Distributed Systems

Operating systems

Rik Sarkar

University of Edinburgh
Fall 2014
Termination detection

• How do we know when a distributed computation has ended?

• We track nodes being in state “idle” Vs “Active”

• Assume: an idle node becomes active only on receiving a message from some other node.
  – (exception : the initiator: leader/server etc..)

• Termination is all nodes being idle

Ref: Wiki, VG
Termination detection (weight throwing)

• We suppose that the computation is started by a process s.
  – This means, other (idle) processes start working (becomes active) after receiving message from s or some other process
  – They have no other way to know that a computation is in progress
• s wants to know when all other processes have concluded working
• S starts with weight = 1.0
• Other processes start with weight = 0
• When a process sends a message, it puts part (say, half) of its weight in the message.
• When a process receives a message, it adds the message weight to its own weight.
• When a process has finished computing, (becomes idle) it sends its current weight to s
• When s has weight=1.0, it knows no other process is active
Termination detection (weight throwing)

• Works on the assumption that no message is lost
  – Methods like TCP give good guarantee for delivery
  – Many other distributed algorithms have this assumption
  – Useful for their termination detection

• Drawback:
  – What if there are many messages?
  – (Homework!)
Termination detection (Dijkstra-scholten)

- Maintains a tree of which node initiated computation at which other node
- Each node has active children counter (cc)
- When node x sends a message to y
  - x increments cc
  - If y was idle
    - y becomes active
    - y remembers x as the parent
  - If y was already active
    - y sends ack to x
- When x receives an ack
  - x decrements cc
- When y finishes all computation and is idle
  - And has cc = 0
    - y sends ack to parent
Termination detection (Dijkstra-scholten)

• How do you describe its Message complexity?
Operating System

• How different operating system issues relate to distributed system design
Operating System

Ref: CDK

• What is an operating system?
• An operating system is a resource manager
• It provides an abstract computing interface to processes
  – A program (and the programmer) does not need to know the details of the hardware
  – It asks the operating system to have something done, the OS gets it done by the hardware
  – Eg. You don’t need to know what modem or LAN card is being used to write a network based program
    • Ask the OS “please send message m to IP address x”
    • OS has “drivers” for the network interface to get the job done
Operating System

- What is an operating system?
- An operating system is a resource manager
- It provides an abstract computing interface to processes
- OS arbitrates resource usage between processes
  - CPU
  - Memory, filesystem
  - Network
  - Keyboard, mouse, monitor
  - Other hardware
- This makes it possible to have multiple processes in the same system
  - If 2 processes ask for use of same resource
  - OS decides who gets is when, how much etc
Operating System

• How OS handles different resources
• Memory:
  – Each process is given a different part of memory to use, they cannot access other’s memory
  – If it needs more memory, OS will allocate from unallocated memory store
• Filesystem
  – OS checks that process has rights to read/write the file
  – Makes sure that 2 processes are not writing the same file
• Network:
  – OS receives messages from processes, sends them to network card one at a time
  – When messages are received, OS delivers to suitable processes
Operating System

• How OS handles different resources
• Keyboard/mouse:
  – User types/clicks. Which application should get it?
  – OS decides
• Apps want to display things on screen.
  – OS decides when/where display will occur
• CPU: the most basic resource
  – Each process runs for a short period, and the control returns to OS
  – OS selects the process to run for the next slice
Operating System

• Hardware is designed so that OS can enforce these actions. E.g.:
  • CPU has kernel mode and user mode
    – Certain commands can only be used in kernel mode

• Memory:
  – Process X thinks it is using memory from 0000 to 1000
  – Actually, it is using 40050000 to 40051000
  – The 4005 is loaded into first part of the memory address register when the process starts executing
  – Process has no way to know or modify it
Operating System

• OS makes processes *oblivious* of environment
• Process does not know details of hardware
• Process does not know about other processes (unless they communicate with each-other)
Threads

• Threads are processes inside a process!
• They have access to the same memory space
• So communication between threads is easier
• Threads need more or less the same information as the process itself, so switching execution between threads is less work for the OS
  – Lightweight context switch
Threads

• Use of threads:
  – Imagine a server interacting with many clients
  – A separate thread per client makes it easier to write a program that works with many clients
  – Suppose client 1 is slow, and client 2 works faster
  – When thread 1 is waiting for client 1 to respond, thread 2 can continue working for client 2
Networked OS (any standard OS)

• A networked OS is aware that it is connected to the network
• Every node has an OS running
• Every node manages the resources at that node
• A process can request communication to processes in other nodes
  – It has to be explicitly aware that it is requesting service at at different node
  – And which node it is requesting (eg. I.P. address)
  – So it also has to know which services/resources are available in the network
• A process cannot request resources in control of a different computer
• It has to communicate with a process on that computer and request it to do the job
• Distributed computing has to be done explicitly
Distributed OS

- The OSes running on the different computers act like a single OS
- A process does not get to know (or need to know) that other resources/processes are at other computers
- E.g.:
  - Process gets input/output from hardware X, which can be on any computer
  - Process A communicates with process B the same way whether they are on same computer or not
  - OS takes care of using the network if needed
- A process may be running on a different computer from where it was started. Processes can be moved among different computers
- The “distributed” nature of the system is hidden from the processes
- The OS manages all the “distributed” aspects
Distributed OS

• One interface to all resources in the network

• Regular program can be made to run in a distributed fashion

• Easier to program applications that make use of networked resources

• Or is it?
Problems with distributed OS

• What happens if part of the network fails, and processes are separated into 2 sets?
  – Now we have to tell processes that the network has failed, and process has to take action
  – What if some OS-processes were moved elsewhere?

