

CSC 546: Client/Server Fundamentals

Fall 2000

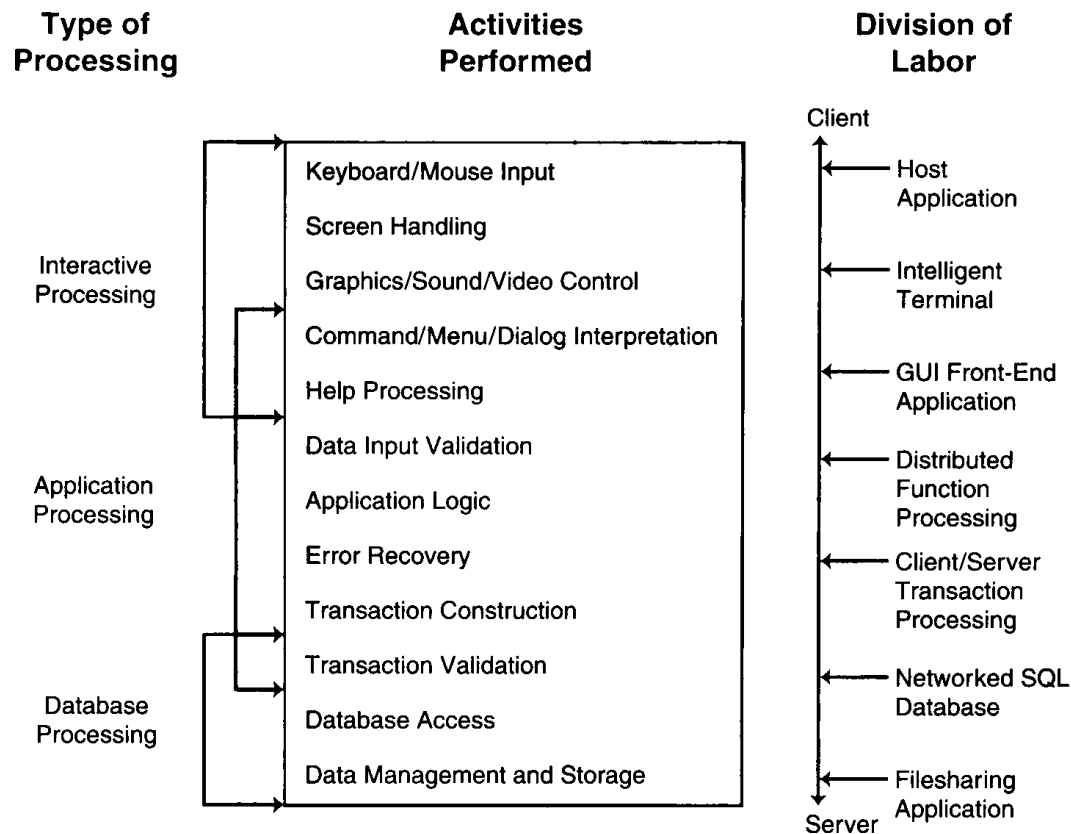
Implementation aspects

- design and development
 - balancing processing
 - development methodology
- structural aspects
 - data distribution, data movement, network efficiency
 - scalability
 - security

Balancing processing

application processing has 3 segments

- **interactive:** screen handling, command interp, help processing should be on client
- **database:** data access and management should be on a central, specialized server
- **application:** app logic, transaction processing, error handling may be split



Design rules of thumb

- push as much processing as possible to the client (especially CPU-intensive)
- manage shared resources (databases, printers, documentation) on the server
- maintain a virtual view of servers during design

- distribute services among several servers to avoid bottlenecks
- use tiered processing to achieve scalability (easier to add new tiers)
- minimize data transfers (caching, data compression, transfer checkpoints)
- avoid unnecessary resource locking

- centralize administration
- design for remote administration and monitoring
- build security perimeters into system applications
- analyze reliability and efficiency
 - throughput chains help spot bottlenecks, queueing models help spot overutilization
- test your client/server integration early
 - measure demand independent of capacity, design for outages

