Executive Summary

In the last decade, Performance Testing (PT) has become commonplace in IT enterprises, thanks to greater availability of skills in the market. Most IT vendors offer PT as a service and follow standard processes such as scripting, execution, and analysis. Given the current state, many think PT has reached a point of maturity.

In reality, although various PT cycles are employed, performance issues persist. Given the rapid increase in applications hosted by data centers and the increasing adoption of service oriented architectures (SOA), it is virtually impossible to performance test every application or service in the enterprise. However, outside enterprise application complexity, fundamental issues persist in PT that are not addressed by the industry.

If you encounter challenges in PT, and are looking for solutions to resolve these issues you will find this whitepaper helpful. You will explore several fundamental aspects of PT that are ignored by service providers and tool vendors. In this paper we cover PT challenges such as:

- Result validation
- Run duration
- Distribution of think times
- Workload prediction
- Necessity of single user tests
- Bridging of environmental gaps
- Extrapolation of results
- Component PT
- PT of interfaces
- And much more…

You will see how simple steps can help to get the basics right for PT. We also make a case for higher quality of PT and shorter turnaround times, by means of investing upfront in single user tests. Testers, PT Managers, and PT Architects, will benefit from the best practices discussed.
Introduction

Many commercial PT tools have swamped the market, today. Does this mean:

- CIOs are not facing performance and capacity issues after applications go live
- PT has reached the state of maturity, where one can be confident about performance if PT results are acceptable
- PT has collapsed to the level where it is confined to a checkbox—once it is checked as complete, nobody will bother about performance

Latest advances in PT tools and increased affordability of tools and services are encouraging; however, PT is still far from maturity. Many basic aspects of testing are hidden under the rug; in the name of 'process maturity,' a lot of extra services are rendered, with no benefits. At the crux, the so-called assurance provided by PT is limited to a sense of hope and conviction within the PT team, and might not be substantiated by facts.

Performance Testing-A Sneak Peek

Various enterprise and open source tools have revolutionized PT and have triggered interest across the IT community. Most of these tools support web protocols, while a few support proprietary protocols such as, OracleNCA and binary protocols over TCP/IP.

An application that supports multiple concurrent users should to be tested to assure, it can handle workload, before it is released to production. Mature IT enterprises plan well in advance to ensure, their applications undergo multiple rounds of PT before they go live. By default PT refers to load testing, where the workload of multiple users is simulated by means of a load testing tool. In a broader sense PT also needs to cover volume testing, which means testing the performance of an application under realistic data volume and batch testing.

Load testing tools allow testers to script scenarios that pertain to real-life use cases. The real-life user is replaced by a virtual user, which is actually an operating system thread that executes the tester's scripts. APT tool can spawn as many virtual users as desired to simulate real-life workload, as showed in Figure 1. PT tools provide controls for think time and inter-iteration time to mimic the delays introduced from user interaction and between one transaction and the next. The tester inputs number of users, think time, and inter-iteration time. The output of the performance test is captured in terms of user response time and throughput that is the number of user requests completed per second.

PT tools allow testers to specify ramp up of virtual users, as well as the overall test duration. Monitoring tools are used to view the progress of the test. Once the test is complete, PT tools output statistical quantities such as throughput, average, percentile, and standard deviation of response time.

In a broader sense PT also needs to cover volume testing, which means testing the performance of an application under realistic data volume and batch testing.
The PT process consists of three phases—scripting, execution, and analysis. Based on analysis and monitoring, bottlenecks are determined and the platform or application is tuned accordingly. If required, the tests are then re-run after scripting changes.

Fundamentals Revisited

We now focus on the fundamentals of PT and revisit the basics—result validation, run duration, think time, and results—for single user testing.

Result Validation

In case of PT, the system behavior is unknown to begin with. Test results should be validated, but it is rarely done in the IT industry. The more mature vendors will, at the most, verify if the test was completed, for example, running some database queries to check whether insertions have occurred.

