WEB CACHING

- Many clients transfer the same information
- Generates redundant server and network load
- Also, clients may experience high latency

WHY WEB CACHING?

- Motivation for placing content closer to client:
  - User gets better response time
  - Content providers get happier users
  - Network gets reduced load
- Why does caching work? Exploits locality of reference
- How well does caching work?
  - Very well, up to a limit
  - Large overlap in content
  - But many unique requests
CACHING WITH REVERSE PROXIES
- Cache data close to origin server → decrease server load
  - Typically done by content providers
  - Client thinks it is talking to the origin server (the server with content)
  - Does not work for dynamic content

CACHING WITH FORWARD PROXIES
- Cache close to clients → less network traffic, less latency
  - Typically done by ISPs or corporate LANs
  - Client configured to send HTTP requests to forward proxy
  - Reduces traffic on ISP-1’s access link, origin server, and backbone ISP

CACHING & LOAD-BALANCING: OUTSTANDING PROBLEMS
- Problem ca. 2002: How to reliably deliver large amounts of content to users worldwide?
  - Popular event: “Flash crowds” overwhelm (replicated) web server, access link, or back-end database infrastructure
  - More rich content: audio, video, photos
  - Web caching: Diversity causes low cache hit rates (25−40%)

CONTENT DISTRIBUTION NETWORKS
- Proactive content replication
  - Content provider (e.g. CNN) pushes content out from its own origin server
  - CDN replicates the content
  - On many servers spread throughout the Internet
  - Updating the replicas
  - Updates pushed to replicas when the content changes

REPLICA SELECTION: GOALS
- Live server
  - For availability
- Lowest load
  - To balance load across the servers
- Closest
  - Nearest geographically, or in round-trip time
- Best performance
  - Throughput, latency, reliability...

OUTLINE
1. Web caching
2. Content-distribution networks
   - Featuring Akamai
3. Web load balancing
4. Availability

PROBLEMS
- Requires continuous monitoring of liveness, load, and performance
AKAMAI STATISTICS

- Distributed servers
- Servers: ~100,000
- Networks: ~1,000
- Countries: ~70
- Many customers
  - Apple, BBC, FOX, GM
  - IBM, MTV, NASA, NBC,
  - NFL, NPR, Puma, Red Bull, Rutgers, SAP, ...

- Client requests
- 20+M per second
- Half in the top 45 networks
- 20% of all Web traffic worldwide

HOW AKAMAI USES DNS

1. GET
2. index.html
3. cache.cnn.com
4. GET
5. index.html
6. cache.cnn.com
7. GET
8. /foo.jpg
9. GET
10. /foo.jpg

End user

DNS TLD server

Akamai global DNS server

Akamai regional DNS server

Nearby Akamai cluster

DNS lookup

cache.cnn.com

ALIAS:
g.akamai.net

DNS lookup

g.akamai.net

ALIAS:
a73.g.akamai.net

DNS lookup

a73.g.akamai.net

Address 1.2.3.4

GET /foo.jpg
Host: cache.cnn.com

End user

End user
HOW AKAMAI USES DNS

HTTP

cnn.com (content provider)

DNS TLD server

Akamai global DNS server

Akamai regional DNS server

Nearby Akamai cluster

End user

GET /foo.jpg

Host: cache.cnn.com

HOW AKAMAI USES DNS

HTTP

cnn.com (content provider)

DNS TLD server

Akamai global DNS server

Akamai regional DNS server

Nearby Akamai cluster

End user

GET foo.jpg

End user

HOW AKAMAI WORKS: CACHE HIT

HTTP

cnn.com (content provider)

DNS TLD server

Akamai global DNS server

Akamai regional DNS server

Nearby Akamai cluster

End user

GET /foo.jpg

Host: cache.cnn.com

MAPPING SYSTEM

- Equivalence classes of IP addresses
- IP addresses experiencing similar performance
- Quantify how well they connect to each other
- Collect and combine measurements
  - Ping, traceroute, BGP routes, server logs
  - e.g., over 100 TB of logs per days
  - Network latency, loss, throughput, and connectivity

ROUTING CLIENT REQUESTS WITH THE MAP

- Map each IP class to a preferred server cluster
- Based on performance, cluster health, etc.
- Updated roughly every minute
  - Short, 60-sec DNS TTLs in Akamai regional DNS accomplish this
- Map client request to a server in the cluster
  - Load balancer selects a specific server
  - e.g., to maximize the cache hit rate

ADAPTING TO FAILURES

- Failing hard drive on a server
  - Suspends after finishing “in progress” requests
- Failed server
  - Another server takes over for the IP address
  - Low-level map updated quickly (load balancer)
- Failed cluster, or network path
  - High-level map updated quickly (ping/traceroute)
TAKE-AWAY POINTS: CDNS

