When Load Testing Large User Population Web Applications
The Devil Is In the (Virtual) User Details

James F Brady
Capacity Planner for the State of Nevada
jfbrady@admin.nv.gov

Many times load testing is dismissed as a waste of time and money because past results didn't conform to real world experience when the application went live. Sometimes it's because the test suite is too narrow but often it is due to the approach used to produce traffic and the way results are interpreted. This discussion focuses on the latter situation because a lack of testing scope is an obvious limitation but poor quality traffic and improper analysis techniques are subtle shortcomings that impact test credibility in ways that aren't always clear until the live application reveals them.

1.0 Introduction

When load testing large user population web applications it is easy to get caught up in load tool mechanics and fail to insure the traffic offered to the target application has real world properties. This paper illustrates implementation techniques which emphasize high quality test traffic creation and concise application scalability analysis. A focus on traffic quality and concise analysis can alter the way the load tool's virtual users are implemented in test scenarios and may change their commonly accepted role in results reporting.

The discussion begins with an overview of the steps normally taken to perform a web application load test. This review is followed by a description of virtual users from a traffic flow perspective and how that flow compares with real user behavior. This comparison leads to a fundamental principles based technique for quantifying traffic quality that expands into a set of quality improvement recommendations. The virtual user's role in scalability testing is discussed next using queuing concepts as motivation. Then, an example load test is provided which illustrates statistically meaningful scalability analysis and demonstrates results reporting within the context of the traffic quality improvement methods described. The summary section puts the virtual user's role into perspective and ends with some concluding remarks.

2.0 Traditional Load Testing Mechanics

Web application load testing is generally performed using one or a few computers running load generation software. The intent is to simulate the web page request behavior of end users accessing an application running on a target platform. Data collected from the tests are used to evaluate the performance characteristics of the application from a response time service level and platform scalability perspective.

Load generators apply the concept of a virtual user as a substitute for a real user. The implementation of this concept is usually a process thread intended to mimic the behavior of an actual user making application requests in a particular event sequence with a representative amount of "think" time between the previous response and the next request.

Setting up and running load tests is generally viewed as a mechanical process whose primary steps include:

1. Identify the application event sequence of interest.
2. Record the sequence with the load tool's recording software.
3. Edit the recorded event sequence file to:
    a. Remove extraneous web events while keeping those which are primary to the application;
    b. Add assertion strings to each event insuring responses are correct;
    c. Implement load tool timers to mimic end user think times.
4. Run a series of tests incrementally increasing virtual users from run to run until the expected number of active users to be supported is reached, a service level constraint is exceeded, or some target system resource is exhausted.
5. Develop a functional relationship between target

system resource consumption levels and number of virtual users supported.

6. Report active users supported and resource scalability results to decision makers using a mixture of graphs, tables, and text.

3.0 Virtual Users Vs Real Active Users

One of the primary objectives of any load test is to determine the number of active users supported by the application environment. The key testing statistic normally used to make this determination is the virtual user count that meets the service objective. Because they have such a significant role in simulating workload, what exactly are virtual users and how do they differ from the real thing? They are described in the load testing literature as simulated users which mimic the function and timing attributes of real users.

Virtual users, when applied to web application load testing, are a fixed set of load tool process threads running on a computer performing GET and POST requests in a closed loop while sleeping a think time between one response and the next request. The usual approach is to have all threads within a thread group perform the same set of requested events in sequence over and over during the testing period.

The traffic flow associated with this environment is shown in Figure 1 where 😐 represent the separate virtual user threads that make GET or POST requests.

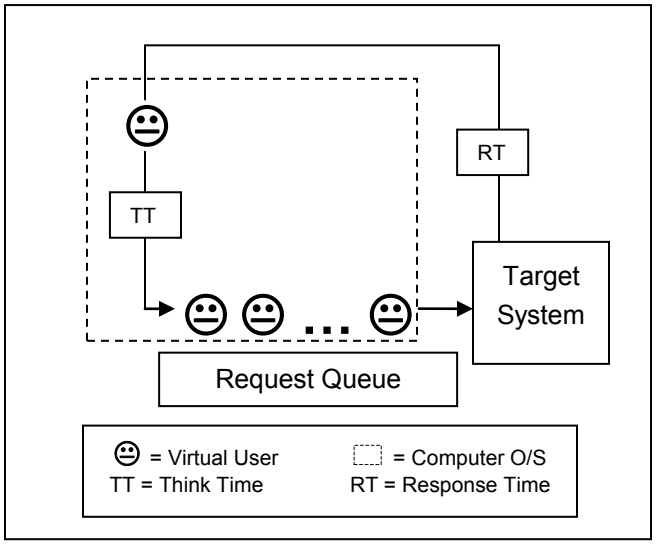


Figure 1: Virtual User Traffic Flow

Each 😐 performs the following steps in a closed loop:

1. Make a web request of the target system.
2. Wait for and time the response (RT).
3. Sleep for the assigned think time (TT).
4. Wake up and wait on the operating system scheduler's ready to run queue.
5. Execute the next event in the sequence when run by the operating system.

Think times are either fixed or drawn from some probability distribution provided by the load tool software. The one dotted line square surrounding all the 😐 represents the single computer operating system tasked with running the full set of virtual user threads.

How does a set of real active users differ from the simulated virtual users intended to represent them? Figure 2 is a traffic flow diagram of the real active user environment where a real user is a ☺ Vs its virtual user counterpart😐.

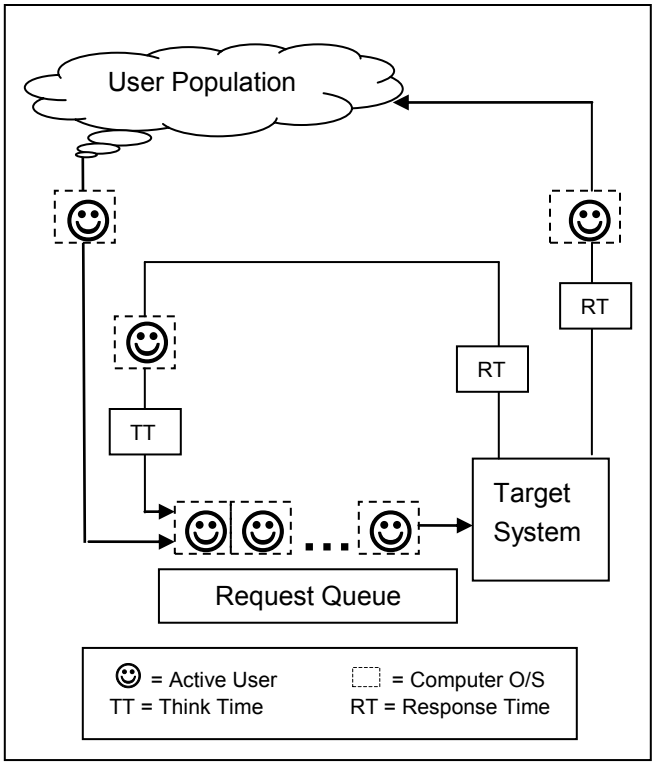


Figure 2: Real Active User Traffic Flow

Some of the lower portion of Figure 2 looks similar in structure to Figure 1 except its TT/RT loop shows each ☺ surrounded by a dotted line square indicating they have separate computer operating system environments. Figure 2 also contains the "User Population" loop which indicates some web requests offered to the target system come from the total population of users.

