Performance Testing and Modeling for New Analytic Applications

Boris Zibitsker, PhD
CEO BEZNext

Alex Podelko, PhD
Consulting Member of Technical Staff at Oracle

ABSTRACT

Traditional load testing (optimized for the waterfall software development process) was mostly focused on pre-production realistic tests. Drastic changes in the industry in recent years – agile development and cloud computing probably the most - opened new opportunities for performance testing. Instead of the single way of doing performance testing, we now have a full spectrum of different tests which can be done at different moments – so deciding what and when to test has become a very non-trivial task heavily depending on the context.

Due to increased sophistication and scale of systems, in most situations full-scale realistic performance testing is not viable anymore. In many cases we may have different partial performance test results. So, results analysis and interpretation have become more challenging – and may require modeling to make meaningful conclusions about performance of the whole system.
Performance testing provides response times and resource utilization for specific workloads. Together with knowledge about architecture and environments, it allows creation of a model to predict a system's performance (to be verified by larger-scale performance tests if necessary). This is a proactive approach to mitigating performance risks – but it requires significant skills and investments to be implemented properly. So, for existing systems it is often complemented (or even completely replaced) by reactive approaches of observing the production system. However, this does not work for new systems. If you are creating a new system, proactive methods such as early performance testing and modeling are needed to make sure that the system will perform as expected.

Modeling becomes important during the design stage because we need to investigate performance and cost consequences of different design decisions. In this case, production data are not available and waiting until the system is fully developed and deployed is too risky for any non-trivial system.

Big data systems are one of the best examples of performance risk mitigated by a combination of performance testing and modeling. The enormous size of the system makes creating a full-scale prototype almost impossible. However associated performance risks are very high – implementing a wrong design may be not fixable and can lead to a complete re-design from scratch. So, building a model to predict the system's cost and performance based on early/partial prototype performance test results and knowledge about architectures and environments becomes the main way to mitigate associated risks. A few examples of such models will be discussed in this paper.

Early load testing provides valuable information, but does not give any insight as to how the new application will perform in a production environment with large number of concurrent users accessing large volumes of data. It does not answer how implementing the new application will affect the performance of existing applications and how to change the workload management parameters affecting priorities, concurrency and resource allocation to meet business Service Level Goals. It does not answer whether the production environment has enough capacity to support expected workload growth and increase in volume of data.

All Rights Reserved

Should new application be part of Data Warehouse or Big Data environment? Should new application use Cloud platform? What is the best Cloud platform for new applications?

In this paper, we will review the value and limitations of available Load Testing tools and discuss how modeling and optimization technology can expand results of Load Testing. We will review a use case based on using BEZNext Performance Assurance software. We will cover data collection and workload characterization in small test and large production environments. We will review results of anomaly and root cause detection and seasonality determination. We will demonstrate how modeling and optimization are used to predict the impact of new application implementation, find the appropriate platform, develop proactive recommendations and set realistic expectations. This approach reduces the risk of performance surprises and enables automatic results verification.
.

KEYWORDS

Load Testing, Performance Testing, Application Performance, Cloud Platform, Big Data, Data Warehouse, Service Level Goals, Workload Characterization, DevOps, Modeling, Optimization.

Traditional Load Testing

Traditional performance testing, optimized for the waterfall development process, was mostly focused on the ability of systems to handle peak load. Usually traditional performance testing was conducted as:

- Last moment before deployment
- Last step in the waterfall process
- Extensive tools requiring special skills
- Protocol level record-and-playback
- Lab environment
- Scale-down environment
- Checking against given requirements/SLAs

There is always the risk of system crash or experiencing performance issues under heavy load – and the only way to mitigate it is to actually test the system. Even stellar performance in production and a highly scalable architecture don't guarantee that the system won't crash under a slightly higher load.

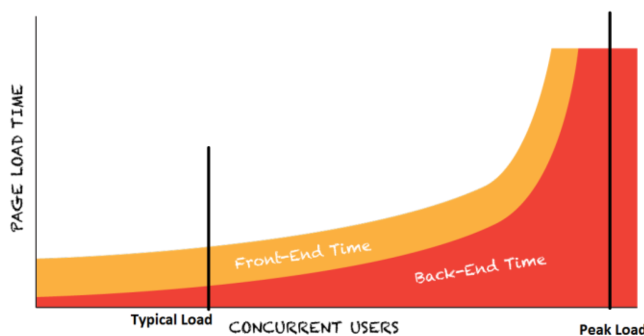

Figure 1. Typical response time curve.

