Genome 540 discussion

February 27th, 2025

Joe Min

Genome Sciences
UNIVERSITY OF WASHINGTON

Agenda

Data storage parallelization

Local computation parallelization

Server-scale computation parallelization

Data storage parallelization

Centralized data storage does not scale

Consider a sign-in table at an academic conference for 100 total people

If we have one sheet with the name of all registered participants on it, we can check for each person’s name by running down the list and checking them off

The longest someone will have to wait is for 99 other people to check in (maybe not too bad)

Centralized data storage does not scale

Now consider a larger conference of 10,000 people

Using one sign-in table, someone might wait for up to 9999 other people before getting to sign in themselves

If we want people to still only have to wait for up to 99 other people, having a single list with everyone’s name on it is no longer viable

Parallelizing data storage scales better

Let’s instead split this into 100 different sign-in tables using everyone’s first names

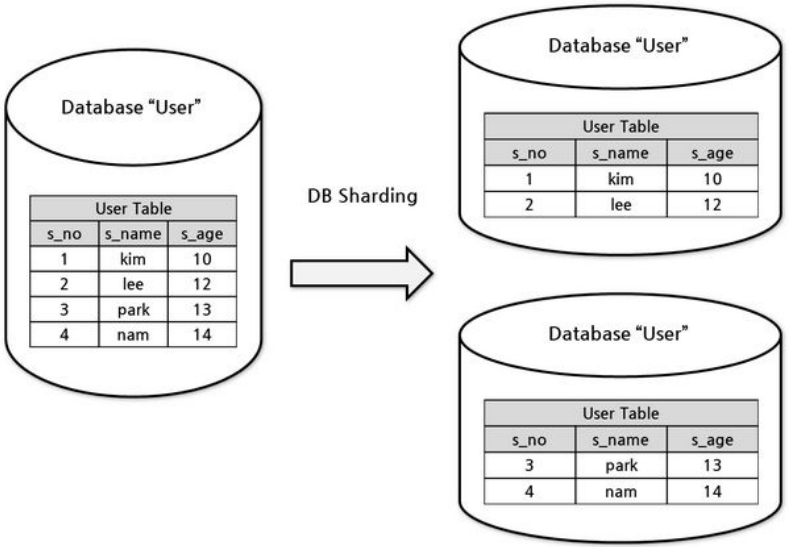
- e.g., Aa-Am: Table 1; An-Az: Table 2, etc.

Now, each individual can check in faster!

Additionally, each checking operation is less expensive (the person at the table has a shorter list they need to consider and carry around)

Parallelizing data storage scales better

This is the basic underlying strategy behind “database sharding”, which effectively splits up your data into smaller, more manageable and efficient pieces



Picking a shard key is crucial

We have been using names for conference attendees/users of a website

Other data also have inherent properties that can be used as shard keys (e.g., genomic coordinates of a gene)

Otherwise, can hash a certain part of the data and use the result as your shard key (but then picking a hash function is crucial)

Desired properties of a shard key

Even distribution

- A good shard key evenly maps to the provided buckets
- In turn, the buckets need to be designed accordingly (e.g., we may want names starting with ‘A’ to have their own bucket; but ones starting with ‘X’, ‘Y’, and ‘Z’ might share a bucket due to their infrequency)

Desired properties of a shard key

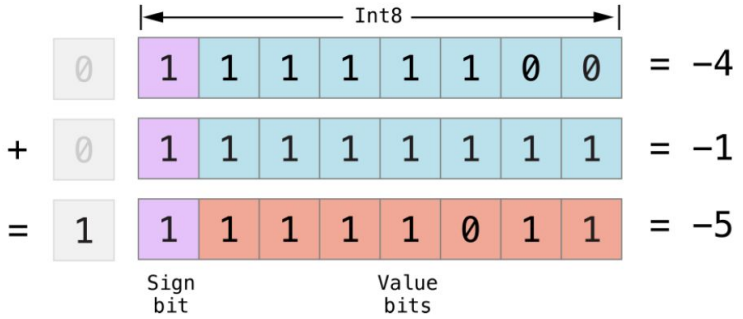
Efficient access patterns

- However, some of the data buckets might simply be accessed more often even though the buckets are the same size
- In this case, we may want to pick a key that shards by data access patterns instead of distribution
- E.g., an online library may shard by book popularity instead of book title

