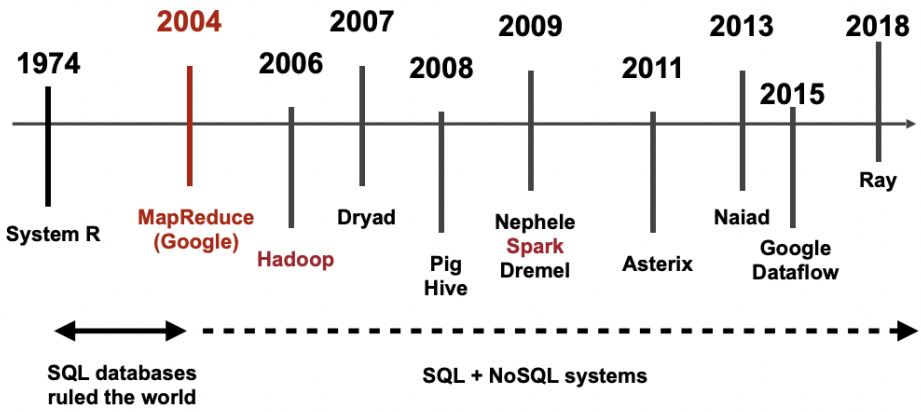
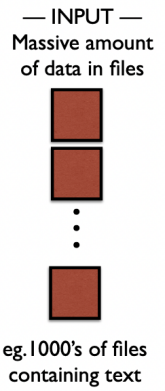
CAS CS 350
InClass Note 2

MapReduce

1. MapReduce → programming model (class of framing work)
2. Timeline of MapReduce



   a.
   b. SQL database → main function of controlling relational data prior to MapReduce
   c. MapReduce (c++ language) influenced many other systems
      - Hadoop (system that is open source of MapReduce in Java)
      - Spark (faster version of Hadoop that also provides more expressive language and programming model → more expressive data flow than MapReduce)
3. Simple MapReduce: Case study and Lab 1
   a. MapReduce merits → provides an intuitive programming model, intuitive abstraction which increases productivity (even a junior developer can solve huge data processing without worrying about paralyzation, concurrency, and etc. but just think about the program itself)
   b. MapReduce merits → fault tolerant (fault tolerant is important since computation job can take long time depending on the workload)
   c. When there exists a failure, the system will handle things for you as if the error does not exist
   d. Google
   e. Reusable infrastructure for doing big distributed computations that alleviates the burden of distributions form the app programer
   f. Provides an abstraction
   g. Programmer focuses on the core of the app, infrastructure does the rest
4. Computational Model
   a. Distributed file system (in this case, it is the Google file system)

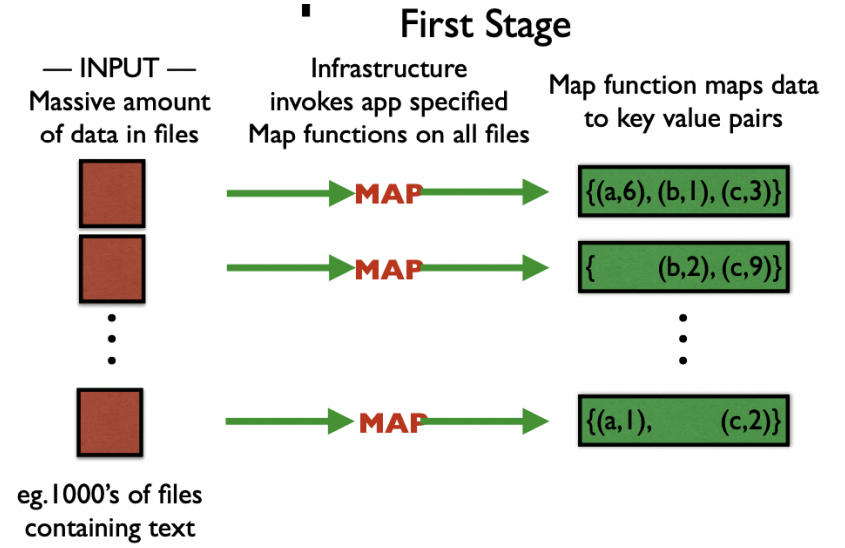


b.