Figure 2 illustrates the fact that real active users are a variable size set of concurrent users which move in and out of the active user pool performing GET and POST requests on separate computers while thinking between each request. Active users supported are the expected number of real users actively making requests of the application at the objective service level.

What impact do these user differences have on the traffic pattern offered to the target application? The main differences can be categorized in three general areas:

1. Computer Resource Sharing,
2. User Participation,
3. User Relationships.

3.1 Computer Resource Sharing

Since the virtual user threads, 😐,all share the same computing resources, distortions in web page request timing and volume may take place due to excessive queuing for limited resources. For example, the sharing of processing execution resources increases contention for CPU cycles, adds to operating system run list size, and magnifies process thread scheduling complexity.

3.2 User Participation

The real users have an extended set of participation rules implied by the user population cloud. One of the implications of this population participation is the real users are not self-throttling like the virtual user TT/RT closed loop. Traffic can be offered to the Figure 2 target system in an unbridled way that will cause it to overload.

Although not shown pictorially, the Figure 1 and Figure 2 TT/RT closed loops differ in the way events are processed because real users are unlikely to follow the fixed order normally set up in virtual user scripts. The additional variety real users offer the target system is partially due to their broader set of web page options but also results from their being independent entities.

3.3 User Relationships

User individualism within its TT/RT closed loop combined with its population cloud requests implies real users behave far more independently than the virtual users in the typical load script. Therefore, load testing setups which make web page request timing less synchronous are likely to produce a better traffic pattern and lead to results that have a greater chance of matching real world experience.

Given this desire to maximize request timing independence, is there an easy way to recognize when independence exists? If there is a way to do that what modifications to the load test environment help achieve independence when it doesn't exist?

4.0 Quantifying Traffic Quality

Mimicking the independent behavior of real users can be difficult to accomplish in the limited resource closed loop environment of Figure 1 so it is important to understand the characteristics of independent events being offered to the target system. This understanding can be used as a guideline for the construction of load testing scripts which generate independent requests when executed by a set of virtual user threads. It can also be leveraged to develop methods for determining when request independence is reflected in data produced by test runs performed.

The first step in gaining these insights is to understand the mathematics of independent requests and how that mathematics relates to the aggregate traffic flowing to the target system.

Mathematically, transactions that are produced by a large population of users which have no coercion between them when pressing the enter key conform to the well known Poisson process [WIKI10]. To the extent that real user behavior is consistent with this no coercion principle, the characteristics of the Poisson process can be used to guide the construction of virtual user scripts and determine how well the aggregate traffic produced by them yields a real world pattern.

The Poisson process is characterized by times between arrivals being Negative-Exponentially distributed and the number of arrivals in constant length intervals possessing Poisson distribution attributes. This is a very powerful result and the basic arrival assumption for most practical queuing models including the well known A.K. Erlang formulas [WIKI12] applied in the telephone industry to size usage sensitive resources. A time line of arriving events under these conditions looks intuitively like they are occurring in bunches with intervals that are somewhere between evenly spaced and simultaneous.

Figure 3, is a pictorial representation of the Poisson process often referred to as a random arrivals pattern where the inter-arrival times, t, are represented by the non-uniformly spaced vertical bars and the frequency counts, x, are the numbers indicating the arrivals occurring within the uniform intervals.

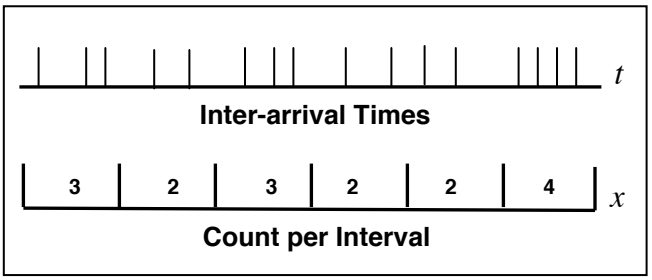


Figure 3: Random Arrivals

Figure 4 contains the Poisson process formulas where the values of t are Negative- Exponentially distributed, Equation 4.1, with a mean time between arrivals of μ and the x counts representing the number of arrivals in constant length intervals, are Poisson distributed, Equation 4.2, with a mean number of arrivals per interval of $\frac{1}{\mu}$. For example, if the mean time between arrivals is $\mu =$ 1/2 sec/arrival then, the mean number of arrivals per sec is $\frac{1}{\mu} = 2$ arrivals/sec.

$$f(t) = \frac{1}{\mu} e^{-\frac{t}{\mu}} \quad and \qquad (4.1)$$

$$p(x) = \frac{\left(\frac{1}{\mu}\right)^{x}}{x!} e^{-\frac{1}{\mu}} \quad for\ x = 0,1,... \qquad (4.2)$$

Where:

$\mu = mean\ time\ between\ arrivals, t$.

$for\ f(t), \quad \mu = mean = std\ dev = \sqrt{variance}$.

$\frac{1}{\mu} = mean\ number\ of\ arrivals, x,\ per\ time\ interval.$

$for\ p(x), \quad \frac{1}{\mu} = mean = variance$.

Figure 4: Poisson Process Formulas

Sampling estimates of mean, $\bar{x}$, variance, s^2, and std dev, s, for these distributions are listed in Figure 5.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \quad s^2 = \frac{\sum_{i=1}^{n} \left(x_i - \bar{x}\right)^2}{n-1}, \quad s = \sqrt{s^2} .$$

Where:

$x_i = is\ the\ i^{th}\ sample\ value.$

$n = is\ the\ sample\ size.$

Figure 5: Sample Statistics

Note the mean of the Negative-Exponential distribution equals its standard deviation and the mean of the Poisson distribution equals its variance. For an intuitive demonstration of the Poisson process using a one meter ruler and two sets of numbered chips see [BRAD09].

The mean equal standard deviation property of the Negative-Exponential opens the door to the possibility that offered traffic quality can be determined by comparing the mean and standard deviation of recorded event request time differences when events are sorted in launch time order. If these two statistics are approximately equal the traffic pattern is realistic but if they are not close to each other the traffic pattern is unacceptable and adjustments need to be made to the load environment.

For example, the web pages used to test an on-line State Budget Web Site are listed in Figure 6. The objectives of the test are to be certain the database is properly tuned and insure sufficient web site resources are in place when the Governor announces its availability to citizens at his State of the State address.

Budget Web Site - Pages Load Tested	
Web Page Name	**Purpose**
Home	Budget Home Page
by_department	Department View
by_department_xx	Department xx Details
by_function	Functional View
by_function_yyy	Function yyy Details
by_general_ledger	General Ledger View
by_general_leger_zzzz	General Ledger zzzz Details
by_revenue	Revenue View

Figure 6: Budget Web Site Pages Load Tested

Figure 7 is a segment of a test run event output file produced by the popular JMeter [JMETER12] load tool with the data sorted by request time, "Time Stamp". If the mean and standard deviation of the test run "Time Stamp" differences are computed and close to each other in value it is reasonable to assume random arrivals are being created and the traffic is high quality. If they differ significantly then modifications to the load generating environment are required.

