Computación distribuida

Sergio Nesmachnow
(sergion@fing.edu.uy)

UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Computación distribuida

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Contenido

1. Computación distribuida y computación cloud
2. Procesamiento de grandes volúmenes de datos
3. El modelo de computación Map-Reduce
4. El framework Hadoop y su ecosistema
5. Almacenamiento: HDFS y HBase
6. Aplicaciones de Map Reduce sobre Hadoop: conteo, índice invertido, filtros
7. Procesamiento de grandes volúmenes de datos con Apache Spark
8. Ejemplos de aplicaciones en Spark y el lenguaje Scala
9. Análisis de datos utilizando Spark y el lenguaje R.
10. Aplicaciones iterativas: Google Pregel y Apache Giraph

Procesamiento de grandes volúmenes de datos (Big Data)

Big Data

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- '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

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- 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

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- 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

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- 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: breve historia

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En 2003 Google presentó el artículo abierto (whitepaper) “Google File System” que influenció a Hadoop Distributed File System
- En 2004 el artículo “Simplified Data Processing on Large Clusters” presentó una descripción del enfoque de Google para procesar (indexar y rankear) grandes volúmenes de datos en el contexto de motores de búsqueda
- El trabajo de Google influenció al Proyecto Nutch, en desarrollo en Yahoo!, que incorporó los principios presentados en los artículos sobre Google FileSystem y Google MapReduce en el proyecto que se denominó Hadoop.

MAP-REDUCE: motivación

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cuando los enfoques tradicionales de procesamiento de grandes volúmenes de datos comenzaron a alcanzar sus límites, surgió un nuevo paradigma de análisis de datos, basado en el uso de sistemas de computación distribuida.
- Los entornos para computación paralela y distribuida existen desde décadas previas a la presentación del artículo sobre Google MapReduce en 2004, los más populares son Message Passing Interface (MPI), Parallel Virtual Machine (PVM), HTCondor, y otros.
- Los entornos tradicionales para programación en clusters y grids cuentan con limitaciones de aplicabilidad y escalabilidad.

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

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- 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 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

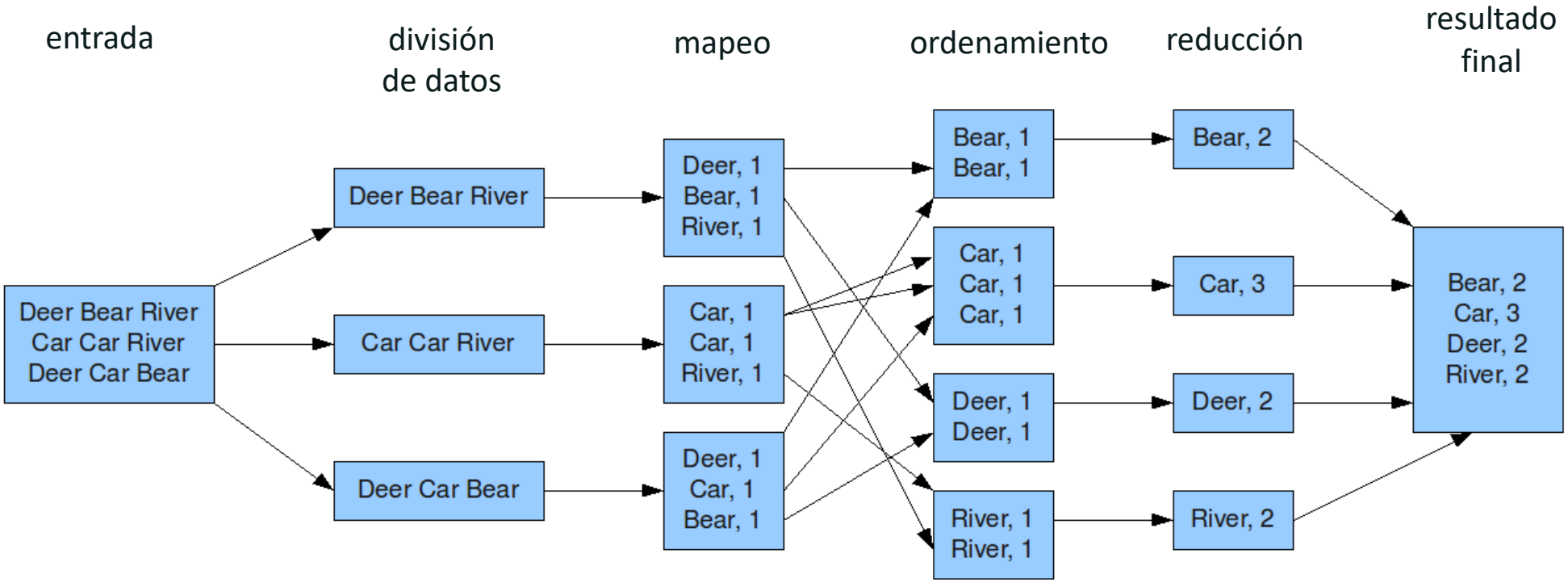
  Page 3: Deer Car Bear
- Lista de pares (k1,v1):

