Performance testing of software systems

Conference Paper · January 1998

DOI: 10.1145/287318.287337 · Source: DBLP

2 authors:

Filippos I. Vokolos
Drexel University
12 PUBLICATIONS **912** CITATIONS
SEE PROFILE

Elaine J. Weyuker
University of Central Florida
194 PUBLICATIONS **10,638** CITATIONS
SEE PROFILE

Performance Testing of Software Systems

Filippos I. Vokolos
AT&T Labs - Applied Tech.
filip@att.com

Elaine J. Weyuker
AT&T Labs - Research
weyuker@research.att.com

Abstract

Approaches to software performance testing are discussed. A case study describing the experience of using these approaches for testing the performance of a system used as a gateway in a large industrial client/server transaction processing application is presented.

1. Introduction

Although there has been very little work published describing approaches to software performance testing, it is nonetheless an extremely significant issue for many large industrial projects. Often, the primary problems that projects report after field release are not system crashes or incorrect system responses, but rather system performance degradation or problems handling required system throughput. When queried, it is not uncommon to learn that although the software system has gone through extensive functionality testing, it was never really tested to assess its expected performance.

The issues that have to be addressed when doing performance testing, differ in a number of ways from the issues that must be addressed when doing typical functionality testing. In this paper, we will examine performance testing issues and describe high level approaches to address some of these issues for certain classes of software systems. These approaches are drawn from experiences in testing the performance of different industrial software systems. We will also describe the approach and present the results of some recent work that we have done to test the performance of an existing gateway system that serves as the middle layer of a 3-tiered client/server transaction processing application.

2. Performance Testing Objectives

In this paper, when we speak of software performance testing we will mean all of the activities involved in the evaluation of how the system can be expected to perform in the field. This is evaluated from a user's perspective, and is typically assessed in terms of throughput, stimulus-response time, or some combination of the two. Alternatively, performance testing could be used to assess the level of system availability. For many telecommunications or medical applications, for example, having the system always available is essential.

Within that framework, there are a number of different goals one could have when considering performance testing. These include:

1. The design of test case selection or generation algorithms specifically intended to test for performance criteria rather than functional correctness criteria.
2. The definition of metrics to assess the comprehensiveness of a performance test case selection algorithm for a given program.
3. The definition of metrics to compare the effectiveness of different performance testing strategies for a given program.
4. The definition of relations to compare the relative effectiveness of different performance testing strategies in general. This requires that we are able to say in some concrete way what it means for this performance testing strategy to be better than that one.
5. The comparison of different hardware platforms or architectures for a given application.

There are also a number of things that need to be measured when evaluating a software system's performance. Included among these are: resource usage, throughput, stimulus-response time, and queue lengths detailing the average or maximum number of tasks waiting to be serviced by selected resources. Typical resources that need to be considered include network bandwidth requirements, CPU cycles, disk space, disk access operations, and memory usage [4]. Other resources that might be of importance for specific projects include switch usage (for telecommunications projects) or database access rates.

Note that it is sometimes impossible to satisfy all the requests for resources simultaneously. If the total "requirements" for disk space by different processes exceeds the space available, this problem should be identified as early as possible in the requirements phase. In some cases the system will have to be re-architectured. This can be a very expensive process with severe consequences. In other cases, processes will simply have to cut back on their resource demands, perhaps by developing better algorithms or by reducing functionality. Another alternative might be to order additional hardware to accommodate demands. In any case, the earlier there is an awareness of potential performance problems, the more likely that an acceptable solution can be developed, and the more economically any necessary re-architecture work can be done. Although performance testing may be expensive and time consuming, we believe that it is nonetheless likely to be cost-effective for the reasons outlined above.

3. Creating Benchmarks

One traditional way to do performance testing is to develop a *benchmark*: a workload that is representative of how the system will be used in the field, and then run the system on those benchmarks. The system's behavior on the benchmarks in considered indicative of how the system will behave in the field. Of course, the notion of a representative workload is itself problematic for several reasons. The first problem is the issue of where that data is to come from. How will we know what a representative workload actually is? This is one of the easier issues for us to resolve. In our (telecommunications) environment, many projects routinely monitor system traffic, thereby providing the tester with the resources to develop a so-called operational profile that describes how the system has historically been used in the field and is therefore likely to be used in the future. An *operational profile* is a probability distribution describing the frequency with which selected "important" operations are exercised. This typically requires that a domain expert makes the decision regarding which operations are actually central for a given application. It also requires that someone make decisions about the window of observation from which the data will be drawn. If no earlier version of this system is available, or historical usage data has not been collected for the system, there is frequently a similar system that could provide guidance in the design of the benchmark. Alternatively, if there is a system whose output becomes the input to the system under consideration, and this older system is monitored, the required data may thereby be available.