Jmeter Run Events									
Time Stamp	Resp Time Page (ms)	Web Page Name	Resp Code	Resp Msg	Thread	Data Type	Success	Byte Count	Resp Time Byte 1 (ms)
1231531501681	421	by_department_xx	200	OK	Thread 1-1	text	TRUE	11623	421
1231531501945	219	by_function_yyy	200	OK	Thread 1-2	text	TRUE	11436	219
1231531502251	12	Home	200	OK	Thread 1-4	text	TRUE	12614	12
1231531502546	35	by_department	200	OK	Thread 1-1	text	TRUE	11853	35
1231531502773	14	Home	200	OK	Thread 1-1	text	TRUE	12614	13
1231531502873	37	by_department_xx	200	OK	Thread 1-4	text	TRUE	11623	37
1231531502956	11	Home	200	OK	Thread 1-3	text	TRUE	12614	10
1231531503898	35	by_department	200	OK	Thread 1-1	text	TRUE	11853	34
1231531503975	33	by_department	200	OK	Thread 1-7	text	TRUE	11853	33
1231531503991	39	by_department_xx	200	OK	Thread 1-9	text	TRUE	11623	39
1231531504116	42	by_department_xx	200	OK	Thread 1-4	text	TRUE	11623	41

Figure 7: Budget Web Site JMeter Test Run Events

As an illustration of these inter-arrival time calculations, the "Time Stamp" difference between the first two events listed in Figure 7 is 264 milliseconds (1231531501954 – 1231531501681 = 264). This calculation is repeated between adjacent timestamps and each pair of timestamps by web page name, e.g., by_department_xx. The mean and standard deviation of the differences are computed and compared for equality.

Unfortunately, arrival pattern statistics are not normally reported by load generating tools so this author wrote a Perl script to process the data. Figure 8 is the inter-arrival time report produced by that script for the full

n = sample size

Inter-arrival Summary Statistics (ms)				
Web Page Name	n	tps	mean	sdev
Home	81322	24.63	40.60	43.11
by_department	20140	6.10	163.75	168.34
by_department_xx	20341	6.17	162.17	167.75
by_function	20287	6.15	162.54	167.32
by_function_yyy	20534	6.23	160.55	166.59
by_general_ledger	20423	6.19	161.60	169.88
by_general_leger_zzzz	20153	6.11	163.69	171.95
by_revenue	20083	6.08	164.39	171.65
Total	223290	67.63	14.79	15.93

Figure 8: Budget Web Site Inter-arrival Statistics

Figure 7 file and shows the mean and standard deviation of the inter-arrival times (green) are close to each other for all web page name events as well as the total set of events. These statistics indicate independent requests are being produced by the testing setup.

5.0 Improving Traffic Quality

If the inter-arrival time standard deviation significantly differs from the mean the traffic generation environment needs adjusting or the simulated load will not match the live application. What steps can be taken to bring a load testing setup in line with this statistical relationship when out of compliance?

Four suggested adjustments are:

1. Choose a different load generator delay timer.
2. Alter the technique used to increase load.
3. Randomize event order where possible.
4. Change the number of load computers.

5.1 Choose a different load generator delay timer

The first potential adjustment suggested is to choose a different load generator delay timer to help make requests look more independent from one another. Given the previous arguments, the obvious choice is the Negative-Exponential distribution because it produces the desired independent delay interval. After this timer change make a test run and check the inter-arrival statistics to determine if they possess the same equality characteristics as Figure 8.

There is a potential problem with this suggestion, however, because as Figure 9, the JMeter timer list reveals, there is no Negative-Exponential distribution option. This is a baffling outcome since delays distributed in this manner occur over a wide range of application environments. It should be noted that JMeter is not the only product which excludes the Negative-Exponential from its timer list and a cursory review of popular load tools reveals exclusion is the rule, not the exception.

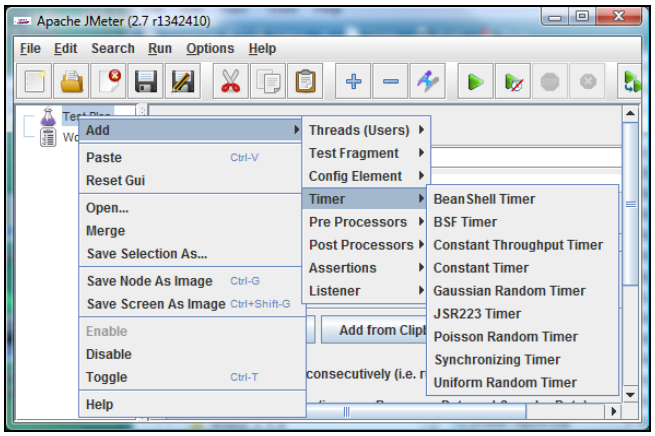


Figure 9: JMeter Delay Timers

The omission of the Negative-Exponential timer cannot be because it is a complex expression and difficult to implement. The formula for drawing the time to delay is Equation 5.1 in Figure 10 which simply says multiply minus the mean think time by the natural log of a random number between zero and one.

$$t_0 = -\mu \ln(r_0) \qquad (5.1)$$

Where:

$t_0 = think\ time\ until\ next\ web\ page\ request.$

$\mu = mean\ think\ time.$

$\ln = natural \log (\ln e = 1).$

$r_0 = random\ number\ 0 \le r_0 \le 1.$

Figure 10: Negative-Exponentially Distributed Think Times.

The Java line of code that represents Equation 5.1 and the function call used to delay the time returned from that call are contained in Figure 11.

```
t = (long)(-(double)mu*Math.log(Math.random()));
Thread.sleep(t);

Where:
t = think time until next web page request.
mu = mean think time.
```

Figure 11: Negative-Exponentially Distributed Think Time Java Code.

A derivation of Equation 5.1 using the Negative-Exponential probability density function, Equation 4.1, as a starting point can be found in [BRAD04] or [BRAD06].

Fortunately, independent requests can be produced anyway using the available random draw probability distributions (Poisson, Gaussian and Uniform) as long as there are a sufficient number of virtual user threads running. This mechanism works because it can be shown mathematically that the superposition of independent arrival processes which come from any distribution will approach a random arrival process as their number increase [KARL75] and [ALBI82]. The concept is analogous to the Normal (Gaussian) distribution's limiting properties when summing random variables. The Figure 9 Uniform Random Timer was implemented for all the examples in this paper.

5.2 Alter the Technique Used To Increase Load

If the Negative-Exponential distribution timer option is not available then superposition dictates that a large virtual user count is needed to produce an independent request environment. One way to be reasonably sure this requirement will be met is to fix the number of user threads at the maximum number needed for all tests within the constraints of the load generating computer's resources. Traffic is increased from test to test by

lowering the think time parameter(s) while maintaining a consistent transaction mix. The fundamental concept behind this fixed virtual user technique is to make these threads a set of independent transaction generators.