  [(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

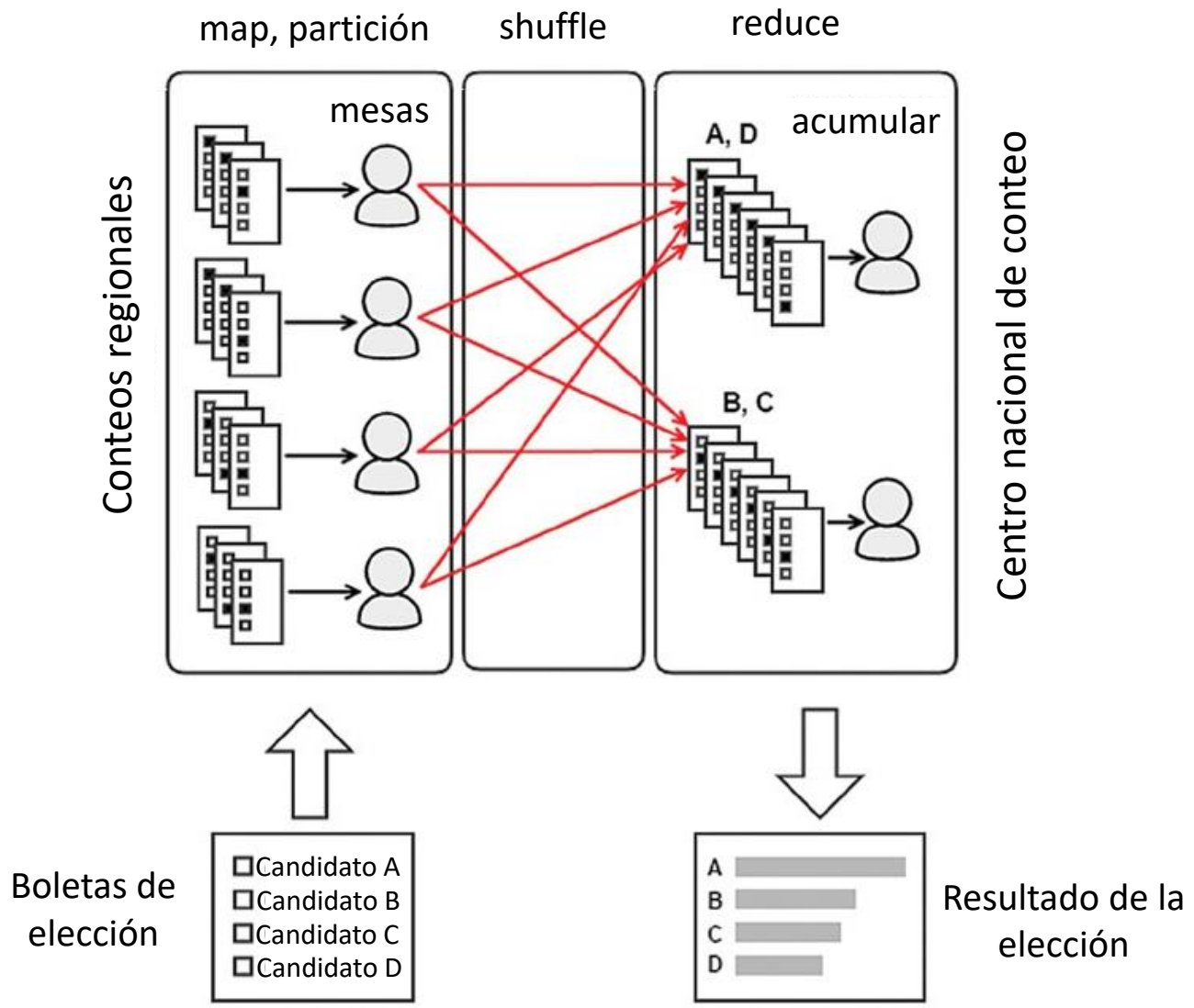
  Page 3: Deer Car Bear
- Lista de pares (k1,v1):

