Scalability: Pushing the limits

Neha Rai (neha_rai@mcafee.com)

Tim Schooley (tim_schooley@mcafee.com)

Tejas Patil (tejas_patil@mcafee.com)

Intel Security Group (McAfee)

Abstract

How do we determine if software is a success? For server based applications, success could mean more load, either as a web app or a client-server application. As the numbers of end users grow, the load on a server often increases substantially. Hence it becomes crucial to address the scalability challenges of the software to meet the rapidly growing need.

Most scalability flaws are deep in the architecture, which are very hard and expensive to change later. The quick and easy fix is to add more servers. However, by including more servers, one can hit diminishing returns very quickly, unless the software is designed to scale that way. So, is there a magical way to determining how much software can scale up?

This paper talks about the scalability of client-server software, which is common in an enterprise environment. Here we want to stress the importance of performing scalability tests on any client server application by creating a simulated environment. We are going to address some key questions that arise during the process, like:

- What should be covered in scalability testing? What is the target?
- How to determine is the scalability limit?
- How to identify scalability bottleneck?
- How to determine if the test results are complete and testing should stop?

Biography

Neha Rai is a QA Engineer at Intel Security Group (McAfee) with more than 6 years of quality assurance experience. She holds a Master degree in Computer Application from National Institute of Technology, Karnataka and currently working in Brighton, UK. Being *highly passionate about quality, she strives to look for ways and methods to improve software reliability and usability. In Last couple of years she has become extensively involved in scale testing of McAfee Drive Encryption product.*

Tim Schooley is an Agile Product Owner and Senior Software Engineer at Intel Security Group, and holds a Master of Science degree in Computer Science from UCL, London. Currently working in Brighton, UK, he develops a number of data protection products and manages their respective backlogs. He is a strong proponent of delivering maximum value and quality.

Tejas Patil is a QA Lead and Scrum Master at Intel Security Group (McAfee) in Brighton, UK. He has worked for past 9 years in McAfee developing, testing and managing software. His main responsibilities include driving the concept of software quality throughout the product development lifecycle. He is passionate about building high quality products that excite customers.

1. Introduction

Scalability is necessary in technology and business settings. The base concept is consistency – the ability for a business or technology to accept increased volume without impacting the contribution margin, which is deduced by subtracting variable costs from the total revenue. For example, a given piece of equipment may have a capacity for 1–1000 users, while beyond 1000 users additional equipment is needed or performance will decline (variable costs will increase and reduce the contribution margin).

Recently, HBO aired the much-anticipated season finale of 'True Detective' on their network, which included a live streaming of their program on HBO Go app. Apparently, they underestimated the popularity of the finale. The live stream and the app crashed due to the massive number of people trying to watch the show, resulting in a blackout for majority of users. And… the debacle continued, the same thing happened again during Game of Thrones premiere. Furious fans went to social media to express their unhappiness at being unable to watch the episode.

Another example where software is facing scalability issues is social networking websites. Many of us are familiar that Facebook, Digg and Twitter have been making improvements to make their apps more scalable. Unsurprisingly, the challenge is not limited to web based applications, but any client-server model can trip over.

Scalability Testing refers to performance testing that is focused on understanding how an application scales. As the product is deployed on larger number of systems or as more loads is applied to it, we need to understand what happens to it. Our goal is to understand at what point the application stops scaling and identify the reasons for this. Scalability testing helps to determine whether your management application scales with your workload growth as the managed network grow in numbers and complexity. Hence, scalability testing forms an essential part of the entire development and testing process of any server based application.

2. Start with understanding the product vision

Before you start scalability testing, be sure that you understand the current project scaling vision. The project vision is the foundation for determining what kind of testing is necessary and what is valuable. For example when we started scale testing for our product we started with the vision to support 'hundred thousand client nodes per server'. Or for any other application the vision could be to support 100, 000 concurrent users.

By having scalability vision, we set the scalability expectation for our product to meet. So, by having a clear vision, when testing is done, we will know if the product meets our quality goals.

