Lecture 5.1

- Dr. Anurag Sharma

Increase efficiency with parallel programming: multithreading

References

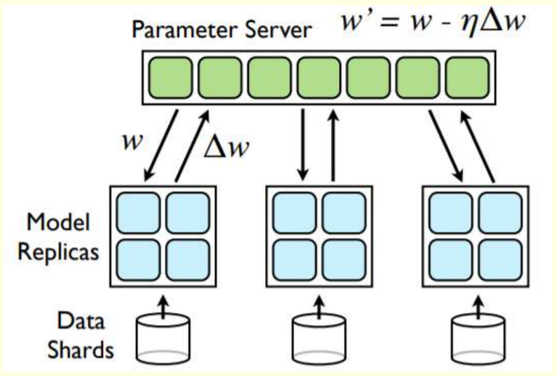
- C. Horstmann, Object-Oriented Design & Patterns 2nd ed., John Wiley & Sons, 2005

Problem statement

- The rapid expansion of data (or big data) poses a challenge for algorithms to handle a vast amount of data in a limited time. Traditionally these learning techniques were confined to centralized processing, but recently, there has been enough attention given in to equip traditional techniques to speed up the training process in parallel on multiple machines/CPUs/GPUs.
- https://www.youtube.com/watch?v=eVSfJhssXUA

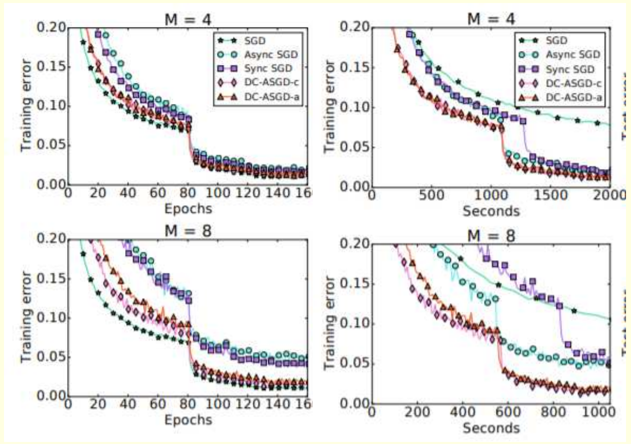
Sequential → Parallel

- J. Dean et al., "Large Scale Distributed Deep Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, USA, 2012, pp. 1223–1231.



High Speed but degraded solution quality

- S. Zheng et al., “Asynchronous Stochastic Gradient Descent with Delay Compensation,” in *International Conference on Machine Learning*, 2017, pp. 4120–4129.



Thread Basics

- Threads enable multiple tasks to run simultaneously
- How could multiple task run concurrently in a single CPU?
  - The operating system frequently switches back and forth between threads (tasks), giving illusion that they run in parallel

Thread Examples

- In a car racing game several cars run simultaneously. You will need separate thread for each car.
- Web browser can load multiple images into a web page at the same time.

Java Threads

- A thread is a program unit that is executed independently of other parts of the program.
- Java virtual machine executes each thread for a short amount of time and then switches to another thread. You can visualize the thread as programs executing in parallel to each other.

Creating thread in Java

1. Define a class which you want to run as a separate thread and implement the *Runnable* interface. This interface has a single method called *run*.

```
public interface Runnable{
    void run();
}
```

2. Place the code for the task into the run method of the class.

Cont.

3. Create an object of the class
4. Construct a *Thread* object and supply the above object in the constructor.
5. Call the start method of the *Thread* object to start the thread.

Java Example

```
class greet implements Runnable{
  private String greeting;
  public greet(String greeting){
    this.greeting = greeting;
  }
  public void run(){
    try{
      for (int i = 1; i <= 5; i++){
        System.out.println(i +": " + greeting);
        Thread.sleep(500);
      }
    }
    catch (InterruptedException e){
      e.printStackTrace();
    }
  }
}
```

Cont.

```
public class mythread {
  public static void main(String [] args){
    Runnable r1 = new greet("Hello");
    Runnable r2 = new greet("bye");

    Thread t1 = new Thread(r1);
    Thread t2 = new Thread(r2);


    t1.start();
    t2.start();
  }
}
```

Terminating Thread

- A thread terminates when the *run* method returns. This is normal way of terminating a thread.

- If there is need to abruptly terminate a thread during execution, a thread must be interrupted.
  - To notify a thread that it should clean up and terminate, use the *interrupt* method

    threadobject.interrupt();
  - This call does not terminate the thread; it merely sets a flag in the thread data structure.

Cont.

- In the initial release of Java library, the Thread class had a *stop* method to terminate a thread. However, that method is now deprecated because stopping a thread can lead to problematic situation if a thread holds a lock on shared resources and it is not released.

Interrupting thread

- To interrupt a thread type:

  threadobject.interrupt();

- To check whether the current thread has been interrupted write the following code:

  if (Thread.currentThread().isInterrupted())…

Race Conditions

- A race condition occurs if the effect of multiple threads on shared data depends on the order in which the threads are scheduled.
- All threads will compete to get hold of shared resources.