  [(Page 1, Deer Bear River),

  (Page 2, Car Car River),

  (Page 3, Deer Car Bear)]

MapReduce: conteo de votos

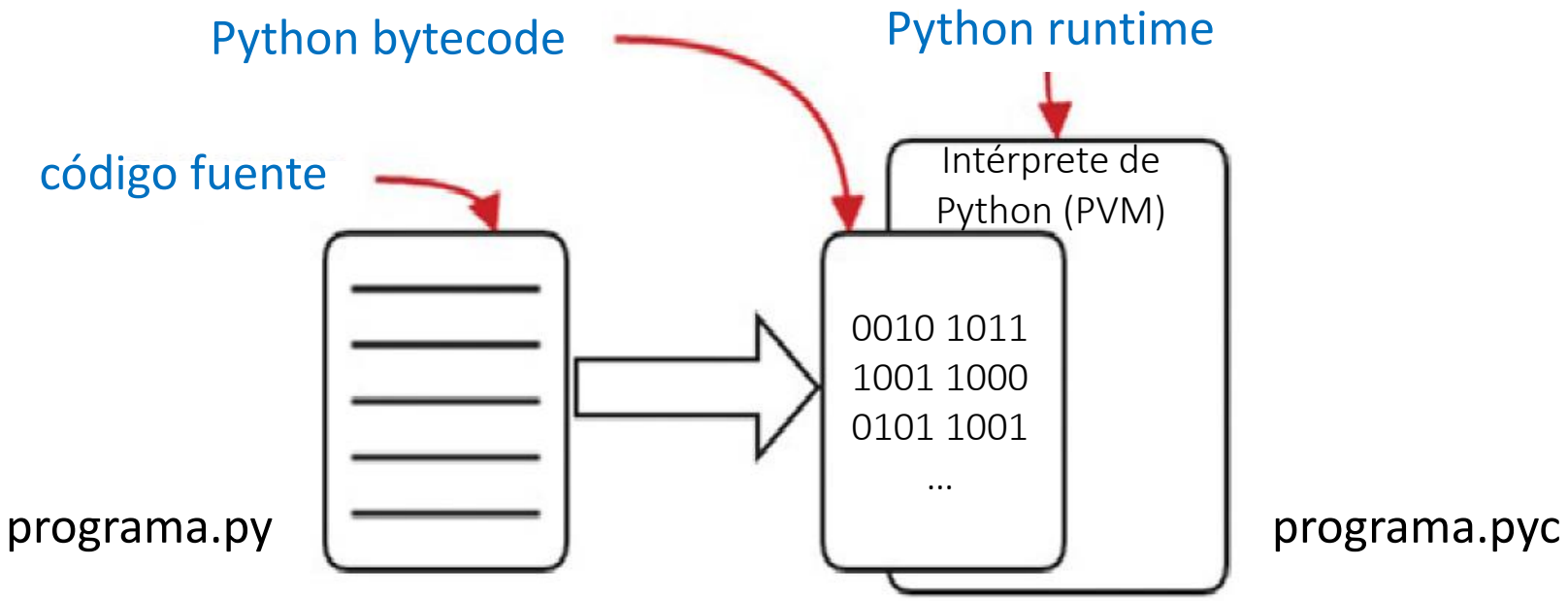


UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Python: programación funcional y Map-Reduce

Python

- Python es un lenguaje de programación multiparadigma: combina los paradigmas procedurales, orientado a objetos y funcional.
- Es un lenguaje interpretado: el *intérprete de Python* lee y ejecuta las instrucciones en el programa de usuario.
- El intérprete de Python compila a Python bytecode, un código multiplataforma que ejecuta en una Python Virtual Machine (PVM).