Some advantages to this fixed virtual user approach are:

1. Provides a consistent ramp-up environment by making the thread startup count constant from one scalability test level to the next.
2. Creates a stable process/thread set for the load computers', memory, processors, and O/S scheduler across test runs.
3. Incorporates a higher think time proportion of total transaction loop time (TT+RT) reducing the impact of the response time distribution on the arrival pattern.
4. Removes the need for one virtual user per load test active user. In [BRAD11] 325 virtual users are invoked to simulate the traffic produced by 1000 active users.

Traffic quality may improve using this fixed virtual user technique but how can active users supported be determined since there is no longer a one to one relationship between active and virtual users?

Under these fixed user thread conditions, the number of active users supported at a specific transaction rate is computed by multiplying the transaction rate by the mean thread cycle time (TT+RT). This technique is used to create the Figure 12 active users supported matrix for the range of composite mean think times and load test transaction rates shown. The Trans/Sec levels for the six test runs are produced with the same set of 325 user threads by adjusting think time settings and maintaining a transaction mix exemplified by the Test Run 6 inter-arrival data detailed in Figure 8.

				Mean Think Time (Sec)				
User Threads = 325				5	10	15	20	25
Test Run	CPU % Use	Trans/ Sec	RT (Sec)	Active Users Supported				
0	0	0.00	0.00	*0*	*0*	*0*	*0*	*0*
1	22	15.54	0.32	*83*	*160*	*238*	*316*	*393*
2	37	30.33	0.35	*162*	*314*	*466*	*617*	*769*
3	52	46.06	0.45	*251*	*481*	*712*	*942*	*1172*
4	63	58.74	1.05	*355*	*649*	*943*	*1237*	*1530*
5	67	62.90	2.70	*484*	*799*	*1113*	*1428*	*1742*
6	70	**67.64**	**4.78**	*662*	***1000***	*1338*	*1676*	*2014*

Figure 12: Budget Web Site Active Users Supported

As an example of how the Figure 12 active user values are computed, the 1000 active user level defined by the intersection of the Test Run 6 row and the 10 second think time column is calculated as follows; 67.64 x (10 + 4.78) = 1000. That is to say, 1000 users, each with a 14.78 second average transaction cycle time, collectively offer 67.64 Trans/Sec to the target environment.

Because think times vary and are educated guesses at best, a table like Figure 12 can be a useful indicator of how sensitive active user support levels are to think time estimates.

5.3 Randomize Event Order Where Possible

If there are still issues with traffic pattern another step is to randomize the order of user events as much as possible. This technique can help in situations where all virtual user threads follow a fixed sequence and specific events have inherently long response times causing threads to bunch up behind these long latency events. This stacking up can cause a systematic pattern of arrivals to occur which does not happen in the real world where each user has his own operating system task queue and makes requests independently.

Figure 13 is the JMeter layout of the Budget Web Site load test script and shows all web pages being selected randomly. Biases in web page selection are created by replicating the page request event as is the case for the Home page which is repeated four times.

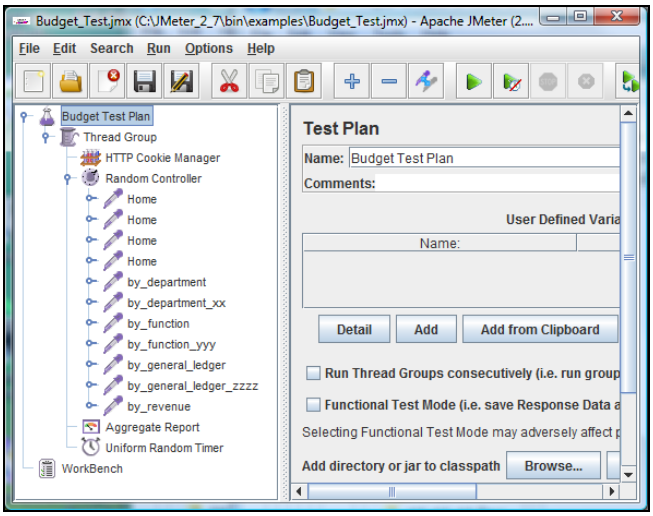


Figure 13: Budget Web Site JMeter Script

Of course the web application's logic may force a particular page sequence like logon and logoff. For this forced order situation consider logging on each user thread once per test, accessing the remaining web pages in random order where possible until the run completes, and never logging off. After all, real users seldom logoff so why should the load tool? This strategy puts logon near the beginning of the test at a fixed count and is consistent with workers logging on to start the day.

5.4 Change the Number of Load Computers

Establishing the right combination of delay timer and virtual user thread count along with randomizing the order of events may produce a more realistic random arrivals traffic pattern but what if the CPU and Memory capacity of a single traffic generating platform is insufficient to yield the needed load? As Figure 14 illustrates, the memoryless property of the Negative-Exponential distribution makes load expansion beyond a single computer seamless.

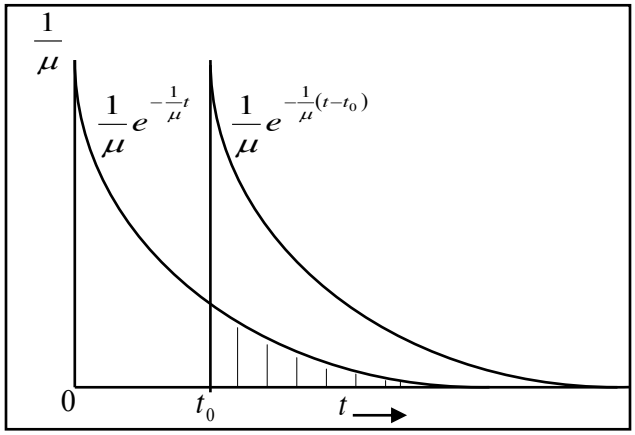


Figure 14: Negative-Exponential "Memoryless" Property

This figure shows the Negative-Exponential density $\frac{1}{\mu}e^{-\frac{1}{\mu}t}$ and its value $\frac{1}{\mu}$ at time 0 on the left side. After t_0 has elapsed, the density function for the time until the next arrival is computed by magnifying the portion of this curve to the right of t_0 (shaded) and increasing its area to unity yielding $\frac{1}{\mu}e^{-\frac{1}{\mu}(t-t_0)}$. The shape of this new density function is identical to the original and is only shifted in time. Since the shape of the distribution remains the same everywhere on the time line it is called time invariant, or memoryless.

As Kleinrock [KLEI75] indicates, "No other density function has the property that its tail everywhere possesses the exact same shape as the entire density function." Gunther [Gun05] describes the memoryless property of the Negative-Exponential with a numerical illustration and a counter example using the Normal probability distribution.

Since the events within a random arrivals stream are memoryless, it can be shown mathematically that the merger of multiple memoryless streams is also memoryless and can be viewed as a single independent stream with intensity equal to the sum of the individual stream intensities [GIFF78].

This random arrivals stream merging property is illustrated in Figure 15 for the number of arrivals per interval Poisson distribution where each a_i is a Poisson distributed random variable whose sum, A, the total arrivals per unit time, is also Poisson distributed with mean equal to the sum of the a_i means.

