Putting the Virtual Users You Use for Load Testing In Their Place

James F. Brady

My new web application needs to support a thousand active users and I am planning to perform a load test before it goes live. How many virtual users do I need to properly complete the test and what role do these simulated users play in the effort? The answer to these questions may surprise you if your test objectives include producing real world traffic and generating meaningful resource scalability numbers.

Introduction

Are a thousand virtual users really needed to simulate the load that will be produced by the active users anticipated to access the new web application? That number of user threads will require multiple load generators but if the test configuration is assembled with a simulated user per active user, isn't the traffic produced guaranteed to look real world? The answer to these questions is "no" and the discussion which follows explains why. As a by-product of this discussion key statistics are produced which help determine traffic quality and an illustration is provided that shows virtual user count is not a reliable resource scalability indicator.

Typical Load Testing Setup

When performing a load test virtual users are an intuitively appealing substitute for real users. They are scripted to perform a sequence of application steps with think time delays between events just like real users. Their creation and implementation when testing resource scalability is usually viewed as a reasonably mechanical process:

1. Identify an application event sequence of interest,
2. Record the sequence with the load tool's recording software,
3. Set up user think time parameters,
4. Run a series of tests incrementally increasing virtual users until the expected real user population is reached.

This seems pretty straight forward but when the real user population is very large, e.g., 1000, the one to one correspondence between virtual and real users can add complexity to the testing environment by requiring a large quantity of load generating equipment. This added complexity and expense can be avoided, or at least minimized, if the virtual user's role in large user population environments is well understood.

Individual User Significance

What does a virtual user represent within the context of the load generation queuing environment? Virtual users are traffic sources, the transactions these sources initiate are the offered traffic, and the computers which process the transaction constitute the system servers. The "traffic source" users occupy three potential states, thinking, queuing, and being served. When transitioning from thinking to making a request they are initiating traffic in the form of a transaction which is processed by the servers. The mix and volume of transactions produced is a function of the event mix, sequence, and generation rate.

When user populations are small the state of the individual sources has a major impact on the traffic produced because a source that is either queuing or in service cannot be an arrival. The state of a particular source has much less impact when the number of sources moves toward the large user population being discussed. Figure 1 confirms this scaling behavior with an example which compares the probability of queuing for a small population random arrival queuing system with its infinite population counterpart. The finite source (/M/M/C/m/m) queuing system converges to the infinite source equivalent (M/M/C) in this example when the number of sources approaches 100. As expected, there is no waiting in the finite source situation when the number of sources is less than or equal to the number of servers, i.e., 4. The infinite source model makes no such distinction, providing the same service level for all source quantities because it is driven solely by the aggregate traffic offered from all sources and the number of available servers. This finite to infinite source convergence rate varies as a function of server quantity and offered traffic level but the principle holds in general. This example demonstrates the concept that as user count increases the focus shifts from traffic produced by individual users to that generated by all users.

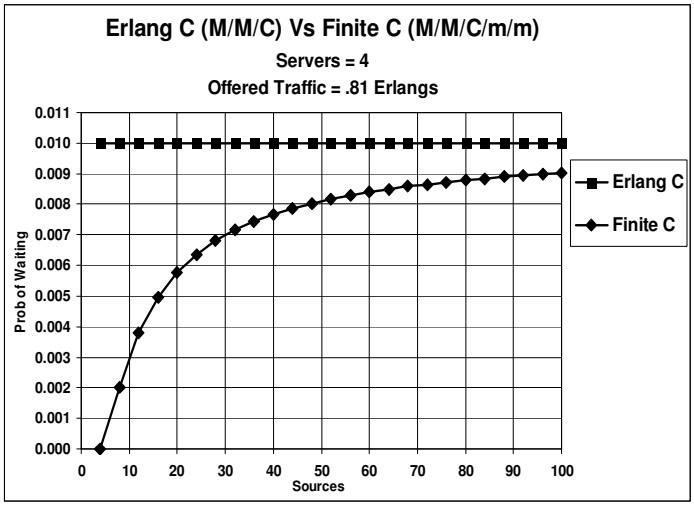


Figure 1: Finite Source Convergence to Infinite Source Model [BRAD07]

