Distributed Systems

- 13A Distributed Systems - Goals & Challenges
- 13B Distributed Systems - Communication
- 13D Distributed Systems - Synchronization
- 13C Distributed Systems - Security

Goals of Distributed Systems

- scalability and performance
  - apps require more resources than one computer has
  - grow system capacity /bandwidth to meet demand
- improved reliability and availability
  - 24x7 service despite disk/computer/software failures
- ease of use, with reduced operating expenses
  - centralized management of all services and systems
  - buy (better) services rather than computer equipment
- enable new collaboration and business models
  - collaborations that span system (or national) boundaries
  - a global free market for a wide range of new services

the end of self-contained systems

- authentication
  - Active Directory, LDAP, Kerberos, …
- configuration and control
  - Active Directory, LDAP, DHCP, CIM/WBEM, SNMP, …
- external data services
  - CIFS, NFS, Andrew, Amazon S3, …
- remote devices
  - X11, web user interfaces, network printers
- even power management, bootstrap, installation
  - vPro, PXE boot, bootp, live CDs, automatic s/w updates

Peter Deutsch's "Seven Falacies of Network Computing"

1. network is reliable
2. no latency (instant response time)
3. available bandwidth is infinite
4. network is secure
5. network topology & membership are stable
6. network admin is complete & consistent
7. cost of transporting additional data is zero

Bottom Line: true transparency is not achievable

Heterogenous Interoperability

- heterogenous clients
  - different instruction set architectures
  - different operating systems and versions
- heterogenous servers
  - different implementations
  - offered by competing service providers
- heterogenous networks
  - public and private
  - managed by different orgs in different countries

Fundmental Building Blocks Change

- the old model
  - programs run in processes
  - programs use APIs to access system resources
  - API services implemented by OS and libraries
- the new model
  - clients and servers run on nodes
  - clients use APIs to access services
  - API services are exchanged via protocols
- local is a (very important) special case

Performance, Scalability, Availability

- old model – better components (4-40%/yr)
  - find and optimize all avoidable overhead
  - get the OS to be as reliable as possible
  - run on the fastest and newest hardware
- new better – better systems (1000x)
  - add more $150 blades and a bigger switch
  - spreading the work over many nodes is a huge win
    - performance – linear with/number of blades
    - availability – service continues despite node failures

Changing Paradigms

- network connectivity becomes "a given"
  - new applications assume/exploit connectivity
  - new distributed programming paradigms emerge
  - new functionality depends on network services
- applications demand new kinds of services:
  - location independent operations
  - rendezvous between cooperating processes
  - WAN scale communication, synchronization

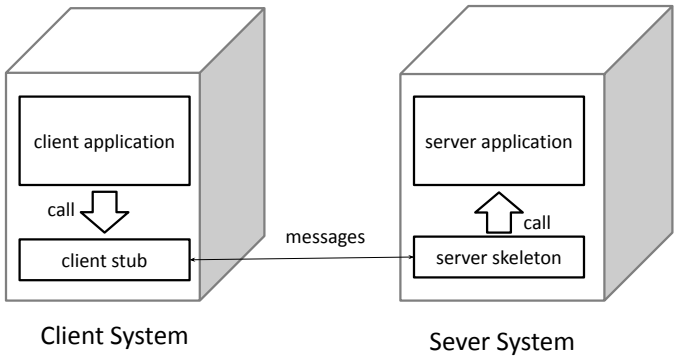
General Paradigm – RPC

- procedure calls – a fundamental paradigm
  - primary unit of computation in most languages
  - unit of information hiding in most methodologies
  - primary level of interface specification
- a natural boundary between client and server
  - turn procedure calls into message send/receives
- a few limitations
  - no implicit parameters/returns (e.g. globals)
  - no call-by-reference parameters
  - much slower than procedure calls (TANSTAAFL)