Therefore, increasing the number of load generating computers can be done seamlessly without altering the flow properties of the traffic produced and traffic quality can be checked by comparing inter-arrival time mean and standard deviation across individual load generator event files. When load generator clocks are synchronized using say, NTP, the separate event files can be merged and the aggregate inter-arrival pair of statistics computed and compared for equality.

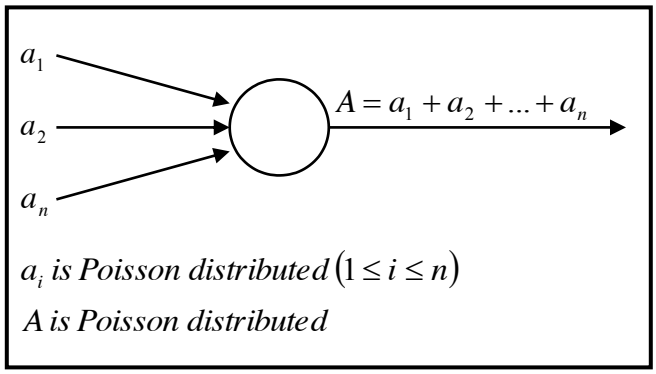


Figure 15: Combining Poisson Distributed Streams

Figure 8 illustrates the independent stream merging concept at the single test and load computer level since each web page name, the quadrupled home page, and the total events list possess inter-arrival time mean and standard deviation equality. No matter how the traffic is sliced, random arrivals is preserved within a single load generator as well as across multiple generators.

The memoryless property of the Negative-Exponential is exploited in many computing system component designs such as Ethernet where it is applied within the retry timing mechanism to make collision events appear to be first offered attempts when reinitiated.

6.0 The Virtual User's Role in Scalability Testing

When increasing load in the traditional way by adding virtual users there is a perception these user threads are traffic but in reality they are traffic sources. This is an important distinction when addressing the issue of resource scalability. Perhaps the best way to sort out resource scalability factors is return to the active users supported matrix in Figure 12 and the CPU % Use column in the same figure.

For illustrative purposes, turn the situation around and assume the 10 second think time column for active users supported represents a series of virtual user quantities implemented in a scalability test which yields the Trans/Sec, RT (Sec), and CPU % Use values listed. For example, assume Test Run 6 was performed with 1000 virtual users, a composite think time of 10 second yielding 67.64 Trans/Sec and a thread cycle time (TT+RT) of 14.78 seconds.

Since Figure 12 contains CPU % Use information, an X-Y plot of that statistic as a function of the appropriate independent variable can determine processor scalability. A plot of CPU % Use Vs the virtual users in the 10 second column is performed and shown in Figure

16. This chart has a trend line that deviates significantly from the actual line. The actual line nearly flattens out and is almost horizontal at the highest active user level plotted, implying that fewer CPU resources are required to process the incremental load from 800 to 1000 users than from 400 to 600 users. This "supports the same incremental number of users with far fewer resources" result is counter intuitive and is so because traffic sources do not logically scale with CPU resources.

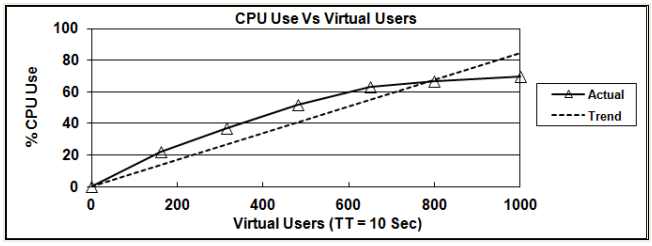


Figure 16: CPU % Use Vs Virtual Users

In contrast, Figure 17 is an X-Y plot of CPU % Use as a function of transaction rate using the data contained in the Trans/Sec column of Figure 12. Since the trend line in Figure 17 is nearly coincident with the actual line connecting the six data points it is reasonable to assume CPU resources do scale with transaction rate to at least 70% Use.

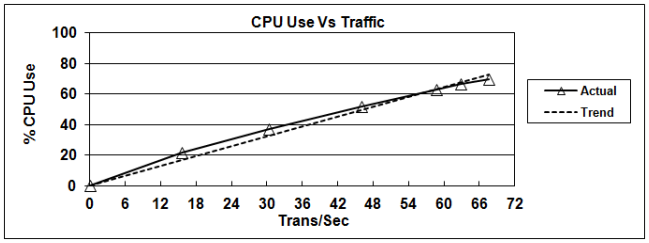


Figure 17: CPU % Use Vs Trans/Sec

Why the difference between the two plots? The short answer is that active or virtual users are not traffic but generate the traffic for target servers to process. Looking more closely, virtual user event cycle time includes response time in addition to think time. As Figure 18, a plot of Figure 12 response times as a function of Trans/Sec illustrates, these times are typically a non-linear function of load at higher traffic levels.

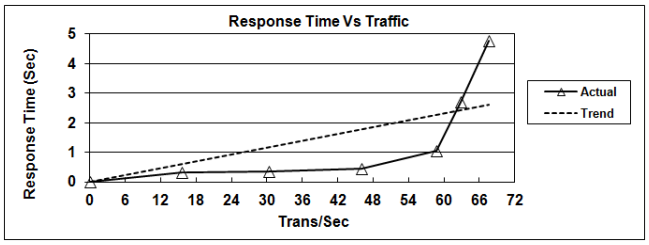
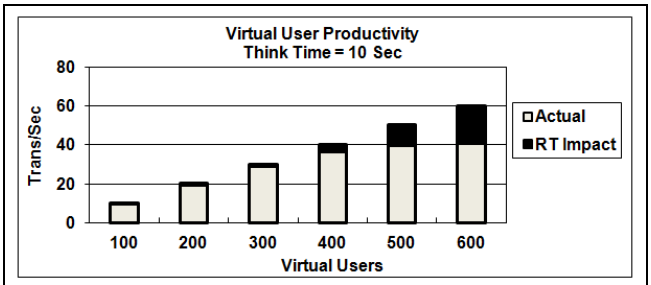


Figure 18: Response Time Vs Trans/Sec

Therefore, response time is a much greater proportion of user thread cycle time at higher traffic rates making each thread less effective as a producer of transactions.

This virtual user productivity argument is more clearly understood within the context of the traditional traffic ramp up technique that is illustrated in Figure 19. This figure depicts virtual user thread productivity when their count is increased incrementally for the10 second mean think time and the response times in Figure 18. As shown, the 4.78 second response time in Test Run 6 versus the Test Run 1 value of .32 seconds increases the average thread cycle time from 10.32 seconds to 14.78 seconds, a 43% loss in efficiency. Without the impact of response time on cycle time Test Run 6 would offer 60 Trans/Sec to the target test environment but it only delivers 40.6 Trans/Sec.