A second issue that must be considered is whether the benchmark should reflect an average workload or a very heavy or stress load. In either case, it is again necessary to consider the window of observation from which the average or stress loads will be taken. Will we look at the average traffic during a one hour window, during a 24-hour period, or during some larger period? Will we distinguish between times of light load and heavy load if this is predictable? Will we look at the heaviest load that occurred during a given 24-hour period, or the heaviest load occurring during the month or during a year? These would almost surely not all be the same. For many telecommunications systems it is usual to consider the period of repeatability to be one week. Thus, except for special occasions such as holidays or natural (or unnatural) disasters, the system usage on a given day of the week will look almost identical to the same day of the week during the following week. In [2], system traffic data was provided for a 24-hour weekday for five successive weeks. Examination of those graphs is consistent with this observation. The five graphs are very similar, having the same shape and same "busy-hour" each week.

Once the answers to these questions have been determined, the project must then determine whether or not the system that is to be performance tested will typically run in isolation on the hardware platform. In our environment, it is common for a project to be the sole application running in the environment. Therefore, provided that a representative load has been selected as the benchmark, the observed throughput or response time is an accurate reflection of the behavior that the user can expect to see. However, if the software system is likely to be running with other unrelated systems on a common platform, it is necessary to assure that the benchmark is run along with a representative background workload. Another way of thinking of this is to say that a benchmark should include a representative portrayal of the workload of the system under test, as well as a representative portrayal of the workload that is likely to be resident with the system when it is operational. The more complex the mix of jobs typically running on a system, the more difficult it is to accurately measure the expected performance.

Another issue to be considered is whether an average event arrival rate will be used, or whether a more accurate probability distribution describing the variation in arrival rate times will be used. Although this probability distribution might have the same average value, it would represent a significantly more accurate picture of the system's behavior. However, it will be much more costly to determine, and the mathematics involved in dealing with it will be more complex.

4. The Role of Requirements and Specifications in Performance Testing

In order to do software performance testing in a meaningful way, it is also necessary to have performance requirements, provided in a concrete, verifiable manner. This should be explicitly included in a requirements or specification document and might be provided in terms of throughput or stimulus-response time, and might also include system availability requirements. Frequently, unfortunately, no such requirements are provided which means that there is no precise way of determining whether or not the performance is acceptable. In addition, one of the most serious problems with performance testing is making sure that the stated requirements can actually be checked to see whether or not they are fulfilled. Just as it is not very useful to select inputs for which it is not possible to determine whether or not the resulting output is correct when testing a software system for correct functionality, when doing performance testing, it is just as important to write requirements that are verifiable. It is easy enough to write a performance requirement for a compiler such as: Every module must be compilable in less than one second. Although it might be possible to show that this requirement is not satisfied by compiling a module that takes longer than one second to compile, the fact that the compiler has been tested on many modules, all of which compiled correctly in less than one second, does not guarantee that the requirement has been satisfied. Thus, even plausible-sounding performance requirements might be unverifiable.

A more satisfactory type of performance requirement would state that the system should have a CPU utilization rate that does not exceed 50% when the system is run *with an average workload.* Assuming that a benchmark has been created that accurately reflects the average workload, it is possible to test whether or not this requirement has been satisfied. Another example of a verifiable performance requirement might be that the CPU utilization rate must not exceed 90% even when the system is being run with a stress load equal to the heaviest load ever encountered during any monitored period. Similarly, it is possible to test for things like maximum queue length or total throughput required under specified loads.

Since performance requirements must be included for average system loads and peak loads, it is important to specify those as early as possible, preferably in the requirements document. As discussed above, detailed operational profile data is frequently available, given that analysis of recorded system traffic data is performed. This is frequently a significant task that needs to be planned for, but one that is extremely important both for functionality testing and performance testing. If detailed traffic data is not available for the system or for newly-implemented features, then estimates should be made by the most knowledgeable team members. If a business case has been made for the addition of new features, then that might include the necessary estimates. Of course, as additional information is acquired, a refinement of this data should be made.

5. Previous Work

In an earlier paper [1], a software performance testing approach was presented. The goal in that work was to compare the performance of an existing production platform and a proposed replacement architecture to assess whether the new platform would be adequate to handle the required workload. The traditional approach to such a comparison is to develop software for the proposed platform, build the new architecture, and collect performance measurements on both the existing system in production and the new system in the development environment.

