Communication

Today

- Basics of IPC in distributed systems
- Models for communication – RPC, MOM and Streaming, multicasting

IPC in distributed systems

- IPC is based on send/receive msgs
- For this to work, both parties must agree on a number of things
  - How many volts to use to signal a 0-bit?
  - How does the receiver knows it got the last bit of a msg?
  - How longs are integers?
  - …
- To simplify this – partition the problem into layers, each layer in a system communicates with the same layer in the other end
  - International Standard Organization's Open Systems Interconnection model – ISO OSI

Protocols in communication

- Lower-level protocols
  - Physical – deals with mechanical and electrical details
  - Data link – groups bits into frames & ensure are correctly received
  - Network – describes how packet are routed, lowest i/f for most distributed systems (IP)
- Transport protocols
  - Transfer messages between clients, including breaking them into packets, controlling low, etc (TCP & connectionless UDP)
- High-level protocols
  - Session – provides dialog control and synchronization
  - Presentation – resolves differences in formats among sites
  - Application – originally to contain a set of standard apps

Middleware

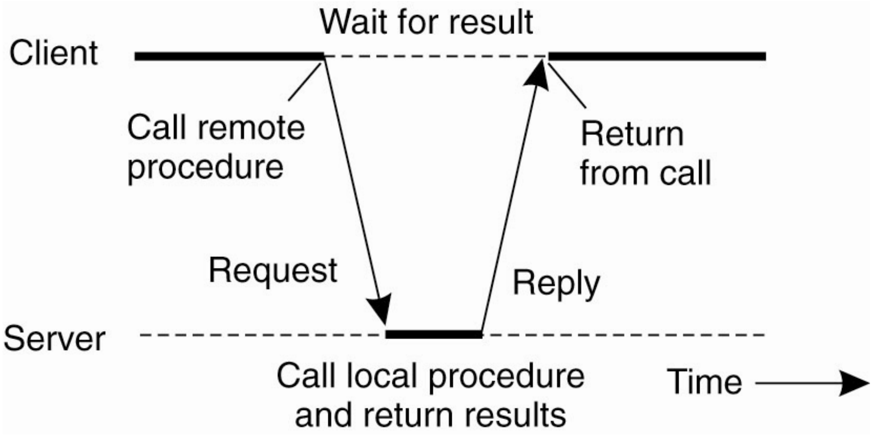
- Basically an “application” providing general-purpose, high-level protocols that can be used by others
  - Rich set of communication protocols
  - (Un)marshaling of data
  - Naming protocols so that different apps can share resources
  - Security protocols
  - Scaling mechanisms such as support for replication and caching
- What’s left are really application-specific protocols

Types of communication

- Persistent or transient
  - Persistent – a message submitted for transmission is stored as long as it takes to deliver it
  - Transient – … as long as the sending/receiving applications are execution (e.g. if transmission is interrupted, msg is lost)
- Asynchronous or synchronous
  - Sender continues or blocks until request has been accepted
  - Points of synchronization
    - At request submission, delivery or after processing
- Client/server
  - Normally based on transient & synchronous communication
- Discrete or streaming
  - Each message is a complete unit of info. or part of whole

