CSx55: Distributed Systems
[Distributed Servers]

Shrideep Pallickara
Computer Science
Colorado State University

Computer Science Department   Colorado State University

1

Topics covered in this lecture

- Threads in Distributed Servers
- Server design issues
- State in Servers
- Distributed Servers
- TCP Handoffs
- Route optimizations using MIPv6

Colorado State University   Professor: Shrideep Pallickara   Computer Science Department   Distributed Servers   L10.2

2

THREADS IN DISTRIBUTED SYSTEMS

3

Threads in distributed systems: Multithreaded clients

- **Hide** communication latencies
  - Initiate communications
  - Immediately do something else

  Interleave

- Web browsers
  - As soon as main HTML page is fetched
    - Display it
  - Activate threads to retrieve other data types

  Identical Code

4

Several connections can be opened simultaneously

- To the same server
  - If the server is overloaded; things get even slower
- To replicated servers
  - Data transfer in **parallel**
  - Much faster rendering of content

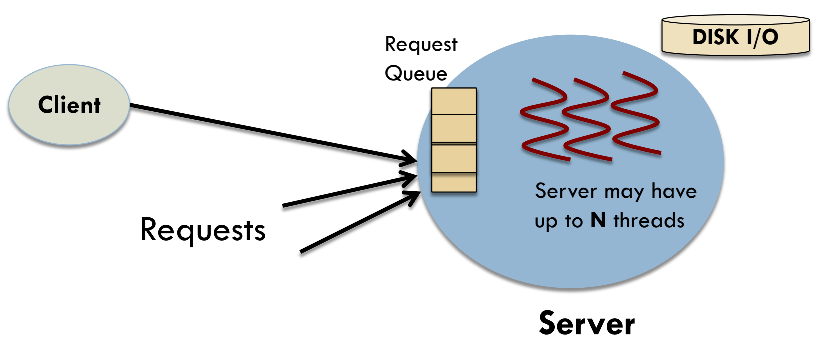
Multithreaded Servers

- Simplifies server code
- Easier to develop servers that exploit parallelism
- E.g.: Handling concurrent connections
  - Each connection managed by a different thread
  - Multiple connections handled by a **pool** of threads

AN EXAMPLE OF PERFORMANCE IMPROVEMENTS WITH THREADS

7

Client and Server with Threads



COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT DISTRIBUTED SERVERS L10.8

8

Server side processing

- Server has **queue** of requests received from clients
- Server also has a **pool** of one or more threads
  - Each thread repeatedly removes requests & processes it
- Each thread applies the same methods to process the requests
  - Each request takes 2 ms of processing PLUS 8 ms of I/O (when server reads from disk i.e. no caching)

Maximum server throughput with 1 thread

- The turnaround time for handling any request is 2+8 = 10 ms
- The server can handle 100 requests per second
- Any new requests that arrive while the thread is handling a request?
  - These will be queued

Server throughput with 2 threads

- We assume that the threads are independently schedulable
  - One thread can be scheduled while the other is blocked for I/O
- Thread T2 can process a second request when thread T1 is blocked, and vice versa
- This increases throughput … but both threads may be blocked for I/O on the single disk drive
- If all I/O requests are serialized and take 8 ms each?
  - Maximum throughput is 1000/8 = 125 requests/second

Server throughput with disk block caching

- Server keeps data that it reads in buffers
- When a server thread tries to retrieve data
  - It first examines the cache and avoids disk accesses if it finds data element there
- If the hit rate is 75%?
  - The mean I/O time per-request reduces to
    $(0.75 \times 0 + 0.25 \times 8)$ = 2 milliseconds
- Maximum theoretical throughput?
  - Becomes 500 requests per second

But there are costs associated with caching

- Average processor time for a request increases
  - This is because it takes time to search for cached data for every operation
  - Let us assume that this is now 2.5 milliseconds
- The server can now handle 1000/2.5 requests per second i.e. 400

