PLANNING HIGH VOLUME LOAD TESTS – A METHODOLOGY
## Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example</strong></td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Textual cross-references to other documents.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>Emphasized words or expressions.</td>
</tr>
<tr>
<td><strong>EXAMPLE</strong></td>
<td>Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td><code>&lt;Example&gt;</code></td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td><strong>EXAMPLE</strong></td>
<td>Keys on the keyboard, for example, F2 or ENTER.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="caution-icon.png" alt="Caution" /></td>
<td>Caution</td>
</tr>
<tr>
<td><img src="example-icon.png" alt="Example" /></td>
<td>Example</td>
</tr>
<tr>
<td><img src="note-icon.png" alt="Note" /></td>
<td>Note</td>
</tr>
<tr>
<td><img src="recommendation-icon.png" alt="Recommendation" /></td>
<td>Recommendation</td>
</tr>
<tr>
<td><img src="syntax-icon.png" alt="Syntax" /></td>
<td>Syntax</td>
</tr>
</tbody>
</table>
## Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>February 2010</td>
<td>Document Version 2.0.</td>
</tr>
</tbody>
</table>
# Table of Contents

1. Introduction .................................................................................................................. 5

2. High-level Preconditions and Tasks .............................................................................. 8

3. Planning the load test ..................................................................................................... 11
   3.1 Determine the bread-and-butter processes ............................................................... 11
   3.2 Document the expected throughput figures and/or user and system response times ... 12
   3.3 Determine functions and define scope ....................................................................... 13
      3.3.1 Determine functions needed & the resulting software components .................... 13
      3.3.2 Define the scope ............................................................................................... 13
   3.4 Determine the data distribution – i.e. the type of master data you need ................. 14
   3.5 Create test procedures ............................................................................................ 14
   3.6 Handling optimization recommendations .................................................................. 15
   3.7 Size and lay out of the test system .......................................................................... 16
   3.8 Determine and request infrastructure of load test system ....................................... 16
   3.9 Set up test scenarios for test systems ...................................................................... 16
   3.10 Decide on methods of load and user simulation ..................................................... 17
   3.11 Install and Set up the system .................................................................................. 18
   3.12 Create test data ...................................................................................................... 18

4. Running the tests ........................................................................................................... 19
   4.1 Run the single-user test or unit performance tests .................................................... 19
      4.1.1 Scalability and linearity testing ....................................................................... 20
   4.2 Run the volume load test ......................................................................................... 21
   4.3 Document the test results ......................................................................................... 21

5. Evaluating the results .................................................................................................... 22

6. Conclusion ...................................................................................................................... 23
1 Introduction

This document provides tips and best practices when planning load tests. It is based on the experience of the SAP Performance and Benchmark Team, whose expertise in load testing activities ranges from the SAP Standard Application Benchmarks (standardized load tests owned by our group, see also www.sap.com/benchmark) to ad-hoc load tests for customers and volume studies planned and executed by SAP together with the consulting or support organization or in collaboration with customers and partners.

Load tests come in various forms and guises. It all depends on the goals that you want to achieve. For example, load tests can be performed as a feasibility study or proof of volume. A load test may be used to validate a previous sizing (for details on sizing see service.sap.com/sizing), to perform integration checks or to understand the system behavior in case of backup or recovery. Often used synonyms are stress test, volume test or baseline performance test. Strictly speaking they are all slightly different. The table below gives you some possible definitions for performance tests.
<table>
<thead>
<tr>
<th>Type of Performance Test</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Analysis</td>
<td>Determine how fast some aspect of a system performs, such as: Single user tests A single transaction test Helps to understand what parts of the system or workload cause the system to perform badly (diagnosis case)</td>
</tr>
<tr>
<td>Scalability Testing</td>
<td>Can be a series of single user performance tests or multi-user stress tests. Measure system behavior with varying workloads. Determine effects of adding additional hardware to distribute work among system components Testing for linear resource consumption to obtain scalability with Number of objects, size of objects, number of concurrent users, ...</td>
</tr>
<tr>
<td>Volume Testing or Load Testing</td>
<td>Blanket term that is used in many different ways across the professional software testing community Often used as a synonym for multi-user (load) testing Practice of modeling the expected usage of a software program by simulating the usage of specific program services, thus representing important business scenarios Concurrent users, that is, online activity Background jobs Various workloads (expected peak load)</td>
</tr>
<tr>
<td>Stress Testing</td>
<td>Subset of load testing Explores the behavior in system overload situation and the recovery mechanisms Demonstrates that the system meets performance criteria Reflects peaks beyond the expected peak load</td>
</tr>
</tbody>
</table>

Table 1: Different performance test definitions

In this document we focus on load tests that are planned into the product lifecycle prior to going live with the expected peak load.

