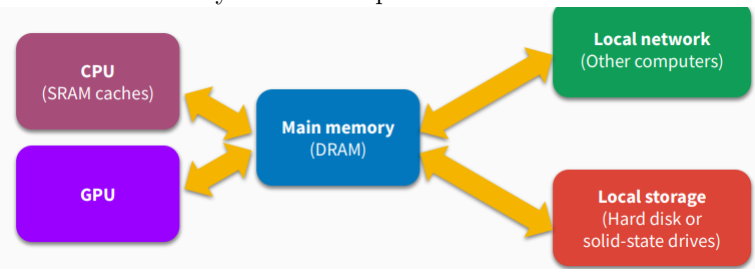
big data Lecture 1: Introduction

wbg231

January 2023

1 big data background

- here is the basic layout of a computer



- we have the following resources
  1. storage that is where and how much data is kept
  2. communication that is how quickly can we move the data from one place to another
  3. computation how quickly can we process data
- the cost of storage has fallen over time
- both the volume of data and the speed it is created have risen over time
- l1 and l2 cachce
- long story short l1 is the lowest building block of the cpu it is the fastest but can not hold much data an l1 cache read takes 1 nano second
- the l2 is a slightly larger piece of memory with more space, an l2 cache read takes 4 ns
- reading to main memory (ie dram) is the next step up and takes 100 ns
- then reading from local storage (ie your hard drive) takes much larger
- than reading from a network takes even longer

- moore's law the number of transistors on a micro chip (not speed) doubles every two years has widely been true
- it is wroth knowing that at this point having faster transistors is less important than having more transistors as that is where we see parallelism
- we need more systems of computers not independent machines

file systems

- to understand why the tools that came later developed as they did we need to go back to file systems
- the key ideas of file systems are using directories to organize data, and grouping structured data into files (that is data can preset across multiple runs)
- pros of file systems
  1. they are easy to implement
  2. they are portable between systems
  3. network file systems like google drive enable limited distributed processing
- file systems work well when
  - the data does not have obvious structure to index, then can put them in files (like for instance system logs)
  - if indexing is not worth the cost (like one off jobs)
  - if you are very concerned about portability between systems

file system cons

- they do not expose or exploit structure of the data
- each query requires a new program is written (hard to re use)
- directories and hierarchies may be too restrictive (like if we have files that are conceded in some non obvious way)
- file systems are bad when
  - the data is structured along multiple axes
  - when the data has complex interactions
  - when analysis is complex
  - when the benefits of indexing data outweigh the costs.
  - in cases like this we can use the relational database model

methods for dealing with large datasets in a file system

- sample ie just take part of the files or a piece of a file
- stream processing look one record at a time
- data structures (is using an RDD a good idea)
- indexing (sorting data so looking up records is fast)
- parallelism

- it is wroth noting that databases did not replace file systems
- file archives are still a common way to share large datasets (but we include metadata to show how that data is structured)
- hadoop relies on a distributed file system, then you build database abstractions on top, but this comes with restrictions on structure
- so the key difference today is that we use standardized file formats

things to consider when choosing what tool is best

- do you need an exact solution to the problem (is an approximation good enough) if so may only need part of the data
- di we expect records to be wrong?
- will the dataset grow over time?
- will the number of features we are looking at ever change?
- how complex is the way we are doing things?

slido

- Say you had to compute the mean and covariance matrix of a "large" collection of 100-dimensional vectors. How would you rank the above strategies (best to worst)?
- sampling will give you an approximation but may miss outliers
- stream processing will limit memory usage but not CPU time
- indexing does not help: this problem uses all the data
- parallelism will work if you have access to many machines
- a good approach would b eto stream first to limit memory usage (which is always helpful) then either use parallelism or sampling depending on if an approximation is ok

what is next

- data base management systems (DBMS) they provide a standardized interface to store load and process data.
- that relation mode which imposes constraints on how the data is organized but gives us speed ups
- putting these two ideas together yields the relational data base model (RDBMS)