A typical response time curve is shown on fig.1, adapted from Andy Hawkes' post discussing the topic [HAWK13]. As it can be seen, a relatively small increase in load near the curve knee may kill the system – so the system would be unresponsive (or crash) under the peak load.

Traditional performance testing as a part of waterfall development process did mitigate that particular risk if was done properly – however the feedback was usually provided in the very end, when the system was ready to be released. If serious issues were discovered, the cost of fixing them and deployment delays were very high.

Meanwhile the cost of fixing performance and scalability issues early may be many times lower. The idea that we need to test early, as the cost of fixing defects skyrocket later in the development lifecycle, may be traced at least to [BOEH76] and further developed in his book [BOEH81]. See fig. 2 from [HICK18] as an attempt to quantify the dependency.

All Rights Reserved

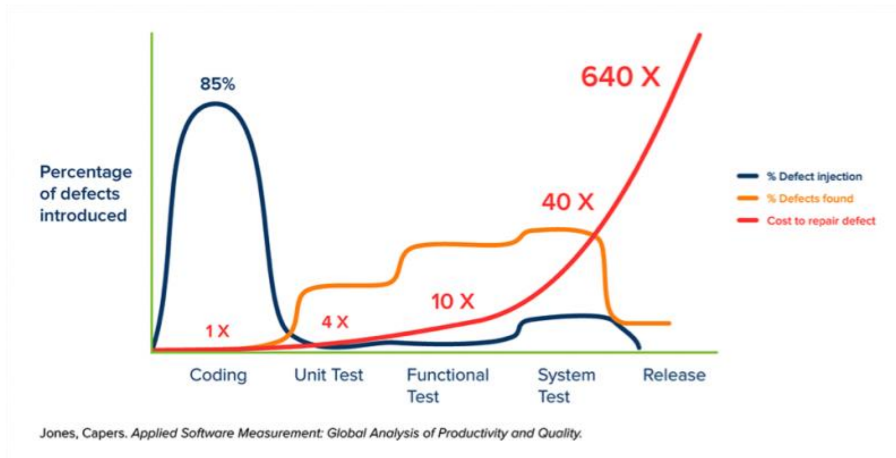

Figure 2. Cost of fixing defects.

However, although it was a common wisdom, not much could be done early as not much was available to test during waterfall development until the very end. Of course, we could define performance requirements and do architecture analysis (fully developed, for example, in [Smith90]). But, without testing, we had very limited information input from the system – until the very late moment when the cost of fixing found defects was very high.

Today's Performance Testing

Agile / iterative development and DevOps provided an opportunity to start performance work early as we are supposed to get a working system (or at least some of its components) on each iteration [PODE16, PODE19]. So, finally, it is possible to get performance feedback from the system from the first development iterations – so architecture could be verified early and defects found as soon as they appear. That alone drastically increases the value of performance testing. Now performance testing becomes an integral part of DevOps cycle as shown on fig. 3.

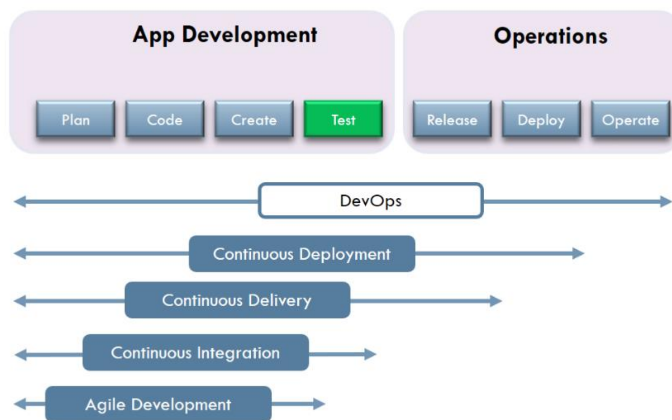

Figure 3. Performance Testing is an Integral Part of DevOps Process

However, it is not viable in most cases to run full-scale performance tests inside Continuous Integration. There are no generic guidelines anymore – what and how should be tested depends on specific context. Instead of the well-defined traditional way of performance testing, we have a continuum of options along multiple dimensions.

Scope. While a full-scale system test still remains an option, performance tests could be run on any level – unit, component, service, subsystem - and with any intensity and kind of load. In many cases load should be selected explicitly to load a specific part of the system deployed in a particular way.

All Rights Reserved

Environment. options nowadays include traditional internal (and external) labs; cloud as 'Infrastructure as a Service' (IaaS), when some parts of the system or everything are deployed there; and service, cloud as 'Software as a Service (SaaS)', when vendors provide load testing service. There are some advantages and disadvantage of each model. Depending on specific goals and systems to test, one deployment model may be preferred over another.

