How to Ensure Your Website or Mobile App Won't Fail on PEAK DAYS

Top Six Performance Testing Mistakes and Their Solutions

Andrey Pokhilko
Chief Scientist

Refael Botbol
Director of Professional Services

BlazeMeter

Table of Contents

INTRODUCTION - HOLIDAY FAILS:
WHEN YOUR PERFORMANCE TESTING LETS YOU DOWN

Holidays like Black Friday, Cyber Monday, Valentine's Day and Mother's Day, present developers and testers of e-commerce sites with an immense challenge.

These are almost always the busiest days of the year as website visitors rush to buy online gifts, goods, and cards, but there's a strong likelihood that the extremely high load of traffic will cause your web app or mobile app to
fail - and there couldn't be a worst time for this to happen.

The consequences of website fails on well-known international holidays like Black Friday or Cyber Monday can significantly damage profits and reputations.

Stores can literally lose hundreds of thousands of dollars from missed sales, shoppers are frustrated, customer loyalties are frayed, and it's incredibly stressful and disappointing for the developers in charge of the site. Stores behind the biggest holiday web fails make national headlines - for all the wrong reasons.

The list of catastrophic holiday failures is extensive - with store giants like Kohl's, Lowe's, Staples, Toys"R"Us, Sears, Home Depot, and Victoria's Secret all falling victim to web crashes at some point in their history.
Kohls is just one example of the high cost of a web failure on such a critical day. On Black Friday 2012, the store ran a huge sales promotion but, due to a massive surge in traffic, it experienced an **outage for several hours.**
Every time customers tried to view an item's details or access their shopping cart, they would simply view an outage message instead. This cost Kohls tens or hundreds of thousands of dollars in lost sales and endless frustrations for its customers[1].

So sorry!
Our team of elves is working hard to keep up with our holiday shoppers, but Kohls.com is not avaliable at the moment. We're working to get the site back up and running smoothly for you. Please check back shortly to shop our great holiday deals!

It's clear that rigorous preparations and robust performance testing is more crucial than ever in the lead up to known times of peak traffic.

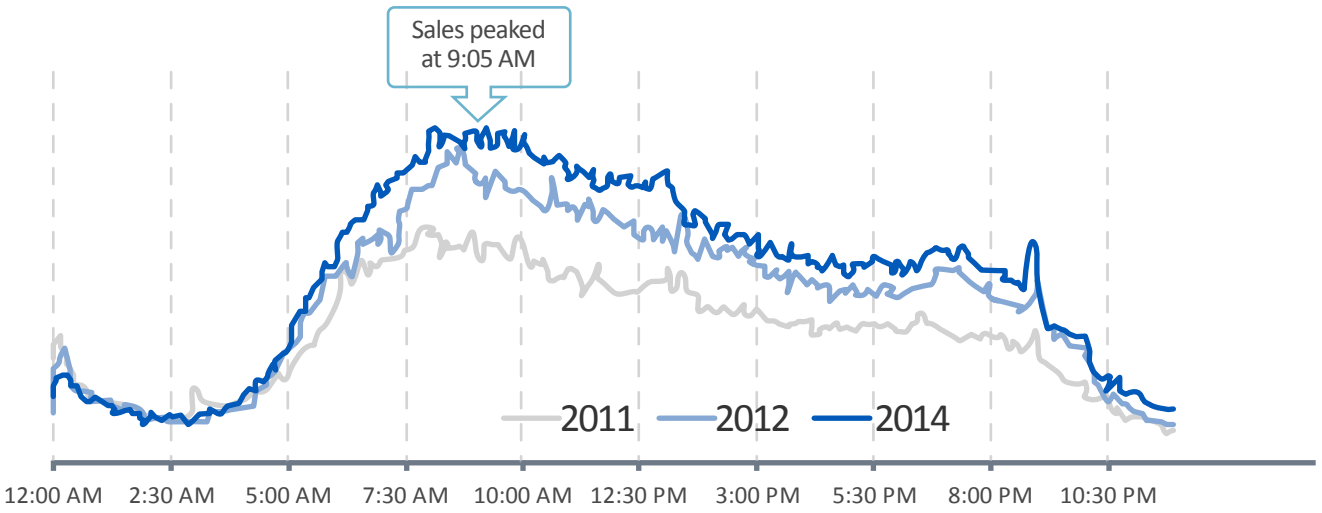
The good news is that most of these failures are preventable and 75% of bottlenecks can be avoided in testing.[2]

The bad news is that the stakes are getting higher. Online and mobile sales are rising and this trend looks set to continue. This is clear to see when analyzing statistics from sales over the past few Black Friday's in 2013, Black Friday desktop online sales were $1.198 billion - 15% higher than the previous year.3 Reports also reveal that traffic and sales from tablets and mobiles are rapidly rising. IBM's Digital Analytics Benchmark Study revealed that mobile traffic grew to 39.7 percent of all online traffic in 2013 - a 34% increase from 2012.[4]

Black Friday 2013

24-hr Realtime Sales Chart

IBM Digital Ananlytics Benchmark



Source: IBM 2013 Digital Benchmarks Report[5]

In this white paper, I will reveal how you can ensure your web or app will be a success during peak holiday traffic despite its perpetually increasing challenges. I will explore the six most commonly made performance testing mistakes - and reveal steps you can take to prevent them.

