lecture 7: Dask

wbg231

January 2023

1 review of the story so far

- started at file systems
  - they are unstructured collections of files
  - there are many custom ways to store data
  - very flexible
  - there is no built in parallelism
- relational data bases
  - restrict the structure of data to relations
  - has a standard interface (SQL)
  - somewhat flexible
  - not easy to parallelize
- map reduce and HDFS
  - data is less structured than RDBMS
  - restrict coding to map and reduce functions
  - very parallel
- spark
  - structured data like RDBMS
  - distributed storage (HDFS)
  - standard-ish interface (sql and spark-API)
  - very parallel

spark is good

- spark integrates well with java based tools like hadoop
- spark is grate at working with data frames and SQL like data processing
- it is 10 years old meaning the implementation is mature and stable
- after RDBMS and SQl it is likely the most used software for data analysis

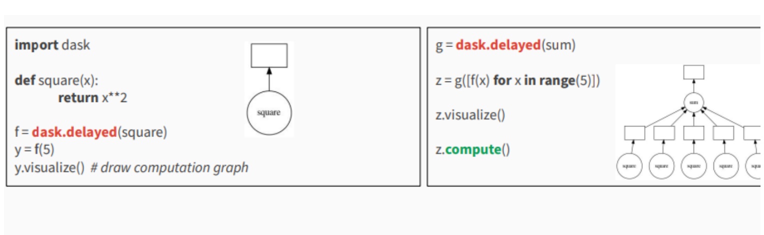
what is spark not good at

- data that does not fit nearly into an RDD or data frame model
- it is not super well suited for working with the python SciPy stack
- modern machine learning is all done using SciPY
- spark is only cluster based what if the data you are working with is to big to hold in RAM but don't need a cluster?

Dask

- Dask is python based distributed computation
- has a lot in common with spark
- has computation graphs similar to sparks linage graphs
- has delayed computation much like RDD and data frames in spark
- collection based interface also has spark like data frames
- some key differences between spark and dask are that
    1. dask prioritizes array based Numpy like computation
    2. it is designed to support single machine out of core (ie parallel or multi thread) use

delayed computation and task graphs

- dask builds complex computation by composing deferred computation into task graphs
- 

- the left shows how a delayed computation square function can be defined
- the right shows how we can pass that computation to a delayed sum function and look at it in parallel

collections in dask

- bags
  - distributed collections of arbitrary structured data. similar to an RDD in spark
- data frames distributed collection of structured tabular data, most similar to a spark data frame (but built on pandas instead of using RDDs)
- arrays: n dimensional distributed numpy arrays.

collection interface bags

- bags are broadly similar to spark RDDs. think of them kinda like a list in python
- they are un ordered collections of generic python objects. portioned into subsets
- they implement basic operations like map filter join sum etc
- a good choice for initial preprocessing and structured objects
- if your data is tabular or array based this is likely not a good data structure

dask bags vs spark RDDs

- both partition a collection of objects across multiple machines
- both are immutable
- all elements of an rdd must be of the same type
- bags are untyped so the contents can be mixed types (but this should be avoided as it slows down processing)
- a common dask work flow is raw data → bags → data frames → deeper analysis
- the earlier you reduce the size of your data the better, as that is less data moving through the system
- we prefer maps and filters on bags over data frame manipulations when simplifying data
- but in general bag operations are slower than those on data frames, since because bags have so few restrictions it is hard to optimize code for them

constructing collections

- there are many ways to construct a collection in dask
- db.from_sequence.([logad(f) for f in files]). all files are loaded first in vanilla python lists and then put in a bag
- db.from_sequence(files).map(load) file names are distributed into a bag, the load function is then applied to each in parallel
- so the second is faster
- in general try to load data using collections when possible

bag folding vs grouping

- try to avoid using group by on bags, this requires inter worker communications which is slow. there are wide dependencies with this method
- use fold or fold by if possible
- this method has similar benefits to a combiner in map reduce, that is it does local aggregation fit to reduce data shuffling
- fold by required a key function and binary operation
- a key function maps elements to a key (think of this as the group)
- a binary operation reduces within the group (think of this as group aggregation)
- a binop and reducer in map reduce are not the same however
- bag binop only sees two values at a time not a list like in map reduce
- output of a binop must match the input type unlike in map reduce
- this can become tricky if your bag elements are structured

collection interface data frames

- they are similar to spark dataframes (they use pandas internally through)
- parallelism ie partitioning is over rows in dask data frames not cols
- these are a good choice for data that can naturally be split into different files. like multiple log records
- managing partitions in dask is important.

- your data may change through the computation graph if this is the case, you may end up with nearly empty partitions
- try to keep your partitions are full and balanced as possible
- dask makes you work harder than spark to get it working well

collections interface arrays

- dask arrays work like those in numpy
- parallelism is not limited to rows, can define chunks alone each dimension
- large arrays assembled implicitly from smaller arrays
- most numpy operations work automatically

where chunks fail

- not good for sorting between chunks (but can sometimes just take the top k elements from each chunk and use those)
- operations where the output size changes like masking operations are hard
- linear algebra operations can be tough

how spark is commonly used

- have 10 models for audio segmentation to compare
- have 2000 audio recordings in a dataset, that is 20,000 model outputs
- the model outputs are a sequence of time intervals
- Model evaluator compares reference annotation to estimate annotation for one track, and produces a dictionary of scores along different metrics. Takes a few seconds to run for each track.
- What I want: a DataFrame containing: model id, recording id, [scores for each metric]

solution

- store model output as separate files on disk
- creat a delayed function to map filenames to scores (calls the evaluator)
- crate a bag from the delayed function
- convert the bag to a data frame and save it

why is this better

- all computation is done using basic python functions (ewe don need to re-write our functions for dask)
- the problem it's self is parallel, so there is a lot of mapping but little reducing that needs to be done
- we like this over spark because we dont have to change any of our data structures or code to fit it

working with large numerical data

- csv and parquet files are not a good option here. (like a single really big matrix)
- collections of files npy or npz files can work well
- Hierarchical data format (HD5) could work well as well

HD5

- basically a new file system within a file (Hierarchical) Directory structures

does dask replace spark

- it kind of depends on use case
- pros for dask
  - works well with SciPY
  - works well with dense multi dimensional data
  - works well with custom algorithms and gpus
- pros for spark
  - more stable
  - more high level, you do not need to think as much about computation graphs or partitions
  - faster for data frame analysis
  - better support for large graph data

HPC

- hpc greene is less restrictive than dataproc
-