c. MapReduce expects a text file and read the file line by line (in modern system we can use any files (not just text file))

d. Computation usually requires multiple map and reduce phases

e. First Stage

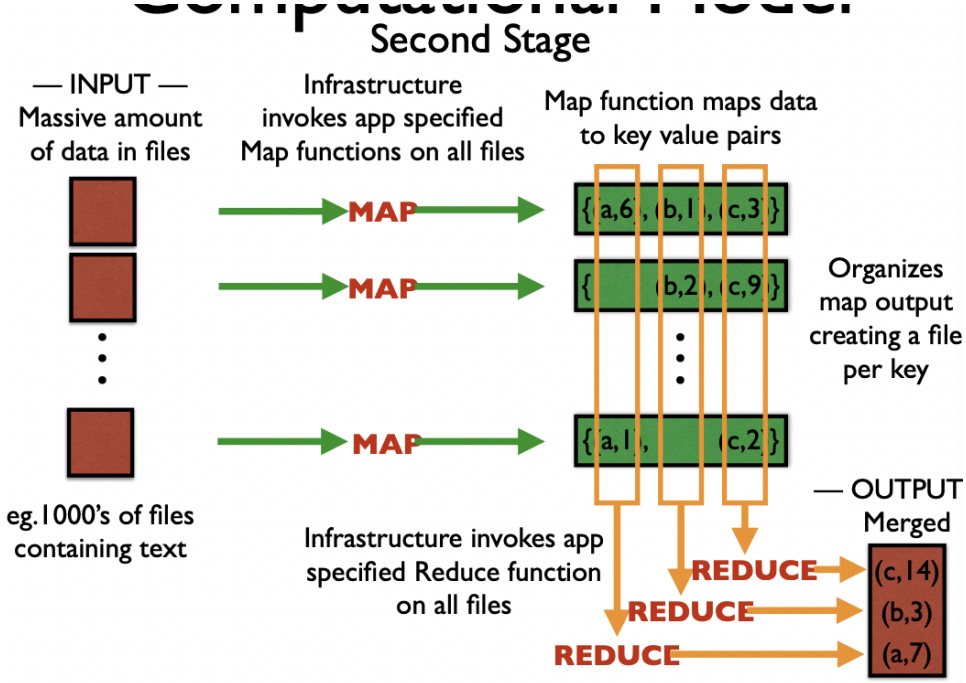


f.

g. The map function is applied to each of the input file and produces an output

h. Result of the application is a collection of pairs (key value pairs which are usually strings) ex) word and number of words, and etc.

i. Second Stage



j.

k. Partitioned to intermediate files based on keys

l. User specifies the number of map outputs

m. K intermediate files are created (on the local disk of the machine that executes the mapper function), which equals the number of reducers that will be used on the second phase

n. How to choose the number of mappers and reducers (chosen by the user) → ex) if there are ten machines, how many mappers and reducers?

- Utilize all machines

o. Is it possible that we can use the same thread for reduce and mappers? Yes

p. The Reduce function (defined by the programmer) → take the bunch of file and output a file that is merged (output file contains key value pairs)

q. Mapper function are applied to all input files line by line

r. Reducer function (first have to collect all values)

s. The merged file can be applied to another MapReduce

t. TakeAway → wait until all the map function ends and then start the reducers. Why?

