Apica

*Achieve Peak Web Performance*

# A Guide to Using Load Testing and Cloud-based Capacity Planning

WHITE PAPER

ApicaSystems.com

Stockholm, London, New York, Santa Monica

## 1. Introduction

On September 12, 2017 Google, one of the few companies that seem to run the world, had a “world-wide meltdown”. The internet giant had been facing application uptime issues with multiple services including Gmail, Drive, Maps and YouTube. This potentially affected millions of users for several hours across parts of the UK, Europe, North America and Japan. The struggles of such a dominant provider should remind companies of all sizes that performance monitoring and testing remains a vital part of their infrastructure; no one is ‘too big to fail’. Application uptime matters more than ever, and if Google can go down, your business most definitely can, too.

It’s not just the really large sites that experience performance problems. An unexpected spike in traffic – from an article that mentions the company, or a marketing TV campaign for a “happy hour discount”, or a vacation deal for hotel reservations that succeeds too well – can bring your web application crashing to a halt.

Website failures and performance problems are costing companies lost revenues and damaged reputations. Even the biggest in the business are suffering. These high-profile website failures could have been avoided with some pre-launch load testing and capacity planning.

No matter how well your system is put together, you can’t really know how it will perform under stress until you test it. There are two technologies to help organizations prepare their websites for both a sudden spike in traffic and for long-term growth.

This white paper is designed to help you understand how you can use load testing to ensure your web application is ready for anything.

## Key Concept: the Load Curve

The most important measurements from a load test are the "load curves." These provide, at a single glance, an overview of the behavior of the target system under various loads. There are four basic load curves of special interest:

1. The average duration of a web surfing session per user. This curve represents the response time under load. Here, the server reaches capacity at 800 users.

2. The total number of successful URL calls, per second, measured over all users. This curve represents the capacity of the web application. This server reached the maximum capacity for handling URL calls at 800 users.

3. The percentage of failed web surf sessions, measured over all users. This curve represents the stability of the web application. Once this server reached 1000 users, it became unstable, with the number of failed sessions growing linearly.

4. The average response time, per web page. This curve represents the load times of various individual web pages under different loads, which can vary widely. Starting at 1000 simultaneous users, the response times for web pages 2, 3, and 4 rise dramatically.

## 2. Why Web Applications Fail

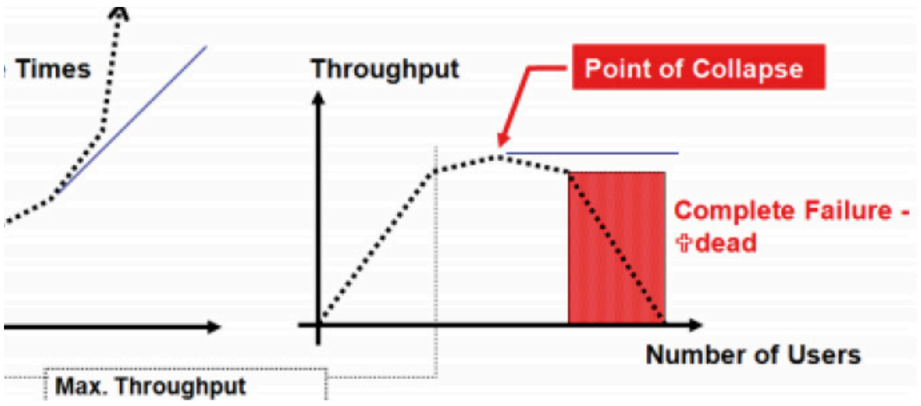
There are a variety of reasons a web application might fail, and not every web application fails in the exact same way. In the sidebar, Key Concept: the Load Curve, we see an application that has yet to actually fail—its response time is increasing in a linear fashion when the application's capacity is exceeded, but the system's total throughput remains constant.

Other web applications, however, will experience deteriorating throughput across the entire system when they are overloaded. This measurement, the point where throughput starts to decrease under escalating load, is the point of collapse.

If this point is surpassed, the response times will tend towards infinity. This means that the application is no longer available or, to use the vernacular, "dead." The fact that the system is dead will not be apparent to web visitors, who continue hoping that they will eventually receive a response from the web application.

