Scaling Web Service

Junghoo Cho

cho@cs.ucla.edu

Capacity Planning

- You built your App. You want to deploy them on real servers
- Q: How many machines do I need?
- Q: How many requests can a machine handle?
- A: Really depends on applications
  - Let us examine static vs dynamic content separately

Capacity Planning: Static Content

- Q: How many static Web pages can a machine serve per second?
- Data flow: Disk → memory → network
  - Q: How fast is each pipe?
- Typical speed
  - Disk/DB IO
    - Transfer rate: 100-3000MB/sec
    - Average seek time: 5-10ms (or ≈100K IOs for SSD)
  - Memory: 10-50GB/sec
  - Network: 1Gbps-10Gbps

Capacity Planning: Static Content

- Q: 10KB per request, 100MB/s disk IO, 1Gbps network, how many requests/sec?
- Disk → memory
  - Sequential ≈ 100MB/10KB = 10,000
  - Random ≈ 1sec/10ms = 100
  - If SSD or data can be cached in RAM, higher throughput
- Memory → network ≈ 100MB/10KB = 10,000

Capacity Planning: Static Content

- A high performance Web server can handle ≈ 10,000-50,000 req/sec/core
  - Nginx, Apache, …
  - Q: How many requests/day is it?
- Main bottleneck is mostly disk/network IO
  - Assuming no SSL encryption
  - Ensure enough RAM to cache hot data
  - DNS look up is often VERY slow
    - Disable reverse DNS lookup

Dynamic Content

- Q: How many dynamic Web pages can a machine serve per second?
- This really depends on the complexity of application
  - No hard and fast answer
  - IO/context switch/CPU, any of them can be bottleneck
- But, rule of thumb is 10 request/sec/core
  - Assuming reasonably simple application logic
  - No video encoding, for example
  - Q: How many requests per day is it?

Remarks on Capacity Planning

1. Set your min acceptable service requirement
2. Characterize the workload
    - Req/sec, res util/req
    - Measure resource utilization from your workload
3. Remember: “premature optimization is the root of all evil”
    - Do not optimize based on your “guess”
    - Do not optimize unless you are sure it is important
    - **MEASURE your workload first**!!!

Basic Unix Monitoring Tools

- CPU/process
  - `top`: load avg: # processes in running or runnable state
  - `ps`: common options: `axl`
  - `pstree`
- Disk io
  - `iostat`
- Network io
  - `netstat`: common options: `-i` or `-s`
- Memory
  - `free -m`, `ps axl`, `vmstat`, `memstat`

Caching

- Q: Can we use caching to improve performance/scalability?
  - Many, if not most, CS problems can be solved with caching!
- Q: At what layer? DB? Application? HTTP?

Encryption Layer
HTTP Layer
Application Layer
Database/Persistence Layer

Below Database Layer

- Cache disk blocks!
  - Add lots of RAM
  - Increase database bufferpool size

Above Database Layer

- Cache database objects in RAM
  - All database access goes through caching layer
  - Minimize # requests hitting DB
- Common tools: Memcached, Redis
  - Most tools support distributed caching
- Typical data model: (key, value) pair
  - Key-based object lookup
  - e.g., memcached

Above Application Layer

- Store generated HTML page as a static file
  - WordPress cache plugin, Varnish cache server, NGINX microcache, …
- May boost capacity significantly
  - Often by orders of magnitude
  - Especially for slow-update dynamic sites or if short delay is tolerable
    - e.g., blogs, web forums, …

Dynamic-Page Cache Example

- Dynamic page: 100ms per page
  Static (or cached) page: 1ms per page
- Q: No caching. How many requests per second?
- Q: Dynamic page is cached for 1 sec. How many requests per second?
  - "Microcaching": caching for a very short period
- Q: How much has the capacity increased?

Edge-Side Include (ESI)

- Q: What if a small part of a page has to change every time?
- A: Separate out uncachable parts from cachable part
- Example

```
<html>
... <esi:include src="part1.html"/> ...
</html>
```

  - ESI/cache server fetches all parts and synthesizes the final page
  - Each part may be cached with different expiration date
- Instead of ESI, AJAX may be used ("client-side include")
  - But ESI is transparent to client

Above HTTP Layer

- Content Distribution Network (CDN)
  - Cache pages/images/videos close to users at the edge of the network
  - Users access cached object located close to them
    - Lower delay
    - Lower load on network
- Q: How can a browser "know" the location of the cached objects that are close to it?

Above Encryption Layer

- Caching end-to-end encrypted content is hard
  - Very nature of encryption
- But…

Browser Cache

- Let browser cache (decrypted) pages locally
  - No load on network or server
- Some relevant HTTP headers
  - From server:
    - `Cache-Control: max-age=31536000` (seconds)
    - `Expires: Wed, 21 Oct 2015 07:28:00 GMT`
    - `Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT`
    - `ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"`
  - From client:
    - `If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT`
    - `If-None-Match: "33a64df551425fcc55e4d42a148795d9f25f89d4"`

