The Pega® Platform is different than the standard Java application in several ways, but the approach to performance testing and tuning Pega is the same. To ensure that each performance test is successful, please take the following key considerations into account.

**Key considerations**

- Define Application Usage Model (AUM) and evaluate with business to ensure it’s realistic.
- Create the scenarios from the key functional areas as detailed in the AUM.
- Evaluate and validate test data requirements.
- Create test user IDs with correct roles and privileges.
- Estimate the amount of work that should come in and do not process more than what was expected. (A common mistake is to process a whole day’s volume in one hour.)
- Tune and optimize the chosen scenarios.
- Use Pega tools PAL / DB Trace to identify issues.
- Fix issues then validate fixes, i.e. new DB index and query tuning.
- Baseline scenarios on a single JVM.
- Validate performance metrics for low user concurrency (10-50).
- Increase concurrent usage to evaluate optimal user concurrency per JVM.
- Diversify test IDs. Using a single test ID to test concurrent scenarios can lead to errors.

**Test approach**

The diagram below illustrates the approach used for performance testing a Pega application.
Define success criteria

The following key performance indicators (KPIs) are applicable to any Pega application. The tested application must be performing within the bounds of these KPIs to be considered successful. The KPI values in the chart below are only examples. KPI values should be defined based on team conversations and an understanding of the application use cases.

Run the test cases before and after each planned release to see if the system is improving or worsening with new code. If necessary, fix the issues before moving new code to production.

<table>
<thead>
<tr>
<th>KPI</th>
<th>Example KPI value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average transaction response time</td>
<td>&lt;3 seconds</td>
</tr>
<tr>
<td>90th percentile transaction response time</td>
<td>&lt;10 seconds</td>
</tr>
<tr>
<td>Transaction failure rate</td>
<td>&lt;1% of transaction fail under load</td>
</tr>
<tr>
<td>CPU utilization of application server</td>
<td>&lt;CPU should not exceed 50% benchmark tests, 80% peak hour</td>
</tr>
<tr>
<td>Java memory of application server</td>
<td>&lt;70% of the available heap</td>
</tr>
<tr>
<td>Time spent garbage collecting</td>
<td>&lt;1% of total processing time</td>
</tr>
</tbody>
</table>

Test objective

The aim of the test is to measure the ability of the environment to process the selected business functions within the specified timeframes or rates of concurrency detailed in the service level agreements (SLA). The tests detailed below should be carried out during this test phase if applicable. The business or other requirements will usually determine this.
## Performance testing

<table>
<thead>
<tr>
<th><strong>Performance testing</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Load test</strong></td>
</tr>
<tr>
<td>A test type concerned with measuring the behavior of a component or system. With increasing load, e.g. number of parallel users or transactions, determine what load can be handled by the component or system. This test can be used to “break” the component or system.</td>
</tr>
<tr>
<td>Expected response times</td>
</tr>
<tr>
<td>Tests that the system is providing users with responses to their requests in accordance with the performance requirements, defined KPIs, and SLAs.</td>
</tr>
<tr>
<td><strong>Batch run time</strong></td>
</tr>
<tr>
<td>Ensures that batch processes are executed successfully in acceptable timeframes.</td>
</tr>
<tr>
<td><strong>Stress test</strong></td>
</tr>
<tr>
<td>Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements. This test can also be used to determine how the system or component handles variations in peak load. Testers may use this test to try to “break” the component or system.</td>
</tr>
<tr>
<td><strong>Soak test</strong></td>
</tr>
<tr>
<td>Ensures that performance bottlenecks do not occur when the application is run for a long period of time with typical (or peak) load (e.g. looks for memory leakage). This testing will normally mimic a typical production day(s) and must include all background processing in an application.</td>
</tr>
<tr>
<td><strong>Scale test</strong></td>
</tr>
<tr>
<td>Testing conducted to evaluate how an application will perform at scale in an environment. The aim is to prove linear scalability within the confines of the testing environment. Extrapolation of results may be required if the testing environment is not identical to production.</td>
</tr>
<tr>
<td>An additional aspect of scale testing will be testing of regional usage of the application to ensure that an acceptable level of performance is achieved for users in different geographical locations or on terminal emulation software.</td>
</tr>
</tbody>
</table>
Tasks for performance testing

Define entry and exit criteria

Entry criteria

- Test scripts exist.
- Controlled testing environment for non-functional testing is configured and available.
- The test environment is uploaded with the application to be tested.
- Production test data is available for test execution.

Exit criteria

- Test results for each test run and executive summary report are completed.

Preparation (pre-test validation phase)

Prior to undertaking the performance testing process, a number of validation steps need to be undertaken to ensure the following:

- Pega has been correctly deployed.
- Pega is operating in a stable manner.
- Supporting infrastructure is in place to ensure testing consistency.

Prepare the test data and test IDs

- Data volumes need to be scaled in relation to the relative size of the test and production systems.
- The minimum test data required for each scenario will need to be estimated, and the data should be provided.
- Test data will need to be made available for batch test executions.
- Existing data needs to be in the correct state to ensure that background processing can also be executed.
- Test IDs need to be setup with correct roles and privileges.
- The system should be tested against a number of test IDs as well as with single test ID load tests (results can be misleading and we can see some unwanted errors in logs.)
Performance testing

Pega PAL and DB trace review

- Identify the processes that are to be executed as part of the test and perform Pega PAL and DB Trace analysis to identify easy tuning opportunities.
- Correct any issues identified in the PAL and DB Trace analysis, i.e. missing DB index, query tuning, blob fetch, or any service call that can run in parallel rather than running sequentially. Look for Pega0004 and Pega0005 alerts in the alerts logs.
- Try looking at the data collected in single call or rendered on a single screen and try to work with the business to find out if the screen or process can be redesigned. Look for Pega0001 alerts in the alert logs.
- Iterate through the process until all significant issues have been corrected.