		Time (Sec)			Trans/Sec		
Test Run	Vusers	TT	RT	TT+ RT	TT Only	Actual	RT Impact
1	**100**	10	0.32	10.32	10	**9.69**	**0.31**
2	**200**	10	0.35	10.35	20	**19.32**	**0.68**
3	**300**	10	0.45	10.45	30	**28.71**	**1.29**
4	**400**	10	1.05	11.05	40	**36.19**	**3.81**
5	**500**	10	2.70	12.70	50	**39.37**	**10.63**
6	**600**	10	4.78	14.78	60	**40.60**	**19.40**

Figure 19: Virtual User Productivity

This illustrates that treating the virtual users as if they are target system traffic can be misleading and show highly scalable resources to not be scalable at all. This misleading result occurs even when the traffic produced is high quality because traffic Vs traffic sources is the issue, not traffic pattern.

7.0 Example Load Test

The following example load test is intended to illustrate application of the techniques just described to produce real world results and demonstrate correct scalability analysis techniques. The example chosen is a web site where citizens obtain state government statistics that is being reconfigured from standalone servers to a virtualized load sharing environment. The names of the web pages being tested are listed in Figure 20.

GOV Web Site - Pages Load Tested	
Web Page Name	**Purpose**
010_Home	Home Page
012_Home_jpg	Background Image
020_Department	Department Information
022_Department_jpg	Department Image
030_Demographics	Demographic Information
040_Statistics	Summary Statistics

Figure 20: GOV Web Site Pages Load Tested

Figure 21 is a topological view of the load testing

environment showing load generator, network interfaces, F5 Load Balancer, and the virtualized Blade Server setup with GOV virtual servers “1” and “2”.

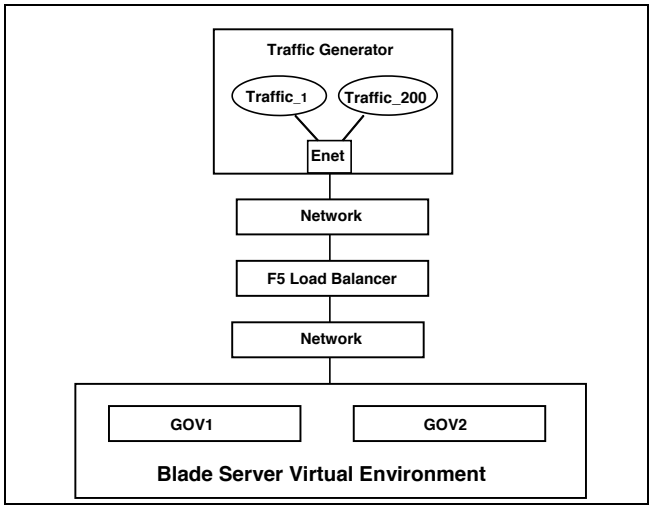


Figure 21: GOV Test Traffic Generation Topology

The major test objectives are to determine the scalability of the two virtual servers and evaluate how well the F5 balances the load between them. Tests are performed on a single traffic generator running the JMeter script in Figure 22.

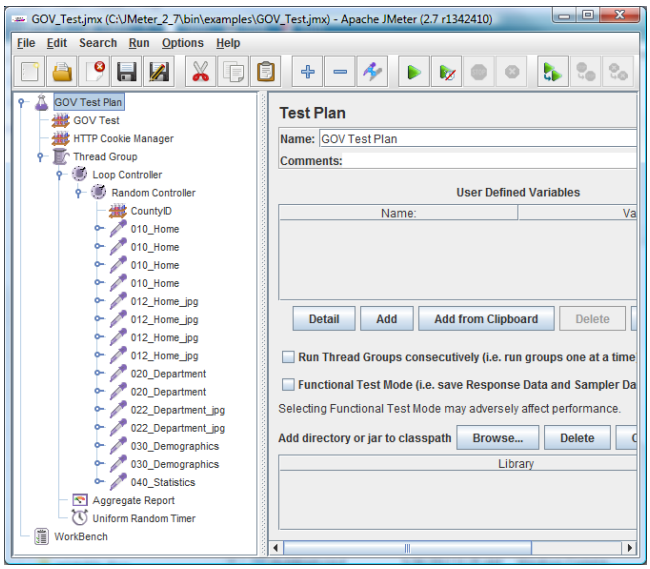


Figure 22: GOV Web Site JMeter Script

The layout of this script may seem a bit strange at first but it is structured to maximize the quality of the traffic produced by applying the ideas presented in this document. The overall test implementation is as follows:

1. Web pages are accessed in random order with multiple instances of them implemented to bias the event count. For instance, there are four 010_Home page instances Vs one 040_Statistics page instance.
2. As the Traffic_200 bubble in Figure 21 implies, two hundred JMeter threads (virtual users) are used for all tests. Traffic is increased from test to test by reducing the mean think time of the one uniform random timer in Figure 22.
3. The load generator, the F5, and the GOV virtual servers are housed in the same building with a 100 megabit link connecting the load generator to the higher bandwidth building network.
4. All web page requests go through DNS and are directed to the F5 Load Balancer which distributes the work to the two virtual servers.
5. Seven 25 minute tests are executed which can be identified by the test start time, e.g., the 1800 test starts at 6:00 PM and ends at 6:25 PM.
6. Assertions are implemented to be sure correct responses are returned. Response assertions are used for html pages and size assertions are implemented for the jpg files.

Because transaction mix consistency is a key ingredient in scalability testing, this JMeter test script is structured to deliver mix consistency across all seven tests performed.

It is customary to produce test output as a set of time series charts with resource activity level plotted as a function of wall clock time like shown in Figure 23 but this method of summarizing results doesn’t generally distill the information in a way management can absorb efficiently. The decision maker is shown chart after chart in time series format and eventually provided the bottom line result in a few words with no intuitively appealing plots directly reflecting the conclusions drawn.

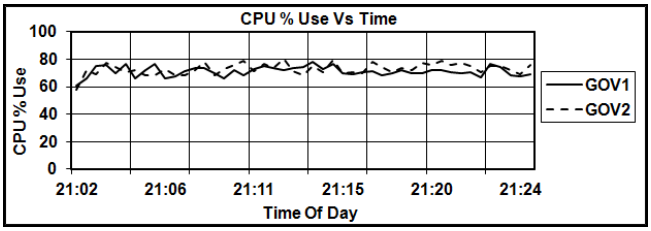


Figure 23: GOV Web Site CPU % Use Vs Time

All of the graphs summarizing the GOV load test results which follow use a different approach removing time as the independent variable and replacing it with Trans/Sec. The graphs produced are X-Y plots using a statistic like mean response time as the dependent variable with each data point graphed reflecting a test run value, e.g., the 2100 test. A discussion of time series data aggregation into statistical information for decision making is found in [BRAD10].

With this reporting structure as background, the results that follow are categorized into three main topics and some concluding comments.

