Tecnologie Cloud e Mobile

Lez. 12

Python e Moduli, Map-Reduce

Giuseppe Psaila

Università di Bergamo

giuseppe.psaila@unibg.it

Python e Moduli

Le funzioni di Callback

- Prima di procedere con i moduli, ci chiediamo:
- Esiste il concetto di «funzione di callback»?
- Si possono passare funzioni come parametri attuali di altre funzioni?
- La risposta è «sì, si può»

Le funzioni di Callback: Esempio

```
def stampa(sum):
    print("Sum = {}".format(sum))

def main(a, b, callback = None):
    print("adding {} + {}".format(a, b))
    if callback:
        callback(a+b)

main(1, 2, stampa)
```

Le funzioni di Callback: Esempio

- Il programma definisce la funzione «stampa»
- Poi definisce la funzione «main»:
  se il parametro «callback» viene passato, ipotizza che sia una funzione e la chiama
- L'ultima riga chiama la funzione «main», passando la funzione «stampa» come terzo parametro

```
adding 1 + 2
Sum = 3
```

I Moduli

- I moduli sono delle parti di codice contenute in un contenitore di codice
- In Python, il modulo ha un nome e incapsula al suo interno variabili e funzioni
- Esporta, quindi, funzioni e classi

I Moduli Built-in

- L’interprete Python contiene molti moduli built-in
- Questi sono già disponibili, non devono essere pre-installati
- Per usare un modulo, si deve usare l’istruzione «import» per dire all’interprete di importarlo nel codice

Prefissi

- Il nome del modulo fa da prefisso:
  - Per poter invocare le funzioni del modulo, occorre usare la notazione puntata *nome_modulo.nome_funzione*
- In questo modo, non si corre il rischio di conflitti tra moduli diversi

Il modulo gc

- Il modulo «gc» consente di gestire il garbage collector
- Vediamo solo alcune funzioni
- gc.collect(): fa partire la raccolta
- gc.enable(): attiva il garbage collector (default)
- gc.disable(): disabilita il garbage collector
- gc.isenabled(): true se è attivo

Il Modulo per JSON

- Un modulo built-in di Python consente di lavorare sul formato JSON
- Serializza un dizionario come documento JSON
- Deserializza un documento JSON come dizionario
- Per utilizzarlo, importarlo come
  `import json`

Il modulo per JSON

- Per leggere da una stringa:
  data = json.loads(stringa)
- Per leggere da file:
  data = json.load(filehandler)
- Per scrivere in una stringa:
  s = json.dumps(data)
- Per scrivere su file:
  json.dump(data, filehandler)

I File Handler

- Sono degli oggetti buil-in che servono per gestire i file
- Per aprire un file in lettura:
  filein = open (stringa_nome_file)
- Per aprire un file in scrittura:
  fileout = open (stringa_nome_file, "w")
- «filein» e «fileout» sono file handler

Chiudere i File

- Per chiudere un file: filehandler.close()
- Ricordatevi di chiudere i file, perché fino a che il processo dell’interprete non è terminato, viene mantenuto un lock sui file stessi che non consente di cancellarli

Leggere i File

- Per leggere l'intero contenuto di un file:
  string = filehandler.read()

- Per leggere una riga per volta
  string = filehandler.readline()
- Si può iterare con for su file handler
  `for line in fielhandler:`

Scrivere sui File

- Per scrivere su un file:
  filelhandler.write(testo)
- Se serve creare delle righe nel file, nelle stringhe si può usare il carattere di escape \n come fine riga

Leggere/Scrivere Liste

- Per leggere tutte le righe di un file:
  lista = filelhandler.readlines( )
- Produce una lista con tutte le righe del file
- Per scrivere una lista su file, un elemento per riga
  filehandler.writelines(lista)

Esempio – Parte 1

```
import json

while 1:
    nfilein = input("File input:")
    nfileout = input("File output")
    if len(nfilein)>0 and len(nfileout)>0:
        break

filein = open(nfilein)
data = json.load(filein)
```