PART 1:
NOT INCLUDING YOUR NETWORK INFRASTRUCTURE IN PRE-EVENT TESTING

Failing to include your network infrastructure in your performance testing for the holidays puts your web or app at risk. In many cases, developers will test all the servers and infrastructures from inside the organization but not from outside. But testing solely in-house is inadequate. When you take this approach, you're failing to test and monitor all the chains of delivery and it's unlikely that you'll get a clear and accurate picture of how it will perform on the actual day.

Many failures aren't caused by the application itself. Maybe your application has a huge capacity - but how do you know that there isn't a problem with your external infrastructure or your hosting server (i.e. AWS or Rackspace)?

For example: Tumblr experienced an outage for six hours in October 2013 due to network issues caused by an issue with one of its uplink providers. The microblogging and social networking site was down from 8:30am to 2:25pm EST.6

Another mistake developers commonly make is when they run load tests without using a network emulator. It's important to see every scenario and check the results from different devices and global locations. As we've already seen, the amount of traffic on smartphones and tablets will differ dramatically to laptops and desktops. Regional differences are also a significant factor. As this table shows, you're much more likely to get a heavy load of traffic from New York and California than states like Ohio and Michigan. Therefore, it's important to take such variables into account by including them into the testing.

Top States Total Online Retail Sales — IBM Digital Ananlytics Benchmark

Thanksgiving 2013	Black Friday 2013
1. California	1. New York
2. New York	2. California
3. Texas	3. Texas
4. Florida	4. Florida
5. Illinois	5. Georgia
6. Pennsylvania	6. New Jersey
7. New Jersey	7. Pennsylvania
8. Georgia	8. Illinois
9. Ohio	9. Ohio
10. Michigan	10. Michigan

Source: IBM 2013 Digital Benchmarks Report[5]

If you ignore or overlook testing these variables in your regular tests - you might just miss the actual problem that will hit you during peak times.

SOLUTION 1:
RUN LOAD TESTS FROM THE PRODUCTION ENVIRONMENT

First of all, I strongly recommend running the test on your live production site. This is the only way that you can ensure the test will be accurate, that the test plan is well organized, and that you're stressing every point in the entire chain of delivery. It's best practice to test in your production environment at a time when you know that traffic will be low (for example: 2:00am on a Sunday morning) - and to notify your customers in advance that there is a possibility of downtime during this period. It's far better to create a 'handmade disaster' during off-peak hours than to encounter a real disaster during the busiest day of the year. Trust me - it will be cheaper in the long run!

Now that you've decided to test in your production environment, there are various ways that you can do this. Some companies take their existing processes and enlist real people sitting at physical machines and devices from around the world to test their web or app. This is still done by some organizations today but it's clearly not the most efficient!

Another, more efficient, way is to use an open source load testing tool like **JMeter** and buy several Virtual Private Servers (VPS) in different geo-locations to test your web or app servers under heavy, concurrent and geographically distributed load. You can also take a cloud performance testing tool like BlazeMeter to simulate the load from multiple geo-locations and various devices with just a few clicks in the User Interface (UI).

PART 2:
UNDERESTIMATING THE ANTICIPATED LOAD

This mistake is very easy to make. Often developers will use simple maths to estimate how many visitors are expected to come the following year. But, in the unpredictable world of e-commerce, 2+2 doesn't always equal 4.

For example: If your web or app sustained a load of 100,000 on Cyber Monday in 2014 and your business has doubled in the past year, you could probably expect around 200,000 (100,000 visitors X 2 times the # of customers = 200,000) visitors this year - right? Right? Wrong. Unlike mathematical formulas, people are unpredictable. Maybe your business has a very special promotion for Cyber Monday in 2015 which means that 3X the number of visitors will come. Maybe the physical store was overwhelming the previous year so far more people decided to shop online this year.

Another, slightly tricky situation can occur, whereby your competitor's website might fail and his customers come flocking to you. This will generate an unprecedented load on your web and, if you also fail, these people will move on to another site, creating a snowball effect. You want to be able to collect all of this culminated traffic - not let it slip through and move on to the competition.

So maybe you've tested your web or app up to 200,000 users and the test went well. You think it will sustain the load - but what happens when visitor #200,001 arrives?

