Process and Techniques of Performance testing

Guidelines Document

COMP 120 Web Engineering

Version No 1.0

05/06/2011

System and Scalability Team

Shawel Negussie, Alex Martineau, Sonny Zhao, JianJeng Tan

Ver	Date	Description
1.0	05/06/2011	Guidelines document

Audience:

The audience of this document is for Application performance testers, business stakeholders, users and anyone who has a stake in the business bottom line.

Table of Contents

Section 1 Requirement Collection

1.1 Business value in performance

Often companies do not consider performance testing as important as perhaps functional testing. After all, if business transactions do not execute correctly, it will inevitably lead to failure. But, performance testing is also equally important. It is worth noting that many companies fail for factors other than functional issues. There are many startup companies that fail because they underestimate how successful their business can be and do not plan for scaling and meet demand during time users need the service.

Additionally, even for most companies that do performance testing, it is done as an ad-hoc measure. This means most performance testing is done after the system is built, tested and in production environment. The reality is, correcting performance problems take a lot of effort and possibly difficult to correct during this time. The reason is performance trouble can happen due to flaw in design, undetected performance bugs, compatibility issues, configuration issues, defects that only show up during high utilization and many more. Therefore, the goal is to integrate performance testing as part of development for every iteration cycle. As a result the process of development takes in to consideration performance of the system. Such an approach will design system with performance in mind versus retrofitting what should be an integral part of a system.

1.2 Identification of Application type

Identification of application type means knowing the nature of the business and the level of services the company provides. Is it a type of service where functionality changes are very high? Does the business need to support high volume of data at any given point in time? For example, does the business involve a lot of media files? Or is it strictly form based? Does it possess aspects of social networking? Is the services mission critical? How likely is performance important in terms of user satisfaction? What are the prospects of growth, both in terms of functionality and user base? All these questions must be answered in the requirements document.

1.3 Identifying User Base

Identifying user base is one of the most important aspects of performance testing. The reason is if we do not know who uses the system, we lose insight in how they use the system.
A key method for collecting requirement about users is to gain insight about the type of business and the services it provides. An important first step is to collect data about how many users are currently using the system. In terms of performance testing, requirements will be put to use, when we decide to model how we are going to handle different aspects of users such as behavior, size and location. When we generate users it will follow requirements mimicking real users.

Additionally, identifying user base will define the various architectural decisions we need to make. This range from, the kind of physical servers, to the level of tiers of servers and implementation tools we need to consider because of estimation of our user base and growth prospects. This will directly define what kind of testing tools we need to use to adequately test the system for performance.

1.4 Identification of Networks

Identification of networks simply means, how do users access the systems for service? We need to determine if the system in focus is only used on a LAN network. In this case this requirement needs to be clearly documented. It is not entirely far-fetched this could be the case since most big corporations develop software to be used in house only. However for the majority of companies that provide service, WAN is a typical requirement. Ultimately location of possible user bases will need to be investigated to decide how much and how frequently is the system accessed according to region. It also goes without saying, all aspects of connection such as VPN, cellphone, or other means of access, need to be considered as well.

1.5 Identification of Critical business transaction

This in my view is one of the biggest and most important requirements for performance testing. The reason is business transactions define the services provided by that company. In this regard, we need to follow the 80-20 percent rule as well[1]. This means 20 percent of the business services are utilized the most and really defines the majority of the services used. This also means performance of this service will directly impact the company's bottom line, either positive or negative.

The way to collect these requirements is to either go through the use-cases and identify the most important business processes or through traffic analysis, using albeit expensive tools such as Wily CEM to track user behavior. It is argued that critical business transactions usually do not exceed more than 20 even for large systems[1]. We have to make sure we collect information about each individual transactions, what they do, how important they are in terms of the companies bottom line and how many users use them.

Failing to identify critical business transactions makes the performance test results invalid since we are measuring the wrong transactions. Even more, we are spending tremendous amount in terms of resource, budgeting and time. Ultimately, it becomes counter productive, if not a complete disaster.

