PERFORIC

performance is the key

WHITE PAPER

The importance of volume data

and other application performance and porting considerations

The importance of volume data

One of the often-overlooked parts of performance testing is testing with volume data. Volume data typically means loading up the database with a volume large enough for production. Depending on the type of business, the data retention rules could require six months to two years of customer data retention, which means the volume data needed for testing should be enough to simulate that amount of data.

During development and QA, data volume is not typically of concern. Most development and QA systems have a small amount of sample data that is perfectly adequate for functionality testing. However, this often means that when the database is loaded up with large volumes, there are issues that crop up which do not during the normal development and QA cycle. It's not reasonable to expect development and QA to have gigabytes, or in some case, terabytes of data storage available. This would make hardware costs prohibitively expensive for the company. The solution is to have a separate lab, either in-house or at a third-party, with the amount of data storage required for volume data testing. Volume data testing does not need to happen as frequently as the normal QA cycle but should be done prior to going in to production or releasing the product. To reduce costs, the data storage could be leased for the period of time needed for volume data testing. Since this activity is typically relevant toward the end of a release, the amount of time needed on the leased hardware is limited.

Volume data testing does not need to happen as frequently as the normal QA cycle but should be done prior to going in to production or releasing the product.

There are tools available that can be used to generate database volume. Some companies might even want to write their own data generation scripts. Here are some things to keep in mind, no matter the route taken to generate data:

- If the data is based on real customer data, the tool should allow for data obfuscation. Confidentiality of customer information is critical so customer data should never be used for volume testing. Typically, the security infrastructure of the volume data environment will not be as rigorous as a production setup so care should be taken with the kind of data in that environment. Also, internal employees should not have access to sensitive customer information and therefore customer data should not be used as is for testing. A good data obfuscation technique will address these issues

- The amount of data for each table should be representative of the way data builds up in the real production system. Volume testing requires a lot of database tuning including the use of proper indexes. If the data is loaded improperly, a lot of time could be wasted optimizing tables that will never have that high a volume of data. Conversely, if a table is not loaded with enough data, it might not show any issues until the database is in production and volume buildup starts to happen. The best way to gauge data buildup in different tables is to have a small sample database and go through a real world use case or scenario while monitoring the amount of data being written to the tables. Once the pattern of data buildup is understood for a single user, it can be extrapolated to the real world number of users.

- Ensure there is no bad data. For instance, if your software is not expecting a null value in a column and your test data populates that column with a null, it might throw off your entire use case. A lot of time and effort might be wasted debugging spurious issues due to the data itself and not the code logic. The volume of data might also make it difficult to identify and isolate issues. It is best to make sure there is some validation of the volume data either during the data load or after by using an intelligent algorithm.

Volume data testing can surface issues entirely different from functional testing.

Volume data testing can surface issues entirely different from functional testing. From performance issues due to a badly tuned database to application issues like overflowing arrays or inefficient business logic, it is best to catch these issues in a test system before it affects a production system with real world users.

The use of a third party lab for performance testing

One of the reasons performance testing is neglected by small to mid-size companies is the hardware expense. In most cases, a performance lab requires large disk arrays and high performance machines that are comparable to production systems. An additional expense to consider is the load driver software, which for established tools can run into the tens of thousands of dollars. It's not feasible to do large-scale performance testing as part of the initial development cycle due to these expenses. QA testing with tools like JUnit should be and have been part of the development cycle for some time. Alas, there is no such easy solution for performance testing. However, there is an answer: third-party lab testing. The needs of performance testing are such that they are well suited to a third-party lab.

Firstly, hardware in an external lab can be leased. There is no need to buy expensive hardware that would sit idle for extended periods of time during the early part of a development cycle when functionality is more the focus than performance. With the advent of the cloud, its easy to lease this hardware and quickly add or remove hardware during the performance testing cycle. This flexibility is key as its often difficult to guess how a system would perform during early performance testing. If a certain tier turns out to be a bottleneck, its easy to add more horsepower to that tier in a hosted lab. With a third party lab, a company does not have to worry about long turnaround times associated with buying hardware and setting it up and often not buying enough or buying too much.

With a third party lab, a company does not have to worry about long turnaround times associated with buying hardware and setting it up and often not buying enough or buying too much.

