Lessons from Giant-Scale Services and Load Balancing Switches

George Porter
CSE 124
January 27, 2015
Announcements

- Project 1 submission website:
  - http://vm134.sysnet.ucsd.edu/
  - (This link will be put on the course webpage)
- Upload in .tar.gz format
  - Instructions will be put on Piazza today
Configuration channel: AC

Question 1:
  • Are you a human being?
    A = Yes!
    B = No
Giant-Scale Services

- Challenges for network services:
  - High availability
    Critical in today’s environment: $1000/sec of lost revenue during downtime
  - Evolution
  - Growth

- This paper does not address
  - Service monitoring, configuration, QoS, security, logging, and log analysis
  - Wide-area replicated services
  - Write intensive services
Benefits of Network Services

- Access anywhere, anytime
- Availability via multiple devices
- Groupware support
  - Calendaring, teleconferencing, messaging, etc.
- Lower overall cost
  - Multiplex infrastructure over active users
  - Dedicated resources are typically 98% idle
  - Central administrative burden
- Simplified service updates
  - Update the service in one place, or 100 million?
Network Service Components

Single-site server

Load manager

Optional backplane

Persistent data store

Client

Client

Client

Client

IP network
Clusters as Building Block

- No alternative to clusters for building network services that can scale to global use
- Key question: what is the lowest-level building block of a cluster?
  - Commodity intel processor or higher-end SMP?
- Cluster benefits:
  - Incremental scalability
    - Adding one machine typically linearly improves performance
  - Independent components
  - Cost and performance
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

- iClicker:
  - A: Does not hide failure or inactive servers
  - B: Can not scale to millions of users
  - C: Exposes structure of underlying service
  - D: A&C
  - E: B&C
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

- Today, L4 and L7 switches can inspect TCP session state or HTTP session state (cookie sticky, etc.)
  - Perform mapping of requests to back end servers based on dynamically changing membership information

- “Load balancing” still an important topic today (more about that later)
Service Replication

IP network

Single-site server

Round-robin DNS

Simple replicated store
Service Partitioning
Case Study: Search

- Map keywords to set of documents containing all words
  - Optionally rank the 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 you
    also need to have portions of the document avail as well)
Case Study: Search

- **Vertical partitioning**
  - Split inverted index across multiple nodes (each node contains as much of index as possible for a particular keyword)
  - Replicate inverted indices across multiple nodes
  - OK if certain portion of document database not reflected in a particular query result (even expected)

- **Horizontal partitioning**
  - Each node contains portion of inverted index for *all* keywords (or large fraction)
  - Have to visit every node in system to perform full join
Replication versus Partitioning

■ Replication
  • Any replica can serve any request
  • iClicker:
    A: Failure reduces system capacity but not data availability or
    B: Failure reduces data availability but not system capacity

■ Partitioning
  • Nodes are no longer identical so certain requests need to be sent to individual nodes
  • No need for coherence traffic
  • Failure reduces data availability and may reduce capacity
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**

- **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
- 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
Graceful Degradation

- 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
- Power failures and natural disasters are not independent, severely reducing capacity
- Under heavy load can limit capacity (queries/sec) to maintain harvest or sacrifice harvest to improve capacity
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)
Online Evolution and Growth

- Internet services undergo rapid development with the frequent release of new products and features.
- Rapid release means that software released in an unstable state with known bugs.
  - Goal: acceptable MTBF, low MTTR, no cascading failures.
- Beneficial to have a 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
Load Balancing
Load balancing

- We’re going to talk about two strategies:
  1. Load balancing switches
  2. Network-wide load balancing
     (specifically within data centers)
Load balancing switches

- WAN Router
- Firewall
- Load Balancing Switch
- Ethernet LAN Switches

Integrated?
Load balancing switches

Load Balancing Switch

- Ethernet LAN Switches

- Application Server

- HTTP (front end)

- Storage Tier (Fileserver/Database)
Load balancing switches

Load balancing switch maps external virtual IP addresses (vip) to (often dynamic) physical IP addresses (dip) in internal network.
F5 “Big-IP”
Big IP: Cookie Encryption

- rule when HTTP_REQUEST {
  #
  # Decrypt the cookie on its way to the server.
  #
  HTTP::cookie decrypt "cookie_name" "password-key"
}

rule when HTTP_RESPONSE {
  #
  # Encrypt the cookie on its way to the client.
  #
  HTTP::cookie encrypt "cookie_name" "password-key"
}
rule when HTTP_RESPONSE {
    #
    # Remove all but the given headers.
    #
    HTTP::header sanitize "ETag" "Content-Type"
    "Connection"
}
Big IP: Remove SSN from Responses

```plaintext
rule ssn_rule {
    when HTTP_RESPONSE_DATA {
        set payload [HTTP::payload [HTTP::payload length]]
        set ssnx "xxx-xxx-xxxx"

        # Find the SSN numbers
        regsub -all {\d{3}-\d{2}-\d{4}} $payload $ssnx new_response

        # Replace the content if there was a match
        ...
    }
}
```