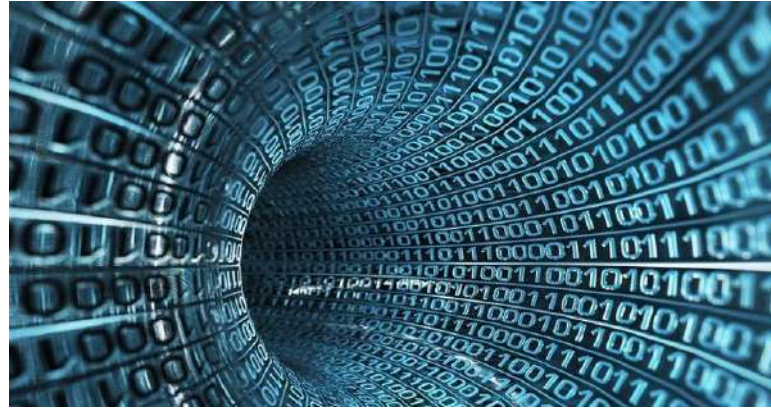


Introduzione alla Cyber Security e ai Big Data



MapReduce – Esempi

Prof. Devis Bianchini

Università degli Studi di Brescia

Dipartimento di Ingegneria dell'Informazione



Call Record Data (CDR) Analytics

Si consideri il file (di testo) seguente, che rappresenta i tabulati telefonici di una compagnia di telecomunicazioni:

```
FromPhoneNumber|ToPhoneNumber|CallStartTime|CallEndTime|STDFlag
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128505|8983006310|2015-03-01 07:08:10|2015-03-01 08:12:15|0
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 09:12:15|1
9665128505|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|0
9665128506|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
9665128507|8983006310|2015-03-01 09:08:10|2015-03-01 10:12:15|1
```



Call Record Data (CDR) Analytics

- Il flag **STDFlag** identifica le chiamate a lunga distanza o effettuate dall'estero
- Identificando i numeri telefonici che effettuano questo numero di chiamate, la compagnia telefonica intende offrire agli utenti titolari dei numeri delle offerte più convenienti
- Si richiede di progettare un programma **Map-Reduce** che identifichi tutti i numeri telefonici che effettuano queste chiamate per più di 60 minuti



Map

- **Input:** tabulati telefonici
- **Output:** coppie **<FromPhoneNumber, Duration>** che, per ogni numero telefonico in un record, riporta la durata (in minuti) calcolata come **CallEndTime-CallStartTime**, solo se **STDFlag==1**

Reduce

- Semplicemente una somma, seguita da un filtro in uscita per scartare i numeri telefonici che complessivamente fanno meno di 60 minuti di chiamata

Pseudo-codice

```
<String, int> map(Object key, String record)
{
    String[] parts = record.split('|');
    if(parts[4]==1)
    {
        emit <parts[0], convertToMinutes(parts[3])-convertToMinutes(parts[2])>;
    }
}

<String, int> reduce(String phoneNum, int[] durations)
{
    int totalMinutes = 0;
    foreach (elem in durations)
    {
        totalMinutes += elem;
    }
    if(totalMinutes>=60)
    {
        emit <phoneNum,totalMinutes>;
    }
}
```



LastFM Listener per Track

Si consideri il file (di testo) seguente, estratto da un sito web, che riporta per ogni **Track** musicale l'utente che ha ascoltato online il **Track**, se il **Track** era ascoltato in maniera condivisa, se il **Track** è stato ascoltato in radio, se l'utente ha saltato il **Track** prima di finirne l'ascolto:

UserId	TrackId	Shared	Radio	Skip
111115	222	0	1	0
111113	225	1	0	0
111117	223	0	1	1
111115	225	1	0	0



LastFM Listener per Track – ES1 ... ES4

- Trovare il numero di volte che un **Track** è stato ascoltato in maniera condivisa tra più utenti
- Trovare il numero di volte che un **Track** è stato ascoltato in radio
- Trovare il numero di volte che un **Track** è stato ascoltato in totale
- Trovare il numero di volte che un **Track** è stato interrotto mentre era ascoltato in radio



