Application Architecture

Lecture 4
Srinivas Narayana

http://www.cs.rutgers.edu/~sn624/553-S23
Monoliths

Language frameworks
Changes coupled
Coordination overheads

HTTP server
CGI
App

Releases, transient functions, etc.
Scaling, troubleshooting incidents, …
Microservices

- Language of choice
- Independently upgrade and deploy
- Independent data models

HTTP server

Distinct from a software library. Out of process.

Must be explicitly related
How to split?

Business Capabilities
Churn Boundaries
Fleeting?
Who should know (or not)?
Changing together?

Heterogeneity in resource use
Refactoring common functionality
Salient new concerns!

Communication Failures
Lots of waiting. Increasing failures.

Remote Procedure Call (RPC)

Serialization format (e.g., protobufs)

```
struct customer {
    string name;
    int customer_id;
    ...
}
```

```
011010101010...
```

```
JSON
XML
```

```
struct customer {
    string name;
    int customer_id;
    ...
}
```
Communication

Synchronous blocking (request-response)

Asynchronous request-response

Event streaming

Shared data

Can do useful work while waiting (but still need timeouts)

callback()
Cost of communication: Performance

Figure 4: 22-27% of WCC cycles are spent in different components of "datacenter tax".

Profiling a warehouse-scale computer (Google). ISCA’15.
Cost of comm: Hotspot spreading

A. NGINX Saturation

B. Memcached Backpressuring NGINX

Deathstarbench. ASPLOS’19.
Cost of comm: high level failure handling

(Soft or hard)

Connection issue

Circuit breaker

Timeout

Fail immediately

trip

Timeout

Timeout
Microservices aren’t always good

• Just a technology. Look at problems first
• Observability
• Deployment automation
• Integration: refactoring service boundaries is hard
• How significant are dev coordination overheads?
• Complexity
Partition-Aggregate
Processing interactive search queries
Web search: some numbers (circa 2003)

- 10s of terabytes of web corpus data
  - Read 100s of megabytes per query

- 10s of billions of CPU instructions per query

- Data accessed depends on the query; hard to predict

- Cannot process on a single machine within acceptable time
Quick Review: Compute & Memory Org

- Instruction pipeline
- L1 I-cache
- L1 D-cache
- L2 cache
- L3 cache
- Main memory
- I-TLB

Fetch (on hit)

Slower but larger

Registers

Branch predictors (out of order + speculative)

Retire

Compute (single threaded core)

Memory hierarchy
Measurements from one (index) server

- Not too fast single-threaded
  - Data dependencies
  - Branches often mispredicted
- Small instruction memory footprint
- Data locality within a block, but not across blocks
  - Can’t drive high single threaded performance

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles per instruction</td>
<td>1.1</td>
</tr>
<tr>
<td>Ratios (percentage)</td>
<td></td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5.0</td>
</tr>
<tr>
<td>Level 1 instruction miss*</td>
<td>0.4</td>
</tr>
<tr>
<td>Level 1 data miss*</td>
<td>0.7</td>
</tr>
<tr>
<td>Level 2 miss*</td>
<td>0.3</td>
</tr>
<tr>
<td>Instruction TLB miss*</td>
<td>0.04</td>
</tr>
<tr>
<td>Data TLB miss*</td>
<td>0.7</td>
</tr>
</tbody>
</table>

* Cache and TLB ratios are per instructions retired.

Web search for a planet, MICRO’03.
How to use parallelism?

- Few fast cores with high-speed interconnect
- Or more slow cores?
- Cost per query processed?
- Power efficiency?

(hyperthreaded or on-chip multicore)
Data parallelism

- Significant parts of computation are independent over **shards** of data
  - Fast interconnects not as critical
  - **Stateless**, no coordination within a request
- Different requests are independent
  - Use parallelism across requests
  - Shard itself can be replicated for throughput
- Need lower latency?
- Compensate slow cores with smaller shard (add more shards)
- Turn throughput into latency advantage
Internet architecture: Review

Routing
Software/hardware organization at hosts

- **Application**: useful user-level functions
- **Transport**: provide guarantees to apps
- **Network**: best-effort global pkt delivery
- **Link**: best-effort local pkt delivery

**Communication functions** broken up and “stacked”

Each layer depends on the one below it.

Each layer supports the one above it.

The interfaces between layers are well-defined and standardized.
Routing
Two key network-layer functions

- **Forwarding**: move packets from router’s input to appropriate router output
- **Routing**: determine route taken by packets from source to destination
  - routing algorithms
- The network layer solves the routing problem.

