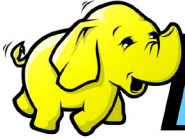


Databázové systémy

MapReduce a  ***hadoop***

NoSQL a agregáty - objednávka

```
{  
  "id" : 99,  
  "customer_id" : 1,  
  "orderItems": [{"productId": 27, "price": 35.47,  
    "productName": "NoSQL Distilled" }, {...}],  
  "shippingAddress" : {"city": {...}},  
  "paymentInfo": {...}  
}
```

NoSQL a agregáty - objednávka

```
{  
  "id" : 99,  
  "c"   Super pre zobrazenie objednávok  
  "o"     
  "p"   Horšie pre prehľad objednávok pre  
  "s"   jednotlivé produkty  
  "paymentInfo": {...}  
}
```

Sme v distribuovanom svete

- Dáta sú distribuované na viacerých uzloch
 - Dáta jedného používateľa práve na jednom uzle
 - Dáta jedného produktu na všetkých uzloch


Sme v distribuovanom svete

- Dáta sú distribuované na viacerých uzloch
 - Dáta jedného používateľa práve na jednom uzle
 - Dáta jedného produktu na všetkých uzloch
- Je výhodnejšie presunúť program k dátam ako dáta k programu
 - Program počíta agregáciu, ktorú chceme vedieť
 - Presúvať agregáciu (čiastkovú) je lacnejšie

