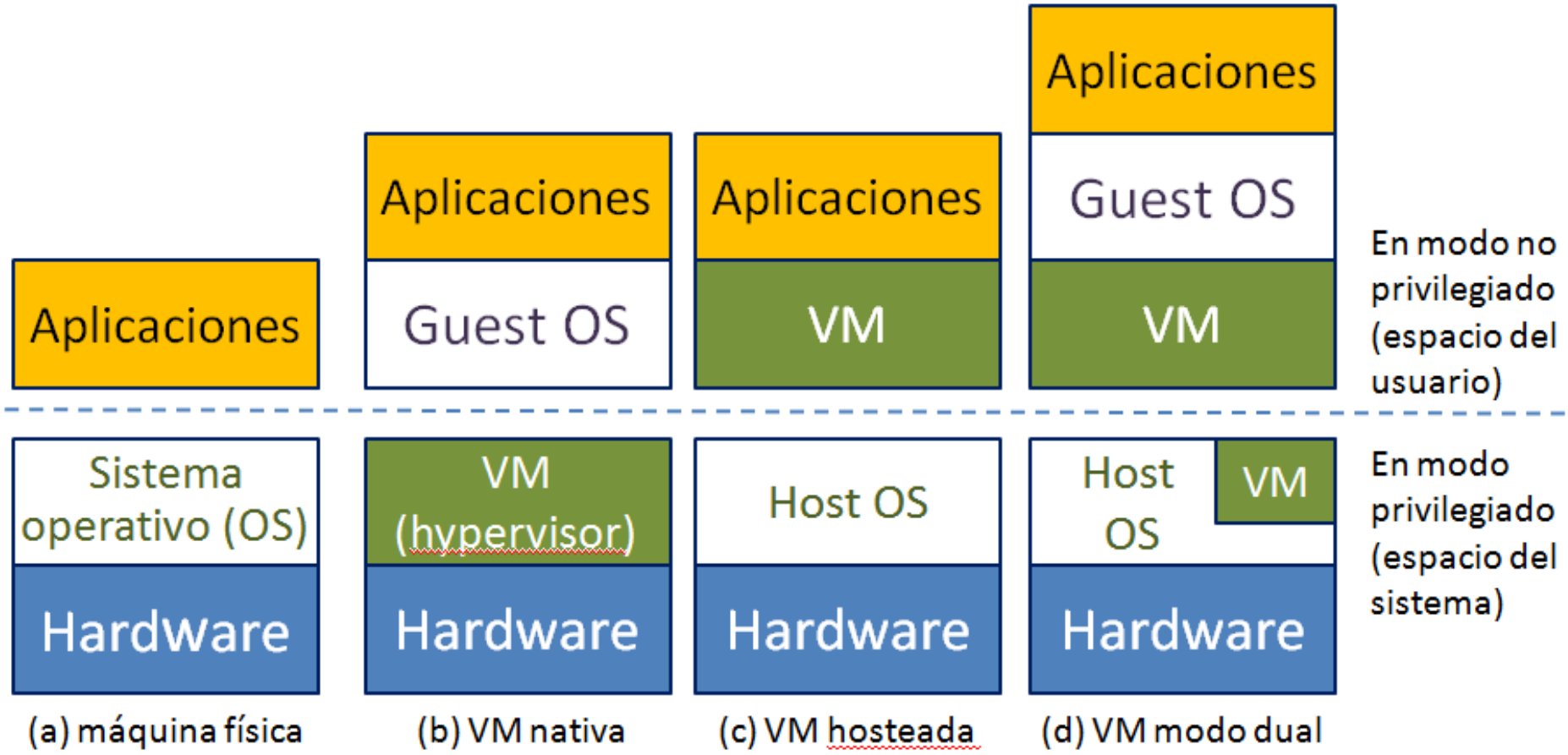
- *Máquinas virtuales y middleware para virtualización*
  - Un computador tradicional tiene una única imagen de sistema operativo, y por este motivo constituye un sistema rígido que acopla muy fuertemente a las aplicaciones con el hardware subyacente.
  - El paradigma de máquinas virtuales (Virtual machines, VMs) es una alternativa que provee flexibilidad para la utilización de recursos computacionales, aprovechando recursos de procesamiento ocioso a la vez que proporciona mayores niveles de manipulación de hardware y software y mejora la seguridad de las aplicaciones.

TECNOLOGÍAS DE VIRTUALIZACIÓN Y SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Soluciones de virtualización
  - Tradicional: las aplicaciones se ejecutan sobre un sistema operativo que gestiona a la propia máquina física.
  - VM nativa: se configuran VMs mediante un monitor (hypervisor), que es un middleware para gestión de las VMs ejecutando en modo privilegiado. Cada VM ejecuta un sistema operativo *guest*, potencialmente diferente al sistema operativo del host físico, y sobre este guest OS se ejecutan las aplicaciones de usuario.
  - VM hosteada: la máquina virtual ejecuta sobre el sistema operativo del equipo físico, en modo no privilegiado, sin requerir modificaciones al sistema operativo de base.
  - Modo dual: permite tener al gestor de máquinas virtuales en modo protegido y parte ejecutando en modo de usuario. Esta solución puede requerir modificar el sistema operativo de base.
- Estas estrategias proporcionan flexibilidad y portabilidad para las VMs, que pueden ejecutar de modo independiente al hardware subyacente.

TECNOLOGÍAS DE VIRTUALIZACIÓN Y SOFTWARE

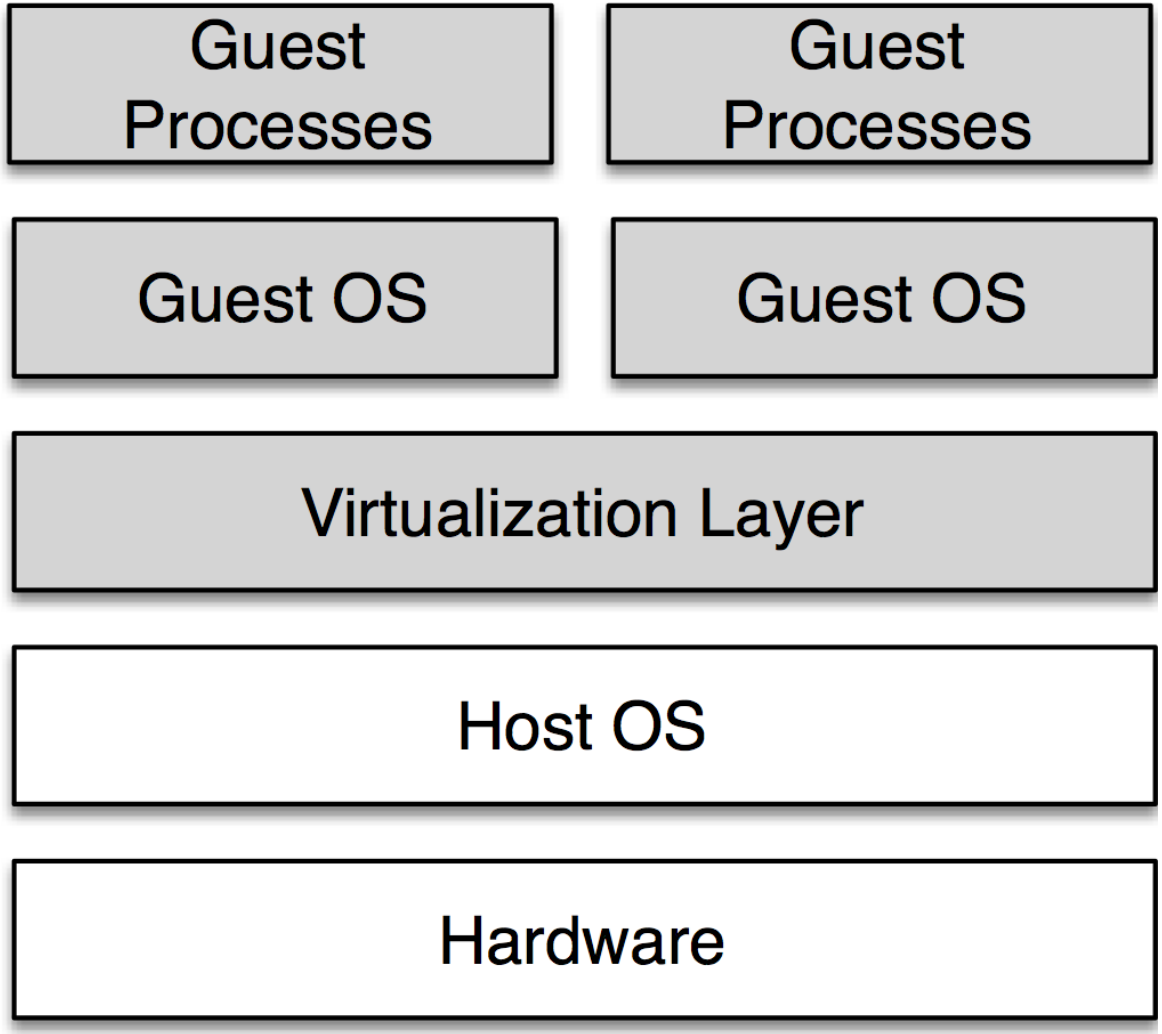


VIRTUALIZACIÓN BASADA en HIPERVISORES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Soluciones basadas en hypervisores
  - Es una implementación del modelo de VM nativa que consiste en un sistema operativo guest trabajando sobre un sistema operativo de host que provee una abstracción total de la máquina virtual.
  - Cada máquina virtual tiene su propio OS que ejecuta completamente aislado del resto de las VMs.
  - Los beneficios del enfoque incluyen:
    - independencia del hardware.
    - aislamiento de las VMs.
    - posibilidad de ejecutar múltiples imágenes de OS en un único host.
  - Combinado con estrategias de puntos de control (checkpointing) y migración de procesos, las VMs pueden ser desplegadas y redesplegadas de acuerdo a las necesidades de los usuarios y la carga de las máquinas físicas.
  - Son populares actualmente para sistemas livianos y aplicaciones de usuario
  - Hypervisores más utilizados: XEN, VMWare, KVM

VIRTUALIZACIÓN BASADA en HIPERVISORES



XEN
VMWare
KVM

Hypervisor-based Virtualization

VIRTUALIZACIÓN BASADA en HIPERVISORES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Soluciones basadas en hipervisores
  - Desventajas:
    - *Incrementan considerablemente la carga* de la infraestructura de base.
    - La abstracción completa de cada VM impone importantes overhead de procesamiento y manejo, que puede degradar notoriamente el desempeño, en especial para aplicaciones intensivas en entrada/salida y en comunicación de datos a través de la red.
    - El costo de alojar un OS completo por sobre del OS de base limita el número de instancias de VM que se pueden ejecutar en un único server, por los requerimientos de recursos (disco, memoria, CPU) del OS guest y de las aplicacíones que ejecutarán sobre él.
      - Aplicaciones usualmente demandan memoria y CPU, y posiblemente almacenamiento en el caso de procesar grandes volúmenes de datos.
      - OS guest demanda disco, memoria y CPU.
  - No son una alternativa recomendable para soluciones complejas o que requieren alta escalabilidad.

VIRTUALIZACIÓN BASADA en CONTENEDORES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

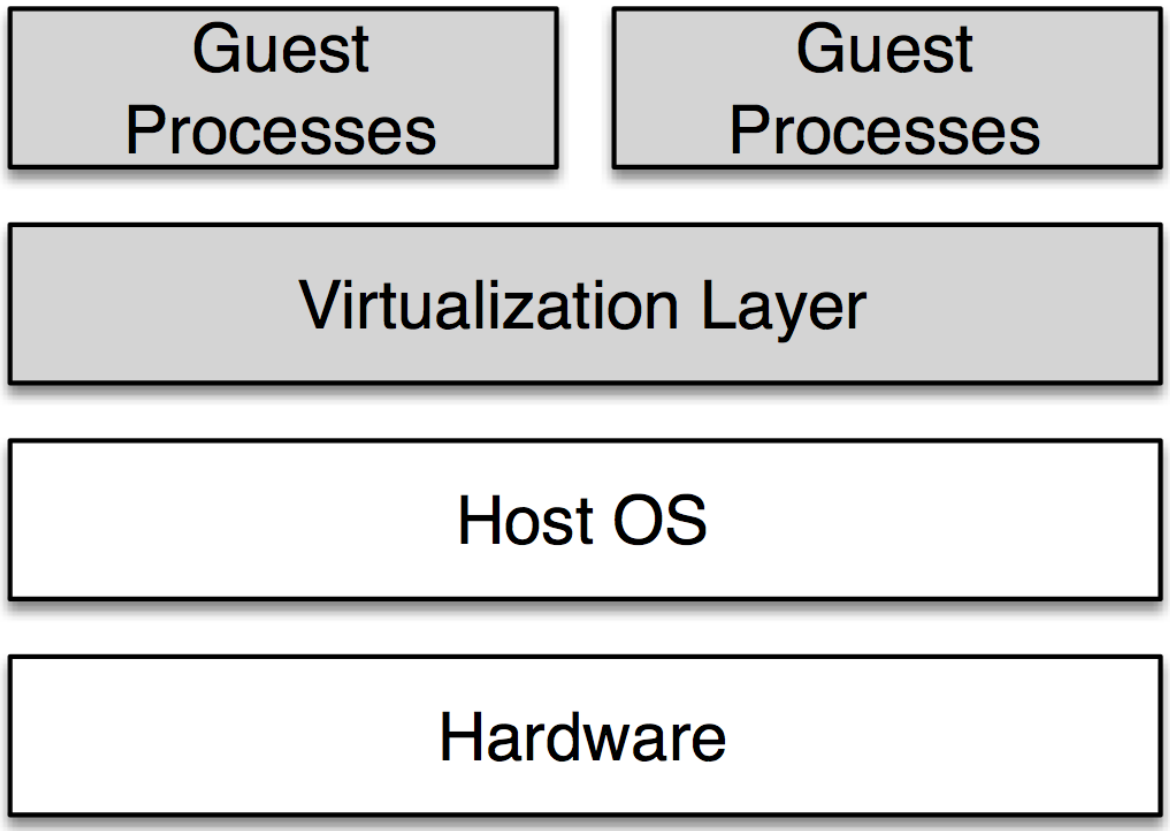
- Soluciones basadas en containers
  - Es una implementación del modelo de VM hosteada.
  - El enfoque se basa en las funcionalidades de los OS de particionar los recursos de la máquina física, creando múltiples instancias de espacios de usuario aislados (*containers*) sobre un único kernel del host.
  - Los containers trabajan a nivel del sistema OS y todas las VMs comparten un único kernel de base.
  - Proporciona menor nivel de aislamiento que las soluciones basadas en hypervisores, pero se han realizado avances recientes en mejorar los niveles de aislamiento para los containers.
  - Las primeras herramientas para containers (e.g. chroot) no proveían suficiente seguridad y aislamiento de recursos, pero actualmente existen opciones que implementan desde sandboxes básicos para procesos individuales a soluciones para sistemas completos.

VIRTUALIZACIÓN BASADA en CONTENEDORES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Soluciones basadas en containers
  - El soporte para containers depende los servicios del sistema operativo del host (BSD jails, Solaris Zones, Linux namespaces/cgroups), pero en general se basan en dos métodos:
    1. un mecanismo de namespaces que permite a diferentes procesos tener diferentes vistas del sistema.
    2. un método de manejo de recursos que restringe el uso de memoria, CPU e I/O para los diferentes containers.
  - En Linux:
    - Linux-Vserver, OpenVZ, Google Lmctfy y LXC.
    - Todos proveen manejo de bajo nivel de los containers, virtualización y manejo de namespaces y aislamiento.

VIRTUALIZACIÓN BASADA en CONTENEDORES



Container-based Virtualization

V-Server
OpenVZ
Lmctfy
LXC

VIRTUALIZACIÓN BASADA en CONTENEDORES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Linux-VServer
  - Solución clásica de Linux, basada en herramientas clasicas (chroot, rlimit) para manejo básico de containers (aislamiento, manejo de recursos) y un patch del kernel para proveer aislamiento de procesos, uso de CPU y red.
  - Usa un espacio global de identificadores de proceso (PID) que maneja las vistas de cada proceso y limita las comunicaciones con procesos fuera del grupo asociado al container.
  - Como desventajas, VServer no permite migración, checkpointing o resumir containers, ya que Linux no admite relanzar procesos con el mismo PID.
- OpenVZ y LXC
  - Se basan en nuevas funcionalidades del kernel (namespaces y cgroups)
  - Namespaces para aislar los containers y garantizar acceso a su subconjunto de recursos. Cada namespace tiene su propio espacio de PID space, permitiendo checkpointed, migración y resumir procesos y containers.
  - También se usan para controlar las capacidades de red e IPC

VIRTUALIZACIÓN BASADA en CONTENEDORES

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- OpenVZ y LXC
  - Utilizan diferentes mecanismos de gestión de recursos
  - OpenVZ implementa su propio componente de gestión (User Beancounters), que controla el scheduling, quotas de disco, etc.
  - LXC utiliza cgroups de Linux, configurando namespaces de red y limitando uso de CPU y recursos.

