WebSite Loading and Capacity Analysis

Edward Miller
Software Research, Inc.
901 Minnesota Street
San Francisco, CA 94107 USA

Email comments to miller@soft.com
See also the companion papers
The WebSite Quality Challenge.
WebSite Testing.

ABSTRACT

Estimating the actual capacity of a WebSite server requires a variety of different types of analyses. There are many deep, highly technical methods for calculating the capacity of a server to meet imposed load. However, a more practical way is to simply measure the perceived response time accurately, and scale up server capacity to make sure that maximum real-user experiences don't exceed unacceptable delays.

Why is WebSite Capacity An Issue?

We all have had the exasperating experience of waiting too long for a page to arrive at our Web browser. Ultimately, if the response time is too long, we "click away" and do something else.

Even when the Web is heavily saturated with requests, if you are patient enough every page you request will -- ultimately -- be delivered to your browser. But that's not good enough. Too slow response times turn users away, or, worse yet, because the user has moved on to another page or context, important session data could be lost.

How Much Capacity Is Enough?

How slow is "too slow"? Jakob Nielsen's figures from his keynote talk at the QualityWeek 1999 Conference & Exhibition suggest that after about 7 seconds users "click away". The real answer to "too slow" probably is more subjective, but at the same time probably not too terribly different. In other words, the WebSite sever is configured properly and effectively when it has "enough capacity to meet the customer demands."

- **How Do You Know If Your Server Capacity Is Large Enough?** Good engineering practice suggests that Web server machines have a Safety Factor of 2 or more. This means, that, when serving the design load, the Web servers ought to be running at about 50% of maximum capacity.

  There is very good software available that will indicate how many URLs are being requested per second, and how many bytes are downloaded per second. If you know the server machine capability you could average these figures and estimate accurately if the average capacity to deliver pages exceeds the average demand. But this leaves unanswered the key question: *What will the peak load be?*

- **How Do You Know If Your Server Capacity Is *Not* Large Enough?** Another way to look at this is to look for the reverse information: when is the server delivering a request too slowly? While you may never really know what the peak deliverable capacity of the server is, you might be able to tell fairly accurately when the imposed server load (i.e. the queue of incomings requests) is not being served within a specified time limit, say 10 seconds. Then you surely will know when there is NOT enough capacity -- when this limit is exceeded.

  But if the WebSite involves a two-tier or three-tier structure -- which is increasingly common for e-commerce sites -- then measuring one machine may lead to false conclusions. For example, all three machines in a three-tier structure could be achieving their performance goals -- that is, not setting off alarms -- but the overall cumulative application could still be "too slow" to satisfy users.

- **Can You Measure Response Time From a Typical Users' Perspective?** It may be simpler to measure response time at the client side of the picture, i.e. from the browser. If you can pre-establish a multi-case scenario that is typical of what users do, then it is possible to engineer series of experiments that measure WebSite performance against that specific scenario.

Even though it is relatively easy to measure if such tests run slower than a threshold -- for example, the 10 second overall response time limit -- this approach has a fundamental limit: it is only as good as the set of scenarios that you develop to emulate actual WebSite use.

What Affects WebSite Performance?

Many factors affect how fast a WebSite appears to a user. There are many, many stages between a RETURN typed on a browser and a completed [or complete enough] page being rendered on the client.

- **Client Machine Speed and LAN Factors.** Before the request is delivered to the Web by the client machine it may have to work its way through the local LAN and enter the Web, where the connect speed and local saturation affects performance.
- **The Outbound Request to the Server.** Before the request gets to the server the Web routing and other technical overhead produces delays.
- **The Server Response Time.** After the request gets to the server the response speed is affected by the current request backlog, where the request is in a multi-layer queue, and how long-running the last-tier request (typically some kind of database access) takes.
- **The Inbound Delivery to the Client.** After the response is delivered by the server the reverse of the above sequence takes over: the delivered pages have to wend their way through the Web back to the client.
- **The Client Response Time.** After the data arrives at the client access point client machine speed and local LAN factors come into play again.
- **The Client Rendering.** The browser now has all the parts it needs; it is a matter of how long it takes the browser to render the page/image -- or at least a minimal part of it -- so that the user can take action.

