COEN 383 Final Study Guide

Question 1 - Parallel Processing

Study parallel processing paradigms in an environment when you have a multi core server? (Page 84 MLFQ, Page 97 (Ch 10))

Parallel programming is a form of computation in which program instructions are divided among multiple processors in combination to solve a single problem. It increases runtime performance, but can be susceptible to race conditions, mutual exclusion, deadlock, and synchronization.

- **Bit-level** - Increase the word size of the processor. It reduces the number of instructions the processor must execute in order to perform an operation on variables while sizes are greater than the length of the word.
- **Instruction-Level** - Measure of how many of the instructions in a computer program which can be executed simultaneously. Both hardware (dynamic) and software (static) based.
- **Data-Level** - Distribute the data across different nodes/cores which operate on the data in parallel. Single instruction Multiple Data OR Multiple Instruction Multiple Data. Use vector processors
- **Task Parallelism** - Distribute tasks (concurrently performed by processes or threads across different processors. Run many different tasks on the same data. A common type is pipelining which is moving a single set of data through separate tasks.

http://web.mit.edu/6.005/www/fa14/classes/17-concurrency/#two_models_for_concurrent_programming

At what level (OS, app, etc) can we can do parallelization (to take advantage of multicore) and which one is more promising and why?

- Parallelization can occur at any level. The OS can simulate a multi core processor through the use of multi threading by distributing processes across threads on the same core. Parallelization can be done most efficiently and effectively at the hardware level because it is not simulating multiple cores, it actually has multiple cores on which it can actually run parallel processes.
- Application - On the application level, we can divide up code into threads so they can utilize multiple cores on a processor. Use user level libraries, ie pthread, condition variables, etc. to make robust multithreaded applications.

- There are also certain architectures we can adhere to. One such is event driven parallelism. We have one thread (event loop) that is responsible for receiving events.
- write about video games (you have a multicast server)
- OS - We can utilize different methods of parallelism, ie Instruction level parallelism.
- Multiprocessor scheduling is also something that the OS can leverage for parallelization. The OS can also schedule load balancing, or in other words monitoring the workload of each core and transferring tasks to and from.
- On a lower level (pipeline), you can use the hardware or have more hardware, ie vector processing architecture to do the same instruction on multiple data in parallel (SIMD). Similarly, you can do multiple instructions on multiple data in parallel.

Question 2 - TLB Misses and Page Faults and VM Migration

What techniques OS can help with minimize TLB misses and their impact on page faults (Page 183) If the bits are missing, what are the page fault?

TLB is a memory cache that stores recent translations of virtual memory to physical addresses. A page fault is an exception raised by computer hardware when a running program The cost of a TLB miss It's main purpose is for faster retrieval. If there is a TLB miss, we then go to the page table to look for the corresponding page frame number. If there is a hit, we do not have to go to the page table.

http://www.ece.ucsb.edu/~strukov/ece154aFall2012/lecture16.pdf

In the case of a miss

1. If the page is loaded into main memory, then TLB miss can be handled by loading translation info from page table to TLB
2. If the page is not loaded in main memory, then this is a true page fault. Something to realize however, is that TLB misses are more common than true page faults

Cache affinity - When a process runs on a CPU, it builds up a fair bit of state in both the CPU and the TLB. When you want to rerun the process, it would be better to run it on the same CPU so you can used what was saved beforehand. As a result, a multiprocessor should take cache affinity into account when scheduling a process.

What problems you will encounter if you try to migrate a VM while it is running?

Live migration refers to the process of moving a running virtual machine or application between different physical machines without disconnecting the client or application. Memory, storage, and network connectivity of the VM are transferred from original to destination.

- **Warm-up Phase**
  - The Hypervisor (manages the VMs) copies all memory pages from source to destination while VM is still running (dirty pages retransmitted)
- **Stop-and-copy Phase**
  - After the warmup phase, the VM will be stopped on the original host. Remaining dirty pages will be copied to destination and VM resumed on destination host
  - Downtime - the time between stopping the VM on the original host and resuming it on destination. Downtime is determined according to size of memory and applications running on the VM
- **Post-copy memory migration**
  - Suspend the VM at the source
  - A minimal subset of execution state of VM (CPU state, registers, non-pageable memory) transferred to target
  - VM then resumed at target.

The problems you encounter

- Memory pages changing during the process - solve by recopying the pages at the rate where the recopying rate less than the page dirtying rate
- Knowing if the destination has the amount of resources to support once everything is copied over
- Maintaining state, ie if your VM is a game server with multiple clients, you want to preserve that connected state instead of having them reconnect
- Tradeoff between down time and migration time. You want to minimize downtime (time a VM is down) as well as migration time (the total time it takes to transfer).
- Dealing with the connection itself when transferring the contents
- Service degradation - Doing the actually copying may starve resources for the actual task, which may cause some bad performance

Question 3 - Interrupts

http://www.cs.iit.edu/~cs561/cs450/ChilkuriDineshThreads/dinesh%27s%20files/

User%20and%20Kernel%20Level%20Threads.html

Can disabling interrupts handle concurrency correct? (Page 294)

- **What is an interrupt**
  - Uses Unix signals to inform applicants when there is an event that needs immediate attention. They were created as a mechanism to eliminate unproductive waiting time in polling loops.
  - An interrupt is a signal to the processor emitted by hardware/software (in our case software).
  - They are called when you want to do a kernel-specific function.
  - Interrupts are mediated by the processor and handled by the kernel.
  - Hardware interrupts were actually introduced as an optimization eliminating unproductive waiting time in polling loops, waiting for external events.