Secondly, performance testing should be attempted once the product is fully functional and has gone through a few rounds of QA and is stable. Its not as iterative as a development/QA cycle where functional issues are uncovered and fixed and tested again. You will want to do performance testing a month to a few months before going live, depending on the complexity of the product. There needs to be planning for a few regression cycles to make sure fixes applied to issues uncovered during performance testing do not cause other problems. However, a lot of issues addressed during performance testing are environmental, so they might not need code fixes and might not need as iterative a testing cycle as QA. A useful rule of thumb is that performance testing typically takes a quarter to an eighth the time of a QA cycle. Given this, it does not make sense to have expensive performance testing hardware sitting idle during the bulk of the development cycle.

Thirdly, a competent performance testing team can operate in isolation of the QA/development teams. They can tune the environment without having to consult with the core development/QA teams and can pinpoint software issues with concrete recommendations to the product team on how to fix them. Given the relatively high degree of freedom of this type of testing, having an independent or separate lab allows the testing to be done without involving core teams that are tasked with new development or bug fixes and have limited resources to dedicate to performance testing.

Some of the keys to successfully test in a third-party lab are:

- Clear understanding of functional flows needing to be tested
- Availability of good analytical tools and knowledge of the performance team in using them
- Clear understanding of the targets to be met as part of performance testing. This type of testing can be a never-ending process as performance can always be improved given enough time. Its important to know when the targets have been met. These targets are often called KPIs (Key Performance Indicators) and could be things like numbers of concurrent users with maximum latency for any given page, maximum network throughput for a given number of users, minimum numbers of rows to be updated in a key database table per second or the maximum amount of memory growth for a process. The KPI depends on what is important for the product so that the end user does not face any performance issues.
- Good relationship with the third party lab and understanding of Service Level Agreements (SLAs) so that new hardware can be quickly added or removed and downtime is minimized

If you are trying to sell your product to a company that does not run the hardware you support, your chances of a successful outcome are greatly reduced.

Reasons to port

Portability is the ability of code to run on different platforms or technologies. Portable code is a fairly broad term and could encompass a wide variety of technologies from browsers to databases to operating systems to application servers. Unfortunately, portability is something that does not happen automatically as part of software development but needs to be consciously planned for. Before we get into more details on portability, the obvious question is why, that is, why do we need to bother with portable code? There are many reasons, including

- Broadening your reach – Many companies build relationships with hardware vendors and get discounted hardware or other perks in buying from them. In other cases, companies have messianic zeal about one platform or the other, as most people who have sat in on technical sales meetings will tell you. If you are trying to sell your product to a company that does not run the hardware you support, your chances of a successful outcome are greatly reduced. A company will not go out and buy hardware just to be able to run your software. They run into support and training issues on top of the obvious cost issue. You need to be able to support their platform.

A portable product is typically a standards compliant product.

- Sales appeal – If you are trying to market your company for sale to another company that either has its own brand of hardware or supports a certain brand, you need to be able to support that same stack (stack is the term used for the combination of hardware and software including web servers, application servers, databases and operating systems). Being able to support multiple platforms increase your chances of being snapped up

- Changing alliances – Today, large software vendors try to own the stack. In other words, they like to have all components in the stack be developed in house. They see this as an advantage to their customers as it gives the customer "one throat to choke". This also means that customers are more likely to make changes to their environment based on the companies they are dealing with. For example, if they already have a relationship with a large database vendor and now that vendor offers their own application server, the customer could decide to change from their current application server to the one offered by the database vendor. If your product is in the mix, having it be quickly functional on the new stack could make the difference between a continued relationship with your client and a lost one.

- Compliance with standards – A portable product is typically a standards compliant product. Standards like J2EE and ANSI are designed to make portable development easier. Adhering to these standards means less risk of your product breaking when a newer version of your currently supported platform comes along. Platforms, such as application server containers, need to be standards compliant and if your product is in compliance, it makes upgrades easier for you and your clients.

Of course, portable development has its challenges, so you need to do the cost benefit analysis to see if it is worthwhile in your case. If you want to go with only the most popular platform for a given tier, that is your choice. As operating systems like Linux have become more and more popular, its sometimes an easy choice to only code for the most popular platforms. However, as with most things, one size does not fit all and if you do need to worry about portability, here are some things to plan for and keep in mind:

Use a type of source code model that easily allows platform independent and platform dependent components to be identified.

