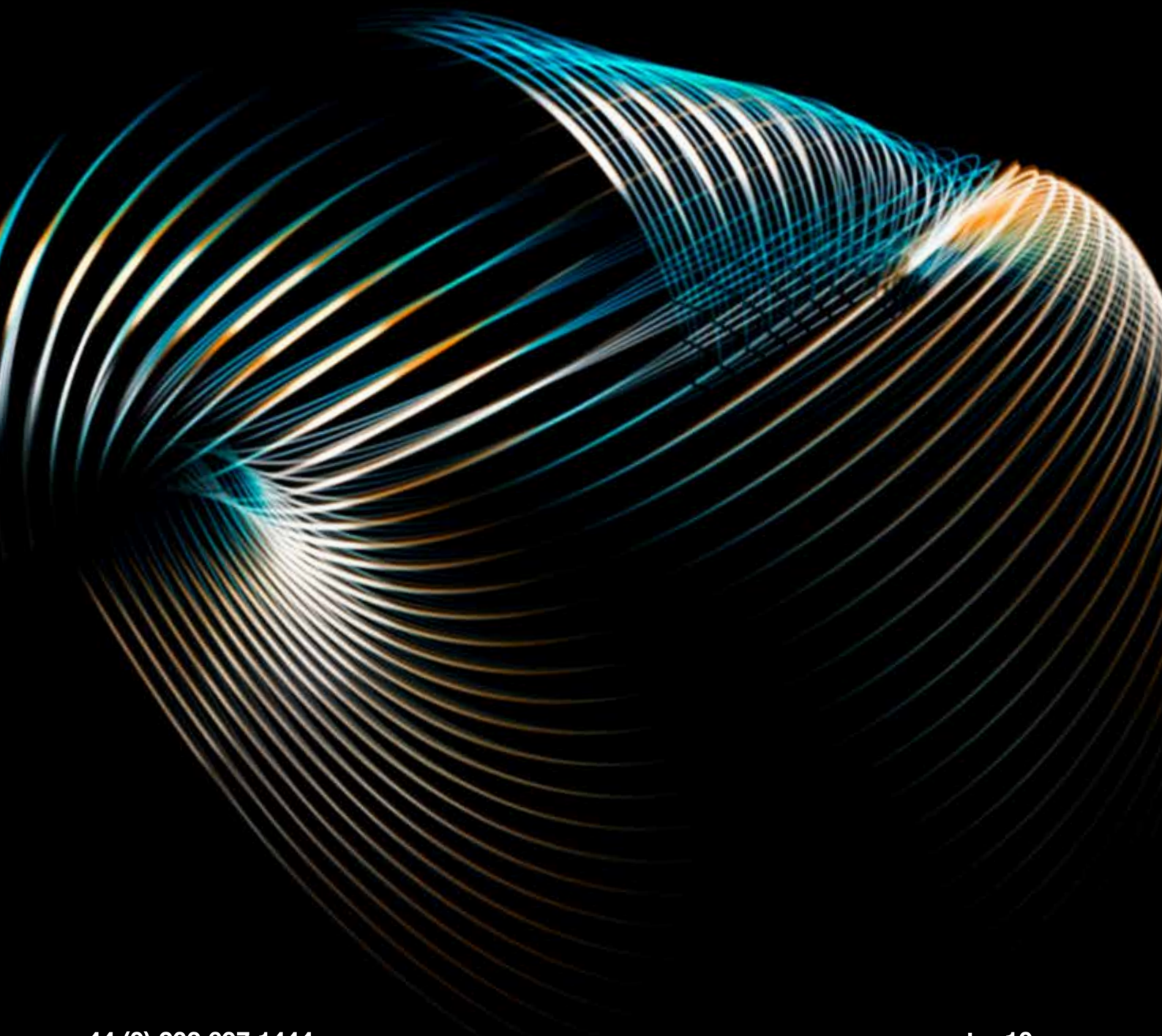
Ten10

Whitepaper

Performance testing in non-representative environments - 10 key factors for consideration



+44 (0) 203 697 1444

www.ten10.com

What is a non-representative environment?