It is possible to handle concurrency by disabling interrupts. By turning off interrupts, we can insure that code in the critical section will not be executed and execute atomically.

- Pros
  - The main advantage is that it is a simple solution
  - Without interruption, you can control what access the critical section
- Cons
  - Requires too much trust in applications, in fact this requires us to allow any calling thread to perform a privileged operation (turning interrupts on and off).
  - Greedy applications may hog resources and monopolize the processor, ie call lock whenever
  - Only works for single processor systems

All these are reasons why turning off interrupts is used in limited contexts as a mutual-exclusion primitive. This makes sense because the trust issue disappears inside the OS.

What are mutexes and conditional variables and how are they used?

https://msdn.microsoft.com/en-us/library/windows/desktop/ms682052(v=vs.85).aspx
https://msdn.microsoft.com/en-us/library/windows/desktop/ms684266(v=vs.85).aspx
https://stackoverflow.com/questions/4742196/advantages-of-using-condition-variables-over-mutex

Mutex object is a synchronized object whose state is set to signal when it is not owned by any thread and nonsignalfed when owned. Only one thread is allowed in a mutex at any given time.
ie **pthread_mutex_lock()**

Conditional variables are synchronization primitives which enable threads to wait until a particular condition occurs.
ie **pthread_cond_signal()**

A condition variable allows a thread to be signal when something of interest to that thread occurs. On the other hand, a mutex does not do this. If you need mutual exclusion, then use a mutex, if you need to notify a thread that a certain event has occurred, then use a condition signal. Both of these primitives are necessary and widely used to create robust multithreading applications.

Question 4 - Authentication

There are multiple ways for authentication. Which way is best?

The best authentication should have a good balance between security (the right person being authenticated) and convenience. For example, a 4-digit passcode is convenient because it is easy to remember, but it could be something which can be easily brute forced. On the other hand, using a physical security key to authenticate may be secure, but also is inconvenient as you have to carry it around with you. New technologies such as face id and fingerprint id are great because they are simple to use, establish authenticity, and hard to hack into. Another issue is that a secure authentication scheme does not always guarantee identity.

What do you think combining some of these techniques to provide stronger authentication?

2 factor authentication confirms someone's identity by using two different methods of online ID. You are going to have security at the cost of convenience. As long as both mechanisms are somewhat doable, ie enter your password and put in a code you get from text.

- **Pros** (Assuming a 2 factor authentication)
  - Establishes identity
  - Harder to exploit
- **Cons**
  - What if someone is disabled? Or what if someone loses their security key?
  - What happens if someone loses their phone?

- What happens if 1 out of 2 methods of authentication are compromised?

Question 5 - Linux

Few questions about Linux - Shell programming and pipeline. Make sure you understand what is Zombie process? How many modes can you ask for a lock on a resource? And when does a user ask for each mode?

Shell Programming and Pipeline

https://www.geeksforgeeks.org/fork-system-call/

- fork() - forks a new process. If the pid is 0, then it is a child. Starts at the exact same place as the parent. A child process use same pc(program counter), same CPU registers, same open files which use in parent process.
- wait() - waits for the child process to finish executing before proceeding
- execvp() - replaces the current running process with a new program (whereas fork just creates a new child process that runs concurrently)

Zombie Process

https://www.geeksforgeeks.org/zombie-and-orphan-processes-in-c/

A zombie process is a process which has completed execution (exit) system call. (SIGTERM SIGNAL) geeks for geeks However, it still has an entry in the process table, but instead is in the "terminated state". This occurs for child processes because the entry is still needed to allow parent process to read its child's exit status. After a zombie is removed, its PID and entry can be reused. However, if the parent process does not call "wait", then the zombie could be left in the process table, causing a resource leak.

How many modes can you ask for a lock on a resource?

There are 3 types of modes - shared, exclusive, and update

- Shared - Used for read operations that do not change or update data
- Exclusive - Used for data modification operations. Does this atomically
- Update - Used on resources that can be updated. Prevents a common form of deadlock that occurs when multiple sessions are reading, locking, and potentially updating resources later.

When does a user ask for each mode?

A user asks for each mode depending on what he/she wants to do with the data.

What does a | b | c | & d | e | f & do?

- 2 processes in the background (c and f)
- result of a gets piped to b which gets piped to c
- result of d gets piped to e which gets piped to f

- Total 6 processes are complete

Question 6 - Study Hadoop

Is Hadoop better for interactive or batch processing jobs? What is the normal topology of hadoop

Hadoop is better for batch processing jobs. Batching is a model where a set of data is collected over time, then fed into an analytics system. In other words, you collect all the data, then send it in for processing.