For example, to check the effect of performance improvement (performance optimization), using an isolated lab environment may be a better option to see even small variations introduced by a change. To test the whole production environment end-to-end to make sure that the system will handle load without any major issue, testing from the cloud or a service may be more appropriate. To create a production-like test environment without going bankrupt, moving everything to the cloud for periodical performance testing may be a solution. For comprehensive performance testing, you probably need to use several approaches – for example, lab testing (for continuous performance testing and performance optimization to get reproducible results) and distributed, realistic outside testing (to check real-life issues that can't simulate in the lab). Limiting yourself to one approach limits the risks you will mitigate.

Testing Approach. Instead of a traditional testing approach of full-scale realistic workload to simulate the production system, we have a whole dimension of options from early exploratory / agile performance testing (somewhat corresponding to new systems which we don't know much about) to automated / regression testing (somewhat corresponding to well-known systems where only small enhancements get implemented) – and the traditional approach may be depicted as a dot on that dimension (see fig.4 to illustrate that idea).

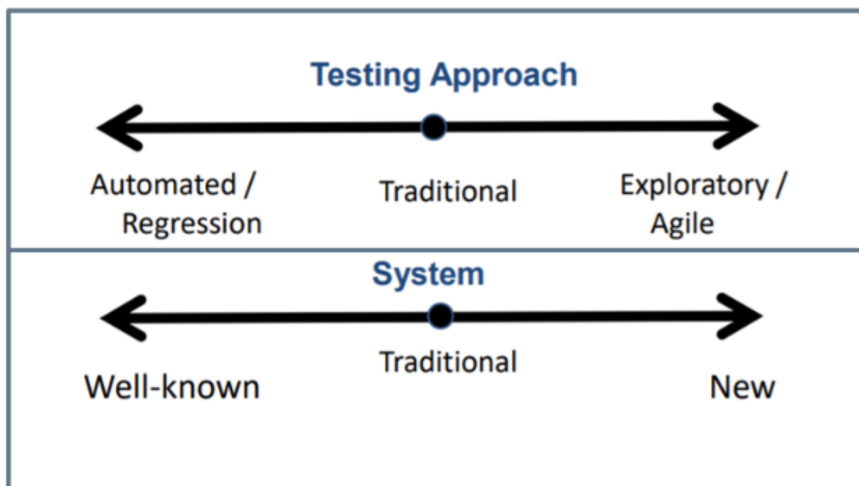

Figure 4. Testing approach continuum of options

Load Generation. Quite often the whole area of load testing is reduced to pre-production testing using protocol-level recording/playback. While protocol-level recording/playback was (and still is) the mainstream approach to testing applications, it is definitely just one type of load testing using only one type of load generation.

The time when all communication between client and server was using simple HTTP is in the past and the trend is to provide more and more sophisticated interfaces and protocols. While load generation is a rather technical issue, it is the basis for load testing – you can't proceed until you figure out a way to generate load. As a technical issue, it depends heavily on the tools and functionality supported.

There are three main approaches to workload generation [PODE14] and every tool may be evaluated on which of them it supports and how.

Protocol-level recording/playback. This is the mainstream approach to load testing: recording communication between two tiers of the system and playing back the automatically created script (usually, of course, after proper correlation and parameterization). As far as no client-side activities are involved, it allows the simulation of a large number of users. Such a tool can only be used if it supports the specific protocol used for communication between two tiers of the system.

All Rights Reserved

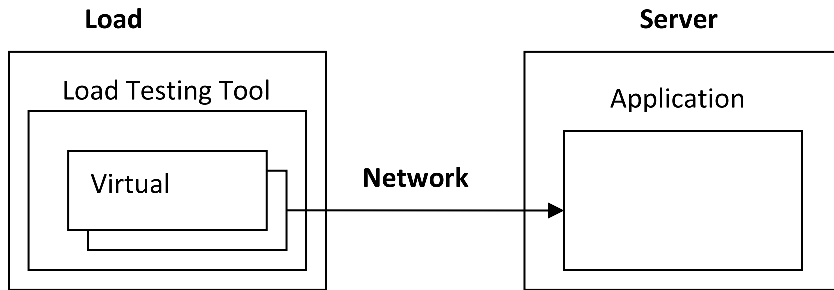

Figure 5. Record and playback approach, protocol level

UI-level recording/playback. This option has been available for a long time, but it is much more viable now. In the past, a separate machine was needed for each virtual user (or at least a separate terminal session). This drastically limited the load level that could be achieved.

New UI-level tools for browsers, such as Selenium, have extended the possibilities of the UI-level approach, allowing running of multiple browsers per machine (limiting scalability only to the resources available to run browsers). Moreover, UI-less browsers, such as HtmlUnit or PhantomJS, require significantly fewer resources than real browsers.