Esempio – Parte 2

```
nuovo= {}
nuovo["Elements"] = len(data["docs"])
nuovo["docs"] = []
for d in data["docs"]:
    nuovo["docs"].append(
    {"name": d["name"], "age": d["age"]})
fileout = open(nfileout, "w")
json.dump(nuovo, fileout)
filein.close()
fileout.close()
```

Esempio

- Il ciclo while serve per richiedere all’utnte il nome del file da leggere (variabile «nfilein») e su chui scrivere (variabile «nfileout»), evitando che l’utente inserisca le stringhe vuote
- Viene aperto il file in input, ottenendo il file handler denominato «filein»
- Quindi si invoca la funzione load del modulo jeson, che produce l’oggetto referenziato dalla variabile data

Esempio

- Supponiamo di aver letto il file seguente:

```
{
    "Total": 1,
    "Selected": 10,
    "docs": [
        {
            " id":
"5ea9b76dbb5ab35514b0b80d",
            "name": "\"Paolino\"",
            "age": "12"
        }
    ]
}
```

Esempio

- La seconda parte del codice trasforma il dizionario nel modo seguente:

```
{
 "Elements": 1,
 "docs": [{"name": "\"Paolino\"", "age":
"12"}]
}
```

Istruzione «with»

- L’istruzione «with» crea un contesto basato su un oggetto
- Se l’oggetto rispetta il protocollo detto «context manager»
- Cioè fornisce un metodo («__enter__») da chiamare all’ingresso del contesto e un metodo («__exit__») da chiamare all’uscita dal contesto
- I file handler lo hanno

Esempio – Parte 1

```
import json

while 1:
    nfilein = input("File input:")
    nfileout = input("File output")
    if len(nfilein)>0 and len(nfileout)>0:
        break

with open(nfilein) as filein :
    data = json.load(filein)
```

Esempio – Parte 2

```
nuovo= {}
nuovo["Elements"] = len(data["docs"])
nuovo["docs"] = []
for d in data["docs"]:
    nuovo["docs"].append(
     {"name": d["name"], "age": d["age"]})
with open(nfileout, "w") as fileout:
    json.dump(nuovo, fileout)
```

Istruzione «with»

- Vantaggio: non doversi ricordare di chiudere il file
- Infatti, il contesto di «with» chiede all'oggetto di svolgere all'inizio le operazioni preliminari e all'uscita di svolgere le operazioni di chiusura
- Nel caso dei file, è importante ricordarsi di chiudere il file, quindi uscendo dal contesto, il file viene automaticamente chiuso

Chiamate HTTP

- Come effettuare chiamate HTTP?
- Occorre usare il modulo built-in «requests»
  `import requests`

Chiamate HTTP

- Effettuare la chiamata con il metodo GET:
  res = requests.get(url, params)
  dove
- url è l'indirizzo al quale inviare la richiesta
- params è un dizionario con i parametri da accodare all'URL, dopo il «?»
- Restituisce un oggetto che consente di gestire la risposta

Chiamate HTTP

- Effettuare la chiamata con il metodo POST:
  res = requests.post(url, data=none, json=none)
  dove
- url è l'indirizzo al quale inviare la richiesta
- data: i dati da mandare (opzionale)
- json: dati json da mandare
- Restituisce un oggetto che consente di gestire la risposta

La Risposta

- È un oggetto che rappresenta la risposta ricevuta
- Campi/metodi di interesse:
  - text: campo che contiene il contenuto testuale della risposta
  - json(): metodo che interpreta il testo come documento JSON e lo deserializza in un dizionario
- Per saperne di più: **`https://requests.readthedocs.io/en/master/api/#requests.Response`**

Esempio

- Vogliamo chiamare il server che abbiamo realizzato con Node.js
- In particolare, il servizio che fornisce la lista di oggetti nel db di MongoDB