Baseline and tune

Baseline the Pega application based on a single user performing the automated scripts.

- This is the best-case performance.

Baseline the Pega application based on a large subset of users (50 or 100) performing the automated scripts.

- Identify performance bottlenecks and fix them.
- Tune and re-baseline.

Approach overview

1. Prepare and audit the test environment (system test, clone, pre-production).
2. Complete, review, and agree upon the engagement document with all stakeholders.
3. Build simulators, test harnesses, and any stubs required.
4. Define AUM.
5. Setup monitoring and diagnostic tools.
8. Execute acceptance testing.
9. Analyze and compile results.
10. Tune application and environment, re-baseline, and retest.
11. Execute assessment testing.
Performance testing

12. Analyze and compile results.
13. Produce interim report on findings.
15. Produce final report and recommendations.

Test data requirements
There are certain elements of the testing process that need to be documented separately. The main four elements are listed below:

- AUM
  - Quantitative details of how the application is expected to be used in production detailing the split of processing by frequency and user volumes.
- Reference data
  - Details of the data required to ensure that the system functions.
- User login and passwords
  - Details of the user login ID, password, and user roles.
- Test data
  - Details of the data required to enable the scripts to be executed including how the data is to be created.

Test results
The following artifacts are required to evaluate the outcome of each iteration of the performance test.

- Pega application and alert log files
- Pega log usage data
- Database statistics data
  - Oracle stats pack reports
    - AWR
    - ADDM
    - ASH
- Operating system data
  - CPU utilization
Performance testing

- Memory utilization
  - JVM memory
    - Verbose garbage collection data
  - Network utilization data

It is a best practice to ensure that each testing iteration starts from the same set of data and that the log files listed above are deleted before each iteration. When using Oracle, the `flashback` functionality should be considered as an option to ensure each test starts from the same point in time.

Before execution of each testing phase, Pega and WebSphere will need to be restarted to ensure the log files represent only the elapsed time for each test.
Frequently Asked Questions

Our organization is new to the Pega Platform; what are some performance testing tools that are supported by Pega?

Pega supports a wide range of software for testing, including JMeter and LoadRunner, please work with your Pega representative for any specific questions. Here are a few examples:

JMeter
- Apache JMeter is a 100 percent pure Java desktop application designed to load test software (such as a web application).
- It may be used to test performance both on static and dynamic resources, such as static files, Java Servlets, CGI scripts, Java objects, databases, and FTP servers.
- JMeter can be used to simulate a heavy load on a server, network, or object to test its strength or to analyze overall performance under different load types.

Types of test plans supported with JMeter
- Web test plan/advanced web test plan
- JDBC
- FTP
- JMS Point-to-Point/ JMS Topic
- LDAP/LDAP Extended
- WebServices (SOAP)

LoadRunner
HPE LoadRunner is a software testing tool from Hewlett Packard Enterprise. It is used to test applications, measuring system behavior and performance under load. HPE LoadRunner can simulate thousands of users concurrently using application software, recording, and later analyzing the performance of key components of the application.

LoadRunner simulates user activity by generating messages between application components or by simulating interactions with the user interface, such as keypresses or mouse movements. The messages and interactions to be generated are stored in scripts. LoadRunner can generate the scripts by recording them, such as logging HTTP requests.
Our organization has completed the first Robotic Process Automation (RPA) implementation using Pega. How can we make sure the performance of the system will not be an issue in production?

An RPA or Robotic Desktop Automation (RDA) application doesn't need any special performance testing. Unlike other technologies, robotics simply overlays the applications that are currently in place. So, in simple terms, the "robots" are pressing the same buttons within any given application that a user would press when performing the same transaction sequence. Load testing is dependent upon the speed and performance of the applications that we lay on top. RPA/RDA applications can move no faster than the speed of the underlying applications and we can always move faster than those apps. So, load burden does not typically fall to RPA/RDA implementation.

In the field, when conducting an RPA ("batch") transaction, it's always good practice to test with higher volumes to make sure nothing crashes. But since the robots keep the load on the underlying systems, things are often "perceived" as being better. That's mostly because the processing can happen off hours or more robots can be assigned to do the same work (as long as the underlying applications can handle more multiple robots).

Our organization has completed the first implementation of outbound marketing. We can test the campaign using seed and distribution tests, but how can we do the load testing?

In addition to multi-channel Next-Best-Action, Pega Marketing enables the creation of simple outbound campaigns in which a message can be sent to a target audience over a single outbound channel. Testing the outbound campaign is tricky, as rules are not tested with the same kind of data needed by an actual campaign.

There are some open source tools that can be used for performance testing of email outbound marketing campaigns.

FakeSMTP is a free fake SMTP server written in Java. It has a GUI for easily testing emails in applications. Configure your application to use "localhost" as your SMTP server, and all emails will be and displayed in this software. Visit http://nilhcem.com/FakeSMTP/ for more information.

DevNull SMTP is an imitation SMTP server designed for testing purposes. It helps identify problems with an email server.

For more information on performance testing, visit: