MongoDB and Map-Reduce

Abdu Alawini

University of Illinois at Urbana-Champaign

CS411: Database Systems

June 28, 2020

Some slides were adopted from D. Maier. S. Davidson with permission

Leaning Objectives

After this lecture, you should be able to:

- Write MongoDB cursor queries.
- Query documents by reference.
- Write aggregation queries
- Write map-reduce programs in MongoDB

Cursor methods

- .count, .pretty, .sort etc are all examples of cursor methods
- **.toArray** returns an array that contains all documents from a cursor

```
nsf= db.awards.find({"by": "National Science Foundation").toArray()
if (nsf.length >0) {printjson (nsf[0])}
```

- **.forEach** applies a JavaScript function to each document from the cursor (similar to .map)

```
db.awards.find(). forEach
 ( function(myDoc) { printjson ("Award: " + myDoc.name); });
```

Joins in MongoDB

1. Embedded Relationships
   - "Do joins while write, not on reads."

2. Referenced Relationships
   - Use semi-joins to get an array of keys from the first collection on which to search the second collection for matches using cursor methods

Relationships: Embedded

```
{ _id: 1,
  name: { first: "John", last: "Backus" },
  birthyear: 1924,
  contribs: [ "Fortran", "ALGOL",
              "Backus Naur Form", "FP" ],
  awards: [ {title: "National Medal of Science" ,
             by: "National Science Foundation",
             year: 1975 },
            {title: "Turing Award",
             by: "ACM" ,
             year: 1977} ,
            {title: "Career Award",
             by: "NIH" ,
             year: 1980} ,
             {title: "Career Success Award",
             by: "NASA" ,
             year: 1982} ]  }
```

Relationships: Referenced

People:

```
{ _id: 1,
  name: { first: "John", last: "Backus" },
  birthyear: 1924,
  contribs: [ "Fortran", "ALGOL",
              "Backus Naur Form", "FP" ],
  awards: [ { award_id: "NMS001", year: 1975 },
            { award_id: "TA99",  year: 1977} ]  }
```

Awards:

```
{_id: "NMS001",
 title:  "National Medal of Science" ,
 by: "National Science Foundation"},
{_id: "TA99",
 title:  "Turing Award",
 by: "ACM" }
```

"SemiJoins"

- Suppose you want to print people who have won Turing Awards using referenced relationship
  - Problem: object id of Turing Award is in collection "awards", collection "people" references it.

```
turing= db.awards.findOne({title: "Turing Award"})
db.people.find({"awards.award_id": turing._id})
```

  - But this only works for one award with title "Turing Award", suppose there were more.

Awards:

```
{_id: "NMS001",
 title: "National Medal of Science" ,
 by: "National Science Foundation"},
{_id: "TA99",
 title: "Turing Award",
 by: "ACM" },
{_id: "CA85",
 title: "Career Award",
 by: "NIH" ,
 year: 1980} ,
 {_id: "CSA19
  title: "Career Success Award",
  by: "NASA" ,
  year: 1982}
```

Iterating using cursors

- Now suppose we want to find all people who won awards with title that starts with 'ca'

```
let ca_award= db.awards.find ({title: /^ca/i},{_id:1})
let award_Ids = new Array();
while (ca_award.hasNext())  {
        let tmp = award_Ids .push(ca_award.next()._id)
}
db.people.find({"awards.award_id":{$in: award_Ids })
```