Esempio

```
import requests

URL = "http://127.0.0.1:8080/list"
PARAMS = {}
r = requests.get(url = URL, params = PARAMS)
data = r.json()

print("Total: {}".format(data["Total"]))
print("Selected: {}".format( data["Selected"]))
for n in data["docs"]:
    print("Name: {}, Age: {}".format(n["name"],
n["age"]))
```

Esempio

- Osservate la facilità con la quale abbiamo effettuato la richiesta
- Poi con un ciclo for iteriamo sul campo «docs» stampiamo solo i campi che ci interessano dei documenti annidati
- Nel complesso, abbiamo usato:
  - MongoDB
  - Node.js con JavaScript
  - Python

Creare un Modulo

- Ma che cosa è un modulo?
- Un normalissimo programma, che contiene solo funzioni, salvato in un file con estensione .py
- L'istruzione «import» semplicemente importa il file (senza specificare l'estensione)
- Automaticamente, tutte le funzioni e classi definite nel file importato vengono assegnate al namespace del prefisso

Esempio

- Prepariamo un file «tcm.py»
- Contiene una sola funzione:

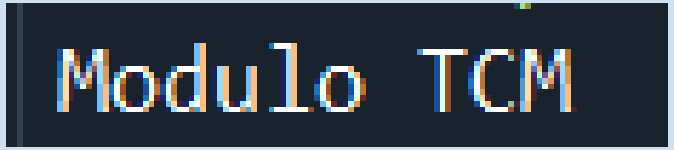
```
def tcm():
    return "Modulo TCM"
```

Esempio

- Creiamo un altro programma
- Importiamo e usiamo:

```
import tcm

print (tcm.tcm())
```



Map-Reduce

Paradigma di Programmazione

- Che cosa è Map-Reduce?
- È un «Paradigma di Programmazione»
- Cioè un modo di programmare
- Pensato per
  - Rendere paralleli gli algoritmi
  - Senza occuparsi del parallelismo

Scrivere Algoritmi Paralleli

- Scrivere algoritmi paralleli non è facile
- mon tanto per capire come distribuire il calcolo sui diversi processori
- quanto per coordinare l'attività e lo scambio di dati tra i processori stessi
- Chiunque abbia fatto questo lavoro, dice che la maggior parte del tempo dello sviluppo è dedicata allo scambio dei dati e al coordinamento dei processi

Scrivere Algoritmi Paralleli

- Conseguenza pratica:
- Solo pochi specialisti sono realmente in grado di scrivere algoritmi paralleli
- Lo sviluppo di algoritmi paralleli è estremamente lungo e (di conseguenza) costoso

Idea: Map-Reduce

- Dalle considerazioni precedenti nasce l'idea alla base di Map-Reduce:
- Separare
  - la definizione dei task da svolgere in modo parallelo
  - dalle attività di scambio di dati e coordinamento
- Lasciando quest'ultimo compito ad un framework software dedicato

Idea: Map-Reduce

Vantaggi per il programmatore

- Lo sforzo di programmazione è rivolto agli aspetti algoritmici
- Un algoritmo è composto da due soli tipi di attività primitive, cioè Map e Reduce, in genere di implementazione relativamente facile
- Lo sviluppo diventa, quindi, molto veloce e (di conseguenza) economico

Il Framework?

- Il framework di esecuzione fa il «lavoro sporco», cioè
- Distribuisce i i task sui processori effettivamente disponibili
- Distribuisce i dati da analizzare ai vari task
- Raccoglie i dati prodotti dai vari task
- Bilancia il carico di lavoro

Su Quale Piattaforma?

