Demands of modern networked services

• Old approach: run applications on local PC

• Now: major innovation in “network services”
  – Web portals (AOL, MSN, Yahoo, Google)
  – Network services (instant messaging, Napster, Bittorrent)
  – Major web sites (eBay, Walmart, CNN)

• Must be able to keep up with Internet-scale demands
  – Need extremely high availability (outages cost millions of dollars in lost revenue)
  – Simple evolution and growth over time
Benefits of networked services

- Access anywhere, anytime
- Availability from multiple devices
- Groupware (coordination with other users)
- Lower cost (hardware multiplexed over different users, centralizes administrative burden, simplifies end devices)
  - End user devices have utilization of 4%, infrastructures often 80%
- Simplified service updates
  - Only need to update within local cluster, don’t have to coordinate with end users. Homogeneous OSs, etc.
  - I can do a search on my mobile phone, but don’t need to index and store petabytes of web sites
Approach to building modern networked services

• Need systems to handle these large demands
• One approach: purchase a large supercomputer
  – But, hard to scale, prone to failure (sometimes), and expensive
• Alternative: build cluster with Commercial Off-The-Shelf components (COTS)
  – Single-site, single-owner, well-connected cluster infrastructures
  – Leverage standard server, storage, network equipment at low-cost
  – Come up with new design techniques to achieve high reliability with low-availability components
Model for Networked Services

Diagram showing the relationship between clients, an IP network, a load manager, a single-site server, and a persistent data store.
Model for Networked Services

- Service provider has limited control over clients and IP network
  - May leverage traffic engineering/DNS redirection techniques to balance load, reduce delay
- Queries drive the service
  - HTTP, XML, SOAP, IMAP, proprietary protocols
- Read-only queries dominate workload
  - Vastly outnumber updates to data store
  - True even for commercial sites, product evaluations vastly outnumber purchases
Model for Networked Services

- **Clients**
  - Web browsers, email readers

- **IP network**
  - Internet, intranet

- **Load manager**
  - Gives level of indirection between service’s external name and server’s physical names (IP/MAC addresses)
  - Balances load, firewalls
Model for Networked Services

- **Servers**
  - CPU, memory, disk
  - Heavy use of virtualization, replication

- **Persistent data store**
  - “database” spread out over servers’ disks
  - Might also include RAID storage

- **Backplane**
  - Used by many services to carry inter-server traffic (redirecting client queries to the correct server, or coherence traffic for data store)
Load management

• Round-robin DNS
  – Distributes different IP addresses for a single domain in a rotating fashion
  – Can also return set of IP addresses for a single query, local host uses them round-robin
  – Balances load well, but does not hide inactive servers
  – Short TTLs reduce outages, but increase load

• Layer-4 switches
  – Understand TCP and port numbers, can make decisions based on this information
Load management

- **Layer-7 switches**
  - Understand HTTP requests and can balance across nodes
  - E.g., same client may make several requests to alter state (e.g., a shopping cart), want to make sure it is directed to the same server
  - Switches can “hot failover” automatically
  - Switches detect down nodes by passively monitoring TCP connections

- **Load balancing randomly often good enough**
  - Randomly throw N objects into bins $\rightarrow$ maximum bin size is $\log(N) / \log(\log(N))$ with high probability
  - Can hash on flow ids to avoid oscillations
Benefits of this approach

• Scalability
  – No other infrastructure can scale to this size
  – Can scale incrementally, by adding new servers on-demand
  – Hardware costs are typically dwarfed by bandwidth and operational costs

• High availability
  – Division between service’s external name and servers’ internal IP/MAC addresses preserve service’s availability during faults
  – Use of independent components provides independent faults (lower chance of correlated failures)
Simple web farm

- Round robin DNS for load balance
- Persistent data store by replicating all data across all servers
- Since all servers have same data, no need for coherence protocol
- (in practice, may have simple LAN backplane for updates to data store)
- Layer-4 switches manage load
- Persistent store partitioned across servers, layer-7 parses URLs to direct requests appropriately
- Backplane allows all nodes to access all data
High Availability

- Extreme symmetry
  - Every machine the same, simplifies failover and management
- Few cables
- Internal disks
- No monitors
- No people
  - Management from offsite
- Contracts limit temperature and power variations
How to measure availability?