- Son sistemas de bajo nivel que han sido extendidos para proveer funcionalidades más avanzadas (desplegar containers, migración avanzada)
  - Sobre OpenVZ: sistema de virtualización commercial Parallels Cloud Server.
  - Sobre LXC: Docker, manejador open-source para containers desarrollado por el proveedor PaaS dotCloud, incluye lógica para reducir el overhead de creación y administración de nuevas VMs.

ENTORNOS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Arquitectura orientada a servicios (SOA)
  - SOA es un paradigma de arquitectura de software aplicable al diseño e implementación de sistemas distribuidos.
  - Las principales funcionalidades del paradigma se relacionan con proveer facilidad y flexibilidad de integración con otros sistemas (en especial sistemas legados), permitiendo reducir los costos de implementación y lograr una adaptación ágil ante cambios en los procesos de negocio.
  - SOA define un conjunto de capas de software orientadas a proveer un entorno para el desarrollo de aplicaciones escalables, basadas en el paradigma de exposición e invocación de servicios (en general, servicios web), simplificando la interacción entre diferentes sistemas

ENTORNOS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Capas de SOA:
  - *Capa de aplicaciones básicas*, que incluye sistemas desarrollados bajo cualquier arquitectura o tecnología, geográficamente dispersos y bajo cualquier figura de propiedad;
  - *Capa de exposición de funcionalidades*, que expone funcionalidades de la capa de aplicaciones en forma de servicios (generalmente servicios web);
  - *Capa de integración de servicios*, que facilitan el intercambio de datos entre aplicaciones para procesos internos o en colaboración;
  - *Capa de composición de procesos*, que permite definir cada proceso en términos de las características de la lógica de la aplicación y sus necesidades, y que varían en función del problema a resolver;
  - *Capa de entrega*, donde los servicios son desplegados a los usuarios finales.
- El paradigma de SOA se aplica a la construcción de grids, clouds, grids de clouds, clouds de grids, clouds de clouds (conocidos como interclouds), y sistemas de sistemas en forma general.

ENTORNOS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Servicios web (web services)
  - Tecnología para la comunicación de datos entre aplicaciones distribuidas utilizando protocolos y estándares diseñados para tal fin. El modelo permite comunicar e interoperar a aplicaciones desarrolladas en diferentes lenguajes de programación y/o que ejecutan sobre diferentes plataformas de hardware y software. Como medio de comunicación se utilizan redes globales, en general Internet.
  - La interoperabilidad se consigue adoptando estándares abiertos, por ejemplo especificando completamente todas las características del servicio y su entorno mediante comunicaciones utilizando el Protocolo Simple de Acceso a Objetos (Simple Object Access Protocol, SOAP).
  - El entorno de aplicación toma las características de un *sistema operativo distribuido universal*, con capacidades totalmente distribuidas y comunicación con mensajes. El enfoque es poderoso, pero los resultados están condicionados a los acuerdos en partes clave del protocolo, que puede ser complejo de implementar por los softwares disponibles.

ENTORNOS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Representational State Transfer (REST)
  - REST es una arquitectura de software para computación distribuida sobre el protocolo HTTP, que ha sido exitosamente aplicada en los últimos años.
  - Adopta la simplicidad como principio universal para la comunicación y delega la mayor parte de los componentes complejos a la propia aplicación.
  - REST utiliza información mínima en el cabezal de los mensajes y es el cuerpo del mensaje quien transporta toda la información necesaria para la comunicación.
  - Como consecuencia de su simplicidad y agilidad, la arquitectura REST es más apropiada para entornos de tecnología "rápidos" sobre infraestructuras livianas, que permiten desarrollar servicios con un herramientas de desarrollo minimales.
  - Se obtiene así un paradigma de desarrollo ágil, escalable y que requiere de poco esfuerzo de desarrollo.
  - REST es una alternativa eficiente a SOAP, que provee una gran integración con HTTP, utilizándose en general un diseño de "XML sobre HTTP".

ENTORNOS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- CORBA y Java RMI
  - Common Object Request Broker Architecture (CORBA) es un estándar para la cooperación entre diversos componentes de software (escritos en múltiples lenguajes de programación) para facilitar el desarrollo de aplicaciones distribuidas en entornos heterogéneos.
  - En CORBA y Java, las entidades que participan en un sistema distribuido están relacionadas entre sí mediante invocación de procedimientos remotos (RPC) y la manera más simple de construir sistemas que combinan aplicaciones es ver a las entidades como objetos distribuidos.
  - En Java, este mecanismo se implementa mediante un programa tradicional donde las invocaciones a métodos están reemplazadas por invocaciones a métodos remotos (RMI).
  - CORBA soporta un modelo similar, con una sintaxis que refleja el estilo de interfaces entre entidades (objetos) de C++.

HERRAMIENTAS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Tres ejemplos de software de programación y gestión para sistemas cluster, grid y cloud de computación distribuida

- *Clusters: la biblioteca Message Passing Interface (MPI)*
  - MPI es una biblioteca de desarrollo de sistemas paralelos y distribuidos basada en el paradigma de pasaje de mensajes para la comunicación entre procesos.
  - Fue definido a mediados de la década de 1990 como un estándar para el desarrollo de este tipo de aplicaciones.
  - Aunque MPI fue originalmente concebido para el diseño de aplicaciones paralelas eficientes en infraestructuras cluster, es posible utilizarlo para combinar clusters, grids y sistemas P2P con servicios web mejorados y aplicaciones de computación utilitaria.

HERRAMIENTAS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- *Grid: Open Grid Services Architecture (OGSA) y Globus*
  - OGSA es un estándar para el uso público y general de servicios grid, definido para sacar el mayor provecho en la ejecución de sistemas distribuidos genéricos y aplicaciones de gran escala, que hacen uso intensivo de recursos y comparten grandes volúmenes de datos.
  - Las principales características de OGSA incluyen un entorno para la ejecución de aplicaciones distribuidas, servicios de autenticación de clave pública utilizando autoridades de certificación, manejo de confiabilidad y políticas de seguridad en el grid.
  - Globus es un middleware para grid (también se utiliza en sistemas cluster) que implementa el estándar OGSA para el manejo de recursos distribuidos. Globus provee componentes y APIs para descubrimiento de recursos, asignación de tareas a recursos y mecanismos para garantizar la seguridad mediante autenticación PKI multisitio, entre otras funcionalidades.

HERRAMIENTAS DE SOFTWARE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- *Cloud: MapReduce y Hadoop*
  - MapReduce es un modelo de programación genérico, muy adaptable el procesamiento de grandes volúmenes de datos en infraestructuras cloud. Ha recibido gran atención recientemente como consecuencia del desarrollo de bibliotecas e implementaciones específicas orientadas al procesamiento web, búsquedas web y problemas genéricos de *big data*.
  - El usuario especifica una función Map para generar un conjunto de valores intermedios clave/valor a partir de un conjunto de datos de gran tamaño. A continuación, una función de reducción especificada por el usuario se aplica para combinar todos los valores intermedios con el mismo valor de clave.
  - Este modelo de paralelismo simple es altamente escalable y permite explotar el paralelismo a diferentes niveles. Una aplicación MapReduce tipica es capaz de manejar varios Terabytes de datos en cientos o miles de recursos de cómputo simultáneamente.

GRIDS vs CLOUDS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Las fronteras entre sistemas grid y cloud son borrosas actualmente, en particular por la utilización de tecnologías de workflow que se aplican para coordinar u orquestar servicios que definen modelos de procesos de negocio transaccionales sobre grid y sobre clouds.
- Las principales diferencias que se siguen manteniendo están relacionadas con el manejo de los recursos:
  - Un sistema grid utiliza un modelo de recursos estáticos, mientras que los sistemas cloud enfatizan en sistemas elásticos, basados en virtualización.
  - Sin embargo, la creación de metasistemas que combinen las características de ambos paradigmas es posible.
  - Por ejemplo, un sistema grid que combine múltiples clouds puede proveer mejores funcionalidades que un sistema cloud puro, porque puede proveer soporte explícito para negociar la asignación de recursos.
  - Los metasistemas distribuidos (clouds de clouds, grids de clouds, cloud de grids, interclouds) proporcionan nuevos desafíos y oportunidades para construir arquitecturas novedosas.

Procesamiento de grandes volúmenes de datos (Big Data)

Big Data

- ‘Big Data’: análisis y procesamiento de grandes volúmenes de datos.

- Según IBM:
  - “… enormes cantidades de datos (estructurados, no estructurados y semi estructurados) que tomaría demasiado tiempo y sería muy costoso cargarlos en una base de datos relacional para su análisis”.

- Según Wikipedia:
  - “Es el sector de las tecnologías de la información que referencia a sistemas que manipulan grandes volúmenes de datos’.

Big Data

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Según un informe de Gartner, las tecnologías relacionadas con 'Big Data' crearon 4.4 millones de puestos de trabajo en el año 2015, y solamente hubo personal capacitado para un tercio de ellos

- Big Data tiene una amplia aplicabilidad en diversos ámbitos: Hadoop/Spark están entre las primeras 10 tecnologías en tendencias de trabajo en EEUU.

- Las tecnologías de Big Data son las que tienen mayor crecimiento y demanda laboral actual y en el futuro.

Big Data

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Hoy en día, con USD 600 se puede comprar un disco que almacene toda la música existente en el mundo
  - Los costos del almacenamiento se han reducido, lo que ha permitido a empresas almacenar cantidades cada vez mayores de datos e información.
- Existen más de 8.000 millones de dispositivos móviles (más móviles que personas!) en uso en todo el mundo, generando datos contínuamente.
- Actualmente se comparten más de 30.000 millones de 'datos' por Facebook por mes y el crecimiento de la información generada a nivel mundial es mayor al 40%.
- El 80% de los datos que se generan en el mundo están desestructurados. Este tipo de datos crece 15 veces más rápido que los estructurados.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Pero …
qué volumen de datos significa ‘big’?

Big Data

- En la actualidad se generan más de 5 trillones de bytes de datos por día.
- El 90% de los datos digitales existentes fueron generados en los últimos dos años (2016 y 2017).
- El crecimiento de información disponible es exponencial, y cada vez más impresionante.
- Poder analizar (de forma eficiente) los datos disponibles para extraer información es de gran relevancia en muchos ámbitos:
  - Ciencia
  - Sociedad (salud, vivienda, movilidad, etc.)
  - Comercio
  - Otros

Big Data

- Existen más de 4.000 millones de usuarios de Internet (50% más que en 2014, más del 50% de la población de la Tierra).

Big Data

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Existen más de 4.000 millones de usuarios de Internet (50% más que en 2014, más del 50% de la población de la Tierra).
- Muchos usos:
  - 840 nuevos usuarios de redes sociales por minuto.
  - Más de 500.000 tweets por minuto (60% más que en 2013).
  - 4.146.600 videos vistos en Youtube por minuto (más del triple que en 2014), 400 horas de nuevos videos se suben por minuto.
  - Más de 50.000 posts en Instagram por minuto.
  - Más de 4 millones de likes y más de 3.5 millones de posts en Facebok por minuto (400% más que en 2011). Más de 550.000 comentarios, más de 300.000 actualizaciones de estado y más de 150.000 fotos.
  - 4 millones de búsquedas en Google por minuto.
  - Más de 15.5 millones de textos enviados por minuto.

Big Data

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Datos por día
  - 1.5 billones de nuevos datos producidos por usuarios de las redes sociales
  - 656 milliones de tweets por día
  - Más de 4 millones de horas de contenido subido a Youtube por día y usuarios mirando más de 6.000 millones de horas de videos por día.
  - Más de 70 millones de posts en Instagram por día
  - Más de 2.000 millones de usuarios activos en Facebook y 1.4 millones que publican todos los días: más de 4.500 millones de mensajes y más de 6.000 millones de likes por día
  - 22.000 millones de mensajes enviados por día (más del 17% que en 2016)
  - Más de 5.500 millones de búsquedas en Google por día.
  - 300.000 millones de mails por día (4-5% de crecimiento anual)

Big Data

- Datos móviles:
  - 8 exabytes (10^{18} bytes) transferidos (carga y descarga) en 2017.
  - Más de 4.000 millones de usuarios moviles de Internet (número mayor que los usuarios fijos).
  - Páginas web accedidas: 51.4% (móvil), 43.4% (fijo), 5% (tablet).
- Fuentes muy variadas generan estos volúmenes de información
  - GPS y dispositivos de Internet of Things (IoT).
  - Información multimedia.
  - Información de la web.
  - Logs e información de sistemas.
- La información generada por usuarios es inmensa, pero es aún mayor el volumen de información generada por sistemas (logs, sensores, GPS).

Big Data: almacenamiento y análisis

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El mayor volumen de información (estimado en más de 80% de la información disponible) se encuentra en datos no estructurados, un conglomerado masivo de elementos sin una estructura interna identificable.
- Procesados apropiadamente, los elementos pueden ser buscados y categorizados para obtener información valiosa.
- Casos típicos: correos electrónicos, archivos (texto, pdf), hojas de cálculo, imágenes digitales, video, audio, publicaciones en medios sociales, etc.

Big Data: almacenamiento y análisis

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La capacidad de almacenamiento de datos se ha incrementado notoriamente, pero no la velocidad de acceso:
  - 1990: disco típico de 1,370 MB se podia leer a 4.4 MB/s, en 5 minutos.
  - 2015: disco típico de 1TB se lee a 100 MB/s, require 2 horas y media.
- Escribir en disco es aún más costoso !
- Solución: almacenamiento distribuido (100 discos, se leen en 2 minutos).
- Con 100 discos se dispone almacenamiento para 100 TB !
- Otro beneficio: tolerancia a fallos !
  - Replicación y redundancia al estilo RAID.
  - Particionamiento/réplicas al estilo de Hadoop Distributed Filesystem (HDFS).
- Se necesita soporte para combinar datos provenientes de multiples fuentes.

Big Data: procesamiento

- Para procesar grandes volúmenes de datos y obtener información valiosa es necesario aplicar algoritmos que permitan aprovechar la escalabilidad de recursos de cómputo y procesamiento de datos.
- Para usuarios no expertos en computación de alto desempeño es necesario disponer de un modelo y de un framework para el procesamiento eficiente de modo sencillo.
- Un framework simple agiliza los procesos de desarrollo, permitiendo al desarrollador enfocarse en resolver el problema en cuestión.

- MapReduce prove un modelo de progamación que abstrae el manejo de datos de diferentes fuentes y prove confiabilidad.
- Hadoop prove un sistema confiable y escalable para almacenamiento y análisis de datos sobre código abierto y plataformas de bajo costo.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

El modelo de computación Map-Reduce

MAP-REDUCE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Modelo de programación paralela/distribuida que permite implementar semi-automáticamente algoritmos paralelizables para resolver problemas complejos
- Presentado por Google en 2004: *"*J. Dean and S. Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters,* 6th Symposium on Operating System Design and Implementation, San Francisco, CA, 2004.*"*
- Modelo independiente de platformas y lenguajes
- Es la base de las plataformas para procesamiento de big data y datos no estructurados (NoSQL)
- Las abstracciones provistas por los frameworks de alto nivel (Spark SQL, Hive, etc.) permiten procesar datos sin implementar explícitamente los components del modelo MapReduce
- Comprender los fundamentos de MapReduce permite entender la programación distribuida para procesamiento de datos

MAP-REDUCE: motivación

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Previo a la presentación de Google File System, los sistemas de procesamiento distribuido daban soporte para el análisis de volúmenes modestos de datos.
- Algunas limitaciones de los entornos tradicionales para programación en clusters y grids:
  - Complejidad de programación: el programador debe resolver la comunicación y sincronización entre procesos distribuidos, el control de dependencias de datos, etc.
  - Fallos parciales (sincronización, distribución de datos, deadlocks, etc.)
  - Cuellos de botella en el acceso a los datos, que usualmente se almacenan en repositorios distribuidos o sistemas de archivos remotos.
  - Limitaciones de escalabilidad, por comunicaciones sobre redes con ancho de banda limitado. Al agregar recursos de cómputo el overhead de comunicación reduce el desempeño efectivo.

MAP-REDUCE: objetivos de diseño

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- A principios de la década del 2000, los buscadores de Internet se encontraron con requisitos importantes de recolectar, almacenar, indexer y procesar un volumen de datos no estructurados que crecía exponencialmente. Nuevos desafíos surgieron con la expansión de los dispositivos móviles, sensores IoT y redes sociales.
- Google planteó diseñar una plataforma de almacenamiento y procesamiento escalable para miles de recursos de cómputo en hardware de bajo costo.
- Objetivos de diseño:
  - Paralelismo y distribución autómática de datos
  - Tolerancia a fallos (parciales) de la infraestructura
  - Planificación dinámica de entrada/salida para limitar el ancho de banda utilizado
  - Monitoreo centralizado para diagnóstico y optimización de desempeño

MAP-REDUCE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Modelo genérico de computación paralela para procesar grandes volúmenes de datos en infraestructuras cluster, grid y cloud.
- Se deben implementar dos funciones:
  - Map: procesa entrada y genera pares intermedios clave/valor (key/value).
  - Reduce: agrupa (merge) todos los pares clave/valor asociados con una misma clave.
- Las funciones Map y Reduce son funciones de alto orden que tienen su origen en la programación funcional.
- Existen diversas implementaciones de MapReduce en distintos lenguajes y frameworks (Hadoop MapReduce, MPI-MapReduce, MapReduce de Google Apps, Mongo DB MapReduce, etc.).
- Ambigüedad entre el modelo y sus implementaciones, en especial la más difundida, la de Hadoop.