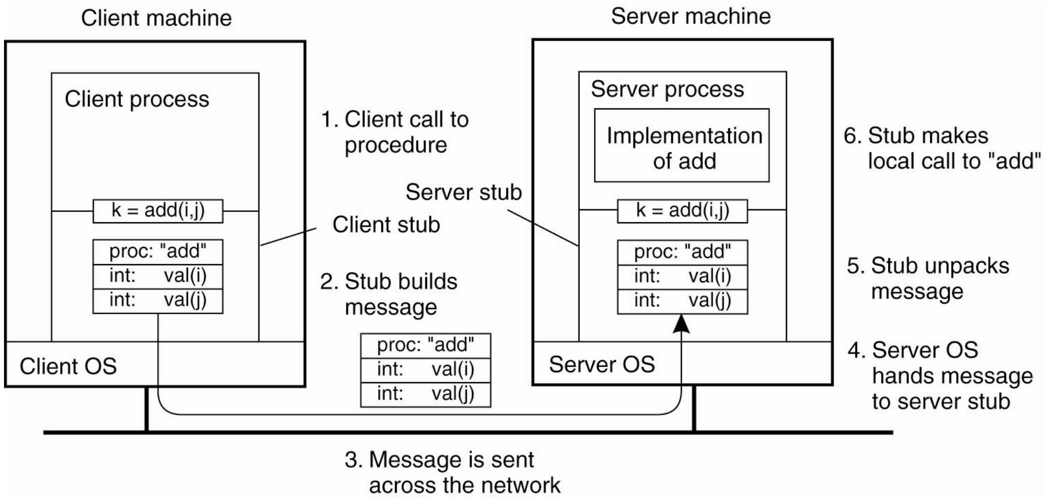
Remote Procedure Call

- Some observations
  - Application developers are familiar with simple procedure model
  - Well engineered procedures operate in isolation
  - There's no fundamental reason not to execute procedures on a separate machine
- Can you hide sender/receiver communication using procedure calls?



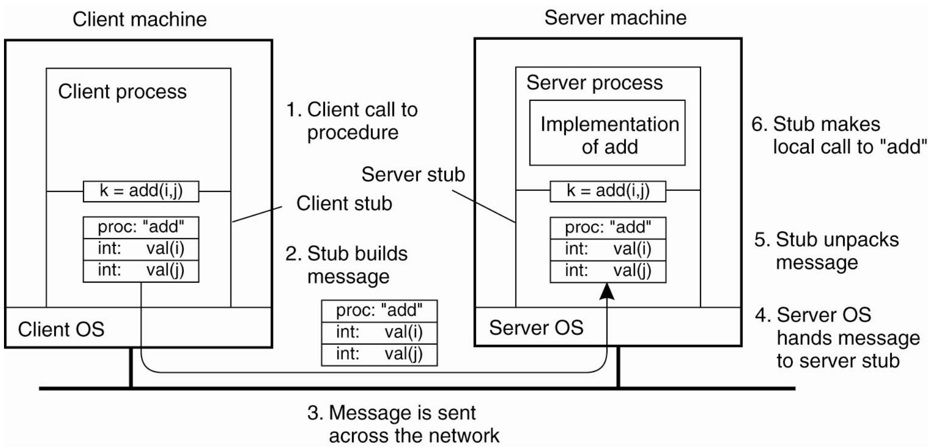
Basic RPC operation

- A RPC occurs in the following steps:
  1. Client procedure calls client stub
  2. Client stub builds msg. and calls the local OS
  3. Client's OS sends msg. to remote OS
  4. Remote OS gives msg. to server stub
  5. Server stub unpacks parameters and calls server
  6. Server does the work and returns the result to stub
  7. Server stub packs it in a msg. and calls local OS
  8. Server's OS sends msg. to client's OS
  9. Client's OS gives msg. to client stub
  10. Stub unpacks result and returns to client



RPC: Parameter passing

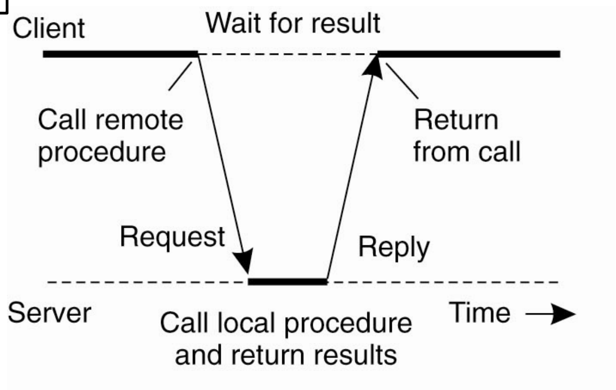
- Marshaling – more than wrapping parameters
  - Client and server may have different data representations
  - Client and server have to agree on encoding:
    - How are basic data values represented (integers, floats, …)
    - How are complex data values represented (arrays, unions)
- RPC assumes
  - Copy in/copy out semantics
  - All data to be worked on is passed by parameters
- How about pointers?
  - Copy/restore instead of call-by-reference
  - Remote reference for more complex structures