- Content distribution is hard
- Many, diverse, changing objects
- Clients distributed all over the world

- Moving content to the client is key
  - Reduces latency, improves throughput, reliability

- Content distribution solutions evolved:
  - Load balancing, reactive caching, to
  - Proactive content distribution networks

OUTLINE

1. Web caching
2. Content distribution networks
   - Featuring Akamai
3. Web load balancing
4. Availability

HOSTING: MULTIPLE MACHINES PER SITE

- Problem: Overloaded popular web site
  - Replicate the site across multiple machines
    - Helps to handle the load
  - Want to direct client to a particular replica. Why?
    - Balance load across server replicas
  - Solution #1: Manual selection by clients
    - Each replica has its own site name
    - Some Web page lists replicas (e.g., by name, location), asks clients to click link to pick

HOSTING: LOAD-BALANCER APPROACH

- Solution #2: Single IP address, multiple machines
  - Run multiple machines behind a single IP address
  - Ensure all packets from a single TCP connection go to the same replica

HOSTING: DNS REDIRECTION APPROACH

- Solution #3: Multiple IP addresses, multiple machines
  - Same DNS name but different IP for each replica
    - DNS server returns IP addresses “round robin”

HOSTING: SUMMARY

- Load-balancer approach
  - No geographical diversity ✗
  - TCP connection issue ✗
  - Does not reduce network traffic ✗

- DNS redirection
  - No TCP connection issues ✓
  - Simple round-robin server selection
    - May be less responsive ✗
  - Does not reduce network traffic ✗
FACTORS OF VARIABLE RESPONSE TIME

- Shared Resources (Local)
  - CPU cores
  - Processors caches
  - Memory bandwidth

- Global Resource Sharing
  - Network switches
  - Shared file systems
  - Daemons
  - Scheduled Procedures

FACTORS OF VARIABLE RESPONSE TIME

- Maintenance Activities
  - Data reconstruction in distributed file systems
  - Periodic log compactions in storage systems
  - Periodic garbage collection in garbage-collected languages

- Queueing
  - Queueing in intermediate servers and network switches

FACTORS OF VARIABLE RESPONSE TIME

- Power Limits
  - Throttling due to thermal effects on CPUs

- Garbage Collection
  - Random access in solid-state storage devices

- Energy Management
  - Power saving modes
  - Switching from inactive to active modes

COMPONENT VARIABILITY AMPLIFIED BY SCALE

- Latency variability is magnified at the service level.

REQUEST LATENCY MEASUREMENT

<table>
<thead>
<tr>
<th>50thile latency</th>
<th>90thile latency</th>
<th>99thile latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1ms</td>
<td>5ms</td>
<td>10ms</td>
</tr>
<tr>
<td>100ms</td>
<td>320ms</td>
<td>700ms</td>
</tr>
<tr>
<td>300ms</td>
<td>1400ms</td>
<td></td>
</tr>
</tbody>
</table>

- Key Observation:
  - 5% servers contribute nearly 50% latency.

KEY QUESTION

- What would you do with that 5% servers which contribute 50% latency?

Eliminate all interactive request latencies.
Live with But it's infeasible to eliminate all variations!
**REDUCING COMPONENT VARIABILITY**

- Differentiating Service Classes
- Differentiate non-interactive requests
- High Level Queuing
- Keep low level queues short
- Reduce Head-of-line Blocking
- Break long-running requests into a sequence of smaller requests.
- Synchronize Disruption
- Do background activities altogether.

**LIVING WITH LATENCY VARIABILITY**

- Within Request Short-Term Adaptations
  - Handles latency of 10+ ms
  - Takes the advantage of redundancy
- Hedged Requests
  - Send redundant requests.
  - Use the results from whichever replica responds first.
  - The overhead can be further reduced by tagging them as lower priority than primary requests.

**WITHIN REQUEST SHORT-TERM ADAPTATIONS**

- Tied Requests
- Hedged requests with cancellation mechanism.