- I Framework Map-Reduce lavorano su «cluster» di nodi di una rete di computer
- Quindi, sono adatti per applicazioni dove i task paralleli non condividono memoria centrale
- Come conseguenza, i framework Map-Reduce sono diventati molto popolari nel mondo del Cloud Computing

Cloud Computing e Big Data

- Il mondo dei Big Data e della Data Science ha bisogno della tecnologia Map-Reduce
- Infatti, spesso il volume dei dati da trattare è troppo grande e non è possibile pensare di elaborarli con un approccio tradizionale
- Serve la potenza di calcolo di molte macchine che lavorano in modo parallelo all'elaborazione dei Big Data

Cloud Computing e Big Data

- Ma quante macchine servono?
- Dipende dal volume dei dati da trattare e dal tempo di elaborazione a disposizione:
- A parità di data set,
  - Più risorse di calcolo abbiamo a disposizione (computer/nodi)
  - Meno tempo durerà l’elaborazione

Cloud Computing e Big Data

- Ma quanto è grande il data set e quanti nodi abbiamo?
- L’approccio del cloud computing viene in aiuto:
  - Si attivano i nodi che servono
  - Per il solo tempo in cui servono

Framework Principali

- Quali sono i framework principali?
  - Apache Hadoop
  - Apache Spark
- Si differenziano in modo significativo, ne parleremo nella parte finale

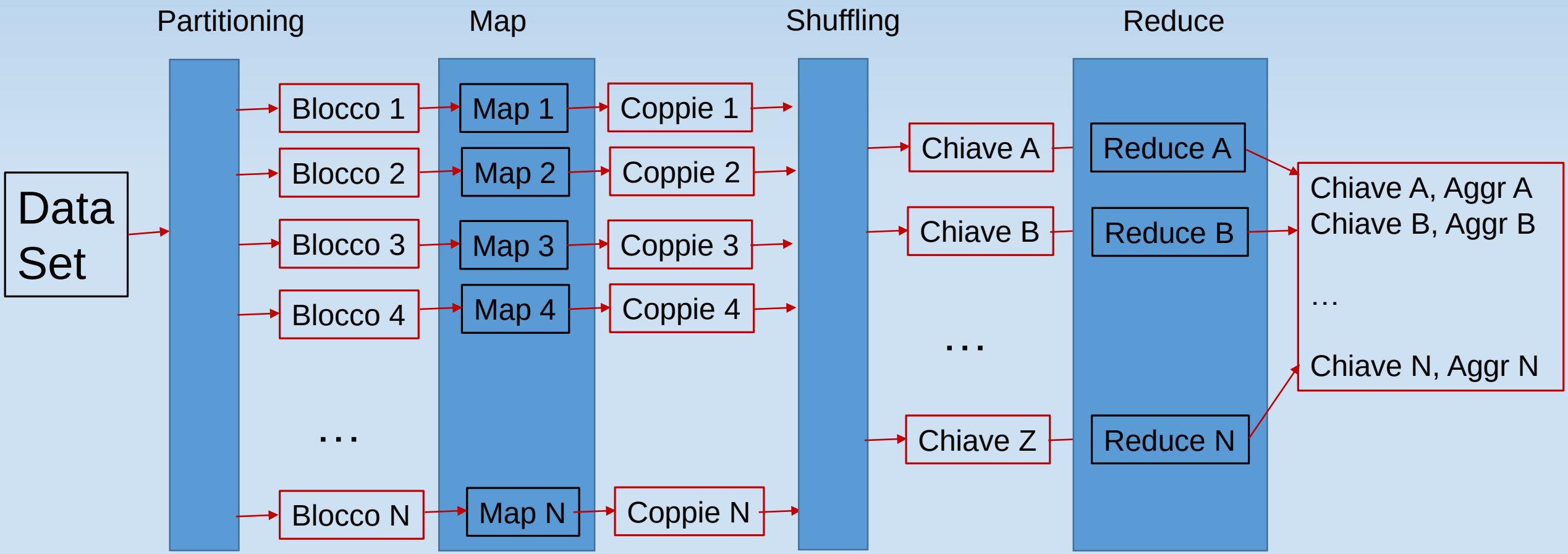
Il Paradigma Map-Reduce

- Come è organizzato un algoritmo Map-Reduce?
- Primitive di base (da scrivere):
- **Map**: un frammento del data set viene trasformato nella forma di coppie <chiave, valore>
- **Reduce**: l'intero insieme di coppie <chiave, valore> viene raccolto, in modo da aggregare tutte le coppie con lo stesso valore della chiave