MAP-REDUCE: registros y pares clave-valor

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Elementos para modelar los datos: conjuntos (datasets), registros y pares clave-valor (key-value).
- Dataset: colección de registros, usualmente procesada en paralelo en una plataforma distribuida (ejemplos: bloques de HDFS, RDD de Spark, etc.)
  - No debe confundirse con la API Dataset disponible en Spark SQL
- Registro: unidad de manejo de datos (entrada, intermedios, salida), que se representa en la forma de par clave-valor (key-value).
- Clave-valor: representan atributos de los datos
  - Clave: identificador del atributo (por ejemplo, el nombre de un atributo)
  - En algunos sistemas distribuidos NoSQL la clave debe ser única.
  - Sin embargo, en Hadoop y Spark no se require que la clave sea única
- Valor: dato correspondiente a la clave, puede ser escalar (número, string, etc.) o complejo (lista de objetos, etc.).

MAP-REDUCE: pares clave-valor

- Ejemplos de pares clave-valor

Clave	Valor
Departamento	Flores
Capital	Trinidad
Temperatura	[22,25,27,23,24,22,29]

- Problemas complejos se descomponen en una serie de operaciones sobre pares clave-valor
  - Por ejemplo, en Hadoop MapReduce en Java los pares clave-valor son unidades atómicas de datos sobre las cuales se realiza todo el procesamiento
- Implementados en muchos lenguajes de programación
  - Por ejemplo, para expresar configuraciones paramétricas y metadatos.
  - Dictionaries (dicts) de Python, hashes de Ruby, etc.
  - Su implementación en general está oculta al programador.
- Implementados en Spark en PairRDD.

MAP-REDUCE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Recordemos que existe una ambigüedad entre el modelo MapReduce y sus implementaciones, en especial la implementación de Hadoop
- En el contexto del curso se presentarán los principales conceptos de MapReduce y ejemplos de su implementación en el framework Hadoop.
- El modelo MapReduce incluye dos etapas de procesamiento que deben ser implementadas por el programador (etapas de mapeo y reducción) y una etapa complementaria (agrupamiento) que es implementada automáticamente por el framework (en Hadoop, se implementa por las fases Shuffle y Sort).

EL MODELO DE COMPUTACIÓN MAP-REDUCE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Un algoritmo Map-Reduce caracteriza a un determinado método o tarea de procesamiento de datos.
- Opera con pares clave-valor: (k,V).
- Maneja strings, números o tipos simples de datos, pero se pueden implementar estructuras más complejas para distintos tipos de problemas.
- El paralelismo está implícito en la aplicación vectorial de las operaciones Map y Reduce.

- La entrada y la salida de una tarea Map-Reduce es una lista de pares {(k,V)}.
- La tarea Map-Reduce queda definida por dos funciones:
  - map: (f,{(k1;v1)}) $\rightarrow${(k2;v2)}
  - reduce: (k2;{v2}) $\rightarrow${(k3;v3)}

MAP-REDUCE: ETAPAS

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Etapas:
  1. Lectura de datos de entrada [carga y asignación de datos para map]
  2. Mapeo [map]
  3. Partición [asignación de datos para reduce]
  4. Comparación [ordenamiento de datos]
  5. Reducción [reduce]
  6. Salida [usualmente a un sistema de archivos distribuido]

  [datos] → Map → [agrupar/ordenar] → Reduce → [salida]

- Las funciones Map y Reduce trabajan sobre elementos estructurados en pares (clave, valor).

MAP-REDUCE: LA FUNCIÓN MAP

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La función Map aplica una transformación (función) a cada elemento en un conjunto o lista y retorna una lista de resultados.

  $Map(f(x),X[1:n]) \rightarrow [f(X[1]),...,f(X[n])]$

- Ejemplo: $Map(x^2,[0,1,2,3,4,5]) = [0,1,4,9,16,25]$

- Por cada elemento que recibe, la función Map retorna una lista de valores (puede ser un único valor, dependiendo del caso).
- La función Map se aplica en paralelo a cada elemento del conjunto de datos. Luego de aplicada la función Map, cada elemento del conjunto de datos original es transformado en una lista de pares (k2,v2).
- Posteriormente, el framework recolecta, de todas las listas, los pares (k2,v2) que comparten la misma clave, los agrupa y los envía a la etapa de Reduce.

MAP-REDUCE: LA FUNCIÓN REDUCE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La función Reduce se aplica en paralelo sobre cada grupo resultado del Map.

  Reduce(k2, list (v2)) → list(v3)

- Iterativamente, aplica una función determinada (reducción) sobre el resultados previo (parcial) y el elemento actualmente procesado.

- En general, la función Reduce retorna un solo valor v3 o la colección vacía, aunque puede retornar todos los valores que sean necesarios.
- Ejemplo: Reduce(x+y,[0,1,2,3,4,5]) = (((((0+1)+2)+3)+4)+5) = 15

- En definitiva, un framework MapReduce toma una lista de pares de elementos de tipo (clave, valor) y los transforma en una lista de valores resultado del procesamiento.

EJEMPLO: WORD COUNT

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Algoritmo Map-Reduce para contar la cantidad de ocurrencias de cada palabra en un conjunto de páginas.
- Entrada: un archivo de texto.
- Salida: el número de ocurrencias de cada palabra en el texto.

- Implementación secuencial: analizador (parser) de strings y una lista de hash clave-valor (HashList <k,V>)

- Implementación distribuida: el diseño de la aplicación es similar
  - Paso 1: cada nodo ejecuta el parser para una porción del archivo de entrada y almacena sus resultados parciales.
  - Paso 2: un proceso distinguido suma los valores correspondientes a cada clave en la lista de hash.

WORD COUNT: EL PROCESO MAP

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La función Map ejecuta en paralelo en un conjunto de procesos.
- La entrada del proceso Map es el (sub)conjunto de palabras $\{w\}^p$ correspondiente al proceso p, según la descomposición de dominio adoptada.
- El cómputo del proceso Map involucra un ciclo de conteo y agregación:
  - Cada vez que se encuentra la palabra w_i en la entrada, se agrega $<w_i, 1>$ al resultado.
- La salida del proceso Map es una lista de pares: $<w_i,1>$, repetidas según la cantidad de ocurrencias de cada palabra $w_i \in \{w\}^p$.

WORD COUNT: EL PROCESO REDUCE

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Se ejecuta una función reduce por cada una de las claves $w_i \in \{w\}$.
- Cada proceso reduce recibe una lista $\{<w_i,1>^p, <w_i,1>^{p+1}, \dots\}$ conteniendo los elementos generados por cada proceso Map p, según la descomposición de dominio adoptada
- El cómputo del proceso Reduce involucra un ciclo de agregación (suma): cada vez que se encuentra un registro conteniendo la palabra w_i, se incrementa el contador $\#w_i$.
- La salida de cada proceso Reduce es un par $<w_i,\#w_i>$ indicando la cantidad de ocurrencias de la palabra $w_i \in \{w\}$.

EJEMPLO: WORD COUNT

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Algoritmo Map-Reduce para contar la cantidad de ocurrencias de cada palabra en un conjunto de páginas.
- Páginas:

  Page 1: Deer Bear River

  Page 2: Car Car River

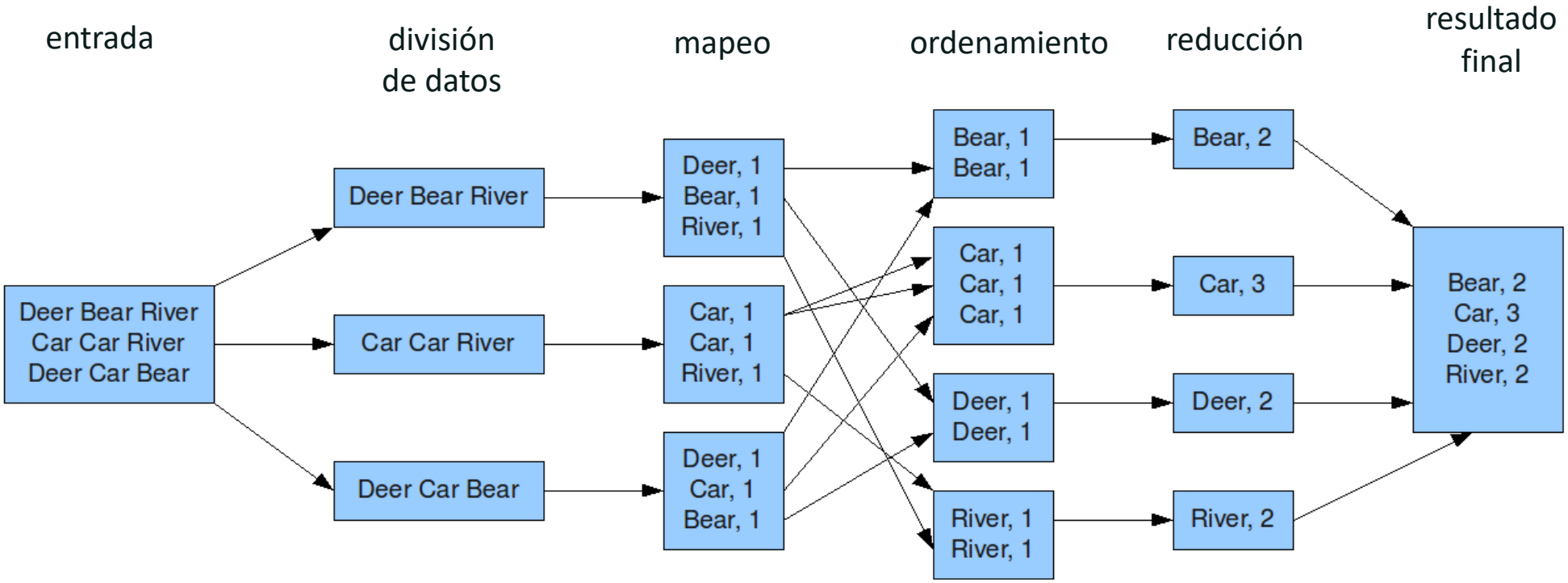
  Page 3: Deer Car Bear
- Lista de pares (k1,v1):

  [(Page 1, Deer Bear River),

  (Page 2, Car Car River),

  (Page 3, Deer Car Bear)]

EJEMPLO: WORD COUNT



Operativa de MapReduce para el ejemplo word count

EJEMPLO: WORD COUNT

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Algoritmo Map-Reduce para contar la cantidad de ocurrencias de cada palabra en un conjunto de páginas.
- Páginas:

  Page 1: Deer Bear River

  Page 2: Car Car River

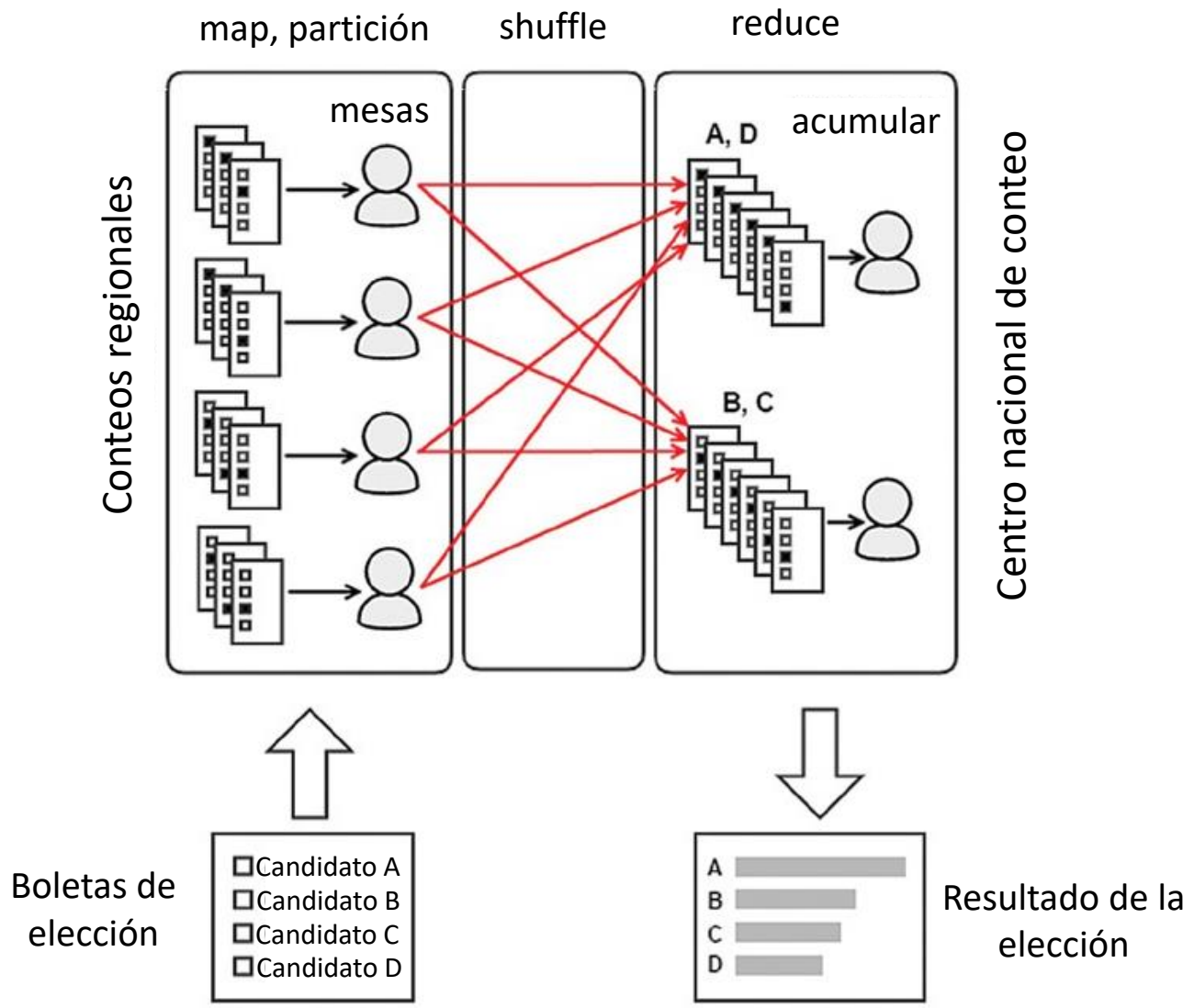
  Page 3: Deer Car Bear
- Lista de pares (k1,v1):

  [(Page 1, Deer Bear River),

  (Page 2, Car Car River),

  (Page 3, Deer Car Bear)]

MapReduce: conteo de votos



UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Apache Hadoop

Apache Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Apache Hadoop es un framework de software para el desarrollo y ejecución de aplicaciones distribuidas bajo una licencia libre.
- Permite a las aplicaciones trabajar con grandes infraestructuras de cómputo (miles de nodos) y grandes volúmenes (petabytes) de datos.
- Hadoop se inspiró en los documentos Google para MapReduce y Google File System (GFS).
- Hadoop fue desarrollado por el centro de investigación de Yahoo!, y es utilizado por una comunidad global de programadores y usuarios.
- Hadoop es la opción más conocida para implementar y ejecutar tareas MapReduce. Sin embargo, el framework también provee otras funcionalidades.

