Parallel and distributed algorithms

MapReduce: Simplified Data Processing on Large Clusters

dr inż. Paweł Kaczmarek

Problem for MapReduce

- Process large amounts of raw data
  - crawled documents, web request logs, etc.
- To compute various kinds of derived data
  - inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host
- Most such computations are conceptually straightforward
  - the input data is usually large

Problem for MapReduce

- Computations have to be distributed across many machines
  - parallelize the computation
  - distribute the data
  - handle failures
- Originates from Google research
  - http://research.google.com/archive/mapreduce.html
  - http://en.wikipedia.org/wiki/MapReduce

General operation

- The computation
  - takes a set of input key/value pairs
  - produces a set of output key/value pairs (in different domain)
- Example
  - counting the number of occurrences of each word in a large collection of documents
    - input: key: index, value: document
    - output: key: word, value: count of occurrences

Occurrences of word in collection of documents

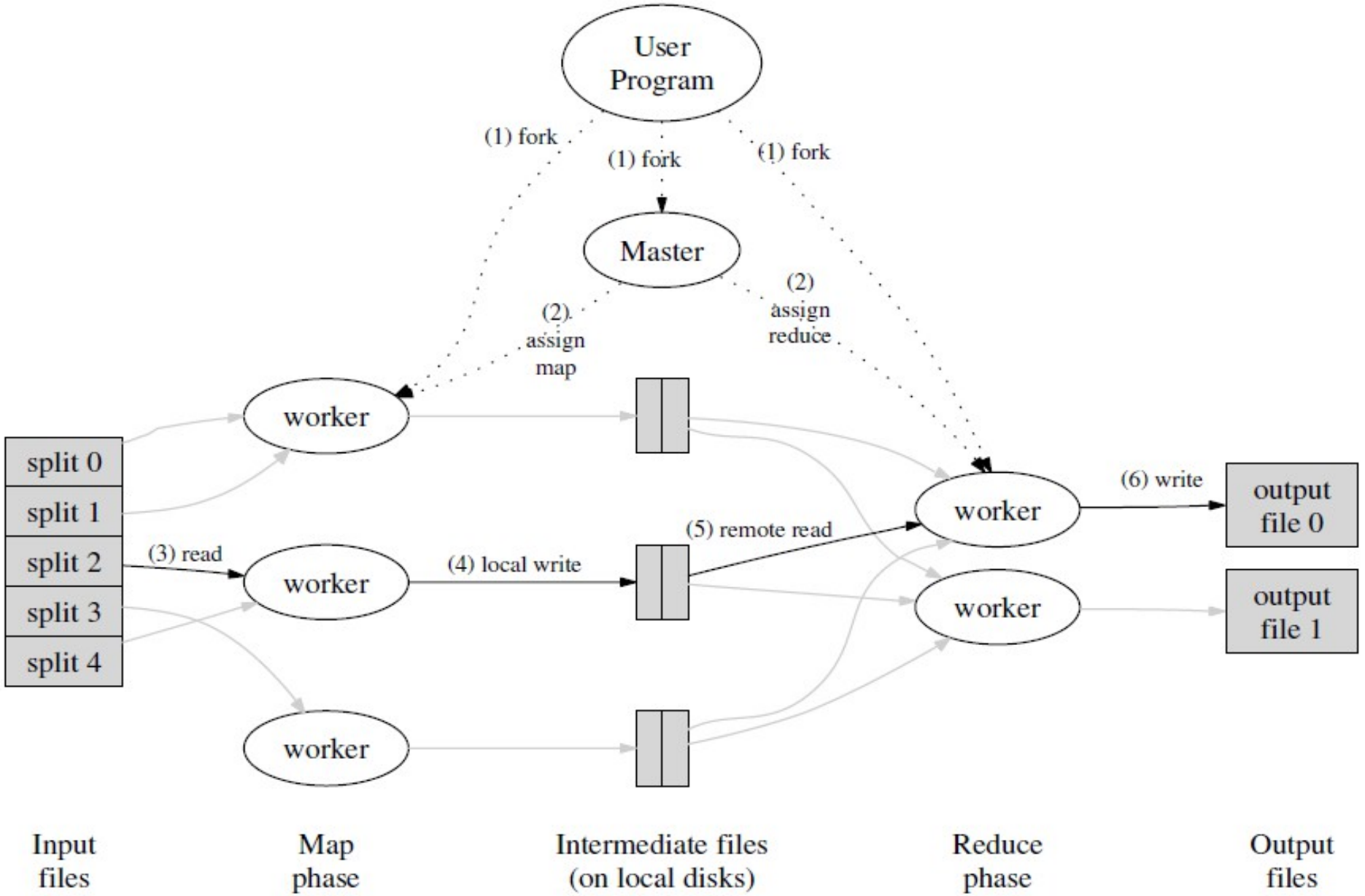

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");
reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