1.6 Identification of Load Variations

Load variation happens when system utilization varies at different times. This means we need to identify peak loads the system might experience trouble. In many circumstances, peak loads can also differ by season, such as shopping season, or can revolve around particular days. It is also the case the system is targeted during nighttime for heavy batch processing originating from both inside the company and outside. Users use automated means to access transactions at such a high rate, it might warrant requirements for extreme but possible cases. Such requirements will be a basis for our testing, varying loads.
As it appears, we need to prepare for all cases of load testing where variations of these loads can happen at various times. We also need to record, how these loads occur, in terms of number of users, size of data requested, and how frequent are these occurrences. Once these data are collected, we need to classify them in terms of priority before we do actual design of the test scripts or the environment in general.

1.7 SLA Requirement of Acceptable User Experience

SLA stands for service level agreement. It is a contractual document that states a set of performance metrics limits that needs to be considered when a system is getting build. Usually a customer will not accept a violation of these performance metrics. Relying on SLA documents is important for companies that deeply care about productivity issues. Companies install front-facing performance measurement applications that track SLA violations for every transaction that enters the companies infrastructure.

For gathering accurate SLA requirement, we need to clearly define business transactions and the level of performance required for them to execute within acceptable limits. It is not always possible to get this data, but one way to collect information is to investigate similar applications and make an inference about what an optimal baseline should be that is acceptable for both the user and the company. These measurements are very concrete in terms of numbers for every transaction and likely will have consequences if performance targets are not met.

1.8 Requirements on Projected growth for users and System Functions

Projecting functional growth and user growth is also important aspect of performance testing. The fact is failing to anticipate the potential growth of a company and stagnation in terms of capacity will damage the reputation perhaps irreversibly. If competitive advantage needs to be maintained, especially in markets where performance is a primary goal, anticipating increasing pressure on a system is vital.

One estimation technique is to start from the business reality of the company. If for example a company has 1000 current users of a product, and if the prospect of growth for the product in terms of market trend is very high in the future, it is important to have a target number that reflects much bigger user base. This number will be set as a requirement goal that we need to monitor and test regularly to anticipate changes before that day comes.

Section 2 Designing of Testing Environment

Admittedly, when it comes to performance testing, test environments widely differ from production setup. Since we cannot do performance testing in production, how we go about setting up test environments will decide whether the data we collected will be valuable or not. If it is done the wrong way, it will result in a huge investment waste attempting to do performance testing. The following section will setup guidelines on how to setup environments for performance testing.

2.1 Types of Testing

2.1.1 Load Testing[1]

Typically the most common type of testing is load testing. In this scenario, a performance testing team will set initial performance goals based on the requirements gathered for various business transactions identified. Transaction scripts will then be generated using one of many performance-testing tools such as IBM rationale or HP Load Runner. The system will start generating number of users, using load generators, executing transactions identified in the requirement phase. Load testing will attempt to stimulate actual number of users and transaction sequences that would exist if it were real users. Once tests are complete the performance metrics will be analyzed for bottlenecks, failures that need to be corrected before next iteration of performance testing. It is important to note, the goal of load testing stays within a realistic performance limits defined in the requirement.

2.1.2 Baseline Testing[1]

Baseline tests reflect an ideal case for testing performance of business transactions. Baseline tests are done for each individual transaction when no other activity is running in the system. Always do numerous iterations of baseline testing to get most accurate data. This test should be done before we do load testing. The reason is since baseline tests are considered an ideal case; the metrics collected during load testing will be compared against it[1].

2.1.3 Stress Testing[1]

Another aspect of performance testing is stress testing. In this case, we are attempting to push the limits of performance testing to the boundaries of breaking point. The assumption is not realistic in terms of real usage of the system. But it gives very good performance analysis revealing symptoms about how the system breaks down.

2.2 Mimic Production conditions

An entirely unrealistic option of setting up testing environment is to mimic production setup. This means using an exact number of servers, clusters and server tiers similar to production environment. Companies, who especially need performance testing the most, will have difficult time with this setup for many reasons, but cost is a primary driver. Mimicking production environment costs not only in terms of getting the systems but also man-hours required to set up and manage it. It might not even be possible at all, if servers are geographically distributed on a massive scale. Therefore alternative setups need to be explored. However test data gathered with this method is the most accurate.

2.3 Setup Scaled down Environment

According to Ian[1], scaled down setup means designing performance test environments that closely mimic the structure but with a much lesser scale. For example, suppose we have a three tier production setup; lets say we have 16 webservers, 4 load balancers, 16 middleware servers, which are connected to 16 application servers and a supporting 4 database clusters. This will be too expensive to replicate for performance testing environment. Instead we can scale down in proportion to each other, preserving the structure of how they are setup in production. In this case, dividing the number servers by two or 4, will give us the fraction of the actual setup in production.