Python: implementaciones

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Cpython: implementación estándar y más utilizada de Python.
- Construida sobre ANSI C portable, es la variante más eficiente de Python.
- Implementación por defecto en la mayoría de las distribuciones de Linux.
- Ejecuta instrucciones Python bytecode instructions en PVM, escrita en C.
- PySpark (Python para Spark)
- Usa la implementación estándar Cpython, disponible en todos los sistemas donde se instala Spark.
- Usa Py4J, que permite a programas Python usando el intérprete CPython accede a objetos Java en una JVM.
- Py4J también permite a aplicaciones Java applications invocar a y utiilzar objetos Python.
- Las bibliotecas de Py4J están incluidas con la instalación de Spark.

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Listas: secuencias indexadas (desde 0) de variables modificables.
- Admiten remover, modificar y agregar elementos.
- Soportan los principales constructores de la programación funcional: map(), reduce() y filter().

```
# Python
tempc = [38.4, 19.2, 12.8, 9.6]
tempf = map(lambda x: (float(9)/5)*x + 32, tempc)
# Pyspark
tempc = sc.parallelize([38.4, 19.2, 12.8, 9.6])
tempf = tempc.map(lambda x: (float(9)/5)*x + 32)
```

- Los elementos se acceden a través de su índice: firstelem = tempc[0].
- Múltiples métodos implementados: count(), sort(), reverse(), append(), extend(), insert(), remove() y otros.
- Las listas de Python son modificables, luego veremos las listas incluidas en Resilient Distributed Datasets (RDD) de Spark, que son inmutables.

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Conjuntos (sets): colecciones no ordenadas de elementos.
- Son modificables, pueden agregarse y eliminarse elementos.
- Los conjuntos pueden crearse como objetos inmutables: frozensets.
  - Son similares a los RDD de Spark que usan conjuntos.
- Soportan operaciones estándar: unión, intersección, diferencia, etc.

```
# Python
tempc0 = frozenset([38.4, 19.2])
tempc1 = frozenset([12.8, 9.6])
tempc2 = frozenset([23.7, 9.6])
tempc1.intersection(tempc2)
frozenset([9.6])
tempc1.difference(tempc2)
frozenset([12.8])
tempc0.union(tempc1)
frozenset([9.6, 38.4, 19.2, 12.8])
```

```
# Pyspark
tempc = sc.parallelize(set([38.4, 19.2, 12.8, 9.6]))
tempc
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:423
```

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Tuplas: secuencias inmutable de objetos. Los objetos en una tupla pueden ser modificables o no.
- Pueden contener objetos de diferentes tipos (string, int, float, etc.) e inclusive otras secuencias como conjuntos o tuplas.
- Las tuplas pueden visualizarse como listas inmutables, pero tienen diferentes propósitos y diferentes constructores.
- Las tuplas son similares a registros en una base de datos relacional: cada registro tiene una estructura y cada campo tiene una posición (ordinal) en la estructura y un significado específico.
- Especificadas como valores separados por comas, delimitados por paréntesis. Los elementos se acceden por su ordinal.
- Tienen métodos para comparación, largo, etc.
- Admiten conversion de lista en tupla con el método tuple(lista).

