Distributed Operating Systems: A Survey

by Tanenbaum and Van Renesse, ACM Computing Surveys, 1985

- What is a distributed system?
  - "looks like an ordinary centralized O.S. but runs on multiple independent CPUs"
  - virtual uniprocessor
  - software
- Differences with a network operating system
- Goals
  - transparency
  - more power, cheaper
  - fault tolerance
- Problems
  - Lacks global state info

Network Operating Systems

- Visibility: how aware are the users of the environment
  - file system
    mounting, export, transparency,
    global *vs.* local name space
    example: SUN's nfs
  - protection
    i-nodes and UIDs
  - process execution location
- Hybrid forms

Design Issues

- Communication Primitives
- Naming and protection
- Resource management
- Fault tolerance
- Services

Communication Primitives

- Semaphores and monitors do not work
- Message passing
  - Different models
  - ISO OSI (7 layer model), not very successful, due to overhead
  - TCP/IP model
    - five relatively independent layers
    - application layer
    - host-to-host (transport)
    - Internet layer (IP)
    - network access layer
    - physical layer
    - applications: SMTP, ftp, telnet...

Message passing (cont.)

- Client/Server
  - client and server “cooperate” in various degrees
  - e.g., client: windows interface, server: database
- Distributed Message Passing
  - SEND and RECEIVE primitives
  - synchronous *vs.* asynchronous
  - blocking *vs.* non-blocking
  - reliability *vs.* unreliability
  - unbuffered (i.e., rendez-vous) *vs.* buffered

Remote Procedure Call (RPC) [Birrell and Nelson, ’84]

- Refinement of reliable, blocking message passing: “Client sends request and blocks until sender sends reply”
- Idea: make semantics of inter-machine communication as similar as possible to normal procedure calls
- Syntax:
  - dummy or *stub* procedure must be included at the caller’s address space
  - stub collects parameters and packs them in a message in standard format
  - *stub* program at the remote machine collects the message, unpacks the parameters
  - makes the local call
  - only the *stub* knows that the procedure is remote

Remote Procedure Call (RPC) (cont.)

- Design Issues
  - implementation
  - passing reference parameters
  - parameter and result representation in messages
  - translation issues in heterogeneous machines
  - in some applications, details (like which specific server) are important, but intentionally hidden by RPC
- Lately, synchronous/asynchronous RPCs

Error Handling

- Server crash/client crash
- Idempotent calls: those that can be repeated without harm
- Taxonomies for semantics of parallel systems by Spector and Nelson
  (exact one remote execution, at least one, etc)

Implementation Issues

- Efficient message passing
  network speeds, message sizes
- Avoid copying
- Minimize overheads
- Virtual circuits versus datagrams
  (reliability issues)

Resource Management

- Differences with centralized systems
  - no central status tables
  - reliability reasons
- Difficult
- Processor allocation
  - workers/managers model
  - hierarchy
  - recursive allocation (upwards)
  - reliability and self-repairs

Scheduling

- Deal with heterogeneity of hardware
- Fork-join model of parallel software
- Time-sharing not straight forward anymore...
- Potential problems
- Ousterhout's co-scheduling (1982), later
  process-control policies
- Main idea: allocation matrix
  - packing schemes
  - sliding window
  - more time-slices?

Load Balancing

- Goals: maximize throughput, minimize response time, keep load uniform
- In conflict...
- Graph theoretic models
  reduce to the problem of partitioning a graph into k disjoint subgraphs
  find cuts that minimize network traffic
  results are somewhat academic...
- Heuristic load balancing
  idea: each processor estimate its balance continually and exchanges this info with other processors
- Process migration
  not an easy problem...

Distributed Deadlock Detection

- Resource deadlock (traditional)
- Harder problem in distributed systems because of lack of central tables
- Communication deadlock (circular wait)
- Chandy-Mishra-Haas algorithm for detection of communication deadlocks
- Idea: when a process blocks it sends a query message to the process(es) that might send it a real message. If one of the processes is blocked, it sends query messages to the processes it is waiting for. If certain messages circle back to the original process, then there is a deadlock.
- Looks for a knot in a directed graph

Fault Tolerance

- Reliability and availability issues
- Hardware *vs.* software failures
- Redundancy techniques
  - checkpoint processes (to revive them after crashes)
  - widely used in parallel disks (multiple RAID levels)
  - diagnosability
- Atomic transactions
  - atomic updates (or run-to-completion)
  - serialization unavoidable
  - not efficient
  - example from parallel file systems...
- Lampson's hierarchy of abstractions
  - stable disks (redundancy) and stable processors (checkpointing)

Services...

- File system service
  - concurrency control
  - robust, atomicity
  - naming
  - garbage collection
  - protection
  - performance (caching/prefetching)
  - reliability (recoverable files)
- Other services: time, terminal, boot...

Case study: Cambridge Distributed Computing System

- Communication primitives
  - ring of fixed sized (5 bytes) slots for datagram service
  - (source, destination, data1, data2, flags)
  - can have bigger slots for virtual circuit service
- Naming/Protection
  - single name server
  - to use service, the user asks the name server who asks access rights and grants the appropriate access and service location number.
- Resource management
  - single processor resource management service
  - selects the best place to run

Cambridge (cont.)

- Fault tolerance
  - to correct a fault: name server sends “reboot” packet, processor reboots, sends request to “boot server” who goes to name server to find out the appropriate service to download from the resource manager.
- Services
  - name server, resource manager, time server, print server, login server, terminal server, error server, file server

Case Study: Amoeba

- Communication primitives
  - message passing
  - RPC - user: SEND-GET, blocks until response
    server: GET-REQUEST then SEND-REPLY
  - unbuffered (datagram)
- Naming/Protection
  - capabilities based: each object has a global unique name in its capabilities
  - collection of objects named by capabilities, on which users perform operations
  - capability: owner, ID # (i-node#), rights, check field

Amoeba (cont.)

- Resource management
  - uses capabilities
  - each machine runs a resource manager which “makes processes” to send RPC call to get what’s requested
- Fault tolerance
  - boot server polls, if you don’t respond, a new copy of the server is brought up on one of the pool servers
- Services
  - block, file directory, free universal storage system, boot

The V Kernel

- Communication primitives
  - back-plane model
  - message based
    client: SEND(message, dest), then blocks until it gets reply
    server: RECEIVE, REPLY
  - nearly RPC
- Naming/Protection
  - context-based
  - 3 levels of names (PID, service string, object name)
  - each server has its own context (no name server)

The V Kernel(cont.)

- Resource management
  - no dynamic CPU allocation
  - each processor has same kernel
  - kernel manages processors+memory+I/O
  - can have process teams that share space
- Fault tolerance
  - very little – "try again" by user
  - only address exception handling by exception server
- Services
  - central file server

Eden

- Communication primitives
  - RPC invocation
  - for programs written in EPL, the compiler sets up the correct RPC stubs
- Naming/Protection
  - capability based
  - data is within objects, processes within objects
  - capability primarily stored in directories

Eden (cont.)

- Resource management
  - objects reside on workstations where created
  - objects have multiple processes: share the address space
  - monitors are used for synchronization
  - each object has own "heap" for storage management
- Fault tolerance
  - no atomic transactions
  - can return "cannot locate object"
  - objects checkpoint themselves
  - multiple file copies on multiple machines
- Services: replicated file system, mail server...