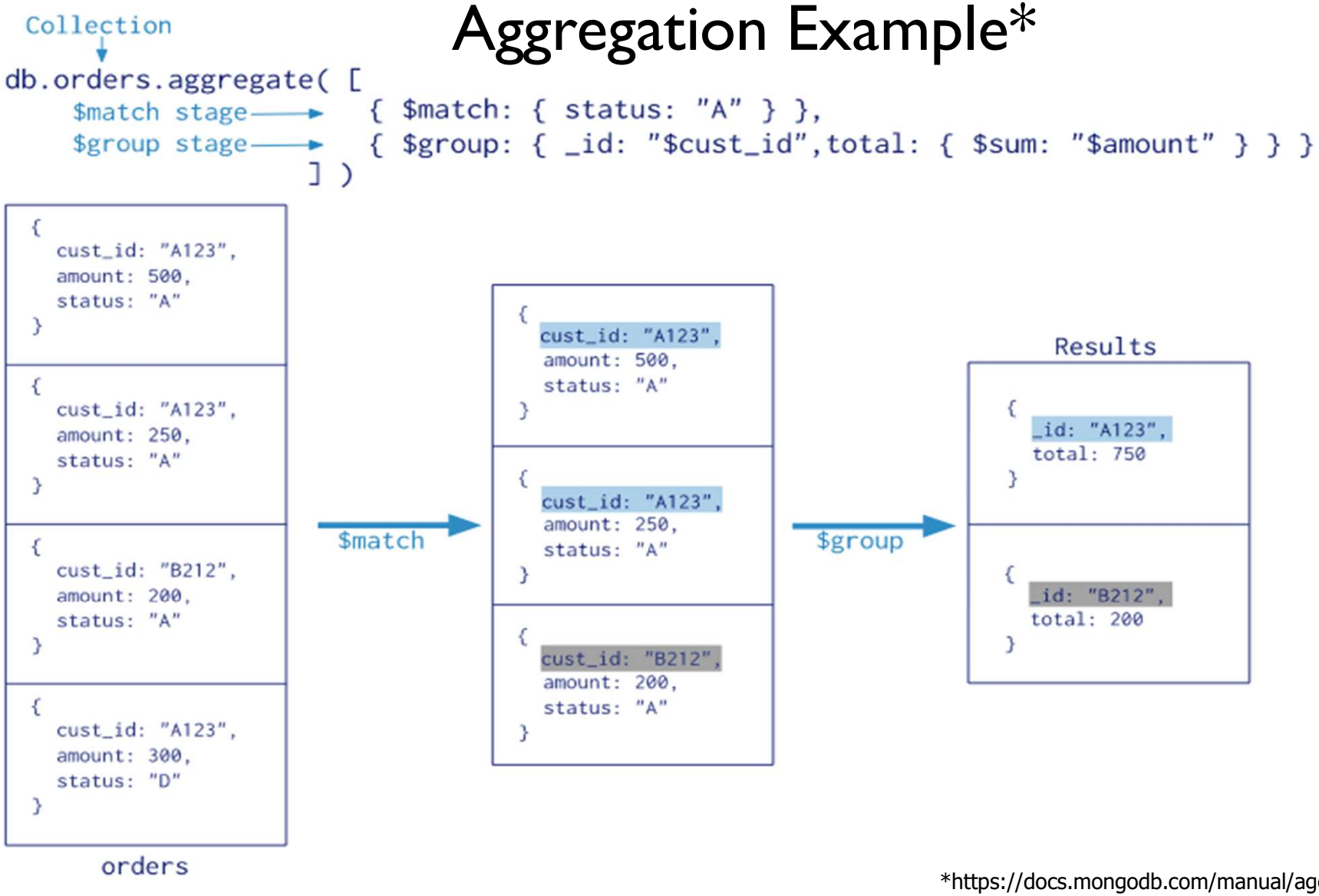
Aggregation

- A framework to provide "group-by" and aggregate functionality without the overhead of map-reduce.

- Conceptually, documents from a collection pass through an aggregation pipeline, which transforms the objects as they pass through (similar to UNIX pipe "|")

- Operators include: $project, $match, $group, $sort, $skip, $limit, $unwind

Aggregation Example*



*https://docs.mongodb.com/manual/aggregation/

Aggregation: $group

- Every group expression must specify an _id field.
- Suppose we wanted to find how many people were born each year

```
> db.people.aggregate( { $group :
     { _id : "$birthyear", birthsPerYear : { $sum : 1}}})
```

```
{ "_id" : 1924, "birthsPerYear" : 1 }
```

- Contrast with aggregate operation over entire result

```
> db.people.count( )
> db.people.find({“name.first”: “John”}).count( )
> db.people.count({“name.first”: “John”})
```

Aggregation: $unwind

- Deconstructs an array field to output a document for each element.

```
Posts:   {
         _id : ObjectId("4c4ba5c0672c685e5e8aabf3"),
         author : "Kevin",
         date : new Date("February 2, 2012"),
         text : "About MongoDB...",
         birthyear:  1980,
         tags : [ "tech", "databases" ]
         }
```

```
>db.posts.aggregate( { $project : { author : 1, tags : 1 }}, { $unwind : "$tags" } )
```