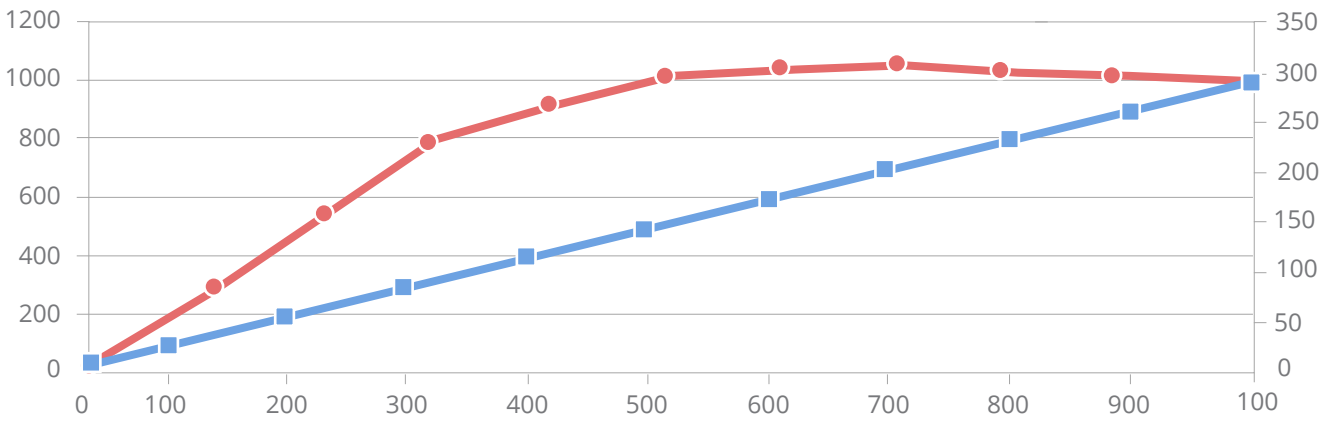
SOLUTION 2:
GO TO THE LIMIT. TRY TO BRING YOUR SYSTEM INTO THE FAILURE.

Whatever your failure is - whether it's a CPU, memory, connection pools, network bandwidth issue or something else - you want to find it. Don't be satisfied with a successful test. You need to know your capacity. This means that you should keep on increasing the load until you see **what** will fail, **when** it will fail, and **how** it will fail.

To do this, run a sequence of tests while continually increasing the load. Monitor the hits/s throughput as you increase your load. Keep on doing this until you hit a scaling problem. Don't rely on theoretical formulas to calculate the limits of your web or app, find the actual limit through testing.

For example: In this chart, you can see where your system reaches its capacity. In this example, the application is unable to increase the hits per second rate after 300 virtual users. This reveals your saturation point and you can show this report to your stakeholders, confident that you're giving them a trustworthy measurement.

SCALABILITY CHART



PART 3:
FAILING TO IDENTIFY YOUR CRITICAL POINTS

So, as we can see from mistake #2, it's vital to identify exactly when your system will break. However, this is only half of the picture. Once you've identified when you will hit a bottleneck, you then need to pinpoint precisely where the underlying issue is.

If you don't know what's causing the problems, you can't resolve them and issues could still arise from this critical point at the worst possible moment.

SOLUTION 3A:
FIND THE CRITICAL RESOURCE

Now is the time to investigate what's going on in your system. Are the transactions taking too long? Are there critical messages in the log? Are there hardware resources that are exhausted?

When preparing for peak traffic times, your operations team sets up resource monitoring dashboards and alerts to keep track of your systems. But there are literally thousands of metrics that can be measured. How many operations teams have the time or resources to monitor all of these?

If your Ops teams' workspace looks like this, it will be hard for them to identify the problems quickly.

Here's where performance engineers can help by enabling them to focus on the critical issues. (APM tools like New Relic can help as they monitor the backend application and enable you to understand where your bottlenecks are occurring). During performance testing, identify around **five to seven** metrics which reflect the system's critical usage and give them to the operations team for tracking. You can also set up automated alerts with pre-defined thresholds on around 20-50 metrics for resources that could lead to further problems. If you have time, you can even get your developers to optimize some of the critical resources that you've identified - eliminating problems before they occur.

Why is this important? Let's say that a last minute change is made to the application which might affect the performance of your web or app. At least you'll know which resources to keep a close eye on and, if you see that you're getting close to the limit, you can quickly take action.

