Networks

CS 475, Fall 2019
Concurrent & Distributed Systems
Latency

• In client/server model, latency is simply: time between client sending request and receiving response

• What contributes to latency?
  • Latency sending the message
  • Latency processing the message
  • Latency sending the response

• Adding pipelined components -> latency is cumulative

![Latency Diagram]

- Camera sends images: 10ns
- Image Service processes images: 10ns
- Image Service: Phase 1: 5ns, Phase 2: 5ns
- Total latency: 30ns
Throughput

• Measure of the rate of useful work done for a given workload
• Example:
  • Throughput is camera frames processed/second
  • When adding multiple pipelined components -> throughput is the minimum value
Introduce concurrency into our pipeline
Each stage runs in its own thread (or many threads, perhaps)
If a stage completes its task, it can start processing the next request right away
E.g. our system will process multiple requests at the same time
Reducing Latency without lots of $$$

- Approach: use concurrency
- Limited by serial section
Thread Pools

- More sensible to keep a pool of long-lived threads
- Threads assigned short-lived tasks
  - Runs the task
  - Rejoins pool
  - Waits for next assignment
Thread Pool = Abstraction

• Insulate programmer from platform
  – Big machine, big pool
  – And vice-versa

• Portable code
  – Runs well on any platform
  – No need to mix algorithm/platform concerns
Multithreaded Fibonacci

class FibTask implements Callable<Integer> {
    static ExecutorService exec = Executors.newCachedThreadPool();
    int arg;
    public FibTask(int n) {
        arg = n;
    }
    public Integer call() {
        if (arg > 2) {
            Future<Integer> left = exec.submit(new FibTask(arg-1));
            Future<Integer> right = exec.submit(new FibTask(arg-2));
            return left.get() + right.get();
        } else {
            return 1;
        }
    }
}

Parallel calls
Multithreaded Fibonacci

class FibTask implements Callable<Integer> {
    static ExecutorService exec = Executors.newCachedThreadPool();
    int arg;
    public FibTask(int n) {
        arg = n;
    }
    public Integer call() {
        if (arg > 2) {
            Future<Integer> left = exec.submit(new FibTask(arg-1));
            Future<Integer> right = exec.submit(new FibTask(arg-2));
            return left.get() + right.get();
        } else {
            return 1;
        }
    }
}
Fib Work

fib(4) → fib(3) → fib(2) → fib(1)

fib(2) → fib(1)

fib(1) → fib(1)
Today

• HW2 discussion
• Gentle introduction to distributed computation
• Computer networks - what do they mean for us?
• We won’t return to the CompleteableFuture material we didn’t get to last class
• Reminders:
  • Midterm
More Abstractions

- Process + Thread -> one computer
- How can we abstract many computers working together?
- What does that even look like?
Distributed Systems

Model:
Many servers talking through cloud
Distributed Systems

Model:
Servers and Clients talking through cloud
Distributed Systems

Model:
Many clients talking through cloud
Distributed Systems

Model:
Two clients talking through cloud
Why expand to distributed systems?

• Scalability
• Performance
• Latency
• Availability
• Fault Tolerance
Distributed Systems Goals

- **Scalability**
- Performance
- Latency
- Availability
- Fault Tolerance

“the ability of a system, network, or process, to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth.”
Distributed Systems Goals

- Scalability
- **Performance**
- Latency
- Availability
- Fault Tolerance

“is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used.”
Distributed Systems Goals

- Scalability
- Performance
- Latency
- Availability
- Fault Tolerance

“The state of being latent; delay, a period between the initiation of something and the it becoming visible.”
Distributed Systems Goals

- Scalability
- Performance
- Latency
- **Availability**
- Fault Tolerance

“the proportion of time a system is in a functioning condition. If a user cannot access the system, it is said to be unavailable.”

Availability = uptime / (uptime + downtime).

Often measured in “nines”

<table>
<thead>
<tr>
<th>Availability %</th>
<th>Downtime/year</th>
</tr>
</thead>
<tbody>
<tr>
<td>90%</td>
<td>&gt;1 month</td>
</tr>
<tr>
<td>99%</td>
<td>&lt;4 days</td>
</tr>
<tr>
<td>99.9%</td>
<td>&lt;9 hours</td>
</tr>
<tr>
<td>99.99%</td>
<td>&lt;1 hour</td>
</tr>
<tr>
<td>99.999%</td>
<td>5 minutes</td>
</tr>
<tr>
<td>99.9999%</td>
<td>31 seconds</td>
</tr>
</tbody>
</table>
Distributed Systems Goals

• Scalability
• Performance
• Latency
• Availability
• Fault Tolerance

“What kind of faults?