Note: An apology is due for the simplifications here. In fact, the sequence is rather more complex because all of the machines along the way between a request by a browser and a response seen in the browser involve, typically, multi-programming, multi-threaded executions that dynamically adapt to changing Web conditions that are at least in part adjusting to the actual requests that are being discussed. Left out are such technologies as threading requests via different routs, packet re-transmits and asynchronous arrivals, and much LAN protocol complexity.

Still, the bottom line is clear: elapsed time perceived by the user actually occurs, as they say, in true real time.

How Do You Impose Loads on a WebSite?

The main goal of creating an artificial load to impose on a WebSite is to permit the load to emulate one or a dozen or a hundred or thousands of users actually using the WebSite. There are two main ways to do this. Caveats: On the Web, this may be quite difficult. On the LAN this is easier, but may require a 100Mbps LAN to saturate the servers. Hope: At some point the capacity will scale linearly: 2X machines means 2X capacity.

- **HTTP Protocol Based.** No matter what a user does on a WebSite the HTTP protocol prevails, so it is natural enough to start thinking about imposing load by finding a way to simulate the HTTP protocol. You can relatively easily create or record HTTP requests by a browser to the candidate WebSite. There are many tools that can do the *get URL page* [a simple command/action] -- and do this under user control. A particularly simple one is called `TorturePL` and runs in PERL on UNIX machines.

  While relatively easy to use to generate basic retrievals of pages, this approach suffers from the fact that **all** of the URLs associate with a page have to be included if the simulation is to be a realistic one. The disadvantage is that you could easily create a test scenario that fails to include important load factors such as download times for images and other slow-to-respond page components.

- **Browser Based Playback Simulations.** In this approach, the test scenarios are scripts that control a test browser and, working through the browser, emulate the typical users actions, responses and timings. This method has the advantage of reality, but may involve more work in deciding on a scenario and making sure that the scenario plays back realistically.

Capacity Testing Exercise

We can now outline a basic experiment format that provides a high level of realistic loading of a candidate WebSite server using eValid's unique client-side browser based testing technology. The goal of each experiment is to determine the ability of the subject WebSite to sustain load from a varying number of realistic simulated users. Generally this goal is obtained if we can develop a set of response-time curves that illustrate typical average response times from the server via the Web, as a function of WebSite load that has been imposed.

- **Phase 1: Create Typical Scripts.** Using the eValid toolset, we record a collection of scripts that represent typical usage of the candidate WebSite. The recordings will be made with the Real Time Recording switched ON so that all wait times

are kept in the script. A few tests will generally allow gaining enough basic experience with the candidate WebSite so that the wait times are typical of a normal user. Experts in use of the WebSite may also contribute eValid scripts.

To make the overall test scenario as realistic as possible these recorded tests should include:

- Some small tests, involving only some simple navigation around the site.
- Some medium tests, involving basic operations on the site.
- Some long tests that involve deep operation of critical features of the WebSite.

After recording we make sure each test script is *self-restoring*, that is, we make sure that running the test script multiple times will not perturb the WebSite being measured in a way that will affect the results. This is a relatively common attribute of a regression test and may involve the use of test accounts or special logins that avoid irreversible second-tier or third-tier database moves.

At this point we can calculate, for each test script, the load each script represents in terms of the total number of URLs involved and the size [or average size] of the URLs retrieved. From this we can make basic estimates of the URLs/sec and KBytes/sec that might represent a serious load on the candidate WebSite.

- **Phase 2: Identify The TestBed.** eValid playbacks from a client site emulate user behavior with very great precision. Normally all tests are run without any data in the browsers' caches -- doing this assures that the tests do in fact really download all of the relevant URLs. [Otherwise one might make measurement runs of the client machine performance rather than of the performance of the servers for the candidate WebSite.]

  The bandwidth from the machine or machines running the scenario affects the meaningfulness of the tests. The bandwidth available from the test machines to the WebSite being tested should be large enough that the duty cycle does not exceed 50%.