The important thing to note here is, performance-testing load will also have to be scaled down. Instead of testing for a full load, we will have to test for half or a quarter as much and we analyze system performance for that number. In this regard, by proportionality, production systems will handle full performance load, if analysis for fraction of the load in test environment provides good measurement.

2.4 Mimicking User distributions

One of the key requirement analyses that need to be done is identifying users. Knowing the users will provide valuable result on so many levels beyond performance testing. Regarding performance testing, identifying the kind of users, their behavior and geographical distribution will give us valuable information to generate users that is very similar to actual users. The impact of not identifying users behavior will give test results that are not entirely realistic. Designing test results need to take various considerations:

- Generate users for testing that reflect the range of IP addresses as real users. The upside of this method is, we will see distributions of users hitting the system that closely approximates behavior for real environments. We want to avoid the danger of underestimating few ranges of users and we end up, changing the behavior of how the systems responds.

- The reverse situation also needs to be considered. Very few users using the system at such a high load is very different from large amount of users using the system at a high or low load. The key point is always mimic user behaviors. A curious example happened recently at a major company. In the company's performance testing design document, there is no mention of testing a single user (coming from the same IP) utilizing the system at such a high load. Even though the company was using many servers, it seemed trivial a single user could not bring down the system. But as it happened, a single user web scraping for data from a database brought down the system because the load balancer failed to balance requests coming from the same IP address. The result is, only one out of 8 integration server acting as middle tier between application server and database was getting hit all the time. While the other seven servers remain idle, and the database cluster working normally, a single server could not handle the load, CPU started running very high and the system hanged with a lot of threads open. This was identified as a Performance testing flaw, as well as a load-balancing flaw, which needed to be corrected. Important to note here is, this behavior could have been caught early by covering aspects of user behavior never considered during testing.

- Always consider users geographical location. As discussed in the requirement section, generating load only in a LAN environment when almost always users are connecting from the Internet is not realistic. User load generators need to be designed in such a way to take account for place not close to the testing environments. If users are coming though alternative means such as a VPN, that needs to be considered too. Always keep proportion of how these users are distributed.

2.5 Approximating Storage Capacities and Test cases

Estimating amount of data in storage for performance testing is very important. If production environment has 50 million records in a database, testing performance with data of 1 million entries is not realistic[1]. In performance testing the amount of data that need to be considered need to be similar. This is because the complexities of having different size of entries give wildly differing result.

Additionally, accessing the range of data requested from the database is key. There might be performance issues with databases that reveal the need for indexing, or even optimizing database hits that need to be considered. And these issues arise if the data size is large enough and close to production size, otherwise they will be totally missed.

2.6 Designing transactions for testing

Another important consideration is to design business transactions we need to test. After all, the most critical aspect of performance testing is, to test the services the business relies on, are performing well. It is said that, not more that 20 business transactions are considered integral to company's success[1]. They are the most utilized and likely the once that need to be tested for performance. Identifying in the requirement section what these business transactions are will be key. Because if we instrument wrong transactions, the test results will not matter since the results we get will not reflect how the system is used.

In many cases, we also have to measure these business transactions individually; in a tier system of importance and how frequently they are used will differ between transactions. Additionally, different sequences of these business transactions also give different results and we need to provide best approximation for how users typically use business transactions in combination. Also, it is important to consider the variations of transaction types, their sizes and sequnces need to be scripted as individual cases and measure their performance. The data is entirely different and even key individual metric performance varies by a good margin when they proceed or follow other sets of transactions. User behavior is a good indication for designing such sequence of transaction execution.

According to Recommendation by Lan Molyneaux[1] , when testing transactions, it is best to start out measuring individual transactions with not other activity running in the system. This will reflect the base line measurement for that particular transaction. Also, this will be the time to recalibrate scripts generating these transactions and check against the expected output of the transaction. After, make a table of each individual transaction, record the base line, the number of users generated, transaction running time, number of iterations, rate of execution and outcomes to get an overview of how the tests measure up against the baseline.

