Big data lecture 7: dask

wbg231

December 2022

1 introduction

1.1 how is lab 3 going

- tbh i have nto started it
- there is key skew so like make sure you start working on it reasonably early.
- converting sql to pyspark is hard.
- item here we are spending a lot of time with the documentation it is of variable quality plan ahead for bad documentation

1.2 road map

- we have done introductio, relational database, map-reduce, hadoop, spark, column oriented storage
- now we are going to do dask
- then we are going to deal with algos

1.3 what we have seen so far

- started with file systems
- then we went into relational databases, this was restricting data into schema
- then map reduce and HDFS that was much more about restricting the data rather than restricting data
- but it is a pain to code in map reduce. it is fun once you learn how to do it, but not a good time

- then we got to spark which adds more high level interface to some of the ideas from map reduce. spark gets away from that by getting granular acess to the data, and having all data stored in an rdd
- then they put the data frame interface on top of the rdd so we can do all the sql stuff.
- then there is PARQUET and column oriented storage

1.4 spark

- Spark is super powerful and super popular
- it works well
- but it does not do everything.
- so... what are some kind of computations that do not naturally fit into spark?
    1. hmmm maybe cases where data is in from many previous rdds so like it has wide dependencies. that could cause spark to be quite slow, but i mean it would not be well suited for it.
    2. it is still fundamentally reliant on the ability to do things in parallel so if there are cases where you need to to compute everything before moving on it could be non-ideal
    3. everything is done in rdds which is bad
    4. some problems can only be broken to some level like gradient descent (all gradients must be computed at a single step before we can move to another step) or full joins, matrix stuff
- the point is spark is good sometimes but not all the time

1.5 issues with spark

- data that does not fit into an rdd (sometimes you can force it)
- there is no python integration
- machine learning and things that are GPU oriented are tough with spark
- hard to zoom in or out of data.
- can not really index data.

2 dask

2.1 intro

- dark is like spark, if spark was written by sci py people
- Python based distributed computation
- spark (is in scala)
- this was written for dealing with scientific computing
- it is scaled up down in an ez way
- there are a lot of little diferences as well, the documentation is also pretty bad.

2.2 delayed computation abd task graphs

- dask builds complex computations by composing deferd computations into a task graph
- like spark nothing happens unitl yoou take an action
- if you use the dask.visualize() then you can visualize your code. that is helpful for understanding what happens
- nothing happens until an action is taken

2.3 collections in dask

- there are three collection interfaces that do difrent things
- bags
  - similar to an rdd,
  - distributed collection of abstrarly structured data. (that is there is no order to the data)
  - it serves the same role, of partioning data into small units
  -
- data frames
  - a wrapper on top of pandas data frames
  - the interfacte is pretty consistent
  - structured tabular data with a schema

  - pretty similar to data frames in spark (but it is on pandas instead of rdds)

- arrays (think tensors (x,y,z,w)) that is like bigger than matricies
  - distributed n dimensional arrays
  - the idea is to devide tensor data in multiple dimensions simultatinasly
  - this is what stands out in dask there is no way to use this in spark
  - these are pretty simialr to nd arrays in numpy
- unlike spark there is no explicit idea of a transformation ie (rdd→rdd)
- that is a coltural difrence between scala and python

2.4 bags

- pretty close to a spark rdd
- in dask you kind of need the bag interface
- bags collect items and partion them, and give you some bassic operations like mapping a funciton over it filtering it, returinign ture or false, slicing, aggreagation etc
- example using a bag to square a range of numbers and sum them.
  - import dask.bag as db
  - b=db.from sequence(range(5)) (these are the numbers one to 4)
  - c=b.map(square)
  - c.compute() (this ellementwise applies the function square to each ellement in the range)
  - c.sum().compute() (this computes the sum of those squares)
- note that nothingn is done until the compute() method is called
- bags are pretty primitive
- but they are really good for pre-processing
- then after things are preprocessed you move on to more high level computation

2.5 dask bags vs spark rdds

- both parition data across multiple machines
- they are both imuteable (that is once the bag is instanciated it can not be modfied that is a very importat feature of dask)
- rdds have types
- bags are untyped they can have elements of any type (this can be good or bad depending on waht you are trying to do)

2.6 common work flow

- have raw data that does not naturally go to a data frame
- import the data put it in a bag, and do some data cleaning
- then once the data is in a state where it is cleaner put it in a dataframe
- **the sooner you reduce the size of your data the better**
- this opens up some space for creativity
- a major bottle neck to efficency is sending data between machines, so the ealirest you can get rid of as much data as possibel the better
- as going through the lab thing what are the absolute min number of things i need to get what i need done, and then how can i most quickly get rid of everything else most often that is about using maps and filters in python and appplying those to bags
- however note that you do not want to do everything with bags (preforming operations on bags is really slow)
- because bags make less assumptions than spark rdds they can not assume as much .

2.7 slido bag question

- have a large collection of data files to load into a dask bak which of these is the most efficent
  - db.from_sequence([load(f) for f in files])
  - db.from_sequence(files).map(load)
  - both are the same

2.8 bag folding vs grouping

- try to avooid using group by
  -
- instead us bag fold by
  - this is better because it does the aggreagtion within each partion and then combines them, (so same thing as combiners)
  - here these have to be binary opperations
  -

2.9 data frames

- they wrok just like ou would expcet simialr to spark data frames
- outside of addding how the data frames are loaded and putting a .compute()
- **repartioning data frame in spark is important for lab 3 part 2**
- dask is not good at managging partions, so think carefully about how your data is partioned so think careuflly about how data is partioned when settigng up yourr data graph
- dask will preserve the partion structre for taht type os structre, so some times the partions will not be well partioned which will cause ket skew
- i got kind of distrcted back to paying attention

3 arrays

3.1 arrays

- dask arrays work like numpy arrays
- parallelism is not limited to rows (can define chunks to each dimension)
- large arrasy are assmebled implicity for many small arrays
- most numpy operations carry over fine
- these are more of less a data structre that has many numpy arrays

3.2 chunking example

- bassically you can treat chunks as numpy arrays
- we kind of rushed threw it

3.3 things that chunks are bad at

- sorting is bad in chunks (but you can often get away with topk instead (that is if you need the top k ellements form a chunk))
- often that is a lot more efficent than sorting teh whole array any Wednesday
- it is not really good at predicting from the computation graph what the dimensions of each collection will be
- operations where output size epnds on value in the data (ie not just on the computation graph) are very tough for it. So like if you want to do $x[x > 0]$ ie just the postive ellements
- linear algebra stuff is kind of dicey becuase it is comutationally expensive

4 wrap up

- does dask replce spark?
- no it is pretty simialr just used with pyhton
- it is all a function of if you enjoy it more or not