This document provides load test project teams with tips and best practices on how to plan and conduct load tests. Often, load tests are simply equaled with certain tools. Tools are actually secondary to setting up meaningful test procedures, which is why they are not part of this paper.

Choose a good point in time to do the test

The best time to conduct a load test project is when you have functionally stable software and a couple of weeks or months to plan and run the test (series), and to incorporate the findings into the coding or system administration blue book.

In real life, however, functional correctness is a moving target, so you may want to consider conducting a series of smaller performance tests on those parts of business processes that are functionally stable. That way you can save some time before the full functional correctness has been reached and you are left with too little time in the
end. The advantage of pre-emptive performance tests is that you can address those performance issues that may have an impact on the functions, for example, issues caused by load coming through interfaces. Experienced performance test specialists even advise to start performance testing as soon as possible, even with only 50% working functionality. This decision depends strongly on the actual situation at the customer.

One of the reasons for recommending partial tests is based on the experience from numerous customers that decided to run the load tests after the functional integration tests were done, say, six weeks prior to going live. The changes to the project plan - while the going live date remained unchanged - left the load test team with (nearly) no room to move and let to poor load test results.

A quick word on test tools

Make sure you make an informed decision on the appropriate test tool. You can save a lot of time in the load test project if the load generation tool has excellent capabilities with regard to:

- What KPIs are monitored
- Does it reliably measure the performance KPIs you defined such as CPU time, memory consumption, database accesses, and so on (see also chapter 3.2 for setting meaningful performance KPIs).
- Scripts which can be easily modified and parameterized
- Automatic correlation analysis
- Integration of monitors from the environment under test so that all (most of) the critical resources are correlated to the response times, and so on
- Features to analyze log files from the environment under test to verify complete correctness of the execution flow
- Mechanisms to check correctness of replies to send requests so that failures during the load tests can be precisely accounted for.
- Stable behavior in long-running tests (no crashes in the middle of a run)
- Features for scheduling unattended runs
- Graphical representation of results
- Adequate record and play functionality (important for regression testing)
- Quality and reproducibility of statistics collection
- Is a specific front end required, if and how is front end data being collected?
- Is there any additional impact on the back end caused by the tool?
- What kind of result interpretation is being offered
- What is the additional resource consumption by tool itself (especially in terms of memory and CPU)
- What are the costs for the tool (licenses for test users)

Some of the above mentioned requirements certainly are more important than others, but do not fall into the trap of focusing on one prominent feature alone. For example, many tools report the server response time. While this KPI is certainly an important one, it is by no means the one to help you understand the actual performance behavior of the application. Many tools fail to deliver the actual CPU consumption during a load test.
2 High-level Preconditions and Tasks

The small checklist below will help you in the long-term planning phase of the load test.

- A dedicated test system is available.
  
  The test system should be as close to the production system as possible. Some clients even take the servers that are meant for the production system and use it for the tests. In this case you need to be very sure the tests will be finalized in time.

- The software will be functionally correct in time.
  
  Test scripts are very sensitive. If they try to simulate functional correct business cases on functionally incorrect software, this will cause problems. Also, check the test scripts before and after each modification to the system such as the installation of patch, configuration changes, and so on. “Before” to make sure there have a good starting situation and “after” to see if the changes affected the scripts.

  If scripts stop running smoothly, do not manually repair them but instead record, correlate and parameterize from scratch. The flexibility and usability of the load generation tool here is very essential to reduce turn-around cycles.