Local computation parallelization

Different levels of parallelism

Bit-level parallelism



- Increasing processor size allows for parallelization
- E.g., if we want to add two 16-bit integers:
  - An 8-bit processor needs to spend extra compute to break this up into two 8-bit addition operations that happen in serial
  - A 16-bit processor can handle this in one single 16-bit operation

https://www.geeksforgeeks.org/introduction-to-parallel-computing/
https://i.sstatic.net/7cfyg.png

Different levels of parallelism

Instruction-level parallelism

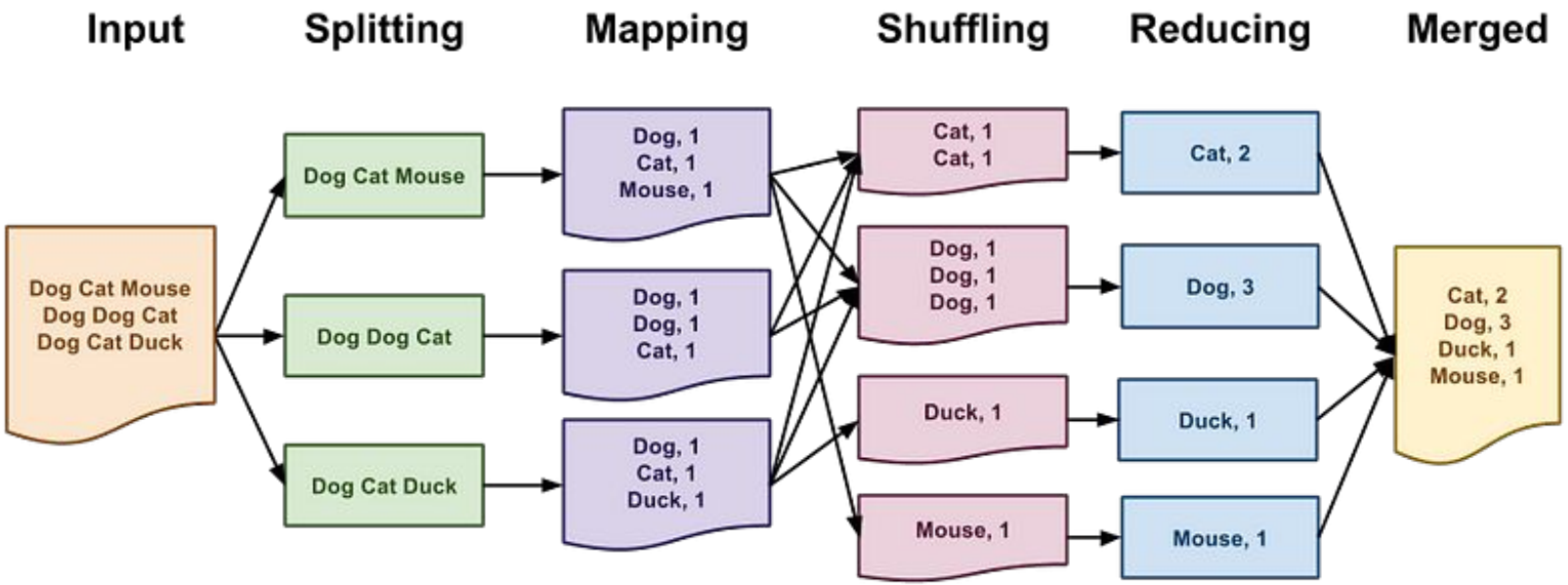
- Each core of a central processing unit (CPU) can only execute one instruction at a time
- Instructions can be ordered such that adjacent instructions do not affect the results of each other and thus can be run on different cores at the same time
- E.g., getting the value of two ints from different memory locations; one value does not depend on the other