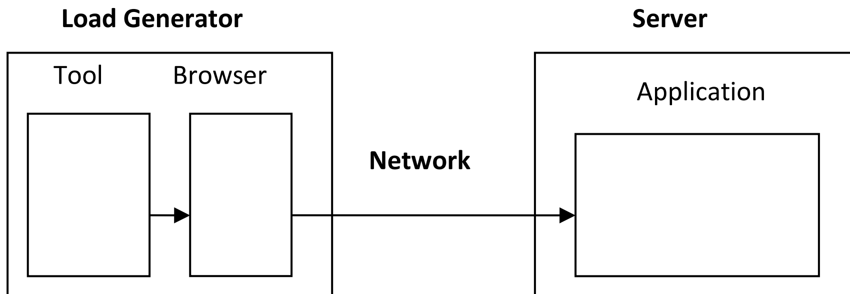

Figure 6. Record and playback approach, browser users

Programming. There are cases when recording can't be used at all, or when it can, but with great difficulty. In such cases, API calls from the script may be an option. Often it is the only option for component performance testing. Other variations of this approach are web services scripting or use of unit testing scripts for load testing. And, of course, there is a need to sequence and parameterize your API calls to represent a meaningful workload. The script is created in whatever way is appropriate and then either a test harness is created or a load testing tool is used to execute scripts, coordinate their executions, and report and analyze results.

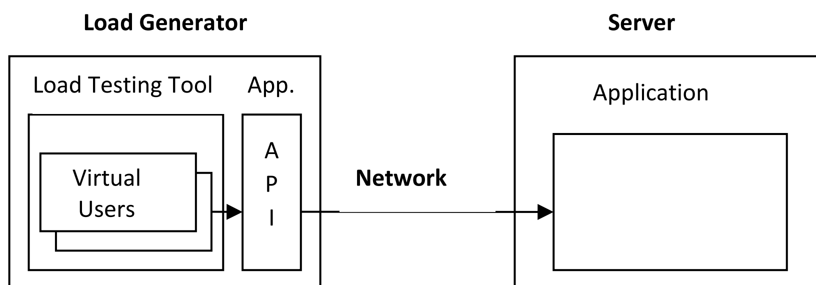

Figure 7. Programming API using a Load Testing Tool.

To do this, the tool should have the ability to add code to (or invoke code from) your script. And, of course, if the tool's language is different from the language of your API, you would need to figure out a way to plumb them. The importance of API programming increases in agile / DevOps environments as tests are run often during the development process. In many cases APIs are more stable than GUI or protocol communication – and even if something changed, the changes usually can be localized and fixed – while GUI- or protocol-based scripts often need to be re-created.

All Rights Reserved

Load Testing Tools. There are quite a few load testing tools [PODE14, LONN20]. While most load testing tools look similar at first glance, they are actually quite different in supporting different performance testing options discussed above. And, unfortunately, generic descriptions (for example, from the vendor website) are usually useless in understanding the differences. Each situation *is* different. A tool may be very good in one situation and completely useless in another. The value of the tool is not absolute; rather it is relative to a specific situation.

To list some examples, Microfocus LoadRunner family, Microfocus Silk Performer, Neotys NeoLoad, IBM Rational Performance Tester, RadView WebLoad, and SmartBear LoadNinja may be mentioned among commercial tools. Apache JMeter, Gatling, k6, and Locust may be mentioned among open source tools. Broadcom BlazeMeter, Tricentis Flood.io, RedLine13, and Octoperf are examples of commercial extensions of open source tools.

Testing Strategy. Seeing that continuum of performance testing options along different dimensions, it is obvious the testing strategies became very non-trivial – as a specific set of tests and their timing is defined by specific context. "Automation" is only one part of it – continuous performance testing is very important for iterative development, but it is just part of performance testing strategy addressing a specific risk – performance regression between builds. Moreover, performance testing should be considered as part of a larger performance engineering strategy [PODE18].

Asking the right questions may help to formulate a proper testing strategy. For example, such questions could be:

- What are performance risks we want to mitigate?
- What part of these risks should be mitigated by performance testing?
- Which performance tests will mitigate the risk?
- When we should run them?
- What process/environment/approach/tools will we need in our context to implement them?

Value and Limitations of Performance Testing

Performance testing provide an immense value as a proactive way to mitigate performance risk. Early problem detection prevents costly redesigns and delays. Considering the flexibility of today's performance testing the strategy may be optimized for specific context to provide the best return on investments.