- The project team is made up of people (either full-time or temporary) who
  
  - Know the software applications from a business perspective so that they can decide which processes are the most critical ones (decision on “bread-and-butter processes”).
  - Have experience in performance testing and optimization so that they are able to deal with performance problems when they arise. This includes knowing how to analyze performance in single-user mode in addition to understanding how to do so in load test mode.
  - Can set up and operate the test system or even test landscape, they are referred to as IT team. They should also be familiar with the particularities of the database and operating system. In some cases, customers also invite the hardware vendor to participate in the load test.
  - Are familiar with the respective load test tool
  - Will do the project management part, especially when it comes to communication
  - Altogether, the team may consist of many people, depending of course on the complexity and duration of the load test project. The good news is that not all of them need to stay on board all the time, because the larger the team the less flexible it will be.

The table below gives you an overview of the different tasks involved in a load test project. For readability, the project team has been subdivided into two teams, application (the business perspective) and IT (the technical perspective). Please note that the list below does not necessarily constitute a sequence, a number of activities can run in parallel.
<table>
<thead>
<tr>
<th>Task</th>
<th>Application</th>
<th>Performance Expert</th>
<th>IT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Determine critical business processes</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>Set throughput and response time expectations</td>
<td>x</td>
<td>(x)</td>
</tr>
<tr>
<td>3.</td>
<td>Determine needed functionality</td>
<td>x</td>
<td>(x)</td>
</tr>
<tr>
<td>4.</td>
<td>Determine data distribution</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>5.</td>
<td>Establish test scenarios for business processes</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>6.</td>
<td>Do initial capacity planning for load test system</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>7.</td>
<td>Determine and request infrastructure of load test system</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>8.</td>
<td>Set up test scenarios for test systems</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>9.</td>
<td>Decide on methods of load and user simulation</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>10.</td>
<td>Provide test systems with basic software installed</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>11.</td>
<td>Create actual test data</td>
<td>x</td>
<td>(x)</td>
</tr>
<tr>
<td>12.</td>
<td>Perform single-user tests or unit testing and optimize coding</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>13.</td>
<td>Perform volume tests</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>14.</td>
<td>Evaluate test results</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Table 2: tasks in a load test project
The graph below gives an overview of which tasks can run in parallel.

Graph 1: different tasks in parallel
3 Planning the load test

The pure planning phase probably is also the most time-consuming one. The more experience the project teams have in setting up and running load tests, the less time they need, obviously.

Whatever decisions you take in this phase will have consequences for the execution of the test cases as well as for the evaluation of the tests. Therefore, allocate 60-70% of the project runtime for the planning steps if you perform load tests before going live. If load testing is included in the development cycle of custom code, teams often drop the careful planning in favor of a more flexible circular approach, namely Plan → test → match to expectation. This circular approach however, should not be used for sizing feasibility checks.

The planning phase includes the initial steps until writing the scripts and setting up the test system, this means tasks 1 to 6 in table 1 above.

3.1 Determine the bread-and-butter processes

There are basically two sets of performance requirements: user-driven and throughput-driven. In the first example you analyze the system from an end user perspective, considering perceived performance requirements as well as system performance requirements. In the second case response time is negligible. Here, you test the nightly background jobs where hundreds of thousands of data items need to be processed in a few hours, or where several processes are interdependent on each other.

Tip: Try to reduce the number of processes as much as you can. Five to ten should be sufficient as it is better to define a few scenarios well than have many ill-defined scenarios. There are mainly two reasons for this: keep the complexity at a minimum. Also, the test cases need to be scripted and analyzed from different perspectives.

Examples for bread-and-butter processes:

- Client A planned to include a single-sign on authorization check in their corporate portal. For them the end user response time for this process was very important.
- Client B ran several different systems and applications within one process chain. Their critical path analysis determined that the full order processing had to be completed within 15 minutes or else they would run into severe business problems.

Determining the bread-and-butter processes sometimes is a trade-off as well. If you have a process that will certainly dominate the resource consumption, you may then decide to skip another important process whose impact will probably not be equally load-intensive.

Note that the process should be designed according to the business usage and not the technical feasibility.