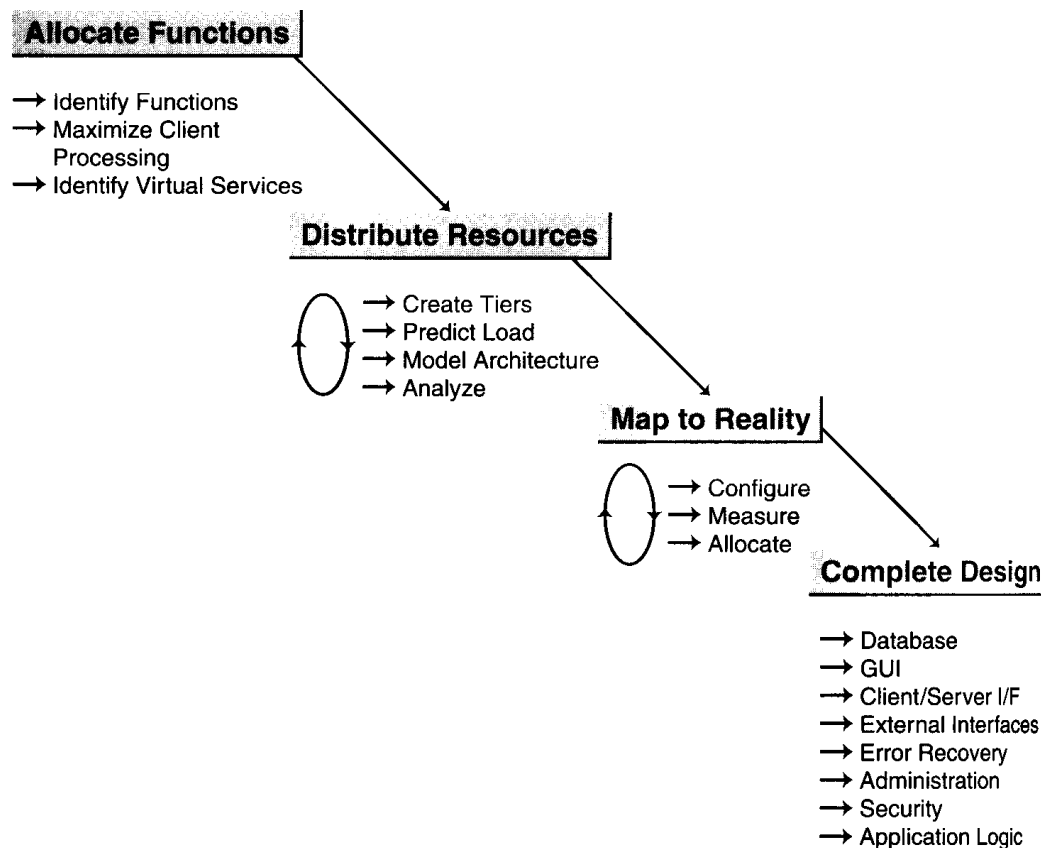
Client/server development methodology

traditional application development involves 1 system

- can hog local resources, only affect local environment
- application integration occurs on the local machine

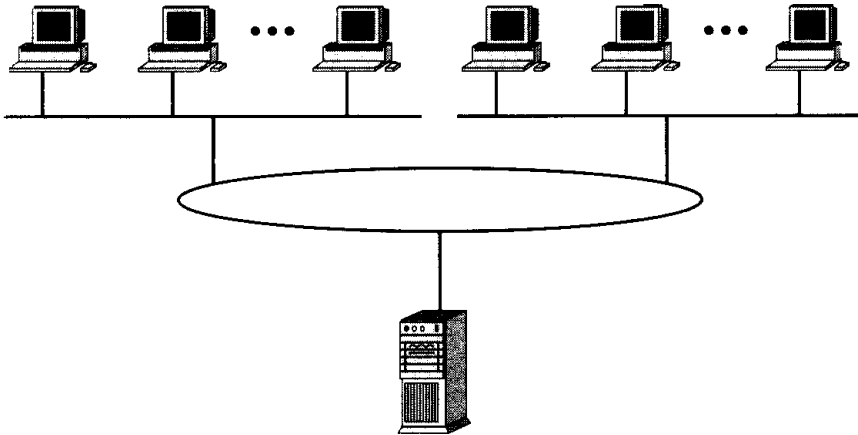
client/server requires the development of 2 different systems

