Adaptive Servers for High Performance Imaging

James C. Hu
Dr. Douglas C. Schmidt
Washington University in St. Louis
{jxh,schmidt}@cs.wustl.edu
http://www.cs.wustl.edu/~{jxh,schmidt}/

December 10, 1996

Overview

- Motivation
- Background
- Requirements
- Design Issues
- Design Goals
- Threading Models
- Survey of Web Servers
- Design overview
- Objectives
- Summary

Motivation

- Problems
  - End users desire image-intensive content
    - Viewing image-intensive content can be slow
      - Image serving is *bandwidth* intensive
      - Image processing is *CPU* intensive
    - Latency also affects performance
      - Latency from connection establishment
      - Latency from server inefficiencies
  - Service providers must handle variation transparently
    - *Hardware platforms*
    - *Service protocols*
    - *Client capabilities*
    - *Network architecture*

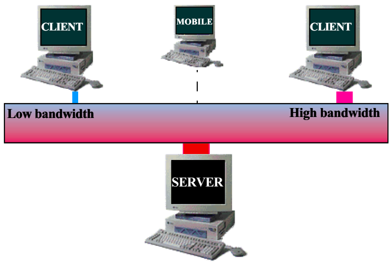
Motivation

- Solution
  - Clients
    - *Adaptive* strategies that allow users view content as quickly as possible
  - Servers
    - *Adaptive* strategies that serve requests in a timely manner
    - *Adaptive* strategies that anticipate variation

Background

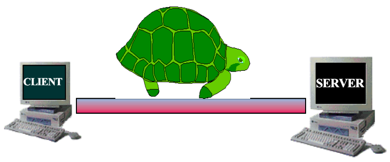
- A history of Web Services
  - E-mail (SMTP), Usenet (NNTP)
  - UUCP
  - DNS
  - Finger, FTP, Gopher, WAIS
  - HTTP/1.0
  - Search Engines, Data Mining
- Web service trends
  - Image intensive data
  - Multi-media presentations

Requirements



- Server-side Requirements
  - Handle many simultaneous connections
  - Handle diverse protocols
    - Customized protocol processing
  - Maximize average end-to-end throughput
  - Optimize for available HW/SW technology
    - Single CPU vs. multi-processor
    - OS support for threads
  - Minimize maintenance and configuration effort
    - E.g., performance tuning and protocol updates

Requirements



- Client-side Requirements
  - Minimize latency
  - Optimize throughput for most relevant requests
    - Improve quality of service
  - Integrate with DCOM/CORBA, Java
  - Optimize for available technology
    - PC's vs. workstations
    - Mobile clients

Design Issues

- Centralized or distributed
  - Reliability
  - Client transparency
  - Variations of concurrency
    - *Per-request, per-session, etc.*
- Iterative or concurrent
  - Proactive versus Reactive programming
  - Threading model

Design Issues

- Integration of new services/protocols
  - Ease implementing updates
  - Allow extensions to server functionality (at run-time)
  - Provide client-side extensions
- Domain-specific forces
  - Multiprocessor support
  - Gigabit networks
  - Real-time quality of service
  - Low speed network connections via modem
  - Mobile clients

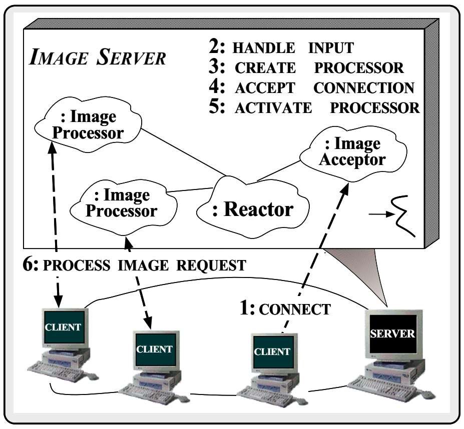
Design Goals

- Late binding of policy and mechanism decisions
  - Supported services and protocols
  - Distribution policy
  - Threading policy
- Simplify performance tuning task
  - High-level specification of parameters
  - Empirically determine appropriate policy
  - Stock configurations from profiles
- Solve complexity once
  - Dynamically configure new services
  - Tune service within existing framework
- Portability
  - Unix and NT
  - C++, Java, Smalltalk, etc.

Threading Models

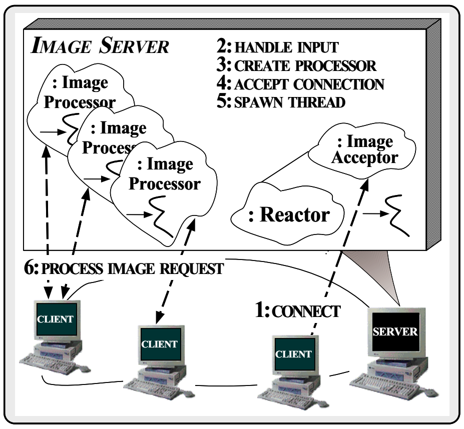
- Alternatives
  - Single threaded
  - Thread-per-request
  - Thread pool
  - Thread-per-session
- Observations
  - No single model is always the best choice
  - Must understand patterns and design tradeoffs
  - Must provide framework for adaptive configuration