In addition to asking the application owners, you can also make use of the legacy system to understand the core business processes and critical applications. The legacy system can also be the source for the data basis used for the load tests (see section 3.4)
3.2 Document the expected throughput figures and/or user and system response times

The goal of this task is to define the scope of the load test. Very often, the project team and the stakeholders have a goal in mind, for example: “The response time of scenario a must be below one second.” This seems straightforward enough. However, when you look at this requirement from the perspective of the test evaluation you would ask questions like this:

- Is response time the total response time?
- Is response time the server response time?
- If the scenario consists of several user interaction steps, does each individual response time have to be below one second, or can the average of all steps be below one second?
- Most load test tools only report the server response time, thus omitting a sometimes important aspect: the (browser) rendering time. This, in addition to the network time, adds to this reported response time. If your tool does not support frontend tests we recommend performing single user tests to understand the performance behavior on the frontend.

Even if the example above refers to response time, it insufficiently describes performance if used as only KPI. We strongly recommend to also define consumption-based KPIs such as memory consumption, actual CPU consumption, network load, disk I/O and others in addition to the business volume throughput. This example is also to show you that it is very important to define the business requirements in as much detail as possible so that they can be turned into meaningful and measurable performance baselines. Very often, you find these requirements in a development scope document.

It is important to include end users or the implementation team in this task.

Other examples of expected data throughput:

- Client A from above planned a load test for their corporate portal. They determined that the average server response time of the login of their anticipated 40,000 users per hour had to be below one second.
- Client B from above required 75,000 sales order line items to be processed per day. Usually this is not at all performance-intensive. In this case they had 15 minutes for fully business processing these line items across three systems: SAP ERP, SAP SRM and SAP SCM, including a global availability check (ATP). In other words, 83 synchronous ATP-checks had to be performed per second.
3.3 Determine functions and define scope

3.3.1 Determine functions needed & the resulting software components

This and the following tasks can be performed in parallel and may bring together the application and the IT team. However, the application team decides which functions are necessary.

The goal of this task is to set the scope for the actual test preparation in terms of software requirements. For example, if you want to test the synchronous global ATP check you will need an SAP SCM system. Usually the ATP check is triggered by sales orders, so you may have to include an SAP ERP system in the load test environment.

Examples for cross-system scenarios
- ATP (Available-to-Promise) Check in SAP SCM
- Workflow or interfaces to other systems

3.3.2 Define the scope

Discuss if the chosen business processes can actually be implemented. This means that you may decide that the performance of a backend system is not the decisive factor, so you can take measures to simplify the matter. An example of this is SAP Standard Application Benchmark for the SAP NetWeaver Portal which tests the Employee Self Service scenario. As the clear focus of this standard load test is the performance of the Portal, we decided to avoid the backend by including a URL that would have normally been sent from the backend (simulation). That avoids having to install a backend and is an alternative, if you need to make measurements across multiple platforms which are perhaps no significant contributors to the load.

The focus here is the view from the IT perspective where you try to simplify the test procedure as best as you can while still ensuring that the real bottlenecks are tested.

Are you testing an entire process or only part of the process? If you test part of the process, make sure it makes sense and is self-contained. Here’s a bad example: In one load test, open bills were created and a report collected all open bills. In the course of the test, the number of bills increased, and thus the report took longer and longer. In real-life, indefinitely increasing open bills would lead to bankruptcy. The customer’s mistake was to test only a partial process that was not self-contained.

An example for simplifying the scope
- The application team wants to test the year-end closing but the IT team finds out that importing the data from another system will take too long
- So the decision may be taken to use only 50% of the resources and test only 50% of the volume and then scale out
3.4 Determine the data distribution – i.e. the type of master data you need

When you simulate user behavior or background processes you must ensure that they have the proper “infrastructure”. Not in the technical sense, but in the content sense. The test cases must not only be fully functional but also repeatable. This requires a sufficient amount of master data and production-like customizing. A very basic question is to decide on whether you want to use production data or artificial data. The advantage of production data is that the quality of the test findings increases. The disadvantage opposed to artificial data is that it may be difficult to exactly reproduce measurements. It can take much longer to load data into the test system. Production data are generally less controllable, for example, not all data may be of equally high quality, may lead to functionally incorrect behavior of the test script. Artificial data are more laboratory-like.

That way you can insure that multiple testing will not impact the performance.

For example:

- You need to process 50,000 sales orders per hour. If you include the identical 5 line items (that is, materials) in this order, you’ll have (50,000/3600) altogether 14 updates on the same database table per second. If you want to test serialization effects, you set up the scenario this way.