- must take into account processing balance, data distribution, security, ...
- application integration is across two environments

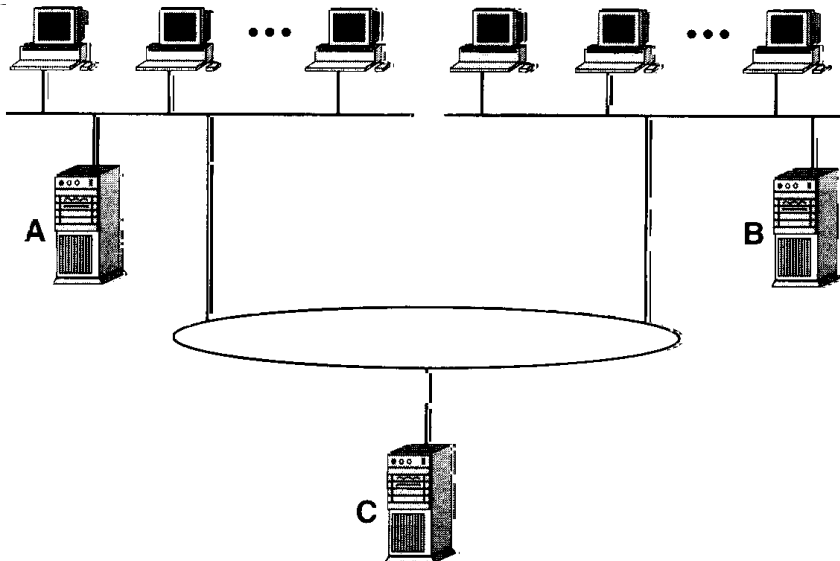


system development should occur in stages

Structural aspects



if data is centralized on a single server, have a 2-tiered architecture



if data is distributed, a 3-tiered architecture results

- servers A & B provide similar, localized services
- server C provides global services to all workgroups

Data distribution

replicated data

- can duplicate data on servers A & B, spread accesses across workgroups
- updates to a workgroup server must be propagated
- data consistency and integrity are issues (e.g., 2-phase commit)

partitioned data

- can partition data across servers A & B (no duplication)
- if desired data not on local server, next tier server C acts as info broker
- note: requires all servers be available

reorganized data

- can store data in different ways on servers A & B (e.g., levels of granularity)
- next tier server C maintains a summary view, directs requests

cached data

- can cache data on servers A & B, next tier server C is central data store
- if cache is updateable, must propagate changes back to central store
- again, data consistency and integrity are issues

federated data

- data is managed in different databases, coordination via transaction managers

Minimizing data movement

affinity analysis can identify localized data interactions

affinity: $F(e)$ is the frequency of any interaction with that entity

mutual affinity: $F(e_1, e_2)$ is the frequency of mutual interaction between 2 entities

affinity factor: $AF(e_1, e_2)$ is the proportion of interactions in common

$$AF(e_1, e_2) = F(e_1, e_2) / (F(e_1) + F(e_2))$$

	Server 1	Server 2	Server 3	F(e)
Client 1	30	200	100	330
Client 2	50	100	175	325
Client 3	150	100	300	550
Client 4	100	75	150	325
Client 5	130	50	125	305
F(s)	460	525	850	

here, numbers could be # transactions, I/O ops, packets sent, ...

	Server 1	Server 2	Server 3
Client 1	0.038	0.234	0.085
Client 2	0.064	0.118	0.149
Client 3	0.149	0.093	0.214
Client 4	0.127	0.088	0.128
Client 5	0.170	0.060	0.108