Result of unwind

```
>db.posts.aggregate( { $project : { author : 1, tags : 1 }}, { $unwind : "$tags" } )
```

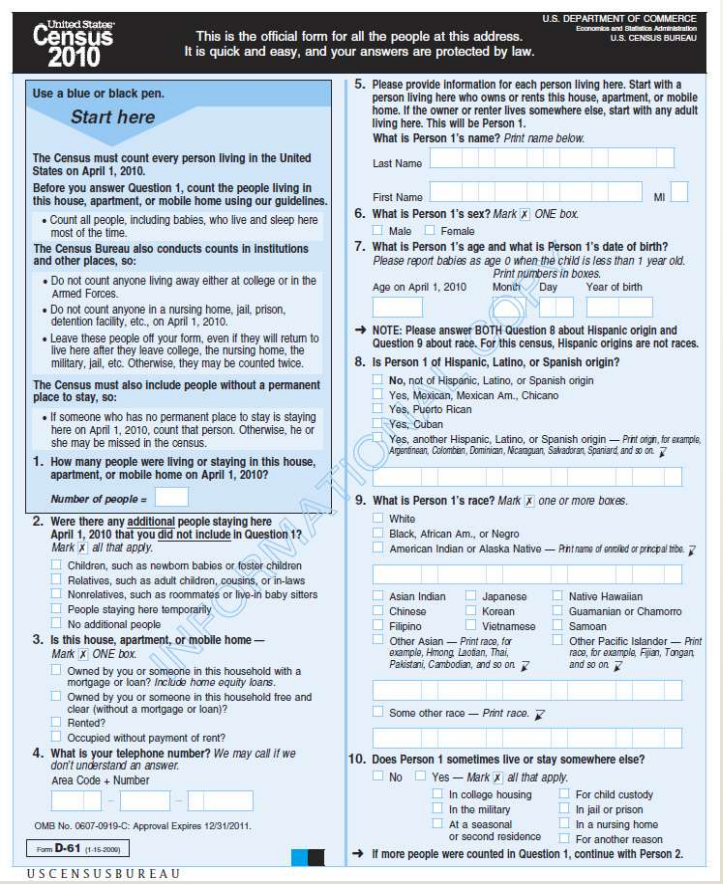
```
{ "_id" : ObjectId("4c4ba5c0672c685e5e8aabf3"),
        "author" : "Kevin",
        "tags" : ”tech" },
{ "_id" : ObjectId("4c4ba5c0672c685e5e8aabf3"),
        "author" : "Kevin",
        "tags" : ”databases" }
```

Map-Reduce: Motivation

- SQL is a great way of doing batch (bulk) operations to collections (tables)
- But it's somewhat limited in the kinds of operations it supports (by default) –
  SELECT … FROM … WHERE … GROUP BY … HAVING …
- What if we want to do *arbitrary* selection predicates (in Java etc.), and arbitrary aggregations (in Java etc.)
  - User defined functions, or alternative frameworks
- What if we wanted to scale this up to run on a 10000 node compute cluster??

Map-Reduce Analogy: National census

- Suppose we have 10,000 employees, whose job is to collect census forms and to determine how many people live in each city
- How would you organize this task?



United States Census 2010

This is the official form for all the people at this address. It is quick and easy, and your answers are protected by law.

U.S. DEPARTMENT OF COMMERCE
Economics and Statistics Administration
U.S. CENSUS BUREAU

Use a blue or black pen.

Start here

The Census must count every person living in the United States on April 1, 2010.

Before you answer Question 1, count the people living in this house, apartment, or mobile home using our guidelines.

- Count all people, including babies, who live and sleep here most of the time.

The Census Bureau also conducts counts in institutions and other places, so:

- Do not count anyone living away either at college or in the Armed Forces.
- Do not count anyone in a nursing home, jail, prison, detention facility, etc., on April 1, 2010.
- Leave these people off your form, even if they will return to live here after they leave college, the nursing home, the military, jail, etc. Otherwise, they may be counted twice.

The Census must also include people without a permanent place to stay, so:

- If someone who has no permanent place to stay is staying here on April 1, 2010, count that person. Otherwise, he or she may be missed in the census.