Moreover, early / continuous performance testing provides a constant stream of real performance-related information. Even for existing systems, it provides important input to see a possible performance impact on production systems. But it becomes really invaluable for new systems as only early performance feedback enables developers to identify and fix performance issues before new application deployment.

However, it is important to understand the limitation of performance testing. It is expensive on a high-scale level, so the number of large-scale tests that can be run is limited. Smaller-scale tests provide very important, but partial information – which by themselves don't provide a holistic view.

Modeling complements performance testing here, enabling a big picture view and answering what-if questions from disjointed performance testing results. It is invaluable for evaluating options and developing proactive recommendations for the system's architecture and design.

Role of Modeling During DevOps Process

Value of modeling results for Application Developers

- **Predict new applications implementation impact**
  - Predict how new application will perform in production environment
  - Identify anomalies and their root causes during testing of new applications
  - Develop recommendations to application developers
- **Predict how new application will affect existing production applications**
  - Predict how implementation of new applications will affect Response Time and Throughput of existing applications
  - Develop capacity planning recommendations
  - Set up realistic expectations

All Rights Reserved

Value of modeling for Operations

- **Develop Proactive Performance Management and Workload Management Recommendations**
    - Compare performance measurement results after implementation of the new application with expected
    - Develop proactive performance tuning recommendations
    - Develop proactive workload management recommendations
    - Reevaluate Capacity Planning recommendations

Modeling plays the central role in Performance Assurance

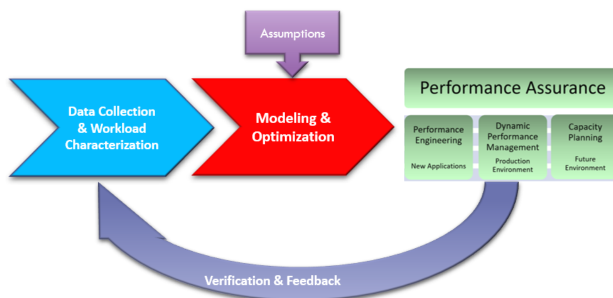

Figure 8. Modeling and Optimization are a foundation of Performance Assurance, which includes Performance Engineering for new applications, Dynamic Performance Management after deployment of the new applications in production and Capacity Planning supporting expected growth and selection of the appropriate platform for new application.

Modeling results predict the impact of the different measures on performance for each workload

Workload and volume of data growth affect workloads' queueing and software delay time
Response time of new application in production has:

- Different Response Time, Service Time, Queueing time and Delay Time for new application
- Response time of production workloads and it's Queuing Time and Delay Time are changed
- Move workloads to the Cloud can affect Service Time, Queueing Time and Delay Time for all workloads
- Change of the Workload Management Rules (Priorities, Concurrency and Resource Allocation) affect the Queueing time and Delay Time of each workload

Figure 9. Modeling results predict how different changes will affect the major components of the Response Time, Throughput and resource utilization by each workload

We use an optimization engine to run models iteratively to find the optimum workload management parameters and additional resources which will be required to meet Service Level Goals (SLGs) for each of the growing and changing workloads.

Let's review 10 steps of using measurement results of performance testing during DevOps to build models and predict new application impact and develop proactive recommendations about what should be done to meet SLGs for new and existing production applications with lowest cost.

All Rights Reserved

The first step in using modeling is data collection during performance tests in typically small test environments and data collection for production workloads in a large production environment.
We use OS agents to collect data about each process and DBMS agents to collect data from systems tables. Measurement data are aggregated and transformed into common format, which includes the following data types used to build models

- Hardware and Software Configuration
- Information by each node/server by user and application, including
- Response Time
- Throughput
- CPU Utilization and CPU Service Time per request
- Disk Utilization, I/O rate, #I/O operations per request and KB/Request, Channel Utilization
- Memory utilization
- Network utilization
- Level of concurrency

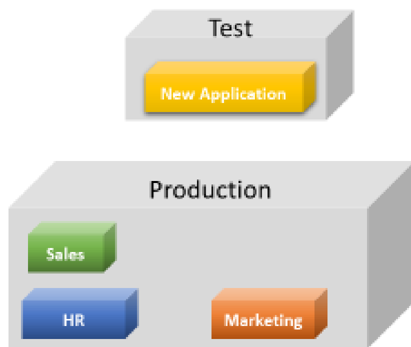

Figure 10. Data Collection in Test and Production Environments

The Second Step is workload characterization of test and production environments.
Each workload represents the activity of a group of users using a set of applications supporting a specific Line of Business. Workload Aggregation process aggregates measurement data by Module of the New Application or by Line of Business / Workload, Aggregation use rules describing user names and program names which belong to the specific module or line of business.