Pseudo-codice – ES1

```
<String, int> map(Object key, String record)
{
    String[] parts = record.split('|');
    if(parts[2]==1)
    {
        emit <parts[1], 1>;
    }
}

<String, int> reduce(String trackID, int[] sharedCounter)
{
    int num_of_shared = 0;
    foreach (elem in sharedCounter)
    {
        num_of_shared += elem;
    }
    emit <trackID,num_of_shared>;
}
```


Pseudo-codice – ES2

```
<String, int> map(Object key, String record)
{
    String[] parts = record.split('|');
    if(parts[3]==1)
    {
        emit <parts[1], 1>;
    }
}

<String, int> reduce(String trackID, int[] radioCounter)
{
    int num_of_radio = 0;
    foreach (elem in radioCounter)
    {
        num_of_radio += elem;
    }
    emit <trackID,num_of_radio>;
}
```

Pseudo-codice – ES3

```
<String, int> map(Object key, String record)
{
    String[] parts = record.split('|');
    emit <parts[1], 1>;
}

<String, int> reduce(String trackID, int[] counter)
{
    int totalSongs = 0;
    foreach (elem in counter)
    {
        totalSongs += elem;
    }
    emit <trackID,totalSongs>;
}
```


Pseudo-codice – ES4

```
<String, int> map(Object key, String record)
{
    String[] parts = record.split('|');
    if(parts[4]==1)
    {
        emit <parts[1], 1>;
    }
}

<String, int> reduce(String trackID, int[] radioSkipped)
{
    int num_of_skipped = 0;
    foreach (elem in radioSkipped)
    {
        num_of_skipped += elem;
    }
    emit <trackID,num_of_skipped>;
}
```

LastFM Listener per Track – ES5

Trovare il numero di volte che un **Track** è stato ascoltato (da utenti diversi, escludendo i doppioni)



111115	222	0	1	0	222	1
111113	225	1	0	0	223	1
111117	223	0	1	1	225	2
111115	225	1	0	0		

Pseudo-codice – ES5

```
<String, String> map(Object key, String record)
{
    String[] parts = record.split('|');
    emit <parts[1], parts[0]>;
}
```

```
<String, int> reduce(String trackID, String[] users)
{
    Set<String> usersSet = new Set<String>();
    foreach (user in users)
    {
        usersSet.add(user);
    }
    emit <trackID,usersSet.size()>;
}
```



L'uso di Set<> impedisce
che un utente già presente
nell'insieme sia aggiunto più
volte

Esercizio – Inverted Index

Dato un insieme di pagine Web:

[URL: ID; Content: page]

costruire un «inverted index»

[(k₁, [u₁, ...u_k]) , ... , (k_n, [u₁, ...u₁])]

dove **k₁, ...k_n** sono keyword e **u₁, ...u_k** sono gli URL delle pagine Web che contengono la keyword **k₁**

Inverted Index – Pseudo-codice

```
<String, String> map(Object key, Object record)
{
    String[] words = StringTokenizer(record.Content);
    while (words.hasMoreTokens())
    {
        emit <words.nextToken(), record.URL>;
    }
}
```



Anche l'uso di `split(' ')`
è accettabile come soluzione

```
<String, String[]> reduce(String word, String[] urls)
{
    emit <word,urls>;
}
```

La funzione `reduce` è la
funzione identità

Calcolo della media in Map-Reduce (I)

Si consideri il file (di testo) seguente, che riporta per ogni record la temperatura minima e quella massima (potrebbero esserci più record per lo stesso giorno); progettare un programma **Map-Reduce** che riporta la temperatura massima media per ogni mese

```
DDMMYYYY, MIN, MAX  
  
01012000, -4.0, 5.0  
  
02012000, -5.0, 5.1  
  
03012000, -5.0, 7.7  
  
...  
  
29122013, 3.0, 9.0  
  
30122013, 0.0, 9.8  
  
31122013, 0.0, 9.0
```



Calcolo della media in Map-Reduce (II)