  PC's with large enough memory and high-enough speed access, e.g. T1 or T3, may have to be identified for large experiments.

- **Phase 3: Establish Scenarios of Tests.** The collection of tests recorded in Phase 1 need to be organized into groups, with specified delay multipliers and playback repetition counts of each test taken into consideration. Each test can be given individual *local test names* to make the reporting easier.

  Guidelines for scenario design include:

  - *Total running time.* The total number of tests and the repetition counts should be enough so that a "steady state" can be reached and sustained for a long-enough period of time to have confidence that the WebSite server performance at that level is adequate.
  - *Scale up.* The tests should begin with a smaller scenario and then scale up on the 1/2/5/10 ratio.
  - *Data Rate.* The total data rate for all of the activity generated while the scenario runs should be *less than* the available bandwidth from the client to the server. Otherwise the test scenario will tend to first saturate the channel rather than the server.

  This planning should be done in advance insofar as possible because the higher loadings will likely involve use of substantial resources. For example if we are testing a high-capacity site, capable of 1000's of simultaneous users, then the tests will have to consume the corresponding amount of bandwidth.

- **Phase 4: Establish Load Target.** Tests you run on a WebSite using eValid technology will, unless interrupted, run to completion. The same is true for your own work on a WebSite. Unless you "click away" even the slowest WebSite will eventually finish downloading. (WebSites that respond with an error are, of course, a different matter.)

  This fact is important because if you load up a site with 1000's of users using eValid playbacks ultimately, depending on the server capacity and the bandwidth of the pipeline to the client, all of the requested pages will, ultimately, be downloaded successfully. Hence, the only really effective measure of the capacity of a Web server is how fast it succeeds in delivering pages to your client, relative to how fast it could do that if the server and the intervening WWW infrastructure were all infinitely fast. This time is called the unloaded performance time or the *base performance time*.

  A very simple way to establish a measurable capacity criteria for your Web server is to require that the slowest overall test time -- i.e. the overall test time for a script in the heaviest load that you expect the server to have -- is always below some multiplier over the *base performance time*. A good, practical measure you can set in advance for this is 2X or 3X or 4X or even 5X.

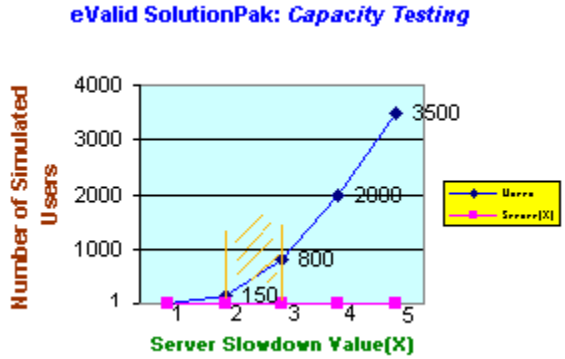
  This figure is called the *server slowdown value*. It is a fixed value that your server must never exceed for a specified *server capacity*. If you choose, for example, a 2X factor, that means your server will be judged to be at 2X *server capacity* when the average download time of a particular scenario is no more than 2 times as long as the *base*

performance time.

- **Phase 5: Run Tests.** If the planning described has been well done running the tests should be straightforward. The basic load experiment structure to be imposed by each scenario is known and the sequence in which to run them is explicit in the *scenario definition file.*

  The first part is to establish the *base performance time* for a very lightly loaded server. After this is established, the tests are run up in stages to the target maximum load. The average response time is measured in each successive stage.

- **Phase 6: Estimate Capacity.** As you run increasingly complex scenarios you build up a simple average time chart like the following. Note that the *base performance time* is at the left of the chart.



  For example, if you have chosen a 2X-3X server capacity target, then this chart suggests that your WebSite server, as measured by the tests you have chosen, will saturate after the Nmax = 800 value shown. So, you have a server with a capacity of 800 simultaneous users.

Summary

While the issues of measuring WebSite server capacity are many and complex, what becomes clear is that users' perception of effective, useful repsonse time is the dominant factor in assuring adequate capacity. This can be accomplished by realistic browser-based experiments that measure aggregate response time as a function of gradually increased parallelism.