



HADOOP
Dariusz Żbik



DLACZEGO?

Przetwarzanie dużych zbiorów danych

Przykład – wyszukiwanie w zbiorze 100TB

- 1 węzeł @ 40MB/s -> 30 dni
 - MTBF ~ 3 lata
- 1000 węzłów @ 40MB/s -> 44 minuty
 - MTBF ~ 1 dzień

Potrzebny framework do obliczeń

- wydajny
- niezawodny
- łatwy w użyciu

JAK?

- klaster zbudowany z PC
- rozproszony system plików
- rozproszony framework obliczeniowy (MapReduce)
- narzędzia OpenSource



KTO I PO CO?

- wyszukiwanie (distributed grep)
- sortowanie
- konstruowanie indeksów
- kategoryzacja dokumentów
- agregacja statystyka

Kto używa?

- SoftwareMind
- Google
- Yahoo

MapReduce – CECHY

Zapewnia:

- skalowalność
- niezawodność

Wymaga:

- zapisanie obliczenia jako ciąg operacji:
 - map()
 - reduce()

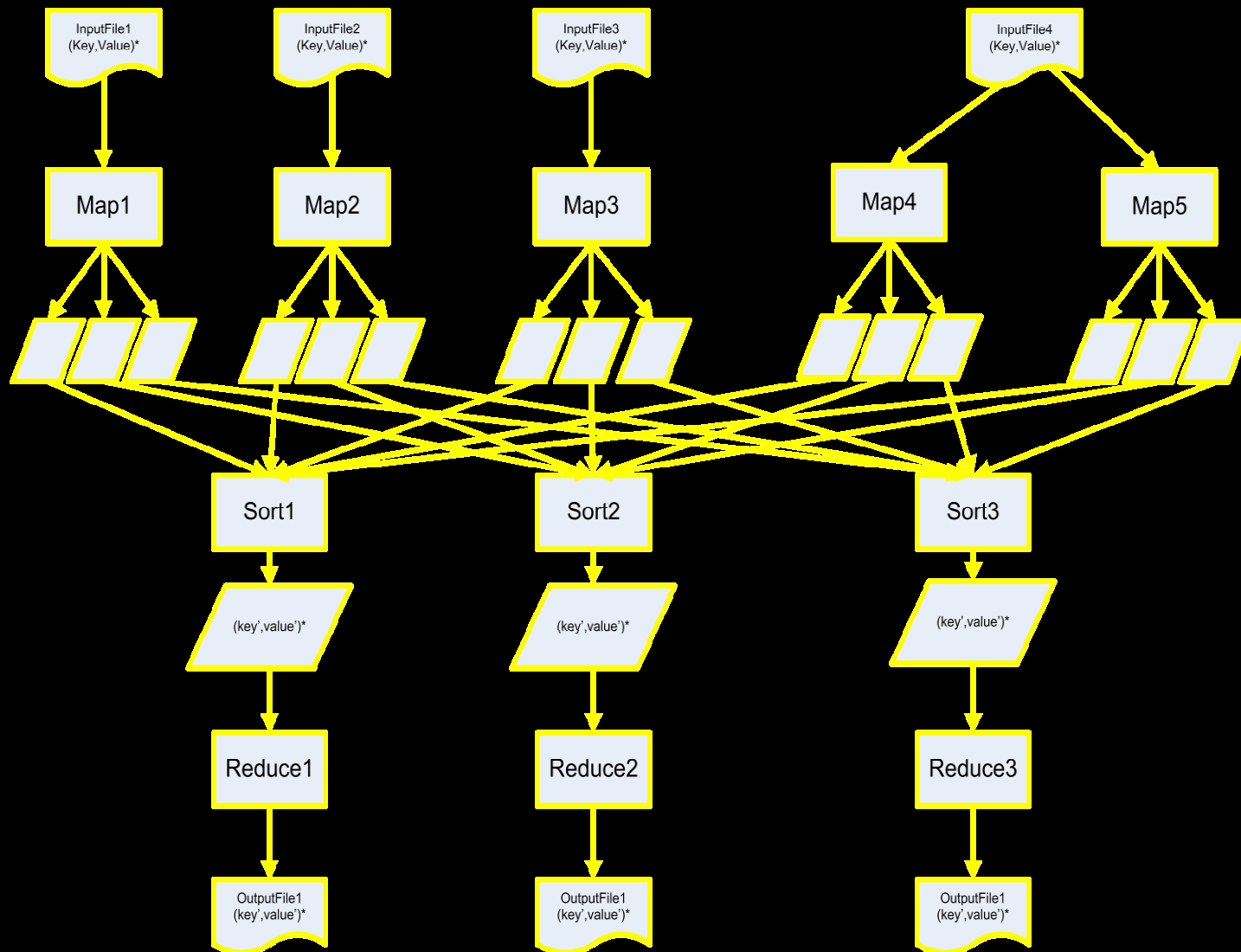
MapReduce – TEORETYCZNIE

- map: $(k,v) \rightarrow \text{list}(k', v')$
- wewnętrzna operacja grupowania/sortowania
- reduce: $(k', \text{list}(v')) \rightarrow \text{list}(k', v')$

MapReduce – TEORETYCZNIE (2)

- czytanie wejścia – input reader
- operacja Map
- podział na kawałki – partitioner
- sortowanie i grupowanie
- operacja Reduce
- zapis wyników

MapReduce – TEORETYCZNIENIE (3)



STATYSTYKA SŁÓW

map:

- input: linia tekstu
- operacja: podział na wyrazy
- output: wyraz, liczba wystąpień

reduce:

- input: wyraz, lista liczby wystąpień
- operacja: sumowanie
- output: wyraz, całkowita liczba wystąpień

OBIEKTY K,V

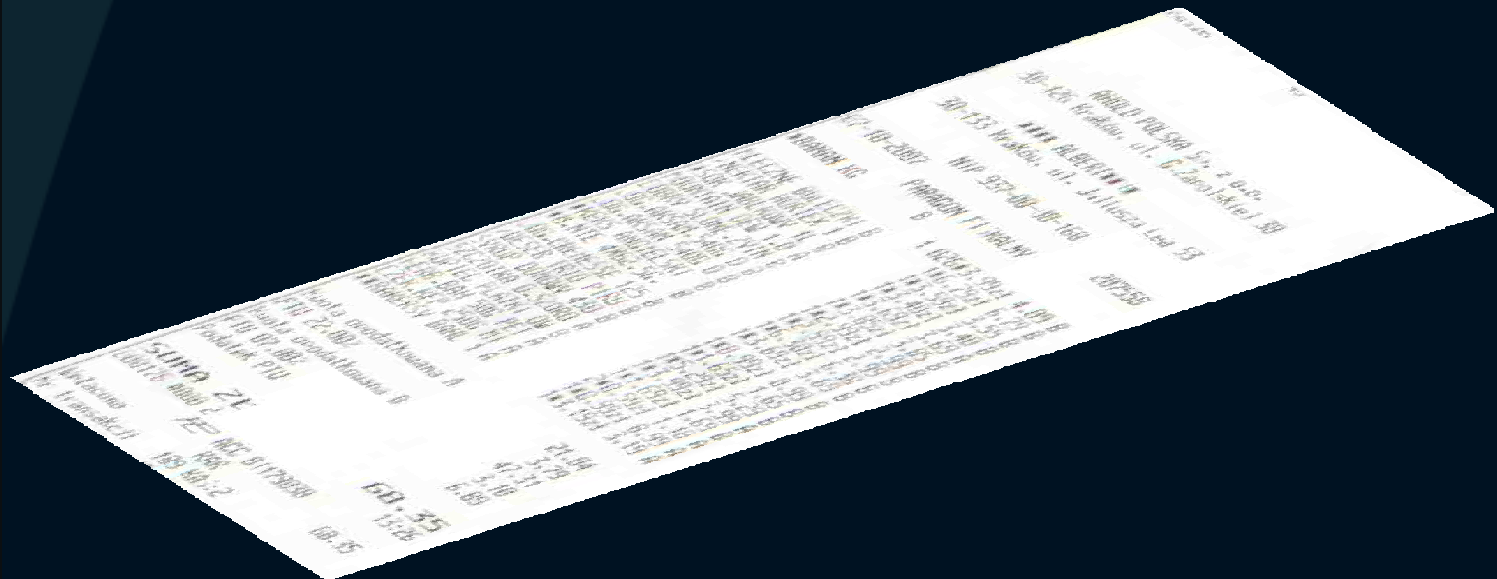
- serializacja/deserializacja
- relacja porządkująca (dla kluczy)