- **Map**: restituisce le coppie **<mese, temperatura_max>** per ogni record
- **Reduce**: effettua prima la somma e successivamente la media

```
<String, decimal> map(Object key, String record)
{
    String[] parts = record.split(',');
    String meseAnno = estraiMeseAnno(parts[0]);
    emit <meseAnno, parts[2]>;
}

<String, decimal> reduce(String meseAnno, decimal[] tempMax)
{
    decimal sum = 0;
    foreach(temp in tempMax)
    {
        sum += temp;
    }
    emit <meseAnno, sum/tempMax.size>;
}
```

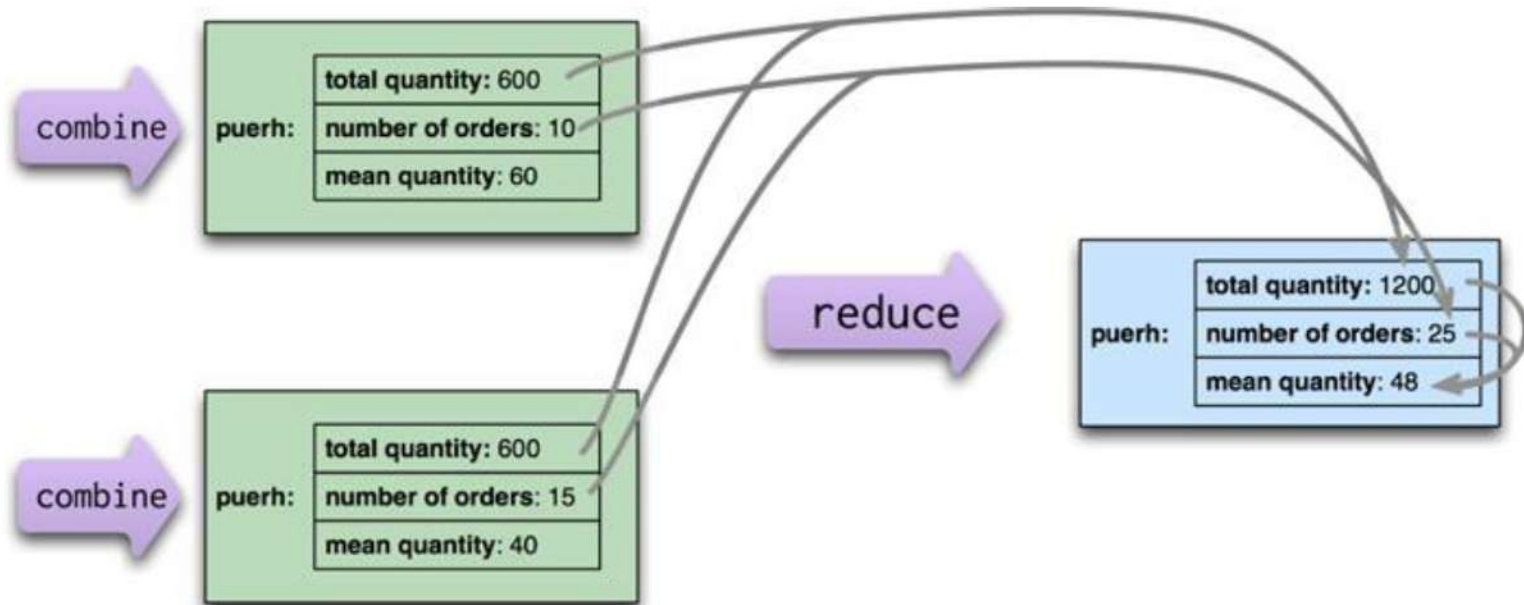


Calcolo della media in Map-Reduce (III)

- **Map**: restituisce le coppie **<mese, temperatura_max>** per ogni record
- **Reduce**: effettua prima la somma e successivamente la media
- La funzione **Map** non contribuisce a ridurre sensibilmente il volume dei dati
- E se usassimo la stessa funzione di **Reduce** anche per il **Combine**?
 - Non è possibile, in quanto la media non è una funzione che gode della proprietà associativa
 - Tuttavia, potremmo combinare le operazioni di somma e di divisione: (a) su ogni nodo viene effettuata la somma parziale e viene contato il numero di record; (b) la funzione di **Reduce** effettua una divisione, con al numeratore la «somma delle somme» e al denominatore il numero totale di record



Quale Combiner per calcolare una media?