Different levels of parallelism

Task-level parallelism

- If a task can be split into independent subtasks, we can allocate each subtask to different computers/nodes and later combine the outputs
- E.g., counting word occurrences in a large body of text; let’s see what an implementation looks like →

MapReduce algorithm



https://cdn-images-1.medium.com/max/800/0*Dx5RUyYZ7p-6GESb.png

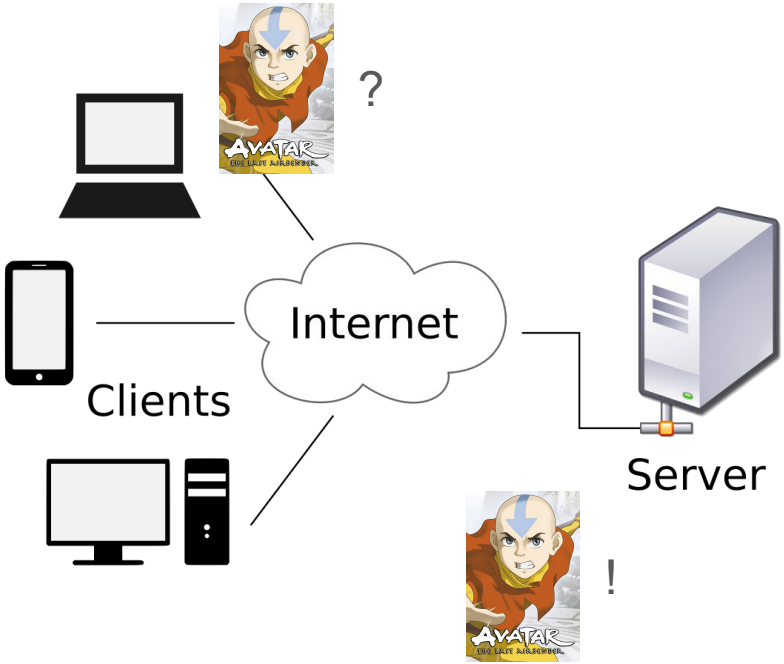
Language-specific APIs

Different languages have different support for accessing these levels of parallelism

- E.g., python has a nice multiprocessing library that gives you access to all cores of your CPU
- I think C++ has similar libraries (like Open MPI) but I've never used them before

Server-scale computation parallelization

What is a server?



Servers are computers that “serve” resources to other computers

E.g., when you access “netflix.com” asking to watch “Avatar: the last airbender”, a computer figures out how to get you that byte stream

https://upload.wikimedia.org/wikipedia/commons/thumb/c/c9/Client-server-model.svg/1200px-Client-server-model.svg.png
https://resizing.flixster.com/-XZAfHZM39UwaGJIFWKAE8fS0ak=/v3/t/assets/p8655066_b_v8_aa.jpg

We sometimes need to scale out servers

For such a popular title, however, simply having one copy may not be sufficient (multiple people may want to access the same content at the same time)

In this case, we may want to duplicate the content and “scale out” the server, e.g., 4x:

https://media.licdn.com/dms/image/C4E12AQGv6K8fizlDDQ/article-cover_image-shrink_600_2000/0/1564486772801?e=2147483647&v=beta&t=yVL_N2YZF5zHOPPN472nQsnYha_B-p_V1irUHoHvf50

Navigating scaled out servers

Who gets which copy?

- We can use a shard key!
- E.g., we can hash and map user_ids into 4 buckets; each “bucket” is now a dedicated server with a copy of the content

We’ll talk more about web servers soon!

Office hours

Reminder:

Homework 7 is due Sunday, March 2nd at 11:59pm!