Analogy: taking a road trip

- **Forwarding**: process of getting through single exit
- **Routing**: process of planning trip from source to destination

The network layer runs everywhere
Control/Data Planes

Data plane = Forwarding
• local, per-router function
• determines how datagram arriving on router input port is forwarded to router output port

Control plane = Routing
• network-wide logic
• determines how datagram is routed along end-to-end path from source to destination endpoint
• two control-plane approaches:
  • Distributed routing algorithm running on each router
  • Centralized routing algorithm running on a (logically) centralized machine
Distributed routing

**Control plane**

Traditional *routing protocols*: per route-change processing
(~ a few tens of seconds)

**Data plane**

per-packet processing
(~ tens of nanoseconds)

Values in arriving packet header, i.e., destination IP address
The Internet is a large federated network.
The Internet is a large federated network

Several autonomously run organizations (AS’es): No one “boss”
Organizations cooperate, but also compete

e.g., AT&T has little commercial interest in revealing its internal network structure to Verizon.
The Internet is a large **federated** network

Several autonomously run organizations: No one “boss”

Organizations cooperate, but also **compete**

Message exchanges must not reveal internal network details.

Algorithm must work with “incomplete” information about its neighbors’ internal topology.
The Internet is a **large** federated network

Internet today: > 70,000 unique autonomous networks
Internet routers: > 800,000 forwarding table entries

Keep messages & tables as small as possible. **Don’t flood**

Algorithm must be **incremental**: don’t recompute the whole table on every message exchanged.
Inter-domain Routing

• The Internet uses Border Gateway Protocol (BGP)
• All AS’es speak BGP. It is the glue that holds the Internet together
• BGP is a path vector protocol
(1) BGP Messages

- Routing **Announcements** or **Advertisements**
  - “I am here” or “I can reach here”
  - Occur over a TCP connection (**BGP session**) between routers

- Route announcement = destination + attributes
  - Destination: IP prefix

- Route Attributes:
  - AS-level path
  - Next hop
  - Several others: origin, MED, community, etc.

- An AS promises to use advertised path to reach destination
- Only route changes are advertised after BGP session established
(2) BGP algorithm

- A BGP router does not consider every routing advertisement it receives by default to make routing decisions!
  - An import policy determines whether a route is even considered a candidate
- Once imported, the router performs route selection
- A BGP router does not propagate its chosen path to a destination to all other AS’es by default!
  - An export policy determines whether a (chosen) path can be advertised to other AS’es and routers

Business policy considerations drive BGP. NOT efficiency considerations.
Policy arises from business relationships

• Customer-provider relationships:
  • E.g., Rutgers is a customer of AT&T

• Peer-peer relationships:
  • E.g., Verizon is a peer of AT&T

• Business relationships depend on where connectivity occurs
  • “Where”, also called a “point of presence” (PoP)
  • e.g., customers at one PoP but peers at another
  • Internet-eXchange Points (IXPs) are large PoPs where ISPs come together to connect with each other (often for free)
A, B, C are provider networks
X, W, Y are customers (of provider networks)
X is dual-homed: attached to two networks
policy to enforce: X does not want to route from B to C via X
So, X will not announce to B a route to C

Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A, B, C are provider networks
- X, W, Y are customers (of provider networks)
- X is dual-homed: attached to two networks
- policy to enforce: X does not want to route from B to C via X
- So, X will not announce to B a route to C
Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- A announces path Aw to B and to C
- B will not announce BAw to C:
  - B gets no “revenue” for routing CBAw, since none of C, A, w are B’s customers
- C will route CAw (not using B) to get to w
Suppose an ISP wants to minimize costs by avoiding routing through its providers when possible.

- Suppose C announces path Cy to x
- Further, y announces a direct path (“y”) to x
- Then x may choose not to import the path Cy to y since it has a peer path (“y”) towards y
Q2. BGP Route Selection

- When a router imports more than one route to a destination IP prefix, it selects route based on:
  1. **local preference value** attribute (import policy decision -- set by network admin)
  2. shortest AS-PATH
  3. closest NEXT-HOP router
Problems with BGP

• Not designed for efficiency

  1. local preference value attribute (import policy decision -- set by network admin)
  2. shortest AS-PATH
  3. closest NEXT-HOP router

• Only a single path per destination

• Slow to converge after a change

• Vulnerable to bugs & malice

Approaches to bring flexibility:
Flexible control logic for path selection
(Google, Facebook)
Detour/overlay routing (Akamai)

Nothing to do with path length, delay, or available capacity.