CS144: Scaling Web Service

Capacity planning

- Q: How many requests can a machine handle?
  - Really depends on your application
  - Possible bottlenecks
    * Disk/DB IO: disk 100-500MB/sec, 5 - 10ms avg seek (or ~100K IOs for SSD)
    * Network: 1Gbps - 10Gbps
    * CPU/memory: 3Ghz
  - Estimation of workload capacity
    * Static content:
      ◦ Q: 10KB per request, 100MB/s disk io, 1Gbps network, how many requests/sec?
        ◦ Disk -> memory -> network diagram

        ◦ Disk -> memory: sequential(~100MB/10KB) vs random (~1sec/10ms)
        ◦ Memory -> network: (~100MB/10KB)
      ▷ A high performance Web server can easily handle 5,000 req/sec/cpu
        ◦ nginx, apache, …
      ▷ Main bottleneck is mostly disk/network io
    * Dynamic content:
      ▷ Depends on the complexity of application
      ▷ Rule of thumb: 10 request/sec/CPU
        ◦ Assuming reasonably simple application logic
        ◦ No SSL, no video/image encoding, …
      ▷ Cpu/context switch/io can be bottleneck
- Capacity planning
  - Characterize the workload:
    * Req/sec, res util/req
    * Measure resource utilization from your workload
– Set your min acceptable service requirement
– Remember: “premature optimization is the root of all evil” - Donald Knuth
  * Do not optimize based on your “guess”
  * Do not optimize unless you are sure it is important
  * MEASURE from your workload first!!

• Tools for Profiling
  – 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`
  – DNS look up often causes lots of problem
    * Disable reverse DNS lookup if possible
  – Code profiling
    * A good first step to identify bottleneck
    * Node: Profiler built in since V4.4
    * Java: JProfiler, Eclipse TPTP (test & performance tools platform), …
    * PHP: Xdebug profiler

### Caching

• Q: Can we use caching to improve performance/scalability?
• Q: At what layer? Storage/DB? Application? HTTP?

<table>
<thead>
<tr>
<th>Transport encryption (SSL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP server (apache/nginx)</td>
</tr>
<tr>
<td>Application server (tomcat/node)</td>
</tr>
<tr>
<td>Persistence/Storage layer (MySQL/Mongo)</td>
</tr>
</tbody>
</table>
– Caching files/disk blocks (disk cache)
– Caching database objects
– Caching dynamic web pages
– Caching content close to the users

• Data object caching layer (memcached, redis)
  – All database access goes through caching layer
  – Minimize # requests hitting DB
  – memcached data model: (key, value) pair. Key-based lookup
  – Can use multiple machines for caching by partitioning the key w/
  – Special care on possible machine failure

• Dynamic page caching layer
  – Store generated HTML page as a static file
    * E.g., WordPress cache plugins
  – Q: What if a page contains a few user-specific content?

• 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 the main server
  – Q: How can a browser “know” the location of the cached objects that are
    close to them?

• How can we scale a Web site as it grows?
  – Scale up: buy a larger, more expensive server
  – Scale out: add more machines
  – Q: Pro/cons of scale up/out?

Scaling-out Web applications

• Typical Web server architecture

<table>
<thead>
<tr>
<th>Transport encryption (SSL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP server (apache/nginx)</td>
</tr>
</tbody>
</table>
• **Q: How to scale out a Web site using cluster?**
  
  – **Q: Scaling out each layer?**
    
    * Encryption layer? http layer? application logic layer?
    * Persistence/database? more discussion later
  
  – Load balancer (TCP NAT request distributor)
    
    * Hardware: Foundary Network ServerIron, Cisco LocalDirector, …
    * Software: nginx, …
    * DNS round robin

• **Q: How can we scale database once the limit is reached?**
  
  – **Scenario 1**: Global read only data (online map, yellow pages)?
    
    * Q: 30 IOs/sec/machine. 3 read IOs/request. How many requests per machine?
    
    * Q: How can we scale if we get 20 requests/sec?

  
  ▶ Remark: no DB synchronization problem

  – **Scenario 2**: Local read and write. All user data is local. No global sharing of data (Web mail, online bank, …)?
    
    * Q: 30 IOs/sec/machine. 2 reads+1write IOs/sec/session How many sessions per machine?

  
  * Q: How can we scale to deal with 20 sessions? Does replication help?

  
  ▶ Remark: again, no DB synchronization problem
– **Scenario 3**: Global read/write. writes are globally visible (online auction, social network)

  * Q: 30 IOs/sec/machine. 2 reads+1write IOs/sec/session. How many sessions per machine?

  * Q: How can we scale to deal with 20 sessions? replication?

  * Q: Maximum # of sessions that can be supported using replication?

  * Q: partitioning?

* Remark:
  ▶ Eventually write requests saturate the DB
  ▶ Scaling out DB is VERY CHALLENGING and requires careful analysis/design
  ▶ Many companies buy larger machine to scale DB for critical data

* General remarks on scaling out
  – CPU is rarely a bottleneck and is very easy to scale
  – After reasonable optimization, DBMS/storage is often the main bottleneck
    * Two basic approaches for DB scaling: replication and partitioning
    * Design your database carefully
    * Identify early on how you will cache/replicate/partition your DBMS