They continue to click on the (unresponsive) web page hyperlinks or use the browser refresh button. The application will continue to be overloaded and unresponsive until the majority of users recognize that there is something wrong. At this point, users will begin to leave the system.

If new users continue to arrive—for example, with an electronic banking application or an Internet concert ticket shop—the web application can remain beyond the point of collapse for several hours.



**TIP:** The cause of the collapse can almost always be traced back to problems with internal parallel processes, indicating errors in the architecture, programming, or configuration. Examples of these kinds of errors are: extensive synchronization of program code, un-optimized database update operations, and a lack of operating system resources.

## 3.1 Choosing Test Cases

The first and perhaps most important step is to choose the correct test cases. The sponsor of a web application often specifies fundamental requirements which must be satisfied; for example:

1. The web application should be able to support 2,000 concurrent users using the system

2. The response time for a web page in the scenario should not be more than three seconds on average, and never more than five seconds.

3. The user should be able to complete the full session with good performance, i.e. scenario: buy from an e-shop, find a flight, complete a stock transaction, find an article.

To test whether the web application meets these requirements, we first need to define which web pages, or sequence of web pages (scenario) that should be included in the test. Also needed is an idea of how long a single (actual) user will look at a web page before clicking on the next hyperlink—the user's "think time."

If the web application is already in production use, some of the missing requirements can be gleaned from an analysis of the application, web server, or log files. For example, the 15 web pages called most often can be identified from the access log files, and then a load test can be defined to use exactly those pages.

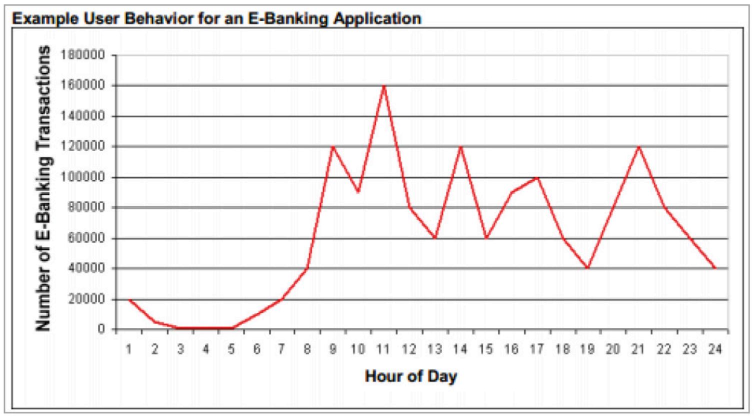
In some web applications, key menu options will not be among the web pages called most often—as, for example, is the case of an online shop. With these applications, users often browse for an extended time in the product catalogue before finally ordering and paying.

There are other web pages, such as the order and payment web pages, where good response time is critical, no matter how infrequently they are used. A slow order or payment web page means the company is leaving money on the table.

In such cases, it is often useful to execute two simultaneous load tests instead of just one. As an example, one load test, with a large number of simultaneous users, can be executed to create a basic "background" load, and a second test—run at the same time as the first—can be executed with fewer users to cover the most important special cases.

If you have only general summary information on how the web application is used, or no information at all, you must fill in the missing data by extrapolation or estimation in order to arrive at a reasonable approximation of actual user behavior.

For example, if the only known fact for an electronic banking application is that it processes 1.5 million financial transactions per day, you must take into consideration that these transactions will likely not occur evenly distributed over the entire day.



If only the average value for the desired load is known, this average load value must be multiplied by some factor for estimating peak-time activity. From experience with electronic banking applications, a realistic peak load factor multiplier is somewhere between 3 and 12, with 5 being the most common value used.

Continuing with the electronic banking example, knowing only the number of financial transactions is not enough to arrive at how many users will be required for the load test. However, by evaluating the load curves produced from load tests, you can discover the web application's maximum capacity, and thereby also the maximum number of financial transactions it can support.

Because load curves highlight the direct relationship between throughput and response times, you will then also know the web application's expected response times under light, medium, and heavy loads.

## 3.2 Performing Load Tests—the top 8 points to keep in mind

A structured, almost scientific approach to planning and executing a load test will provide the hard data needed to make informed decisions about tuning a web application and planning for both future growth and unexpected surges in usage.