Let's look at caching plus multiple threads

- Each request takes about 2.5 (processing) + 2 (I/O)
  - Total time per request is now 4.5 mSecs when disk accesses are serialized
  - Each thread can do 1000/4.5 requests per second i.e. 222 requests/second
- With two threads?
  - 444 requests/second
- With three threads?
  - 500 requests (bound by the I/O time)

THREADING ARCHITECTURES FOR SERVERS

15

Worker pool architecture

- Server creates a fixed **pool** of worker threads to process requests
  - Pool is initialized when server starts up
- Incoming requests are placed into a queue
  - Workers *retrieve* requests (work units) from the queue and process them

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT DISTRIBUTED SERVERS L10.16

16

Managing priorities in the worker pool?

- Introduce *multiple* queues
- Worker threads **scan** queues in the order of descending priority

Disadvantages of the worker pool model

- Number of worker threads is fixed
  - So, threads in the pool may be too few to adequately cope with the rate of requests
- Need to account for coordinated accesses to the shared queue

Thread-per-request architecture

- Worker thread is spawned for **each** incoming request
  - Worker thread *destroys itself* after processing request
- Advantages:
  - Threads do not contend for the shared work-queue
  - Throughput is potentially maximized
- Disadvantage
  - Overhead for thread creation and destruction operations

Thread-per-connection architecture

- Associates a thread per connection
- New worker thread created when a client makes a connection
  - Destroyed when client closes the connection
- Client may make many requests over the connection

Thread-per-object architecture

- Associate a thread with each remote object
- A separate thread receives requests and queues them
  - But there is a queue per-object

Thread-per-connection & Thread-per-object

- Advantages
  - Server benefits from lower thread management overheads compared to thread-per-request
- Disadvantages
  - Clients may be delayed when a worker thread has several outstanding requests, but another thread has no work to perform

SERVER DESIGN ISSUES

23

Server Design Issues

- **Iterative** Servers
  - Handles request
  - Returns response to requesting client
- **Concurrent** Servers
  - Pass request to a separate thread/process
    - **Multithreaded server**
  - Await new incoming request

24

The endpoint issue

- Clients send their requests to an endpoint
  - **Port** to which a server listens to
- But how do clients know about a port?
  - Globally assign endpoints for well-known ports
    - Internet Assigned Numbers Authority (IANA)
    - FTP {TCP, 21}, HTTP {TCP, 80}

Implementing each service with a separate server could waste resources

- Instead of having multiple servers awaiting client requests
  - Have a single **super-server**
- INETD daemon on Unix
  - Listens to **several ports** for Internet services
    - Pop3 {110}, FTP {21}, Telnet {23}
  - When request comes in:
    1. **Fork** process to handle it
    2. Process **exit**s once done

Designing Servers: Support interruption

- Terminate client session
  - Server will eventually detect connection loss (TCP)
- Send **out-of-band** data
  - Data to be processed before any other client data
- But how can we send this out-of-band data?
  1. Send to a different port
  2. Reuse same connection
     - TCP **urgent data** e.g., `socket.sendUrgentData(int data)`

27

STATE

28

Tracking State in Servers

- Stateless servers
- Stateful servers

Stateless servers

- No state information about clients
  - E.g., Web Servers
- Usually, <u>some state</u> is maintained
  - Log of documents accessed by client
  - But if this is lost, there should be **no disruption** of service
- **Soft state**: track state for a limited time
  - When timer elapses, revert to default behavior

Stateful servers

- Maintain **persistent** information on clients
- Use this to improve **performance**
  - Real and perceived
- Special measures needed to recover from failures

31

Stateful servers: A file server example

- Allows client to maintain **local copy** of file
  - Even for updates to the file
  - Maintain {client,file} tuples to track file state
- Identify who has most recent version of file
- If server crashes it must recover the {client,file} entries

32

A hybrid approach: Have the client send its state to the server

- **Cookies** serve this purpose for Web pages
- Tells a site about the pages accessed by a user
  - Use this to decide how to manage client
  - Sent back to browser every time state info changes
- Cookies don't stay where they are baked!