Calcolo della media in Map-Reduce (IV)

```
<String, decimal> map(Object key, String record)
{
    String[] parts = record.split(',');
    String meseAnno = estraiMeseAnno(parts[0]);
    emit <meseAnno, parts[2]>;
}
```

```
<String, TempStruct> combine(String meseAnno, decimal[] tempMax)
{
    decimal sum = 0;
    foreach(temp in tempMax)
    {
        sum += temp;
    }
    emit <meseAnno,{sommaTemp:sum,numTotTemp:tempMax.size}>;
}
```



Oggetto di tipo TempStruct
con due proprietà:

- sommaTemp
- numTotTemp

Calcolo della media in Map-Reduce (V)

```
<String, decimal> reduce(String meseAnno, TempStruct[] tempMaxStruct)
{
    decimal sum = 0;
    int total = 0;

    foreach(elem in tempMaxStruct)
    {
        sum += elem.sommaTemp;
        total += elem.numTotTemp;
    }

    emit <meseAnno,sum/total>;
}
```



Array di oggetti di
tipo TempStruct

Calcolo della media in Map-Reduce (VI)

- **Map**: restituisce le coppie **< mese, temperatura_max >** per ogni record
- **Reduce**: effettua prima la somma e successivamente la media
- E se volessimo misurare la differenza di temperatura massima media, mese per mese, del 2022 rispetto al 2000?
 - Serve un programma **Map-Reduce** a «*doppio job*»
 - Il primo «*job*» **Map-Reduce** filtra le temperature massime per ogni singolo mese di ogni anno (aggiungendo il filtraggio per anno, per considerare solo il 2000 e il 2022)
 - Il secondo «*job*» **Map-Reduce** calcola le temperature medie del 2000 e del 2022 e confronta i dati del 2022 con quelli del 2000 (senza **Combiner**)



Calcolo della media in Map-Reduce (VII)

```
<String, decimal> map(Object key, String record)
{
    String[] parts = record.split(',');

    int anno = estraiAnno(parts[0]);
    if((anno == 2000) || (anno == 2022))
    {
        String meseAnno = estraiMeseAnno(parts[0]);
        emit <meseAnno, parts[2]>;
    }
}


<String, TempStruct> combine(String meseAnno, decimal[] tempMax)
{
    decimal sum = 0;
    foreach(temp in tempMax)
    {
        sum += temp;
    }
    emit <meseAnno, {sommaTemp:sum, numTotTemp:tempMax.size}>;
}
```

Calcolo della media in Map-Reduce (VIII)

```
<String, TempStruct> reduce(String meseAnno, TempStruct[] tempMaxStruct)
{
    decimal sum = 0;
    int total = 0;

    foreach(elem in tempMaxStruct)
    {
        sum += elem.sommaTemp;
        total += elem.numTotTemp;
    }

    emit <meseAnno,{sommaTemp:sum,numTotTemp:total}>;
}
```



Non viene calcolata
ancora la media,
perché seguirà un
altro job *Map-Reduce*



Calcolo della media in Map-Reduce (IX)

```
<int, TempStructComplex> map(String meseAnno, TempStruct temp)
{
    int anno = estraiAnno(meseAnno);
    int mese = estraiMese(meseAnno);
    if(anno == 2000)
    {
        emit <mese,{sommaTemp2000:temp.sommaTemp,totTemp2000:temp.numTotTemp,
                    sommaTemp2022:0,totTemp2022:0}>;
    }
    else {
        emit <mese,{sommaTemp2000:0,totTemp2000:0,
                    sommaTemp2022:temp.sommaTemp,totTemp2022:temp.numTotTemp}>;
    }
}
```



Oggetto di tipo
TempStructComplex con
quattro proprietà:

- sommaTemp2000
- totTemp2000
- sommaTemp2022
- totTemp2022



Calcolo della media in Map-Reduce (X)

```
<int, Statistica> reduce(int mese, TempStructComplex[] temperature)
{
    decimal sum2000,sum2022 = 0;
    int total2000,total2022 = 0;

    foreach(elem in temperature)
    {
        sum2000 += elem.sommaTemp2000;
        sum2022 += elem.sommaTemp2022;
        total2000 += elem.totTemp2000;
        total2022 += elem.totTemp2022;
    }

    emit <mese,{tempMedia2000:sum2000/total2000,tempMedia2022:sum2022/total2022,
        variazione:(tempMedia2022-tempMedia2000)}>;
}
```