Single Threaded



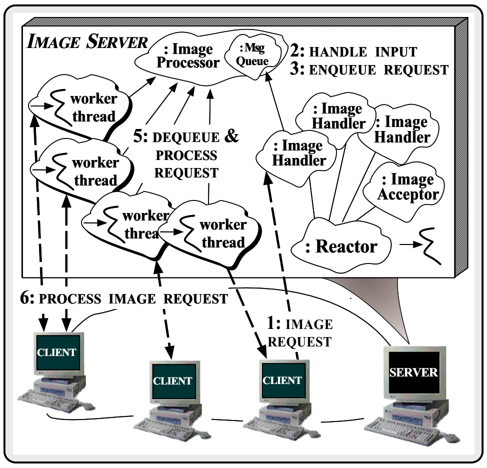
- Advantages
  - May be more efficient than multi-threading
  - No synchronization, less context switching
- Disadvantages
  - Complicated, error-prone programming model
  - Does not scale for multi-processor platforms

Thread-Per-Request



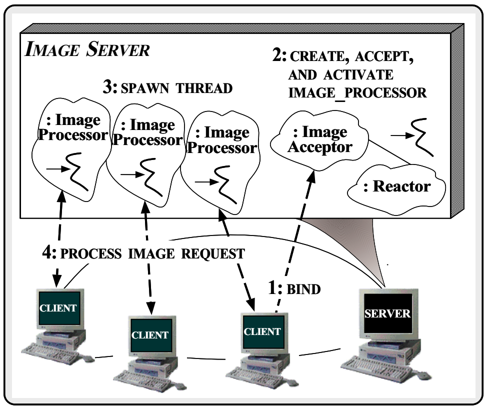
- Advantages
  - Simplifies programming model
  - Can scale with multi-processor platforms
- Disadvantages
  - Can consume excessive server resources
  - Difficult to clean up resources gracefully

Thread Pool



- Advantages
  - Bound threading resources by pre-spawning
- Disadvantages
  - Latency increases with number of requests
  - Queueing may incur extra overhead

Thread-Per-Session



- Advantages
  - No extra connection establishment overhead
  - Can optimize per-session QoS parameters
- Disadvantages
  - Resources can still be totally consumed
  - Does not work for HTTP/1.0

Survey of Web Servers

- Servers examined
  - Apache
  - CERN
  - Jigsaw
  - Phttpd
  - Roxen
  - Zeus
  - Enterprise
- Criteria
  - Concurrency support
  - Threading model
  - Extensibility
  - Portability
  - Implementation language
  - Availability

Apache

- Uses process level concurrency
  - Avoids process creation overhead with pre-forked servers
- "Thread"-per-request model
- Supports "modules" for server extensions
  - Requires server recompilation and relinking
- Ports to many flavors of Unix
- Implemented in C
- Free software

Jigsaw

- Uses threads for concurrency
- Thread-per-session model (HTTP/1.1)
- Extensibility through Java objects
- Portable to any Java capable system
- Implemented in Java
- Free software

Phttpd

- Uses threads, dynamic linking and kernel modifications
- Thread-per-request model
- Dynamically linkable modules (SIGHUP)
- Implementation targeted for Solaris 2.4
- Implemented in C
- Free software

Roxen

- Iterative server implementation
- Single threaded model
- Extensible through modules
- Portable to Unix
- Interpreted object oriented language, Pike
- Commercial Web server

Enterprise

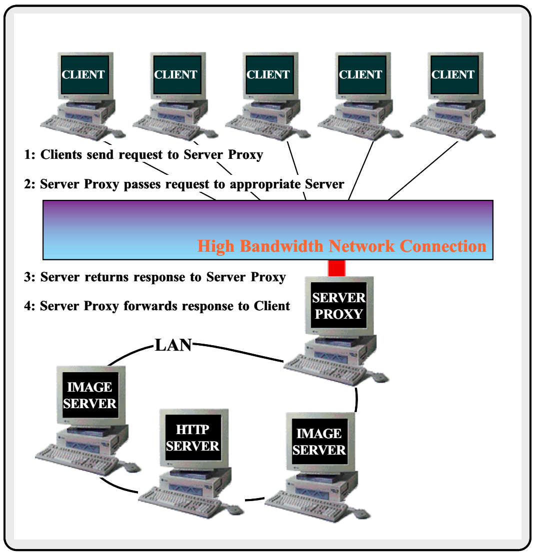
Netscape
Enterprise
Server

- Support for kernel level threads and multiprocessors
- Threading model unknown
- Extensible through Netscape IFC
- Portable to Unix and Windows NT
- Implemented in C/C++
- Commercial Web server