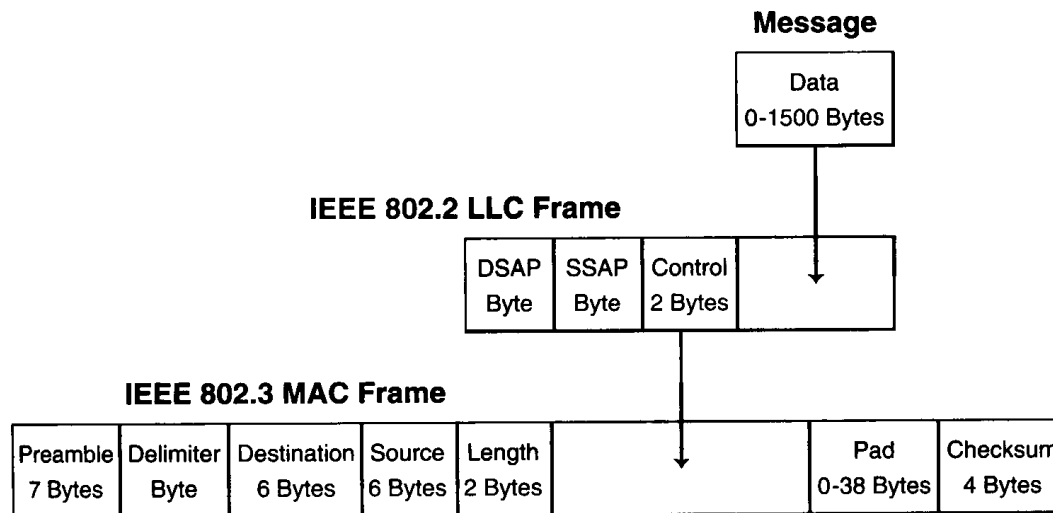
Client1 & Server2 have highest affinity factor, should be located as close as possible

can also minimize movement with stored procedures, caching, compression, ...

Network efficiency

measures the proportion of real data transferred to network overhead

e.g., Ethernet packets use 30 bytes of routing & validation info
message acknowledgement requires 64 bytes
network delay occurs for sending the message & acknowledgement



efficiency = data / (data + routing info + acknowledgement + network delay)

for 300 byte message: efficiency = $300 / (300 + 30 + 64 + 2*64) = 57 \%$

for 1200 byte message: efficiency = $1200 / (1200 + 30 + 64 + 2*64) = 84 \%$

Network efficiency (cont.)

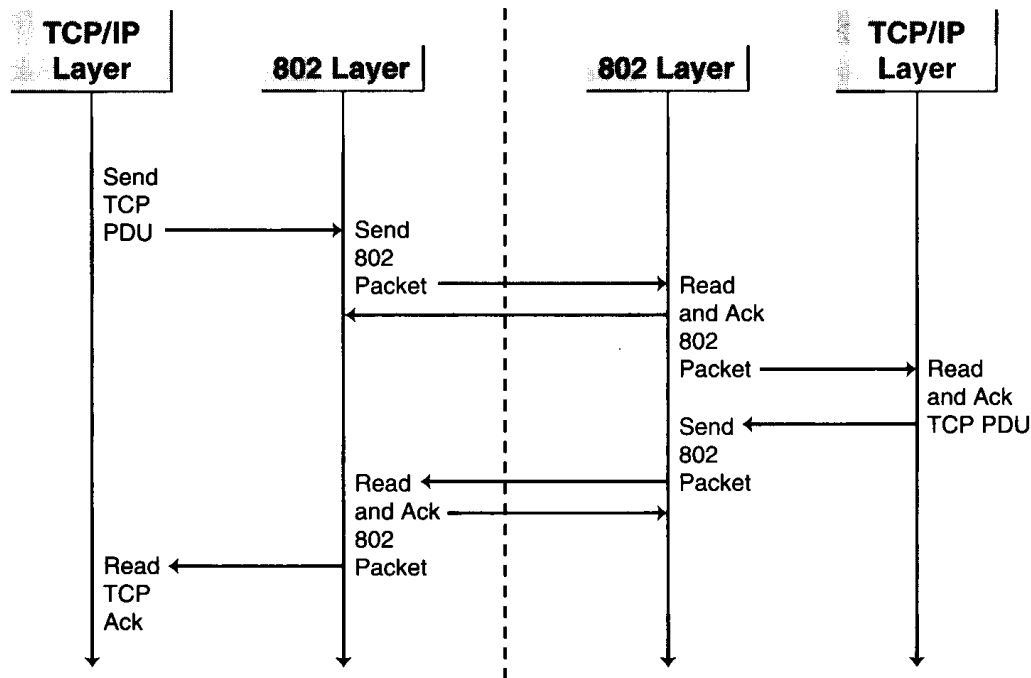
protocol layering reduces network efficiency