3. Identify the test environment

Identify the physical test environment and the production environment as well as the tools and resources available to the test team. The physical environment includes hardware, software and network configurations. Having a thorough understanding of the entire test environment at the outset enables more efficient test design and planning and helps you identify testing challenges early in the project. To achieve the goals of scalability testing, a unique test bed would need to be built. It is important to create a separate testing environment that is comparable to production. If the machine configuration, speed, and setup aren't the same, extrapolating performance in production is nearly impossible. To make sure your environment is right, be sure to consider:

- Hardware where server software runs. Identify where database, primary server and additional servers will run.

- Hardware that the client software (or simulator) runs on.
- Software you plan to use to capture the performance data like SQL profiler, perfmon etc.

In addition to the resources necessary to run the scalability lab, success of your load testing effort depends on other roles within your organization. Identify a scalability testing team, there will be scenarios when we encounter issues that are hard to debug. This would require re-testing with a debugger attached and would need involvement of the development team. Considering these situations well ahead will save you from last minute surprises. Other than QA and a Development manager for the project, you need support of other team members to execute the scalability testing successfully. These elements are key to that success:

- Scale lab team - Takes ownership of the effort and runs the system test. At least one person is required.
- Development team - Identifies and tracks problems involving stability and performance. At least one person is required.
- Database Engineer - Identifies and solves database problems. At least one person is required.

Scalability problems are most likely rooted in the database, the data access strategy (such as stored procedures, prepared statements, or inline SQL) or data access technologies (such as ADO, ODBC, and so on). A well-qualified DBA available as a full-time resource plays a key role in scale testing effort.

4. Identify Acceptance Criteria

When identifying acceptance criteria, we need to identify the response time, throughput, resource utilization goals and constraints. In general, response time is a user concern, throughput is a business concern and resource utilization is a system concern.

Counter to this, sometime you would need to start scale testing without knowing what the desired response time; throughput or resource utilization is for an application. Start testing with a smaller number of clients in the setup, in other words simulate only the number of clients that you think your application can handle without any problems. Once you confirm that the test case is successful, start increasing the number of clients. Continue with this approach and work slowly to identify the sweet point (or target throughput), the point where we can say the scale performance is its most efficient. It may be that you reach to a point where you observe a decline in the performance, like the server response time will go down, or when you add more servers and you don't see any performance improvements. In this situation, revisit the last test you did and record the details, like the number of clients, number of servers or any other parameters. This gives you a baseline to calculate the current product throughput and response time that will help to identify if the current state of the product meets the product vision.

Another approach could be to design your test cases around these key questions:

- Where is the system bottleneck, and how many synchronized concurrent requests can it handle?
- How many additional clients can your system support without diminishing results?
- Do the results scale linearly as you add additional hardware?
- Are there any stability issues that prevent the server from operating in a production environment?

This approach uses additional information from the development team which anticipates where problems might arise.

During scale testing, if we can determine the targeted throughput required to support our application in a production environment, then we can easily determine if the current scalability meets the expectation, the ultimate goal of scale testing!!

5. Plan and Define test scenarios

This is most important part of the scale testing. In most cases the major part of your success comes from correct planning.

To plan successfully, be sure to address these key points:

- Identify key scenarios,
- Determine variability among different parameters and how to simulate that variability,
- Define test data
- Establish metrics to collect.
- Consolidate this information into one or more models of system usage to implement, execute and analyze.

At this stage, one should be able to determine if their setup or test environment needs a separate tool for simulating test data. For instance, if your test scenario involves putting load on the server using multiple users or multiple client machines, then you would require an additional tool which can simulate the load. For a scale testing project to be successful, both the approach to testing scalability and the testing itself must be relevant to the context of the project. If you don't have a clear understanding of your project context, scale testing is bound to focus on only those items that the performance tester or test team believes to be important, as opposed to those that truly are important. So by using something which nearly simulates the application behavior, you can cover most of the functionality and discover the scaling bottlenecks.

6. Creating a simulation tool

You should identify all the test scenarios that need to be executed as part of scalability testing before implementing a simulation tool so that we can eliminate any detailed work. In other words, you need to be clear about what is expected of a simulator so that you do not spend time developing features that are unnecessary. Either we use some existing software or develop our own simulator, but identifying the right simulation tool is vital.