It is highly unusual for performance testing to be carried out in a perfectly live-like environment.There's normally at least a couple of differences between the performance test environment and the live environment such as data sizing, backup and storage infrastructure, interfaces to other (live) systems, or cross-site data logging and data replication.

More realistically, there may be compromises due to the costs of creating and maintaining a full-size non-production environment which mean that CPU and disk-write speeds may be specified at a lower capability in the performance test environment than for the production system, even if the number of physical devices are equivalent between the two environments.

Any one of the examples mentioned above could arguably be said to be a reason why a performance test environment is non-representative. This does not mean that you should not execute performance testing, however you should acknowledge that the performance testing will therefore not necessarily represent the outcome of testing in a live-like environment.

> "An environment that is nominally 50% in scale of the live environment doesn't then equate to 50% of the performance"

Working with limitations, such as time, is an accepted part of a risk-based approach to testing; in performance testing this can often manifest as focussing on the core transactions around the use of the system rather than all possible transactions. While test environment limitations with server resources, test data and architecture are not ideal in performance testing, the risks these limitations present need managing as with any testing risk, when using a risk-based approach.

This whitepaper takes a closer look at some of the key factors that should be considered when performance testing in a non-representative test environment.

Designing your performance test environments and performance tests

Testing against non-representative environments is largely an art rather than a science. Although extrapolations and trends may indicate different degrees of performance of a system, the number of factors to consider when the environment is not a true representation of live makes it very difficult to apply a generic formula or rule of thumb.

An environment that is nominally 50% in scale of the live environment does not usually equate to 50% of the performance of the live environment, for example a marginal decrease in memory on a single server could theoretically grind an entire system to a halt if that server is already utilising close to all the memory.

In the same way, adding lots of new servers and resources to a system will not necessarily improve overall system performance if the existing servers are not being highly utilised.

In some situations using a non-representative environment for your performance testing may be unavoidable. This paper explores a number of common considerations that should be taken into account when designing and executing your performance tests in this situation.

Left-shift performance testing

1. Using non-representative environments to left-shift performance testing

Firstly, it is important to understand that sometimes testing in a non-representative environment is actually the most appropriate thing to do.

Depending on the design of the system architectures, it is sensible to strategically test with a non-representative environment, particularly in systems comprised of services and/or APIs, which are typically more suited to left-shifting the performance testing (performing the testing as early in the software development life cycle as possible) to achieve test coverage earlier. This is best achieved within Agile projects and in some cases where suitable, in monolithic applications following a more traditional approach.

Although it is technically possible to deploy a single service on an environment representative of live, without the other parts of the system deployed there is little value in doing this for the purpose of performance testing as you will not be measuring the overall system performance. In reality, when left-shifting performance testing a better approach is often to deploy and run the performance test on your local machine, or on a scaled down test environment.

At this stage you are trying to identify issues around the code rather than the overall infrastructure and system capacity, so using a non-representative environment or local PC is the correct approach.

2. Network infrastructure

An important aspect of system performance, which can sometimes be overlooked, is the network architecture. While the architecture is often not designed with performance in mind, it should be noted that differences in architecture between live and test can result in small differences in performance.

An example of when network infrastructure impacts on system performance is the use of external and internal firewalls. Firewalls can contribute to general performance degradation and often the presence of firewalls or the rules employed are significantly reduced in test environments, especially in

Network infrastructure

a system of systems that rely on traffic from multiple internal and external sources.

Another example is stubbing, where a part of your system (typically an external part) is simulated during a test. If you are stubbing some external traffic this can lead to a reduction in latency over the network as it is unlikely you are exposing the stubs behind firewalls that would be employed in live. This is in addition to a reduced number of network hops to reach the stub's location. Furthermore if the stub is closer to the system under test (SUT), then propagation delays (the time it takes a message to travel between sender and receiver) are likely to be much lower, so consideration should be given to simulate them within the stub itself when possible.

