Today's Objectives

- Services

Review

- Compare and contrast threads and processes
- What problem is caused by light-weight threads?
  - How do we solve that problem?
- What are the benefits and challenges of network services?

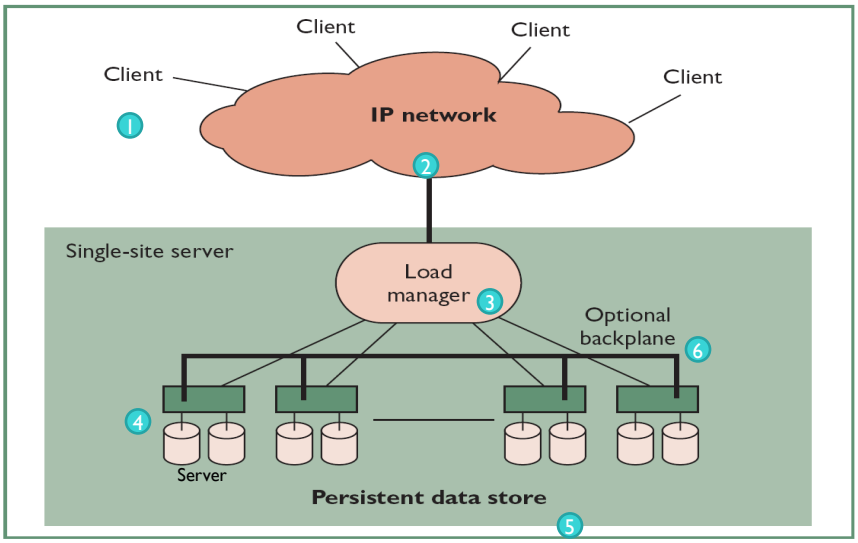
Giant-Scale Services

- Paper does not address
  - Service monitoring, configuration, QoS, security, logging, and log analysis
  - Wide-area replicated services
  - Write-intensive services
  - Database management systems
- Challenges for network services:
  - High availability
    - Critical in today's Internet-dependent society
    - Each second of downtime = lots of lost money and revenue
  - Evolution
  - Growth

Buzzword: Commodity Components

- Defn: [relatively] cheap components, widely available, interchangeable
  - "Off-the-shelf" components
- Allow creation of [relatively] cheap clusters of machines that work together
- In contrast to specialty hardware, e.g., supercomputers

Network Service Components



Clusters as Building Blocks

- No alternative to clusters for building network services that can scale to global use
- Cluster benefits:
  - Incremental scalability
    - Adding one machine typically linearly improves performance
  - Independent components
  - Cost and performance

Switch

- Forward traffic to appropriate devices, typically on the LAN
  - **System Area Network** (SAN)
    - still LAN
    - higher connectivity, performance
- Layer 2 – Mac/Ethernet Address
- Layer 3 – IP address
- Backplane
  - On System Area Network
  - Direct client queries to appropriate server

Buzzword Bingo

- QoS: Quality of Service
- DBMS: Database Management System
- RAID: Redundant Array of Independent Disks
  - More later
- Web Cache
- Total seeks per second

Load Management Discussion

- What is load management?
- What are some approaches to load management and their benefits and limitations?
- What do “L4” and “L7” refer to?
- Violation of end to end argument?

Layer 4 vs Layer 7 Routers

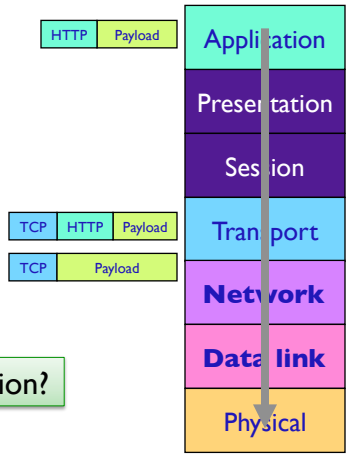
- Distinction: to what information does the router have access?

Layer 4 vs Layer 7 Routers

- Distinction: to what information does the router have access?