1. How many people were living or staying in this house, apartment, or mobile home on April 1, 2010?
   Number of people =
2. Were there any additional people staying here April 1, 2010 that you did not include in Question 1? Mark (X) all that apply.
   - Children, such as newborn babies or foster children
   - Relatives, such as adult children, cousins, or in-laws
   - Nonrelatives, such as roommates or live-in baby sitters
   - People staying here temporarily
   - No additional people
3. Is this house, apartment, or mobile home — Mark (X) ONE box.
   - Owned by you or someone in this household with a mortgage or loan? Include home equity loans.
   - Owned by you or someone in this household free and clear (without a mortgage or loan)?
   - Rented?
   - Occupied without payment of rent?
4. What is your telephone number? We may call if we don't understand an answer.
   Area Code + Number

OMB No. 0607-0919-C: Approval Expires 12/31/2011.

Form D-61 (1-15-2009)

5. Please provide information for each person living here. Start with a person living here who owns or rents this house, apartment, or mobile home. If the owner or renter lives somewhere else, start with any adult living here. This will be Person 1.
   What is Person 1's name? Print name below.
   Last Name
   First Name MI
6. What is Person 1's sex? Mark (X) ONE box.
   - Male
   - Female
7. What is Person 1's age and what is Person 1's date of birth? Please report babies as age 0 when the child is less than 1 year old. Print numbers in boxes.
   Age on April 1, 2010 Month Day Year of birth

→ NOTE: Please answer BOTH Question 8 about Hispanic origin and Question 9 about race. For this census, Hispanic origins are not races.

8. Is Person 1 of Hispanic, Latino, or Spanish origin?
   - No, not of Hispanic, Latino, or Spanish origin
   - Yes, Mexican, Mexican Am., Chicano
   - Yes, Puerto Rican
   - Yes, Cuban
   - Yes, another Hispanic, Latino, or Spanish origin — Print origin, for example, Argentinean, Colombian, Dominican, Nicaraguan, Salvadoran, Spaniard, and so on.
9. What is Person 1's race? Mark (X) one or more boxes.
   - White
   - Black, African Am., or Negro
   - American Indian or Alaska Native — Print name of enrolled or principal tribe.
   - Asian Indian
   - Japanese
   - Native Hawaiian
   - Chinese
   - Korean
   - Guamanian or Chamorro
   - Filipino
   - Vietnamese
   - Samoan
   - Other Asian — Print race, for example, Hmong, Laotian, Thai, Pakistani, Cambodian, and so on.
   - Other Pacific Islander — Print race, for example, Fijian, Tongan, and so on.
   - Some other race — Print race.
10. Does Person 1 sometimes live or stay somewhere else?
    - No
    - Yes — Mark (X) all that apply.
      - In college housing
      - In the military
      - At a seasonal or second residence
      - For child custody
      - In jail or prison
      - In a nursing home
      - For another reason

→ If more people were counted in Question 1, continue with Person 2.