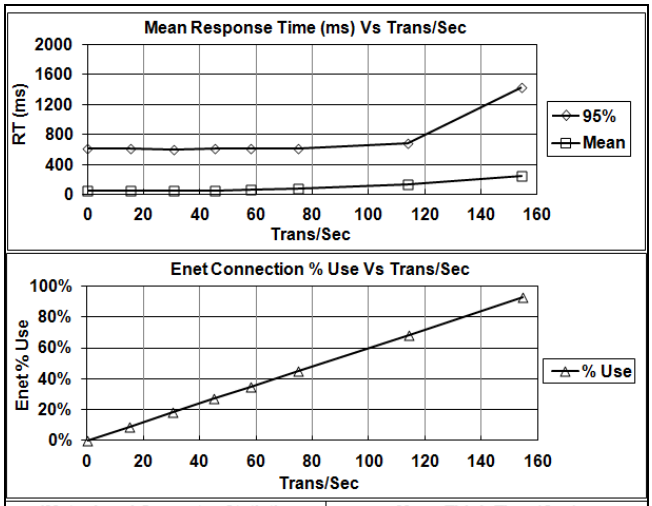
1. Active Users Supported
2. Target System Scalability
3. Traffic Quality Determination

Each topical section contains a set of X-Y plots and a supporting table possessing the graphed statistics.

7.1 Active Users Supported

Figure 24 is a summary of the test results from a load generator or aggregate user perspective. The top graph contains mean and 95% response time statistics as a function of Trans/Sec, the graph below it is a similar plot for Enet % bandwidth used, and the table at the bottom provides the data used to produce the two graphs on the left and an Active Users Supported matrix on the right.

The Active User Supported matrix is developed by applying the same technique used in Figure 12. For example, Test Run 2100 yields 154.56 Trans/Sec with a mean response time of .249 seconds. If the mean think time is 30 seconds 4675 active users are supported, 154.56 x (.249 + 30) = 4675 active users.



JMeter Load Generator Statistics						Mean Think Time (Sec)				
	Load	RT (ms)		Enet		10	20	30	40	50
Test Run	Trans/ Sec	Mean	95%	Rec KB/s	% Use	----------Active Users Supported----------				
0	0.00	52	607	0	0%	*0*	*0*	*0*	*0*	*0*
1800	15.25	52	607	917	9%	*153*	*306*	*458*	*611*	*763*
1830	30.56	53	606	1843	18%	*307*	*613*	*918*	*1224*	*1530*
1900	45.18	54	607	2713	27%	*454*	*906*	*1358*	*1810*	*2261*
1930	58.15	60	611	3496	35%	*585*	*1166*	*1748*	*2329*	*2911*
2000	74.96	74	616	4508	45%	*755*	*1505*	*2254*	*3004*	*3754*
2030	114.08	131	690	6840	68%	*1156*	*2297*	*3437*	*4578*	*5719*
2100	**154.56**	**249**	1437	9280	93%	*1584*	*3130*	***4675***	*6221*	*7766*

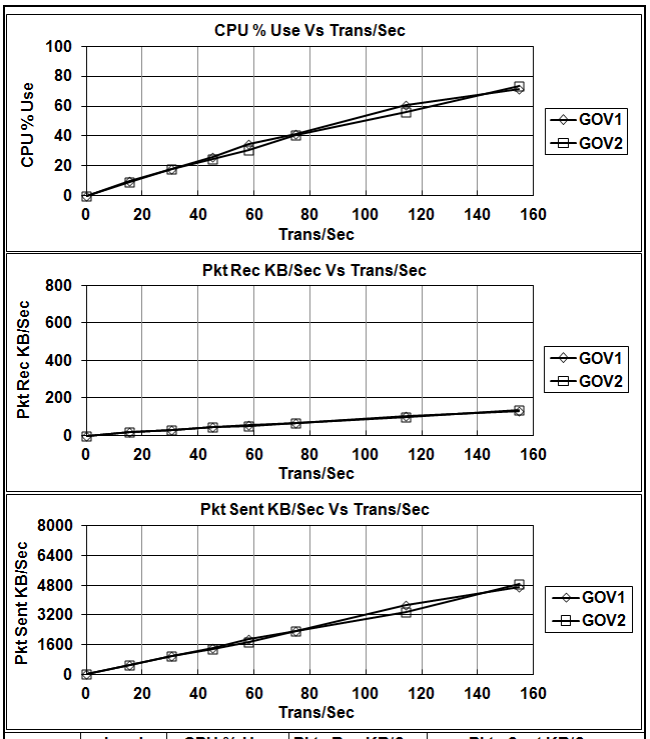
Figure 24: GOV Test Results Aggregate User View

An average think time of 30 seconds may seem like a long interval for a read only web site but it is a typical user experience within the context of this application environment. Since, however, there is nearly always uncertainty surrounding the actual think times produced by the set of Active Users, the five columnar user levels are listed. These quantities help make this uncertainty explicit and permit results reviewers to perform sensitivity analysis.

7.2 Target System Scalability

Figure 25 is a set of three target system resource usage X-Y plots and associated table values including CPU % Use, Pkt Rec KB/Sec, and Pkt Sent KB/Sec graphed as a function of Trans/Sec. Other resource statistics such as real memory pages/sec and disk I/O rates were recorded but are not shown since usage was negligible.

A cursory view of the three graphs in Figure 25 shows the F5 is balancing the workload very well across GOV1 and GOV2 since there is little separation between the GOV1 and GOV2 lines for all seven tests by resource charted.



	Load	CPU % Use		Pkts Rec KB/Sec		Pkts Sent KB/Sec		
Test Run	Trans/Sec	GOV1	GOV2	GOV1	GOV2	GOV1	GOV2	GOV1+GOV2
0	0	0	0	0	0	0	0	0
1800	15.25	9.83	9.12	19	19	479	481	960
1830	30.56	17.89	17.78	32	32	963	957	1920
1900	45.18	25.75	24.65	44	43	1435	1387	2822
1930	58.15	34.52	30.47	57	53	1896	1742	3638
2000	74.96	41.48	40.81	69	68	2358	2331	4689
2030	114.08	60.54	55.92	106	96	3726	3382	7107
2100	154.56	71.36	73.28	133	137	4726	4887	9613

Figure 25: GOV Test Results Target Server View

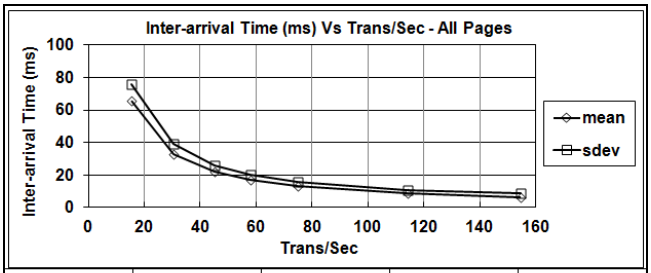
A comparison of the Enet Rec KB/Sec statistics in Figure 24 Vs the last column in Figure 25, Enet Sent KB/Sec for GOV1+GOV2 yields a good data validation cross-check. As shown, these two statistics are reasonably close to each other at all transaction test run levels indicating the traffic leaving the target system is approximately the same as that received by the traffic generator. As an example, Run 2100 yields, TG = 9280 Rec KB/Sec and GOV1+GOV2 = 9613 Sent KB/Sec.

From a target system scalability perspective it is clear the key limiting resource, CPU, is very scalable through the 71%+ CPU % Use in Run 2100.