Operazioni Svolte dal Framework

- **Partitioning**: il data set originale è partizionato in tanti blocchi relativamente piccoli
- **Map**: per ogni blocco di dati, un task di Map è eseguito (output: coppie <chiave, vaolre>)
- **Shuffling**: Le coppie <chiave, valore> sono raccolte: per ogni chiave si forma un blocco con tutte le coppie
- **Reduce**: un task di reduce è eseguito per ogni valore della chiave

Flusso Map-Reduce



Distribuzione dei Task

- Nelle due fasi parallelizzabili Map e Reduce
- Un task di Map viene attivato per ogni blocco
- Un task di Reduce viene eseguito per ogni valore della chiave
- Il framework cerca di eseguire il maggior numero di task in parallelo, ma dipende dalle risorse disponibili e dal tempo di esecuzione di ogni task

Macro-Fasi Map-Reduce

- Gli algoritmi Map-Reduce più semplici sono composti da una sola macro-fase Map-Reduce
- Ma si possono avere macro-fasi Map-Reduce multiple:
  - Uguali (tipicamente per gli algoritmi a convergenza, come il Page Rank)
  - Diverse (per esempio, pre-processing e elaborazione principale)

Modellare le Primitive

- Abbiamo quindi capito che l'attività di programmazione consiste nell'implementare le primitive Map e Reduce
- Mentre l'attività di design consiste nell'identificare le macro-fasi Map-Reduce
- Senza legarsi ad un linguaggio di programmazione particolare, conviene ragionare usando dello pseudo-codice

Pseudo-codice?

- Se non lo avete ancora visto,
- Per «pseudo-codice» si intende
- Un programma scritto in un linguaggio di programmazione infomale
- Ispirato a linguaggi reali
- Ma non rigorosamente vincolato alla sintassi e alla semantica di uno specifico linguaggio

Quale Pseudo-linguaggio

- È prassi ispirarsi al Pascal
- Usando le parole chiave **begin** e **end** per contenere i blocchi annidati
- Usando **function** per definire le funzioni e **procedure** per definire le procedure
- Usando **:=** come operatore di assegnamento e **=** come operatore di confronto
- Usando **var *nome* : *tip*** per definire una variabile

Esempio: Word Count

- Questo è un esempio tipico
- Dato un testo, contare le occorrenze di ogni singolo termine
- Perché Map-Reduce?
  - Con testi molto lunghi, con un elevato numero di parole, la struttura dati per memorizzare i contatori delle parole potrebbe saturare la memoria centrale

Esempio: Macro-Fasi

- Quante macro-fasi servono?
- Una, perché:
  - Nella fase di partitioning, il testo viene spezzato in blocchi, senza spezzare le parole, per esempio 1000 parole per blocco
  - La primitiva di map, per ogni parola nel testo, genera una coppia <termine, contatore> dove «contatore» è il numero di volte che il termine (parola) compare: al massimo, 1000 coppie
  - La primitiva di Reduce riceve un insieme di coppie <termine, contatore>, tutte con lo stesso valore per «termine». Sommando tutti i valori di «contatore» (semplice ciclo for), produce una coppia <termine, somma>

Primitiva Map

```
Procedure Map(input: Seq(termine),
                        output: Map(<termine,  contatore>) )
begin
 for each termine in input do
 begin
  if output.exists( termine ) then
   old := output.get( termine )
   output.replace(termine, old+1)
  else
    output.add(termine, 1)
 end
end
```

