Silberschatz, et al.
Topics based on Chapter 15

Distributed Communication

Distributed System Structures

- General structure of distributed systems
  - Network Operating Systems
  - Distributed Operating Systems
  - Remote Services
- Robustness
- Design issues

Network Operating Systems

- Multiplicity of machines is visible to users
- Explicit access to remote resources; alternatives:
  - logging into remote machine
    - examples: telnet, rlogin, rsh
  - transferring data to local machine
    - examples: ftp, rcp

Distributed Operating Systems

- Multiplicity of machines hidden from users
- Access to remote resources uses similar commands as access to local resources
- Data and process may migration between sites, under the control of the distributed operating system
  - Data migration: transfer data by transferring entire file or transferring only those portions of the file necessary for the immediate task
  - Process migration (computation migration): transfer the computation, rather than the data

Distributed Operating Systems: Data Migration

- Accessing data from site A that resides on site B
  - Transfer complete file from B to A; transfer back on modification
    - "automatic FTP"
    - older versions of Andrew file system
    - Inefficient
  - Transfer portions of file that are necessary for the immediate task
    - NFS
    - newer versions of Andrew file system
  - Issues: amount of information that is needed; concurrent access to same portion of file

Distributed Operating Systems: Process Migration

- Process Migration – execute an entire process, or parts of it, at different sites.
  - Load balancing – distribute processes across network to even the workload.
  - Computation speedup – subprocesses can run concurrently on different sites.
  - Hardware preference – process execution may require specialized processor.
  - Software preference – required software may be available at only a particular site.
  - Data access – run process remotely, rather than transfer all data locally.

Distributed Operating Systems: Process Migration

- Migration without requiring user input
  - program does not need to be coded for migration
  - often used for load balancing and computational speedup among homogeneous systems
- Migration, specified by user
  - movement to satisfy hardware or software preference

Remote Services

- Requests for data at another site are expressed as requests for services from the remote site
- Requests are transferred to a remote server, which accesses necessary data, computes desired results, transferring them back to the requester.

Remote Services

- Remote Procedure Call (RPC)
  - One of most common forms of remote service; abstracts procedure-call mechanism
  - Messages are addressed to an RPC daemon listening to a *port* on the remote system.
  - Messages include the name of the process to run and the parameters to pass to that process.
  - Process is executed as requested and any output is sent back to the requester in a separate message.
  - Port: a number included at the start of a message package that is used to differentiate services located at the system.
  - Many ports, one network address

Remote Services: Remote Procedure Calls

- RPC semantics are not *precisely* the same as local procedure calls
  - local calls fail rarely; remote calls can fail (or be duplicated) because of network errors
    - timestamps needed to keep track of what has been processed
  - binding of formal and actual (i.e., client and server port) not as simple as local because client and server do not share memory
    - advance agreement on port addresses at RPC procedure compile time; can't change number once compiled
    - dynamic binding via a rendezvous dameon on a well-known RPC port (client sends message requesting port address of the RPC it needs to execute)

Remote Services: Remote Procedure Calls

- A distributed file system (DFS) can be implemented as a set of RPC daemons and clients.
  - The messages are addressed to the DFS port on a server on which a file operation is to take place.
  - The message contains the disk operation to be performed (i.e., **read**, **write**, **rename**, **delete**,or **status**).
  - The return message contains any data resulting from that call, which is executed by the DFS daemon on behalf of the client.

Remote Services: Threads

- Threads can send and receive messages while other operations within the task continue asynchronously
- Pop-up thread – created on “as needed” basis to respond to new RPC.
  - Cheaper to start new thread than to restore existing one.
  - No threads block waiting for new work; no context has to be saved, or restored.
  - Incoming RPCs do not have to be copied to a buffer within a server thread.
- RPCs to processes on the same machine as the caller made more lightweight via shared memory between threads in different processes running on same machine.

Robustness

- To ensure that the system is robust, we must:
  - *Detect* failures.
    - link
    - site
  - *Reconfigure* the system so that computation may continue.
  - *Recover* when a site or a link is repaired.

Failure Detection: Handshaking Procedure

- At fixed intervals, sites A and B send each other an *I-am-up* message. If site A does not receive this message within a predetermined time period, it can assume that site B has failed, that the link between A and B has failed, or that the message from B has been lost.
- A can try to differentiate between the cases by sending B an *Are-you-up?* message.
- At the time site A sends the Are-you-up? message, it specifies a time interval during which it is willing to wait for the reply from B. If A does not receive B's reply message within the time interval, A may conclude that one or more of the following situations has occurred:
  - Site B is down.
  - The direct link (if one exists) from A to B is down.
  - The alternative path from A to B is down.
  - The message has been lost.

Reconfiguration

- Procedure that allows the system to reconfigure and to continue its normal mode of operation.
- If a direct link from A to B has failed, this information must be broadcast to every site in the system, so that the various routing tables can be updated accordingly.
- If it is believed that a site has failed (because it can no longer be reached), then every site in the system must be so notified, so that they will no longer attempt to use the services of the failed site.

Recovery from failure

- When a failed link or site is repaired, it must be integrated into the system gracefully and smoothly.
- Suppose that a link between A and B has failed. When it is repaired, both A and B must be notified. We can accomplish this notification by continuously repeating the handshaking procedure.
- Suppose that site B has failed. When it recovers, it must notify all other sites that it is up again. Site B then may have to receive from the other sites various information to update its local tables.

Design Issues

- Transparency and locality – distributed system should look like conventional, centralized system and not distinguish between local and remote resources.
- User mobility – brings user's environment (i.e., home directory) to wherever the user logs in.
- Fault tolerance – system should continue functioning, perhaps in a degraded form, when faced with various types of failures.

Design Issues

- Scalability – system should adapt to increased service load.
- Large-scale systems – service demand from any system component should be bounded by a constant that is independent of the number of nodes.
- Servers' process structure – servers should operate efficiently in peak periods; use lightweight processes or threads.