Ch4: Distributed Systems Architectures

- Typically, system with several interconnected computers that do not share clock or memory.
- Motivation: tie together multiple independent computers over the network to
    - share resources, enhance performance, improve reliability/availability, provide expandability.
- One possible classification of architecture
    - several “minicomputers” (machines with CPU/user < 1)
    - several hundred / thousand “workstations” (CPU/user ~ 1)
    - Processor pool (CPU/user > 1)
        - user typically has a dedicated CPU as well.

- What is an OS
  - An interface between hardware and user processes that provides an abstraction of the machine and manages its resources.
- What is a Distributed OS
  - the same, except for distributed systems
  - aims at transparency of distribution and presents a virtual uniprocessor to the user.
  - according to some, the holy grail is to create a “metacomputer” and a “Problem Solving Environment”
    - The user sees a single machine that automagically provides enough resources to do the task. The task can range from simple hello world programs to complex calculations.

Issues in Distributed OS

- Global “state” is not known
  - due to lack of shared memory and clock, unreliable message transmission.
  - Need decentralized controls
  - temporal order of events ?
- Naming
  - how should an object be identified ?
    - Transparent ? Translucent ? Explicit ?
    - URIs, URNs, URLs
  - what if the object is replicated
- Scalability
  - system should continue to work efficiently as resources are added
    - consider a system that resolves IP addresses using broadcast

- Compatibility / Interoperability
    - binary level, execution level or protocol level
    - homogeneous vs heterogeneous systems
- Process Synchronization
    - mutual exclusion problem w/o shared memory
- Resource Management
    - how do you get data to the location of the computation
        - distributed filesystem ? distributed shared memory ?
    - how do you migrate a computation (remote evaluation)
        - RPC ? Client-Server ?
    - how do you migrate running processes (code on demand, mobile objects/agents)
- Security
    - authentication and authorization in a distributed system

Structure of Distributed OS

- Monolithic kernel
  - a (large) single entity that provides all the services of the distributed OS
  - may not be a good idea since “computer configurations” will vary
    - do you need to load disk drivers on a diskless client ?
- Collective kernel
  - base OS functionality is in a relatively small *microkernel*. All other OS services are processes that run on top.
  - Microkernel will run on all machines
- OO approach
  - same as collective kernel, but OS services are implemented as objects
  - can provide a more structured approach than collective kernel

Communication

- Review Section 4.6 yourself.
- Message Passing Model
    - send and receive type primitives
    - can be blocking(reliable, unreliable) or non-blocking
    - use of buffers
    - synchronous vs asynchronous

Millennium Project

- Work at Microsoft Research(paper by Bolosky et al.)
- Features
  - seamless distribution, worldwide scalability, fault tolerance, self-tuning, self configuring, secure, resource control
- Principles
  - aggressive abstraction, storage irrelevance, location irrelevance, JIT binding, introspection.