- **Uptime:** fraction of time site is handling traffic
  - Uptime = (MTBF – MTTR)/MTBF, where
  - MTBF: Mean Time Between Failures
  - MTTR: Mean Time To Repair
  - Can improve uptime by changing MTTR and MTBF (how?)
  - ___ often easier to change in practice
  - ___
How to measure availability?

- **Yield**: fraction of completed queries
  - Yield = (queries completed)/(queries offered)
  - Unlike uptime, more directly maps to user experience
    - Not all time units have equal value

- **Harvest**: fraction of the data store that the response reflects
  - Harvest = (data available)/(complete data)
  - Harvest may be less than one due to coherency issues, failures
    - E.g., eBay might have some features unavailable while rest of site works perfectly

- Replicated systems map faults to reduced ________ (and ____ at high utilizations)

- Partitioned systems map faults to reduced________
The DQ principle

• Observation: Data per query * queries per second → constant

• Systems usually have some physical bottleneck which is tied to data movement
  – Total I/O bandwidth
  – Number of disk seeks/second

• DQ constant is changeable
  – Increase: adding nodes, software optimizations
  – Decrease: failures
Improving system DQ

• System-wide DQ can be tuned
  – Compute DQ for each node, do load-balancing appropriately
  – Can measure and provision for DQ impact of faults
  – Small test cluster is often good DQ predictor for large system
    • Inktomi uses 4-node clusters to predict performance improvements on 100-node clusters
  – Best case under faults: linear DQ reduction
    • Can be worse due to centralization

• DQ bottleneck differs per system
  – Computation, I/O, bandwidth
  – For most (top 100) sites, it’s bandwidth
Graceful Degradation

- Hard to overprovision systems
  - Peak-to-average ratio commonly 1.6:1 to 6:1, expensive to overprovision this much
  - Large bursts during “flash crowds”
    (moviephone.com added 10x capacity to handle tickets for Star Wars: The Phantom Menace and still got overloaded)
  - Some faults are correlated (power outages, natural disasters), dropping overall DQ greatly
- Need to gracefully degrade service under heavy loads
Graceful Degradation: Techniques

- Cost-based admission control: drop requests that will require a high DQ
- Value-based admission control: drop unimportant requests
  - Drop users least likely to make purchase
  - Stock trades more important than queries
- Reduce data freshness: don’t update database as frequently
  - Improves caching, reduces write workload
System maintenance

- Fast reboot: quickly reboot all nodes into new version
  - Guaranteed to reduce yield, but can improve yield by upgrading during off-peak
  - Requires staging area, because need automation for upgrades to occur quickly

- Rolling upgrade: reboot one node at a time
  - Reduces impact, as compared to fast reboot
  - But requires two software versions to coexist (hard if changes to namespace, or intracluster protocols)
System maintenance

- Big flip: upgrade half of cluster at a time
  - “dry out” half of cluster, upgrade, flip traffic over to it and repeat on other half
  - “flip” atomically switches traffic from one half to the other using a layer-4 switch
  - 50% DQ loss, but can be translated into 50% reduction in replica capacity, or 50% reduction in harvest
  - Used by Inktomi moving from Berkeley to Santa Clara (using DNS for flip), between two cages in data center
Summary: Lessons learned

• Get the basics right
  – Use a professional data center, layer-7 switches, symmetry to simplify analysis and management

• Decide on availability metrics
  – Everyone should agree on goals and how to measure them daily
  – Harvest and Yield metrics more useful than Uptime

• Focus on MTTR at least as much as MTBF
  – MTTR often easier to change than MTBF

• Understand load redirection during faults
  – Not enough to just replicate, also need spare bandwidth capacity to handle redirected traffic
Summary: Lessons learned

• Graceful degradation is critical to high availability
  – Intelligent admission control, dynamic database reduction

• Use DQ analysis on all upgrades
  – Evaluate all upgrades before deploying them, and do capacity planning

• Automate upgrades as much as possible
  – Reduces downtime, potential for mistakes
  – Using a staging area reduces downtime
  – Make sure you have a fast, simple way to revert to the old version
Summary

• Bottom line: scalable networked services should (and are) built with commodity components

• Other challenges:
  – Multiple administrative domains
  – Low/intermittent bandwidth (developing regions)
  – Service monitoring and configuration
  – Security, logging, log analysis