Scientists test their hypotheses using carefully controlled and fully reproducible experiments—if other scientists cannot understand the experiment or reproduce the results, then the research is considered invalid. Likewise, a good load test needs to be carefully controlled, and conducted in such a manner that repeating a test using the same initial variables will yield the same results.

**Here are eight points to keep in mind when conducting a successful load test:**

1. Consistency is essential. Load curves are produced by repeating the same exact load test with varying numbers of users; for example, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 10,000, and so on. It is recommended that the load be increased step-by-step, increasing the number of users geometrically or logarithmically. This is the fastest way to gain an overview of performance. Notice also the phrase “repeating the same exact load test.” It is essential that the load test be exactly same, with the exception of the number of users, so that the results are comparable.

2. Do it long enough. Do not run load tests with too short a duration. As a rule of thumb, divide the number of users in your load test by 10 to arrive at the minimum test duration in minutes. If you are testing with 50 users, the test should last at least 5 minutes. With the very large number of users involved in a mega load test, start the test with a smaller number of users and then gradually “ramp up” to the target number of users before starting the test duration “clock.”

3. Anticipate failure. Sometimes, for example, a web application stops working during a load test and does not recover by itself after the end of the test. Make sure that someone is available, during load testing, to restart the web application so you can continue with testing after an application failure.

4. Keep an eye on the load simulator. A load test is not valid if the load-releasing system—that is, the system creating the load—is itself overloaded. In this case the measured response times will not be valid. Always watch the CPU activity on the load-releasing system during the test for signs of overloading. If the load-releasing system does not have enough CPU power for the load test, additional systems can be added, and combined, to form a load-injector cluster whereby the load test program is distributed across many computer systems.

5. Every simulated user should have its own login, if needed. If the web application is a secure one, requiring authentication, each load test user should have its own login account during the test; otherwise, the test will not be "fair" to the person or company delivering the application. If many users share a single login account during the test, errors may occur, and be measured, which would not occur in real life.

6. Monitor the CPU activity. During the load test, arrange for someone to observe the CPU activity on the web or application server machine on which the web application is running. This information will be needed later to see if the server's CPU is 100% used at the time the maximum throughput (or performance limit) of the web application is reached. If the server's CPU usage at maximum load is below 70%, this indicates that the web application is not able to fully use server resources. This presents you an opportunity for performance tuning.

7. Eliminate any potential network bottlenecks beforehand. Ensure that the network capacity between the load-releasing system and the web application server is sufficient; that is, at least 100 Mbit/second. You are testing the web application, not the network.

8. Have the right people involved in the test. Ideally the system administrator, the database administrator, and a web application developer will be present during the load tests in order that you can analyze the load test results together and immediately decide on and implement any necessary performance tuning measures. This test, analyze, and tune process can be repeated until the web application is performing satisfactorily. Having all involved parties present and immediately available will save a huge amount of time and help bring you quickly to your goal.

## 4. Tuning Tips

Once you have run your load test, you can begin the process of tuning and optimizing your web application. The first step is always to try to establish the reason for poor response times or stability problems, based on an analysis of the measurement data. Optimize your web application infrastructure solely on the basis of facts ascertained by this kind of analysis.

### 4.1 Optimizing Response Times

When the results of a load test show unacceptable response times, it is often assumed that the "network" is the cause of the problem. In reality, this is seldom the case. Fortunately, it is quite easy to check this using test measurement data. Simply compare the response time for a static image (such as JPEG, GIF, or PNG), with the response time for a dynamically created HTML page of approximately the same size. Consider this example:

• The response time for a static GIF image of 20,000 bytes is measured at 60 milliseconds.

• The response time for a dynamically created HTML page with a size of 40,000 bytes is measured at 3 seconds.

Assuming the response time of the static image depends solely on the network bandwidth (which is not really true, but is useful for our purposes in this example), the dynamically created HTML page should have a response time of 120 milliseconds since it is "only" twice the size of the image. Since the actual response time of the HTML page is 3 seconds, the web application uses at least 2880 milliseconds (3000—120) to produce and send the page.

If your network is functioning correctly, here are some of the steps you can take to optimize the response time of your web application.