Oggetto di tipo Statistica
con tre proprietà:

- tempMedia2000
- tempMedia2022
- variazione



Design Pattern in Map-Reduce

- **Map-Reduce** è un framework, non uno strumento
 - È necessario adattare il proprio algoritmo al framework e non viceversa
 - In alcune situazioni potrebbe essere non banale
- Spesso può essere utile spezzare l'algoritmo in step di filtraggio e aggregazione
 - Il filtraggio diventa parte della funzione di **Map**
 - L'aggregazione diventa parte della funzione di **Reduce**
- Talvolta può essere necessario creare diversi job **Map-Reduce**
- **Map-Reduce** non è la soluzione ad ogni problema, ha senso principalmente quando:
 - I file di partenza sono grandi e «aggiornati» poco frequentemente
 - È necessario iterare su tutti i file per generare delle proprietà interessanti dei dati contenuti in tali file

Filtering Design Pattern

- Obiettivo: trovare record/file/tuple con particolari caratteristiche
- Esempi:
 - ispezionare Web logs in cerca di un indirizzo IP o di una stringa specifica
 - trovare i file che contengono la parola «Apple» o «Jobs»
- **Map**: applica il filtro (e questo è tipicamente la maggior parte del lavoro)
- **Reduce**: potrebbe semplicemente essere la funzione identità

Summarization Design Pattern

- Obiettivo: calcolare il minimo, il massimo, la media, la somma, etc. su un insieme di valori
- Esempi:
 - contare il numero di richieste HTTP per un certo sito Web
 - calcolare il numero medio di richieste HTTP per pagina per sito Web
- **Map**: seleziona le coppie $\langle k, v \rangle$, dove « v » è uno dei valori su cui applicare l'aggregazione
 - Può essere molto semplice, come una funzione identità
- **Reduce**: partendo dalle coppie $\langle k, [v] \rangle$, applica la funzione di aggregazione



Esempio – Top-k (I)

Dato un insieme di record di impiegati:

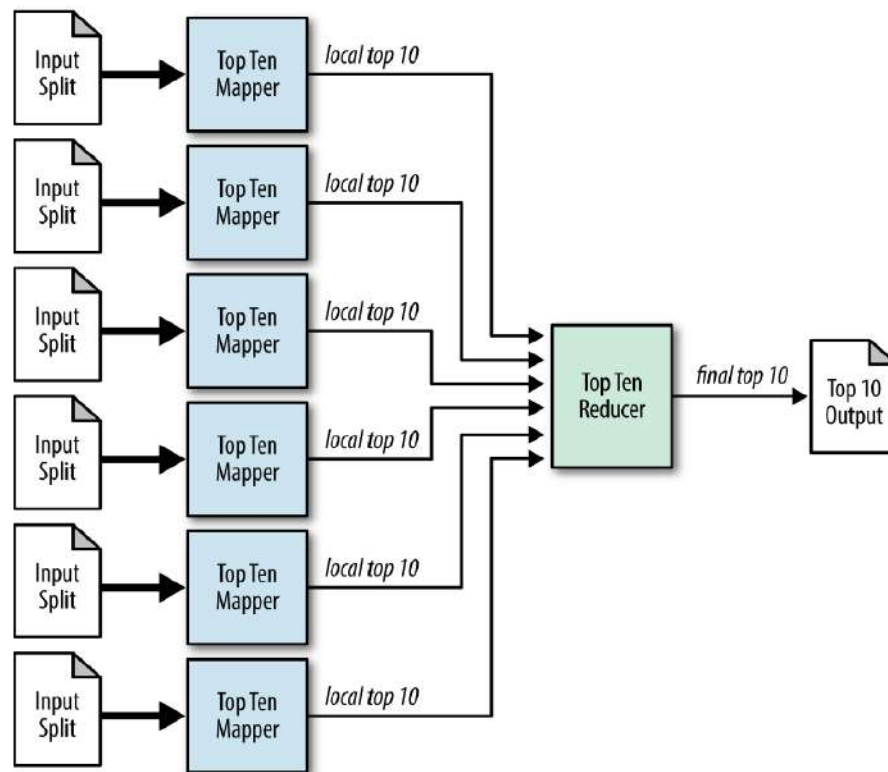
[Name: n; Age: a; Salary: s]

trovare i primi dieci impiegati (top-10) più giovani di 30 anni con i salari più alti