• Suppose we start processes A and B on the same computer
  – OS moves them to different computers
  – But A and B communicate a lot, so it would have been efficient to have them on the same computer!
Problems with distributed OS

• Access to offsite resources
  – Has to be through explicit network connection
  – All computers in the world cannot be in same system!

• Adding new nodes to a distributed computing
  – May be part of a different instance of the OS
  – We will still need explicit connections

• Distributed OS does not help a lot with distributed computing
Problems with distributed OS

• A network/computer failure means part of the OS failed
  – Hard to design OS with tolerance to such failures
• Distributed OS has to allow for lots of different possibilities in distributed computing
  – Harder to design
  – In fact, it is not possible to allow for all different possibilities
• “Distributed computing” means different things in different cases
• Better to let the application programmer decide how it will be distributed, and how to handle communication, failure etc
• OS provides only the basic infrastructure
Networked OS vs Distributed OS

- As a result, we do not have any distributed OS in regular use
- Networked OS are popular
- Provide communication facilities
- Let software decide how they want to execute distributed computation
  - More flexibility
  - Failure etc are application’s responsibility
  - OS continues to do basic tasks
Distributed computation in Networked OS

• Use distributed algorithms at the application layer for
  – Synchronization
  – Consistent ordering
  – Mutual Exclusion
  – Leader election
  – Failure detection
  – Multicast
  – Etc..

• And design distributed computing applications

• Different applications will need different sets of features
Virtualization

- A virtual machine runs as an application on a computer
- It *emulates* the hardware of a computer
- It is possible to run an operating system in a virtual machine
  - The VM application takes the OS executable as input
  - It then meticulously executes the steps a real computer would have taken
  - But does this in an application environment
  - That is, instead of a real CPU, the VM has a data structure representing a CPU
  - It then modifies the variables in the data structure exactly the way the registers of a CPU would have changed when executing those instructions
  - Same with memory, hard drive, network card etc
Virtualization

• When an application is run inside the “guest” OS running in the VM, the VM emulates the process of the OS as well as the application
Virtualization

• Useful for sandboxing, testing, backup
• Suppose you have a new OS to test
• Or trying to add a new component to the OS, such as a new device driver
• Running on actual hardware and having it crash is a lot of hassle to manage, reboot etc
• VM gives a nice way to test
• Also, you don’t have to waste an entire machine just because you are playing with the OS!
Virtualization

• VM gives a nice way to test
• Easy to modify the executable code and run again
• Since everything is just variables in the VM’s memory, the VM can write all this to a file, which can be used to debug and find exactly what happened
• In general, VMs can store “snapshots” for analysis and backup
Virtualization

- VM gives a nice way to test
- Easy to modify the executable code and run again
- Since everything is just variables in the VM’s memory, the VM can write all this to a file, which can be used to debug and find exactly what happened
- In general, VMs can store “snapshots” for analysis and backup
Virtualization and distributed computing

• Consider a server farm
• Many different servers are running
• Instead of giving a physical server to each, many server farms consist of real servers running virtual machines
• For example, renting a server to host a web site is likely to give you a VM based server
Virtualization and distributed computing

• Advantages: more flexibility
  – Multiple VMs on same computer
    • Need fewer physical machines
  – Easier to turn on/off
  – Easier to backup
  – VMs can be moved from one computer to another while preserving state
    • Useful when the work load changes, some servers need more computation, others need less..
Virtualization and distributed computing

- This is *not* a good strategy for CPU intensive computation such a large data mining
- Because running a large computation in a virtual machine is inefficient
- However, many systems need computation running all the time, but not so intensively
- Virtualization is most useful when flexibility is critical
Virtualization and distributed computing

- Hardware -> OS -> VMapp -> VOS -> Vapp -> thread
Virtualization

• Server farms and clusters
• Cloud computing
• Dynamic resource usage
• Testing
Some current trends in Distributed computing

• Mobile
  – Heavily contested area
  – Adaptation to mobility
  – Harder to network when moving
  – Adaptation to low energy system
  – Different style of user interaction
  – Needs better synchronization across multiple mobile user devices
Some current trends in Distributed computing

• Sensors
  – For sensor networks
  – TinyOS, LiteOS, Contiki
  – Small, low power sensor devices
  – Needs efficient operation
  – Needs specialization to process and handle sensor data and related operations in place of application interface
Some current trends in Distributed computing

• Embedded systems
  – Computers all around us, in every device/machine
  – Needs OS and Distributed computing, since they need to communicate with each-other
  – Adaptation to low power, low resource environment
  – Has to run without supervision/interaction