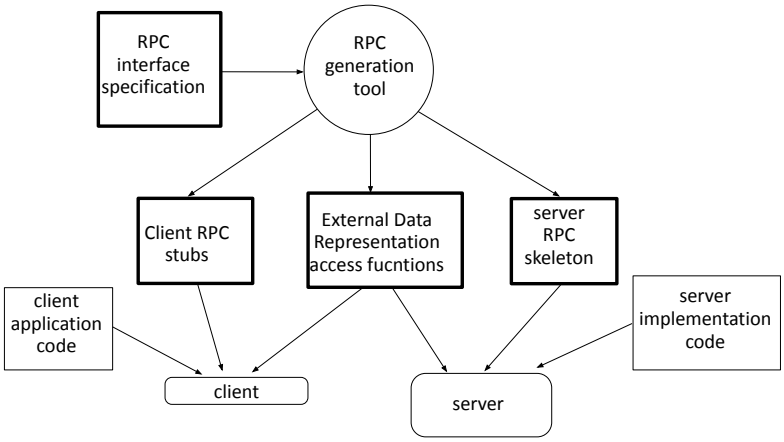
Remote Procedure Call Concepts

- Interface Specification
  - methods, parameter types, return types
- eXternal Data Representation
  - language/ISA independent data representations
  - may be abstract (e.g. XML) or efficient (binary)
- client stub
  - client-side proxy for a method in the API
- server stub (or skeleton)
  - server-side recipient for API invocations

Remote Procedure Calls – Data Flow



Remote Procedure Calls – Tool Chain



(RPC – Key Features)

- client application links against local procedures
  - calls local procedures, gets results
- all rpc implementation is inside those procedures
- client application does not know about RPC
  - does not know about formats of messages
  - does not worry about sends, timeouts, resents
  - does not know about external data representation
- all of this is generated automatically by RPC tools
- the key to the tools is the *interface specification*

The Interoperability Challenge

- S/W, APIs and protocols evolve
  - to embrace new requirements, functionality
- A single node is running a single OS release
  - all s/w can be upgraded at same time as OS
- A distributed system is unlikely homogenous
  - rolling upgrades do one server at a time
  - newly added servers may be up/down-rev
  - we may have no control over client s/w versions
- we must ensure they all "play well" together

Ensuring Interoperability

1. restricted evolution
   - all changes must be upwards compatible
2. compensation (run-time restriction)
   - all sessions begin with version negotiation
3. better tools that embrace polymorphism
   - every agent speaks his own protocol version
   - RPC language and tools are version-aware
     - messages are un-marshaled as each client expects
     - default behaviors are based on older expectations
   - equally applicable to messages and at-rest data

eXtensible Data Representations

- Upwards compatible serialized object formats
  - platform independent data representations
  - client-version sensitive translation
    - old clients never see new-version fields
    - new clients infer upwards compatible defaults
- Example: Google Protocol Buffers
  - very efficient translation
  - applicable to both protocols and persisted data
  - supports many representations (e.g. binary, json)
  - has adaptors for many languages (e.g. C, python)

Marshal (and un-marshal)

- English

  to arrange or assemble a group into order
  - usually a group of people or soldiers
  - also assembling *devices* into a *coat of arms*
- Computer Science

  transforming the in-memory representation of an object into a suitable format for storage or transmission

RPC is not a complete solution

- client/server binding model
  - expects to be given a live connection
- threading model implementation
  - a single thread service requests one-at-a-time
  - numerous one-per-request worker threads
- failure handling
  - client must arrange for timeout and recovery
- higher level abstractions
  - e.g. Microsoft DCOM, Java RMI, DRb, Pyro

Evolving Interaction Paradigms

- HTTP is becoming the preferred transport
  - well supported, tunnels through firewalls
- Simple Object Access Protocol (SOAP)
  - HTTP transport of (XML encoded) RPC requests
  - options for other transports and encodings
  - supports non-RPC interactions (e.g. transactions)
- REpresentational State Transfer (REST)
  - all resources are identified/accessed via a URI
    - client doesn't know where data/services are implemented
  - only operations are Create/Read/Update/Delete
    - stateless (server maintains no sessions) interactions
    - object states are cacheable

Sample SOAP Request

```
<?xml version="1.0"?>
 <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
   <soap:Header>
   </soap:Header>
   <soap:Body>
     <m:GetStockPrice xmlns:m="http://www.example.org/stock/Surya">
       <m:StockName>IBM</m:StockName>
     </m:GetStockPrice>
   </soap:Body>
</soap:Envelope>
```

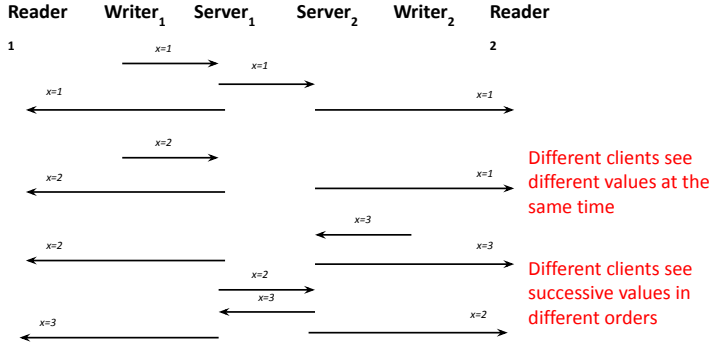

Sample REST (json) Request

```
{
  "username" : "my_username",
  "password" : "my_password",
  "validation-factors" : {
      "validationFactors" : [
        {
          "name" : "remote_address",
          "value" : "127.0.0.1"
        }
      ]
   }
}
```


