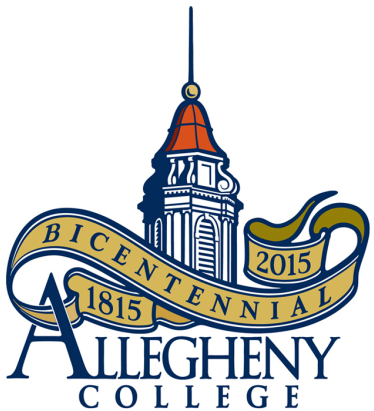
CS402 - Cloud Computing

Large Scale Computing 1

Aravind Mohan

Allegheny College

September 2, 2020



Lesson Overview

- How to scale a computational block within your machine?
- How to execute a computational block in parallel within your machine?

A Practical Example (follow up)

Compute: Find the total number of primes between 0 and a given number α

- **Recall:** We are given β, which represents the total number of workers.
- To scale the computation, we are required to **divide** the work equally between each worker, and **solve** the problem in parallel and fast.
- Assume α and β is unknown, what is the range for each worker? **(formalize the sequence)**
- Look at the (Prime.java) code file in the class repo.

A Practical Example (follow up)

Example 1:

$$\alpha = 100 \text{ \& } \beta = 5$$

(0 - 20)	(20 - 40)	(40 - 60)	(60 - 80)	(80 - 100)
w_1	w_2	w_3	w_4	w_5

Example 2:

$$\alpha = 240 \text{ \& } \beta = 4$$

(0 - 60)	(60 - 120)	(120 - 180)	(180 - 240)
w_1	w_2	w_3	w_4

Scalability

Scalability is the property of a system to handle a growing amount of work by adding resources to the system. (definition from wiki)

- In the context of cloud computing, scalability is defined as the ability to handle **growing** or **diminishing** resources to meet computational needs in a Robust way.
- Scalability is achieved through MultiTasking, Virtualization, and Cloud Computing. It is important to understand the first two items in the list, to fully understand, Cloud Computing.

Thumb rule: Adaptable to any number of workers and any data size.

A Practical Example (follow up)

Formula:

$$start = wid \times \frac{\alpha}{\beta}$$

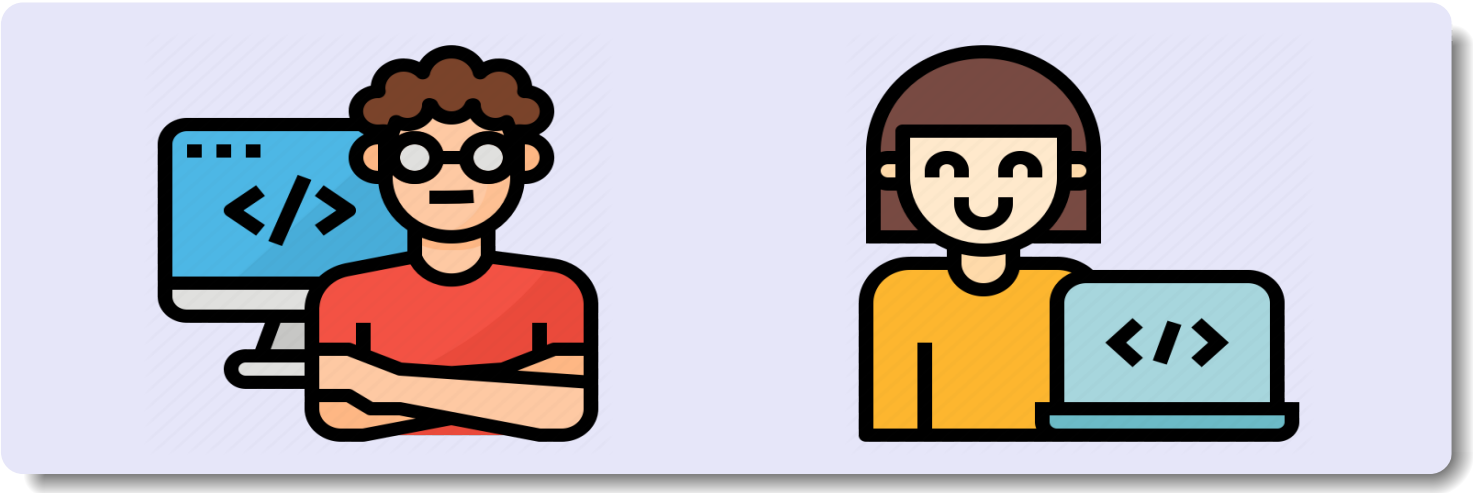
$$end = start + \frac{\alpha}{\beta}$$

Example 1:

$\alpha = 100$ & $\beta = 5$

wid	$start$	end
0	0	20
1	20	40
2	40	60
3	60	80
4	80	100

Let us code



Limitations of the formula

- What if worker id start from 1?
- What if worker id start from a random number? that is, worker id start differently on different machines? assume worker id start number is represented as γ

We will come back to these points later.

MultiTasking

- Process based: Two or more programs executing concurrently.
- Thread based: In one program, multiple tasks are executed concurrently.

Threads is a way in which a program is divided into concurrent tasks.

MultiTasking

MultiTasking	MultiThreading
1. **Process** based.	1. **Thread** based.
2. **Processes** are heavyweight.	2. **Threads** are lightweight.
3. Separate address space.	3. Same address space.
4. Costly to maintain (memory-intensive)	4. Cheap to maintain (less memory taking)

MultiThreading can take a toll on memory usage for cpu intensive tasks.

Why do we need Threads?

- To enhance parallel processing.
- To increase response to the user.
- To utilize the idle time of the CPU.
- Prioritize your work depending on the priority.

Example: Web Server Request Handling!

Thread State

Some important states of threads are provided below:

- Running
- Ready to run
- Suspended
- Resumed
- Blocked

Thread Implementation in Java

- Extending the **Thread** class.
- Implementing the **Runnable** Interface.
- Threads implemented using a runnable interface shares the same object. Wheras, extending thread class, creates unique object for every thread. (Implementing runnable interface is much more efficient than extending the Thread class.!)

Take a look at Thread1.java and Thread2.java files in the repo.

What's coming next?

Further Study on Threads:

- Discussion on the executor service framework to execute multiple threads in parallel.
- Discussing on the Synchronization concept in MultiThreading.
- Complete the Prime number generator example using MultiThreading.

More Advanced Study on Large Scale Computation

- Discussion on client-server based computation.
- Discussion on virtualization & computing through virtualization.
- Discussion on Cloud framework & computing through Cloud.

Thinking Exercise Before Next Class

- How to make the **Prime.java** program work for worker id starting from a random number?

Very interesting problem to solve from scalability perspective!

Reminder

- **Practicals:** on Friday, during class time.
- **No need for the class to join through Google Meet.**
- A practical worksheet will be given before class. Everyone will work on the tasks listed in the worksheet individually. All discussions will be done through **Slack**.

Reading Assignment

- **MultiThreading:** Read through the chapter on MultiThreading from the ”Object-Oriented Programming with JAVA Essentials and Applications” textbook by Rajkumar Buyya.
- **Reading** material available in both git repo and google drive.

Questions?

Please ask if there are any Questions!