Another key consideration is how users are accessing your systems and services. If you have a large user base using mobile devices where the connection and degree of packet loss/ dropouts is higher, then this needs to be factored into your load model to ensure this experience is taken into consideration.

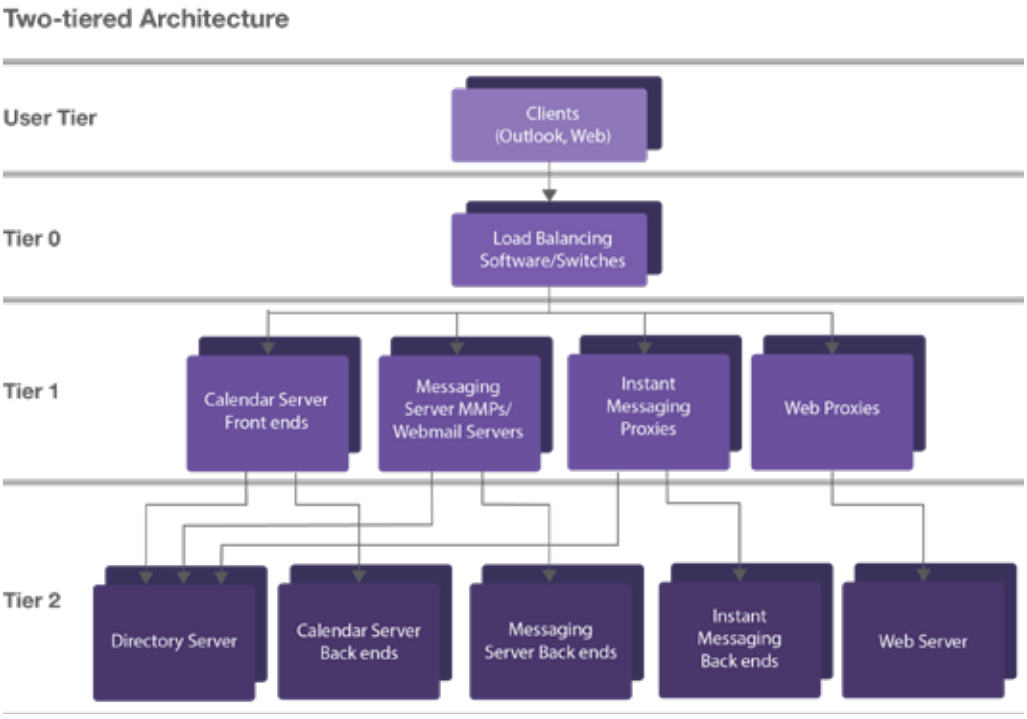
The use of content delivery networks (CDNs) should also be taken into account, and is discussed in detail below.

3. Logical system architecture

Similar to network architecture, the way you logically architect and deploy your applications can have an impact on the observed performance of a system.

For example, queuing of requests into a service and data source is a common pattern used to manage traffic. To properly test this it is important to configure the queues, services and databases in the same way such that the queue is subjected to the same performance variables. A queue configured differently in test for any reason can lead to bottlenecks being exposed in different parts of the system in test and live. A smaller configured queue size in live would result in a larger queue and reduced processing, that would otherwise not be seen in test.

Another example is how traffic is segregated. If your live environment is using clustered databases it may be setup to run updates against one database and any queries or reporting are run against another database. If you are only running against a single database in a test environment the lack of traffic segregation will impact the overall performance, as well as the lack of memory and CPU segregation.