Validation of performance test output is not as difficult as it sounds. A very simple way to cross check the accuracy of PT output is to validate against a manual test, concurrent with the PT tool. Another simple way is to validate the throughput reported by the PT tool against the throughput derived from the web server log. A scientific approach to validation is to cross check whether the inputs and outputs are balanced. We can easily use Little’s Law to check for the balance:

$$\text{Number of concurrent users} = \text{Throughput} \times \text{Average Cycle Time}$$

Where,

$$\text{Average Cycle Time} = \text{Average Response Time} + \text{Average Think Time} + \text{Average Inter-iteration Time}$$

Figure 1: Performance Testing Background
Test Duration

Statistically, if you compute the mean or standard deviation of any random variable such as response time, you need a sufficient number of samples to obtain an accurate estimate. Running a test for too small a duration will provide completely meaningless estimate, whereas running it for too long will lead to reduced PT cycles.

PT tools leave the choice of run duration to the tester. We have seen instances where regular performance tests (and not availability test) are run for durations ranging from 30 seconds to 2 hours or more. So what is the ideal test duration? Experienced performance testers usually observe a running graph of throughput, or mean response time to visually check for convergence. The 'transient' phase of the test is ignored, and only the steady state phase is used to compute throughput and response time statistics.

Now, consider a scenario where PT is performed in the cloud and the service provider handles hundreds of applications per day with a limited number of experienced people. Visually determining the test duration will be a significant bottleneck. How about the test determining its own duration? It is possible if the tester uses confidence intervals.

A confidence interval simply tells you that your estimate belongs to an interval with a certain level of confidence or probability. For example, a 95% confidence interval of (2.25, 2.75) states that your mean response time of 2.5 lies in this interval with a confidence of 95%. Statistical theory is available to automatically compute these intervals. You can use this effectively to determine test convergence, for example, you could let the test run until the 95% confidence interval becomes narrow enough, such that it is within 5% of the computed mean.

Think Time

While reviewing a number of PT projects we observed that testers use a default value of think time and inter-iteration time, which is typically 5 seconds or less. But real-life businesses reveal a different picture.

For example, banking transactions often take a minute or two to complete their entire cycle even if there are just a couple of screens. Therefore, the overall think time per screen would range from 30 seconds to over a minute, if we assume that response time is a couple of seconds! Likewise, inter-iteration time could run into several minutes in real life or a few seconds, depending on the business at hand. Again this highlights that there is no default setting for think time or inter-iteration time. What matters to a business is the overall cycle time of a transaction.

Performance testers also tend to ignore the variability in think time or inter-iteration time. The conventional wisdom across the PT industry is that randomizing think time equates to sampling from a uniform distribution, where every sample is equally likely. But there will be instances where a tester would step out for a break or there may be a large variation in the way users view output. These instances lead to variation in think time, which cannot be captured by a uniform distribution. There is reported evidence showing, varying performance with variation in think time (See reference [4] for more details).
Single User Testing

Perhaps the most fundamental flaw in PT is the optimistic assumption that the first load test will be successful. We have reviewed many projects at Persistent, which reveal that in the first round of PT, the single user response time is way above the multi-user response time target. The good news is that it is very easy to get a detailed profile or trace of a single user's response time and therefore easy to determine bottlenecks. The bad news is that if you are still optimistic that your multi-user tests will yield acceptable performance right from the first run, you could be in for a significant project overrun.

It may seem like extra work, to run a single user test but the detailed output that you get and the ease of running tests even without automation make a case in point. We have seen PT for several large projects run into months and months without getting anywhere, and the only resort was to stop load testing and stick to single-user tests. Within a week all bottlenecks were ironed out and PT progressed as scheduled. This does not mean that single-user performance testing is enough, it simply says, not to ignore the basics.

One can even create advanced models out of single user tests by noting visits per resource or component for each business transaction, and also by noting the service time spent per visit. The bottleneck resource or component is the one with the highest demand, which means the number of visits multiplied by the average service time per visit. If these bottlenecks are determined during single-user tests, a lot of performance bottlenecks can be fixed by developers, way before PT is done for a large number of concurrent users.

Beyond the Basics

PT is expected to rise up to higher challenges and not just be confined to a checklist. In the following sections we discuss how result extrapolation, environmental gaps, reliability testing, and other such techniques can elevate PT to meet these challenges.

Result Extrapolation

Application owners, CIOs, and business leads are keen to know what would happen if workload changed, over time. For example, what if the number of concurrent users doubled from 2500 to 5000? What if additional capacity was procured for the same workload?

Current levels of PT maturity are restricted to test scripting, execution, and analysis of the test. There are a few tools such as OPNET and Hyperformix that take the input of load tests and extrapolate the results for higher workloads. However, these tools are not commonly used in the industry. Simple extrapolation techniques are also not easily available.