Traffic Pattern

The initial "M's" in the Figure 1 (/M/M/C/m/m) and (M/M/C) queuing model designations indicate they are based on a random arrival traffic pattern which is referred to in the literature as a Poisson process [COOP84] [WIKI11]. Random Arrivals represents a broad array of large population traffic flow environments that exist in telecommunications as well as computing and is characterized by a traffic pattern where there is no coercion between users requesting service from the system. This arrival pattern has the property that the number of arrivals in non-overlapping constant length intervals is Poisson distributed and the times between arrivals are Negative-Exponentially distributed.

This author used a "One Meter Ruler" to provide an intuitive demonstration of this model in a previous MeasureIT article [BRAD09]. The intent of this demonstration is to illustrate that independent request environments appear to be chaotic on the surface but actually possess a great deal of mathematical structure which can be exploited when attempting to construct a real world load testing framework. The article further contains an example load test which shows that conformance to a random arrivals traffic pattern can be checked by determining if request time mean and standard deviation are statistically equal, a property of the Negative-Exponential distribution. Figure 2 illustrates this relationship with output from a Perl script this author wrote which processes inter-arrival time stamps generated by a popular free load generator, JMeter.

Arrival Summary Statistics (ms)									
label	n	tps	median	mean	sdev	p95	p99	min	max
Home	81322	24.63	28	40.60	43.11	123	193	0	2229
by_department	20140	6.10	113	163.75	168.34	495	769	0	2233
by_department_xx	20341	6.17	111	162.17	167.75	491	778	0	3581
by_function	20287	6.15	111	162.54	167.32	491	757	0	2648
by_function_yyy	20534	6.23	110	160.55	166.59	476	766	0	4865
by_general_ledger	20423	6.19	109	161.60	169.88	489	772	0	2974
by_general_leger_zzzz	20153	6.11	112	163.69	171.95	490	765	0	5063
by_revenue	20083	6.08	111	164.39	171.65	498	782	0	3276
Total	223290	67.63	10	14.79	15.93	45	70	0	799

Figure 2: JMeter Test Run Arrival Statistics

Large Population Traffic Generation

These characteristics of large population end user environments facilitate load generation flexibility by shifting the focus from viewing virtual user threads as individual entities to being a set of random arrival transaction generators. This perspective allows a fixed number of threads, which conform to the CPU and Memory capacities of the traffic generator, to be used for testing a range of traffic levels by altering think times.

A key to successfully representing large population traffic in this manner is the proper selection of think time probability distribution from the load tool's option list. The obvious choice for the desired random arrivals traffic pattern is the

Negative – Exponential distribution [BRAD06] but surprisingly many load tools, including JMeter, do not support this delay timer option. Fortunately, the Poisson process can be produced anyway using the available random draw probability distributions, e.g., Uniform Random, as long as there are a sufficient number of test 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 Poisson process as their number increase [KARL75] [ALBI82]. The concept is analogous to the Normal Distribution's limiting properties when summing random variables [HOEL62].

Ramping Up Traffic

Viewing virtual users as transaction creators improves traffic generation scalability but what if the CPU and Memory capacity of a single traffic generating platform is insufficient to yield the needed load? Because the traffic being produced by the platform is a Poisson stream, it can be shown mathematically that the merger of multiple Poisson streams is a single Poisson stream with intensity equal to the sum of the individual stream intensities [GIFF78] [GUN05]. This stream merging property is illustrated in Figure 3 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.