Logical system architecture

Operating systems across devices

4. Operating systems, software and runtime

There are limits to how non-representative a test environment can be and still deliver meaningful performance test results. If the test environment does not have the same operating systems, supporting software (e.g. web server, database server, system libraries), or the same runtime/ scripting engines (e.g. JDK/JRE version) then the behaviour of the SUT may differ greatly from the behaviour of the production service. Even with the same versions, the configuration (e.g. Java Runtime Garbage Collection options, heap size) of these can have a noticeable impact on performance.

While it is possible that using a fundamentally different operating system may invalidate your performance tests, perhaps less obvious is the fact that, using different patch versions could also have an impact in some cases. It is possible for different minor versions of operating systems and software to exhibit different performance characteristics.

The operating systems, software and runtime you use, and more specifically the minor versions of these, should be considered for all types of testing, including operational acceptance testing (OAT) and functional testing, not just performance testing.

5. Scaling

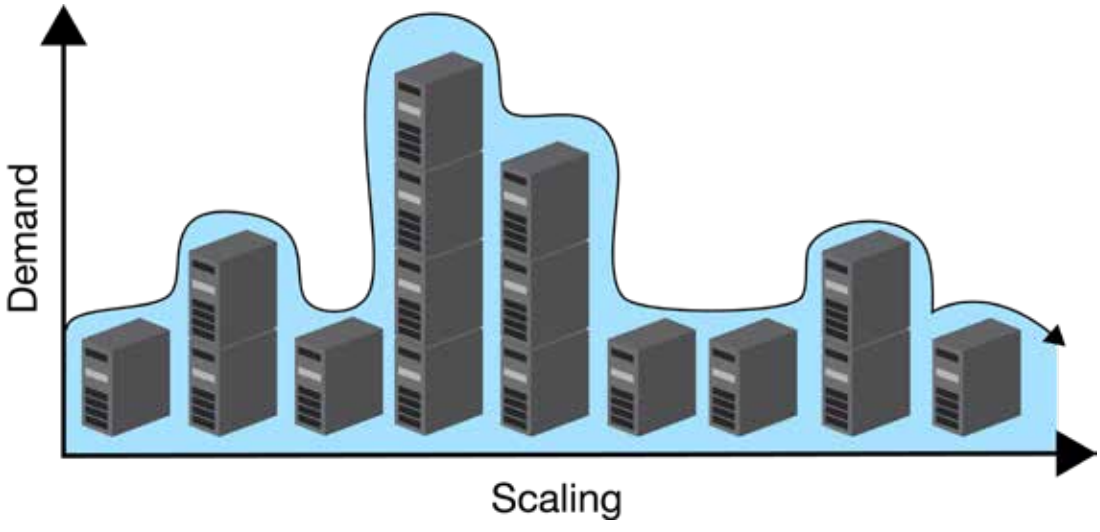
Scaling is a key decision when designing both your live and test environments and your performance test.

The key benefit of scaling is the ability to add resources to your system as a greater amount of existing resources are utilised. Two scaling options are available; horizontal, where further servers are added to complement the existing servers, and vertical, where additional resources are added to an existing server or servers.

Scaling using the cloud is an increasingly common deployment pattern used to negate performance risks in a live system. Indeed, testing the scalability of a system is a common feature of a performance test approach.

For the purposes of testing in non-representative environments, it is important to apply the same type (horizontal or vertical) and configuration of scaling across the whole system to ensure the results are representative as the behaviour of system can vary greatly depending on how it is scaled.

If you are only using one node in your test



Scaling graph

environments but you are using multiple nodes in live due to scaling, it is important to consider how things like load balancing and server affinity may result in additional risk; if you are only using one node in test, specific functionality like server affinity may not even be tested.

It should be noted though that scaling alone will not deliver reliable performance testing results. For example, poor coding practices and techniques can result in issues in live regardless of any scaling employed. These could be identified through early component-level performance testing in an environment not designed to be representative, as described in section 1.