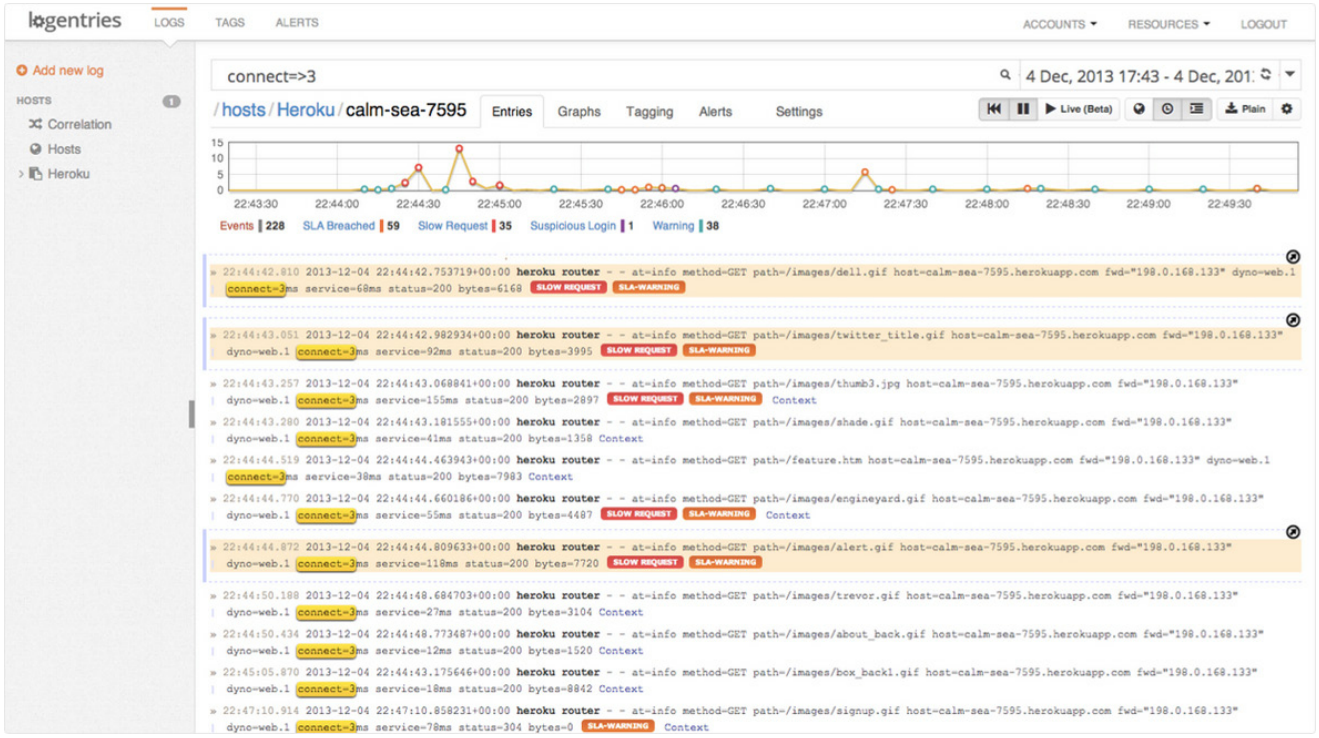
Unfortunately, issue identification can be quite tricky as the problematic resource might not be at all obvious. A good example of this is when you study the time spent on the application lock. Quite often the application will lock due to intensive writes into the database table - not because the system resources are at full capacity, but because of data consistency requirements. Hopefully, most database servers are able to track the locking time. But this is by no means guaranteed, so take some time to verify it.

Finding all the critical resources manually just isn't a viable option. And this is why I recommend using monitoring or profiling tools. They will pinpoint your critical resources and identify all the weak links in the chain when you reach your limit. Find an appropriate tool for your platform. For modern Linux systems, this is the perf tool. The Windows platform has a lot of free and commercial solutions, you can choose one that suits your budget.

Observe Your Application Logs.

Your application logs can contain important information that you simply can't see in the load test results.

To ensure that you get a full picture of **when** the problems occur and **why**, it's worth using Application Performance Management (APM) tools like BlazeMeter along with Log Aggregation Tools like Logentries and Splunk. These log aggregation tools can read your logs and create reports for you - making your life a lot easier.



A Logentries Screenshot Displaying Tags and Alerts of Important Events

SOLUTION 3B:
DIVIDE YOUR SYSTEM - THEN STRESS EACH SECTION INDIVIDUALLY

Web traffic can be unpredictable; the load isn't going to be evenly distributed across the site and it might not come in the pattern you're expecting.

For example: let's say your company is selling an incredibly popular item at an incredibly competitive price. There's a good chance that the traffic to this product page will be 10 or even 100 times higher than other pages on your site. If an unprecedented load of traffic comes to this particular page, it could crash the entire system.

To avoid such an event, it's good practice to divide your system into logical sections - and then stress each one separately. Every single section should be stressed to the limit (as outlined in #2 of this document) - then the underlying critical resource must be found. This will enable you to identify and fix all the problematic scenarios in every section of your system.

Is it time-consuming? Maybe.
Is it vital? Absolutely.

This might sound like a lot of work but it's an important and valuable process. You may have a critical underlying problem that won't be revealed when you run a general test of the system. When you run a general test, the first bottleneck that you hit could be hiding other issues.

By stressing every section of your web or app, you can identify every possible problem you might run into - and deal with them accordingly. Let's say you have a live chat window on your website. The number of people using the chat window won't be equal to the number of people viewing the homepage. What's more; the capacity of each object won't be the same.

By identifying the capacity of each item separately, you can make sure the 'weakest link' won't break the entire chain. For example: if you see that your chat window only has the capacity for 100 users per minute, you can keep it separate and ensure that it won't crash the entire site. You can take proactive measures to minimise frustration from your customers, such as setting up a message notifying users that the chat operatives are busy and that they are next in line.

PART 4:
FAILING TO RECOVER QUICKLY FROM TECHNICAL PROBLEMS

Power outages, technical problems and crashes happen. The key question is: how do you deal with them?
In many cases, the power outage won't last for more than a couple of minutes - but you may spend hours recovering from it (for example: you may have to load broken data from the cold backup storage or wait for the database server to perform a consistency check on the database).