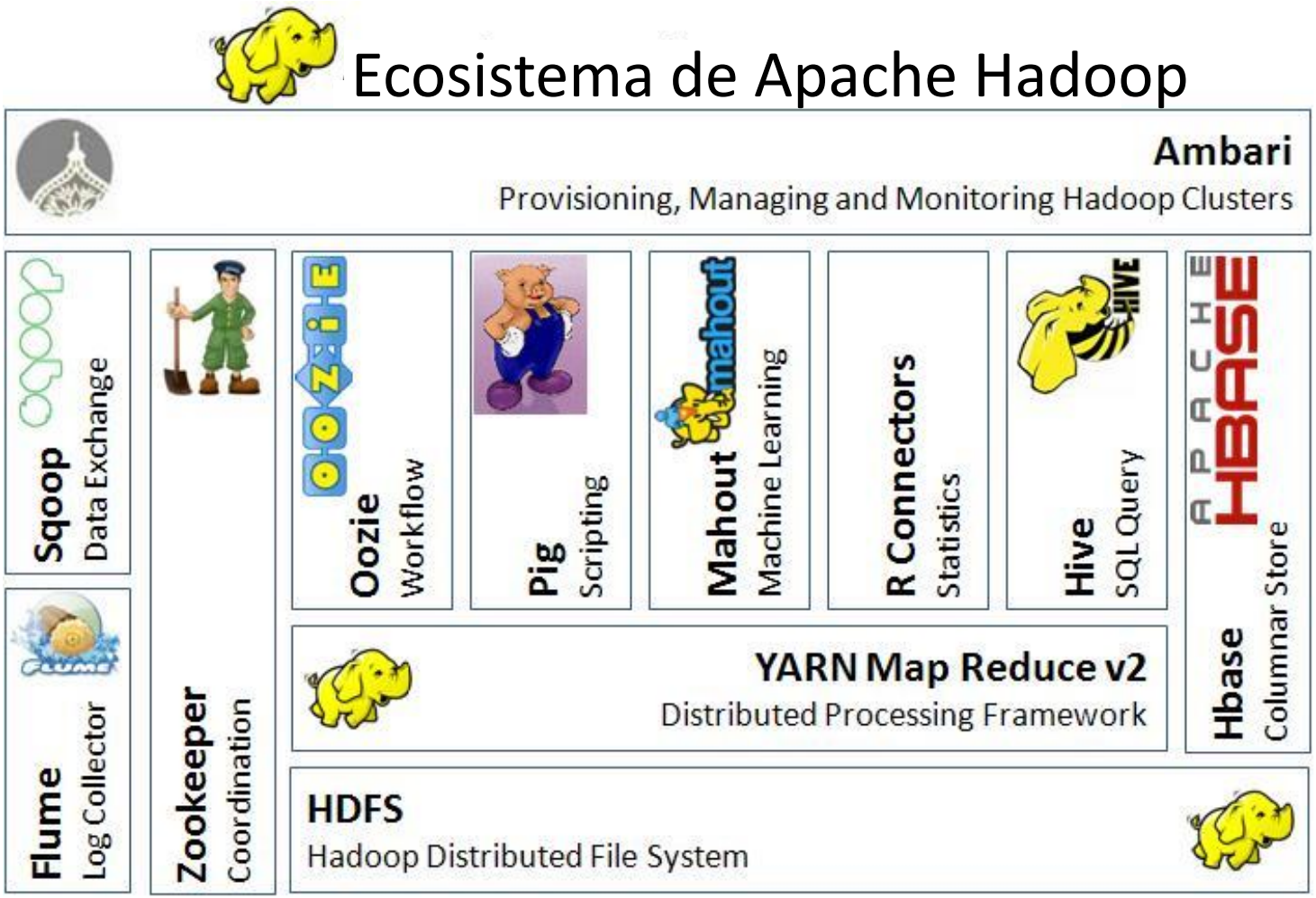
Apache Hadoop: componentes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Common: componentes e interfaces para sistemas distribuidos y entrada/salida (serialización, persistencia, RPC). Contiene los jars y scripts necesarios para la ejecución.
- Avro: sistema de serialización eficiente, RPC y persistencia de información.
- MapReduce: motor de ejecución de tareas MapReduce.
- Hadoop Distributed File System (HDFS): sistema de archivos distribuido de Hadoop.

Apache Hadoop: ecosistema

- Existe un gran conjunto de proyectos/frameworks basados en Hadoop



Apache Hadoop: frameworks

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Pig: lenguaje que permite ejecutar rutinas MapReduce de forma similar a sentencias en lenguaje SQL.
- Hive: datawarehouse distribuido inicialmente desarrollado por Facebook. Provee un lenguaje (HQL) muy similar a SQL que traduce en tiempo de ejecución a rutinas MapReduce (en general, utiliza varias rutinas encadenadas).
- Hadoop Database (HBase): base de datos distribuida orientada a columnas. Hbase utiliza HDFS y está basado en Google BigTable.
- Zookeeper: sistema de coordinación distribuido de alta disponibilidad.
- Sqoop: herramienta de migración de datos entre RDBMS y HDFS.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Hadoop: arquitectura

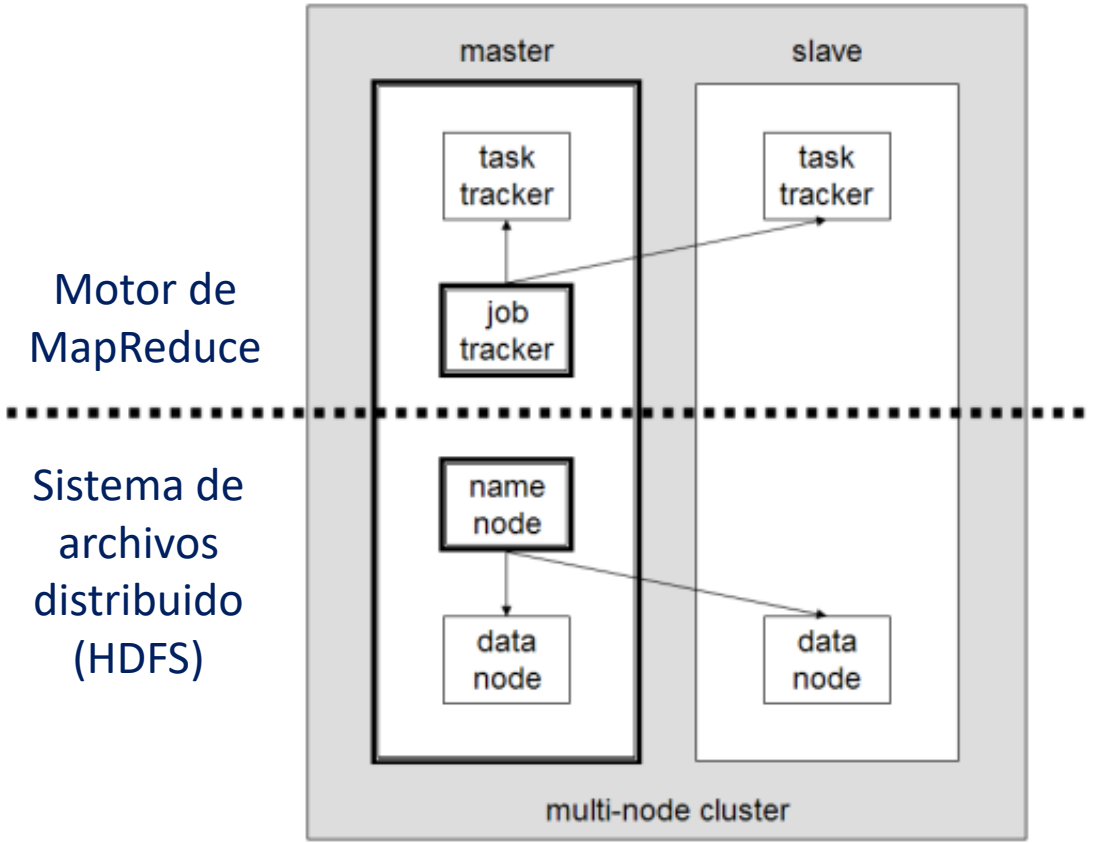
Hadoop: arquitectura

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Componentes (clusters virtuales para almacenamiento de datos y procesamiento distribuido):
  - Sistema de archivos distribuido
  - Motor de MapReduce: cluster de nodos, con master y slaves.
- Hadoop v1: manejo de recursos en los propios nodos de procesamiento.
- Hadoop v2: desacopla manejo de recursos de procesamiento de datos. Permite ejecutar otros tipos de aplicaciones (no solamente MapReduce).

Hadoop v1: arquitectura

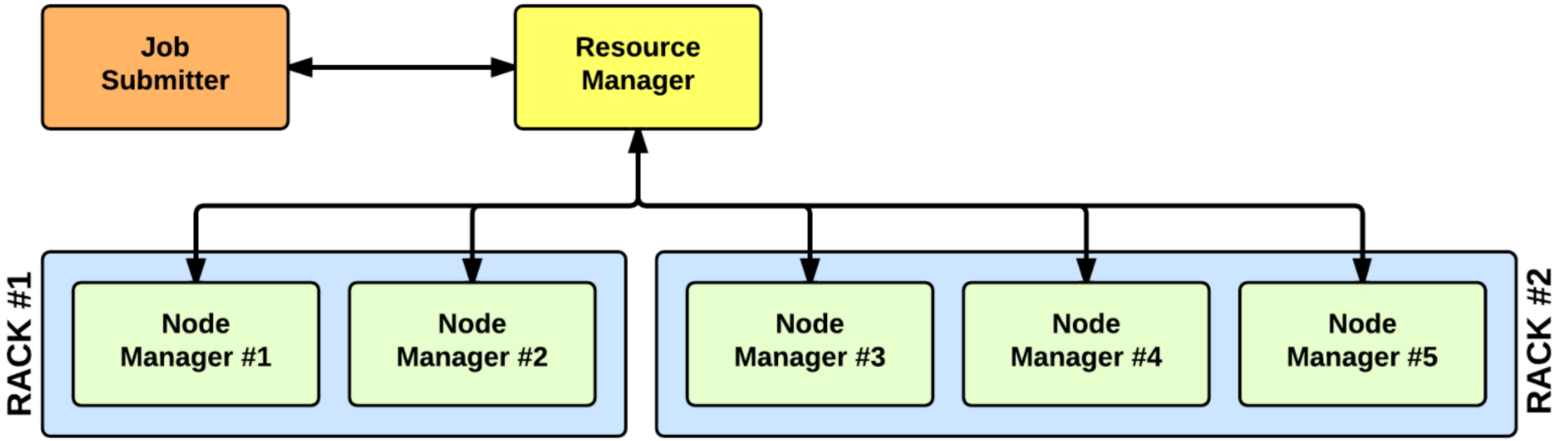
- En Hadoop v1 un cluster Hadoop típico incluye un nodo maestro y múltiples nodos esclavos.
- El nodo maestro se compone de un JobTracker (rastreador de trabajos), un TaskTracker (rastreador de tareas), un NameNode (nodo de nombres) y un DataNode (nodo de datos). Un esclavo (nodo de cómputo) está formado por un DataNode y un TaskTracker.



Hadoop v2: arquitectura

- En Hadoop v2 se desacopla el manejo de recursos del cómputo propiamente dicho. Se dividen las funciones del JobTracker en componentes separados.

- Entidades:
  - Job Submitter (el cliente)
  - Resource Manager (el master)
  - Node Manager (el slave)



Hadoop: protocolo de comunicaciones

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En Hadoop, todas las comunicaciones entre procesos se realizan utilizando una implementación de RPC personalizada para el framework:
  - Simple de modificar y extender.
  - Está definida a través de interfaces Java.
  - Las interfaces están implementadas a través de server object.
  - Client proxy objects son automáticamente creados.
- Todos los mensajes se originan en el cliente
  - De este modo se previenen ciclos y posibles deadlocks en las comunicaciones.

Hadoop: sistema de archivos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Hadoop Distributed File System (HDFS) es un sistema de archivos distribuido, escalable y portátil, escrito en Java, para Hadoop.
- Cada nodo en una instancia Hadoop típicamente tiene un único (o ningún) nodo de datos; un cluster de datos forma el cluster HDFS.
- Cada nodo sirve bloques de datos sobre la red, usando un protocolo de bloqueo específico para HDFS.
- El sistema de archivos usa TCP/IP para la comunicación, mientras que los clientes usan RPC para comunicarse entre ellos.
- HDFS almacena archivos grandes a través de múltiples nodos (hosts): el tamaño de particionamiento (*split*) en Hadoop HDFS es de 128 MB.
- La confiabilidad se implementa mediante replicación de datos en múltiples hosts (no se requiere almacenamiento RAID en ellos). Con el valor de replicación por defecto (3), los datos se almacenan en tres nodos: dos en el mismo rack, y otro en un rack distinto.

Hadoop: sistema de archivos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Existe un protocolo propietario entre nodos para realizar tareas específicas de HDFS como reequilibrar datos, mover copias, y conservar alta la replicación de datos.
- HDFS es portable entre varias plataformas, pero no cumple estrictamente con el estándar POSIX para sistemas de archivos.
- HDFS fue diseñado para proveer máxima eficacia y rendimiento en aplicaciones MapReduce desarrolladas sobre Hadoop y para gestionar archivos muy grandes.
- HDFS tradicional no proporciona alta disponibilidad (HDFS-HA está disponible en Hadoop 2).
- Aunque HDFS es el sistema de archivos nativo de Hadoop, el framework también soporta otros sistemas de archivos para cloud computing, como Amazon S3, CloudStore, FTP, HTTP(S).

HDFS: funcionalidades

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

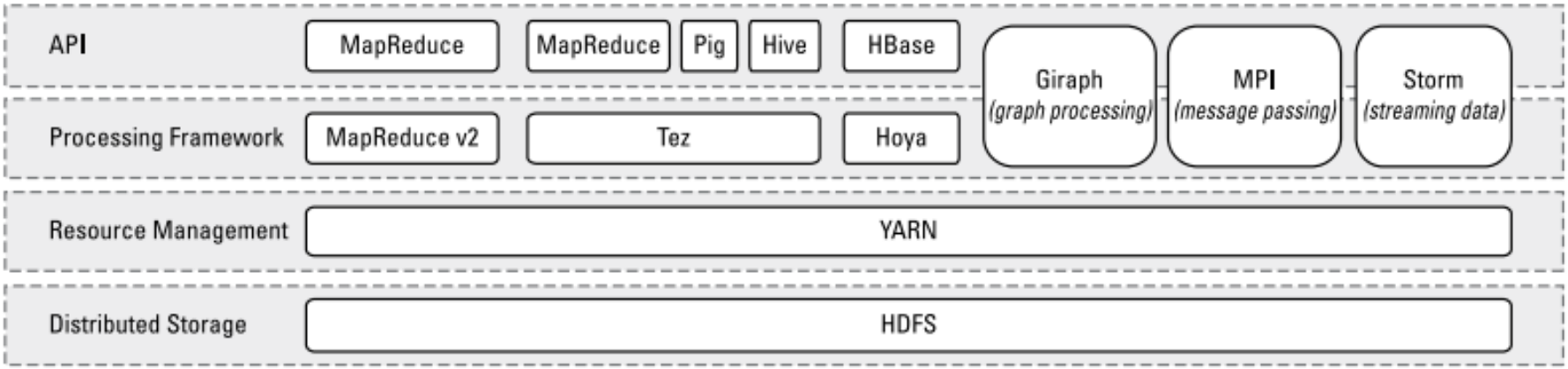
1. Tolerancia a fallos. El mecanismo de replicación proporciona un alto nivel de tolerancia a fallos utilizando almacenamiento redundante configurable.
2. Acceso eficiente a los datos. HDFS prove un gran ancho de banda para que las aplicaciones MapReduce desarrolladas sobre Hadoop puedan procesar grandes volúmenes de datos. El ancho de banda es más importante que la latencia de acceso a los datos, ya que en general el modelo de computación se aplica en procesos batch (fuera de línea) y no en tiempo real.
3. Modelo de coherencia simple. HDFS almacena los datos según el modelo write-once-read-many, soportando aplicaciones Hadoop donde los datos de entrada se escriben una vez y se leen tantas veces como sea necesario.

HDFS: funcionalidades

4. Conviene mover la computación y no los datos. Hadoop sigue la idea de que es menos costoso mover las aplicaciones/algoritmos que mover los datos. Esta idea es válida en general al considerar juegos de datos muy grandes. HDFS provee interfaces especializadas en este sentido.
5. Portabilidad entre hardware y software heterogeneo. HDFS es portable entre distintas plataformas (Java compatibles).

Hadoop v2: gestión y administración

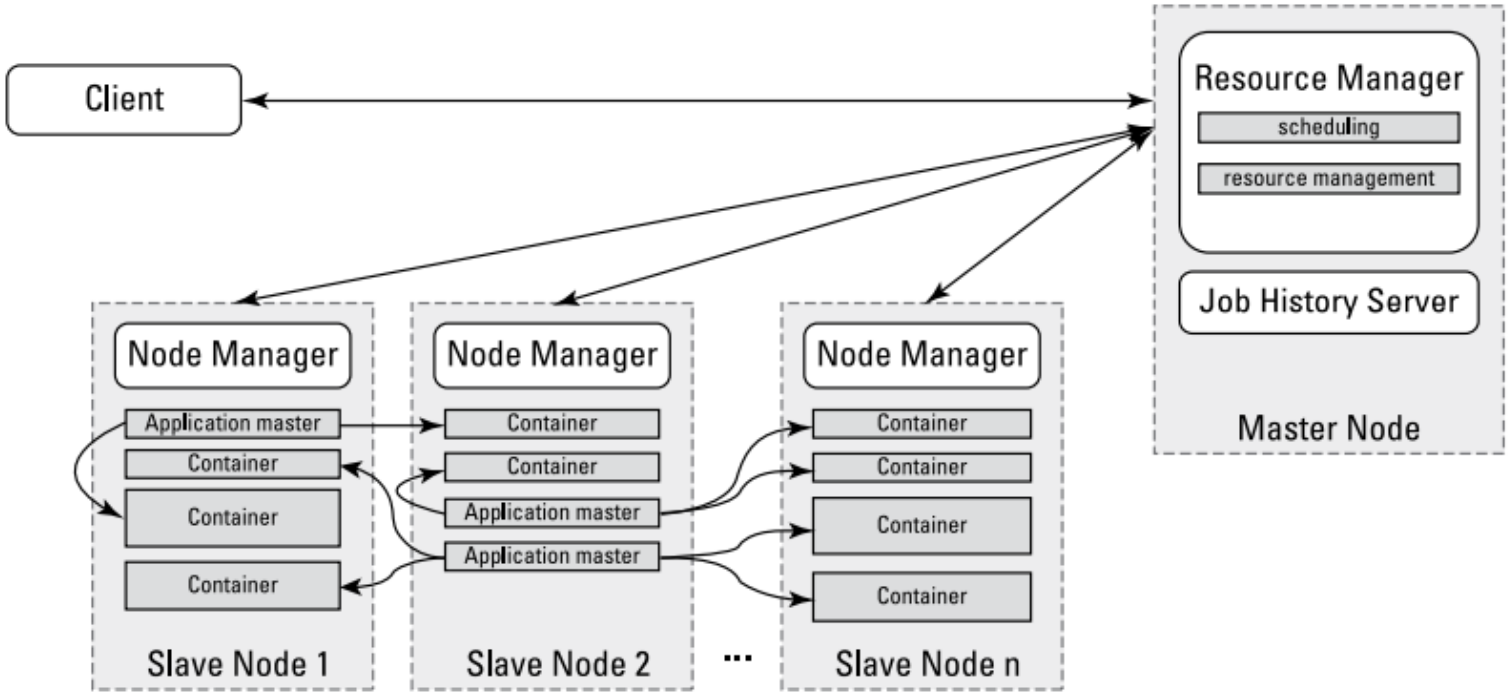
- YARN: sistema de administración de recursos, prove servicios de gestión y planificación para ejecutar otros tipos de tareas en un cluster Hadoop.