- The same holds true if you only include one business partner to whom all the orders are being sold. In a load test you will quickly run into serialization problems.

If you create an order and simulate adding line items in different iterations, be aware that each iteration increases the processing times more than linearly.

Avoid artificial bottlenecks, for example all orders use the same customer and products. It is not enough to create a vast amount of data by simply copying it.

3.5 Create test procedures

By test procedures we mean a couple of things: determining the sequence of scenarios and setting up the test script.

The starting conditions must allow reproducible measurements, that is, you must be able to repeat the same test all over again with either the same or equivalent data. Depending on the circumstance, there are different possibilities. One frequently used technique is to create database “snapshots” provided by the different RDBMSs. Alternatively, you can clean up data created or modified during the tests, for example by completely deleting tables using truncates or programs/SQL statements to undo changes.

However, even if you test in a small system you ought to have regular back-ups in order to start a test from scratch again or to repeat a test phase.

One way to ensure repeatable test procedures is to specify all the details and document them well. List individual and detailed process steps, the expected input and output for the steps, and also the customizing environment.

This is also independent of whether you will run the load test with a test automation tool or with testers following a script.
The table below is an excerpt from a load test where testers followed a script to create load in the system. You can clearly see the input and the expected output. In addition, the client later added the KPIs obtained.

<table>
<thead>
<tr>
<th>Description</th>
<th>Step</th>
<th>Verbal Description of step</th>
<th>Fields</th>
<th>Value</th>
<th>Expected result</th>
<th>Time in CPU secs.</th>
<th>Memory measured in MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer calls company</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customer is diverted to agent</td>
<td>2</td>
<td>Search in GP-Locator</td>
<td>Name</td>
<td>Smith</td>
<td>Customer is not found. System displays message</td>
<td>0.13</td>
<td>6.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Choose search</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent creates business partner</td>
<td>3</td>
<td>Choose /nbp to create business partner</td>
<td>Go to screen “create business partner”</td>
<td>0.05</td>
<td>3.8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Enter new business partner</td>
<td>4</td>
<td>User enters contact information</td>
<td>NAME, STREET, CITY</td>
<td>Smith, Mews, London</td>
<td>Message: “data saved”</td>
<td>0.05</td>
<td>4.2</td>
</tr>
</tbody>
</table>

- Table 3: Example for test case plus expected values

Another method of documenting the process is to produce screenshots of each individual step. Just make sure the writing is large enough.

### 3.6 Handling optimization recommendations

A load test must always include prior single user tests or unit tests to predict if the target load can be handled at all and to optimize coding as well as more technical set ups such as customizing, disk layout, parameterization and others. You can be sure that the team of performance experts will make suggestions for optimizations both in the single user/unit tests and the volume tests, for example in the hardware, in the customizing, parameterization, application coding, and many others. You must then decide which actions to optimize the performance can be done quickly and with no effort as opposed to those which will perhaps take longer and have more impact on the source code because of interdependencies. It has proven useful to establish a process in which all parties, application (functional and development), performance experts, IT, even hardware vendor or storage partner, depending on the complexity of the project, regularly meet to discuss how to deal with the recommendations issued during the load test.

A matrix like the one below can be helpful:

- How quickly can the recommendation be implemented?
- How complex is the recommendation, can other areas be affected?
- How can the recommendation’s effectiveness be tested?
- Who is responsible to implement the recommendation?
3.7 Size and lay out of the test system

The application and IT teams use a combination of standard sizing procedures such as the Quick Sizer (http://service.sap.com/quicksizing) and other sizing guidelines. The sizing can be done in parallel to the previous phases, although it makes sense to consider findings of the other phases/steps, such as the setup of the scenario. On the other hand you will need to order the hardware well in advance.

Some clients already have a specific hardware in place. In this case you need to check if the server(s) are capable of processing your target load or if you need to change the performance baseline according to the power of the hardware.

You may want to contact your hardware vendor for assistance in the sizing process.

3.8 Determine and request infrastructure of load test system

In this task you also define the test system landscape and system configuration, if applicable. It can be done together with the sizing task.

Here you prepare the test environment, including server(s), front ends, disk layout, instances and clients. Make sure that the system landscape is well configured and that a backup and restore strategy is in place.

In some cases it makes sense to start testing with a small system which you gradually extend.

