big data Lecture 3: map reduce

wbg231

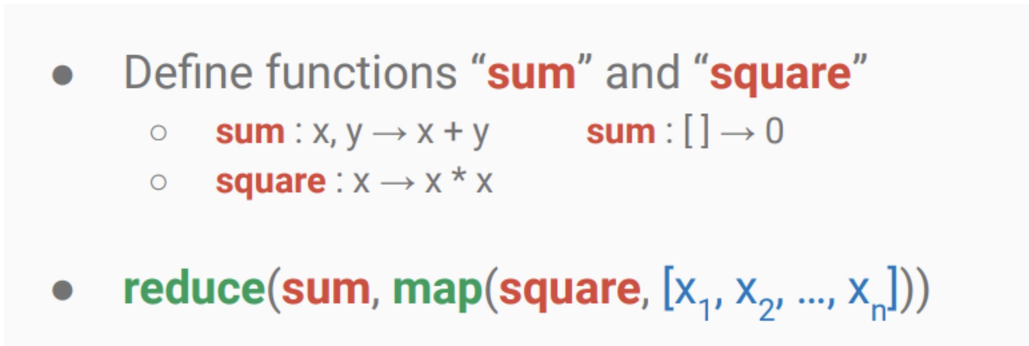
January 2023

1 motivation: text indexing

- if you have N documents and want to make an index mapping all words to the document it appeared in on a single machine it would take $\Omega(N)$ where N is the number of documents (that is you would at least have to look at all documents once and do something with them)
- but this problem is parallel, can just look at all documents on there own and combine results
- a parallel implementation with M computers would be $\Omega(N/M)$
- so that is a good idea but need a way to distribute the work and collect results that is where map reduce comes in

map reduce

- distributed programs are hard to write well, so if we restrict how we program it becomes easier to get parallelism
- so we are again getting power by restricting what we can do
- map and reduce are common operations in functional programming
- a map function looks like like map(function f, values$[x_1 \cdots x_n]$) $\rightarrow [f(x_1) \cdots f(x_n)]$ so that is it takes a function and applied it to some list of values and outputs a list with that funciton applied to all values
- a reduce function works like reduce(function g, values$[x_1 \cdots x_n]$) $\rightarrow g(x_1, reduce(g[x_2 \cdots x_n]))$
- a reduce function takes maps a list to a value. it does this by recursively applying g to pairs of items

- 

- the above example, explains a simple example. the map functions maps all elements in a list to its squared values

- the reduce function holds a variable called sum that is initialized at zero at recursively calls sum on that variable and the next list element

working with map reduce

- the programer must write a mapper and reducer function. the more simple the better

- the mapper consumers key value pairs as input and outputs intermediate key value pairs

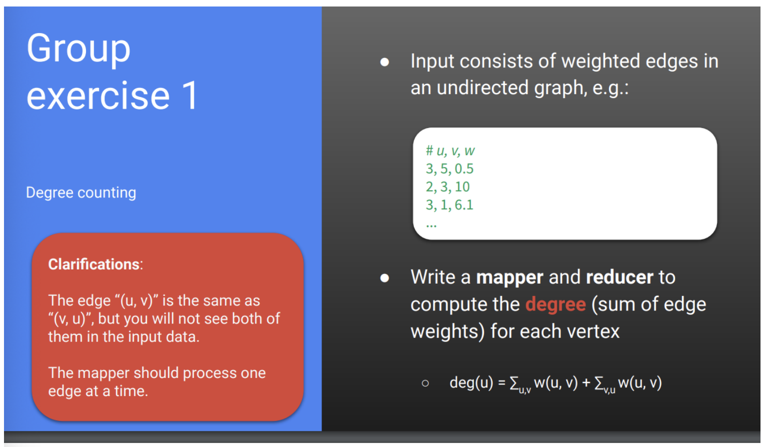
- the reducer consumes a single key and list of values and produces values for each key. note that in map reduce unlike in functional programming the reducer is not applied recursively

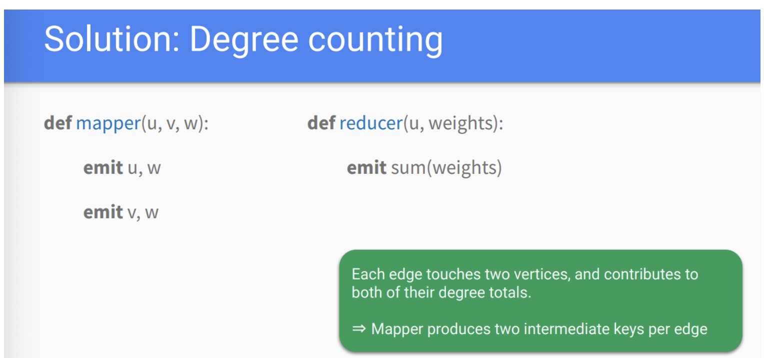
workflow

- map phase
  - distribute data to mappers
  - run mappers on chunks of data to get intermediate key value pairs
- sort shuffle phase
  - assign a parton of the intermediate results to each reducer by key
  - move data from mappers to reducers
- reduce phase
  - run the reducer call on chunk keys
  - collect the output