- La principal diferencia con la version 1 es que trabaja con contenedores (containers), unidades genéricas de recursos para ejecutar aplicaciones.
- A diferencia de los slots de JobTracker/TaskTracker los containers pueden ejecutar cualquier tipo de aplicaciones y no solo tareas Map y Reduce.
- Los containers pueden requerirse y dimensionarse con cualquier número de recursos (no como los slots, que son uniformes).

Hadoop v2: arquitectura

- Componentes: Resource Manager (scheduler, orquestador, similar a JobTracker), Node Manager (similar a TaskTracker), Application Master y Job History Server (ambos sin análogos en version 1).
- El Application Master maneja recursos y heartbeats para cada aplicación.



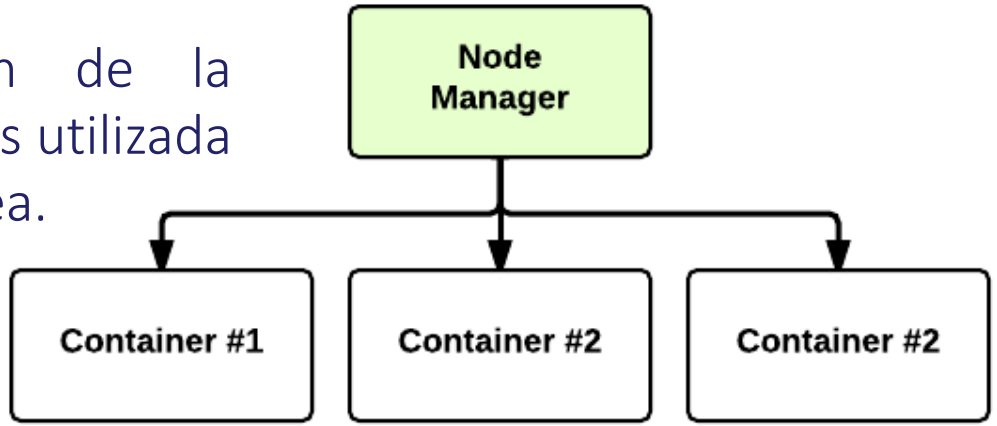
Hadoop v2: el motor MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El Resource Manager es el master del motor.
  - Existe un Resource Manager por cluster Hadoop.
  - Conoce la ubicación de los esclavos (rack awareness) y cuántos recursos dispone cada uno.
  - Ejecuta varios servicios, incluyendo el scheduler, los monitores de nodos y manejadores de eventos.
- El Resource Manager es el púnto único de falla.
  - Debe ejecutar en un nodo dedicado.
  - Utilizando Application Masters, YARN distribuye en el cluster la información y las tareas relacionadas con la ejecución de aplicaciones. De este modo se reduce la carga del Resource Manager y hace más simple su recuperación en caso de fallos.

Hadoop v2: el motor MapReduce

- El Node Manager es el esclavo de la infraestructura.
  - Existen muchos Node Managers por cluster.
  - Cuando inicia se anuncia al Resource Manager, al cual le envía heartbeats periódicamente.
  - Cada Node Manager ofrece recursos al cluster. La capacidad del Node Manager queda dada por la cantidad de memoria y el número de cores virtuales que ofrece.
  - En tiempo de ejecución, el scheduler utiliza la información sobre la capacidad para asignar tareas.
  - Utiliza containers: una fracción de la capacidad del Node Manager que es utilizada por el cliente para ejecutar una tarea.

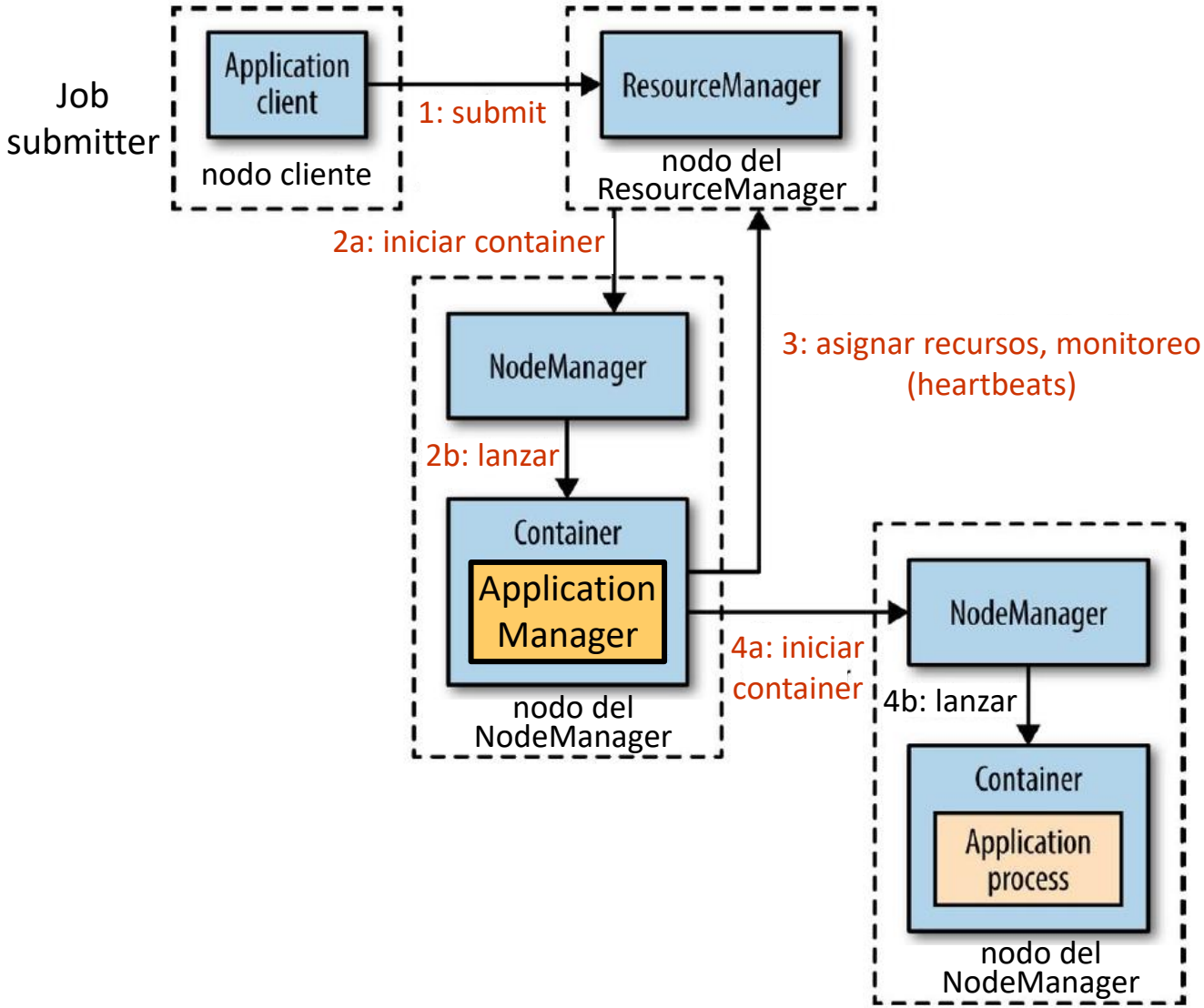


Hadoop v2: el motor MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El Application Master es responsable de la ejecución de cada aplicación.
  - Solicita containers al Resource Scheduler (en Resource Manager) y ejecuta programas específicos (main de clases Java) en los containers obtenidos.
  - Conoce la lógica de la aplicación, por lo cual es específico del framework que se ejecuta. El framework MapReduce prove su propia implementación de un Application Master.
- El Job Submitter envía trabajos al cluster.
  - El método submit crea una instancia de JobSubmitter y envía un job.
  - Solicita un application ID al Resource Manager, chequea directorios de entrada y de salida (que no debe existir), computa las particiones de entrada (input splits), copia los recursos necesarios para ejecutar el job al filesystem distribuido: jar (con muy alto factor de replicación), configuración del job, input splits.
  - Envía el job invocando submitApplication() en el Resource Manager.
  - waitForCompletion() chequea el progreso del trabajo (polling cada 1s).

Hadoop v2: el motor MapReduce



Hadoop v2: ejecución de una aplicación MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

1. Una aplicación cliente (Job Submitter) envía una solicitud al Resource Manager.
2. El Resource Manager solicita a un Node Manager crear una instancia de un Application Master. El Node Manager obtiene un container para la aplicación y la inicia.
3. El nuevo Application Master se inicializa y se registra con el Resource Manager.
4. El Application Master determina los recursos necesarios, examinando los archivos que la aplicación necesitará del NameNode y calculando el número de tareas Map y Reduce, y los solicita al Resource Manager. El Application Master envía mensajes (heartbeat) al Resource Manager con una lista de recursos requeridos y sus modificaciones (por ejemplo, una solicitud de fin).
5. El Resource Manager acepta el pedido y lo encola junto con otros pedidos.
6. Cuando se identifican recursos disponibles en los nodos slave, el Resource Manager otorga al Application Master (containers) en los nodos slave. El AM solicita el container (con todos sus detalles) al NM, que lo crea y lo inicia.

Hadoop v2: ejecución de una aplicación MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

7. La aplicación ejecuta en el container y el Application Master monitorea su progreso. Si se detecta un error del container se reinicia la ejecución en otro container (hasta un número máximo de reintentos). El Application Master también puede enviar una señal de fin al Node Manager para terminar un container (por un cambio de prioridades o porque la aplicación finalizó).
8. Para una aplicación MapReduce, luego que finalizan las tareas map, el Application Master solicita recursos para ejecutar las tareas reduce para procesar los resultados intermedios.
9. Cuando todas las tareas finalizan, el Application Master envía los resultados a la aplicación cliente, informa al Resource Manager que la aplicación finalizó con éxito, se desregistra del Resource Manager y finaliza su ejecución.

Hadoop: tolerancia a fallos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El mecanismo de tolerancia a fallos en Hadoop se basa en reasignar tareas cuando una ejecución particular falla.
  - Cuando el Application Master recibe una notificación de fallo de una tarea la reasigna. Intenta evitar el uso de un Node Manager que falló previamente.
  - El máximo número de intentos de ejecución de una tarea se indica con mapreduce.map.maxattempts y mapreduce.reduce.maxattempts.
  - Algunas aplicaciones pueden no abortar si unas pocas tareas fallan. El porcentaje de tareas que pueden fallar sin abortar se indica por mapreduce.map.failures.maxpercenty mapreduce.reduce.failures.maxpercent.
  - Las tareas pueden ser matadas por ejecución especulativa o porque un Node Manager fue marcado por fallos. Tareas matadas no cuentan en los intentos.
- El esquema funciona correctamente en cluster locales, pero no es adecuado para grandes sistemas distribuidos
  - El mecanismo de reasignar trabajos puede ser muy ineficiente.
  - No es aplicable para entornos pervasivos, dinámicos y/o voluntarios.

Hadoop: tolerancia a fallos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Fallos del Application Master
  - Se reintentan (propiedad mapreduce.am.max-attempts), por defecto es 2.
  - Debe modificarse junto a yarn.resourcemanager.am.max-attempts.
- Fallos del Node Manager
  - Por caída o por ejecución muy lenta (no envía heartbeat en 10 minutos): se elimina del pool de recursos para asignar. Tareas y Application Master ejecutando en ese nodo se recuperan como se explicó.
  - Maps que corresponden a jobs que no finalizaron pueden ser reejecutados (la salida intermedia puede no ser accessible a los nuevos reducers).
  - Node Managers pueden ser colocados en una lista negra (por parte del Application Master) si tienen fallos muy frecuentes.
- Fallos del Resource Manager
  - Serio, no se pueden ejecutar jobs containers.
  - Replicación es necesaria para lograr alta disponibilidad: un RM secundario y datos en un sistema confiable (ZooKeeper, HDFS), para recuperación.

Hadoop: HDFS + MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El framework Hadoop trabaja sobre el sistema de archivos HDFS y el motor de MapReduce, a los que considera como 'clusters virtuales' para la ejecución de aplicaciones.

- HDFS y Map-Reduce están acoplados, y fueron diseñados para trabajar conjuntamente.
  - HDFS para manejar datos (se almacenan y acceden como archivos y directorios).
  - Map-Reduce para la ejecución de trabajos y el scheduling.

- De todas formas, nada impide que una tarea MapReduce en particular consuma información de otras fuentes (ej: Hbase, REST API, etc).

Modos de Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

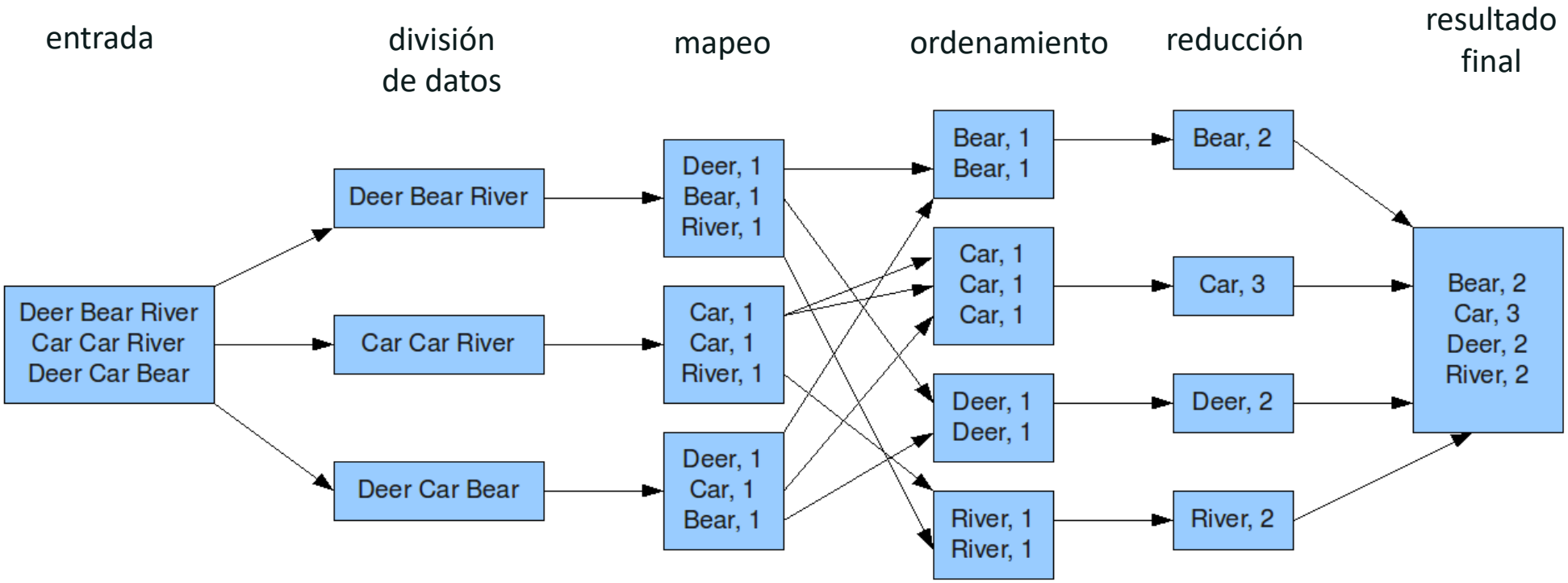
- Hadoop propone tres modos de ejecución:
  1. Local (standalone)
     - Útil para paqueños testeos y debug de aplicaciones.
  2. Local (seudo-distribuida)
     - Las tareas se ejecutan en diferentes máquinas virtuales.
     - Se configura (casi) del mismo modo que una instalación completa en modo cluster.
  3. Totalmente distribuida (cluster)
     - Las máquinas del cluster se configuran individualmente.
     - Se utiliza una topología master-slave para los nodos que proveen recursos y almacenamiento.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Hadoop: ejemplo de aplicación MapReduce

Hadoop: word count

- Ejemplo de implementación de aplicación MapReduce para conteo de palabras en un archivo de texto en Hadoop.



Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Creación del trabajo (job) con la clase main

```
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```java
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Clave y valor de la salida a utilizar: palabra (texto) y contador (entero).

Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```java
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Caminos para archivos de entrada y salida

Word count: main

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```java
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "word count");
  job.setJarByClass(WordCount.class);
  job.setMapperClass(TokenizerMapper.class);
  job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(IntSumReducer.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Ejecutar el job MapReduce y esperar que finalice

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Extiende la clase org.apache.hadoop.mapreduce.Mapper, que tiene 4 tipos genéricos: KEY_IN,VAL_IN,KEY_OUT,VAL_OUT

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Implementa la función map(KEY_IN key, VAL_IN value, Context context) que recibe un par clave-valor para el Mapper y retornará pares clave-valor

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Cada mapper recibe datos (líneas) de entrada, de acuerdo al formato (InputFormat) que se utilice. Por defecto se envían 128MB del archivo de entrada a cada mapper, partiéndolo por líneas. La clave es el offset (en bytes) desde el comienzo del archivo.

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Separa la línea recibida por espacio. StringTokenizer es un iterador que permite recorrer los valores separados.

Word count: mapper

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

Por cada token (palabra) se genera un tipo de dato Text y se retorna <palabra, 1>.

Word count: reducer

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Extiende la clase org.apache.hadoop.mapreduce.Reducer que recibe como clave una palabra y un iterador con todos los valores emitidos por el Mapper

```
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

Word count: reducer

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```java
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

Implementa el método reduce(KEY_IN key, Iterable<VAL_IN> values, Context); KEY_IN es el tipo de dato de la clave emitida por el Mapper, VAL_IN es el tipo de dato del valor emitido por el Mapper.

Word count: reducer

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

```
public static class IntSumReducer
    extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

Los valores recibidos son una lista de unos. El Reducer suma los valores y escribe como salida <palabra, suma de ocurrencias>

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El ejemplo, de implementación concisa y simple, está disponible en https://wiki.apache.org/hadoop/WordCount

Word count en Hadoop

- Ejemplo de implementación en C++ usando Hadoop Pipes

```
#include "hadoop/Pipes.hh"
#include "hadoop/TemplateFactory.hh"
#include "hadoop/StringUtils.hh"

class WordCountMap: public HadoopPipes::Mapper {
public:
  WordCountMap(HadoopPipes::TaskContext& context){}
  void map(HadoopPipes::MapContext& context) {
    std::vector<std::string> words =
      HadoopUtils::splitString(context.getInputValue(), " ");
    for(unsigned int i=0; i < words.size(); ++i) {
      context.emit(words[i], "1");
    }
  }
};
```

https://wiki.apache.org/hadoop/C++WordCount

Word count en Hadoop

- Ejemplo de implementación en C++ usando Hadoop Pipes

```
class WordCountReduce: public HadoopPipes::Reducer {
public:
  WordCountReduce(HadoopPipes::TaskContext& context){}
  void reduce(HadoopPipes::ReduceContext& context) {
    int sum = 0;
    while (context.nextValue()) {
      sum += HadoopUtils::toInt(context.getInputValue());
    }
    context.emit(context.getInputKey(),HadoopUtils::toString(sum));
  }
};

int main(int argc, char *argv[]) {
  return HadoopPipes::runTask(HadoopPipes::TemplateFactory<WordCountMap,
                              WordCountReduce>());
}
```

https://wiki.apache.org/hadoop/C++WordCount

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de implementación en Python
- Mapper.py

```
#!/usr/bin/env python
import sys
# la entrada llega desde STDIN (entrada estándar)
for line in sys.stdin:
    # remover espacios en blanco al inicio y al final (si los hubiera)
    line = line.strip()
    # separar la línea en palabras
    words = line.split()
    # incrementar contadores
    for word in words:
        # escribir los resultados en STDOUT (salida estándar);
        # la salida será la entrada de la etapa de Reduce (reducer.py)
        print '%s\t%s' % (word, 1)
```

http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/

Word count en Hadoop

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de implementación en Python: Reducer.py

```
#!/usr/bin/env python
import sys
from operator import itemgetter
current_word = None
current_count = 0
word = None
for line in sys.stdin: # la entrada llega desde STDIN (entrada estándar)
    line = line.strip() # remover espacios en blanco al inicio y al final (si los hubiera)
    word, count = line.split('\t', 1) # analizar a entrada que llega desde mapper.py
    # convertir count a entero (llega como  string)
    try: count = int(count) except ValueError:
         continue # si hay error se ignora la línea (count no era un número)
    # este if funciona porque Hadoop ordena la salida de los mappers por clave (cada palabra)
    if current_word == word:
         current_count += count
    else: if current_word:
              print '%s\t%s' % (current_word, current_count) # escribir a salida estándar
    current_count = count
    current_word = word
# imprimir las estadísticas de la última palabra
if current_word == word: print '%s\t%s' % (current_word, current_count)
```

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

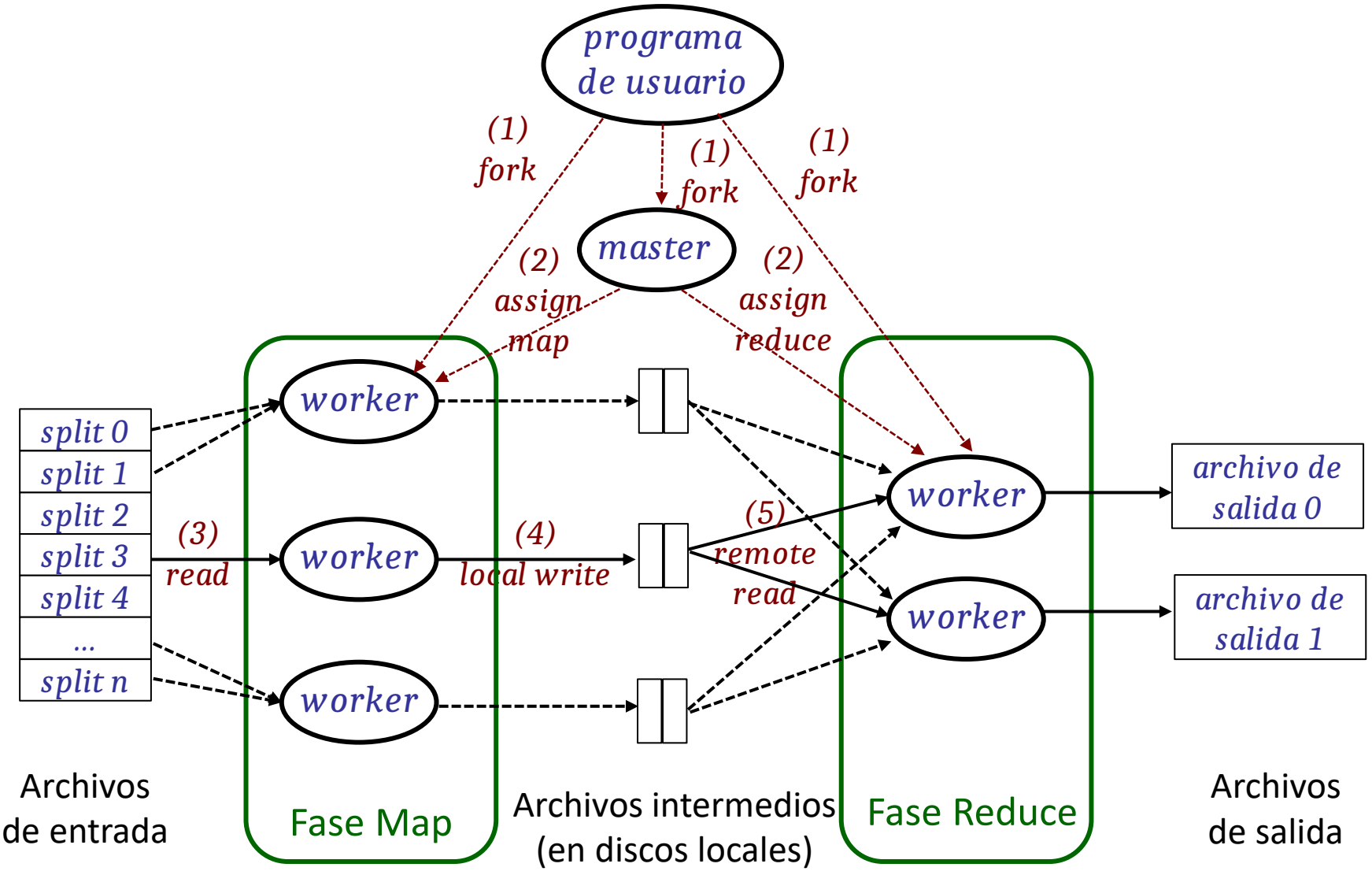
MapReduce en Hadoop

MapReduce en Hadoop

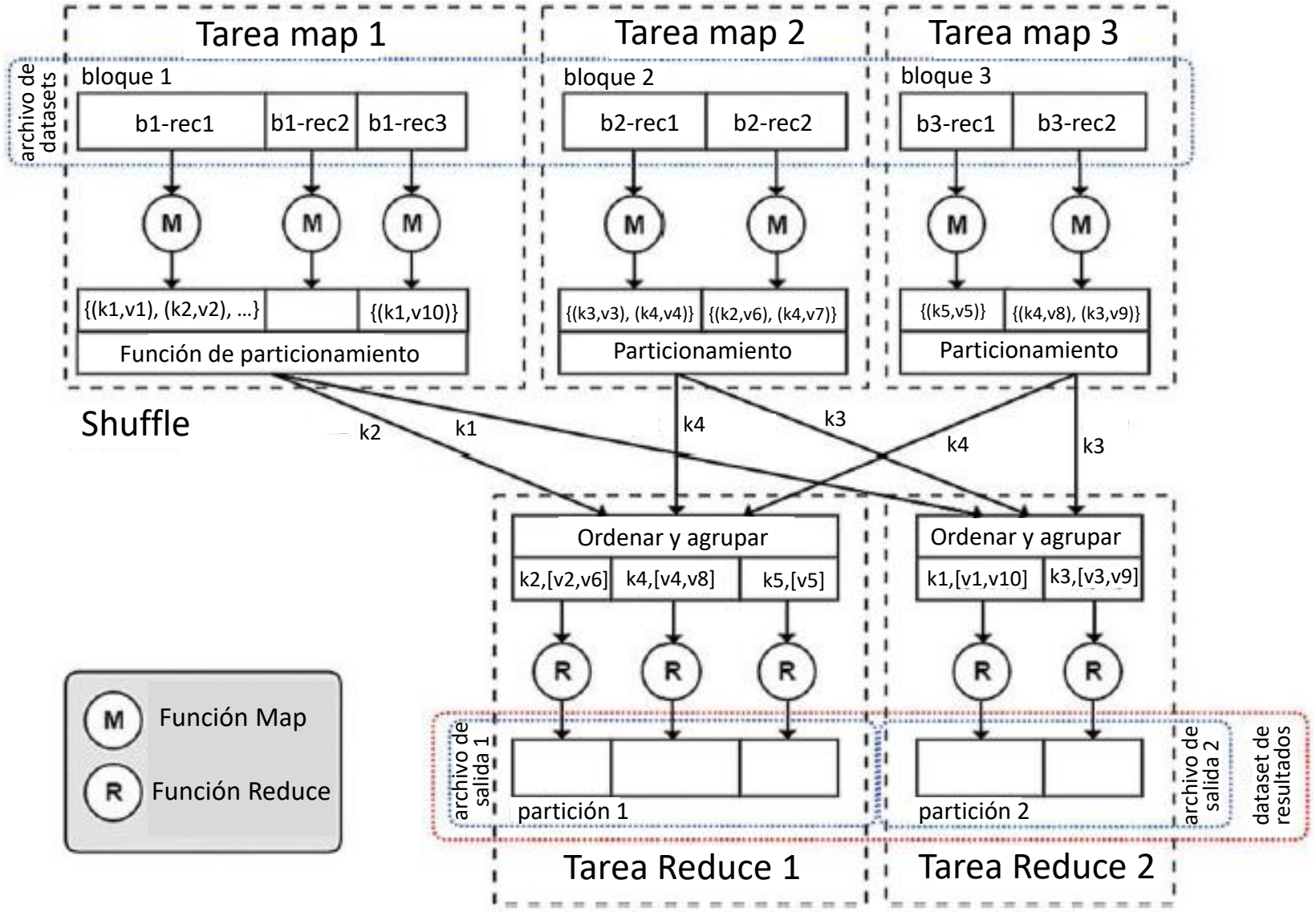
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

1. Esquema genérico
2. Etapas
3. Map (entrada, particionamiento, shuffle)
4. Reduce
5. Combiner
6. Implementación

MapReduce en Hadoop: esquema genérico



MapReduce en Hadoop: etapas



MapReduce en Hadoop

- La tarea Map-Reduce queda definida por las dos funciones:
  - map: (f,{(k1;v1)}) →{(k2;v2)}
  - reduce: (k2;{v2}) →{(k3;v3)}
- Los tipos corresponden a

```
public class Mapper<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  public class Context extends MapContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    // ...
  }
  protected void map(KEYIN key, VALUEIN value,
      Context context) throws IOException, InterruptedException {
    // ...
  }
}

public class Reducer<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
  public class Context extends ReducerContext<KEYIN, VALUEIN, KEYOUT, VALUEOUT> {
    // ...
  }
  protected void reduce(KEYIN key, Iterable<VALUEIN> values,
      Context context) throws IOException, InterruptedException {
    // ...
  }
}
```

MapReduce en Hadoop: map

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En Hadoop, una tarea Map tiene cuatro fases: input, mapper, combiner y partitioner.
- Input utiliza las funciones input format y record reader para leer los registros en formato clave-valor para el procesamiento.
- Map aplica función(es) a cada par clave-valor en una porción del dataset.
  - Si el dataset está almacenado en un sistema de archivos como HDFS or S3, la porción es un bloque en el filesystem.
- Si hay n bloques de datos en el dataset de entrada existirán n tareas Map (Mappers) → Los datos de entrada determinan el número de Mappers !
  - El programador puede especificar el número de tareas Reduce (reducers).
- La fase Map está diseñada para no compartir estado ni datos entre procesos: cada Mapper procesamiento sus datos independientemente.
- Se asegura que cada registro es procesado una única vez (salvo en caso de fallos o de ejecución especulativa).

MapReduce en Hadoop: map

- En el ejemplo, tres Mappers operan sobre tres bloques en el filesystem (block1, block2 y block3). Cada Mapper ejecuta la función map() para cada registro (clave-valor), b1-rec1, b1-rec2, etc.
- La función map() recibe un par clave-valor y emite como salida cero o más pares clave-valor, que son datos intermedios del procesamiento.

  `map (in_key,in_value)` → `list(intermediate_key,intermediate_value)`
- Ejemplos de función map:
  - Filtrar mensajes de error en un log:

    ```
    let map(k,v) = if (ERROR in v) then emit(k,v)
    ```
  - Convertir valores a minúsculas: `let map(k, v) = emit(k,v.toLower())`
  - Las funciones map pueden concatenarse y combinarse con operadores lógicos.
- El Mapper recolecta las listas de pares clave-valor emitidas por cada función map en una única lista agrupada por la clave del registro intermedio. Esta lista se pasa como entrada a la función de particionamiento.

MapReduce en Hadoop: entrada

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La entrada a una tarea MapReduce es un conjunto de archivos almacenados en HDFS. Los archivos son formateados por un *input format* que define cómo se divide un archivo en *input splits*.
  - Un input split es una partición del archivo de entrada a la tarea map.
- Se pueden implementar input format personalizados. El input format de texto (TextInputFormat) de Hadoop es útil para la mayoría de tareas de procesamiento de datos

- El *record reader* transforma un input split generado por el input format en registros. El record reader parsea los datos en registros pero no parsea los registros. Pasa los datos al mapper en formato clave-valor
- Generalmente la clave es información de posición (offset) en el archivo de entrada (tipo long) y el valor es una partición (chunk) de datos (texto).
- Se pueden implementar record readers personalizados, el que se incluye por defecto en Hadoop es útil para la mayoría de tareas de procesamiento de datos.

MapReduce : particionamiento

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La función de particionamiento (*partitioner*) se encarga de que cada clave intermedia y su lista de valores se envíe a una y solo una tarea de Reduce.
- La implementación más común es mediante un particionamiento por correspondencia ('hash'), que crea un hash (identificador único) para la clave y divide el espacio de claves 'hasheadas' en n particiones.
- El número de particiones está relacionado con el número de tareas reduce (Reducers) que especifica el programador.
- Particionamientos específicos pueden ser implementados dependiendo de la lógica del procesamiento a realizar.
  - Por ejemplo, particionar datos cronológicos por año, mes o semana.
- La función de particionamiento se invoca para cada clave y su salida corresponde al Reducer encargado de procesar el resultado.
  - Típicamente, retorna un entero entre 0 y n-1, sindo n el número de Reducers especificado por el programador.

MapReduce: shuffle

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El procesamiento realizado por los Mappers es independiente y no require comunicar datos ni sincronizar procesos.
- La etapa de recombinar claves intermedias y sus valores asociados provenientes de multiples Mappers (en general, ejecutando en diferentes recursos de cómputo) require sincronizar y comunicar datos.
- La etapa shuffle toma la salida de cada Mapper y la envía a cada Reducer especificado por la función de particionamiento.
- En general, esta es la tarea más costosa en una aplicación MapReduce, ya que require la transferencia física de datos entre nodos a través de la red.
- Shuffle incluye el ordenamiento (por clave), mediante 'merge' de los datos recibidos de diferentes Mappers.
- Shuffle es asincrónico, permite solapar cómputo y comunicaciones.
- Shuffle soporta encriptado (HTTPS/SSL), pero reduce el desempeño.

MapReduce: reduce

- Cada Reducer ejecuta la función reduce() para cada clave intermedia y su lista de valores asociados.
- Su salida es un conjunto (puede ser vacío) de pares clave-valor, que pueden ser parte de la salida del procesamiento o pueden ser entrada de una nueva fase Map en un workflow o pipeline.