Another example: Salesforce recently suffered from a power failure at an Equinix data center in Silicon Valley. The power outage itself only lasted for one minute. However, it took Salesforce more than nine hours to get their service fully up and running again.5 Now imagine if this happened to you on a busy holiday day. A power outage of 60 seconds could ruin your entire sales day.

SOLUTION 4:
SET UP BACK-UP SERVERS AND LOCATIONS

Don't take any chances - have back-up servers and locations ready so you can recover quickly if you are unfortunate enough to experience a power outage during a time of peak traffic.

Set up a database replication, database failover cluster or application failover cluster. As soon as there's a problem, you can just switch over to the failover location as quickly as possible. This means that you don't have to wait for your main server to recover, your back-up can be running while you firefight and resolve the critical issues. It's very easy to do this with modern technologies like **Redis**, **mongoDB**, and **Cassandra** offering various data redundancy schemes for automated failovers - and is well worth considering for developers and engineers of online shops.

You can even switch to your failover location manually. You just need to prepare a procedure in advance and make sure your Ops/DevOps team members know exactly what to do if a problem does occur.

PART 5:
OVERLOOKING THE END USER PERFORMANCE

Tracking the server performance alone isn't enough. You also need to monitor the performance from the perspective of the end user. Your server times might not appear to be degrading but it's possible that there are still problems in part of your application - and you can't predict how they will affect the rendering of your web page. Problematic requests might make your page look very slow on the browser but they don't always trigger a significant raise in response times during a load test.

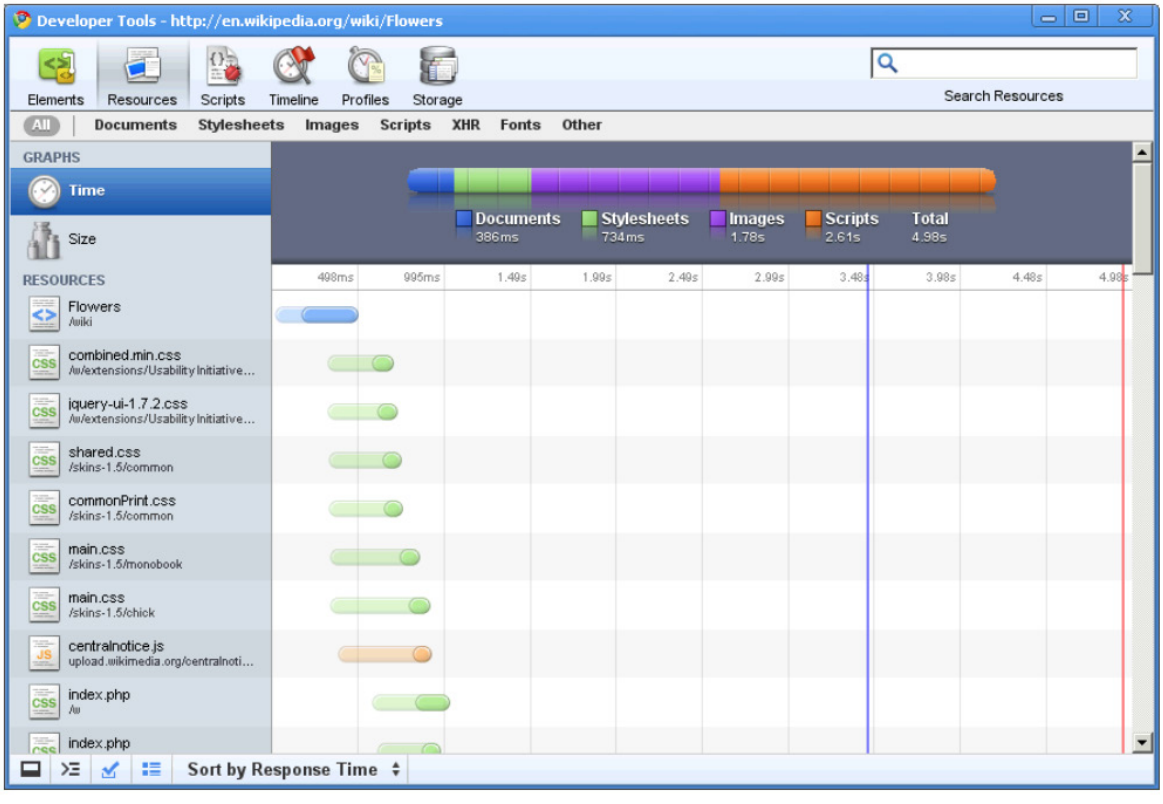
You also need to verify that the full page loads within a reasonable time - not just the first piece of text. After all, your users aren't interested in seeing half or a third of your web page or app. The slow loading image or piece of text might just be the area that they want to see.

SOLUTION 5:
TRACK THE END USER PERFORMANCE

Incorporate end user performance tracking into your backend testing. This is easy to do and there are plenty of tools on the market to help you.