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Creación y manejo de tuplas.
  - Creación de tupla

```
rec0 = "Jeff", "Aven", 46
rec1 = "Barack", "Obama", 54
rec2 = "John F", "Kennedy", 46
rec3 = "Jeff", "Aven", 46
rec0
('Jeff', 'Aven', 46)
len(rec0)
3
print("first name: "+rec0[0])
first name: Jeff
```

  - Comparación

```
cmp(rec0, rec1)
1 ## ningún campo es igual
cmp(rec0, rec2)
-1 ## algunos campos son iguales
cmp(rec0, rec3)
0 ## todos los campos son iguales
```

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Creación y manejo de tuplas
  - Creación de tupla de tuplas

```
all_recs = rec0,rec1,rec2,rec3
all_recs
(('Jeff', 'Aven', 46), ('Barack', 'Obama', 54),
('John F', 'Kennedy', 46), ('Jeff', 'Aven', 46))
```

  - Creación de lista de tuplas

```
list_of_recs = [rec0,rec1,rec2,rec3]
list_of_recs
[('Jeff', 'Aven', 46), ('Barack', 'Obama', 54),
('John F', 'Kennedy', 46), ('Jeff', 'Aven', 46)].
```

- Es muy importante distinguir entre listas (paréntesis rectos) y tuplas (paréntesis curvos), ya que su significado estructural es diferente !

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Tuplas en PySpark: estructuras frecuentemente generadas como salida de una función map().
- Ejemplo: temperaturas en Celsius y Fahrenheit

```
tempc = sc.parallelize([38.4, 19.2, 12.8, 9.6])
temp_tups = tempc.map(lambda x: (x,(float(9)/5)*x + 32))
temp_tups
PythonRDD[1] at RDD at PythonRDD.scala:43
temp_tups.take(4)
# RDD containing a list of Tuples
[(9.6, 49.28), (19.2, 66.56), (38.4, 101.12), (12.8, 55.04)]
```

- Las tuplas son el principal tipo de objeto en la programación funcional, las usaremos en el análisis de datos en Spark.

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Diccionarios (dicts): conjuntos modificables y no ordenados de pares clave-valor.
  - Especificados entre llaves (‘{ }’).
  - Las claves se separan por dos puntos (‘:’).
  - Los pares clave-valor se separan por coma (‘,’).
- Los elementos no están asociados a un ordinal. Se accede a ellos a través de su clave.
- Sus elementos son autodescriptivos, no dependen de un esquema o de un ordinal.
- Soportan operaciones de agregado, eliminación, selectors (keys(), values()) y otras funciones (cmp(), len(), etc.).

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de diccionarios.

```
dict0 = {'fname':'Jeff', 'lname':'Aven', 'pos':'author'}
dict1 = {'fname':'Barack', 'lname':'Obama', 'pos':'president'}
dict2 = {'fname':'Ronald', 'lname':'Reagan', 'pos':'president'}
dict3 = {'fname':'John', 'mi':'F', 'lname':'Kennedy','pos':'president'}
dict4 = {'fname':'Jeff', 'lname':'Aven', 'pos':'author'}
len(dict0)
3
dict0['fname']
'Jeff'
dict0.keys()
['lname', 'pos', 'fname']
dict0.values()
['Aven', 'author', 'Jeff']
cmp(dict0, dict1) # comparar diccionarios
1 ## las claves y/o valores no concuerdan
cmp(dict0, dict4)
0 ## todas las claves y valores concuerdan
cmp(dict1, dict2)
-1 ## algunos pares clave-valor concuerdan
```

Python: estructuras de datos

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Los diccionarios pueden utilizarse como objetos inmutables en un RDD de Spark.
- Ejemplo: crear un RDD como lista de diccionarios y aplicar operaciones filter() y map().

```
people = sc.parallelize([dict0, dict1, dict2, dict3])
people
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:423
presidents = people.filter(lambda x: x['pos'] == 'president') \
.map(lambda x: x['fname'] + " " + x['lname'])
presidents.take(3)
['Barack Obama', 'Ronald Reagan', 'John Kennedy']
```

Python: serialización

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Serialización: proceso de convertir un objeto a una estructura que se puede empaquetar (serializar) y desempaquetar (deserializar) en el mismo o en un diferente sistema.
- La serialización es una característica de los sistemas distribuidos y es fundamental para Hadoop y Spark.
- JSON (Java Script Object Notation): estructura estándar y multiplataforma, especialmente usada para servicios web.
- JSON es soportado en Python de forma nativa, utilizando el objeto json, un modulo incorporado (built-in) para codificar y decodificar JSON.
- Los objetos JSON son pares clave-valor (diccionarios) y/o arrays (listas), que pueden estar anidadas.
- Los objetos JSON de Python incluyen métodos para buscar, agregar y eliminar claves, actualizar valores y desplegar objetos.