Workload characterization is an automated process performed hourly. Each workload is characterized by performance, usage of resources and patterns of accessing data. Example of the workload characterization results showing CPU utilization and number of I/O operations by each workload during 24 hours is shown on Figure 11.

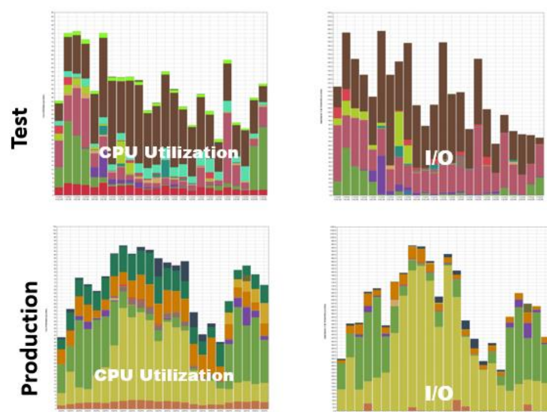

Figure 11. Workload Characterization in Test and Production Environments

Results of the workload characterizations are used for Anomaly and Root cause determination and as input for a model.

All Rights Reserved

The third step is anomaly detection

Measurement data collected during performance tests after each build are used to detect the anomalies and their root causes. Information about the most severe anomalies and root causes determined in test and production environments are passed to application developers and operations (Figure 12.)
We assume that the most severe anomalies will be fixed prior to deployment of the new application in production.

Figure 12. Detection of the workloads with highest frequency of Anomalies and Programs and Users causing the most severe problems during DevOps provide immediate information to Application developers and narrows down the scope of tuning efforts

Fourth Step is Workload Forecasting for New and existing Production Applications

Workload Forecasting for s new application is typically based on Business Plan and for existing Production workloads based on analysis of the historical data.

Expected Workload and Volume of Data Growth

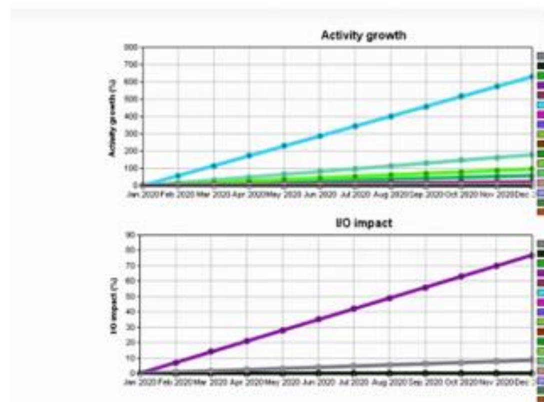

Figure 13. Workload Forecasting shows expected increase in number of transactions and volume of data accessed by workload

Fifth Step is predicting impact of the expected increase in number of users and volume of data on performance of existing workloads in Production environment.

Predicted results show when existing production workloads will not meet SLGs on current production environment even without the new application impact. [ZIB1]

All Rights Reserved

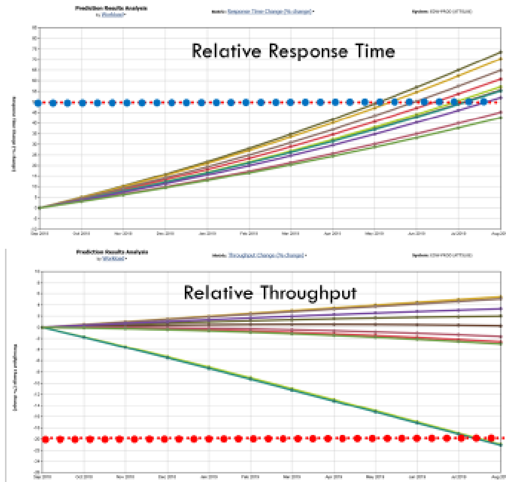

Figure 14. As a result of the workload growth and increase in Volume of Data SLGs will not be met

Sixth Step is Predicting the New Application Implementation Impact

Performance prediction results showing expected Performance (response time, throughput), Resource utilization (CPU utilization, Disk utilization, Network utilization, Memory utilization) after deployment of new application.