• Look to the images. If, under increasing load, the response times for both static images and dynamically-created HTML pages rise in tandem, there may indeed be a network problem or, more likely, an error in the configuration of the web or application server hosting the web application. Examples of configuration errors would be not enough worker threads or too few worker processes.

• Look to the URL calls. If, under increasing load, some URL calls become disproportionately slower than others in relation to the rise in load, the error will most likely be in the programming of the web application, including perhaps inefficient database queries.

1. Check the database connection pool. If the web or application server uses a database connection pool, check first if this pool is big enough. For a single web page access, each user will usually need one connection out of the pool, unless the web application itself has implemented some form of caching.

2. Check the database engine itself. Arrange with the database administrator (DBA) to collect statistics on the slowest SQL queries and their respective response times or use APM tools to gather this statistic.

3. Check the application programming. If these first two investigations do not uncover the cause of the problem(s), the cause will be application programming errors. The mistake made most often is inappropriate use of code synchronization in inner loops, resulting in code blocks needed by all users being executed sequentially instead of in parallel. A dead giveaway for this situation is when the CPU of the web or application server is only lightly used under heavy load, because waiting for synchronized code to be released does not use CPU time.

## 4.2 Solving Stability Problems

Analyze the most frequent errors, and try to decide if the cause is inside the web application itself, or in the surrounding environment.

• If network errors or timeouts occur relatively evenly over all URL calls, the problem will not be with the web application. In this case, try first to tune the TCP/IP stack of the web or application server machine using operating system network and system parameters. Other possible sources of these errors are firewalls, load balancers, and reverse proxy servers. Perform tests from various different network locations, proceeding always from inner areas to outer areas. For example, test first directly and as close as possible to the web or application server—from the same LAN—and then move progressively outwards to test indirect access via reverse proxies and/or load balancers.

• If stability problems occur only with specific URL calls, then the problem lies in the web application itself. This also applies if HTML pages are received which contain error messages, or partially incorrect or unexpected content.

• To solve stability problems with specific URLs, you will need the support of the web application developers. Describe the problems to them, and motivate them to analyze their web application source code with a view to understanding the observed behavior. To help locate the source of the problems, application code changes will be necessary in order to insert log messages and/or checkpoints, with timestamps. This process is sometimes known as "white box testing." Internal checkpoints are first inserted at a high level in the application code, and then iteratively refined and moved to lower level suspect code areas as the tests zero-in on the cause. This process reduces the number of test iterations required to identify the source of the problems. "White box tests" can, of course, also be used in optimizing response times.

## 4.3 Preventing Application Overload

First, try to tune the web application to the point where it can provide acceptable response times under a load which is higher than that expected in actual production. If this is achieved, the overload point of the application would never be reached in actual use.

If this is not possible, the alternative is to insert a front end cache, like Varnish for example, in front of the web or application server. Varnish will act as a buffer, caching the static content such that the overload point would never be reached.

CDN vendors will do the same trick but for a fee, but they will also reduce the latency from different remote locations, if you have users distributed globally.

Dynamic database transaction data however cannot be cached. For those cases you need to implement a URL-queuing system to prevent overload and/or design the database in a fully scalable way. Limiting the number of direct calls is also a good idea.

## 4.4 Using the Cloud for Configuration Planning

The dynamics of cloud computing allow you to gauge your system's limits through capacity planning in a way that was never possible before.

Now, you can go beyond a simple stress test and try different configurations to determine how to gain the best performance from your application. Traditional servers, and even traditional hosting services, hindered capacity testing and planning with exorbitant fees and difficult-to-add server resources.

Once upon a time, you had to have a physical box to run your applications. It was impossible to do any serious scalability testing because of how difficult it was to bring in another server to see what the effects of more resources might be. Web hosting took us a step forward by eliminating the need to have a physical box, but it still lacked the flexibility needed to perform "full-out" capacity planning and testing.

So how are you able to perform this "full-out" capacity planning in the cloud? By using the flexibility of the cloud environment to manipulate your tests. For example, let's say that use of your application is growing at a rate of 10 percent per month. At the end of two years, you will have nine times the number of visitors. Can your application scale to that extent?

To find out, you can set up a temporary test configuration in the cloud and stress it with some simulated peak traffic to see how it performs. You can double the number of web servers, or you can use larger and more powerful web servers, and see what difference it makes in performance.