33

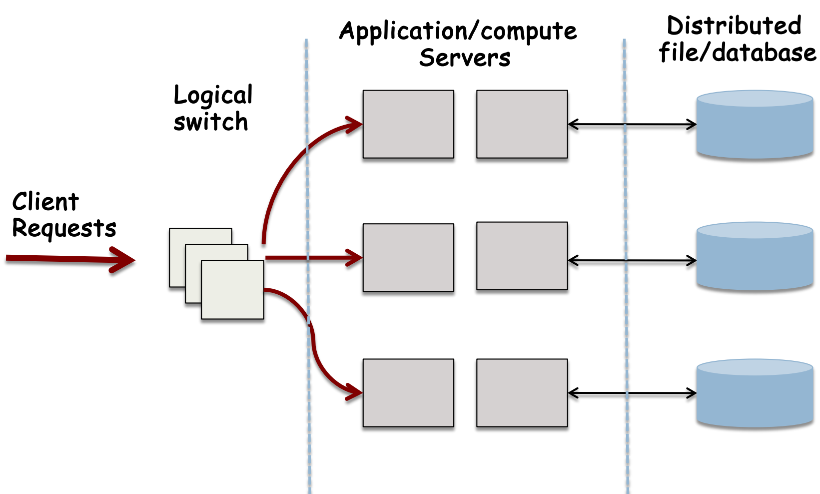
DISTRIBUTED SERVERS

34

Mean time for failures and the premise for distributed servers

- Group several machines together
- Don't rely on the availability of any single machine
- Together, achieve better stability than each component individually
  - The sum is greater than the parts

Server Clusters



Server Clusters

- Switch is also responsible for **load balancing** requests
  - Simplest way to do this is using round-robin
- If there are different services offered within the cluster?
  - Switch needs to dispatch requests appropriately

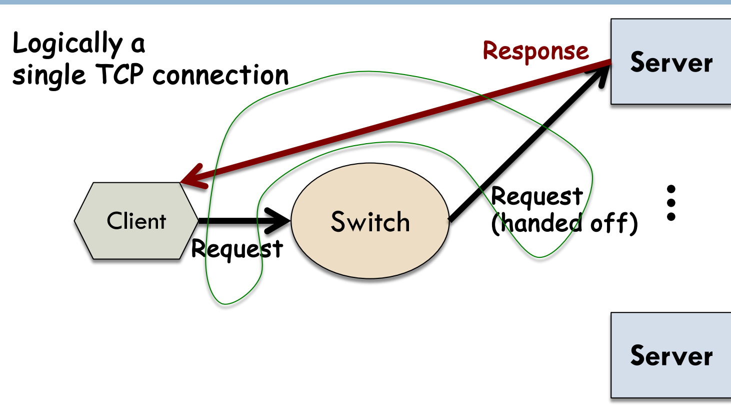
But what about transparency?

- An important consideration is that the server cluster is **transparent**
- Clients typically set up network connections over which requests are sent

But TCP expects an answer from the switch not some arbitrary node

- When server responds to client
  - Inserts **switch's IP address** in source field of the IP packet
- Requires **OS-level modifications**
- Also used in content-aware request distribution

The principle of TCP handoffs



When a cluster offers a single point …

- When there is a failure at that access point?
  - The entire cluster becomes unavailable
- Several access points are typically provided
  - DNS can return **several addresses** all mapped to the same host name
  - Client makes several attempts if there are failures
  - Still requires static access points

Pulls and trade-offs

- Stability
  - Long lived access point
- Flexibility
  - Ability to configure a server cluster including the switch

What would be really nice

- Distributed server with a **dynamically changing** set of machines
- And also varying access points

43

Mobility support in IP version 6 (MIPv6)

- A **mobile node** has a **home-network**
- This node has a **home-address**
- The node has a **home agent**
  - Takes care of traffic to the mobile node while it is away