- Reducer might not know whether there are another value for the key in the mappers that are not completed (might not take all the values)

u. This simplicity is what makes MapReduce successful

5. MapReduce: Programming
   a. Programmer provides MAP and REDUCE function
   b. Infrastructure provides everything else! (that is hidden to the programmer)

Lab 1 Part B: Write map and reduce for word count

```
map(k,v) {
 split v into words
 for each word w
     emit(w,1)
}
```

```
reduce(k,v) {
  emit(len(v))
}
```

Typically simple functions — easy for app programmer

c.

d. All machines in the cluster get a copy of the map and reduce functions so it can be executed (programmer does not care about the errors → what if something breaks?)

e. All other details are handled by the infrastructure system

f. Example: URL access frequency

Input: request logs

```
GET /dumprequest HTTP/1.1
Host: rve.org.uk
Connection: keep-alive
Accept: text/html,application/
xhtml+xml,application/xml;q=0.9,*/
*;q=0.8
User-Agent: Mozilla/5.0 (X11; Linux
i686) AppleWebKit/537.22 (KHTML, like
Gecko) Ubuntu Chromium/25.0.1364.160
Chrome/25.0.1364.160 Safari/537.22
Referer: https://www.google.be/
Accept-Language: en-US,en;q=0.8
Accept-Charset:
ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

Output:

access count per URL

https://www.google.be/, 3567
http://maps.google.com/, 3564
http://www.facebook.com/, 1234

g.

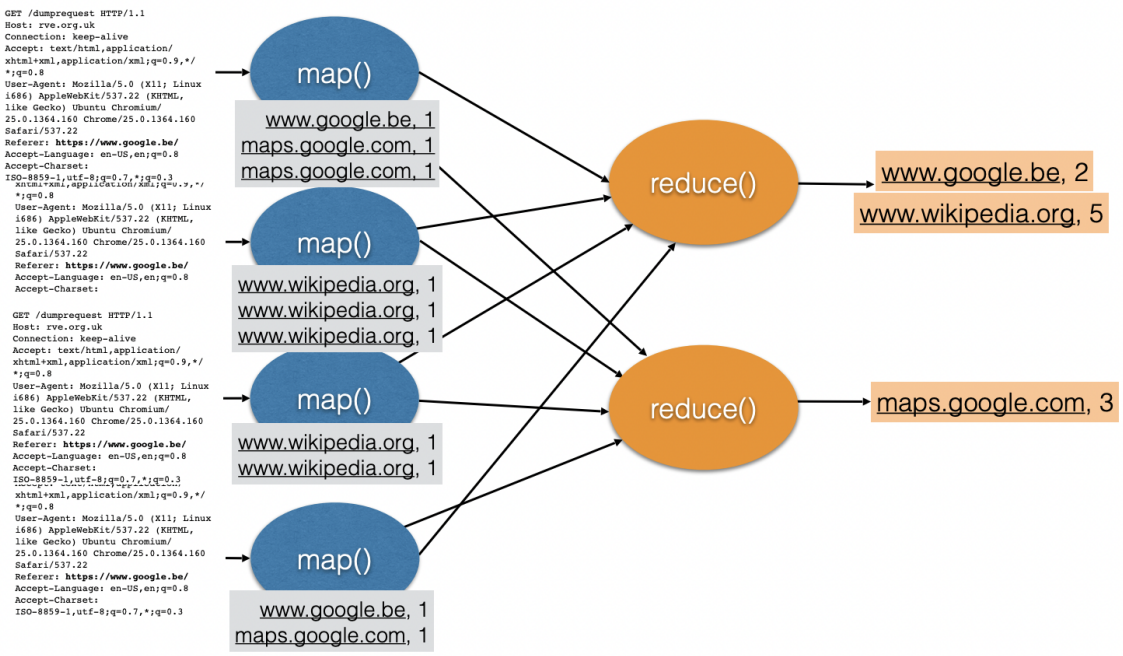
h. Key: URL

i. Value: number of access per URL

```
map(String key, String value):
  // key: document name
  // value: document contents
    for each URL u in value:
         EmitIntermediate(u, "1");

reduce(String key, Iterator values):
  // key: a URL
  // values: a list of counts
    int result = 0;
    for each v in values:
         result += ParseInt(v);
    Emit(key, AsString(result));
```

j.



k.

l. You can have same machines that run the mapper function and the reducer function (but cannot run on the same time)

m. Between the blue phase and the yellow phase