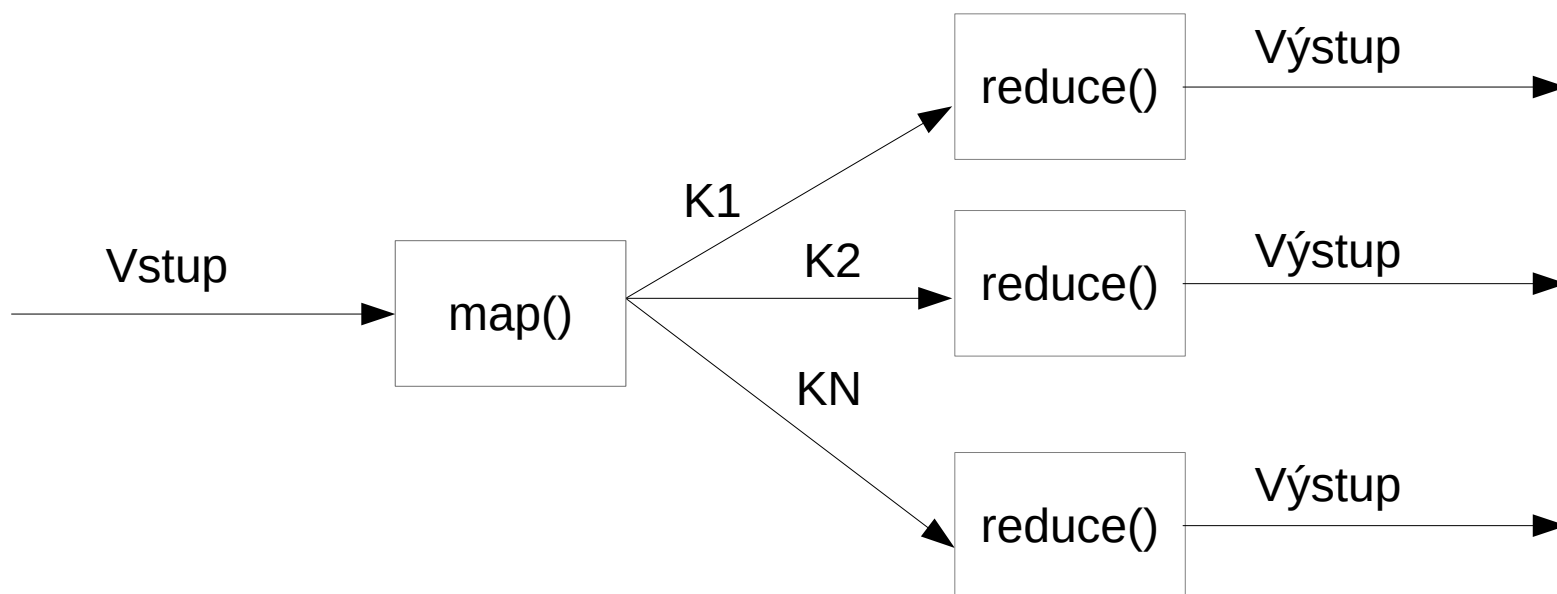
MapReduce výpočtový model

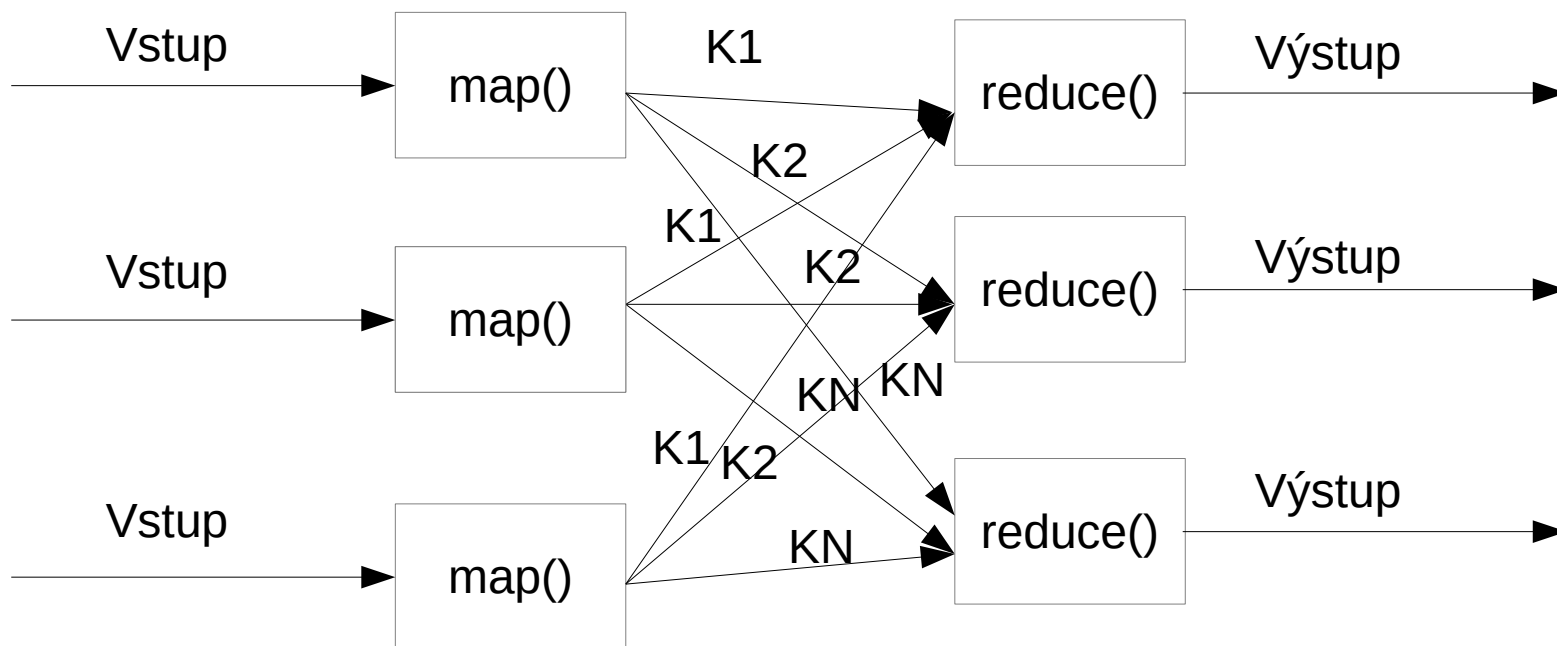
- Akýkoľvek výpočet nad dátami rozdelím na dve fázy
 - Map – beží pri dátach, v jednotlivých uzloch
 - Reduce – redukujem (agregujem) výstupy z map fázy
- Niektoré NoSQL databázy majú MapReduce “built-in” v sebe
- Použiteľný aj samostatne...