- For volume throughput test with a high I/O activity, it makes sense to install more and smaller disks than a few large ones.
- To avoid overloading and bottlenecks, make sure that when you do the database layout, hotspot tables are not located on the same disk.
- If you run a complete test of a critical central business process that is closely connected with an interface, you have to incorporate the interface into the test as well — at least enough to allow you to run the test. For example, if you have a tax calculation interface, you must set this up and run it, too. And if your production system is to include a WAN connection, you should test the WAN connection rather than the LAN connection.
- It is a good idea to prepare backup and restore strategies for the two test systems — the development system and the load test system. Even the small development system ought to have regular backups so that you can repeat a test phase or start a test from scratch again. Keep in mind that load testing is an iterative process — during the test, you are likely to come across problems, and you might have to go back and start a phase again.

We recommend you perform backups in addition to documenting your efforts as you go along — for example, anytime you make a change to a program, to the customizing settings, to the master data, or to the database layout.

3.9 Set up test scenarios for test systems

This depends very much on the goals of the technical infrastructure team. One possible test scenario is to test the recovery time including the disk I/O behavior of application and storage. Another test could be to test how long it takes to fill the buffers and caches when starting a cold system.
3.10 Decide on methods of load and user simulation

Decide what data is inserted by real users or by tools. The advantage of using testers is that the script also serves as some kind of training for them. Also, they can provide feedback on the content, usability, or perceived performance as opposed to the actual service and response times provided by a tool.

Whichever tool is going to be used is determined by the Performance Expert together with the IT Team because they are responsible for the final implementation and possibly have the experience. If you do not know which tool to use, the following list may give you some tips how to evaluate the different offerings:

- Outsourcing/leasing may be possible, for example for Web applications.

With this task, which can occur in parallel to some of the previous tasks, the planning phase has come to an end and the preparation for the load test can start.
3.11 Install and Set up the system

This task follows along the lines of the implementation guidelines issued by SAP. There should be no surprises here. Make sure

- The buffers are large enough
- The people who will monitor the system performance later on have all the necessary authorizations

3.12 Create test data

Fill the test system with master data according to the strategy defined in task 3.4. Also do the required customizing.

In many situations it is reasonable to keep on creating test data in the process of testing. This approach is closer to the actual runtime of the system in production. Let’s look at one example: load tests are planned for a public “Web Forums” application. The test data of course will consist of some initial users which are empowered to create forums, threads and messages or reply on them. It will be best if the test begins with no volume of data, as this is how the system will be delivered for production. Then, build load test scripts with suitable parameterization which will keep on growing the content in the system while at the same time other scripts simulate other typical user activities like searching by pattern for occurrences in forums, deleting some threads, closing answered messages, etc.

Another example could be a search engine test where good idea delta indexing (increase of indexed content) is to be tested in parallel to the searching.

By using the load testing to create test data you forgo the target of reproducible results in favor of discovering issues related to volume growth over time. This is just another aspect of load testing.
4 Running the tests

One basic principle for achieving meaningful performance results is to let the system warm up with pre-runs. This fills buffers and caches and in general minimizes the influence of artifacts. The rule applies to single-user tests and load tests alike. The advantage is that the results will be reproducible, a very important factor for performance analysis and regression testing.

Pre-runs also bear the advantage that you can still check if there are any functional issues with the test cases. However, you could also decide to measure the performance of a “cold” system, because a cold start shows non-optimal programming on first access of the applications (optimization potential). This could become critical in situations where you cannot increase buffers and caches due to hardware limitations and eviction of cached data is necessary or cached content simply expires. By having the system restarted automatically by the load testing environment just before the load generation is triggered, you can get a reproducible initial state of all buffers and caches. On the other hand, these results are not reproducible and should not be used as a basis for regression testing.

It is important that you understand the implication of each method for your test results.

4.1 Run the single-user test or unit performance tests

A single user test refers to one user clicking through a series of screens entering data as required while these activities are being performance-monitored. Of course the test can also be automated. However, the important aspect here is a manageable well-defined test scenario. A unit test describes roughly the same, where a huge background application is split into smaller jobs for an initial analysis.