- Builds – Builds, by definition, have to be platform dependent, as they need to compile the code differently, sometimes using native compilers, based on the operating system. Use a portable build script like Ant which is an open source tool. This will shield most of your development staff from operating system oddities and you will only require a small staff of cross platform build experts to keep the builds running on all the operating systems you support

- Install and Packaging – Use a cross platform install tool so that you can have one set of documentation for all your different environments and one packaging/install vendor to deal with

- Source code – Use a type of source code model that easily allows platform independent and platform dependent components to be identified. A technique like source code layering or reuse layering allows this. Most code will be platform independent but the platform dependent parts of the code will have their own buckets for easy maintenance.

- Application servers – Languages like Java were developed to allow code to run in containers that shielded them from the native operating system. If you are using J2EE, Sun's site has tools available to ensure compliance with the standards. All J2EE application servers comply with these standards though each might have platform specific features for better performance. The balance here is between having completely cross platform code or having some platform specific features to enhance performance on that platform. This is a common dilemma and there is no easy solution other than strict source code control and clear understanding and maintenance of cross platform vs platform specific code (minimizing the latter as much as possible)

- Operating systems – If your product runs on an application server and does not need to access the native operating system directly, you are mostly shielded from operating system dependencies. If however, you do have operating system level dependencies, it is very hard to make the code be cross platform. There are very few standards between operating systems, so its difficult to apply the WORA (write once run anywhere) paradigm here. You need to put the platform specific code in clearly marked sections as described above in the 'Source code' bullet

- Databases – Adherence to ANSI SQL standards and staying away from stored procedures are the basic tenets to database platform independence. However, stored procedures are a powerful tool and can often give degrees of magnitude better performance than dynamic SQL. There are also potential security issues with dynamic SQL that are not there in stored procedures. Again, the issue is one of a balance between maintenance on one side and performance and security on the other, and its up to the development team to figure out where that line lies.

- Browsers – Use the minimum functionality to be able to do your design. In other words, stay away from the latest thing out there as it might not have been tested across browsers. Also, run tools like html validators and browser compatibility checkers to catch problems early.

The above steps are a good way to ensure compatibility and portability across platforms, but there is no substitute for testing. Its easy to see how quickly your test matrix can expand as you add more platforms and the permutation of different platforms pile up. Its essential to plan enough testing cycles for the various platform combinations. There are often platform specific issues uncovered during testing that require either a platform specific fix to the source code or a fix to the platform itself.

The best strategy for cross platform testing is to identify a primary platform and focus most of the testing on that platform. If your code is fairly portable, this will solve most of the issues on all platforms. The primary platform selected should be the platform most of your customers use. However, there will be a set of issues that are unique to secondary platforms, hopefully a much smaller set than the issues uncovered during the primary platform testing. For this reason, secondary platform testing should not be ignored. Secondary platform testing should be treated like performance testing (see article 'The use of a third party lab for performance testing' above). It should be done toward the end of the development lifecycle and can even be handed off to a third party company or team to run in an external lab. Portability testing, much like performance testing, is not closely tied to core development/QA, and a good porting team can not only do the testing but also make specific recommendations to the product team on how to fix porting issues.

The best strategy for cross platform testing is to identify a primary platform and focus most of the testing on that platform.

Summary

One of the biggest concerns for any business and IT organization is uptime. Application performance can impact revenues, customer satisfaction, employee productivity, data center efficiency and software and hardware licensing costs. There are several ways to manage application performance including performance testing, benchmarking, capacity planning and performance monitoring. These services are available both on-premise and as hosted service offerings and can help you ensure the health and the performance of your enterprise applications both before you launch and while in production. Organizations of all sizes should consider these services when application uptime is critical to business productivity.

About Perforic

Perforic provides flexible, high quality and cost-effective enterprise application porting and performance testing services for software product companies and internal development organizations. Our team consists of seasoned veterans with vast experience in performance and porting issues. We have worked with companies of all sizes – from startups to Fortune 500 companies - and have deep expertise in J2EE, .NET and database technologies in highly scalable, three-tier enterprise application environments. The company is based in Cambridge, MA and has 24 x 7 operations with locations and resources in the US and Asia. For more information, please visit http://www.perforic.com/index.html.

PERFORIC
performance is the key

Perforic LLC
955 Massachusetts Avenue, #309
Cambridge, MA 02139
info@perforic.com