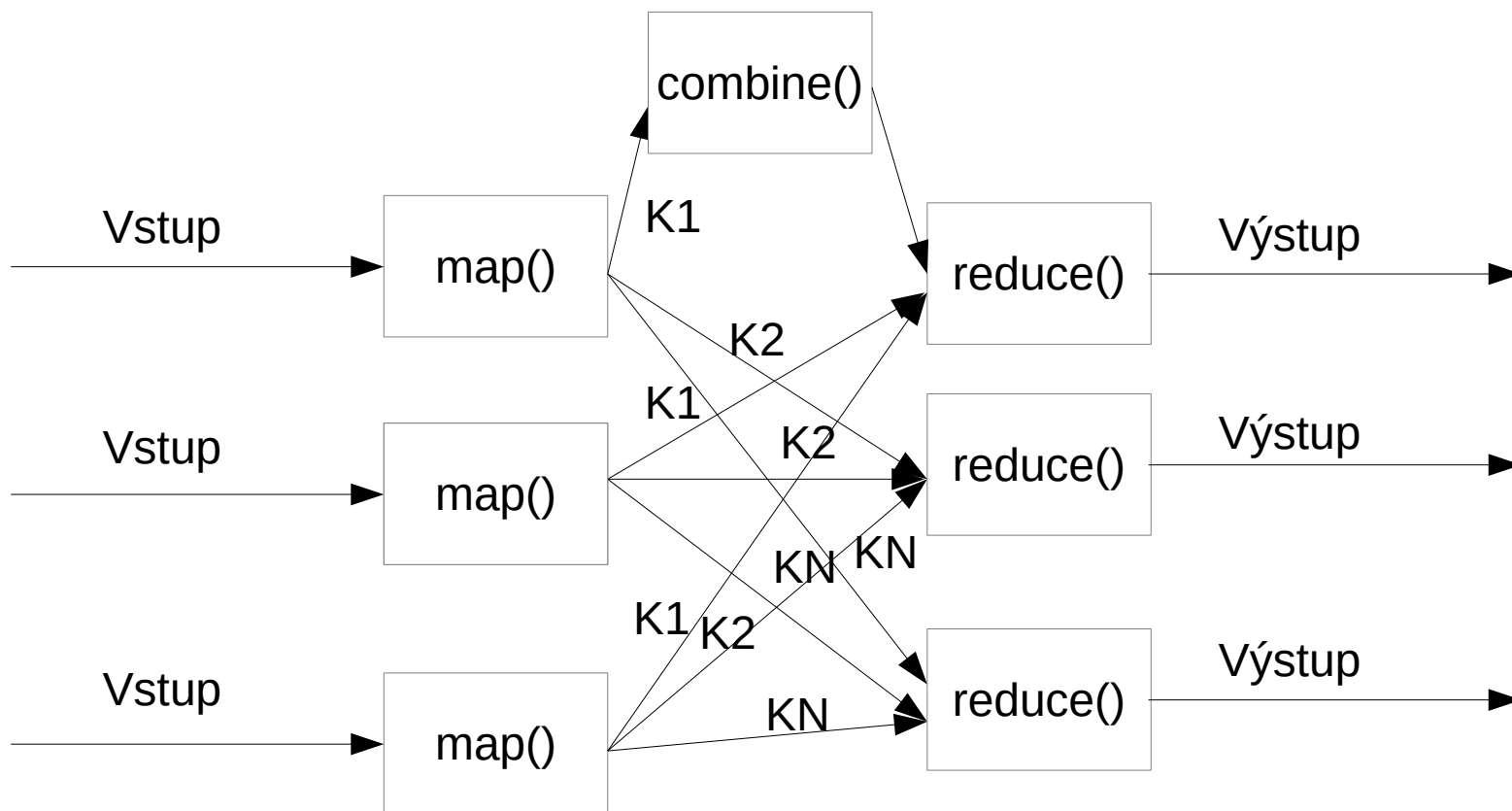
MapReduce framework

- Originál z Google, open source 
- žiaden dátový model, všetko v súboroch
 - GFS, resp. HDFS
- používateľ dodáva základné funkcie
 - map
 - reduce
 - combine
 - reader, writer
- framework sa postará o všetko ostatné

- map
 - $\text{map}(\text{item}) \rightarrow 0 \text{ a viac } \langle \text{Key}, \text{Value} \rangle \text{ párov}$
- reduce
 - $\text{reduce}(\text{key}, \text{list-of-values}) \rightarrow 0 \text{ a viac záznamov}$







Príklad - weblog

- CSV: UserID, URL, timestamp, additional-info
- Spočítaj všetky prístupy do domény (v URL)
- $\text{map}(\text{record}) \rightarrow \langle \text{domain}, \text{NULL} \rangle$
- $\text{reduce}(\text{domain}, \text{list of NULLs}) \rightarrow \langle \text{domain}, \text{count} \rangle$

Príklad - weblog

- CSV: UserID, URL, timestamp, additional-info
- Spočítaj všetky prístupy do domény (v URL)
- $\text{map}(\text{record}) \rightarrow \langle \text{domain}, \text{NULL} \rangle$
- $\text{combine}(\text{domain}, \text{list of NULLs}) \rightarrow \langle \text{domain}, \text{count} \rangle$
- $\text{reduce}(\text{domain}, \text{list of counts}) \rightarrow \langle \text{domain}, \text{sum} \rangle$

Ako to vyzerá v Jave?

