MapReduce & hadoop

Michal Barla

NoSQL and aggregates - order

{

“id” : 99,

“customer_id” : 1,

“orderItems”: [{“productId”: 27, “price”: 35.47, “productName”: “NoSQL Distilled” },{...}],

“shippingAddress” : {“city”: {…}},

“paymentInfo”: {…}

}

NoSQL and aggregates - order

{

“id” : 99,

“c

Great for showing the orders

“o

“p

Worse for overviews of orders for a particular product

“s

“paymentInfo”: {…}

}

We are in a distributed world

- Data are distributed on several nodes
  - Data of a single one user only on one node
  - Data of a single one product on every node

We are in a distributed world

- Data are distributed on several nodes
  - Data of a single one user only on one node
  - Data of a single one product on every node
- Idea: It is better to move program to data then the other way
  - The program can compute a partial aggregation in-place, where the data sit
  - Transmission of computed aggregation does not hurt that much

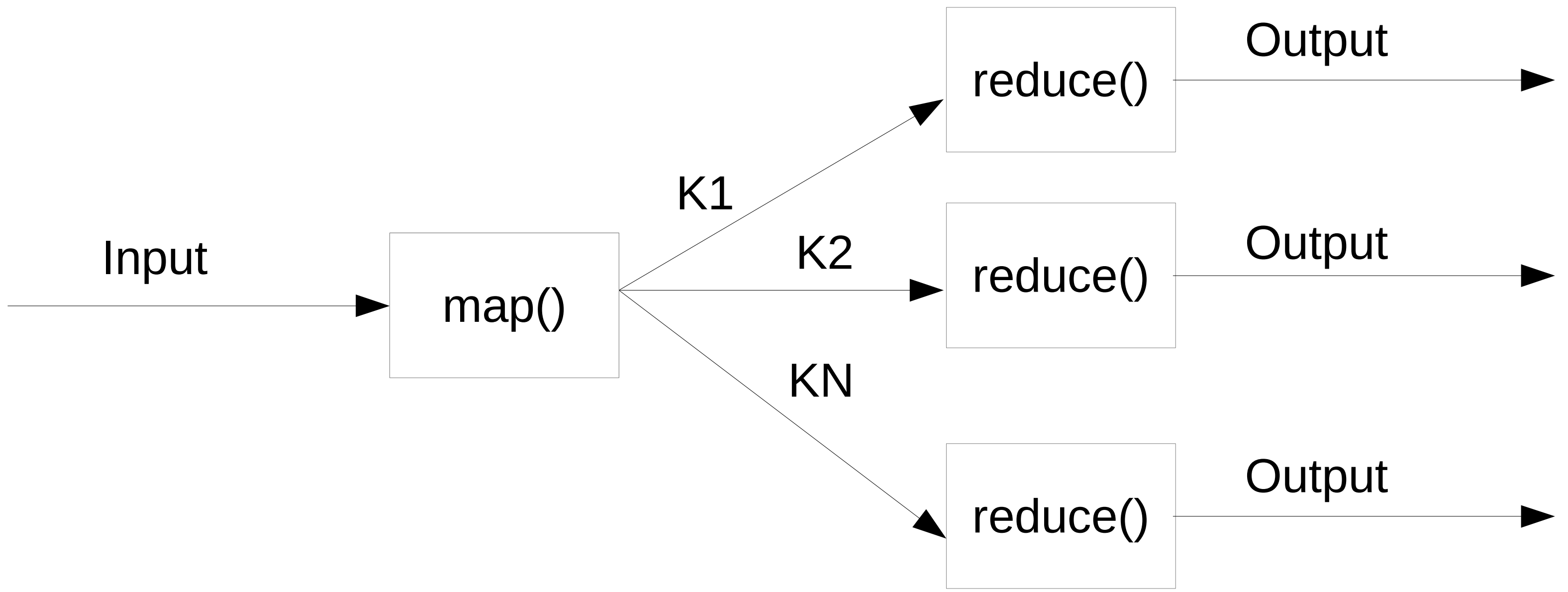
MapReduce computation model

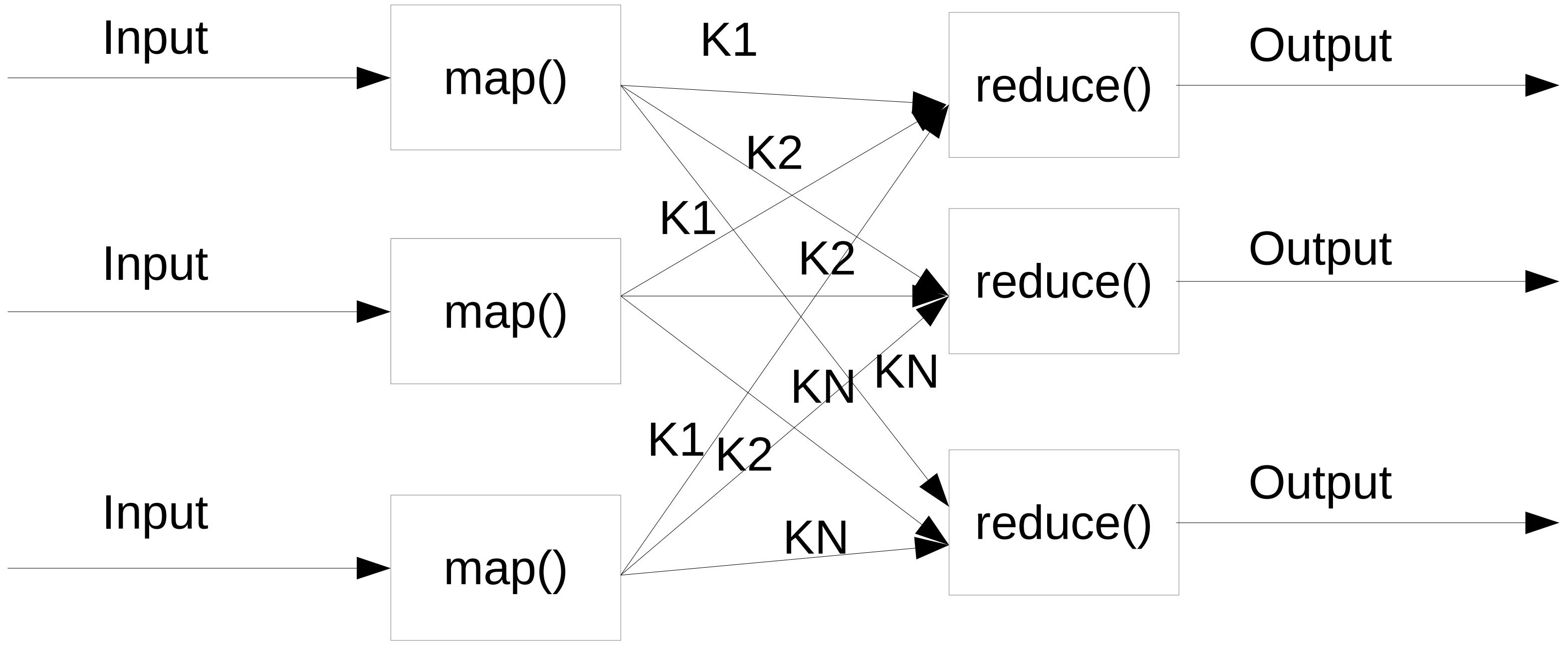
- Any computation can be divided into two stages
  - Map – Runs along the data, in individual nodes
  - Reduce – aggegates outputs of map stage
- Some NoSQL databases have MapReduce "built-in"
- Useful as a standalone solution as well