Python: serialización

- Ejemplo: creación y uso de objeto JSON en Python.

```
import json
from pprint import pprint
json_str = '''{"people" : [
  {"fname": "Jeff", "lname": "Aven", "tags": ["big data","hadoop"]},
  {"fname": "Doug", "lname": "Cutting", "tags":
      ["hadoop","avro","apache","java"]},
  {"fname": "Martin","lname": "Odersky","tags":
      ["scala","typesafe","java"]},
  {"fname": "John", "lname": "Doe", "tags": []} ]}'''
people = json.loads(json_str)
len(people["people"])
4
people["people"][0]["fname"]
u'Jeff'
```

Python: serialización

- Ejemplo: creación y uso de objeto JSON en Python (continuación).

```
# agregar un tag a la primer persona
people["people"][0]["tags"].append(u'spark')
# eliminar la cuarta persona
del people["people"][3]
# imprimir el objeto JSON
print(people)
{u'people': [{u'fname': u'Jeff',
u'lname': u'Aven',
u'tags': [u'big data', u'hadoop', u'spark']},
{u'fname': u'Doug',
u'lname': u'Cutting',
u'tags': [u'hadoop', u'avro', u'apache', u'java']},
{u'fname': u'Martin',
u'lname': u'Odersky',
u'tags': [u'scala', u'typesafe', u'java']}]}
```

Python: serialización

- Los objetos JSON pueden usarse en RDD de Spark de modo similar a como se trabaja con los dict de Python

```
import json
json_str = ... # como el del ejemplo anterior
people_obj = json.loads(json_str)
people = sc.parallelize(people_obj["people"])
people
ParallelCollectionRDD[0] at parallelize at PythonRDD.scala:423
hadoop_tags = people.filter(lambda x: "hadoop" in x['tags']) \
.map(lambda x: x['fname'] + " " + x['lname'])
hadoop_tags.take(2)
[u'Jeff Aven', u'Doug Cutting’]
```

Python: serialización

- Pickle: método de serialización propietario de Python, alternativo a JSON.
  - Más eficiente, pero no portable. cPickle (en C) es aún más eficiente.
  - Convierte objetos Python a un stream de bytes que puede transmitirse, almacenarse y reconstruirse a su estado original.

```
try: import cPickle as pickle # intenta usar cPickle si está disponible
except: import pickle
obj = { "fname": "Jeff",  "lname": "Aven", "tags": ["big data","hadoop"]}
str_obj = pickle.dumps(obj) # retorna la representación pickle del objeto (string).
pickled_obj = pickle.loads(str_obj) # carga el objeto pickle
pickled_obj["fname"]
'Jeff'
pickled_obj["tags"].append('spark')
str(pickled_obj["tags"])
"['big data', 'hadoop', 'spark']"
pickled_obj_str = pickle.dumps(pickled_obj)
pickled_obj_str
"(dp1\nS'lname'\np2\nS'Aven'\np3\nsS'fname'\np4\nS'Jeff'\np5\nsS'tags'\np6\n(lp7\nS
'big data'\np8\naS'hadoop'\np9\naS'spark'\np10\nas."
pickle.dump(pickled_obj, open('object.pkl', 'wb')) # serializa el objeto y lo salva
                                                   en un archivo
```

Python: serialización

- PySpark usa PickleSerializer por defecto (con cpickle) y permite leer objetos serializados desde otros sistemas (por ejemplo, SequenceFiles de Hadoop) y convertirlos a objetos Python.
- Es un mecanismo eficiente para almacenar y transmitir archivos entre procesos Spark.
- Métodos (sobre RDDs de Spark)

`saveAsPickleFile(path, batchSize)`: salva un objeto RDD a un nuevo directorio

```
shakespeare = sc.textFile("file:///opt/spark/data/shakespeare.txt")
pklfile = shakespeare.saveAsPickleFile("file:///opt/spark/data/pickled")
```

