Performance 6.s080 Lecture 10/21/2019
===========

Performance overview -- what to do when Pandas isn't fast enough or memory isn't large enough

Today - Computation is too slow
Next time - Data is too large

General approach:
- measure & quantify problem
- identify bottleneck
- propose fix:
  - strategies
    - better hardware
    - better execution environment
    - better algorithm
    - better implementation

<u>Suppose you find yourself working with a dataset where performance is slower than you like?</u>

What does "slow" mean?
- In some cases, days of delay might be ok, e.g., for a final training run of a ML model
- In other cases, a few seconds may be too long, e.g., for an online visualization being rendered for a web page.

<u>So how do we systematically approach performance problems?</u>

1) Find the bottleneck
2) Improve performance of bottleneck, e.g., by adding hardware, or redesigning software

<u>How do we quantify performance? [slide]</u>

Throughput: requests/time, for many waiting requests.
Latency: time/request, for a single request.

<u>Are these always the inverse of each other?</u>

A: Sometimes
E.g., 1 request takes 0.1 seconds of CPU, 1 CPU => tput 10 req/s.

Often not inverses:

Concurrency: with 2 CPUs, 0.1 second latency => tput 20 req/s.

Which of these matters more? A: Depends on setting

In most data science applications, focus is on latency, but if your work is being used in an online or multiuser setting, throughput may matter.

Often a few hundred milliseconds of latency is immaterial, but may want to scale to very high throughputs.

OTOH, adding seconds of latency, e.g., in a web server, is bad. And even 10s of milliseconds, if you are doing high frequency trading, may be super important.

<u>What do these look like as systems scale, typically?</u>
<u>[slide: graph]: # users on x-axis, tput on y-axis, linear then flat (queueing).</u>

[slide: graph]: # users on x-axis, latency on y-axis, ~zero then linear (queueing).

<u>How do we find the bottleneck?</u>

Example w/ Pandas

```
df = pd.read_csv(PATH, delimiter='|', header=None, names=header)

print df[df['NAME'].str.contains("MADDEN")][["NAME","EMPLOYER","TRANSACTION_AMT"]]
```

Measure utilization of each resource (CPU, disk, network, ..)
Profiling / sampling is a common approach.
Sometimes easy: e.g., CPU is 100% busy, disk is 20% busy.
Sometimes not: e.g., CPU is 50% busy, disk is 50% busy, but alternating.

<u>Tools: print statements & time, top and system profilers, codes profiler -- show demo & slides.</u>

- show examples

Model the performance of your system.
E.g., say net should take 10 msec, CPU takes 50 msec, disk takes 10 msec. Helpful in estimating bottlenecks in overall system design.
Helpful in debugging components that aren't performing as expected.

Some of you may not be used to thinking about disk and CPU performance separately, but it's important to understand the bottleneck.

Modern SSDs are very fast, so less and less likely to be the bottleneck [show slides], but still can be an issue.

<u>How fast should my laptop be able to read this file?</u>

Sequential vs random I/O -- why does this matter and what does it mean?

How much performance difference do we expect?

What should Pandas be doing here -- Sequential I/O

Very important to have numbers at hand [show important numbers slide]

<u>Back to example:</u>

Based on numbers 1GB/sec, 300 MB, so .3 seconds to do read.

<u>Show profiling -- takes 7+seconds -- why?</u>

What about time to find Madden records? Harder to model CPU work. But 2M records, a few hundred CPU instructions per record, ~ 400M instructions? On a 2 GHz processor, should take ~0.2 seconds! Takes 5-10x more.

This is a guessing game.
In a complex system, bottlenecks may not be obvious.
Will have to iterate on guesses based on the above approaches (+ others). Fix candidate bottleneck and see what happens.

<u>How do we fix a bottleneck?</u>

Amdahl's law

Some things we might try:

- better hardware
- better execution environment
- better algorithm
- better implementation
    - caching
    - indexing
    - partitioning & parallelism -- not today

Better hardware?

Can I just buy more hardware, or wait for the next processor generation?

Improvements in technology do not solve all problems, in particular due to incommensurate scaling, where some resources don't improvement as much as others. For example, memory bandwidth and latency are not improving much, and CPU performance has largely stalled.

Better execution environment -- often different languages will offer different performance

Example w/ Postgres, Pandas, C.

Clicker question:
Which do you think is going to result in best performance:

- rewrite to use lower level python instead of pandas, e.g., loops w/ readlines
- rewrite in C
- rewrite to use a relational database
- none of these, pandas implementation is best

Strategy ies:
- maybe Pandas is slow? try rewriting using loops in Python -- slow.py
- try rewriting in C -- slow.c
    - go over code
    - show console timing
- try running in SQL -- slow.sql
    - not a lot better -- but we'll see how we can use indexing to speed this up a lot

Conclusions:
- parsing data is the bottleneck
- python is very slow
    - why? [show slide]
- pandas is actually not bad, because it uses C implementations underneath the covers
- rewriting in C is painful but can be a big win
- you can call into C from python if you have a specific algo you want to rewrite

Better algorithm -- lots of things we can do.

How could we do better than a linear scan through the data?

Can't! But what if we know we're going to do lots of lookups?

Show slide - trigram index

```
explain analyze select NAME, EMPLOYER, TRANSACTION_AMT from donations where NAME ~ 'MADDEN' ;
```

This is one of the huge advantages to a database -- when we know we are reaccessing data like this, we can improve performance a lot.

What other general techniques can we apply to improve data process performance?

1. Eliminate nested loops

Show python example with nested loops, 10K filter

Show how to rewrite has a hash join -- slow_pandas_2.py, note commented blocks

Show how to rewrite w/ Pandas

Show what happens on full dataset -- memory explosion

Show what happens in Postgres -- slow_join.sql

```
explain select count(*) from donations d join donations2 d2 on d.NAME = d2.NAME;
                                          QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=1035901.63..1035901.64 rows=1 width=8)
   ->  Gather  (cost=1035901.42..1035901.63 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=1034901.42..1034901.43 rows=1 width=8)
               ->  Merge Join  (cost=506961.98..960766.03 rows=29654155 width=0)
                     Merge Cond: ((d.name)::text = (d2.name)::text)
                     ->  Sort  (cost=159406.60..161464.19 rows=823036 width=16)
                           Sort Key: d.name
                           ->  Parallel Seq Scan on donations d  (cost=0.00..64474.36 rows=823036 width=16)
                     ->  Materialize  (cost=347516.24..357350.72 rows=1966896 width=16)
                           ->  Sort  (cost=347516.24..352433.48 rows=1966896 width=16)
                                 Sort Key: d2.name
                                 ->  Seq Scan on donations2 d2  (cost=0.00..74673.96 rows=1966896 width=16)
```

Note that merge join is a poor choice.

```
\pset enable_mergejoin=false
```

Hash join yields performance that is about 2x faster

2. Predicate push down
3. Project away unneeded data

Show example in Pandas

4. Cache intermediate results

Suppose we know we are going to work with Madden records a lot -- might prefer to save those to a separate file, or a table.
Can write out pandas arrays, or select into an intermediate table in Pandas

[Show summary]