- at TCP/IP layer, data is encapsulated into protocol data unit (PDU) → +41 bytes
- PDU is then encapsulated into an Ethernet 802 packet → +30 bytes
- receiver acknowledges the 802 packet, delivers PDU to TCP/IP layer
- TCP layer must acknowledge receipt, send packet back to sender

efficiency = data / (data + overhead + 802 ack + TCP ack + 802 ack + network delay)

for 300 byte message: efficiency = $300 / (300 + 71 + 64 + 71 + 4*64) = 39 \%$

for 1200 byte message: efficiency = $1200 / (1200 + 71 + 64 + 71 + 4*64) = 72 \%$



performance can be improved
by "piggybacking" data transfers

clearly, want to maximize the
amount of actual data sent in a
packet

Multiserver dataflows

in a multi-tier network, servers must communicate with servers

peer-to-peer dataflow

- 2nd tier server acts as surrogate client to 3rd tier server, ...
- service requests are passed up the hierarchy until they can be handled

broadcast dataflow

- if server-server interactions are rare & servers are close, can broadcast requests to all and the appropriate server will respond
- doesn't scale well (heavy traffic), not secure

filtered dataflow

- sometimes want server to filter its response
 - e.g., image server, images are stored in compressed form
 - could require client to decompress (may require specialized software/hardware)
 - could require server to decompress (processing can slow server)
 - have decompression filter on separate server, send response through that server

Scalability

in a large client/server system, minimizing network traffic is essential

- reduce unnecessary communication if possible
- smooth out traffic loads (e.g., avoid 5:00 uploads)
- whenever possible, have the server initiate communication (pull, not push)
 - e.g., avoid clients polling server for needed resource
 - instead, have server queue the request and notify when available

server capacity

- increasing server performance can improve performance (up to a point)
- for large demand, need to explore multi-tiers, data partitioning, ...

WAN vs. LAN

- wide area networks are slower, more error-prone
- minimizing transfers is even more essential
- must provide for error recovery, checkpoints to save partial transfers

Security

3 facets of security: integrity, availability & confidentiality

security perimeters

physical security: control physical access to hardware

system security: control who can log on to the system

application security: control who can run an application

data security: control which client applications can access

a well designed perimeter provides

identification: determine the identity of the user crossing the perimeter

authentication: verify their identity

access control: limit their activities within the perimeter

auditing: track their activities within the perimeter

surveillance: monitor activities and assets within the perimeter for violations

Encryption

encryption can ensure confidentiality of data transfers

- private key encryption, e.g., DES
sender & receiver share a secret key
encryption algorithm uses key to encrypt/decrypt

`encrypt(message, key) → code` `decrypt(code, key) → message`

in principle, coded message cannot be decoded without the key

PROBLEM: how do you generate, share, protect the key?

- public key encryption, e.g., RSA
each person has a public key and a private key
encode message with receiver's public key, can only be decoded with private key

`encrypt(message, public_key) → code`
`decrypt(code, private_key) → message`

does not require the exchange/security of a secret key, but much less efficient

- hybrid approach, e.g., Kerberos
for each transfer, have sender generate a secret key
use public key encryption to send the secret key to the receiver
then, use private key encryption

Next week...

final exam: Thursday, Dec. 14 6:15

- similar format to midterm
 - true or false
 - discussion/short answer
 - explain or modify code (HTML, CGI, sockets)
- cumulative, but will emphasize material since the midterm
- designed to take 90-120 minutes, will allow full 165 minutes
- study hints:
 - review lecture notes
 - review text
 - look for supplementary materials where needed (e.g., Web search)
 - think big picture (assimilate the material)
 - use [online review sheet](#) as a study guide (*but not exhaustive*)

REVIEW/STUDY SESSION?