Payload: data to be sent

How TCP views the message:

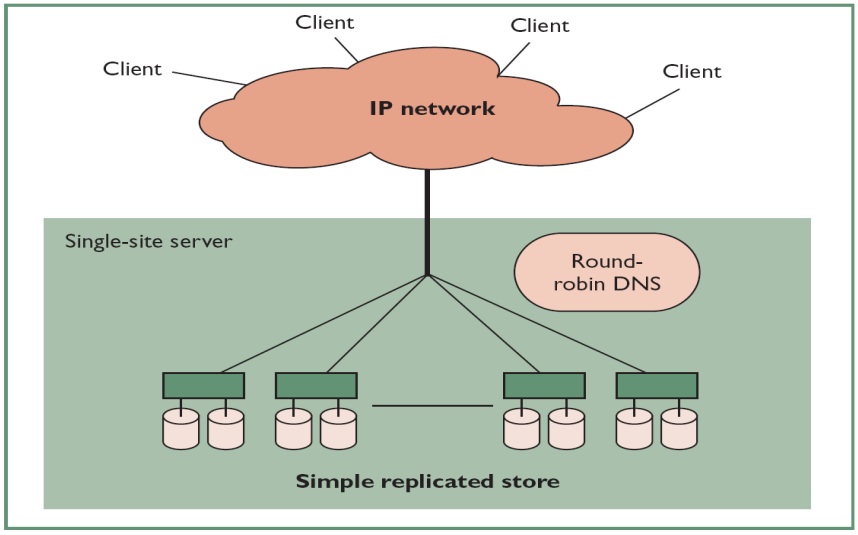


Benefits of having the HTTP information?

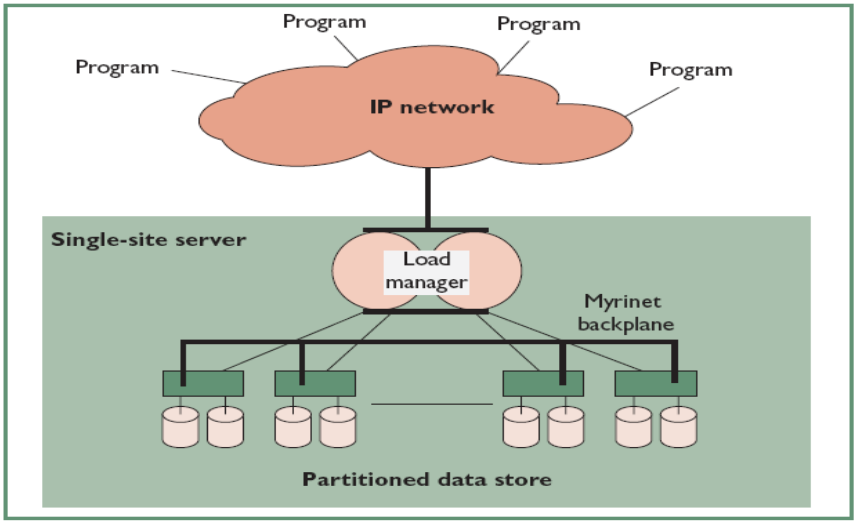
Load Management

- Started with "round-robin" DNS in 1995
  - Map hostname to multiple IP addresses, hand out particular mapping in a round robin fashion to clients
  - Does not hide failure or inactive servers
  - Exposes structure of underlying service
- L4 and L7 switches can inspect TCP session state or HTTP session state
  - Perform mapping of requests to backend servers based on dynamically changing membership information

Load Management Option 1: Service Replication



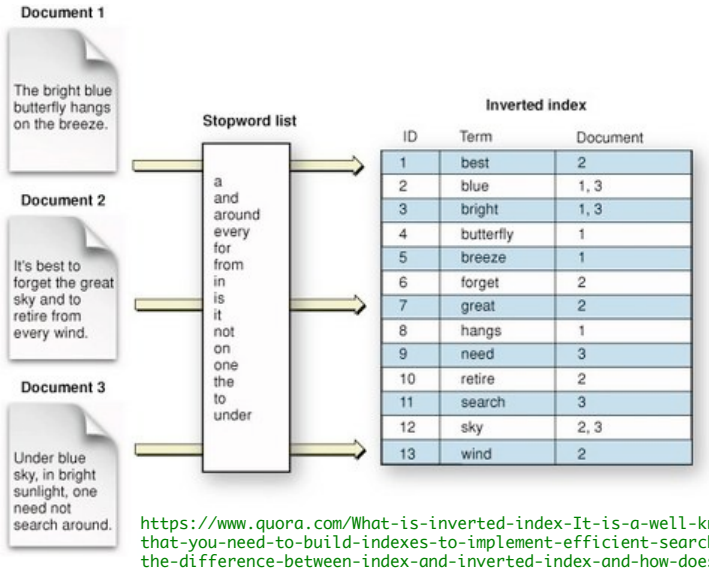
Load Management Option 2: Service Partitioning



Case Study: Search

- Map keyword to a set of documents containing that word
  - Optionally rank document set in decreasing relevance
    - E.g., PageRank from Google
- Need a web crawler to build *inverted index*
  - Data structure that maps keywords to list of all documents that contains that word