You can analyze your web page with **Firebug** or HTTP waterfall charts from **Chrome's Developer Tools**. Waterfall charts show you what's going on behind the scenes when a browser loads your app or webpage. You can view every piece of content that is being loaded - from CSS files and javascript to third party banners and images. More crucially, they show you how long each content piece takes to load and are therefore very valuable for identifying bottlenecks in end-user performance.

A Screenshot of Chrome's HTTP Waterfall Charts



You can incorporate this into your performance testing by taking your test plan, stressing the server, opening your Firebug or Developer Tools and manually checking how long your pages take to load while being stressed.

This is the manual way of tracking your end user performance as part of your load test. However, you can also use a tool that does it automatically for you. For example: you can compose a JMeter test plan with 100 threads performing your backend testing and one thread for the WebDriver Sampler. The **WebDriver Sampler** will automate the execution and collection of your browser's performance metrics by mimicking the behavior of a real user and interacting with the HTML of the application.

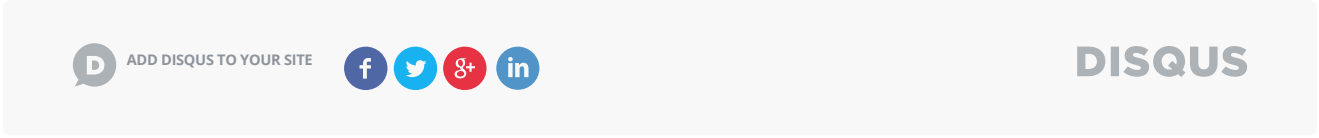
When using the WebDriver with JMeter, you'll need to write the Selenium code to load the page and ensure you're measuring the rendering time correctly. Other tools like **BlazeMeter** have features which do all the tracking and information collection for you automatically. With BlazeMeter, you can create a heavy load on your application and integrate with tools like WebDriver, **Perfecto Mobile** and **Sauce Labs** to create Selenium tests from various regions across the globe. This is easy to do and it will give you a very accurate picture of the actual user experience.

PART 6:
FAILING TO CONSIDER YOUR THIRD PARTY INTEGRATIONS

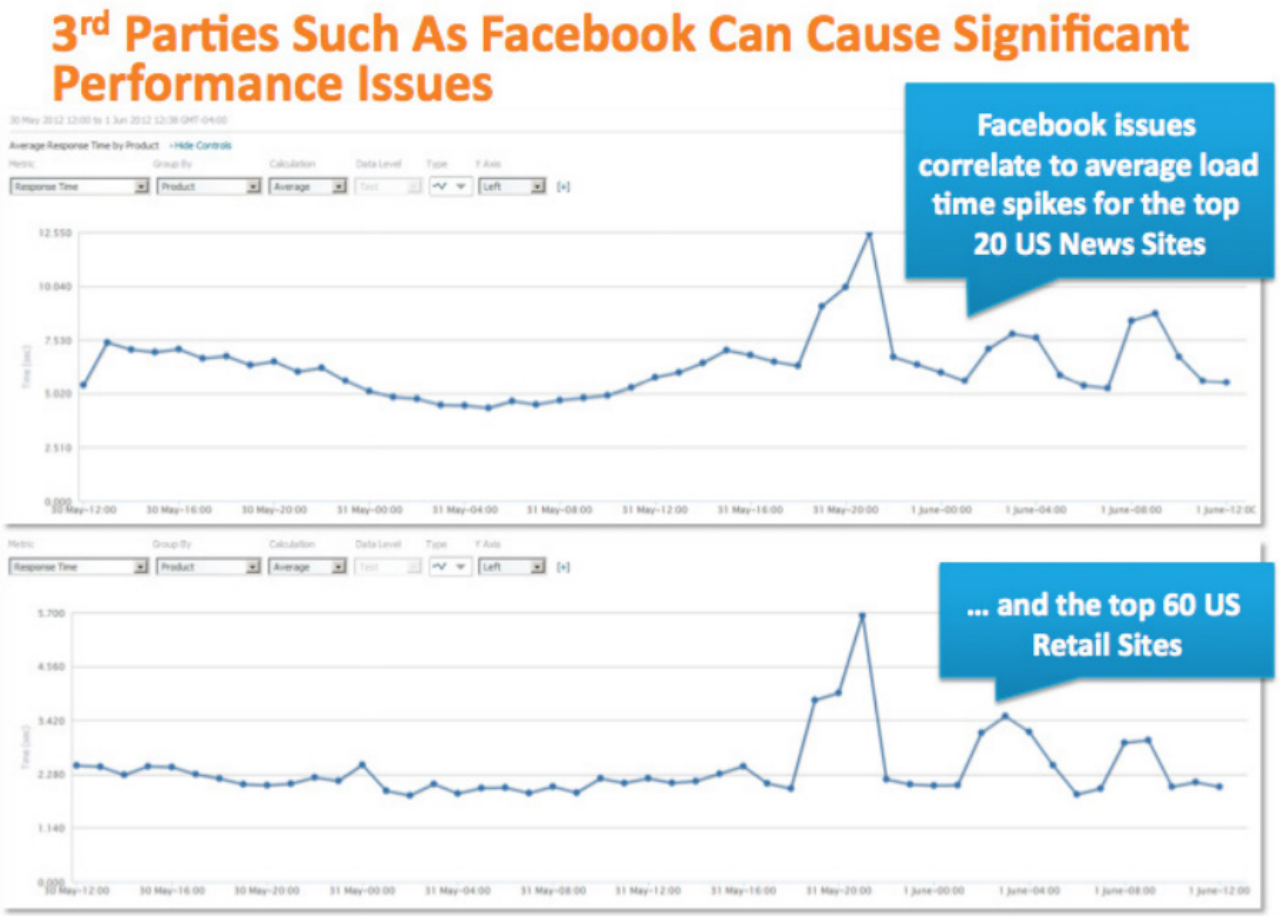
So you've checked and tested everything thoroughly. You've avoided all of the mistakes listed above and you're sure that your web or app will be able to cope with whatever gets thrown your way.