6. Database configuration and data sizing

As described above in section 2 - where we looked at database clustering -, the configuration of databases can impact the results of performance tests. A number of factors come into play with database configuration, behaviour and sizing. Some of the factors include:

Query structure and optimisation

Basic good practice in the structure and formation of your SQL can go a long way. There is no sensible reason for the structure of your SQL to be different in different environments, but needless to say it is important. As is carrying out query optimisation. Again, optimisation of queries should not result in any differences between environments, but is an important consideration in the performance of a database and system as a whole.

Database instances

It is common for multiple database instances to exist on a single server, and in test environments it is just as common to have multiple instances on the same server for entirely different systems or different versions of the same system where testing is taking place on different builds. While typically in performance testing this would be avoided, it is important to ensure that instances for other systems or versions are not running on the same database server being used for performance testing.

Even with the same memory and IO configuration it is likely the performance of the actual server will be degraded when running multiple instances.

Indexing

Indexes improve lookup times in tables by removing the need for a linear search of a table. an index is vital for large, often-searched, tables. Whenever new or updated data is applied to a table the index slowly becomes fragmented. Indexes need to be rebuilt and can be scheduled through configuration applied to the database. It is common for this configuration to be different in live and test environments to reduce maintenance and support time on test environments, but this can have a significant impact on the performance of test systems.

Clusters

Using database clusters can significantly improve the performance of your databases and the system as a whole. This is mainly around operational considerations around failover of services. Not using clustering in this way doesn't always have any impact on the results of a performance test unless you need to simulate failover as part of the performance test.

For example, if you need to look at the performance

Database configuration and data sizing

of the actual failover process and the system performance post failover.

Row / table locking

Locking of rows, tables, connections and entire databases is an important functional aspect of databases that protect shared resources, for example multiple queries updating a single piece of data. However locks can result in unwanted contention, slowing down the processing of data. While the process of locking data in a database will not differ between environments, the way in which the test runs may impact the degree of locking that takes place between a test and live environment. This relates closely to data sizing described below.

There are many more factors for which should also be taken into account such as connection pool configuration and memory configuration among others.

Many of the above considerations can be attributed to differences in manual configuration of servers and databases. If you are using DevOps practices (including using the Cloud) and tools like Docker, there is less need to consider manual configuration of databases (and services) as the assumption is that you would apply the same configuration across environments.

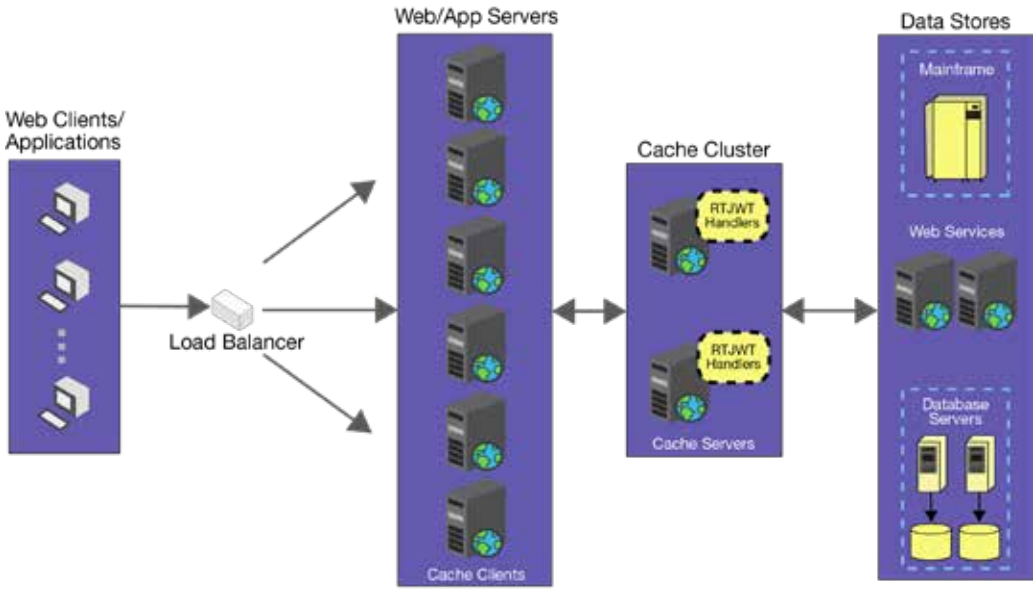
Further to database configuration, data sizing is another important consideration. That is, the volume of data available in a database. A good example of how the volume of data can cause differing results is data locking. For example, if the amount of customer data in a database is significantly reduced in a test environment it is far more likely to result in locks in the database as it is more likely different users will be trying to access the same data in the database at the same time.