Esempio – Top-k (II)

- **Map:** applica il filtro (**Age<30**) e restituisce le coppie **<record_Id, Salary>**
- **Combine:** effettua un ordinamento «locale» e tronca la lista dopo i primi 10 (**k**)
- **Reduce:** effettua un ordinamento «globale» e tronca la lista dopo i primi 10 (**k**)



Esempio – Top-k (III)

```
<String, decimal> map(Object key, Object record)
{
    if(record.Age <= 30) {
        emit <record.Name, record.Salary>;
    }
}
```



Esempio – Top-k (IV)

La HashMap elenco è definita al di fuori di tutti i metodi, per essere condivisa tra i metodi stessi

```
HashMap elenco = new HashMap;  
<String, decimal> combine(String nomeImpiegato, decimal[] stipendi)  
{  
    elenco.add(<nomeImpiegato,stipendi[0]>);  
}
```

```
void cleanup()  
{  
    elenco.sortByValue(descending);  
    int counter = 0;  
    for(key in elenco.keySet())  
    {  
        if(counter++ == 10)  
        {  
            break;  
        }  
        emit <key,elenco.get(key)>;  
    }  
}
```

Il metodo cleanup è eseguito alla fine, dopo che il metodo combine è stato applicato a tutte le coppie <chiave, valore> in ingresso

Esempio – Top-k (V)

```
HashMap elenco = new HashMap;  
<String, decimal> reduce(String nomeImpiegato, decimal[] stipendi)  
{  
    elenco.add(<nomeImpiegato,stipendi[0]>);  
}  
  
void cleanup()  
{  
    elenco.sortByValue(descending);  
    int counter = 0;  
    for(key in elenco.keySet())  
    {  
        if(counter++ == 10)  
        {  
            break;  
        }  
        emit <key,elenco.get(key)>;  
    }  
}
```

Join Design Pattern

- Obiettivo: combinare tuple rispetto a dei valori condivisi
 - I valori condivisi possono essere identici oppure soddisfare un determinato predicato
- Esempi:
 - trovare tutti i documenti con le parole «big» e/o «data» partendo da un «inverted index»
 - Trovare tutti i professori e gli studenti che tengono/frequentano gli stessi corsi e produrre le coppie **<professore, studente>**
- **Map**: genera le coppie **<k, e>**, per ogni elemento «e» nell'input, dove «k» è il valore condiviso
- **Reduce**: genera una sequenza di coppie **[<k₁, r₁>, ..., <k_n, r_n>]** per ogni **<k, [e₁, ..., e_k]>** in input



Esempio: Relational Join

- **$R(AB) \text{ JOIN } S(BC)$**
- **Map:** for ogni tupla (a, b) di **R**, produce la coppia $\langle b, (R, a) \rangle$; per ogni tupla (b, c) di **S**, produce la coppia $\langle b, (S, c) \rangle$
- **Reduce:** ogni chiave «**b**» viene associato ad una lista di valori che possono essere « (R, a) » oppure « (S, c) »; per ogni elemento « (R, a) » e « (S, c) » nella lista, genera la coppia $\langle k, (a, b, c) \rangle$
- Esempio:
 - **$R(AB) = \{(a, b), (c, b)\} \text{ JOIN } S(BC) = \{(b, e), (f, g)\}$**
 - **Map:** $\langle b, (R, a) \rangle, \langle b, (R, c) \rangle, \langle b, (S, e) \rangle, \langle f, (S, g) \rangle$
 - **Shuffle and Sort:** $\langle b, [(R, a), (R, c), (S, e)] \rangle, \langle f, [(S, g)] \rangle$
 - **Reduce:** $\langle k1, (a, b, e) \rangle, \langle k2, (c, b, e) \rangle$



Alcuni riferimenti