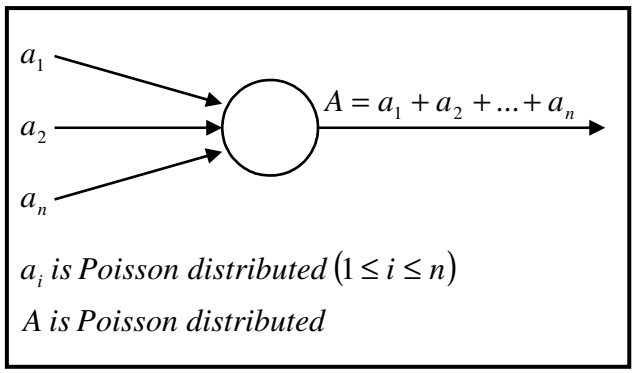


Figure 3: Combining Poisson Streams [BRAD07]

Active Users Supported

Under these fixed user thread conditions, the number of active users supported at a specific transaction rate can be estimated by performing the multiplication of the transaction rate produced by the mean cycle time (response time + think time). For example, assume the composite mean (weighted average) think time for the Figure 2 transaction events is 10 seconds and the overall response time during this test run is 4.78 seconds, then the number of active users supported is 1000, i.e., 67.64 x (4.78 + 10) = 1000. That is to say, 1000 users, each with a 14.78 second average transaction cycle time, collectively offer 67.64 transactions per second to the target environment.

Because think times vary and are educated guesses at best, a table like that shown in Figure 4 can be useful. This figure contains a matrix of active users supported as a function of multiple mean think times and load test transaction rates. The Trans/Sec levels for the six test runs were all produced with the same set of 325 user threads by adjusting think time settings. The 1000 user example above and the arrival statistics in Figure 2 are specific to Test Run 6.

				Mean Think Time (Sec)					
User Threads = 325				5	10	15	20	25	30
Test Run	CPU % Use	Trans/Sec	RT (Sec)	Active Users Supported					
0	0	0.00	0.00	0	0	0	0	0	0
1	22	15.54	0.32	83	160	238	316	393	471
2	37	30.33	0.35	162	314	466	617	769	921
3	52	46.06	0.45	251	481	712	942	1172	1403
4	63	58.74	1.05	355	649	943	1237	1530	1824
5	67	62.90	2.70	484	799	1113	1428	1742	2057
6	70	67.64	4.78	662	1000	1338	1676	2014	2353

Figure 4: Active Users Supported

Resource Scalability

What are the primary factors for determining resource scalability for the Figure 4 load test results? Since this figure

contains CPU % Use information, an X-Y plot of that statistic as a function of the appropriate independent variable can help determine processor scalability. Often the independent variable chosen for this purpose is active users. A graph of this functional relationship for the 10 second think time column is shown in Figure 5.

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 significantly less CPU time is required to process the incremental load from 800 to 1000 users than from 400 to 600 users. This “does as much work with far fewer resources” result is counter intuitive and is so because traffic sources (Active Users) do not logically scale with CPU resources but the traffic they produce does potentially scale.