Primitiva Map

- La procedura riceve due parametri:
- Input: una sequenza di termini (per esempio, 1000 termini) estratti dal testo
- output: una mappa <termine, contatore>, dove termine è la chiave e contatore è il valore associato
- Un ciclo **for each** scandisce la sequenza di termini
- Per ogni termine, verifica se esiste già nella mappa:
  - Se esiste, incrementa il contatore
  - Se non esiste, lo inserisce con il contatore 1

Primitiva Reduce

```
Procedure Reduce(input: Seq(<termine, contatore>),
                                output: <terine, conteggio>)
begin
 var chiave: String
 var somma: integer
 somma := 0
 for each coppia in input do
 begin
  if somma = 0 then chiave = coppia.termine end if
  somma := somma + coppia.conteggio
 end
output.terine := chiave
output.conteggio := somma
end
```

Primitiva Reduce

- La procedura riceve due parametri:
- input: la sequenza di coppie con la stessa chiave, cioè «termine»
- output: una coppia (diciamo, passata per reference) che descrive l'output della procedura, cioè la chiave «termine» con il «conteggio»
- Nel ciclo for each, la procedura fa una semplice somma dei contatori di ogni coppia in input

Altro Esempio: Parole Frequenti

- Supponiamo di avere una collezione di documenti testuali
- Potremmo cercare:
- Le parole che in un singolo documento occorrono più di 10 volte
- Di queste, estrarre quelle che occorrono più di 10 volte in più di 10 documenti

Macro-Fai Map-Reduce

- Quante macro-fasi? Due
- Prima fase:
  Si estraggono le parole che occorrono più di 10 volte in un documento. Output: coppie <termine, documento>
- Seconda fase:
  Si estraggono le parole che occorrono in piu di 10 documenti. Output: singoli termini

Prima Fase

- Si partiziona il data set, in modo tale che ogni blocco di dati contenga una parte di documento. Un blocco è una sequenza di coppie <documento, termine>
- La primitiva di Map produce coppie
  <chiave: <termine, documento>, conteggio>
  dove la chiave è costituita dalla coppia termine, documento
- La primitiva di Reduce aggrega usando la chiave <termine, documento> e se un termine ha una somma di conteggio maggiore di 10 viene prodotta la coppia <termine, documento>, altrimenti non produce niente.

Seconda Fase

- Le coppie <termine, documento>, prodotte dalla prima fase, devono essere aggregate per contare il numero di documenti in cui un termine occorre
- Non serve alcuna fase di Map
- La fase di Reduce aggrega le coppie in base alla chiave «termine»; la primitiva di Reduce conta le coppie e, se sono più di 10, produce termine in output, altrimenti non produce nulla

Primitiva Map Fase 1

```
Procedure Fase1_Map(input: Seq(<documento, termine>),
     output: Map(<chiave:<termine, documento>, conteggio>))
begin
 for each p in input do
 begin
   if output.exists(<p.termine, p.documento>) then
    nuovo = output.get(<p.termine, p.documento>) + 1
    output.replace(<p.termine, p.documento>, nuovo)
 else
   output.add(<p.termine, p.documento>, 1)
 end if
 end
end
```

Primitiva Reduce Fase 1

```
Procedure Fase1_Reduce (
 input: Seq(<chiave:<termine, documento>, conteggio>)),
 output: <terine, documento>)
begin
 var chiave: <termine, documento>
 var somma: integer
 somma := 0
 for each coppia in input do
 begin
  if somma = 0 then chiave := coppia.chiave end if
  somma := somma + coppia.conteggio
 end
If somma > 10 then output := <chiave.termine, chiave. documento>
else output := null
end
```

Primitiva Reduce Fase 2

```
Procedure Fase2_Reduce (input: Seq(<terine, documento>),
                                output: String)
begin
 var chiave: String
 var somma: integer
 somma := 0
 for each coppia in input do
 begin
  if somma = 0 then chiave := coppia.termine end if
  somma := somma + 1
 end
If somma > 10 then output := termine
else output := null
end
```

Il Framework Hadoop

- Apache Hadoop è stato il primo framework a grande diffusione per Map-Reduce
- La pagina ufficiale del progetto è **`https://hadoop.apache.org/`**
- Linguaggi di programmazione:
  Java è il linguaggio nativo
  si può usare anche con molti altri linguaggi, ad esempio C++, C#, Python, Ruby, ecc.