W praktyce wypełnienie interfejsów:

- `org.apache.hadoop.io.Writable`
 - `write(DataOutputStream)`
 - `readFields(DataInputStream)`
- `org.apache.hadoop.io.WritableComparable`
 - `write()/readFields()`
 - `compareTo()` z interfejsu `java.lang.Comparable`

ANALIZA KOSZTKA - PROBLEM

- dane źródłowe: wykonane operacje kasowe
- cel: znalezienie produktów powiązanych
- problem: skala



KOSZTYK SQLem

```
create table items (item_id serial, prod_id int,  
                    transaction_id int);  
  
-- insert into items(product_id, transaction_id) values(...);  
  
select  
    i1.prod_id,  
    i2.prod_id,  
    count(distinct(i1.transaction_id))  
from  
    items as i1  
    join items as i2  
        on i1.transaction_id = i2.transaction_id  
        and i1.prod_id > i2.prod_id  
group by i1.prod_id, i2.prod_id;
```

KOSZYK – Hadoop

- operacja pomocnicza – pobranie danych z bazy
 - `select * from items group by transaction_id`
- zapis w dogodnej postaci:
 - (k)klucz: `transaction_id`
 - (v)wartość: lista identyfikatorów produktów
- podział na osobne [pliki](#)



KOSZYK – Hadoop – MAP

input

- k: transaction_id
- v: list(prod_id)

output

- k': (prod_id,prod_id)
- v': int // 1

po ludzku:

- wejście – rachunek
- operacja – wyszukanie par na rachunku
- wyjście – para, liczebność (=1)

KOSZYK – Hadoop – MAP

```
public class BasketMap extends MapReduceBase implements Mapper {
    ProductsPair pair = new ProductsPair();
    LongWritable one = new LongWritable(1);

    public void map(WritableComparable key, Writable values,
        OutputCollector output, Reporter reporter) throws IOException {
        Integer[] items = ((BillItems)values).getProducts();

        for (int i=0; i<items.length; i++) {
            for (int j=i+1; j<items.length; j++) {
                if ( items[i].intValue() < items[j].intValue() ) {
                    pair.setProducts(items[i], items[j]);
                    output.collect(pair, one);
                }
            }
        }
    }
}
```

KOSZYK – Hadoop – Reduce

input

k': (prod_id,prod_id)

v': lista(long)

output

k': (prod_id,prod_id)

v': long //całkowita l.w.

po ludzku:

- wejście – para, lista(liczb wystąpień)
- operacja – podsumowanie
- wyjście – para, całkowita liczebność

Implementacja:

org.apache.hadoop.mapred.lib.LongSumReducer

PRZERWA NA REKLAMĘ





DFS – Distributed File System

- dane są przechowywane na wielu węzłach (Distributed)
- dane dostępne z wielu węzłów
- niezawodność (fault tolerant)



HDFS – UPROSZCZENIA

- plik jest „jednokrotnego” zapisu
 - brak możliwości dopisywania do pliku
 - brak możliwości modyfikowania pliku
 - brak operacji seek() podczas zapisu
 - zapis jest wykonywany przez **jeden** proces
- optymalizacja dla dużych plików
 - nieefektywność w przypadku dużej liczby małych plików

DZIĘKUJĘ ZA UWAGĘ

- <http://lucene.apache.org/hadoop>
- MapReduce Tools for Eclipse
- PigLatin
 - <http://wiki.apache.org/incubator/PigProposal>
 - <http://research.yahoo.com/project/pig>
- Hbase