Map reduce operation

- Map function
  - processes a key/value pair to generate a set of intermediate key/value pairs
    - the example: emits each word plus an associated count of occurrences - just `1' in this simple example
  - takes one pair of data with a type in one data domain, and returns a list of pairs in a different domain
  - groups together all intermediate values associated
  - with the same intermediate key I and passes them to the Reduce function

Reduce operation

- Merges all intermediate values associated with the same intermediate key
  - example: sums together all counts emitted for a particular word
- accepts an intermediate key I and a set of values for that key
- Merges together these values to form a possibly smaller set of values
  - typically just zero or one output value is produced per Reduce invocation
- Intermediate values are supplied to the user's reduce function via an iterator


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Types

- map
  - (k1,v1) -> list(k2,v2)
- reduce
  - (k2,list(v2)) -> list(v2)
- The input keys and values are drawn from a different domain than the output keys and values.

Parallel / distributed issues

- MapReduce system (framework) usually splits the input data-set into independent chunks
  - processed by map / reduce tasks in a completely parallel manner
  - each mapping operation is independent of the others
  - sorts the outputs of the maps, which are then input to the reduce tasks
- Processing can occur on data stored either in a filesystem (unstructured) or in a database (structured)
- The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks

Parallel / distributed issues

- Map step: Each worker node applies the "map()" function to the local data, and writes the output to a temporary storage. A master node orchestrates that for redundant copies of input data, only one is processed.
- Shuffle step: Worker nodes redistribute data based on the output keys (produced by the "map()" function), such that all data belonging to one key is located on the same worker node.
- Reduce step: Worker nodes now process each group of output data, per key, in parallel.
- Similarly, a set of 'reducers' can perform the reduction phase, provided that all outputs of the map operation that share the same key are presented to the same reducer at the same time, or that the reduction function is associative

Exemplary tool

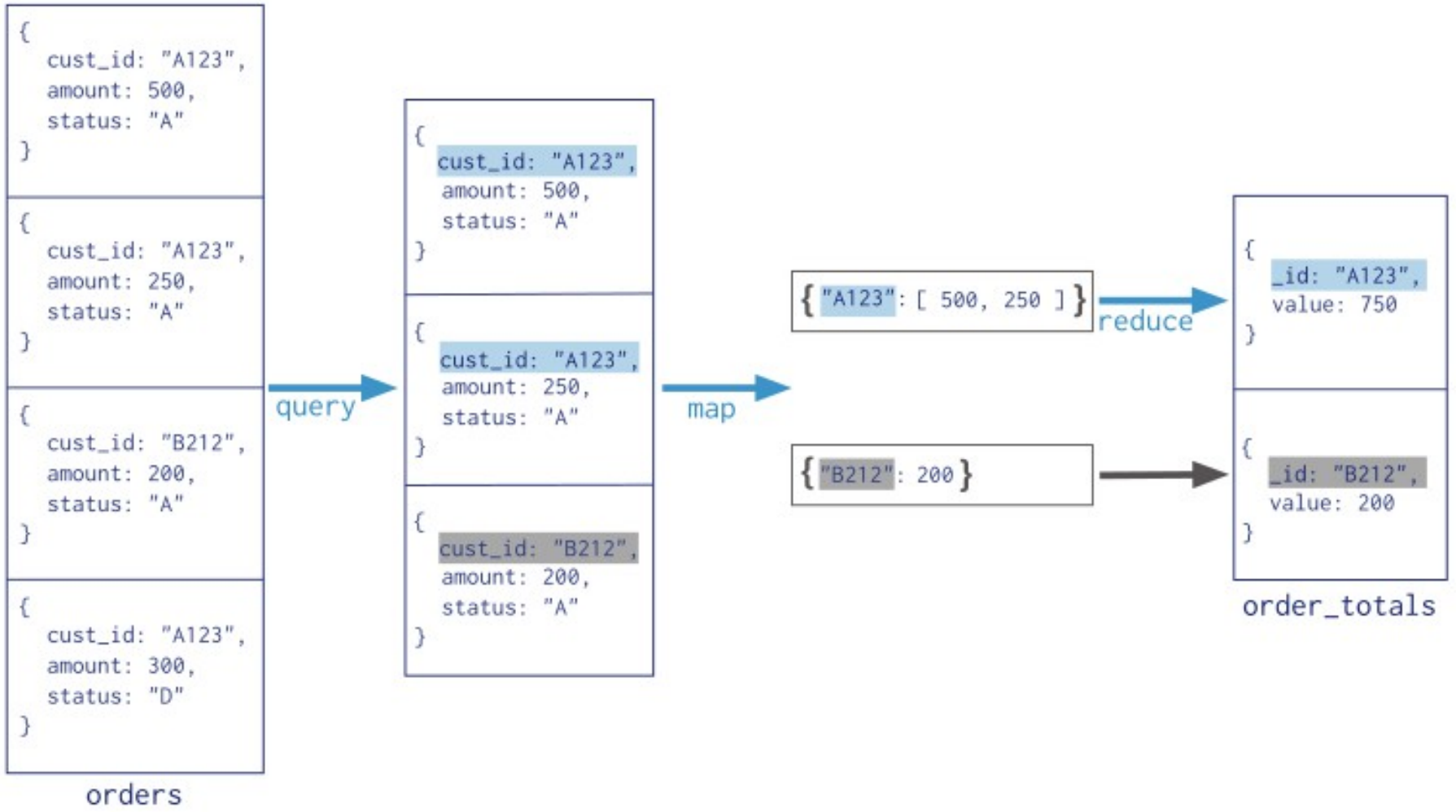
- MongoDB provides the mapReduce database command
- Demo
  - run mongo
    - mongod, mongo
  - create db
    - use algorithmdb
    - db, show dbs
    - show collections