7.3 Traffic Quality Determination

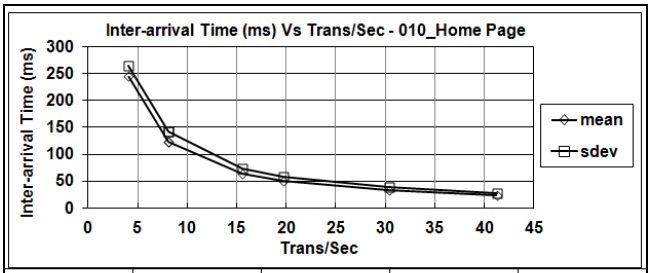
Traffic quality is not typically part of the results analysis process but, as mentioned in Section 1.0, poor quality

test traffic is revealed by the live application when it is too late to make improvements and credibility is lost. Figure 26 contains an X-Y plot and data table of inter-arrival time mean and sdev statistics as a function of Trans/Sec for all events recorded on a test run basis. Figure 27 contains the same information as Figure 26 but only includes the 010_Home page events. Figure 28 is similar to Figure 27 but is based on one instance of the 040_Statistics page Vs the four instances of the 010_Home page.



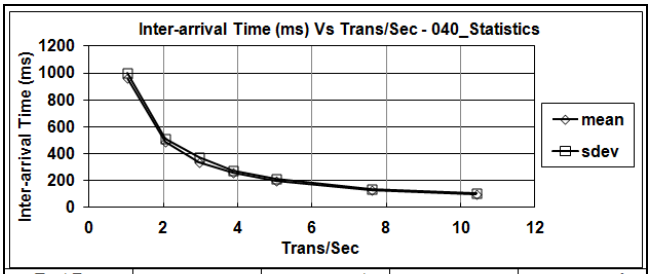
Test Run	n	tps	mean	sdev
1800	23171	15.25	**65.58**	**75.77**
1830	46110	30.56	**32.73**	**38.83**
1900	68082	45.18	**22.13**	**25.97**
1930	87516	58.14	**17.19**	**20.33**
2000	112716	74.96	**13.34**	**15.71**
2030	171376	114.08	**8.77**	**10.78**
2100	231931	154.56	**6.47**	**8.50**

Figure 26: GOV Inter-arrival Stats – All Pages



Test Run	n	tps	mean	sdev
1800	6179	4.08	**244.84**	**264.63**
1830	12205	8.10	**123.46**	**142.48**
1900	12205	8.10	**123.46**	**142.48**
1930	23396	15.56	**64.29**	**73.00**
2000	29655	19.76	**50.62**	**57.82**
2030	45653	30.39	**32.90**	**38.98**
2100	61919	41.31	**24.21**	**27.39**

Figure 27: GOV Inter-arrival Stats - 010_Home



Test Run	n	tps	mean	sdev
1800	1547	1.04	**965.45**	**996.67**
1830	3096	2.06	**484.83**	**512.91**
1900	4439	2.96	**337.53**	**375.21**
1930	5817	3.87	**258.52**	**276.95**
2000	7571	5.04	**198.19**	**211.58**
2030	11388	7.60	**131.45**	**140.08**
2100	15610	10.42	**95.92**	**103.19**

Figure 28: GOV Inter-arrival Stats - 040_Statistics

All three graphs show a mean and sdev curve very close to each other in value indicating the desired random arrivals traffic pattern is produced. Whether traffic is aggregated by web page for a specific test run as in Figure 8, by a specific web page across test runs like Figure 27 and Figure 28, or all pages across test runs as in Figure 26, the mean and sdev inter-arrival time statistical equality is preserved.

As discussed in Section 5.4, this relationship between the two statistics holds even across multiple traffic generators and the data from all of them can be merged for analysis purposes if their clocks are in sync.

7.4 GOV Test Conclusions

These test results demonstrate high quality traffic is produced by the load generating environment. They further indicate the GOV virtual environment is balanced and tuned and capable of supporting a little less than 5,000 active users with a mean think time between web page requests of 30 seconds.

It should be noted this simple example is chosen for conceptual clarity purposes but the ideas illustrated apply equally well in more complex large user population circumstances as long as there is no coercion between users when hitting the enter key.

8.0 Summary

When load testing large user population web applications the devil truly is in the virtual user details. This discussion shows that simply attempting to mimic the real world by creating a virtual user per active user, while ignoring fundamental traffic principles, gives the analyst a false sense of security that the load test being performed yields valid results. As Figure 1 in Section 3 illustrates, a load generating computer is one traffic source attempting to operate like the large number of independent sources in Figure 2. The discussion in Section 4 provides a way to make a determination of how close the load testing setup is to accomplishing that mission.

Section 5 contains traffic quality improvement suggestions and Section 6 describes post testing scalability analysis methods that are useful as well as those that are misleading. Section 7 puts it all together with an example load test that shows how to compute active users supported from traffic data, determine the scalability of target system resources by applying appropriate statistical inference techniques, and demonstrate the robustness of the traffic generated by aggregating the inter-arrival data into various web page event levels.

Virtual Users are an important part of any load test and maximizing traffic quality should be a primary goal of their implementation. Otherwise, the live application will

reveal their shortcomings and the test will be viewed by management as a waste of time and money.

References

[ALBI82] S.L. Albin, "On Poisson Approximations For Superposition Arrival Processes In Queues", Management Sciences, Vol. 28, No2, February 1982.

[BRAD04] J. Brady, "Traffic Generation Concepts – Random Arrivals," www.perfdynamics.com, Classes, Supplements, 2004.

[BRAD06] J. F. Brady, "Traffic Generation and Unix/Linux System Traffic Capacity Analysis," CMG Journal, 117:12-20 (Spring 2006).

[BRAD09] J. F. Brady, "The Rosetta Stone of Traffic Concepts and Its Load Testing Implications," CMG MeasureIT, (September 2009).

[BRAD10] J. F. Brady, "Making Statistical Sense out of Time Series Data with 'Home Grown' Perl Scripts," CMG MeasureIT, (August 2010).

[BRAD11] J. F. Brady, "Putting the Virtual Users You Use for Load Testing In Their Place," CMG MeasureIT, (November 2011).

[GIFF78] W.C. Giffin, *Queueing: Basic Theory and Applications*, Grid, Inc, Columbus, Ohio, 1978.

[GUN05] N. Gunther, *Analyzing Computer System Performance with Perl::PDQ*, Springer-Verlag, Berlin Heidelberg, 2005.

[JMETER12] The Apache Software Foundation, "Apache JMeter", (2012), http://jmeter.apache.org/index.html

[KARL75] S. Karlin, H.M. Taylor, "A First Course In Stochastic Processes", Academic Press Inc., New York, N.Y., (1975).

[KLEI75] L. Kleinrock, "Queueing Systems Volume 1 and 2", John Wiley & Sons, New York, N.Y., (1975).

[WIKI10] Wikipedia, "Poisson process", (2010) http://en.wikipedia.org/wiki/Poisson_process.

[WIKI12] Wikipedia, "ERLANG", (2012) http://en.wikipedia.org/wiki/Erlang_(unit)

Copyrights and Trademarks