Disks fail
Power supplies fail
Power goes out
Networking fails
Security breached
Datacenter goes offline

“ability of a system to behave in a well-defined manner once faults occur”
More machines, more problems

• Say there’s a 1% chance of having some hardware failure occur to a machine (power supply burns out, hard disk crashes, etc)
• Now I have 10 machines
  • Probability(at least one fails) = 1 - Probability(no machine fails) = 1-(1-.01)^{10} = 10%
  • 100 machines -> 63%
  • 200 machines -> 87%
• So obviously just adding more machines doesn’t solve fault tolerance
More machines, more problems

• PLUS, the network may be:
  • Unreliable
  • Insecure
  • Slow
  • Expensive
  • Limited
Constraints

- Number of nodes
- Distance between nodes
Networks as Abstractions

• A network consists of communication links
• Networks have several “interesting” properties we will look at
  • Latency
  • Failure modes
• What is the abstraction?
Networks as Abstractions

- Stuff goes in, stuff goes out?
- Not a perfect abstraction, because:
  - Speed of light (1 foot/nanosecond)
  - Communication links exist in uncontrolled/hostile environments
  - Communication links may be bandwidth limited (tough to reach even 100MB/sec)
- In contrast to a single computer, where:
  - Distances are measured in mm, not feet
  - Physical concerns can be addressed all at once
  - Bandwidth is plentiful (easily GB/sec)
Networks are Shared

- With processes, we considered how one CPU could be shared between multiple programs running at once
- With networks, communication links are probably shared even more widely
Networks are Shared

- With processes, we considered how one CPU could be shared between multiple programs running at once.
- With networks, communication links are probably shared even more widely.
Network as Abstractions

• What do we send, what gets received?
• At the lowest level, we call what gets sent frames
• Each frame is limited in size
  • Ethernet: max 1522 bytes
• Frame is packed with source/destination info into a packet
• Network knows what to do with packets to get them to their destination
Networks as Abstractions

PH: Internet packet header
FH: LAN frame header
Packet Switching Delays

- As these packets flow through a network and are routed, we might see delays due to:
  - Propagation (traveling across the link, speed of light, etc)
  - Transmission delay (big packets take longer to transmit)
  - Processing delay (once switch sees packet, might be slow to process)
  - Queuing delay (link might be busy)
Packet Loss

• Some packets could be delayed, others might never reach their target, due to:
  • Buffers overflowing (e.g. on switch)
• Networks are usually considered **best-effort**
  • Aka third-class mail
  • We’ll try to get your packet there, but if it doesn’t, sorry.
• Solved by requiring recipient to send a confirmation message was received
  • If no confirmation received, assume didn’t get sent
• What happens to duplicates?
  • Each message includes a unique ID, can be discarded if duplicate received
Resending Packets

send request, set timer

A

request 1

B

time

forwarder discards request packet.

request 1

response 1

request 2

response 2

set timer

reset timer

set timer

reset timer

Fig © Saltzer & Kaashoek
Resending Packets

send request, set timer

request 3

overloaded forwarder discards response 3

Fig © Saltzer & Kaashoek
Resending Packets

A

send request, set timer

request 4

B

receive response

receive response, reset timer

duplicate arrives at B

B sends response

duplicate gets delayed

set timer

request 4

response 4

response 4
Resending Packets

• That ID is **really** important to put on the packets!
• Note: it works, but can result in **lots** of duplicate packets sent back and forth
• Also, note: no guarantee that packets are delivered in order!
• Obviously, we don’t think or care about packets
• We think and care about sending data!
• We want abstractions, like RPC (Remote Procedure Calls)
• Abstractions (try to) hide the complexity of what’s below them
• Next class: all RPC
3 Layer Abstraction

- The typical network abstraction model has 7 layers
- Take CS 455 if you want to know more about these
- We’ll think about 3 abstraction layers, and really focus on the top one

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link layer</td>
<td>Physical links: care about how to deliver packets</td>
</tr>
<tr>
<td>Network layer</td>
<td>Figures out where to send packets</td>
</tr>
<tr>
<td>End-to-End layer</td>
<td>Handles packet loss, etc. Translates from application-data to packets, implements a protocol</td>
</tr>
</tbody>
</table>
Transport Protocols

- Anything in the end-to-end layer is likely built on top of some lower level protocol (more abstractions)
- TCP, or UDP
- Data integrity (checksumming)
- Ordering control
- Flow control (not worrying about congestion)
Reminder: Leaky Abstractions

- From this lecture, you should have found out that networks:
  - Can vary in
    - Data rates
    - Propagation, transmission, queuing and processing delays
  - Traverse hostile environments and may corrupt data or stop working
  - Even best-effort networks have:
    - Variable delays, transmission rates, can discard packets, duplicate packets, have a maximum packet length, can reorder packets
  - Even if using TCP, this can still show up!
    - Messages might still arrive late
Sockets as an Abstraction

• Simplest way to build our end-to-end layer is using a **socket**, which gives us an interface to TCP or UDP
• Socket looks **just** like reading/writing to a file (e.g. file descriptor in C, InputStream in Java)
• Sockets are identified by:
  • IP address - identifies the device on the network
  • Port number - identifies the application on the device
Preview for Next Class

**Spoiler alert:** You can not tell the difference in a distributed system between a computer failing and network being arbitrarily slow!

...well, I can still talk to these guys so I guess internet is ok
This work is licensed under a Creative Commons Attribution-ShareAlike license

- This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/
- You are free to:
  - Share — copy and redistribute the material in any medium or format
  - Adapt — remix, transform, and build upon the material
  - for any purpose, even commercially.
- Under the following terms:
  - Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - ShareAlike — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.