U.S. CENSUS BUREAU

http://www.census.gov/2010census/pdf/2010_Questionnaire_Info.pdf

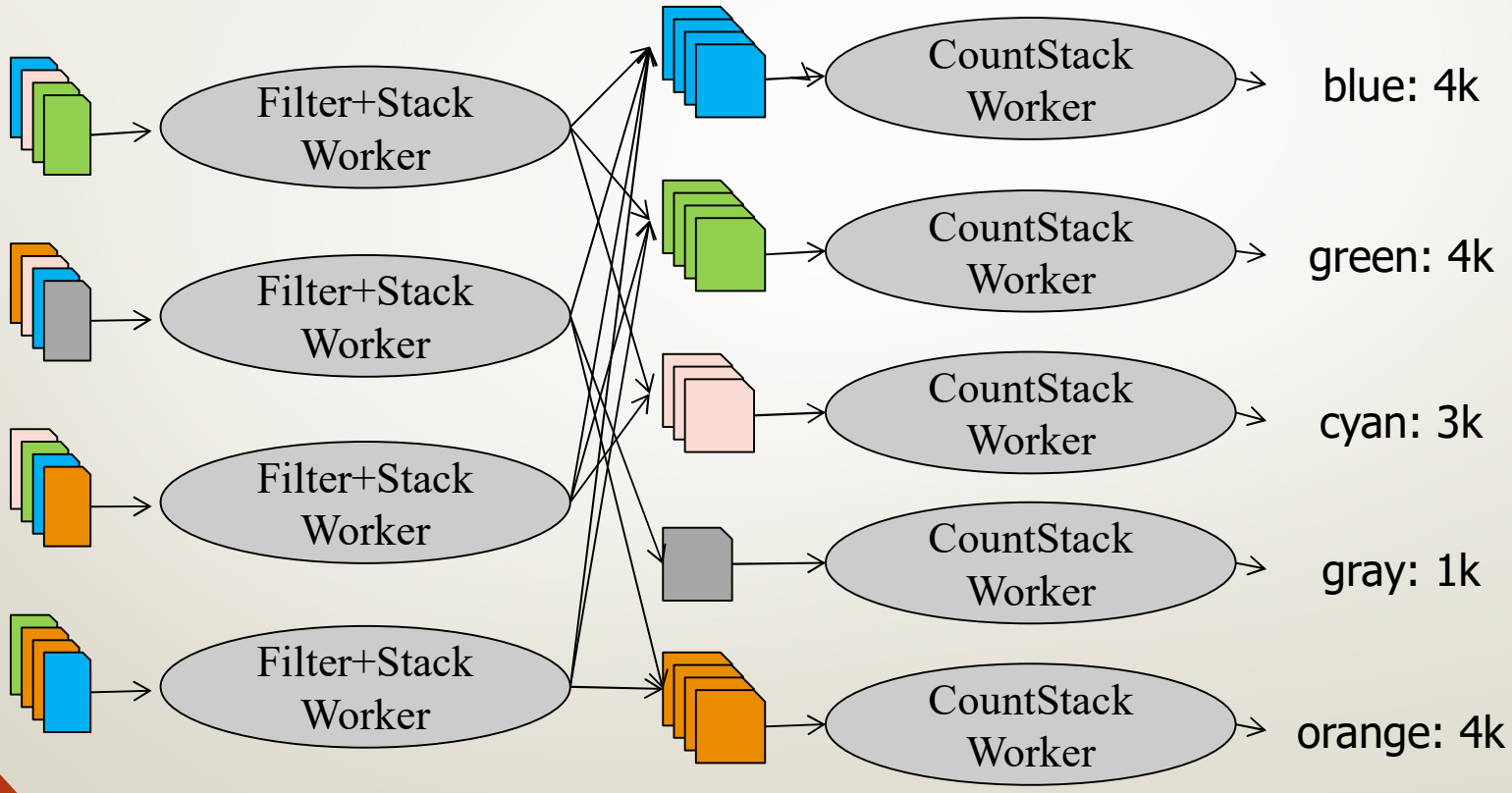
Making things more complicated

- Suppose workers take vacations, get sick, work at different rates
- Suppose some forms are incorrectly filled out and require corrections or need to be thrown away
- What if the supervisor gets sick?
- How big should the stacks be?
- How do we monitor progress?
- …

I don't want to deal with all this!!!

- Wouldn't it be nice if there were some system that took care of all these details for you?
- Ideally, you'd just tell the system what needs to be done
- That's the MapReduce framework.

Abstracting into a digital data flow



Abstracting once more

- There are two kinds of workers:
  - Those who take input data items and produce output items for the "stacks"
  - Those who take the stacks and aggregate the results to produce outputs on a per-stack basis

map does SELECT + PROJECT

- We'll call these:
  - map: takes (item_key, value), produces one or more (stack_key, value') pairs
  - reduce: takes (stack_key, {set of value'}), produces one or more output results – typically (stack_key, agg_value)

We will refer to this key
as the reduce key; it is like the GROUP BY key

Why MapReduce?

- Scenario:
  - You have a huge amount of data, e.g., all the Google searches of the last three years
  - You would like to perform a computation on the data, e.g., find out which search terms were the most popular
- Analogy to the census example:
  - The computation isn't necessarily difficult, but parallelizing and distributing it, as well as handling faults, is challenging
- Idea: A programming abstraction / template!
  - Write a simple program to express the (simple) computation, and let the language runtime do all the hard work