Distributed Synchronization

- spatial separation
  - different processes run on different systems
  - no shared memory for (atomic instruction) locks
  - they are controlled by different operating systems
- temporal separation
  - can't "totally order" spatially separated events
  - before/simultaneous/after lose their meaning
- independent modes of failure
  - one partner can die, while others continue

Distributed Temporal Separation



1. The system does not have a scalar state. State is a vector.
2. There is no total ordering; There are only partial orderings.

Distributed Locking - Leases

- Synchronization must be centralized
  - a single server is responsible for issuing locks
  - traditional mechanisms can ensure atomicity
  - locks should be managed with message exchanges
- Authorization must be distributed
  - lock servers issue signed "cookies"
  - servers verify cookies before performing requests
- Client failures must be recoverable
  - locks automatically expire after lease time
  - automatic preemption prevents deadlock

Leases and Enforcement

- all requests are exchanged via messages
  - in general, all resources are on other nodes
  - client does not have direct access to resources
- each request includes a lease "cookie"
  - from resource manager (possibly signed)
  - identifies client, resource, and lease period
  - lease automatically expires at end of period
- validate cookies before performing operation
  - requests with *stale cookies* should be rejected
- handles a wide range of failures
  - process, client node, server node, network

Lock Breaking and Recovery

- revoking an expired lease is fairly easy
  - lease cookie includes a "good until" time
  - any operation involving a "stale cookie" fails
- this makes it safe to issue a new lease
  - old lease-holder can no longer access object
  - was object left in a "reasonable" state?
- object must be restored to last "good" state
  - roll back to state prior to the aborted lease
  - implement all-or-none transactions

How does the OS ensure security?

- all key resources are kept inside of the OS
  - protected by hardware (mode, memory management)
  - processes cannot access them directly
- all users are authenticated to the OS
  - by a trusted agent that is (essentially) part of the OS
- all access control decisions are made by the OS
  - the only way to access resources is through the OS
  - we trust the OS to ensure privacy and proper sharing
- what if key resources could not be kept in OS?

Network Security – things get worse

- the OS cannot guarantee privacy and integrity
  - network transactions happen outside of the OS
- authentication
  - all agents may not be in local password file
- "man-in-the-middle" attacks
  - wire connecting the user to the system is insecure
- systems are open to vandalism and espionage
  - many systems are purposely open to the public
  - private systems may still be on internet

Man-in-the-Middle Attacks

- assume someone watching all network traffic
  - your traffic is being routed through many machines
  - most internet traffic is not encrypted
  - snooping utilities are widely available
  - passwords may be sent in clear text
- assume someone can forge messages from you
  - your traffic is being routed through many machines
  - some of them may be owned by bad people
  - they can hijack connection after you log in
  - they can replay previous messages, forge new ones

Goals of Network Security

- secure conversations
  - privacy: only you and your partner know what is said
  - integrity: nobody can tamper with your messages
- positive identification of both parties
  - authentication of the identity of message sender
  - assurance that a message is not a replay or forgery
  - non-repudiation: he cannot claim "I didn't say that"
- they must be assured in an insecure environment
  - messages are exchanged over public networks
  - messages are filtered through private computers

Elements of Network Security

- simple symmetric encryption
  - can be used to ensure both privacy and integrity
- cryptographic hashes
  - powerful tamper detection
- public key encryption
  - basis for modern digital privacy & authentication
- digital signatures and public key certificates
  - powerful tools to authenticate a message's sender
- delegated authority
  - enabling us to trust a stranger's credentials

Reading and Assignments

Reading:

- Reiher: Cryptography
- Reiher: Distributed Systems Security
- Secure Socket Layer communication

Projects:

- start looking at project 4C (secure communication, big)

Supplementary Slides

Conclusion

- Distributed systems offer us much greater power than one machine can provide
- They do so at costs of complexity and security risk
- We handle the complexity by using distributed systems in a few carefully defined ways
- We handle the security risk by proper use of cryptography and other tools