When choosing a simulator, remember that in order to determine the capacity of a target environment, we need to adjust the variables to find the maximum scalable point of the system. In addition, the test should simulate other expected loads that will run on the same production environment as the application under test. Since simulation costs typically increase quickly with the level of details, we need to fine tune the balance.

6.1 Testing the simulator before relying on it:

Perform an acceptance test on the simulator you choose. The acceptance tests should be ready before selecting the simulator; this will help confirm that the simulator meets all the feature expectations. All the result and testing are going to rely on how closely the simulator mimics your actual product.

The acceptance test should cover all the scenarios that would be executed as part full-fledged scale test. This will help you judge the simulator's capabilities and let you know about any limitations well in

advance. Furthermore, you can use this as an opportunity to educate yourself about the tool you are going to rely on.

7. Executing Tests

7.1 Collecting performance data:

Performance counters should be set to capture the system state as well as the application state. Use counters to provide information as to how well the operating system, the application, service, and driver are performing.

When you need to measure how much of system resources your application consumes, you need to monitor these items:

- Disk I/O Read and write disk activity. I/O bottlenecks occur if read and write operations begin to queue.
- Memory Available memory, virtual memory, and cache utilization.
- Network Available bandwidth being utilized, network bottlenecks.
- Processor utilization, context switches, interrupts and so on

It is important to choose performance counters according to the application under test so that you can measure the performance of these components efficiently.

The counter data can help identify system bottlenecks, it is also used fine-tune the system and application performance. However, performance counters are only useful to identify the symptoms of a problem, not the cause. Therefore, it's important to use application level logging as well. Every software system has logging requirements so application processing can be monitored and tracked. Enable logging in debug mode which provides detailed information about an event. For application-level data generated by a logging system, it is a good idea to use a viewer that lets you to get immediate access to error and performance information.

7.2 Running small scale tests (eliminating bugs):

Setting up a big scale test with lots of machines and simulators could turn out to be expensive if on the first run you find blocking bugs. First, you need to rectify all the bugs that reduce the scalability, like thread contentions, memory leak, deadlocks etc. To achieve this, start your test on small scale setup but try to keep impact on the server high. In other words, stress the server with lower number of clients by changing other parameters. For example, you target to support 100K client nodes on one server within a randomized interval of suppose 4 hours. In order to keep the impact on the server the same, reduce the number of clients and the number of hours so the overall client request being processed by the server remains the same. Repeating small scale tests are more easily compared to full-fledged scale tests which we can save for later.

In our applications, scalability is affected by the number of users, the number of nodes and agent-server-communication interval. We reduced the number of nodes and reduced the agent-server-communication interval (ASCI) as well to keep the throughput nearly the same. This allowed us to stress the server and to expose issues like database deadlock and memory leaks. During our small scale test cycle, we introduced an additional server while keeping other parameters constant to observe thread contention. The intent here is to keep the server under load, not by increasing the client node count, but by increasing other parameters that affect the product scale performance. What we found that, 90% of the scalability issues came out during small scale tests and the remaining 10% were observed when small scale test

data was extrapolated on a large scale setup. Another advantage of starting with a small scale setup is that they are easy to reset in case you need to verify any bugs found and fixed during the scale testing.

Try identifying a scalability sweet spot in the setup, the point where the scale performance is at its peak with the given parameters. For example, start testing with smaller number of clients with your server. Once the test passes, add more clients while keeping all other parameters constant. If no issues arise, continue to add few more clients, if there is no drop in the server performance you can continue adding the clients until you are concerned with server performance. Once you find the maximum scaling point, write down the configuration details so that you can use it to extrapolate the results to a larger scale. If your application supports additional servers, you should be running similar tests by increasing the number of servers so that you can identify how many additional servers can be included in the system before you observe the scalability loss.