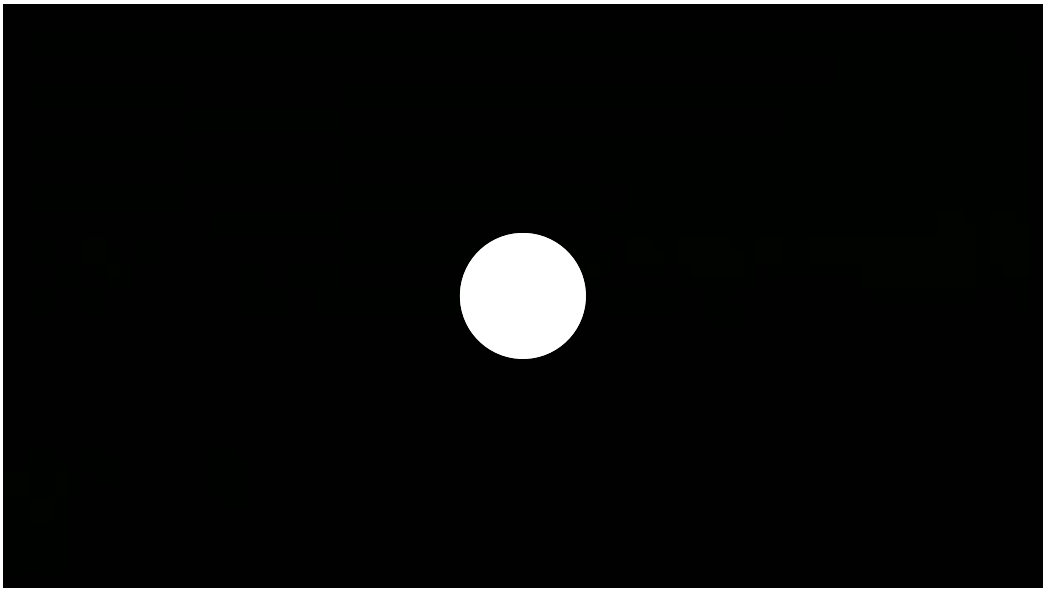
Simple, Performant Setup: Example

NGINX (with microcaching)
Node/Tomcat/WordPress
Mongo/MySQL/Postgress
Linux
Hardware (Multicore, Enough RAM, SSD)

Scaling Web Server

- We have deployed our server
- Our user base is outgrowing what we originally planned!
  - Even after we used caching at the right places
  - Remember, most services never get to this stage
  - Plan for it, but don't spend too much time until it really happens
- Q: How can we scale a Web site as it grows?
- Two high-level approaches
  - *Scale up*: buy a larger, more expensive server
  - *Scale out*: add more machines
- Q: Pro/cons of scale up/out?

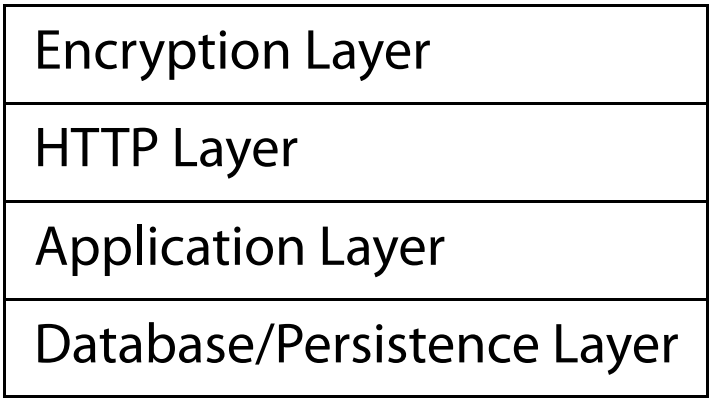
Scaling-Out: Google Data Center



(Video credit: CBS)

Scaling Out Web Applications

- Q: How to scale out a Web site using cluster?
- To see possible ways, let us revisit web server architecture



Scaling Out Each Layer

- Q: Can we scale out encryption layer? How?
- Q: Can we scale out http layer? How?
- Q: Can we scale out application layer? How?
- Load balancer: distribute incoming requests to one of backend servers
  - Hardware vendors: Cisco, F5, …
  - Software: HAProxy, Apache Traffic Server, NGINX, …

Scaling DB Layer

- Q: How can we scale database?
- DB scaling is hard and complex. Let us consider concrete examples

Scenario 1: Read-Only Access

- Global read only access
  - Example: online map, yellow pages, …
- Q: 30 IOs/sec/machine. 3 read IOs/request. How many req/sec per machine?
- Q: How can we scale if we get 20 req/sec?
- Remark: Replication has no synchronization issue for read-only access

Scenario 2: Local Read and Write

- Local read and write.
  - All user data is local. No global sharing of data between users
  - Examples: Web mail, online bank, …
- Q: 30 IOs/sec/machine. 2 reads+1write IOs/sec/request. How many req/sec per machine?
- Q: How can we scale to deal with 20 req/sec?
- Remark: Partitioning (or sharding) has no synchronization issue

Scenario 3: Global Read and Write

- Global read/write. Writes are globally visible
  - Example: online auction, social network
- Q: 30 IOs/sec/machine. 2 reads+1write IOs/sec/requests. How many req/sec per machine?
- Q: How can we scale to deal with 20 req/sec? replication? partitioning?
- Q: Maximum # of req/sec that can be supported using replication?
- Remark: Eventually write requests saturate the DB

Scaling Out: General Remarks

- CPU is rarely a bottleneck and is relatively easy to scale
- Eventually, DB/storage becomes the main bottleneck
  - Scaling out DB is VERY CHALLENGING and requires careful analysis/design
  - DB is the layer where scaling up makes most sense
- Two approaches to DB scaling: replication and partitioning
  - Design your database carefully
  - Identify early on how you will cache/replicate/partition your DBMS when users grow

What We Learned

- Capacity planning
  - Static vs dynamic content
- Caching
  - Disk, database, application, CDN, browser cache
- Scale up vs Scale out
  - DB layer is the hardest to scale out
    - General approach: replication and partition
    - Write synchronization problem

Scaling Web Service - Junghoo Cho - cho@cs.ucla.edu