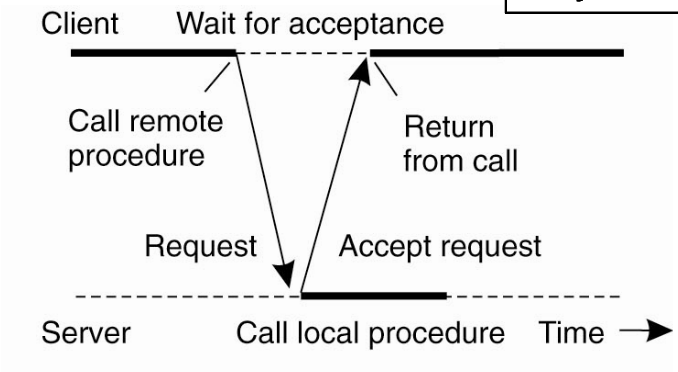
Asynchronous RPCs

- Get rid of the strict request-reply behavior, but let the client continue w/o waiting for server's answer

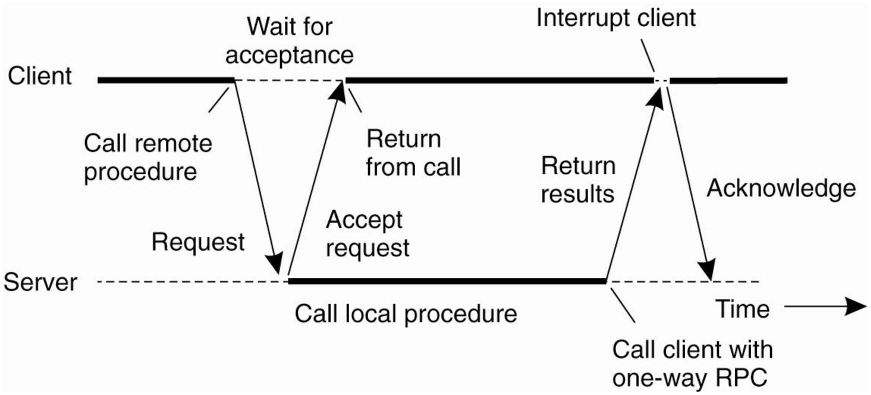
Synchronous



Asynchronous



- A variation – deferred synchronous RPC



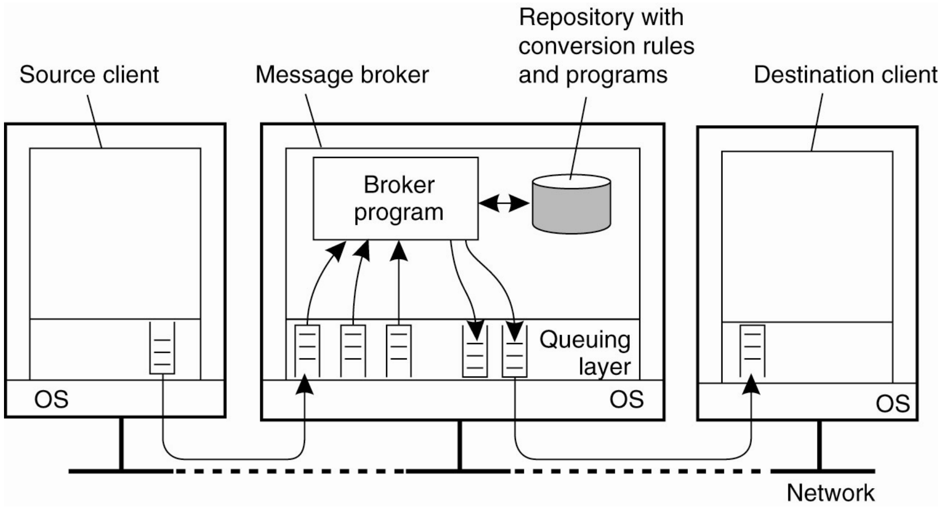
Message Oriented Communication

- *What if we cannot assume the receiver side is going to be executing at the time the request is issued?*
- Asynchronous persistent communication through support of middleware-level queues – queues correspond to buffers at communication servers