Inverted Index



https://www.quora.com/What-is-inverted-index-It-is-a-well-known-fact-that-you-need-to-build-indexes-to-implement-efficient-searches-What-is-the-difference-between-index-and-inverted-index-and-how-does-one-build-inverted-index

Case Study: Search

- Map keyword to a set of documents containing that word
  - Optionally rank document set in decreasing relevance
    - E.g., PageRank from Google
- Need a web crawler to build *inverted index*
  - Data structure that maps keywords to list of all documents that contains that word
- Multi-word search
  - Perform *join* operation across individual inverted indices
- **Where to store individual inverted indices?**
  - Too much storage to place all on each machine (esp if also need to have portions of document avail as well)

Partitioning Keywords in Search

- Think about keywords as columns and documents as rows
- Vertical partitioning
  - Split inverted index across multiple nodes (nodes=data storage devices)
  - Each node contains as much of index as possible for *a particular* keyword
  - Essentially like reducing the number of columns in table, and using extra tables to store remaining columns
- Horizontal partitioning
  - Each node contains portion of inverted index for *all* keywords
  - Have to visit every node in system to perform full join (or search)
  - Essentially like splitting table up into multiple tables (with same number of columns) by putting different (complete) rows in different tables

Replication vs Partitioning?

- What is replication?
- What is partitioning?
- What are their tradeoffs?
- What should you use when?

Replication versus Partitioning

- Replication
  - Any replica can serve any request
  - Failure reduces system capacity but not data availability
  - Must make sure replicas are kept in-sync
- Partitioning
  - Nodes are no longer identical so certain requests need to be sent to individual nodes
  - No need for coherence traffic for syncing data
  - Failure reduces data availability and may reduce capacity
- Optimal solution? Which is better?

Availability Metrics Discussion

- What are some availability metrics?
- Which are the most important?
- What should we do to improve availability?

Availability Metrics

- Uptime: fraction of time service is handling traffic (usually measured in "*nines*")
- Mean time between failures (MTBF)
- Mean time to repair (MTTR)
- Availability = (MTBF – MTTR)/MTBF
- Can improve availability by increasing MTBF or by reducing MTTR
  - Ideally, systems never fail but easier to test reduction in MTTR than improvement in MTBF

Harvest and Yield

- *yield = queries completed/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…
- *harvest = data available/complete data*
  - How much of the database is reflected in each query?
- Should faults affect yield, harvest, or both?

DQ Principle

- Data per query * Queries per second → constant
  - System's overall capacity has a particular physical bottleneck

DQ Principle

- Data per query * Queries per second → constant
  - System's overall capacity has a particular physical bottleneck
- DQ is amount of data that has to be moved per second on average
- At high levels of utilization, can increase queries per second by reducing the amount of input for each response (reducing data per query)
- Adding nodes or software optimizations changes the constant

Graceful Degradation Discussion

- What is graceful degradation?
- What are some approaches to GD?

Graceful Degradation

- Hard to avoid complete system saturation
- Peak to average ratio of load for giant-scale systems varies from 1.6:1 to **6**:1
- Single-event bursts can mean 1 to 3 orders of magnitude increase in load (9/11/2001)
- Power failures and natural disasters are not independent, severely reducing capacity
  - How can we avoid correlated failures like this?
- Under heavy load we can limit capacity (queries/sec) to maintain harvest or sacrifice harvest to improve capacity. Which is better?

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)
- Which technique do you prefer as an end user? as a system maintainer?

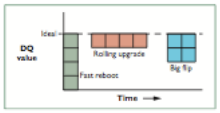
Online Evolution and Growth Discussion

- What are the issues with online evolution and growth?
- What are some approaches? What are their tradeoffs?

Online Evolution and Growth

- Internet services undergo rapid development with the frequent release of new products and features
- Rapid release means that software is released in unstable state with known bugs
  - Goal: acceptable MTBF, low MTTR, no cascading failures
- Beneficial to have *staging* area such that both new and old system can coexist on a node simultaneously
  - Otherwise, will have to transfer new software after taking down old software → increased MTTR
  - Also makes it easier to switch back to old version in case of trouble

Online Evolution and Growth



- **Fast reboot**
  - Simultaneously "reboot" all machines to new version
  - Simple but guaranteed downtime
- **Rolling upgrade**
  - Upgrade one node at a time in "wave" moving across cluster
  - Old and new versions must be compatible because they will coexist (hard in practice)
- **Big flip**
  - Update one half at a time
  - Remove one half of system from view of load balancing switch
  - Wait for existing connections to complete
  - Upgrade this half with new software
  - Atomically flip load balancing switch to upgraded software

Looking Ahead

- Web Server