Section 3 Metrics Identification and Mapping to Requirement

Requirement collection for application performance is a necessary first step. Historically many systems fail simply because corporations fail to know their audience and access how important or pervasive their application will be in the future. A key primary motivation for requirement collection is also to identify various measurements that give us statistics about performance aspects for our application. The following metrics will give us guidelines to understand the metrics involved in performance testing. We measure performance against this numbers and progressively compare them during times of schedule testing to see how we isolated and improved them. Additionally, these measurements will serve as service level objective measures (as suppose to SLA which are contractually binding) by which we track progress and communicate them with different stakeholders.

3.1 Availability Metrics

Availability metrics is a very important metrics. It indicates whether a particular system is responding to requests. It is obvious if a system is down all other metrics do not matter. So the primary goal is to have the system all up and running. Availability metrics are usually indicated by only two values either 1 or 0, up or down in a time series data or graph. All tiers of systems will have availability metrics. For example a database instance might be down but the webservers might be up. Either way service will be disrupted if any of the connected systems are down. With the exception of some clustered systems, where other instances might pick up the slack. The reason we test for Availability is, at various peak times, some systems either shut themselves or decide not to respond either because they cannot handle the load, or in some cases overheat. It is much better to find out these behaviors in simulated environments that real-time production environment while users are using the system.

3.2 Response Time Metrics

Response time metrics usually signify how fast a given system responds to requests. Response time metrics are measured in time (usually milliseconds, seconds or minutes). For all purposes, as discussed above, all critical business processes that define vital services a company provides, how fast they are delivered could make or break a company. The key reason for response time testing is not only measure performance under ideal conditions, but also, in many cases, when traffic is heavy. Simulations of response time will give us good measure of what load, system response time start to slow down and decide if that measure is reasonable against SLA and user satisfaction requirements indicated above.

3.3 Response Per Interval Metrics

Response Per Interval metrics is another name for throughput measurement. Typically interval is defined as 15 seconds, where as throughput usually measures for one second. This metrics measures how many requests of a particular size a given the system can handle at any given time; therefore it is usually defined as size/second (i.e. kb/s, mb/s, gb/s). This metrics measure system at a time where many request are coming and a how fast they are getting processed. Therefore it will give us an indication about the rate of processing. Usually bigger bandwidths, multiple server instances contribute to response rate since they can distribute processing over several systems. We say response rate becomes less ideal, when requests are coming at much faster than they are processed. I.e. they break the threshold ratio of incoming request/outgoing request, which should be < 1.0

3.4 Concurrent Use Metrics

Concurrent uses is a metrics that gives information about how many requests a system can handle at the same time. For example if Request A happens to occur at the same time as Request B, we have a case where, they will be processed at the same time. Normally this metrics is measured using load generators performing same business transactions in parallel over a given time.

3.5 Transaction Stall Metrics

Transactions stalls are mostly rare metrics, but I have experience with them using Wily Application performance management tool. Usually business transactions that take more than 30 seconds are counted as stalls. So the metrics is an integer number of stalled transaction. This measurement is a very good indicator of performance degradation over time. Even more, during development if there is an application upgrade, this metrics will immediately indicate a problem, if there are more requests taking more time than before. A very high number is also associated with processors hanging, CPU running high, during times of high utilization or even database SQL transactions performing poorly for large databases.

3.6 Server Load Metrics

Server Load metrics measures the mechanical aspects of systems. These metrics test how system hardware is performing. Memory, CPU, Disk I/O operations, Network utilization[1] and many more are primary targets of these measurements. It is important to note, when load testing, this metrics will indicate if there is capacity problem and sustainability problems enabling these systems to perform poorly. For example, if the machine is having a problem, usually these metrics run high indicating either we need to add more memory or switch CPU, disks and so on to support the load described in the requirement document.

3.7 Comments about Metrics

Performance metrics are rarely independent of each other. When a system is performing poorly, Server load metrics will start reflecting these numbers. Response time will also be very slow and stalls will start to accumulate. In such cases the system might not be able to handle it and might hang and not respond any more. Additionally, this might be a reflection on what the system can handle since the throughput might be very high and connections start getting dropped. But all this measures tell different aspects of the same system from a very different angle that will help us identify the problem. Whether it is increasing the number of apache worker process, or increasing the amount of CPU on an application server or re-indexing large databases, this metrics are good indicators in combination. They will help us isolate problem by looking at combination of metrics from different point of view.

Reference:

[1]Ian Molyneaux, The art of Application performance testing, 2009
[2] CA Wily Corporations, Willy Application Performance Management Guidelines