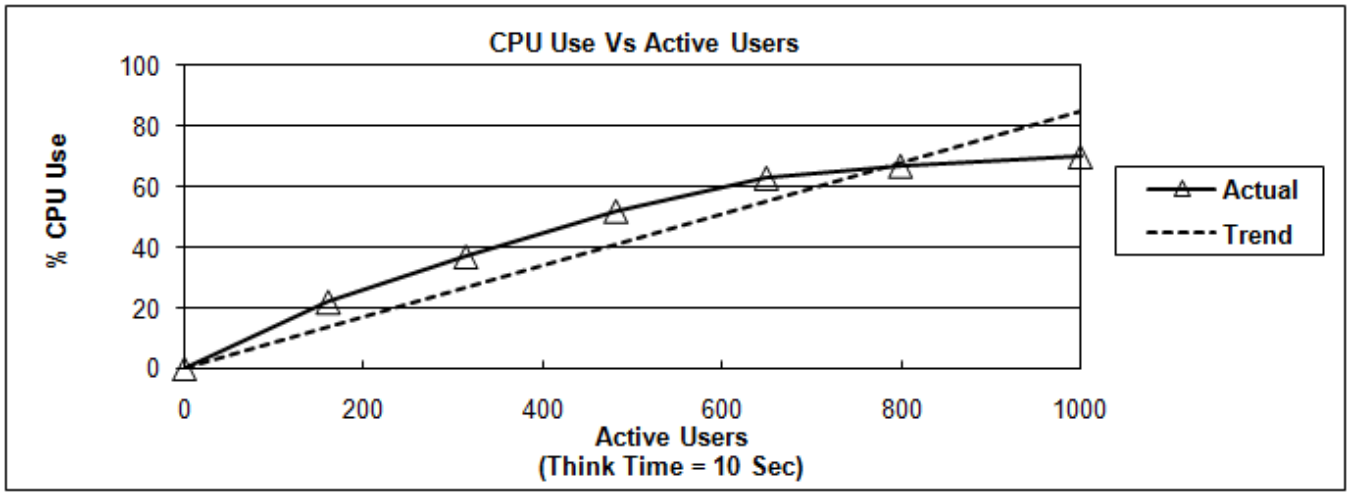


Figure 5: CPU Use as a Function of Active Users

In contrast, Figure 6 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 4. Since the “Trend” line in Figure 6 is nearly coincident with the “Actual” line connecting the six data points it is reasonable to assume CPU resources are scalable to at least 70% Use.

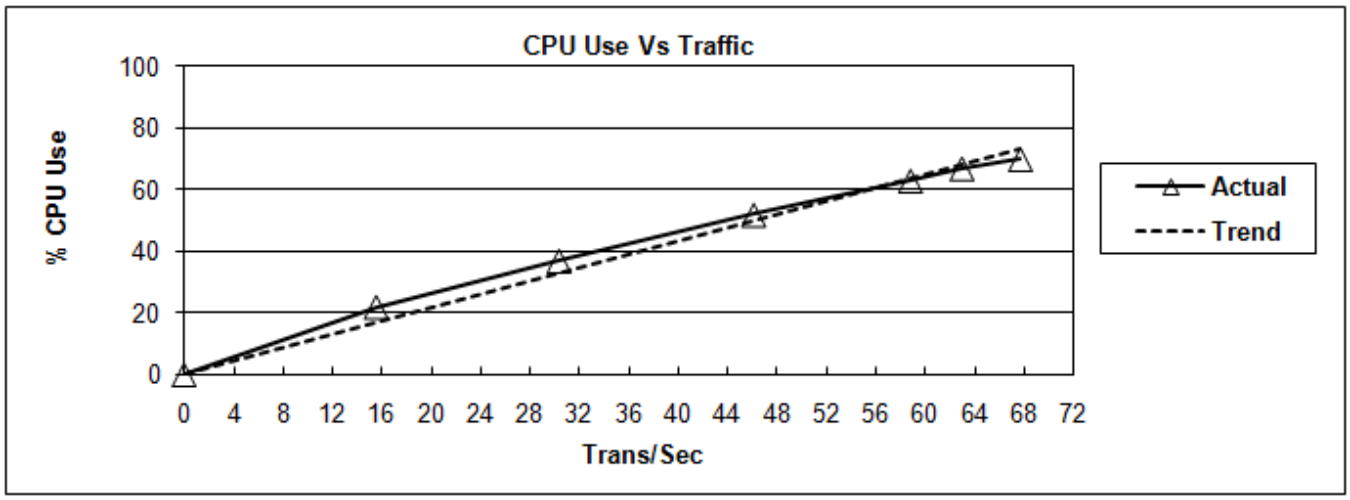


Figure 6: CPU Use as a Function of Traffic (Trans/Sec)

Why the difference between the two plots? The short answer is that active users are not traffic but generate the traffic the servers’ process. Looking more closely, the active user event cycle time includes response time in addition to think time and, as Figure 7, a plot of Figure 4 response times illustrates, these times are typically a non-linear function of load at higher Trans/Sec levels. Therefore, response time is a much greater proportion of user thread cycle time at higher traffic rates making each thread less efficient as a transaction delivery mechanism.