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

Message brokers and apps. integration

- Message queuing systems assume a common messaging protocol: all applications agree on message format
- To use MQ systems for integration – message broker: takes care of application heterogeneity
  - Transforms incoming messages to target format
  - Often acts as an application gateway
  - May provide subject-based routing capabilities



Stream-oriented communication

- All communication facilities discussed so far are essentially based on discrete, exchange of information
- Continuous media – values are time dependent
  - Audio, video. sensor data (temperature, pressure, etc.)
- Transmission modes – different timing guarantees with respect to data transfer
  - Asynchronous: no restrictions with respect to when data is to be delivered
  - Synchronous: define a maximum end-to-end delay for individual data packets
  - Isochronous: define a maximum and minimum end-to-end delay (jitter is bounded)

Streams

- A (continuous) data stream is a connection-oriented comm. facility that supports isochronous transmission
- Some common stream characteristics:
  - Streams are unidirectional
  - There is generally a single source, and one or more sinks
  - Often, either the sink and/or source is a wrapper around hardware (e.g., camera, CD device, TV monitor, dedicated storage)
- Stream types:
  - Simple: consists of a single flow of data, e.g., audio or video
  - Complex: multiple data flows, e.g., stereo audio or combination audio/video
- Streams are all about timely delivery of data. How do you specify this QoS? What do you do in the Internet?

Group communication – multicast

- A key service for many interesting applications
  - Online gaming, video conferencing, content distribution …
- Approaches to group communication

Multicast

Decouples # of receivers from amount of state kept at nodes

Reduces redundant network communication

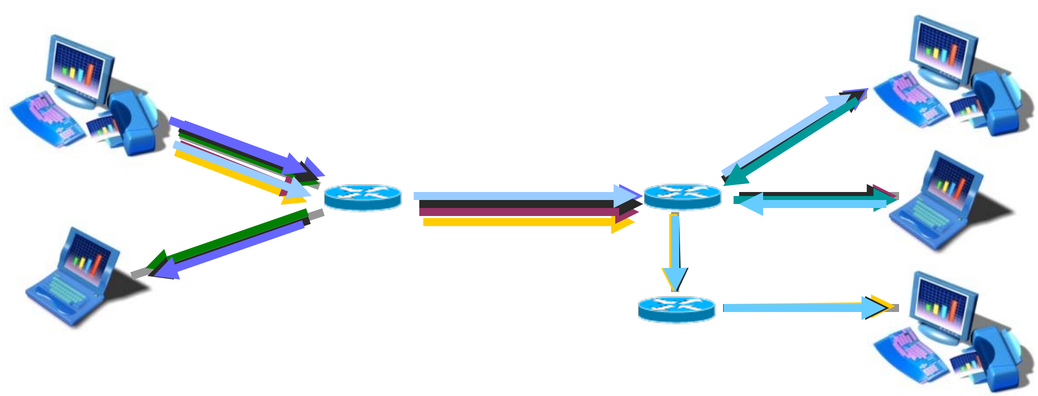
Basic unica

Scalability is

replication at

link stress, …

group state)



Application level multicast - issues

- Minimize link stress – how often does an overlay message cross the same physical link
- Minimize stretch – delay between overlay path and network-level path
- Performance-centric or DHT-based?
- Churn, i.e. high transiency of end systems
- Root-bottleneck problem for bandwidth-intensive applications
- Uneven load distribution of tree-based protocols

Gossip-based data dissemination

- Assuming there are no write-write conflicts
  - Update operations initially performed at one (few) nodes
  - Node passes its updated state to a limited set of neighbors
  - Update propagation is lazy, eventually each update should reach every node
- Anti-entropy
  - Node chooses another at random, and exchanges differences
  - Push, pull or push/pull
- Gossiping
  - Node just updated, tells others about it; if the node contacted already knows about it, the source stops w/ probability 1/k
  - If you need everyone to know, gossiping along doesn't do it
- And how do you delete items?!
  - Death certificates and dormant death certificates

Summary

- Communication is at the heart of distributed systems
- Powerful primitives makes programming them a lot easier
- Solutions for large distributed systems should consider a number of different issues
  - Referential and temporal decoupling
  - Group communication
  - ….