Intro to Distributed Systems – 10-18-2022

* What is a distributed system?
  + A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system
  + Two Characteristics
    - Collection autonomous computing elements (nodes)
      * Work independently, can be hardware or software processes
    - Single coherent system
      * Autonomous nodes collaborate so users/application perceives it as a single system
* Collection of Autonomous Nodes
  + Independent behavior
    - Each node is autonomous and will thus have its own notion of time
    - No global clock
  + Collection of nodes
    - How to manage group membership?
    - How to know that you are indeed communication with an authorized nonmember?
  + Managing group membership
    - Open group
      * Any node is allowed to join the distributed system
    - Closed group
      * Only members of that group can communicate with each other
  + Concerning distributed system’s organization
    - Often organized as an overlay network
      * Virtual structure imposed over a physical network
    - A node is typically a software process equipped with a list of other processes it can directly send messages to
    - Message passing is then done through TCP/IP or UDP channels or higher-level RPC
  + Overlay Networks
    - Structured: each node has a well-defined set of neighbors with whom it can communicate
    - Unstructured: each node has references to other nodes randomly selected from the system
  + An overlay network should, in principle, always be connected
    - Meaning that between any two nodes there is always a path for which a message can be sent
* Coherent System
  + The collection of nodes as a whole operates the same, no matter where, when, and how interaction between a user and the system take place
  + The user generally can’t tell where computation takes place
* Coherent system Cont.
  + The computation being invisible to the user is referred to as distribution transparency
  + This raises the issue of individual failures of specific components
    - These failures are basically inevitable
    - Should try to hide the partial features that do not work properly
* DS and the Middleware
  + Provides the means for components of a single distributed application to communicate with each other, nut also to let different applications communicate
  + The middle ware can be compared to the OS of the distributed network
  + Offers services
    - Inter application communication
    - Security services
    - Accounting services
* Middleware Services: Communication
  + Remote procedure call (RPC):
    - Allows an application to invoke a function that is implemented and executed on remove computer as if it was locally available
  + Transactions
    - Support for executing multiple services in an all-or-nothing
      * The middleware makes sure that every service is invoked
* Middleware Services: Service Composition
  + Common to develop new applications by taking existing programs and gluing them together (web services)
  + Eg. Can aggregate data from different sources called mashups
  + Think of a travel site that does many different stages of planning in one website
    - Many services provided from one place
* Middleware Services: Reliability
* Design Goals for Distributed Systems
  + Resource sharing
    - Resources can be accessed by different users and applications on the system
    - Example: cloud-based sharing, multimedia streaming
  + Distribution transparency
    - Invisible to end users and applications
    - Degree of transparency
      * There are communication latencies that can’t be hidden
      * Completely hiding failures of networks and nodes is impossible
        + You can never tell a slow computer from a failing one
        + Can’t tell if an operation was performed prior to a crash
      * Exposing distribution may be good
        + Making use of location-based services (finding nearby friends)
        + When dealing with users in different time zones
        + Makes it easier for a user to understand what’s going one (say when a server isn’t responding for a while)
* Cont. Design Goals
  + Being open
    - Offers components that can easily be used by, or integrated into other systems
    - Systems should easily interoperate
    - Systems should support portability of applications
    - Systems should be easily extensible
    - Systems should conform well to interfaces
    - Implementing openness policies
      * What level of consistency do we require for client cached data?
      * Which operations do we allow to be downloaded code to perform?
      * What level of secrecy do we require for communication?
    - Implementing openness mechanisms
      * Allow dynamic setting of caching policies
      * Support different levels of trust for mobile code
      * Offer different encryption algorithms
    - Stricter the separation between policy and mechanism, more we need to ensure proper mechanisms
      * Potentially leading to many configuration parameters complex management
    - Finding a balance
      * Hard coding certain policies
  + Being scalable
    - At least three components
      * Number of users and/or processes
      * Maximum distance between nodes (geographical scalability)
      * Number of administrative domains
    - Scalability problems with centralized solutions
      * The computation capacity, limited by the CPU’s
      * The storage capacity, including the transfer rate between CPU’s and disks
      * The network between the user and the centralized service
    - Problems with geographical scalability
      * Cannot simply go from LAN to WAN
        + Many distributed systems assume synchronous client-server interactions

CLIENT sends requests and waits for answer

* + - Problems with administrative scalability
      * In essence, conflicting policies concerning usage, management, and security
      * Examples
        + Computational grids