`pickleFile(name, minPart)`: carga un objeto RDD desde un archivo pickle.

```
pklfile = sc.pickleFile("file:///opt/spark/data/pickled")
pklfile.count()
```

Python: programación funcional

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Programación funcional: paradigma de programación declarativa basado en el uso de funciones matemáticas
  - Contrasta con la programación imperativa, que enfatiza los cambios de estado mediante la mutación de variables.
- Principios de la programación funcional:
  - Las funciones son la unidad fundamental de programación.
  - Las funciones tienen entrada y salida solamente.
  - Las sentencias, que pueden producir efectos laterales no deseados, no están permitidas.
  - Existen funciones de primera clase y funciones de alto orden.
  - Debe proveerse soporte para funciones anónimas.

Python: programación funcional

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Funciones anónimas
  - Se implementan en Python con el constructor lambda. No se emplea la palabra clave def utilizada para funciones con nombre.
  - Aceptan cualquier número de parámetros pero retornan un único valor.
  - El valor de retorno puede ser otra función, un valor escalar, una estructura de datos (lista, etc.)
  - Función con nombre

```
def plusone(x): return x+1
plusone(1)
2
type(plusone)
<type 'function'>
```

  - Función sin nombre

```
plusonefn = lambda x: x+1
plusonefn(1)
2
type(plusonefn)
<type 'function'>
```

Python: programación funcional

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ambos métodos para crear funciones son similares y funcionalmente equivalentes.
- La función con nombre tiene una referencia a su nombre, la anónima no (de hecho, el nombre no es necesario para ser usada dentro de otra función).

```
plusone.func_name
'plusone'
plusonefn.func_name
'<lambda>’
```

- Las funciones con nombre pueden contener sentencias, las anónimas solo una expresión (que puede incluir la invocación a otra función)
- Las funciones con nombre utilizan return, las anónimas no.
- Las funciones anónimas proven soporte para encadenamiento de funciones de alto orden (map, reduce, filter), para crear pipelines de procesamiento.

Python: programación funcional

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Funciones de alto orden: aceptan otras funciones como argumento y pueden retornar funciones como resultado.
  - Map, flatMap, filter, map, reduceByKey (veremos ejemplos más adelante)
- Funciones callback: retornan otra función.
- Tail Calls: funciones que se invocan a si mismas (recursivas).
- Cortocircuitos: permiten evitar cálculos innecesarios (and/or por ejemplo).

```
…
.filter(lambda x: (len(x) > 0) and (len(x) < 3))
```

  - La segunda condición solo se evalúa si la primera es verdadera

Python: paralelismo

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- En la programación funcional pura, las funciones que producen efectos laterales (fuera del ámbito de la función) están desaconsejadas o directamente prohibidas.
- Ejemplos: desplegar un valor, escribir a un archivo externo o a una base de datos, mantener un estado, etc.
- La motivación principal es que si una función no tiene efectos laterales y no tiene dependencias con otra función, ambas pueden ser ejecutadas en paralelo.
- Las capacidades intrínsecas de paralelismo en la programación funcional en Python son adecuadas para el procesamiento paralelo. Las veremos en el curso utilizando Spark.

Python: closures

- Clousures
- Un closure es una función que usa una o más variables declaradas fuera de la función (variables libres), en conjunto con variables locales a la función. La función se encapsula con los valores de las variables libres en el momento en que fue definida.
- En Python, los closure son objetos funcionales que delimitan (enclose) el ámbito en el momento en que son instanciados.

```
def generate_message(concept):
  def ret_message():
     return 'This is an example of ' + concept
  return ret_message

call_func = generate_message('closures in Python') # crear closure
call_func
<function ret_message at 0x7f2a52b72d70>
call_func()
'This is an example of closures in Python'
```

Python: closures

- Clousures (continuación)

```
call_func.__closure__ # inspeccionar closure
(<cell at 0x7f2a557dbde0: str object at 0x7f2a557dbea0>,)
type(call_func.__closure__[0])
<type 'cell'>
call_func.__closure__[0].cell_contents
'closures in Python’

del generate_message # eliminar la función

call_func() # invocar el closure nuevamente
'This is an example of closures in Python’ # el closure todavía
                                            funciona !
```

  - ret_message() es el closure. El valor de la variable concept se encapsula en el ámbito de la función. La información del closure se visualiza con el campo __closure__ de la función.
  - Las referencias encapsuladas en la función se almacenan en una tupla de celdas, que se accede con cell_contents. Cuando se elimina la función externa generate_message, la función referenciante call_func aún funciona.

