Performance Measurement of Web Servers

Kai Shen
Dept. of Computer Science, University of Rochester

Web Servers and Performance Measurement

- What are Web servers?
  - implement the server-side HTTP
  - accept concurrent requests
- Popular Web servers
  - Apache, Microsoft IIS, Netscape Enterprise, Zeus, ...

- Main goal: measure the performance of a Web server (as a combined software/hardware system)
  - compare competitive designs
  - project system behavior under real workload/setting
- Extended goal: basis for performance evaluation of more complex concurrent servers
  - commercial database servers, online digital libraries, etc.

Performance Metrics for Web servers

- throughput
  - the number of requests completed per time unit
- mean response time
  - the average request completion time
- other service qualities
  - image resolution of an image retrieval
  - quality of search query results
- a combination of them

Measuring the Throughput

- How?
  - let a testing client send requests with increasing rate.
- Design #1:
  - measure the highest rate at which the server can successfully complete all requests.
  - well, it looks like there is no end!
- Design #2:
  - measure the highest rate at which the server can successfully complete all requests with response time less than 2 seconds.
  - poor stability!
- Design #3:
  - measure the highest rate at which the server can successfully complete at least 98% of all requests with response time less than 2 seconds.
Measuring the Mean Response Time

- How?
  - let a testing client send requests at a certain rate and then measure the mean response time
- What is the rate?
  - e.g. 90% the maximum throughput

Where are we now?

- We have the basic methodology but this is not enough.
  - Design of workload
    - what are the individual request arrival intervals?
    - what is the actual service data?
    - what are the actual requests?
  - Implementation of testing clients
    - how does the testing client work?
    - how to make it more efficient?

Workload Design for Web Server Performance Measurement

- Measure the throughput of a Web server
  - 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

- Be representative!
  - trace-driven workload - modeling a real Web server and real traces
  - synthetic workload - based on accumulated statistics
Trace-driven Workload Design

- server side – HTTP objects
  - total size
  - individual object sizes
    ⇒ build a copy of an actual Web site, e.g. 
- 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.

Synthetic Workload Design

Server side – HTTP objects

- total size
  ⇒ pick a size, 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 (how to get this?)

Client side – HTTP requests

- individual request arrival intervals
  - the average is an external input
  - exponential distribution, this is not quite based on statistics, but rather analytical model
- object popularity
  - based on statistics, Zipf distribution

A Web Server Access Log

```
```

imprecise timing.
Synthetic Workload Design (cont.)

Putting it altogether
- 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.

Where are we now?
- We have the basic methodology.
- We have the workload design:
  - what are the individual request arrival intervals?
  - what are the actual service data?
  - what are the actual requests?
- Implementation of testing clients
  - how does the testing client work?
  - how to make it more efficient?

Design of the Testing Client
- send requests at specified intervals
- make a TCP connection with the Web server
- send a HTTP request
- wait for the response
- bookkeeping and done
- you can’t sleep arbitrarily short

Design of the Testing Client (cont.)
- send requests at specified intervals
- make a TCP connection with the Web server
- send a HTTP request
- wait for the responses
- bookkeeping and done
- hopefully we have at least two processors
- keeps checking the clock until the time is up, not giving up the CPU in the mean time
**Emulating Wide-area Users**

- low bandwidth
- long latency

**Performance of the Testing Client**

- why is the performance important?
  - we want to know the performance limit of the server, not that of the client.

- 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 much better than SPECweb99

  - including many Web traces as well as Ethernet/IP/TCP traces