These tests can also run in a smaller test system, you do not necessarily need the complete load test infrastructure for these kinds of tests. Whichever system you will use, for optimal results make sure you have the system all to yourself so only your activities are captured.

If you really cannot be alone, then at least make sure there are no background processes, and that the CPU utilization is below 10%.

If there are serious time restraints, you can capture only the performance KPIs (such as the number of database accesses, amount of data retrieved, CPU time per step, peak memory consumption, number of roundtrips to front end and between two servers and transferred kilo bytes to front end).

If you have more time you can analyze the quality of the database accesses (index design, identical selects, memory allocation, quality and quantity of cached data, and so on) and remove bugs.

These findings can then be returned to the application team which is responsible to modify the application software, as appropriate.
4.1.1 Scalability and linearity testing

To test whether the solution is scalable, perform "linearity tests" of the different test cases. A performance specialist can use the test scripts and make the appropriate analysis. These tests are basically a repeatable pattern for (example):

- Scalability with an increasing number of objects
  - 1 object, five line items
  - 10 objects, five line items
  - 100 objects, five line items

- Scalability with an increasing of line items
  - 1 object, 1 line item
  - 1 object, 10 line items
  - 1 object, 100 line items

The results of these controlled tests can be analyzed to determine if any performance bottlenecks exist. Ideally, you would see a graph like the one below:

![Graph showing CPU time vs. number of bidders](image)

Figure x: Good example for a single-user test checking linear resource consumption according to number of bidders.

In reality, however, these single user or unit tests already deliver recommendations to improve the performance. See section 3.6 on dealing with these recommendations.

If you are satisfied with the results of the initial performance tests, at whichever point in time they have taken place, you should start with the load test.
4.2 Run the volume load test

Start the exercise by performing dry runs to fill buffers and caches. If you want to include the warm-up period in the tests you can start testing immediately, but in this case you need to make sure to separate the “cold tests” (focus system behavior) from the “warm tests” (focus on application behavior).

- Start with a small load and increase the load step-by-step towards the final throughput target. Most customers double the load in iterations.
- Measure the throughput
- Measure the resource consumption using appropriate monitoring tools (ideally integrated in load generation tool)
- Carefully monitor the expected performance bottlenecks
- Optimize system parameter settings and repeat the test executions if necessary

4.3 Document the test results

In order to properly evaluate the success of the results, they must be documented:

- A meticulously maintained journal tracking in an overview-like manner all runs, all modifications (application, customizing, operating system, hardware, storage, database, and so on) and results so that you can analyze the different impacts in the course of time and perhaps undo some actions.
- Archive throughput numbers and the corresponding resource consumption, including additional monitoring data so you can see progress or setbacks over time
- Limiting components (bottlenecks)
- Test system environment (hardware and software releases)
- If you optimized the performance at some stage (results before and after)
- If the business functions need to be changed
- If application coding needs to be changed the application team needs to know
- Lessons learned in terms of operating the system (tuning parameters and know-how on relations between throughput/load and parameter values as well as dependencies between different parameters, etc)

Some of this documentation will form the basis for the project report document that you can distribute to the stake holders.
5 Evaluating the results

All parties get together to review the results of the multi user load tests to see whether they are ready to go live with the SAP system.

Ideally, your test was successful if the results (throughput, response time, etc.) are in line with your documented predictions and expectations. If the expectations set in the beginning have been met, the test was successful. Otherwise, analyses should be made as to why the goals of the load test were not met — go back through the 13 steps and rethink each step.
6 Conclusion

This article described a process for successfully conducting a load test. The pragmatic approach is designed to help you avoid many mistakes if each of the steps described in the procedure is taken into consideration. The Performance & Benchmark team become involved in many load tests at a very late stage, and we have seen firsthand what happens, so the urgency with which we stress many of the steps here stems from experience. It often seems that load testing has become l'art pour l'art rather than a process focused on well-specified needs, and while for many large customers it is always a good idea to incorporate load testing into a project implementation, it should be thoroughly analyzed beforehand. You certainly do not need to test business processes that resemble standard application programs, for example, but if you have created your own coding, you should definitely analyze the processes in single-user mode.

Most customers underestimate the time and skill required to conduct a custom load test, and in such cases, vendor experts, SAP, or consultants are called in to help solve problems that would not have arisen had thorough research been in place at the beginning.