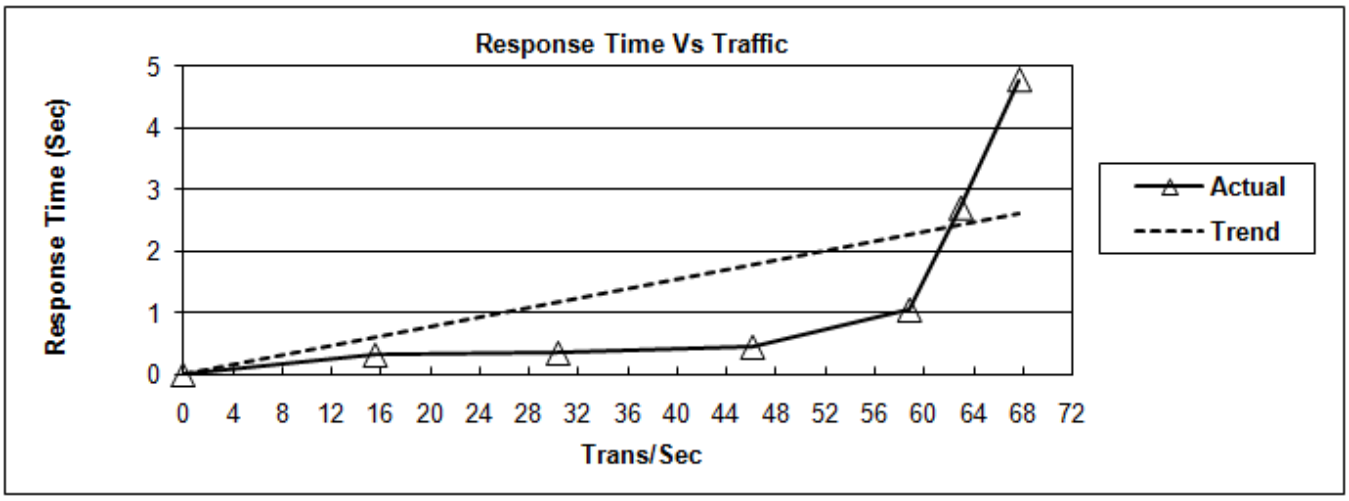
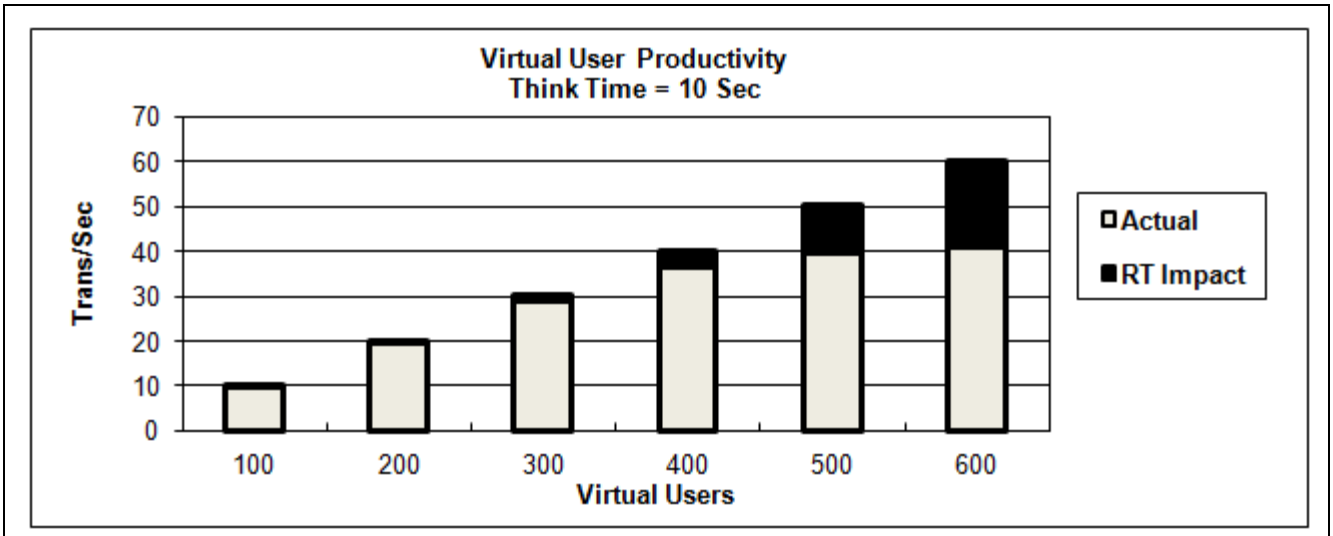