There are, however, cases of simple extrapolation that have proved to be very useful, especially when dealing with systems under saturation. In other words, tests can be run with less number of concurrent users and close-to-zero think times to saturate the system, as soon as possible, and determine the limiting throughput. This limiting throughput can then yield useful bounds on system performance with a higher number of concurrent users (See reference [4] for more details).
Environmental Gaps

One of the most common flaws in PT is that it is done on a LAN, where network response time is not noticeable; while the production usage is on a world-wide WAN, where the effect of bandwidth and latency on response time can be alarming. Surprisingly, no major effort is being taken on part of service vendors to bridge this gap.

WAN emulators seamlessly plug into the PT environment and one can manipulate network properties to reproduce a WAN within the LAN. Testers can use WAN emulators such as Shunra, Netem, WANem, NIST Net, and Linktropy for this purpose. In this way a lot of screen designs can be optimized for high performance over the WAN. If one does not have a WAN emulator it is quite easy to get a trace of application http round trips using tools such as, httpwatch and fiddler and optimize the application for operation in WAN.

Another major environmental gap is database volume—they are significantly small in magnitude in PT than in production. This aspect of volume testing has to be planned meticulously, especially when production data cannot be replicated in PT for privacy reasons. One can nevertheless manipulate database statistics to obtain query costs, as they would appear in production and get an estimate of poor performing queries during production. This aspect of volume testing is not commonplace in the IT industry and database volume emulators are not available in the market.

Reliability Testing

It is common for 24x7 businesses to run at least one round of long duration test, where a number of concurrent users continue to inject load on the system, over time. This stresses the system's availability and determines if memory leaks and other such defects will bring the system down, over time. Given the limited time available for such tests it is not always possible to fix all reliability flaws before go-live. But it pays to categorize the type of recovery that is possible. In general there are three types of system collapses:

- Recoverable via simple restart (of server, JVM, or component)
- Recoverable via parameter change (and restart)
- Non-recoverable

The first scenario is a classic case of software aging, where a defect such as a memory leak causes resources to be consumed but not released over time. It is easy to detect the problem but not always trivial to determine the root cause and fix it. One can effectively use PT to model the rate of leakage and determine if the number of restarts in a day are well within allowable limits to go live (with the hope that over a period of time these defects will get resolved).

The second case is encountered when some resource is saturated, such as a file directory, and needs to be cleaned or resized by changing some parameters. Again, it may not always be possible to run a test for a duration long enough to hit the wall, but abnormal rates of consumption can always be tracked during PT to build a simple predictive model.

The last case is the most difficult to test. For example, when the entire file system collapses beyond a number of files and reaches a point of non-recovery it falls under this category. Determining the limits of all components may be an impossible task during PT and one may need to run these tests and build models for very specific cases such as, government document repositories.
Component PT

There is a mix and match of custom-built and off-the-shelf components in the market, when it comes to IT applications. Therefore, it comes as a surprise that one has to wait for integration testing prior to the first round of PT and determine if any of the components pose a significant bottleneck.

One does not have to be a rocket scientist to convert end-user performance targets for throughput and response time into component-level performance targets. Essentially, one first arrives at the throughput target by estimating the number of visits to a given component vis-à-vis a given business transaction. Typically, the following should be true:

\[
\text{Avg. demand per resource} < \frac{\text{Avg. cycle time per business transaction}}{\text{number of users}}
\]

Once visit counts per component are known, we can derive the average service time permitted per resource for the component.

Once performance targets are estimated per component, individual components can be tested by PT tools or through custom scripts. If there are hundreds of components to worry about, then it makes sense to choose the top few that are either accessed the most or are likely to take up the maximum time in the business processing cycle.

Interface PT

Today applications have a number of interfaces, to both internal and external services. For example, a brokerage firm would need to connect to the stock exchange and an in-house application would need to connect to the in-house mainframe. While it is not always possible to have external interfaces up and running during PT, one can model the workload and create stubs for this purpose. This approach is adopted by a few industries only, as rest prefer ignorance is bliss.