<table>
<thead>
<tr>
<th>Mostly idle cluster</th>
<th>With concurrent load</th>
<th>Mostly idle cluster</th>
<th>With concurrent load</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50/50</td>
<td>10ms</td>
<td>30ms</td>
<td>10ms</td>
</tr>
<tr>
<td></td>
<td>(10%)</td>
<td>(24%)</td>
<td>(21%)</td>
</tr>
<tr>
<td>80/20</td>
<td>30ms</td>
<td>60ms</td>
<td>30ms</td>
</tr>
<tr>
<td></td>
<td>(24%)</td>
<td>(37%)</td>
<td>(37%)</td>
</tr>
<tr>
<td>90/10</td>
<td>60ms</td>
<td>120ms</td>
<td>60ms</td>
</tr>
<tr>
<td></td>
<td>(37%)</td>
<td>(50%)</td>
<td>(37%)</td>
</tr>
<tr>
<td>99/10</td>
<td>90ms</td>
<td>180ms</td>
<td>90ms</td>
</tr>
<tr>
<td></td>
<td>(50%)</td>
<td>(52%)</td>
<td>(52%)</td>
</tr>
</tbody>
</table>

**LARGE INFORMATION RETRIEVAL SYSTEMS**

- Google search engine
- No certain answers
- “Good Enough”
  - Google’s IR systems are tuned to occasionally respond with good-enough results when an acceptable fraction of the overall corpus has been searched.

**LARGE INFORMATION RETRIEVAL SYSTEMS**

- Canary Requests
- Some requests exercising an untested code path may cause crashes or long delays.
- Send requests to one or two leaf servers for testing.
- The remaining servers are only queried if the root gets a successful response from the canary in a reasonable period of time.

**HARDWARE TRENDS AND THEIR EFFECTS**

- Hardware will only be more and more diverse
  - So tolerating variability through software techniques are even more important over time.
- Higher bandwidth reduces per-message overheads.
  - It further reduces the cost of tied requests (making it more likely that cancellation messages are received in time).
CONCLUSION

- Variability of latency exists in large-scale services.
- Live with variable request latency
  - Infeasible to eliminate variable request latency in large-scale services.
- The importance of these techniques will only increase. (scale of services getting larger)
- Techniques for living with variable latency turn out to increase system utilization

OUTLINE

1. Web caching
2. Content-distribution networks
   - Featuring Akamai
3. Web load balancing
4. Availability

AVAILABILITY METRICS

- Mean time between failures (MTBF)
- Mean time to repair (MTTR)
- Availability = (MTBF – MTTR)/MTBF
- Example:
  - MTBF = 10 minutes
  - MTTR = 1 minute
  - A = (10 – 1) / 10 = 90% availability
- Can improve availability by increasing MTBF or by reducing MTTR
  - Ideally, systems never fail but much easier to test reduction in MTTR than improvement in MTBF

HARVEST AND YIELD

- \(\text{yield} = \frac{\text{queries completed}}{\text{queries offered}}\)
- In some sense more interesting than availability because it focuses on client perceptions rather than server perceptions
  - If a service fails when no one was accessing it…
- \(\text{harvest} = \frac{\text{data available}}{\text{complete data}}\)
  - How much of the database is reflected in each query?
  - Should faults affect yield, harvest or both?

DQ PRINCIPLE

- \(\text{Data per query} \times \text{queries per second} \rightarrow \text{constant}\)
- At high levels of utilization, can increase queries per second by reducing the amount of input for each response
- Adding nodes or software optimizations changes the constant

PERFORMANCE “HOCKEY STICK” GRAPH

- Response time vs. system load
- Knee of the graph indicates the point of diminishing returns
GRACEFUL DEGRADATION

• Cost-based admission control
  • Search engine denies expensive query (in terms of D)
  • Rejecting one expensive query may allow multiple cheaper ones to complete

• Priority-based admission control
  • Stock trade requests given different priority relative to, e.g., stock quotes
  • Reduced data freshness
  • Reduce required data movement under load by allowing certain data to become out of date (again stock quotes or perhaps book inventory)

MEMCACHE

• Popular in-memory cache
  • Simple get() and put() interface
  • Useful for caching popular or expensive requests

```matlab
function get_foo(foo_id)    % get_foo: Get a foo with id $foo_id$
    foo = memcached_get("foo": foo_id) % get_foo does a memcached_get
    return foo if defined foo

foo = fetch_foo_from_database(foo_id) % fetch_foo_from_database returns foo
memcached_set("foo": foo_id, foo) % memcached_set sets foo in memcached
return foo
end
```

MEMCACHED DATA FLOW

TAIL TOLERANCE: PARTITION/AGGREGATE

• Consider distributed memcached cluster
  • Single client issues request to S memcached servers
  • Waits until all S are returned
  • Service time of a memcached server is normal $\mu = 90 \text{us}$, $\sigma = 7 \text{us}$
  • Roughly based on measurements from my former student

MATLAB SIMULATION

COMPARING MATLAB TO THE REAL WORLD
TAIL TOLERANCE: DEPENDENT/SEQUENTIAL PATTERN

• Consider iterative lookups in a service to build a web page
  • E.g., Facebook
• Issue request, get response, based on response, issue new request, etc...
• How many iterations can we issue within a deadline D?