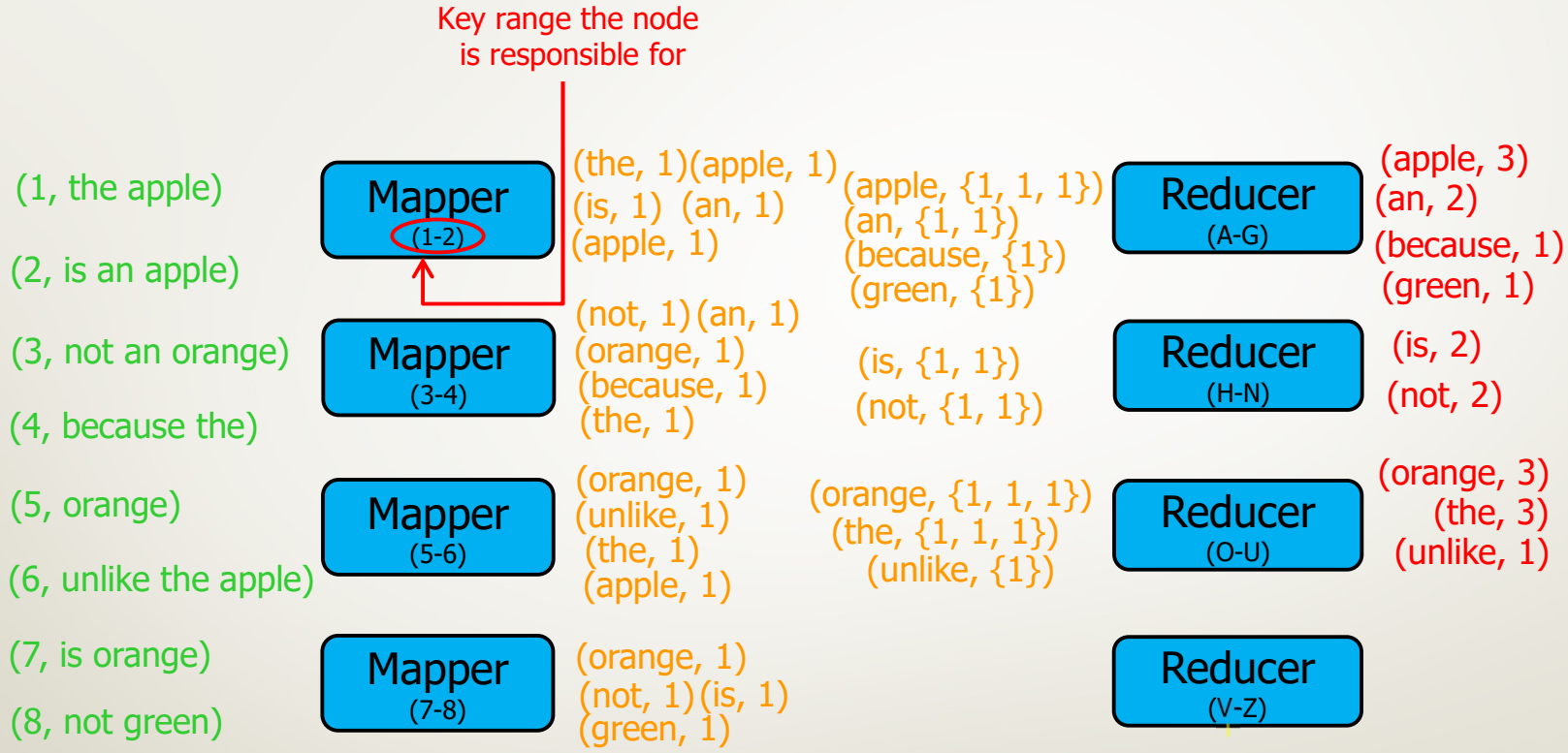
Simple example: Word count

```
map(String key, String value) {
  // key: document name, line no
  // value: contents of line
  for each word w in value:
    emit(w, "1")
}
```