Delta Testing

With software applications it is common to have a number of minor releases and a few major releases during the product lifecycle. The PT industry is far from mature to determine whether a given release needs to undergo PT or not. Usually, project leads take a call based on their gut feeling and hope for the best. In other cases, some customers take a pessimistic approach of conducting PT on every version, whether major or minor, before release to production. This is no doubt a good practice, but may not be feasible for all industries, either due to budget or time restrictions.

Based on the principles of single user performance testing, as highlighted earlier, one can build a simple model to estimate the amount of performance change caused by a new version release. This deviation can then be analyzed to determine whether PT needs to be done or not, before releasing the application to production.
A Holistic Approach to Performance Testing

Table 1 summarizes the important points as well as solutions discussed in this paper. Several of the proposed solutions can be built in to custom tools for PT or built around industry standard tools.

<table>
<thead>
<tr>
<th>PT Aspects</th>
<th>PT Tool Maturity</th>
<th>PT Process Maturity</th>
<th>Possible Solution(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Validation</td>
<td>Left to the tester</td>
<td>Weak</td>
<td>Little’s law, human cross check, validation against logs</td>
</tr>
<tr>
<td>Test Duration</td>
<td>Left to the tester</td>
<td>Ad hoc, Visual</td>
<td>Confidence intervals, experienced professional</td>
</tr>
<tr>
<td>Think Times</td>
<td>Left to the tester, limited support for multiple distributions</td>
<td>Ad hoc</td>
<td>Derive from cycle time and Little’s law, analyze impact of variability</td>
</tr>
<tr>
<td>Single User PT</td>
<td>Can be done manually too</td>
<td>Typically left out</td>
<td>Combined with detailed profiling and modeling is a must do for large applications</td>
</tr>
<tr>
<td>Extrapolation of Test Results</td>
<td>Absent in PT tools, niche tools available separately</td>
<td>Absent</td>
<td>Easy to do for systems under saturation</td>
</tr>
<tr>
<td>Environmental Gaps</td>
<td>Partially addressed for WAN, need to enrich the environment with emulators</td>
<td>Rarely addressed</td>
<td>WAN emulation tools are available, DB volume extrapolation can be done during the initial stages of testing</td>
</tr>
<tr>
<td>Reliability Testing</td>
<td>Can schedule desired load, but analysis is left to the user</td>
<td>Often ignored</td>
<td>Simple models can be built to determine recoverability points</td>
</tr>
<tr>
<td>Component PT</td>
<td>Niche tools can be used to inject load, component level targets need to be determined externally.</td>
<td>Typically absent</td>
<td>Targets can be easily derived and regular PT or custom built tools can be used well in advance, right after integration testing</td>
</tr>
<tr>
<td>Interface PT</td>
<td>Absent</td>
<td>Typically absent</td>
<td>Needs to be planned, in some cases emulators can be built</td>
</tr>
<tr>
<td>Delta Testing</td>
<td>Absent</td>
<td>Absent</td>
<td>Simple models can be built to determine whether a minor release needs to undergo PT</td>
</tr>
</tbody>
</table>

Table 1: State of the Art in PT and Possible Solutions
Conclusion

Regardless of the advances in PT and performance profiling tools, performance testing practices have not matured, both in terms of tools and in terms of processes. We have seen how simple solutions can rectify fundamental flaws in the way PT is conducted. Likewise, many challenges in PT, which are dealt with only partially, even in mature organizations, can be addressed systematically by service providers with a good knowledge base. It is important to train the next generation of performance engineers for a strong foundation and process maturity. Latest PT tools make it easy to monitor the performance of applications, as the results can be captured easily, and therefore simplify the analysis process. The insightful information pulled out can significantly help to resolve bottlenecks and improve performance.

References

About Persistent Systems

Persistent Systems (BSE & NSE: PERSISTENT) builds software that drives our customers' business; enterprises and software product companies with software at the core of their digital transformation. For more information, please visit: www.persistent.com

India
Persistent Systems Limited
Bhageerath, 402, Senapali Bapat Road
Pune 411016.
Tel: +91 (20) 6703 0000
Fax: +91 (20) 6703 0009

USA
Persistent Systems, Inc.
2055 Laurelwood Road, Suite 210
Santa Clara, CA 95054
Tel: +1 (408) 216 7010
Fax: +1 (408) 451 9177
Email: info@persistent.com

DISCLAIMER: “The trademarks or trade names mentioned in this paper are property of their respective owners and are included for reference only and do not imply a connection or relationship between Persistent Systems and these companies.”