Distributed Systems

13A. Distributed Systems: Goals & Challenges
13B. Distributed Systems: Communication
13H. Public Key Encryption

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

Fundamental 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. global variables)
  – 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)

RPC is not a complete solution

- client/server binding model
  - expects to be given a live connection
- threading model implementaition
  - 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)
  - stateless, scalable, cacheable, layerable
  - operations limited to Create/Read/Update/Delete
Sample SOAP Request

```xml
<?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

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

Asymmetric Cryptosystems

- Encryption and decryption use different keys
  - $C = E(K_E, P)$
  - $P = D(K_D, C)$
  - $P = D(K_D, E(K_E, P))$
- Often works the other way, too
  - $C = E(K_F, P)$
  - $P = D(K_E, C)$
  - $P = D(K_D, E(K_E, P))$
- Public Key (PK) encryption is such a system
  - $K_E$ is called the public key, $K_D$ is called the private key
  - It is very difficult to infer $K_D$ from $D, E, C, P$ and $K_E$

Asymmetric Encryption (public key)

- RSA
  - The most popular public key algorithm
  - Used on pretty much everyone's computer
- Elliptic curve cryptography
  - An alternative to RSA
  - Tends to have better performance
  - Not as widely used or studied

Example Public Key Ciphers

- an asymmetric (two key) encryption technique
  - One key is private – (not shared) only key owner knows it
  - One key is public – it is advertised to the entire world
- It can be used to implement "your eyes only" privacy
  - Encrypt a message with the recipient's public key
  - The message can only be decrypted with his private key
- It can be used to implement guaranteed signatures
  - Sender encrypts message with his own private key
  - If it decrypts $w/ sender's public key, it must be from sender
- These can be combined for authentication + privacy
Digital Signatures

- **Signing a message**
  - encrypting a message with private key signs it
    - only you could have encrypted it, it must be from you
    - it has not been tampered with since you wrote it
  - encrypting everything with private key is a bad idea
    - if you use a key too much, someone will eventually crack it
    - asymmetric encryption is extremely slow
  - no need to encrypt whole message with private key
    - compute a cryptographic hash of your message
    - encrypt the cryptographic hash with your private key
    - faster and safer than encrypting whole message

Using Digital Signatures

- much better than ink signatures or fingerprints
  - uniquely identify the document signer
  - uniquely identify the document that was signed
  - signature cannot be copied onto another document
- we know document has not been tampered with
  - we can recompute the cryptographic hash at any time
  - confirm it matches message the sender signed
  - sender cannot later claim not to have signed message
- digitally signed contracts can be legally binding
  - several states have passed such legislation

Can we trust public keys?

- if I have a public key
  - I can authenticate received messages
  - I know they were sent by the owner of the private key
- but how do I know who that person is?
  - can I be sure who a public key belongs to?
  - how do I know this is really my bank's public key?
  - could some swindler have sent me his key instead?
- I would like a certificate of authenticity
  - a digital Notary stamp
  - certifying who the real owner of a public key is

Public Key Certificates

(What Is a PK Certificate?)

- Essentially a data structure
  - name and description of an actor
  - public key belonging to that actor
  - validity/expiration information
- Signed by someone I trust
  - whose public key I already have
  - a digital Notary Public
- Testifying that the actor owns the public key
  - and (by implication) the matching private key
Using Public Key Certificates

• if I know public key of the authority who signed it
  – I can validate the signature is correct
  – I can tell the certificate has not been tampered with
• if I trust the authority who signed the certificate
  – I can trust they authenticated the certificate owner
  – e.g. we trust drivers licenses and passports
• but first I must know and trust signing authority
  – everybody knows and trusts RSA as an authority
  – does that mean that only RSA can sign certificates?

Delegated Authority

• I can accept certificates from a known authority
  – not practical for one authority to issue all certificates
  – how to validate certificates from unknown authority
• what if he has a certificate
  – that is signed by an authority I know and trust
  – that authorizes him to issue certificates
• if I trust RSA, I should also trust their "delegates"
  – perhaps I can also trust people they delegate
  – but I would need to see the entire chain of certificates

Certificate Authority Hierarchy

A Chicken and Egg Problem

• certificate is a formal introduction to a new partner
  – I can trust he is who he claims to be
  – if I can validate the certificate
    – by following the chain of delegated trust
• How do I trust the authority at the end of the chain?
• Ultimately through some other mechanism
  – OS or browser comes with an initial set of certificates
  – hand delivered (as in our IOT security project)
  – down-loaded, over a secure channel, from trusted site
  – you decide to accept a new certificate

Assignments

• Reading
  – A/D 48 NFS (Network File System)
  – SSL (Secure Socket Layer)
  – Resource Leases
  – Authentication Services

Supplementary Slides
new view of “system architecture”
• customers pay for services
  – we design and build systems to provide services
• services are built up from protocols
  – service is delivered to customers via a network
  – service is provided by collaborating servers
  – servers are commissioned/controlled by network
• the fundamental unit of service is a node
  – provides defined services over defined protocols
  – language, OS, ISA are mere implementation details

Marshal (and un-marshall)
• 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