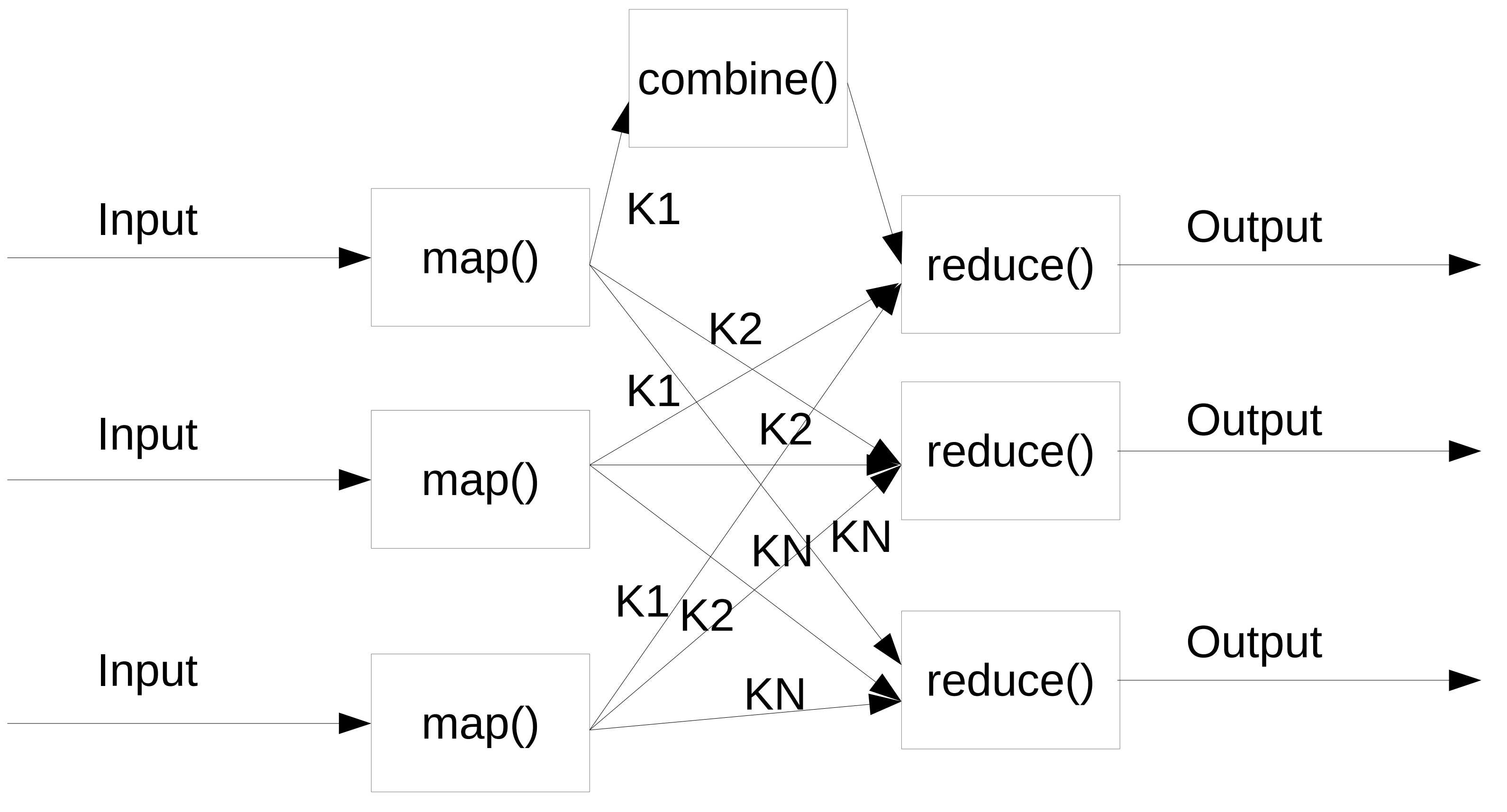
MapReduce framework

- Original from Google, open source
- No data model, everything in files
  - GFS, resp. HDFS
- User supplies basic functions
  - map
  - reduce
  - combine
  - reader, writer
- fframework handles everything else

- map
  - map(item) → 0 and more <Key, Value> pairs
- reduce
  - reduce(key, list-of-values) → 0 and more records


Input
map()
K1
K2
KN
reduce()
reduce()
reduce()
Output
Output
Output


Input
Input
Input
map()
map()
map()
K1
K2
K1
K2
KN
KN
K1
K2
KN
reduce()
reduce()
reduce()
Output
Output
Output


combine()
Input
map()
K1
reduce()
Output
K2
K1
Input
map()
K2
reduce()
Output
KN
KN
K1
K2
Input
map()
KN
reduce()
Output

weblog

- CSV: UserID, URL, timestamp, additional-info
- Count all access to a domain (part of URL)

- map(record) → <domain, NULL>
- reduce(domain, list of NULLs) → <domain, count>

weblog

- CSV: UserID, URL, timestamp, additional-info
- Count all access to a domain (part of URL)

- map(record) → <domain, NULL>
- combine(domain, list of NULLs) → <domain, count>
- reduce(domain, list of counts) → <domain, sum>

Hadoop

- Does all the management around tasks
- HDFS – redundant network filesystem
- Fault-tolerant (a node can die at any time)
- Scalable – we can add nodes at any time

- Map/Reduce tasks written in java :(

Hive, Pig & Cascalog

- Hive – adds schema, SQL-like interface
  - If you know SQL, you know Hive
- Pig – special language (Pig Latin a Pig Commands) for data manipulation
- Cascalog – Clojure/Java logic programming over Hadoop

- All get translated into MapReduce jobs

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
- Pig Latin
  - Procedural language for data processing
- Typical procedure
  - LOAD statement
  - transformations
  - 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;
```

Spark

- “Map-Reduce in memory”
- Python, Scala, Java

- MLlib
  - Distributed versions of ML algorithms

WordCount in Spark

```
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://...")
```

Summary of MapReduce

- Parallel computation
  - map & reduce
- Hadoop does all the management around
  - plus High availability – detects and solves failures
- Missing declarative programming?
  - Extensions (Hive, Pig, Cascalog, …)
- Built-in or supported by many databases
- Interesting libraries – Apache Mahout, ...