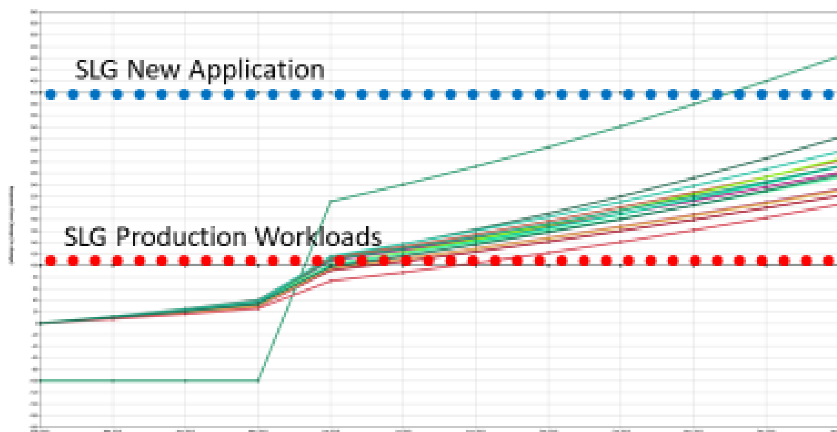

Figure 15. After deployment of new application, existing production workload will not be able to meet SLGs and new application will have performance problems starting in November. So, can tuning and workload management optimization avoid an expensive upgrade?

Seventh Step is Predicting Impact of the Workload Management Optimization

Workload management rules of changing priorities, limitation of concurrency and allocation of resources for critical workloads allow operations to influence reallocation of resources between workloads to meet SLGs without adding new instances.

BEZNext modeling and optimization technology [ZIB2] automatically evaluates different options and recommends rules for different time of day, and sets realistic expectations of response time, throughput and resource utilization for each workload.

Figure 16 shows that changes of workload management rules will not be sufficient to meet SLGs and additional resources will be required in 6 months.

All Rights Reserved

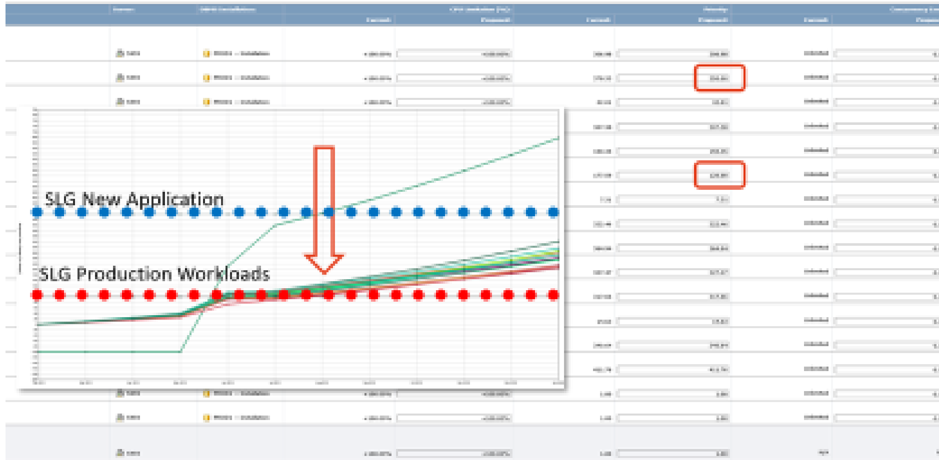

Figure 16. Change of the workload management rules affecting priorities between all workloads will not be sufficient to meet SLGs.

Eighth Step is Predicting the Minimum On Prem Upgrade Required to meet SLGs after Deployment of new Application

According to the performance prediction results [ZIB3] as we can see from Figure 17 at least 4 additional nodes will be required in 6 months to meet SLGs through the end of the year. It is pretty expensive and an alternative platform can be evaluated.

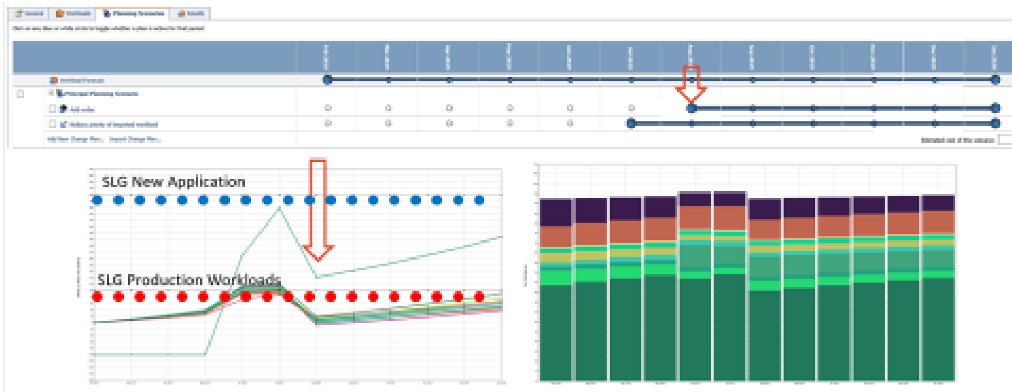

Figure 17. Predicting when and how much additional resources will be required to meet SLGs for existing and new workloads. According to the model, an additional 4 nodes will be required in 6 months On Prem to meet SLGs through the end of the year.