Being able to manipulate your environment during testing is the key to identifying any bottlenecks or other issues. In fact, our experiences with capacity planning in the cloud prove that many applications don't scale, no matter how many web servers are added to handle the workload. So, your test might find the bottleneck in the database or some other aspect of the application.

When you're doing capacity planning in the cloud, you can also mix and match the resources you use in your test. You can choose a small, medium, large, or even an extra large server; you can add memory; and you can add CPU. By testing every different kind of combination, you can see which server base and which configuration best suits your application.

The flexible nature of the cloud also allows you to create a contingency for any potential high-traffic scenario that you might face in the future. Because of the different combinations of testing you've tried, you'll know when to deploy which plan of action, whether that means adding another web server or more memory to the database server.

## 5. Other Tests to Help Prepare for the Worst

Load curve measurements are involved in all types of testing commonly known as "load tests" and "stress tests." There are other kinds of testing that can produce additional useful information to help avoid high-profile failure of a web application.

### 5.1 Duration or Endurance Testing

An endurance test provides data on the long-term stability of the web application; that is, it answers the question of whether the application is stable when running for many days. Normally, such a test is started in the evening and left to run overnight (for 12 hours) at 50 percent of the application's maximum capacity.

An endurance test usually makes many more URL calls than is normally seen in actual production for a single day. This acts as a sort of "time-lapse photography," in that many days of "production activity" can be simulated in a single night of testing.

### 5.2 Starting Under Load

When the web application is brand new, and has never been in production before, plans should be made for this type of test. Leave the web application turned off, and start a load test which would establish a load on the application at near 100% capacity. Naturally, you will receive only error measurements, because the web application is not yet online.

Now, start the web application during the on-going load test and observe the test results in real-time. The web application must be able to stabilize by itself under these conditions within a few minutes. The idea behind this kind of test is that a new web application will become usable on the network before it has finished its internal initialization, and the first URL calls could cause it to immediately fail. This should not happen, and a "starting under load" test will allow you to check this.

## 5.3 Degradation Testing

If the web application is running in an environment equipped with components—such as a load balancer or a web application cluster—that provide redundancy and high-availability, perform a few degradation tests whereby the high-availability components are individually turned off and then turned back on.

For example, stop a branch of the load balancer or turn off a computer in the web application cluster. This way, the effect of partial system failures in the application environment on the availability, performance, and stability of the web application can be tested.

## 5.4 Short-Circuit Testing

A "short-circuit test" is a type of test where the performance of the infrastructure is measured without the web application running. A few large, static image files are put on the web or application server, and the load curves are produced using only these images. This way, the maximum throughput of the system in the absence of the overhead associated with producing dynamic HTML pages can be determined. In addition, this test will indicate if there is a performance or stability problem even before the web application is running.

Another obvious test of this type is to measure the performance of direct accesses to the web application, and then to compare these results to measurements made while accessing the application via a reverse proxy. This test measures the performance loss associated with using a reverse proxy server.

## 5.5 Stripped Test

In a "stripped test", all URL calls which produce static content are removed from the load test configuration; for example, removing static images, style sheets, and JavaScript files. Only pure HTML content produced by the web application is involved in the test.

This kind of test is especially useful for developers who need to optimize specific dynamically generated web pages. Removing unnecessary—from the point of view of the test—static content such as images reduces the network bandwidth required and lightens the CPU load on the load-releasing systems, enabling a larger number of simultaneous users than would otherwise be possible on systems with limited resources. This also reduces the number of measurements to be analyzed, allowing a clearer picture of the situation.

## 6. Conclusion

Website failures and performance problems are costing companies lost revenues and damaged reputations. A high-profile website failure can be avoided with load testing and capacity planning. No matter how well your system is put together, you can't really know how it will perform under stress until you test it.

Two technologies—load testing and cloud computing—are now in place to help organizations prepare their websites for both a sudden spike in traffic and for long-term growth.

With the cloud and load testing, you can go beyond a simple stress test and try different configurations to determine how to gain the best performance from your application. Not only will you avoid becoming yet another high-profile website failure, you will have a fully tested contingency plan for any potential high-traffic scenario that you might face in the future.