MapReduce en Python

- Map: Dada una lista, transforma cada uno de sus elementos aplicando una función que recibe por parámetro.
- La firma de la función map es `map(funcion[x->y],secuencia)`
- La función[x→y] recibe un elemento y devuelve otro elemento.

- Ejemplo: dada una lista de números, obtener la lista de esos mismos números multiplicados por 2.
- Una implementación procedural es

```
def multiplicadosPorDosProcedural(lista):
    multiplicados = []
    for i in lista:
        multiplicados.append(i * 2)
    return multiplicados
print multiplicadosPorDosProcedural([1, 4, 5, 2, 3, 10])
```

Imprime: [2, 8, 10, 4, 6, 20]

MapReduce en Python

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Map: Dada una lista, transforma cada uno de sus elementos aplicando una función que recibe por parámetro.

- Una implementación utilizando la función de orden superior map es

```
def porDos(n): return n * 2

def multiplicadosPorDos(lista):
    return map(porDos, lista)

print multiplicadosPorDos([1, 4, 5, 2, 3, 10])
```

Imprime: [2, 8, 10, 4, 6, 20]

MapReduce en Python

- Reduce: Dada una lista y una función, reduce la lista a un único elemento, aplicando la función en forma progresiva de izquierda a derecha.
- La función debe recibir dos parámetros y retornar un único valor.
- Ejemplos: dada una lista de precios, obtener el promedio; data una lista de números obtener el máximo (o la suma); dada una lista de personas (con un atributo edad) retornar la de mayor edad; etc.
- La firma de la función reduce es `reduce(funcion[x,y->z], lista)`
- Ejemplo: reducción mediante suma.

```
def sumar(a, b): return a + b

def sumaConReduce(numeros):
    return reduce(sumar, numeros)

print sumaConReduce([2, 3, 10, 11])
```

Imprime: 26

MapReduce en Python

- El patrón es similar al de la función map. Reduce recibe como parámetros una función y una lista de elementos. La función recibe dos elementos y devuelve uno (en el ejemplo, el resultado de la suma de los elementos de entrada).
- La función se aplica 'en forma progresiva', 'de a pares' y 'de izquierda a derecha'.
- Por ejemplo, para la lista [2, 3, 10, 11]:

  sumar(2, 3) -> 5

  sumar(5, 10) -> 15

  sumar(15, 11) -> 26

  resultado -> 26

El orden es relevante !

- La primera ejecución se realiza con los dos primeros elementos de la lista y luego se utiliza el resultado previo para cada invocación subsiguiente: resultado = f(f(f(2,3),10),11) = 26

Python: recursión

- Función recursiva que genera una lista de pares aleatorios (x,y) que se mapea para generar tuplas con x, y, y el MCD entre x e y.

```
def gcd(x, y):
    if x < y: return gcd(y, x)
    r = x%y
    if r == 0: return y
    else: return gcd(y, r)
```

```
import random
low = 1
high = 100
numpairs = sc.parallelize([(random.randint(low, high),random.randint(low,
           high)) for k in range(10)])
numpairs_gcd = numpairs.map(lambda x: (x[0], x[1], gcd(x[0], x[1])))
numpairs_gcd.take(5)
[(81, 3, 3), (91, 100, 1), (99, 18, 9), (28, 34, 2), (90, 99, 9)]
```

MapReduce en Python Spark

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

- Ejemplo de funciones de alto orden que implementan MapReduce en Python Spark

```
lines = sc.textFile("file:///opt/spark/data/shakespeare.txt")
counts = lines.flatMap(lambda x: x.split(' ‘)) \
.filter(lambda x: len(x)>0) \
.map(lambda x: (x, 1)) \
.reduceByKey(lambda x, y: x + y).collect()
for (word, count) in counts:
   print("%s: %i" % (word, count))
```