Design and Engineering of Computer Systems

Lecture 31: Performance measurement

Mythili Vutukuru
IIT Bombay
Performance engineering

• So far in the course: how computer systems are developed
• This week: performance engineering
  • Measuring and optimizing performance of computer systems
  • How is performance measured? What to measure, how to measure, what are parameters we can change, what output metrics should we observe, ..
  • Performance analysis: simple back-of-the-envelope calculations to make sense of the performance measurements
  • Identify performance bottlenecks, which component is slow, and why
  • Techniques to optimize performance: within a single machine and across the entire system
Example: multi-tier web application

- Consider a multi-tier application (many examples seen so far)
  - Front-end web servers, multiple application servers, backend databases
  - Clients make requests, processed by system, responses sent back
- Incoming traffic into the system can be measured using: load?
  - Number of concurrent clients connected to the system
  - Incoming rate of requests per second
  - Mix of various types of requests in the incoming traffic
- Performance of the system measured by (average values of):
  - Throughput: number of requests per second handled successfully
  - Response time / RTT / latency / delay: time taken by the system to return back a response to a client request (varies by type of request)
Performance bottleneck

- The performance of a system is limited by the slowest component in the system (bottleneck)
- Consider a web application servicing one type of requests
  - Front-end can serve 1000 req/s, app server can handle 5000 req/s, backend database can process 100 req/s
  - Max throughput of the system is 100 req/s (capacity)
  - Database component will be the performance bottleneck and will be fully occupied at peak input load (other components will not be as busy)
  - Database component takes approx. 1/100 seconds = 10 milliseconds to process each request (service demand)
  - Response time of system will be at least 10 millisecond + time needed at other components
- Sometimes, the network can also be a bottleneck, not any component
  - Some switch on path between clients and server can only forward 50 req/s
Understanding system performance

• Consider web application with max throughput (capacity) 100 req/s

• Suppose incoming traffic into system is only 10 req/s
  • The system throughput is 10 req/s, no performance bottleneck
  • No component overwhelmed, quick (low) response time

• As incoming load approaches capacity, e.g., 90 req/s
  • All incoming requests are served, throughput is 90 req/s
  • Bottleneck component starts to get busy, queue builds up, responses take longer

• If incoming load exceeds system capacity
  • Bottleneck component fully saturated, max throughput (capacity) achieved, throughput flattens (cannot increase any more)
  • Response times are higher due to queueing delays, continue to increase with increasing load
Understanding overload

• What happens when incoming load to server greatly exceeds capacity?
  • Throughput flattens, cannot increase beyond bottleneck capacity
  • Response times keep increasing as requests get queued up at bottleneck (high queueing delay)
  • App software cannot process requests, returns error messages (e.g., HTTP server returns code of “503 service unavailable” if it is overloaded)
  • TCP sender does not get acks, terminates connection (socket syscalls fail)
  • Networking routers/switches/NIC may get overloaded, packets fill up queues inside router or device driver, packet are dropped

• Result of overload: very high response time (for requests that complete) or errors (requests get no response at all, or get error messages)
Understanding saturation

• Ideal operating point of a system: just below max capacity or saturation
  • Close to max throughput, not too long response times, no errors

• At saturation, some hardware resource at bottleneck is fully (100%) utilized
  • E.g., all CPU cores are fully busy with no spare CPU cycles
  • E.g., hard disk is running at full capacity performing reads/writes

• How to improve capacity?
  • Increase hardware resources at bottleneck component
  • Or, optimize system to use hardware resources more efficiently

• Sometimes, bottleneck due to software issues only (no hardware resource is fully utilized)
  • E.g., maximum number of file descriptors opened by process, cannot open any more
  • E.g., threads wasting time waiting for locks, even though CPU is free
  • Such issues can be fixed by rewriting code or tuning OS parameters
Performance: parameters and metrics

• Given a computer system (with certain capacity and configuration of various components), how to measure its performance?

  **Input parameters** on which performance depends / **incoming load**
  • Number of concurrent users/requests in the system
  • Rate (requests/sec) of incoming traffic
  • Mix of various types of requests (which require different amounts of work)

• **Performance metrics** / outputs measured
  • Throughput of the system (averaged over a time window, end-to-end and per-hop)
  • Response time or latency (averaged, end-to-end and per-hop)
  • Utilization of various resources at components, especially bottleneck (averaged)
  • Various kinds of errors and failures (counts)

• **Load testing of a system**: vary incoming load, measure performance
Types of load testing

• Different types of load tests based on which input parameter is varied
  • **Open loop load testing**: vary rate of incoming traffic into system
    • Generate \( N \) req/s for increasing values of \( N \) (mix of requests can also be varied)
    • Can be implemented by firing a request every \( 1/N \) second
  • **Closed loop load testing**: vary number of concurrent users of system
    • \( N \) concurrent users, each user sends a request, gets response, waits for some amount of time ("think time"), sends next request
    • Can be implemented by having \( N \) threads/processes emulating \( N \) concurrent users

• Both techniques are valid ways of varying input load of system
  • Open loop testing can lead to higher number of concurrent users, more queueing

• **Load generators**: software programs or hardware appliances that generate load to test a computer system in open/closed loop manner
  • Provide knobs to vary rate of requests, or number of concurrent users etc.
Running a load test

• How to run a load test
  • Setup load generator, system under test and connect them suitably
  • Generate increasing amounts of load from load generator to system (by varying rate of incoming traffic or number of concurrent requests)
  • Measure output metrics (throughput, response time, errors, utilizations) for each value of input load
  • Eliminate sources of non-determinism (e.g., pin threads to CPU cores)
  • Ensure load generator or connecting network is not the bottleneck

• Results of load test: performance metrics vs. incoming load

• What after load test? Performance analysis and engineering
  • Analyze and understand performance metrics, identify bottleneck
  • Optimize system, repeat load test
  • Stop when system performance meets the expected incoming load
Summary

• In this lecture:
  • What is performance: input parameters, output metrics
  • How to measure performance

• Programming exercise. Setup a simple web server using the freely available Apache server. Apache JMeter is a load generator to test web servers. Try to run a simple load test using JMeter.