But you might be forgetting one crucial thing. Even if your web or app is 100% prepared, your third party plugin or module might not be. Third party plugins and widgets like social media icons, advertisements or even Google Analytics codes might just trigger the Single Point of Failure (SPOF) that will bring down your entire system.

Widgets from Facebook, Google and Disqus Could Crash Your System



In July 2012, Facebook suffered sporadic outages for a three hour period, triggering a widespread ripple effect and slowing the performance of thousands of news and retail sites. The reason? All of these companies have the Facebook 'like' button on their sites. The code generating its appearance sits on Facebook's servers. So when Facebook went down - everyone fell with it.[7]



Source = Compuware APM[8]

There are two aspects of third party dependency. I've just described an example of a failure with the frontend. However, failures in the backend due to slow third-party API response times can also occur.

For example: let's say your online shop is integrated on the server side with a delivery company and you're using API calls for information on delivery costs. If their service goes down a peak holiday day, your sales and product pages might also slow down. You can't put your destiny in someone else's hands - you must have full control.

SOLUTION 6:

APPLY ASYNCHRONOUS SCRIPTING, SET UP FALLBACKS, AND CHECK YOUR CONTRACTS

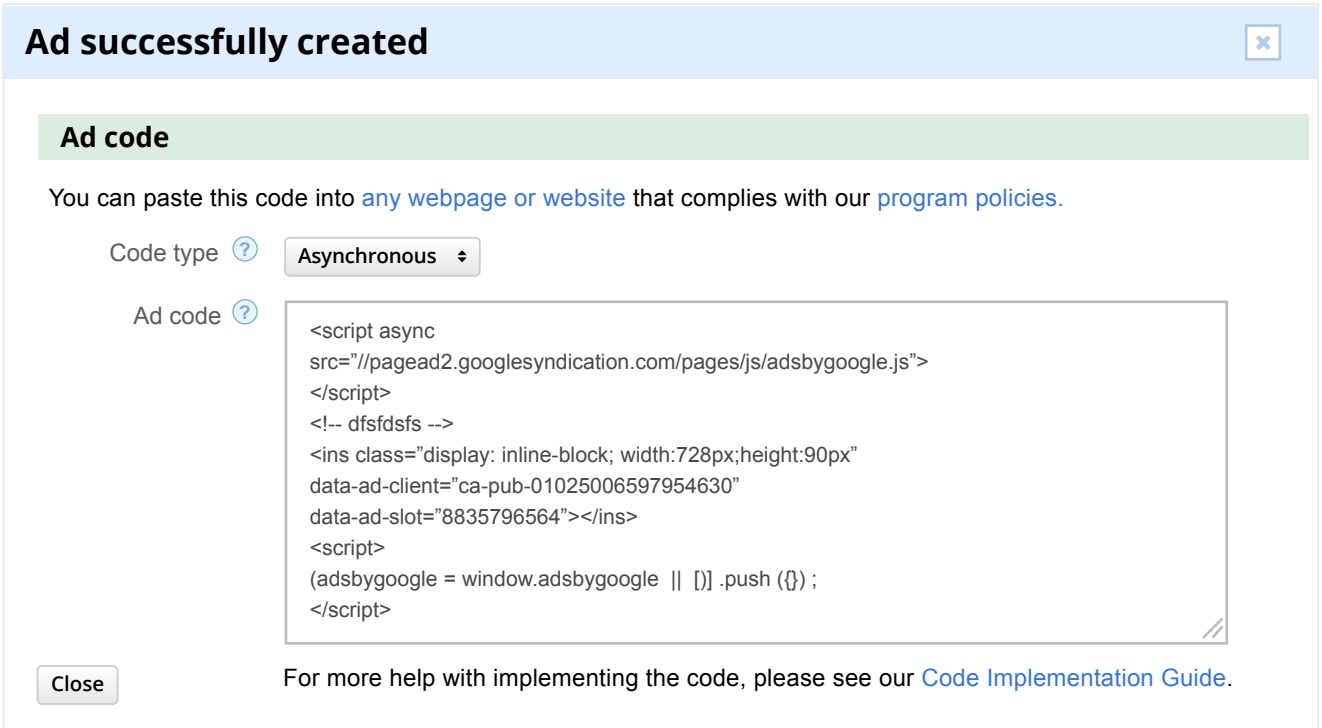
Each type of third party failure needs to be examined and dealt with differently.