Figure 7: Response Time as a Function of Traffic (Trans/Sec)

This virtual user productivity argument is more clearly understood within the context of the traditional traffic ramp up technique described at the beginning of this paper and illustrated in Figure 8. This figure shows virtual user thread productivity when the virtual user count is increased incrementally for a 10 second think time and the response times in Figure 4. As shown, the 4.78 second response time in Run 6 versus the Run 1 value of .32 seconds increases the average cycle time from 10.32 seconds to 14.78 seconds, a 43% loss in efficiency. Without the impact of response time on cycle time Run 6 would offer 60 Trans/Sec to the target environment but it only delivers 40.6 Trans/Sec.



		Time (Sec)			Trans/Sec		
Test Run	Vusers	Think	Response	Cycle	Think	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 8: Virtual User Productivity

Summary

This article puts virtual users in their place by showing that load tests intended to represent a large user population can be performed effectively with fewer of them than the actual users they are intended to represent. An example is provided which illustrates how 325 virtual users can reflect 1000 active users of workload while maintaining a traffic

pattern consistent with real world conditions. This example and the resource scalability discussion that followed it lead to seven basic steps for the practitioner to follow.

1. Establish a fixed thread count for all tests within a scenario that satisfies maximum user population requirements and operates comfortably within the load generating computer(s) memory and processor capabilities.
2. If multiple load generators are required, as reflected in Figure 3, produce consistent loads across them using identical computers (if possible) and network connections with the same bandwidth.
3. Increase the traffic from one load run to the next by incrementally reducing traffic generator delay time values.
4. Use an appropriate think time distribution from the list of traffic generator delay timer options available. For large populations the Negative-Exponential is the appropriate choice but if this distribution is unavailable the best selection is usually the Uniform Random distribution.
5. Check traffic pattern quality with an inter-arrival distribution analysis tool as illustrated in Figure 2.
6. Calculate active users supported by creating a table like Figure 4.
7. Check resource scalability using X-Y plots of resource consumption as a function of aggregate transaction rate, e.g., Figure 6. For a resource to be considered scalable, the data should closely approximate a straight line over its range.

The technique these seven steps reflect shifts the emphasis away from traffic produced by individual users to that generated by all users. Traditional approaches that simply attempt to mimic the real world by creating a virtual thread per real user, while ignoring fundamental traffic principles, give the analyst a false sense of security that the load test being performed yields applicable results. After all, a load generating computer is one traffic source attempting to operate like a large number of independent sources but it will fail to accomplish that objective if the traffic pattern produced is not representative.

References

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

[ALLE78] A.O. Allen, “Probability, Statistics, And Queueing Theory”, Academic Press, Inc., Orlando, Florida, (1978).

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

[BRAD07] J. F. Brady, “Load Testing Virtualized Servers – Issues and Guidelines,” CMG *Journal*, 121:44-53, (Fall 2007).

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

[COOP84] R. B. Cooper, “Introduction to Queueing Theory”, Elsevier Science Publishing Co., Inc., New York, N.Y., (1984).

[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).

[HOEL62] P.G. Hoel, “Introduction to Mathematical Statistics”, John Wiley & Sons, New York, N.Y., (1962).

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

[WIKI11] D. R. Cox, “Poisson Process”, Wikipedia, http://en.wikipedia.org/wiki/Poisson_process. (July 2011).

Copyrights and Trademarks