Ninth Step is Determining Appropriate Cloud Platform for New Application

BEZNext Modeling and Optimization use measurement data characterizing new application performance, resource utilization and hardware and software configuration of the test environment to predict how new application will perform on different Cloud platforms.

It takes into consideration the expected workload and volume of data growth and predicts the minim number of instances and instance types which will be required to meet SLGs for new workload. It also predicts the impact of moving a new workload to one of the existing Clouds [ZIB1].

Predicted information about the number and type of instances which will be required during different hours of the day and different months of the year is used to predict the cost of supporting a new workload in different Cloud environments.

All Rights Reserved

Figure 18. Applying Modeling and Optimization to select appropriate Cloud Platform

Tenth Step is Automatic Results Verification

Automatic comparison of the actual measurement data with expected / predicted results identifies new anomalies and enables creation of the continuous Performance Assurance Process.

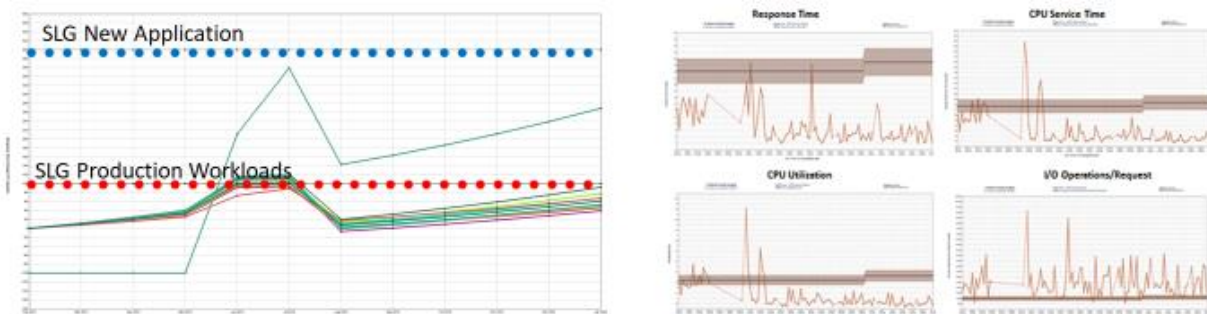

Figure 19. Verification by comparing actual results with expected

Summary

- Performance testing is the main source of performance measurement data during development process
- Performance measurement data is needed to create and validate models predicting new applications performance
- Modeling complements performance testing allows fast and inexpensive analysis of what-if scenarios
- Modeling results provide value to Application Developers and Operations during DevOps process
- Combination of Performance Testing and Modeling is a way to mitigate performance risks early and avoid performance surprises.

References

[LONN20] Lönn, R. *Open source load testing tool review 2020*. https://k6.io/blog/comparing-best-open-source-load-testing-tools
[HAWK13] Andy Hawkes, A. *When 80/20 Becomes 20/80*.http://www.speedawarenessmonth.com/when-8020-becomes-2080/
[LOAD14] *Load Testing at the Speed of Agile*. Neotys White Paper, 2014.
http://www.neotys.com/documents/whitepapers/whitepaper_agile_load_testing_en.pdf
[PODE19] Podelko, A. *Context-Driven Performance Testing, CMG imPACt, , 2019.*
[PODE18] Podelko, A. *Context-Driven Performance Engineering, Performance Calendar, , 2018.*
[PODE16] Podelko, A. *Reinventing Performance Testing, CMG imPACt, , 2016.*
[PODE14] Podelko, A. *Performance and Capacity Conference by CMG, 2014*
[ZIB1] B. Zibitsker, A. Lupersolsky, *"How to Apply Modeling to Compare Options and Select the Appropriate Cloud Platform"*, ICPE 2020, Canada
[ZIB2] B. Zibitsker, IEEE Conference, Delft, Netherlands, March 2016, *Big Data Performance Assurance*
[ZIB3] B. Zibitsker, *Proactive Performance Management for Data Warehouses with Mixed Workload,* Teradata Partners, 2008, 2009
[HICK18] Hicken, *A. The Shift-Left Approach to Software Testing,* 2018 https://www.stickyminds.com/article/shift-left-approach-software-testing
[SMIT90] Smith, C. *Performance Engineering of Software Testing*, 1990.
[BOEH76] Boehm, B. *Software Engineering*, IEEE Trans. Computers,1976.
[BOEH81] Boehm, B. *Software Engineering Economics*, 1981.

All Rights Reserved