Performance Measurement of Web Servers

Kai Shen
Dept. of Computer Science, University of Rochester

Workload Design

- Measure the throughput of a server (as a software/hardware system)
  - let a testing client send requests with increasing rate.
  - measure the highest rate at which the server can successfully complete at least 98% of all requests with response time less than 2 seconds.

- server side - HTTP objects
  - total size
  - individual object sizes

- client side - HTTP requests
  - individual request arrival intervals
  - object popularity

Goal of Workload Design

- trace-driven workload - using a real-world instance/traces
- synthetic workload - based on accumulated statistics

Trace-driven Workload Design

- server side - HTTP objects
  - total size
  - individual object sizes

- client side - HTTP requests
  - individual request arrival intervals
  - object popularity
  ⇒ replay the access log at http://www.cs.rochester.edu.
  ⇒ may need to scale up or down the request arrival interval to generate requests at certain rate.
Done? Not so fast!

Synthetic Workload Design

Server side - HTTP objects
- total size => an input variable, relevant to the system memory size
- individual object sizes
  - average object size
    => based on statistics, say, 5KB for HTML files
  - distribution of object sizes
    => based on statistics, Lognormal/Pareto/Weibull distributions

Client side - HTTP requests
- individual request arrival intervals
  - we already know the average, why?
  - exponential distribution, this is not quite based on statistics, but rather assertion, why?
- object popularity
  - based on statistics, Zipf distribution

Putting it together
- generate a set of HTTP objects for the Web server based on distribution etc.
- having the testing client to send a set of requests based on the determined rate and distributions etc.
- do it many times with different sets of data out of respective distributions until the result converges.

The bottom line is that many people don't like this trouble and simply use trace-driven workload.
Design of the Testing Client

- start a new thread to send/receive a HTTP request
- sleep for a specified period of time
- make a TCP connection with the Web server
- send a HTTP request
- wait for the response
- bookkeeping and done

Design of the Testing Client (cont.)

- start a new thread to send/receive a HTTP request
- keep checking the clock until the time is up
- hopefully we have at least two processors
- make a TCP connection with the Web server
- send a HTTP request
- wait for the response
- bookkeeping and done

Emulating Wide-area Users
- low bandwidth
- long latency

Performance of the Testing Client
- why is the performance important?
  - we are testing the server’s performance, not the client’s.
- how to find out the client is not fast enough and how to fix it?
- Distributed/parallel clients.
  - having a group of clients involved in the testing simultaneously, coordinating on requests composition, request arrival intervals, etc.
Some benchmarks/traces

  - by now, you should be able to do better than them

- The Internet Traffic Archive - http://ita.ee.lbl.gov/
  - including many Web traces as well as Ethernet/IP/TCP traces