Il Framework Hadoop

- Componenti principali:
- YARN: Yet Another Resource Negotiator
  è il componente software che gestisce la distribuzione dei task e dei dati sui nodi del cluster
- HDFS: Hadoop Distributed File System
  Il file systm distribuito di Hadoop, che viene usato per lo scmabio dei dati

HDFS

- Il file system distribuito di Hadoop è il componente che consente la distribuzione e la raccolta dei dati
- Il data set da elaborare deve essere pre-caricato su HDFS
- Tutti i blocchi da processare e le mappe <chiave, valore> vengono salvati su HDFS
- HDFS garantisce che vengano replicati e trasferiti su tutti i nodi della rete
- Soluzione molto efficace, che richiede poca memoria centrale sui nodi, ma lenta (la sincronizzazione del file system è un’attività costosa)

IL Framework Spark

- Apache Spark è un prodotto di impostazione più moderna, rispetto a Hadoop
- Sito ufficiale: **`https://spark.apache.org/`**
- Linguaggi di programmazione supportati: Java, Scala, Python, R

Il Framework Spark

- A differenza di Hadoop, Spark non è basato su un file system distribuito, come HDFS
- Per contro, lavora esclusivamente in memoria centrale
- Introduce il concetto di RDD, Resilient Distributed Dataset

Resilient Distributed Dataset

- Un RDD è un blocco di dati, sia in input a qualche task che di output
- Per ogni blocco, viene memorizzata la catena di elaborazione necessaria per calcolarlo
- Perché «resilient»? Perché se serve fare spazio in memoria centrale, cancellando un RDD, di queso rimane il modo in cui è stato calcolato.
- Se serve il suo contenuto, la catena di elabroazione viene rieseguita

Successo di Spark

- Il concetto di RDD è fondamentale per il successo di Spark
- Lavorando in memoria centrale, è molto più veloce di Hadoop, dicono da 10 a 100 volte più veloce.
- Alcuni ricercatori hanno verificato che con data set molto grossi, in relazione alla meoria centrale disponibile, il continuo ricalcolo degli RDD può rendere Spark più lento di Hadoop
- Ma la quantità di memoria centrale ora disponibile sposta questo limite molto in alto …

Altri Vantaggi di Spark

- Con Spark, non è necessario ragionare secondo il paradigma Map-Reduce
- **Spark SQL** è un potente linguaggio di trasformazione di dati in forma tabellare:
  - Il programmatore specifica le trasformazioni sulla tabelle (tipo algebra relazionale)
  - Il sistema le implementa come macro-fasi multiple di tipo Map-Reduce

Altri Vantaggi di Spark

- **Streaming:** un’altro servizio costruito sopra il layer Map-Reduce consente di gestire flussi di streaming
- A questo si aggiungono le librerie «ML», che forniscono molti algoritmi di Machine Learning
- In definitiva, Spark è un ambiente potente per realizzare applicazioni di Data Science sui Big Data

Spark con Hadoop

- Più che essere in concorrenza con Hadoop
- Sparrk lo integra
- Infatti, Spark non ha un file system distribuito
- Quindi può attingere ai file memorizzati in HDFS, al fine di ottenere il servizio di persistenza

Conclusioni

- Nel laboratorio vedrete un piccolo esempio di uso di Spark
- Nella Laurea Magistrale, vedrete l’utilizzo del linguaggio Scala con Spark
- Sia Hadoop che Spark sono diventati strumenti importanti nel mondo della Data Science
- Sono competenze molto ricercate dalle aziende di un certo livello