Another key consideration in data sizing is ensuring your actual data is representative. For example, a customer table may have the same number of records, but if the makeup of these surnames is different in your test environment it can lead to false positives. A typical example is fuzzy matching surnames - such as Smith and Smythe.

7. Caching

Caching refers to the storage of previously retrieved content for future use, typically larger more static content like images. Caching takes place for anything being sent by web, application and database servers. This caching may take place in two different ways, most simply defined as client caching and server caching.

Client caching is where content is stored on the user's machine such as



Caching

the caching which takes place in a browser. Server caching is where common content is made more readily available by the system, such as storing content in the memory of the server to make retrieval faster than it otherwise be when retrieving from disk, or by storing the content on a proxy located somewhere nearer to the client (reducing network factors).

All of these factors need to be taken into account not only when designing your performance test but also when looking at how your live and test environments are configured. For example, if your user base includes users throughout the world, does your test make use of cloud-based load injectors and does your test environment employ the same type of server caching?

When using a protocol-level performance test, it is also important to configure caching simulation to accurately represent what the users will see. This simulates what takes place for client-side caching, for example in a web browser. Most of the common tools used for performance testing include some sort of caching management that performs the function of caching simulation. It is important to configure these accurately, including ensuring the HTTP headers are properly parameterised so the test continues to work in the future.

For systems with very high utilisation a CDN may be in place. CDNs also include elements of server-side caching, and is discussed further below.

8. Content delivery networks (CDNs)

Performance testing a system that makes use of Content Delivery Networks (CDN) can add complications and some unknowns if the implementation of a CDN is new, especially when looking to benchmark the performance of a system. If the CDN in the test environment is not representative of live this can further the complications and unknowns.

The following questions should also be asked in order to better-understand how the CDNs may affect performance test results:

- Is the CDN available in the test environment?

If the CDN is not available at all in the test environment, the results can be significantly different to what may be experienced in Live with the CDN in place. If you are making use of a CDN in Live it is vital a CDN is used in test to get meaningful results related to caching and content delivery.