```
reduce(key,list(intermediate_value)) → key, out_value
```

- Las funciones reduce() usualmente incluyen operaciones de agregación como sumas, conteos, promedios, etc.
- Ejemplo de función reduce() de suma:

```
let reduce(k,list<v>) = sum=0 for int i in list<v>:sum+=i emit(k, sum)
```

- El conteo se implementa sumando el valor 1 por cada elemento a contar.

MapReduce en Hadoop: reduce

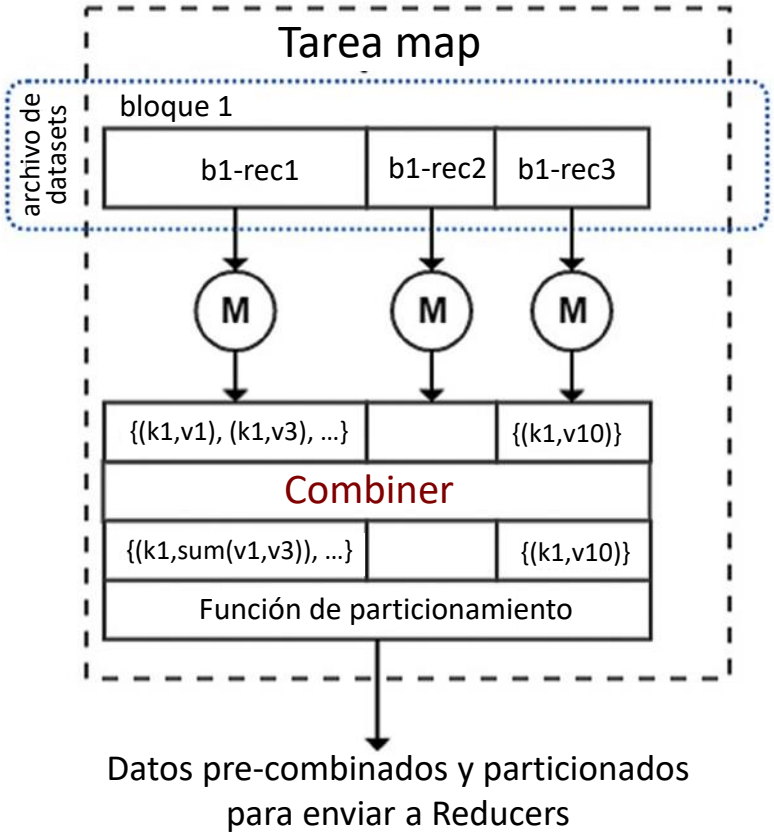
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Normalmente el reducer retorna un único par clave/valor para cada clave que procesa. Sin embargo, los pares clave valor pueden ser expandidos tanto cómo se necesite.
- Cuando una tarea reduce finaliza, retorna un archivo de resultados y lo almacena en HDFS.
- Como se presentó en el ejemplo, HDFS replica automáticamente los resultados.

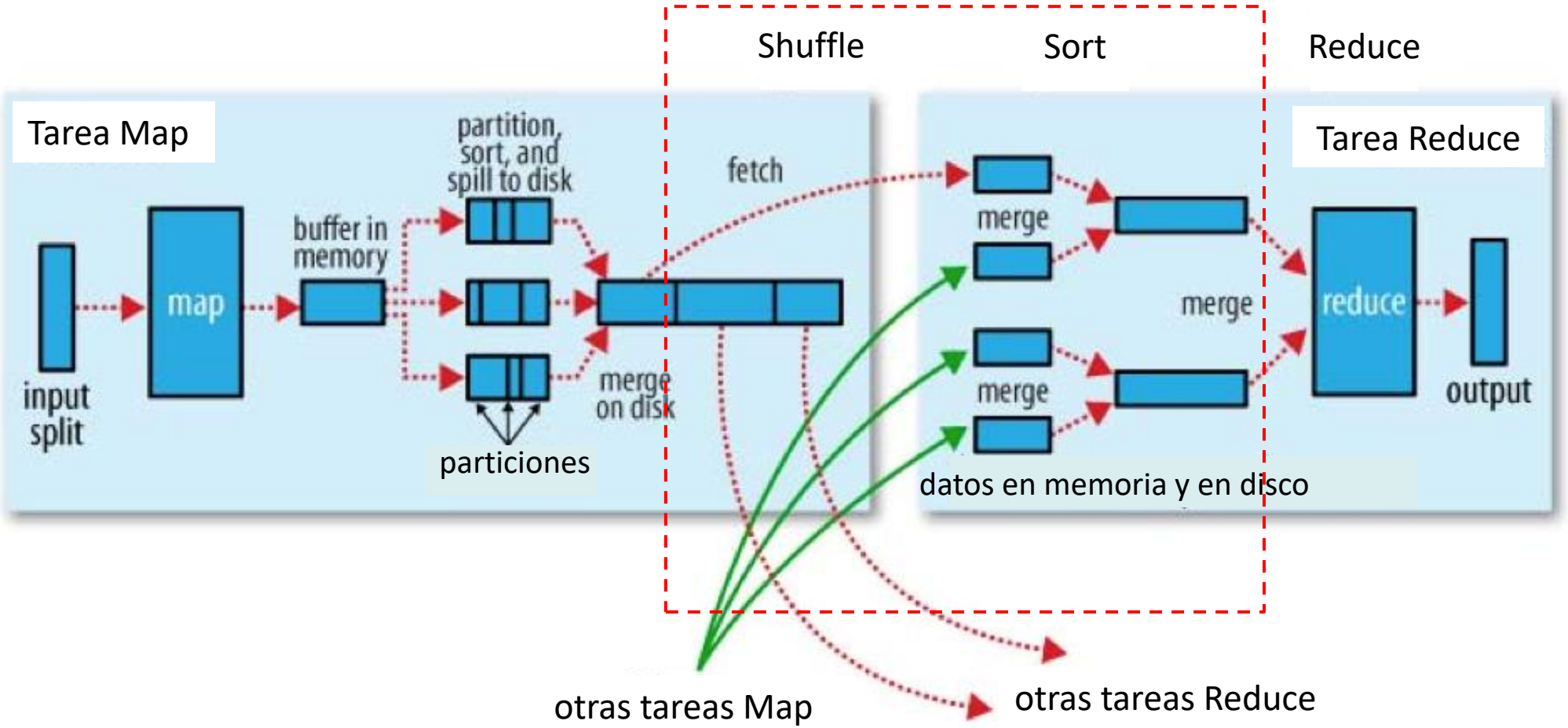
MapReduce: combiner

- Es un reducer localizado (opera sobre datos generados en un solo recurso de cómputo), que permite combinar información al final de la etapa Map, para reducir la cantidad de información enviada a la etapa Reduce (disminuye los tiempos de transferencia de datos).

- Si la función de reducción en conmutativa y asociativa, es posible utilizar el mismo Reduce como Combiner.
- En general aporta importantes mejoras al desempeño del algoritmo implementado, al reducir el ancho de banda de las comunicaciones.



MapReduce: etapas en Hadoop



MapReduce en Hadoop: componentes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Driver (mandatorio): aplicación (shell) invocada desde el cliente. Configura la clase MapReduce Job (no personalizable) y realiza el submit al Resource Manager (o JobTracker en Hadoop 1).
- Clase Mapper (mandatoria): implementa la definición de los formatos clave/valor para entrada y salida utilizados al procesar registros. La clase tiene un único método map, donde se codifica el procesamiento y la clave/valor de salida.
- Clase Reducer (opcional para aplicaciones map-only): implementa la operación de reducción.
- Clase Combiner (opcional): generalmente definida como el reducer, pero puede ser diferente.
- Clase Partitioner (opcional): para personalizar el particionamiento (ejemplo, sort secundario, datos dispersos, etc.)
- Clases RecordReader y RecordWriter (opcionales): para lectura y escritura de formatos personalizados.

MapReduce en Hadoop: componentes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Desde el driver se puede utilizar la API de MapReduce, que incluye métodos de fábrica para crear instancias de todos los componentes de la aplicación.
- La API genérica Hadoop Streaming permite utilizar código implementado en otros lenguajes (comúnmente C, Python y Perl).
- Ventaja: permite a usuarios con poco conocimiento de Java desarrollar código MapReduce.
- Desventaja: requiere capas adicionales de abstracción para implementar el streaming, que impacta en el desempeño (tiempo de ejecución y uso de memoria).
- Las funciones map y reduce pueden implementarse con Hadoop Streaming, pero Record readers/writers y partitioners deben estar escritos en Java. Por este motivo las aplicaciones Hadoop Streaming son utilizables para procesar datos de texto solamente.

MapReduce en Hadoop: detalles

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Map

- Las tareas map no escriben sus salidas a disco. Utilizan buffers circulares en memoria (de tamaño io.sort.mb, por defecto 100 MB). Cuando se supera una cota (io.sort.spill.percent, por defecto 80%) un nuevo thread escribe a disco local (en mapred.local.dir) de modo Round-Robin. Luego, la escritura a buffer y a disco continuan en paralelo.
- El thread que escribe divide los datos en particiones para los reducers, en cada partición los datos se ordenan.
- Si hay combiner, ejecuta sobre los datos ordenados y produce datos más compactos y ordenados, de modo de transmitir menos al reducer.
- La salida de los mappers pueden comprimirse para más eficiencia (debe habilitarse en settingmapred.compress.map.output).
- Las salidas de los mappers quedan disponibles para los reducers sobre HTTP.

MapReduce en Hadoop: detalles

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Reduce

- El reducer verifica qué particiones necesita. La copia se inicia apenas están disponibles, con threads que copian en paralelo (mapred.reduce.parellel.copies, por defecto 5).
- La copia se realiza directamente a la memoria de la JVM del reducer (a disco para gran volumen de datos). Un thread que ejecuta en background los combina en archivos grandes ordenados. Si las salidas de los mappers están comprimidas, deben descomprimirse primero.
- Cuando se copian todos los datos se inicia la etapa de sort (en realidad es un merge, porque los datos ya fueron ordenados en el mapper).
- El merge se realiza en rondas: salidas de mappers/archivos a combinar (io.sort.factor, por defecto 10). El merge envía directamente los archivos a memoria del reducer para no escribir a disco.
- La reducción trabaja sobre su entrada y escribe la salida al filesystem, típicamente HDFS.

MapReduce en Hadoop: eficiencia

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- La etapa crítica es Shuffle and Sort. Claves para mejorar la eficiencia:
  - Usar Combiner para reducer los datos transferidos a los reducers.
  - Elegir el número óptimo de reducers. Muchos datos: no usar un solo reducer. Muchos reducers implican muchas particiones en los mappers.
  - Iniciar los reducers luego que un porcentaje de mappers ha finalizado. Se selecciona con mapreduce.job.reduce.slowstart.completedmaps (defecto 0.05). Reducers que se inician más temprano quedarán ociosos.
  - Comprimir la salida de los mappers.
  - Configurar parámetros: [mapreduce.task.io].sort.mb/io.sort.factor, mapreduce.map.sort.spill.percent, mapreduce.shuffle.max.threads, [mapreduce.reduce.shuffle].input.buffer.percent/merge.percent,, [mapreduce.reduce].input.buffer.percent/shuffle.parallelcopies, etc.
  - Lista completa en https://hadoop.apache.org/docs/r2.4.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml.

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Patrones de MapReduce

Patrones de MapReduce

- Patrones: modelos para soluciones reutilizables a problemas similares. *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma et al. 1995).
- Patrones de MapReduce: enfocados en el procesamiento de grandes volúmenes de información.
- Independientes del dominio de aplicación.
- No son 'recetas', no proporcionan una solución 'mágica' a un problema.
- Son modelos genéricos que deben instanciarse para resolver cada problema en particular.
- Proporcionan la base para herramientas de alto nivel (incluyendo Pig, Hive, etc.)

Patrones de MapReduce

- Patrones de conteo/suma/índices
- Patrones de filtro
- Patrones de organización de datos
- Patrones de combinación (join)
- Metapatrones
- Patrones de entrada y salida

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de conteo/suma/índices:
  - Algoritmos que cuentan o retornan un conjunto resumido de datos.
  - Idea: agrupar registros por un campo y calcular una función numérica de agregación por grupo. Los datos deben ser numéricos y admitir agregación.
  - Ejemplos: cálculo de índices, conteo de determinado tipo de datos, cálculo de datos estadísticos.
  - Ejemplos concretos: cálculo de visitas a una determinada página web, cálculo de ganancias en una determinada transacción, etc.
  - Similar a SELECT MIN(col1), COUNT(*) FROM Table GROUP BY col2; (de SQL) o a b = GROUP a BY col2; c = FOREACH b GENERATE group, MIN(a.col1); de Pig
  - Patrones incluidos: *numerical summarizations*, *inverted index*, *counting with counters*.
- Permiten obtener una vision de alto nivel de valores numéricos resultantes del análisis de los datos.

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de filtro:
  - Se basan en encontrar un subconjunto de datos del conjunto inicial (sin modificarlo).
  - Ejemplos: descartar datos no válidos o no útiles, obtener listas de mejores rankeados, obtener información sobre un determinado tema o generada en un determinado período.
  - Se basan en resumir información para poder analizar un conjunto reducido de los datos. Se pueden utilizar funciones y bibliotecas de Java para expresiones regulares.
  - Similar a SELECT * FROM table WHERE value < 3; (de SQL) o a b = FILTER a BY value < 3; (de Pig)
  - Patrones incluidos: *sampling, filtering, bloom filtering, top ten, distinct*.
- Permiten filtrar datos de interés

Patrones de MapReduce

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Patrones de organización de datos:
  - Se basan en transformar o reorganizar un conjunto de datos (creando nuevos datos con diferente estructura).
  - En ocasiones, las tareas MapReduce son utilizadas para realizar un procesamiento primario de la información (‘masticar’) y organizarla de forma que luego otros procesos puedan procesarla y manipularla eficientemente.
  - Ejemplos: transformar datos no estructurados a estructurados, particionar, ordenar, barajar, etc.
  - Patrones incluidos: *structured to hirarchical pattern, partitioning and binning patterns, total order sorting, shuffling patterns.*
- Permiten modificar la organización de los datos

Patrones de MapReduce

- Patrones de combinación (join)
  - Se basan en realizar cruzamientos de información entre distintos conjuntos de datos.
  - En una base de datos tradicional el join es una tarea sencilla, sin embargo, al trabajar con datos no estructurados es una tarea compleja y muchas veces no muy eficiente.
  - Inner join, outer join, antijoin, product cartesiano.
  - Patrones: *reduce side join*, *reduce side join with bloom filter*, *replicated join*, *composite join*, *cartesian product*.

- Permiten combinar diferentes conjuntos de datos, procedentes de diversos repositorios o de diversas fuentes.

Patrones de MapReduce

- Metapatrones:
  - Definen patrones combinando los patrones básicos comentados previamente.
  - Proponen diferentes estrategias para el manejo de los patrones y cómo relacionarlos entre si.
  - En general, una única tarea MapReduce no realiza todo el trabajo. Es necesario encadenar varias tareas, utilizando diferentes patrones.
  - Patrones: *job chaining* (encadenamiento), uno de los más utilizados en MapReduce y *job merging*.
  - El encadenamiento se implementa con un *master driver*, que lance varios drivers (mains) para los diferentes jobs.
- Permiten resolver problemas complejos combinando multiples patrones simples.

MapReduce y bases de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Diferencias entre problemas que se resuelven con MapReduce y con una base de datos relacional

Característica	RDBMS tradicionales	MapReduce
Tamaño de datos	gigabytes	petabytes
Acceso	interactivo y fuera de línea	fuera de línea
Actualizaciones	escribe y lee muchas veces	escribe una vez, lee muchas veces
Estructura	estática	dinámico
Integridad	alta	bajo
Escala	no lineal	lineal

MapReduce y bases de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Una base de datos tradicional en general es una solución complementaria y no sustituta a MapReduce.
- Una base de datos relacional es eficiente para actualizaciones puntuales y para el manejo de datos transaccionales.
- La base de datos relacional también garantiza cierto tiempo de respuesta (al hacer un buen uso de índices y estructura de la DB en disco). Por otro lado, MapReduce está pensado para ser ejecutado en modalidad fuera de línea y no en tiempo real.
- MapReduce es bueno para problemas de grándes volumenes de datos que son escritos una vez y leidos muchas veces.

MapReduce: beneficios

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Reduce la complejidad de la sincronización entre procesos.
- Realiza un particionamiento automático de datos.
- Maneja la tolerancia a fallos de forma transparente.
- Maneja el balance de carga.
- Implementación sencilla (el desarrollador solo se debe encargar de implementar la función Map y la función Reduce).
- Permite el análisis de información que anteriormente era inviable (por los tiempos de ejecución o implementación).
- Provee códigos y patrones fácilmente reutilizables.

MapReduce: algunas desventajas

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Enfocado en aplicaciones que ejecutan en modo batch (fuera de línea).
- Puede forzar a ‘adaptar’ aplicaciones que no siguen estrictamente el modelo MapReduce.
- Puede ser necesario crear etapas adicionales para adaptarse al modelo.
- Puede ser necesario emitir valores intermedios ‘extraños’ (no intuitivos).
- Puede ser necesario crear funciones superfluas (por ejemplo, funciones map y/o reduce de tipo identidad).

MapReduce de Google Apps

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- El modelo fue extendido para su aplicación a la resolución eficiente de aplicaciones distribuidas para procesar grandes volúmenes de datos.
- Google App Engine es un servicio cloud de plataforma (PaaS) para el desarrollo y hosting de implementaciones distribuidas (web) en datacenters manejados por Google.
- Las aplicaciones ejecutan aisladas (en un sandbox) en múltiples servidores. Google App Engine escala automáticamente las aplicaciones a demanda (dependiendo del número de peticiones o requests).
- App Engine MapReduce es una implementación de Map-Reduce que incluye límites de performance, permitiendo a los desarrolladores controlar los costos de ejecución de sus aplicaciones.
- Soporta Python, Java (y otros lenguages JVM como Groovy, JRuby, Scala, Clojure, etc.), y (en versión experimental) Go y PHP.

MapReduce de Google Apps

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Procesamiento por etapas:
  1. Lectura de datos: desde storage (u otra fuente). Existen lectores predefinidos (por ej. leer registros de un tipo específico).
  2. Map: ejecuta una vez para cada valor de entrada, retornando una lista de pares (clave, valor). El framework divide la entrada en porciones que son manejadas en paralelo en múltiples instancias de la aplicación.
  3. Shuffle: agrupa los pares (clave, valor) que tienen la misma clave, para pasarlos a la etapa de reducción.
  4. Reduce: ejecuta una vez para cada clave en la lista agrupada de (clave, valor). Retornal una lista de valores que se pasan a la siguiente etapa.
  5. Escritura de salida: un proceso escritor concatena las salidas de la función de reducción en un orden arbitrario y las escribe en un medio de almacenamiento persistente. Es posible elegir entre un conjunto de escritores predefinidos con formatos específicos.

MapReduce: Hadoop vs Google Apps

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Diferencias entre ambos frameworks

- Hadoop: El reducer recibe una clave y un iterador sobre todos los valores asociados con esa clave no ordenados.
  - En Hadoop, puede 'resolverse' esta limitación formando claves compuestas que incorporen parte del valor, pero se requiere procesamiento adicional.
- Google Apps: Tiene una fucionalidad built-in que permite especificar una clave secundaria para ordenar los valores (si así se desea).

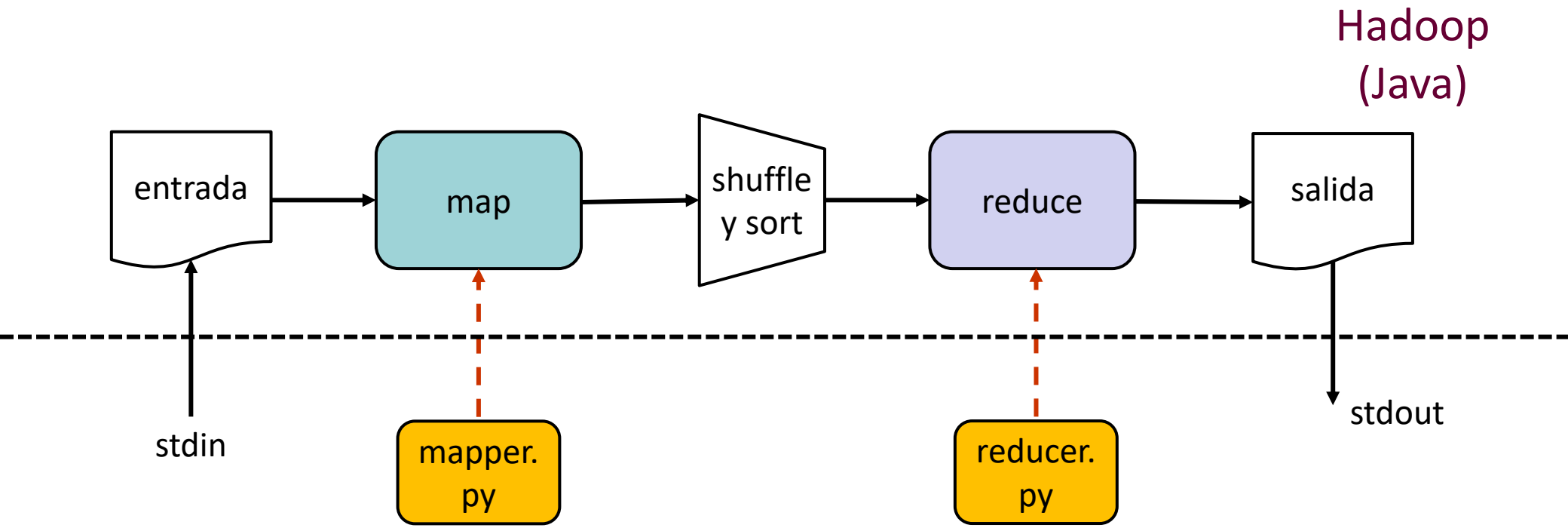
- Google Apps: La clave de salida del Reducer debe ser exactamente la misma que la clave de entrada.
- Hadoop: En Hadoop no existe tal restricción, y el Reducer puede emitir un número arbitrario de pares (clave, valor) de salida, con diferentes claves.

Hadoop Streaming

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Hadoop prove una API para MapReduce que permite escribir las funciones map y reduce en lenguajes diferentes a Java.
- Utiliza las herramientas de entrada/salida de Unix y Linux, permitiendo utilizar cualquier lenguaje que pueda leer de entrada estándar y escribir a salida estándar.
- Streaming está muy adaptado para el procesamiento de texto.
  - Los datos llegan a la tarea Map a través de la entrada estándar, se procesan línea por línea y el resultado se escribe en salida estándar, en formato clave-valor (delimitado por tabulador).
  - La entrada para la tarea Reduce corresponde a la salida de Map (formato clave-valor delimitado por tabulador), los pares están ordenados por clave, y se leen de entrada estándar.

Hadoop Streaming



Otros
lenguajes

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Map-Reduce: Índice invertido (Inverted Index)

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Se utiliza para generar un índice de un conjunto muy grande de datos, para poder aplicarlo para realizar búsquedas eficientes en el conjunto.

- Es generalmente utilizado cuando se requiere que consultas sobre determinados textos sean lo más rápidas posible. También es utilizado para la búsqueda de datos y documentos, y es muy aplicado por los motores de búsqueda actuales.

- Un índice invertido almacena las relaciones de datos/palabras/ información a sus ubicaciones en los documentos de destino, bases de datos y otros repositorios de información.

Índice Invertido

- Un ejemplo básico….
  - Dados los textos:
    - T1 = “BigData es un tema … ”
    - T2 = “Si es intersante … ”
    - T3 = “… si es un tema ☺”
  - Se tendrá el siguiente índice invertido:
    - “BigData”: {1}
    - “es”: {1, 2, 3}
    - “un”: {1,3}
    - “tema”: {1,3}
    - “si” : {2,3}
    - “☺” : {3}

Índice Invertido

- Un ejemplo básico….
  - Dados los textos:
    - T1 = “BigData es un tema … ”
    - T2 = “Si es intersante … ”
    - T3 = “… si es un tema ☺”
  - Se tendrá el siguiente índice invertido:
    - “BigData”: {1}
    - “es”: {1, 2, 3} ← ciertas palabras pueden repetirse muchas veces
    - “un”: {1,3}
    - “tema”: {1,3}
    - “si” : {2,3} ← es la comparación sensible a mayúsculas/minúsculas?
    - “☺” : {3} ← como se manejan los símbolos y otros caracteres?

Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- MapReduce es muy utilizado para la construcción de índices invertidos.
- Los índices invertidos se guardan en bases de datos relacionales para ser utilizados en búsquedas.
- Permiten el análisis de información no estructurada fuera de línea.
- Realizan un pre-procesamiento de la información para hacerla disponible de forma más ágil.

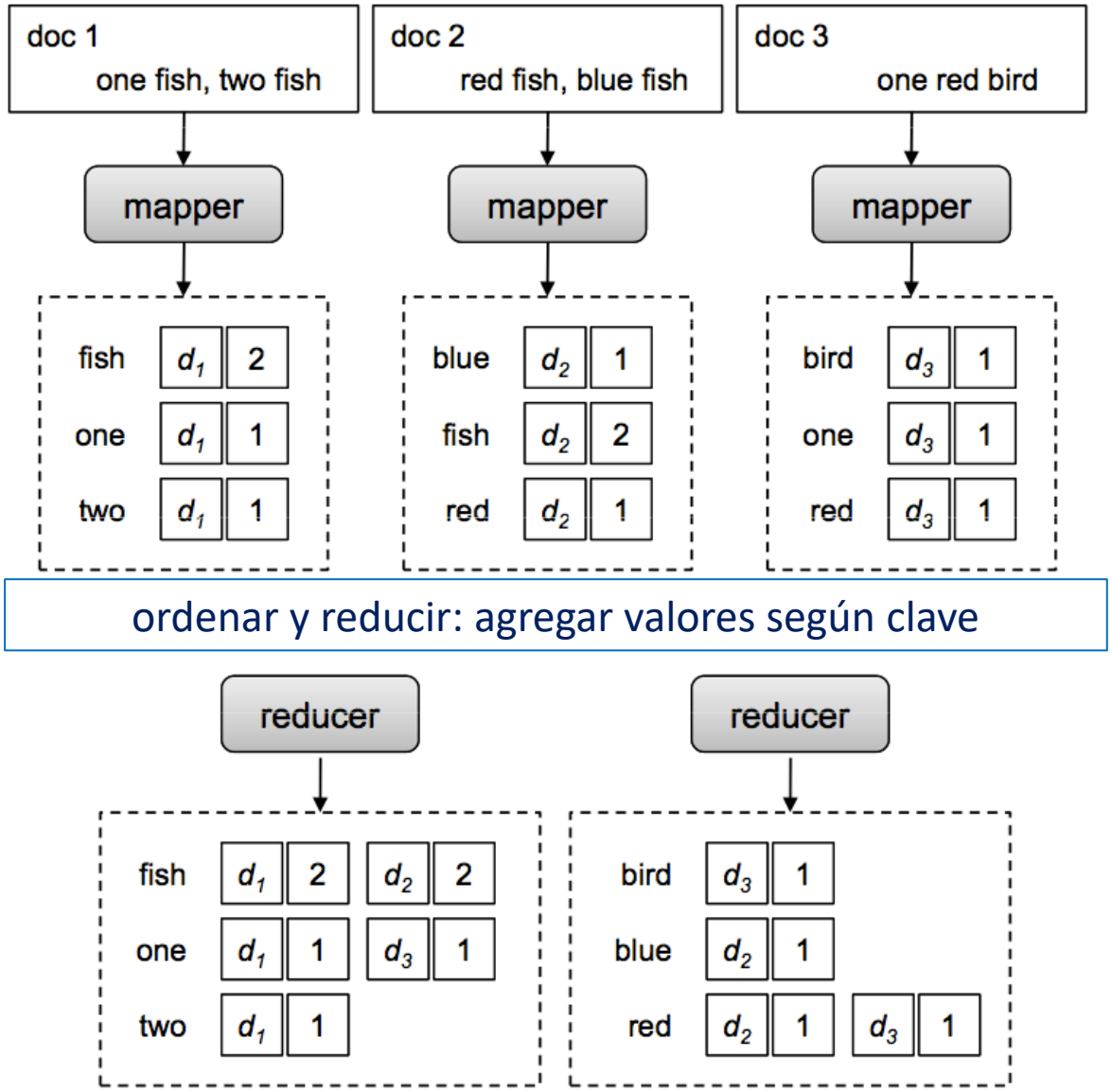
Índice Invertido

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Un índice invertido, además de la información de ubicación (en que documento está presente), puede contener información adicional:
  - Frecuencia de aparición en el documento.
  - Posición de la palabra en el documento.
  - Información adicional relacionada a la palabra que pueda servir para categorizar su relevancia en el texto (ej: si es un título o un subtítulo o una palabra simplemente, si aparece en negrita, etc).

- La información del índice invertido se puede guardar de diversas maneras: en una base de datos, en archivos de forma semi-estructurada, etc.

Índice Invertido



- La estructura general para la creación de un índice invertido involucra determinar el mapeo (documento, frecuencia) por palabra.
- El mapper emite clave=palabra, valor=(doc_id, frecuencia).
- El reducer almacena la información de forma conveniente (en una base de datos, concatenando los identificadores de los documentos, etc).

Índice Invertido

- El costo computacional asociado a la creación de un índice invertido está asociado a los mappers.
- Depende también del costo de parsear la información y a la cantidad de claves utilizadas y el balanceo de las mismas.
- Para datos semi-estructurados (json, xml), se tiene un costo adicional al tener que interpretar la información que se está consumiendo.
- Como se vio anteriormente, hay muchas palabras que dependiendo del idioma se repiten demasiado y producen un desbalance de carga (“the” en inglés, “el”/“la” en español). En estos casos, muchas veces se opta por no mapear estas palabras. Lo mismo sucede para las diferencias entre mayúsculas y minúsculas.

Índice Invertido

- Índice invertido de referencias a Wikipedia en StackOverflow.
- Para cada comentario en StackOverflow, buscando referencias a Wikipedia. Agrupa todos los ID de comentarios que referencian a la misma página de Wikipedia.
  - Puede ser usado para actualizar cada página de Wikipedia con los comentarios que la referencian.

Índice Invertido

- Índice invertido de referencias a Wikipedia en StackOverflow.
- Para cada comentario en StackOverflow, buscando referencias a Wikipedia. Agrupa todos los ID de comentarios que referencian a la misma página de Wikipedia.
  - Puede ser usado para actualizar cada página de Wikipedia con los comentarios que la referencian.
- Mapper: lee posts en StackOverflow y obtiene los ID de los que referencian a una página de Wikipedia.
- Extrae atributos XML para texto, tipo de post e ID.
- Para tipos que no son respuesta (tipo 2) se busca una URL de Wikipedia, si se encuentra emite (clave = URL, valor = ID).

Índice Invertido: mapper