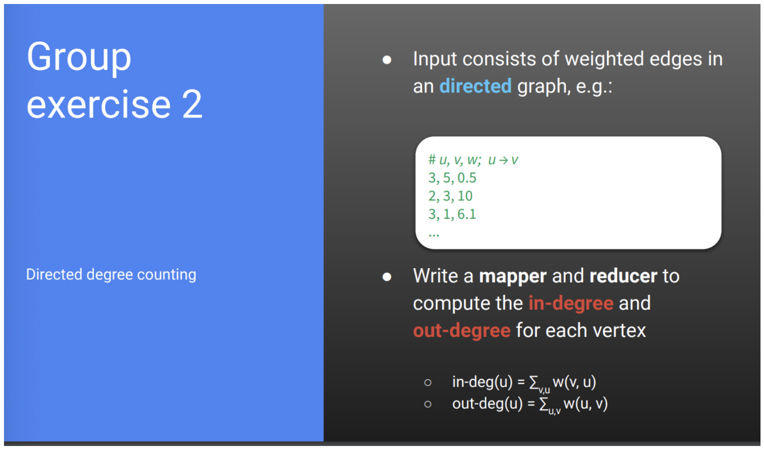
- make the mapper simple
- let the mr frame work route the intermediate results
- keep the reducer simple if possible

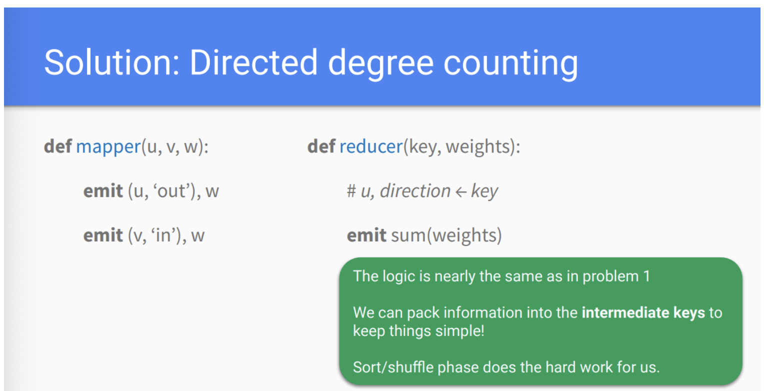
slido example

- 

- 

- this one is pretty straight forward
- tips for map reduce
  1. do not use floating point keys they don't hash well
  2. keep map and reduce simple (the sorting is what is optimized so let that do as much work as possible)
  3. compare your algorithm to a simple implementation

- 

- 

- the logic is the same as the last problem the real point is think about changing your keys before making your map program more complex

map reduce in practice

- can intermediate outputs be randomly assigned to reducers
- no we need to make sure all intermediate outputs with the same key go to the same reducer
- item this leads to key skew
- so suppose there is are 4 keys and 4 reducers but one key shows up 95% of the time, then that one reduce worker is doing 95% of the work this is called key skew or data skew

combiners

- key skew causes high latency

- lots of keys also means a lot of communication
- we can make the reducer's job easier with combiners which reduce intermediate key value pairs within the mapper worker, before shuffeling the data

Heuristics for using map reduce well

- have fewer mappers than inputs
- having fewer reducers than intermediate keys
- combiners can help but sometimes a more complex map is better
- sometimes doing some other sorting stuff can reduce communication

criticism of map reduce

- map reduce has criticism because it is
  1. too low level
  2. not well implementation
  3. not novel
  4. missing key DBMS features
  5. and not compatible with DBMS tools
- i mean i read the paper 1 4 and 5 are for sure, and i would say probably 2 (due to key skew)

too low level

- map reduce does not have schemas, so a programer has to infer them as they work, and there is no garuntee of data consistency
- map reduce does not use high level access language like sql which people think is better
- there are tools that can be used in tandom with map reduce that can address this

poor implementation

- map reduce does not have the ability to index
- is this a major issue? depends on the context

not novel

- map reduce uses old ideas so was not novel but that does not really matter

missing features

- many features from DBMS are not in mpa reduce
- why are they missing? is this a fair comparison? I am not sure

lack of DBMS compatability

- since this was published this has shifted a lot so not really an issue any more

why was map reduce so sucessfull

- map and reduce are simple abstractions that are powerful
- many jobs are only one shot (ie only need to be done once) so it may not be worth the time to build a database infrastructure

map reduce is not a database managment system

- but there is some overlap, simple queries can be done in map reduce but not hard queries

map reduce is not a general comutation engine

- maps are reducers are really flexible so a lot can be done with map reduce
- it can not however work with iterative or recursive algorithms
- not can it work with non deterministic functions

no transactions

- data base management systems use transactions to make sure data is consistency
- map reduce does not have transactions but still avoids these issues by having data be immutable, and require programs to be deterministic
- it has error tolerence if a node fails we can re assign idle workers to do its task

real issues with map reduce

- key skew so there are latency and scheduling issues
- we store intermediate results in many files
- not all tasks fit neatly into map reduce including non deterministic tasks, iterative algorithms or recursive algorithms
- cant do visualizations

what is the role of map reduce today

- map reduce is good for large batch jobs that run once or infrequently
- like data transformations or feature extaction

why is map reduce studied

- it is important for the development of big data tools
- map and reduce functions are a good way to break down problems
- the hadoop eco system is much bigger than map reduce
- there are still legacy code bases run in map reduce