MapReduce

- žiadny dátový model, dáta v súboroch
- poskytneme len zopár metód (map, reduce)
- systém vykoná ostatné
 - fault-tolerant (nejaký uzol môže zomrieť)
 - škálovateľne (môžeme pridávať uzly)

MapReduce

- žiadny dátový model, dáta v súboroch
- poskytneme len zopár metód (map, reduce)
- systém vykoná ostatné
 - fault-tolerant (nejaký uzol môže zomrieť)
 - škálovateľne (môžeme pridávať uzly)
- predsa len je to veľa programovania
- chýba nám deklaratívnosť



Hive, Pig a Cascalog



- Hive – schéma, SQL-like rozhranie
 - ak viete SQL, tak viete aj Hive
- Pig – špeciálny jazyk (Pig Latin a Pig Commands) pre manipuláciu s dátami
- Cascalog – Clojure/Java logic programming over Hadoop
- Všetky sa prekládajú do MapReduce jobov

Hive CLI

```
CREATE DATABASE proxy;
```

```
CREATE EXTERNAL TABLE proxy.access_logs  
(user_id STRING, url STRING, happened_at STRING)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','  
LOCATION '/barla/proxy/';
```

```
LOAD DATA LOCAL INPATH './proxy.log' OVERWRITE INTO TABLE  
proxy.access_logs;
```

```
INSERT OVERWRITE LOCAL DIRECTORY '/tmp/pv_gender_sum'  
SELECT...
```

Pig

- Interactive and batch mode
- Používa tzv. Pig Latin
 - Jazyk na zápis procedúr na spracovanie dát
- Typická procedúra
 - LOAD statement
 - transformácie
 - DUMP/STORE statement

LOAD

```
A = LOAD 'data' AS (a1:int,a2:int,a3:int);
```

```
DUMP A;
```

```
(1,2,3)
```

```
(4,2,1)
```

```
(8,3,4)
```

```
(4,3,3)
```

```
(7,2,5)
```

```
(8,4,3)
```

FILTER aka WHERE

```
X = FILTER A BY a3 == 3;
```

```
DUMP X;
```

```
(1,2,3)
```

```
(4,3,3)
```

```
(8,4,3)
```

SELECT

X = FOREACH A GENERATE a1, a2;

DUMP X;

(1,2)

(4,2)

(8,3)

(4,3)

(7,2)

(8,4)

GROUP

```
A = load 'student' AS  
(name:chararray,age:int,gpa:float);
```

```
B = GROUP A BY age;
```

Cascalog/JCascalog

- Clojure/Java
- Abstrakcia nad Hadoop
- Pripomína logické programovanie
 - Deklarujeme želané vzťahy medzi vstupmi a výstupmi cez predikáty

JCascalog ukážka

```
Api.execute(  
  new StdoutTap(),  
  new Subquery("?person")  
    .predicate(People.AGE, "?person", 25));
```

JCascalog ukážka

```
Api.execute(  
  new StdoutTap(),  
  new Subquery("?person", "?age")  
    .predicate(People.AGE, "?person", "?age")  
    .predicate(new LT(), "?age", 30));
```

JCascalog ukážka

```
Api.execute(  
    new StdoutTap(),  
    new Subquery("?person", "?double-age")  
        .predicate(People.AGE, "?person",  
            "?age")  
        .predicate(new Multiply(), "?age", 2)  
        .out("?double-age"));
```

JCascalog

- Môžem si implementovať vlastné operátory

```
public class Split extends CascalogFunction {  
    public void operate(FlowProcess flowProcess,  
        FunctionCall fnCall) {  
        String sentence = fnCall.getArguments().getString(0);  
        for(String word: sentence.split(" ")) {  
            fnCall.getOutputCollector().add(new Tuple(word));  
        }  
    }  
}
```

Použitie – príklad WordCount

```
Api.execute(  
    new StdoutTap(),  
    new Subquery("?word", "?count")  
        .predicate(Corpus.SENTENCE, "?sentence")  
        .predicate(new Split(), "?sentence").out("?  
word")  
        .predicate(new Count(), "?count"));
```

Spark

- “Map-Reduce v pamäti”
- Python, Scala, Java
- MLlib
 - Distribuované verzie ML algoritmov

WordCount v Sparku

```
text_file = sc.textFile("hdfs://...")  
counts = text_file.flatMap(lambda line:  
    line.split(" ")) \  
    .map(lambda word: (word, 1)) \  
    .reduceByKey(lambda a, b: a + b)  
counts.saveAsTextFile("hdfs://...")
```

Zhrnutie

- Paralelizovateľný výpočet
 - map & reduce
- Hadoop rieši všetku réžiu
 - plus High availability – detekuje a rieši zlyhania
- absencia deklaratívneho dopytovania
 - Nadstavby (Hive, Pig, Cascalog, ...)
- Priama súčasť alebo existujúci adaptér pre mnohé databázy
- Zaujímavé knižnice – Apache Mahout, ...