```
public static class WikiExtractor extends Mapper<Object, Text, Text, Text> {
private Text link = new Text();
private Text outkey = new Text();
public void map(Object key, Text value, Context context)
    throws IOException, InterruptedException {
    Map<String, String> parsed = MRDPUtils.transformXmlToMap(value.toString());
    // obtener atributos XML
    String txt = parsed.get("Body");
    String posttype = parsed.get("PostTypeId");
    String row_id = parsed.get("Id");
    // saltear posts con cuerpo vacío o posts que son de tipo pregunta (1)
    if (txt == null || (posttype != null && posttype.equals("1"))) {
        return;
    }
    txt = StringEscapeUtils.unescapeHtml(txt.toLowerCase()); // Unescape HTML
    link.set(getWikipediaURL(txt));
    outkey.set(row_id);
    context.write(link, outkey);
    }
}
```

Índice Invertido: reducer

- Reducer: itera sobre los valores de entrada y agrega cada ID a un string. Se emite la clave recibida y la concatenación de IDs.

```
public static class Concatenator extends Reducer<Text,Text,Text,Text> {
private Text result = new Text();
public void reduce(Text key, Iterable<Text> values, Context context)
    throws IOException, InterruptedException {
    StringBuilder sb = new StringBuilder();
    boolean first = true;
    for (Text id : values) {
        if (first) {
            first = false;
        } else {
            sb.append(" ");
        }
        sb.append(id.toString());
    }
    result.set(sb.toString());
    context.write(key, result);
    }
}
```

Puede ser usado como Combiner para concatenar antes de la fase reduce.

MapReduce: Top N

Mapper

```
public static class TopNMapper extends Mapper {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    @Override public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
        String cleanLine = value.toString().toLowerCase(). \
            replaceAll("[_|$#<>\\^=\\[\\]\\*/\\\\,;,.\\-:()?!\"']", " ");
        StringTokenizer itr = new StringTokenizer(cleanLine);
        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken().trim());
            context.write(word, one);
        }
    }
}
```

- Separa palabras y emite (clave = palabra, valor = 1).

MapReduce: Top N

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Reducer

```
public static class TopNReducer extends Reducer {
    private Map countMap = new HashMap<>();
    @Override public void reduce(Text key, Iterable values, Context context)
        throws IOException, InterruptedException {
        // acumular ocurrencias de cada palabra
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        // salvar ocurrencias en el HashMap
        countMap.put(key, new IntWritable(sum));
    }
```

- Calcula la suma de los valores recibidos de los mappers para cada clave (ocurrencia de cada palabra). Clave y suma se agregan a un HashMap.

MapReduce: Top N

- Reducer (continuación)

```
    @Override protected void cleanup(Context context)
        throws IOException, InterruptedException {
        Map sortedMap = sortByValues(countMap); // función Java
        int counter = 0;
        for (Text key: sortedMap.keySet()) {
            if (counter ++ == 20) {
                break;
            }
            context.write(key, sortedMap.get(key));
        }
    }
}
```

- El ordenamiento solo debe realizarse luego de recibir todos los valores !
- El método cleanup() es invocado por Hadoop luego que el reducer ha recibido todos los datos. Se ordena el HashMap por valores y se retornan los primeros 20.