Share expensive resources

Shared equipment: how to control, manage, and use

* Techniques for Scaling
  + Scaling up (improve capacity, upgrade CPUs, etc)
  + Scaling out (deploying more machines)
    - Hiding communication latencies
    - Partition and distribution of work and replication
    - Hiding latencies
      * Make use of asynchronous communication
    - More computation on the client side
  + Partitioning and distribution
    - More computation on the client side
  + Replication
    - Make copies of data available at different machines
      * Replicated files and servers
      * Mirrored web sites
      * Web caches
    - Applying replication is easy, except for one thing
      * Having multiple copies leads to inconsistencies
      * To keep copies consistent, require global synchronization
* Pitfalls of the DS development
  + False assumptions about the DS
    - The network is reliable
    - The network is secure
    - Latency is zero
    - Bandwidth is infinite
    - Transport cost is zero
    - There is only one admin
* Three types of DS
  + High performance distributed computing systems
    - Super computers
    - Weather prediction computers
  + Distributed information systems
  + Distributed systems for pervasive computing
    - Pervasive meaning among us, around us
* High performance distributed computing
  + Started with parallel computing
  + Shared memory vs private memory (in the slides)
  + Cluster computing
    - Essentially a group of high-end systems connected through a LAN
    - Usually, an access node or manager node
    - Homogenous: same OS
      * Near identical hardware
* Grid Computing
  + The next step: lots of nodes from everywhere
    - Heterogeneous
    - Dispersed across several organizations
    - Can easily span a wide area network
    - Example
      * Einstein@Home
* Cloud Computing
  + Providing the facilities to dynamically construct an infrastructure and compose what is needed from available services
  + Provides lots of resources
  + Ex) Microsoft Suite, iCloud, OneDrive

Thursday

* A light summary/example
  + Organization confronted with many networked apps, buy achieving interoperability was difficult
    - A networked app runs on a server making its services available to remote clients
* Distributed Information Systems
  + Middle ware offers communication solutions for integration
    - Remote procedure call
      * Requests are sent through local procedure call, packaged as message, and sent to the callee, and result returned as return from call
    - Message oriented middleware
      * Messages are sent to logical contact point and subscribed to applications
* Distributed Pervasive Systems
  + Emerging next generation of distributed systems in which nodes are small, mobile, and often embedded in a large system
  + System naturally blends into the user’s environment
  + Three overlapping subtypes
    - Ubiquitous computing systems: pervasive and cont. present
      * Usually, constant interaction with user
    - Mobile computing systems
      * Pervasive, but emphasis on the fact that the device is inherently mobile
    - Sensor networks
      * Pervasive with emphasis on the sensing and actuation of the environment
* DS: Communication
  + Types of communication
    - Persistent:
      * message is stored until its delivered to the receiver
      * Ex. Email
    - Transient:
      * message is stored while the sending and receiving application are executing
      * Ex, transport level services
    - Synchronous:
      * sending and receiving processes synchronize at request submission, delivery, or after processing by server
    - Asynchronous:
      * sender sends a message and then continues
      * This means that the message is temp stored immediately by the middleware upon submission
  + Diagram(s) in notebook
* Remote Procedure Call
  + Goal: allow programs to call procedures located on other machines
  + When a process on machine A calls a procedure on B, the calling process on A is suspended and execution of the called procedure takes place on B
  + Information can be transported from the caller to the calle in the parameters and can come back in the procedure
* Client and Server Stubs
  + RPC makes a remote call look as much as possible to a local call
  + Suppose a program has access to a data base that allows it to append data to a stored list, after which it returns a reference to the modified list
    - newList = append(data, dbList)
  + When append is a remote procedure, a different version of append, called client stub
    - Client stub will pack the parameters into a message and request to be sent to the server
    - (pack message, send it off, wait for result, unpack result and return to caller)
    - Diagram in notebook
* RPC: Passing parameters
  + Theres more than just wrapping parameters into a message
    - Client and server machines may have different data representations
      * Wrapping a parameter means transforming a value into a sequence of bytes
      * Client and server have to agree on the same encoding
  + External data representation
    - An agreed standard for the representation of data structures and primitive values
  + Marshalling
    - Is the process of taking a collection of data items and assembling them into a form suitable for transmission in a message
    - In the diagram, the marshalling happens at the client stub (vice versa upon response)
  + Unmarshalling
    - Is the process of disassembling hem on arrival to produce an equivalent collection of data items at the destination
    - The unmarshalling takes place at the server stud (vice versa upon the response)
  + Other approaches to external data representation and marshaling
    - Java’s object serialization: concerned with flattening and external data representation of any single object or tree of objects
    - XML defines a textual formation for representing data
* Variations on RPC
  + Conventional RPC: client will block until a reply is returned
    - This strict request reply behavior is unnecessary when there is not resulted to return
    - Or may hinder efficiency when multiple RPC’s need to be performed
* Synchronous RPC at receive
  + In essence:
    - Eliminate the strict request-reply behavior, but let the client continue without waiting for the result
* Multicast RPC’s
  + Sending an RPC request to a group of servers