First of all, let's look at your front end dependencies.

Have your third party integrations been added **synchronously or asynchronously**? If they've been added synchronously, they could bring your entire page down.

Synchronous scripts block all the subsequent elements on a page from rendering in the browser - **on every type of browser**. If your advert, widget or plugin has been added synchronously and it fails to load - your entire web page will be blank for around 30 seconds. That's thirty seconds of a completely blank screen. How many users do you think will still be there when it finally does load?

Go through all of your code and make sure that all of your third party integrations have been added asynchronously. If they haven't, then change them! This ensures that, if there's a problem with your third party, only their content will fail to appear. Your content and most of your site will appear perfectly normally to your users.



But of course, you will still need to test that it will work within an acceptable timeframe and to see if any of the third party widgets slow down. Try changing your DNS resolving process so it points the browser to a nonexistent address instead of an actual third party and observe the results.

Now for the backend failures…

It's worth setting up a 'fallback' in case an API timeout or failure occurs.

Your response time can't be faster than your third party API. It's important to set in advance the limit that you're willing to wait until a call is considered a failure. Your limit usually will not go over one second.

If there is a failure, you may want to switch off the advert or widget so that this one piece of functionality won't appear. This is preferable to letting it bring down the entire site. If the functionality is important, you should inform the users that you've disabled the feature for technical reasons. Going back to our example about the product delivery service integrations, your customer would get a message saying that the delivery information is currently unavailable.

Another option is to link it to previously cached information. Let's say you don't want to tell your customers that your delivery information is unavailable. When the API to the delivery page fails, rather than letting it crash your site or displaying "Sorry", you can show users the previously cached response adding "approximately" to the cost. This means that you can still provide people with some information about delivery costs and zones - without promising them the most updated information.

Finally, make sure that you check your contracts and that you're well aware of all the limitations and conditions of your third party providers. And, of course, learn from every issue by logging the details of all the timeouts and API call failures.

CONCLUSION

HOW TO ENSURE YOUR WEBSITE OR MOBILE APP WON'T FAIL ON PEAK HOLIDAY DAYS

There are evidently many reasons why your web or app might fail on busy holiday days - and the consequences of such a failure are serious. However, if you take these steps, you can feel confident that everything will be ok on the big day:

1. **Run Load Tests From the Production Environment**
   Choose a time that traffic to your site is low and run a live test in your production environment. Use tools to simulate the load from multiple geo-locations and devices.

2. **Try to Bring Your System into the Failure**
   Don't be satisfied with a successful test. Keep on increasing the load on your site until you see what will fail, when it will fail and how it will fail.

3. **Find the Critical Resource**
   Know what your weak points are; set up alert monitoring dashboards for these critical issues and keep them running.

4. **Observe Your Application Logs**
   Find out when the problems are occurring and why.

5. **Divide Your System and Stress Each Section Individually**

6. **Set Up Back-Up Servers and Locations.**
   If you have a power outage, make sure you can be back up and running within minutes.

7. **Incorporate End User Performance Testing Into Your Backend Testing.**
   Use tools to run these measurements automatically

8. **Check Your Third Party Integrations.**
   Make sure all scripts are applied asynchronously, set up a fallback in case of API failures and check your contracts.

That's it! Follow these guidelines and I'm sure your company will make headlines for all the right reasons on every big event!

FURTHER READING AND USEFUL TOOLS

Sources & Further Reading:

- FierceRetailIT.com
- Appdynamics
- comScore
- Mobilefomo
- IBM Digital Analytics Benchmark Report
- Yotta - Site Optimization and Web Performance Blog
- Forbes - Facebook Outage Slowed 1000s of Retail, Content Sites
- Compuware APM
- Steve Souders: "Your Script Just Killed My Site"

Links to Useful Tools

Throughout this whitepaper, I've made reference to a number of valuable tools and solutions. Here are the website links for further reading:

- BlazeMeter - Cloud-Based Performance Testing Solution for Mobile, Web & APIs
- Cassandra - High Scalability Database
- Chrome Developer Tools - Developer and Debugging Tool
- Firebug - Web Development Tool
- JMeter - Open Source Load Testing Tool
- Logentries - Log Management and Analytics Tool
- mongoDB - Open Source Document Database
- New Relic - Software Analytics
- Perfecto Mobile - Mobile Testing and Monitoring
- Redis - Open Source Data Structure Server
- Sauce Labs - Mobile. Cross-Browser Testing
- Splunk - Operational Intelligence Platform
- WebDriver Sampler - Executes and Collects Performance Metrics