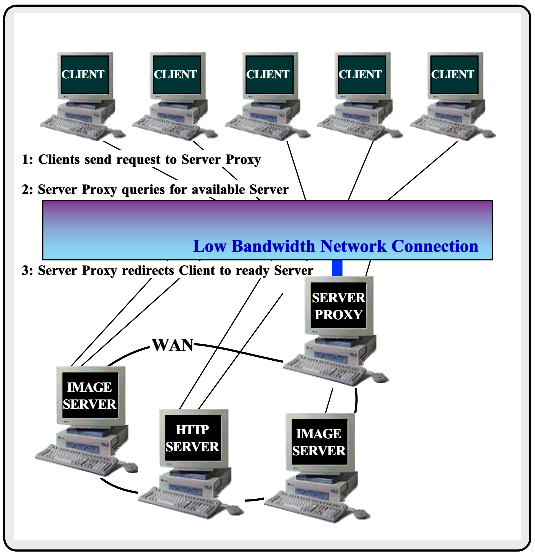
Distributed Server Architectures

- Server request proxy
  - Server has high bandwidth connection to network
  - Expect many simultaneous connection requests
  - Request processing can cause unacceptable jitter
- Server redirect proxy
  - Server has low bandwidth connection to network
  - Expect many simultaneous connection requests
  - Request processing can starve out other requests

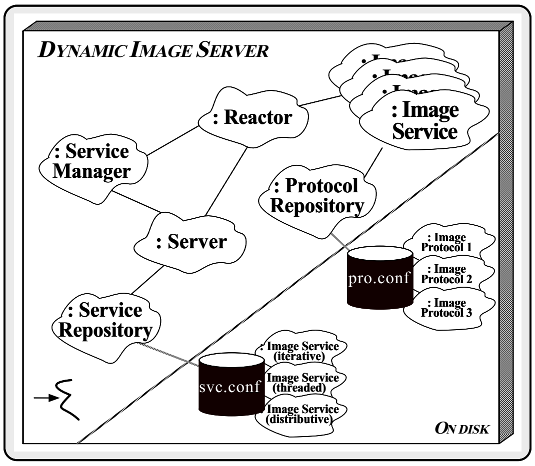
Server Request Proxy



Server Redirect Proxy

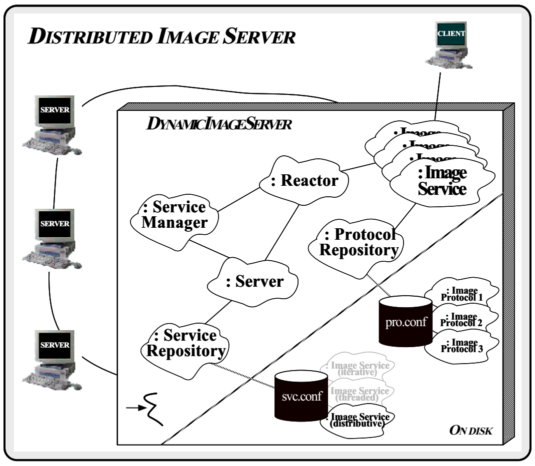


Server Design Overview



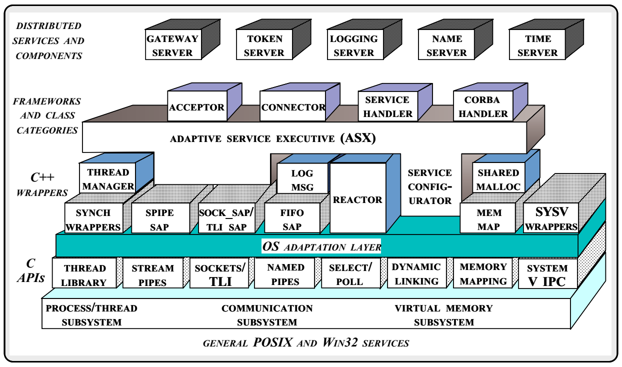
- Combines multiple patterns
  - Reactor, Service Configurator, Acceptor, Active Object
- Supports dynamic (re)configuration without disruption of service

Server Design Overview



- The Dynamic Image Server as a Distributive Server Proxy
- Independent of the policies and mechanisms of other servers

ACE Overview



- A set of C++ and Java components and frameworks based on key design patterns for concurrent communication software
  - `http://www.cs.wustl.edu/~schmidt/ACE.html`
- Ported to UNIX, Win32, MVS and embedded platforms
- Currently used by dozens of companies
  - E.g., Siemens, Motorola, Ericsson, Kodak, Bellcore, MDA, etc.

Research Proposal

- Implementation of high performance, adaptive Web system server
  - *Adaptive protocols and runtime architectures*
- Implementation of imaging service
  - HTTP 1.0 and HTTP 1.1
  - DICOM/HL7
  - Internet Imaging Protocol (draft)
  - Direct ATM
- Implementation of imaging client plug-in
  - *Add support for Kodak Photo CD*
- Server and Client-side image processing services
- Demonstration of portability and performance
  - *End-to-end performance*
  - *Aggregate and individual performance*
- High-level performance analysis and simulation tool

Summary

- In general, current solutions are static and inflexible
  - *Use static threading models*
  - *Ignore distributive models*
  - *No adaptive protocols*
- Clients are also inflexible
  - *Cannot dynamically adopt new protocols*
- Different threading strategies that can be used to achieve performance for different conditions
- Server performance may be increased by utilizing an adaptive design and support for distributive models
- Adaptive protocols may allow improved end-to-end performance