The Hadoop MapReduce model involves reading a batch of data from file, processing it, then writing it back to file. You want to minimize the number of I/O operations, which means that you are better off doing it batch instead of interactive. Since Hadoop 2.0, there is now compatible of stream processing (interactive).

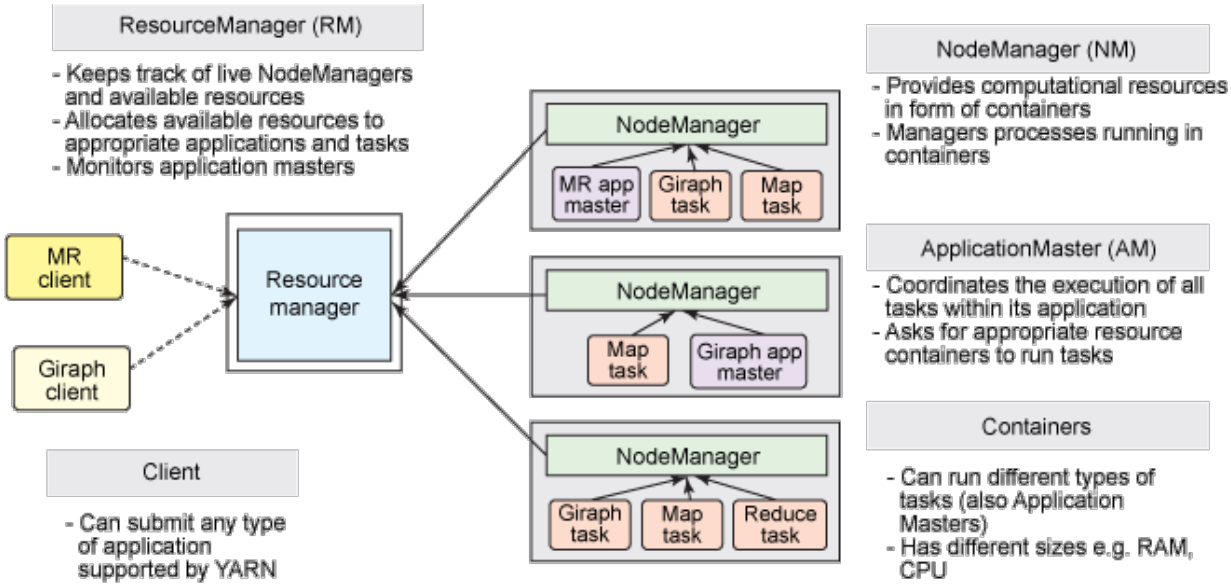
Batch = multiple inputs. Batch processing involves storing the data over a period of time and then processed using Hadoop.
Interactive = single, one by one

What is YARN?

http://searchdatamanagement.techtarget.com/definition/Apache-Hadoop-YARN-Yet-Another-Resource-Negotiator
Yarn stands for "Yet another resource navigator". It is essentially a large scale, distributed operating system for big data applications. It essentially decouples MapReduce's resource management and scheduling capabilities from the data processing component. What this does is to enable Hadoop to support more varied processing approaches to a broader array of applications. For example, Hadoop can now support interactive querying and streaming.

https://www.linkedin.com/pulse/hadoop-10-vs-20-parvathi-k/
YARN splits up the two major functionalities of overburdened JobTracker (resource management and job scheduling/monitoring) into two separate daemons: A global resource manager and a per-application Master
The RM focuses on manages cluster resources and the application manager manages each running application (MapReduce). With Yarn, you can now run multiple applications on Hadoop, all sharing a common resource. There is a NodeManager instead of a Task Tracker. There is a distributed application instead of a MapReduce job.

https://www.ibm.com/developerworks/library/bd-yarn-intro/
In the YARN architecture, a global ResourceManger runs as a master daemon on a dedicated machine. The resource manager tracks how many live nodes and resources are available on the cluster and coordinates what applications submitted by users should get these resources and when. An application manager which also coordinates all the tasks within its application



https://www.quora.com/What-is-YARN-in-Hadoop
It enables Hadoop to support more varied processing approaches and a broader array of applications.
The central resource manager as well as the Node manager manage the state of applications.
The separation of HDFS from MapReduce with YARN has made Hadoop more suitable for operational applications that can't wait for batch jobs to finish.

How YARN runs

- Client submits app to RM
- RM allocates container
- AppMaster contacts NM
- NM launches container
- Container executes AppMaster

Differences between Hadoop 1.0 and Hadoop 2.0

http://saphanatutorial.com/hadoop-1-0-vs-hadoop-2-0/

- You can run non MapReduce applications in Hadoop 2.0

    - Because YARN decouples the resource management element from the data processing element, you can run different applications on Hadoop 2.0
- Hadoop 2.0 has YARN which allows for this
- There is improved resource utilization
    - Previously there was a JobTracker in MapReduce 1
    - Now we have a global **Resource Manager** and each application has an **Application Manager**
        - Resource Manager focuses on managing cluster resources
        - Application Manager focuses on each application (ie a MapReduce job)