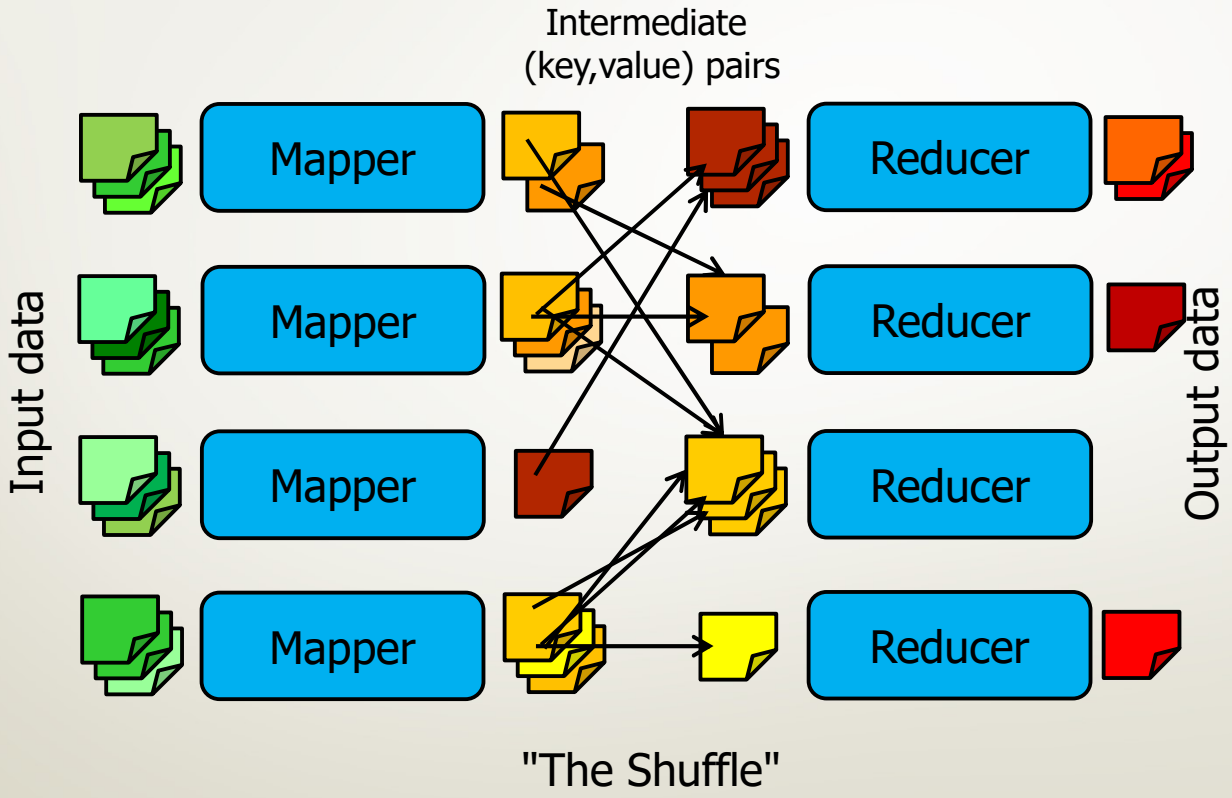
```
reduce(String key, Iterator values)
{
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
      result += ParseInt(v);
    emit(key, result)
}
```

- **Goal:** Given a set of documents, count how often each word occurs
  - Input: Key-value pairs (document:lineNumber, text)
  - Output: Key-value pairs (word, #occurrences)
  - What should be the intermediate key-value pairs?

Simple example: Word count



MapReduce dataflow



More examples

- Count URL access frequency
  - Map: output each URL as key, with count 1
  - Reduce: sum the counts
- Reverse web-link graph
  - Map: output (target, source) pairs when link to target found in source
  - Reduce: concatenates values and emits (target, list(source))
- Inverted index
  - Map: Emits (word, documentID)
  - Reduce: Combines these into (word, list(documentID))

Designing MapReduce algorithms

- Key decision: What should be done by `map`, and what by `reduce`?
  - `map` can do something to each individual key-value pair, but it can't look at other key-value pairs
    - Example: Filtering out key-value pairs we don't need
  - `map` can emit more than one intermediate key-value pair for each incoming key-value pair
    - Example: Incoming data is text, `map` produces (word,1) for each word
  - `reduce` can aggregate data; it can look at multiple values, as long as `map` has mapped them to the same (intermediate) key
    - Example: Count the number of words, add up the total cost, …
- Need to get the intermediate format right!
  - If `reduce` needs to look at several values together, `map` must emit them using the same key!

Common mistakes to avoid

- Mapper and reducer should be stateless
  - Don't use static variables - after `map` + `reduce` return, they should remember nothing about the processed data!
  - Reason: No guarantees about which key-value pairs will be processed by which workers!

```
HashMap h = new HashMap();
map(key, value) {
  if (h.contains(key)) {
    h.add(key,value);
    emit(key, "X");
  }
}
```

Wrong!