For example in the table below, tests have gone to the point where the number of clients is 12500 and agent server communication (ASCI) per second is 13.88. Beyond this point, the tests started to fail at a higher ASCI/sec rate. Hence 13.88 ASCI/sec is the sweet spot in this setup.

ePO server without handlers					
No. of clients	ASCI (sec)	Total ASCIs/second	Configuration	Observations	
				Activation	1 Token Change
5000	900	5.555555556	1x5000k, 100 Users	Passed	Passed
7500	900	8.333333333	2x3750k, 100 Users	Passed	Passed
10000	900	11.11111111	2x5k, 100 Users	Passed	Passed
12500	900	13.88888889	4x3125k, 100 Users	Passed	Passed
15000	900	16.66666667	3x5k, 100 Users	Passed	Server busy; unresponsive
17500	900	19.44444444	4x4375k, 100 Users		

7.3 Extrapolating (verifying results on big scale):

Once you have results from small scale tests you can use the data to extrapolate the results to a larger scale setup. As in the previous example, we have seen that system works well when ASCI/second is 13.88. Hence by general calculation, if we intend to support 100K nodes per server, we need to distribute the clients over 7204 seconds (nearly 4 hours). Once you have extrapolated the data, the numbers should be verified by testing in a larger scale setup.

Here you can choose to test directly with the target number if you find your build quite stable. Otherwise, depending upon the test results, rate and type of bugs found, you can choose to start with half of the target number or less. For example, test with 50,000 clients and if no issues were found, then you are good to verify the results with your targeted number of clients, 100,000. While extrapolating our small scale results, we observed issues related with thread contention and database deadlocks.

Extrapolating and verifying the data is a mandatory step in the whole process. During our testing, we couldn't validate the results obtained by small scale tests because of the existing issues that were bringing the performance down. These issues only got exposed while testing on a really large scale setup. Therefore, do not just extrapolate the results and give a recommendation, make sure any extrapolations done are well tested.

7.4 Identifying possible bottlenecks:

When scaling results in degraded product performance, it is typically the result of a bottleneck in one or more resources. Bottlenecks are elements of your system that impede the normal flow of traffic.

When your application does not meet performance requirements, you should analyze data from the test results to identify bottlenecks in the system and to formulate a hypothesis of the cause. Sometimes the test data are not sufficient to form a hypothesis, and you must run additional tests using other performance-monitoring tools or with a debugger attached to isolate the cause of the bottleneck. Alternately, you can eliminate the parameters involved in the test to pinpoint the problem.

Once you've taken steps to identify and potentially resolve the bottleneck, repeat the tests to see if the fix has introduced or uncovered any new bottlenecks. Test to see if the scalability limit has increased.

8. Analyze Results and Report

Technical team members need more than just results — they need analysis, comparisons, and details of how the results were obtained. Your technical report should contain information that can be used to re-run the scalability test on later versions. The report should contain all the bugs found during the scale testing cycle and details of fixes made so that one can design the next test cycle accordingly. The document should state any concerns and recommendations. And it should clearly state the maximum scalability limit the product has so they can be verified in subsequent test cycles.

On the other hand, non-technical team member like Managers and other stakeholders need more than just the results from various tests — they need conclusions based on those results, and consolidated data that support those conclusions.

These two groups tend to look for very different things in a performance report and are inclined to prefer different presentation methods. When reporting, make sure that you identify which group you are reporting to and know what their expectations are before deciding on the best way to present the results you have collected.

9. Conclusion

Scalability testing should be implemented as part of the development effort. By implementing a scalability testing plan early, you can minimize any surprises at deployment time. Before going to production, scalability testing is the only way to uncover major problems that are inherent in the architecture. To make scalability testing successful, we need a separate environment with comparable production hardware, a robust simulator, and the teamwork of people in your organization.

Hopefully you will find this paper helpful when planning scalability testing for your projects. I'd like to leave you with the following final suggestions for your next project:

- Start Early
- Include scale testing as part of the development effort
- Choose the right simulator
- Start with small scaled setups and work up to your bigger scale setup.
- Establish distributed logging
- Monitor and interpret your results carefully.
- Analyze the results and report
- Celebrate your success!

10. References

[1]. Scaling Your Internet Business
http://www.scribd.com/doc/15493750/GoGrid-Scaling-Your-Internet-Business

[2]. Testing for Scalability
http://msdn.microsoft.com/en-us/library/aa292189(v=vs.71).aspx

[3]. Estimating the Throughput
http://msdn.microsoft.com/en-us/library/ff648064.aspx

[4]. Avoid bottleneck when your web app goes live
http://msdn.microsoft.com/en-us/magazine/cc188783.aspx