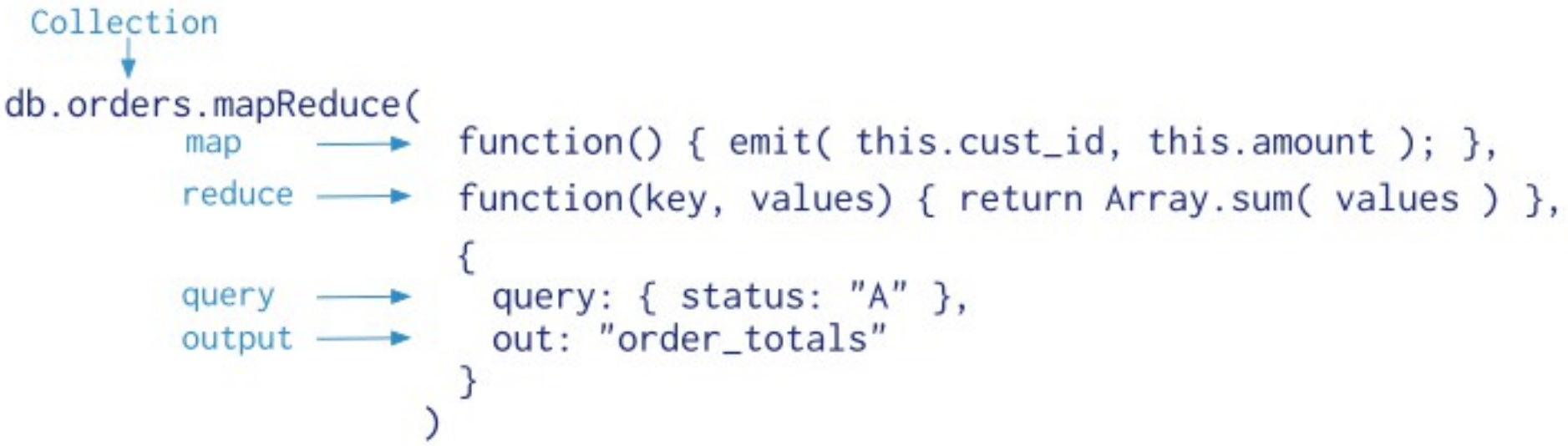
Mongo map - reduce

- Insert demo data ("insert" file)

```
var customers =
[
{
    cust_id: "A1",
    status: 'A',
     amount: 500
},
... other customers
    ]
```

- ... functions


{
cust_id: "A123",
amount: 500,
status: "A"
}
{
cust_id: "A123",
amount: 250,
status: "A"
}
{
cust_id: "B212",
amount: 200,
status: "A"
}
{
cust_id: "A123",
amount: 300,
status: "D"
}
orders
query
{
cust_id: "A123",
amount: 500,
status: "A"
}
{
cust_id: "A123",
amount: 250,
status: "A"
}
{
cust_id: "B212",
amount: 200,
status: "A"
}
map
{"A123": [500, 250]}
{"B212": 200}
reduce
{
_id: "A123",
value: 750
}
{
_id: "B212",
value: 200
}
order_totals


Collection
db.orders.mapReduce(
map
function() { emit(this.cust_id, this.amount); },
reduce
function(key, values) { return Array.sum(values) },
{
query
query: { status: "A" },
output
out: "order_totals"
}
)

Mongo data

- load file - as JavaScript file
  - load ("fileName")
  - and file location
- db.algorithmdb.insert(customers)
  - db.algorithmdb.find()

Mongo map-reduce functions


```
var mapFunction1 = function() {
                    emit(this.cust_id, this.amount);
                };


var reduceFunction1 = function(keyCustId, valuesAmounts) {
                      return Array.sum(valuesAmounts);
                  };

db.algorithmdb.mapReduce(
                    mapFunction1,
                    reduceFunction1,
                    { out: "map_reduce_example" }
                )
```

- result in map_reduce_example collection
- db.map_reduce_example.find()

Bibliography

- http://research.google.com/archive/mapreduce.html
- http://en.wikipedia.org/wiki/MapReduce
- http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
- http://docs.mongodb.org/manual/core/map-reduce/