- Don't try to do your own I/O!
  - Don't try to write to files in the file system
  - The MapReduce framework does the I/O for you (usually):
    - All the incoming data will be fed as arguments to map and reduce
    - Any data your functions produce should be output via emit

```
map(key, value) {
  File foo =
    new File("xyz.txt");
  while (true) {
    s = foo.readLine();
    ...
  }
}
```

Wrong!

More common mistakes to avoid

```
map(key, value) {
  emit("FOO", key + " " + value);
}
```

Wrong!

```
reduce(key, value[]) {
  /* do some computation on
  all the values */
}
```

- Mapper must not map too much data to the same key
  - In particular, don't map *everything* to the same key!! Otherwise the reduce worker will be overwhelmed!
  - It's okay if some reduce workers have more work than others
    - Example: In WordCount, the reduce worker that works on the key 'and' has a lot more work than the reduce worker that works on 'syzygy'.

MapReduce and MongoDB

- Suppose we have a collection of orders that look like the following:

```
{
        _id: ”0123”,
        cust_id: "abc123",
        ord_dat: new Date("Oct 04, 2013"),
        status: 'A',
        price: 17,
        items: [ {sku: "mmm", qty: 5, price: 2.5},
                 {sku:"nnn", qty: 3, price: 1.5} ]
}
```

Example 1: MapReduce and MongoDB

- Return the total cost per customer.

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

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

db.orders.mapReduce(mapFunction1, reduceFunction1,
                    {out:  {inline:1} })
```

Output on the following input…

input

order1= {_id: "0123", cust_id: "abc123",…, price: 17, …}
order2= {_id: "0124", cust_id: "abc124”,…, price: 20, …}
order3= {_id: "0125”, cust_id: "abc123", …, price: 13,…}

output

{"results" : [{"_id" : "abc123”,"value" : 30},
{"_id" : "abc124”,"value" : 20}],
"timeMillis" : 23,
"counts" : {"input" : 3,"emit" : 3,"reduce" : 1,"output" : 2},
"ok" : 1}

Example 2: MapReduce and MongoDB

- Return the total and average number of each item (SKU) in orders with status "A".

Example 2: MapReduce and MongoDB

- Return the total and average number of each item (SKU) in orders with status "A".

```
var mapFunction2 = function () {
        for (var idx=0; idx<this.items.length; idx++) {
          var key= this.items[idx].sku;
          var value= { count: 1, qty: this.items[idx].qty };
          emit(key,value); } }

var reduceFunction2= function(keySKU, countObjVals) {
       reducedVal= {count: 0, qty: 0};
       for (var idx=0; idx< countObjVals.length; idx++)  {
            reducedVal.count += countObjVals[idx].count;
            reducedVal.qty += countObjVals[idx].qty;  }
       return reducedVal;   }
```

Example 2, cont

- But we need more to calculate the average, hence have a "finalize" function.

```
var finalizeFunction2= function (key, reducedVal) {
      reducedVal.avg= reducedVal.qty/reducedVal.count;
      return reducedVal;
 };
```

- Putting it all together:

```
db.orders.mapReduce(mapFunction2, reduceFunction2,
       {out:{inline:1},
        query: {status: "A"},
        finalize: finalizeFunction2 } )
```

Output (Ex.2)


```
{"results" : [{"_id ": "mmm",
               "value" : {"count" : 3, "qty" : 14,
                          "avg" : 4.666666666666667} },
              {"_id" : "nnn",
               "value" : {"count" : 3, "qty" : 13,
               "avg" : 4.333333333333333} } ],
             "timeMillis" : 13,
             "counts" : {"input" : 3,
                         "emit" : 6,
                         "reduce" : 2,
                         "output" : 2},
             "ok" : 1 }
```

Conclusions

- Big data is two problems: analysis and storage
  - **Analysis**: relies on modeling, ML, statistics
  - **Storage**: relies on database technique to store and manipulate huge amounts of data to facilitate fast queries
- Due to "variety" and "volume", relational solutions frequently fall short
  - Need for less structure and nested information
  - Need for parallelism
- Map-reduce is part of many of the key-value store NoSQL solutions.