44

Mobility support in IP version 6 (MIPv6)

- When a mobile node attaches to a **foreign network**
  - Gets a temporary **care-of address**
- Care-of address reported to the home-agent
  - **Forward** all traffic to the mobile node

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT | DISTRIBUTED SERVERS | L10.45

45

Apps communicating with mobile node only see the home address and not the care-of-address

- Offers a stable address for a distributed server
  - A single, unique contact address is initially assigned
- Contact address is server's **lifetime** address

COLORADO STATE UNIVERSITY | Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT | DISTRIBUTED SERVERS | L10.46

46

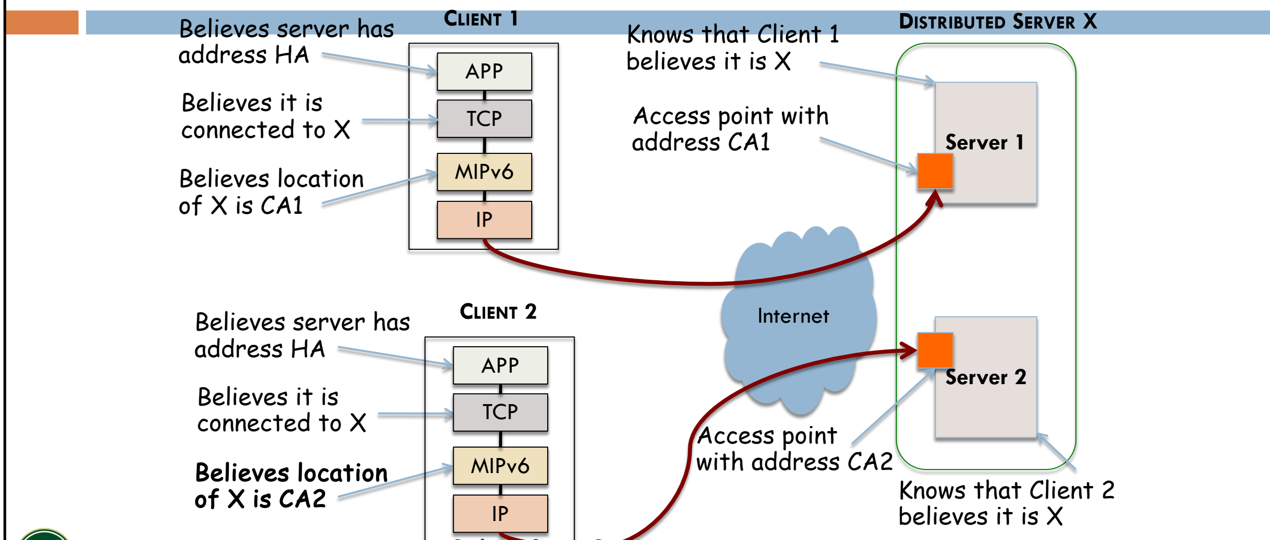
Any node can act operate as the access point

- Record own address as the care-of address
- All traffic will be directed to the access point
- If there's a failure at the access point?
  - Another node takes over
- Potential bottlenecks?
  - Home agent and access point
  - All traffic **must flow** through them

The route optimization feature in MIPv6

- When a mobile node reports its care-of address (CA) to the home-agent (HA)
  - The HA reports the CA to a client
- Client keeps **{HA, CA}**
- Communications will be with the CA
  - Applications can still use the HA
  - MIPv6 protocol stack will translate HA to CA

Depicting Route Optimizations



49

The contents of this slide-set are based on the following references

- *Distributed Systems: Principles and Paradigms. Andrew S. Tanenbaum and Maarten Van der Steen. 2nd Edition. Prentice Hall. ISBN: 0132392275/978-0132392273.* [Chapter 6, 2]
- *Distributed Systems: Concepts and Design. George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair. 5th Edition. Addison Wesley. ISBN: 978-0132143011.* [Chapter 7, 14]

50