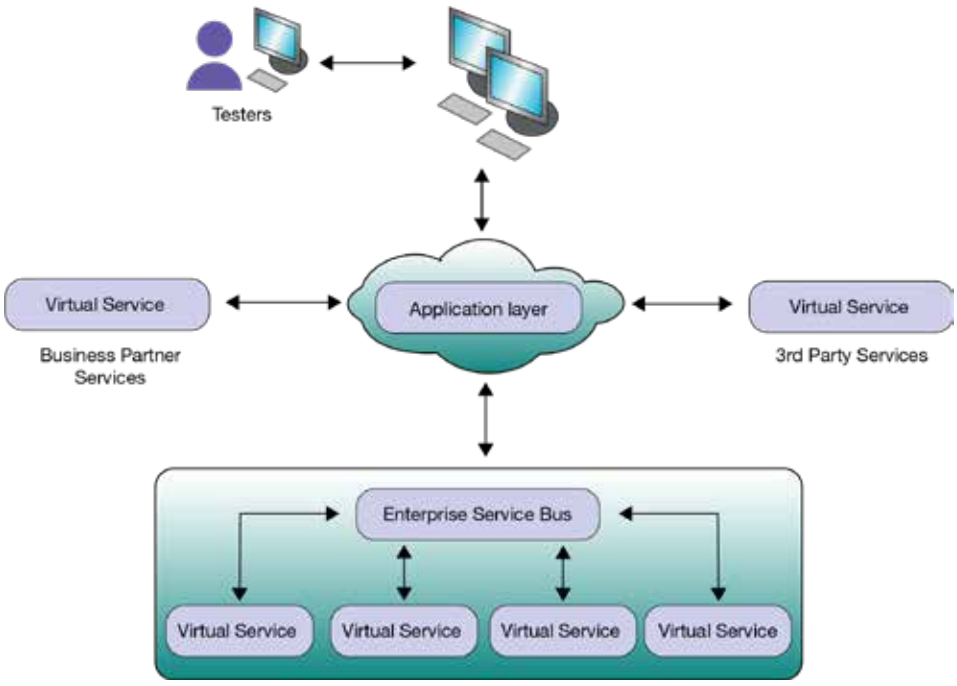
- How do CDNs handle traffic coming from one set of IP addresses (the load injectors)?

- Do they see these connections as they would individual connections from users, or are they treated as a DoS attack?

- Is the CDN configured the same? Is it available in the same locations and at the same scale for a performance test?

- Related to caching above, what caching is being employed and is it configured the same in test and live?

It should be noted that sometimes CDNs do not allow any form of performance testing against their systems (through test or live environments) and you may have to accept the contractual SLA of the CDN provider will be met.



Downstream and upstream systems/service virtualisation

9. Downstream and upstream systems/service virtualisation

Service virtualisation (also known as stubbing or mocking) is normally used to prevent overloading of external service outside your control or where specific scenarios can't be accomplished on an external service. Performance testing external systems/ services indirectly through a test of your system may either result in the service being blocked or being throttled in some way, or require you to schedule the use of these systems/services, which may cause contention with other activities.

Virtualised services can be setup to work in different ways, either by simulating normal usage or by simulating exceptional conditions like peak load. It is important to consider what you want to achieve when using virtualised services, over and above ensuring you are not overloading external systems.

10. Data entry points and end points

Pushing data into a system at the protocol layer, rather than the graphical user interface, is a common approach in performance testing. This is designed to look at the performance of the server rather than the client. This approach will result in a difference in the end to end performance of a system in live compared to the results measured in a test, as the processing taking place on the client machine is not taken into account.

Another consideration is to ensure the data entry point being used in the test is actually correct, and that a different end point isn't being used for some reason. While this may seem obvious, it is not uncommon to see performance tests use an endpoint that doesn't represent live usage, bypassing part of the system. An example is bypassing a secure service that sits in front of an internal firewall, that then forwards on traffic to another internal service not using encryption. While this may simplify the test by removing encryption, the true performance of the system will not be measured (notwithstanding the fact the client performance isn't being considered).

Data entry points and end points

Conclusion

Performance testing can always add value, but as described above, there are many considerations that need to be taken into account when using an environment that differs to Live.

There are certainly benefits to using non-representative environments for performance testing purposes, including; where an Agile approach is employed to identify potential performance issues early and against specific services, to perform basic benchmarking, analysing controlled increases in concurrency, identifying threading issues in the code and identifying locking issues in the database, to name just a few..

For performance testing it is important to properly assess and consider all the factors that will come into play including how the environments are designed and how you design your test. You cannot just assume that using 50% of the load against an environment with 50% of the capacity will provide you with accurate results.

Whilst you can identify and understand risks by applying some degree of extrapolation or formulae to determine the performance of a live system from a non-representative test environment, it is important to spend time understanding what the differences are and assess:

- What the impact on the performance test results is likely to be?

- What observations and risks identified in test are likely to be replicated in live?

- Which differences between environments are likely to expose the most risk?