In [1], in contrast, Avritzer and Weyuker introduced a new way to design an application-independent workload for doing such a performance evaluation. It was determined that for the given project, the quantity of software and system availability requirements made it impossible to port the system to the new platform in order to do the performance testing. Therefore, a novel approach was designed in which no software was ported at all. Instead, a synthetic workload was devised by tuning commercially-available benchmarks to approximate resource usage of the actual system.

Motivating this work was the insight that, given that the project had systematically collected system usage data, this information represented a major asset that could be exploited. In particular, it was not necessary to run the actual system on the new platform, only to run software that behaved the way the system did from the point of view of resource usage. It was recognized that it was totally irrelevant what the software was actually doing, provided that it used resources in ways similar to the software to be ported. The result of testing the software using this approach was a decision not to port the system to the proposed platform because the testing approach indicated that although the new platform would be adequate to handle average workloads, it would not be able to handle peak loads that the system was likely to encounter.

This technique proved to be a very efficient way to arrive at this decision, and saved the project significant resources both in terms of personnel costs for porting the

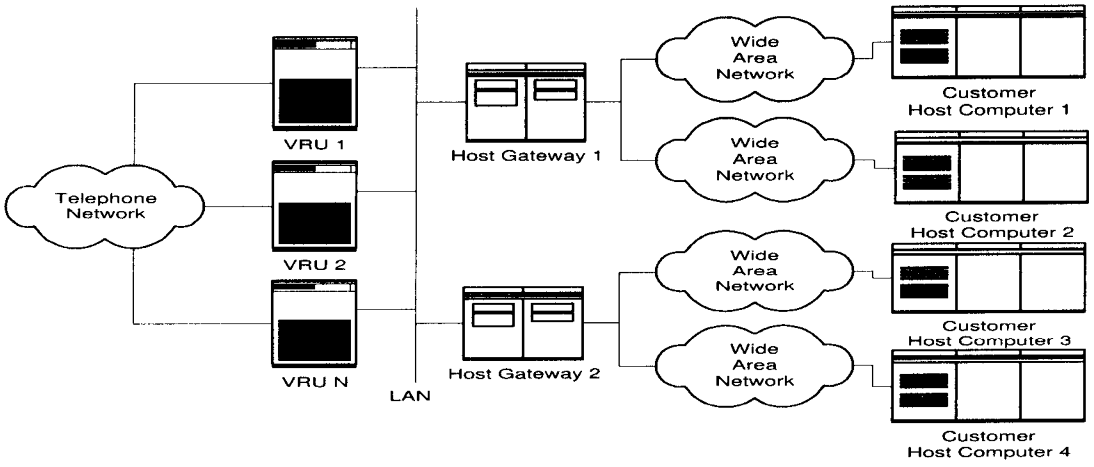

Figure 1. High Level System Architecture

software that would ultimately have had to have been back-ported to the existing platform, plus very significant savings made by not purchasing what this testing approach determined would have been inappropriate (and very expensive) hardware for the system. In what follows we will discuss a related approach to software performance testing that we have found useful when testing the performance of an existing system redeployed on a new platform.

6. A Case Study

In this section, we describe how we used our performance testing approach to test a gateway system that is the middle layer of a 3-tiered client/server transaction processing application. The system accepts input from a caller, and returns information that resides on a host server. The input may be in the form of dual tone multifrequency signals (touch tones) or limited vocabulary speech. Output is in the form of prerecorded digitized or computer generated speech.

The client or first tier of this architecture consists of Voice Response Units (VRUs). These are computer systems that terminate the call. The server or third tier consists of mainframe host computers that contain the information requested by end-users. The middle layer or second tier consists of the gateways: computer systems that allow the clients to interface with the servers. To be able to communicate with a variety of host servers, a gateway supports three different network protocols: SNA/3270, TN3270 and TCP/IP.

From an architectural standpoint, the purpose of the gateway is to allow concentration of network connections and to off-load some of the processing that would otherwise have to be performed by each VRU. The gateways and VRUs share a LAN. The host servers are remote systems. One gateway provides service to many VRUs. Conversely, a VRU may access the services provided by multiple gateways. One gateway may also connect to multiple remote hosts. The system architecture is shown in Figure 1.

An Application Script running on the VRU interacts with the end-user. When the application needs to perform a host transaction, an inter-process communications (IPC) message is sent to the appropriate server process running on the gateway. For our purposes, a transaction is defined as a request sent to the host and the reply received from the host. The server process on the gateway formats the request according to the host requirements and sends it to the host. When a reply is received, the process on the gateway parses the returned host data, formats an IPC message and sends the reply to the application on the VRU. The high level software architecture is shown in Figure 2.

The existing gateway systems are based on PC hardware (Intel Pentium processors, ISA bus) running the UNIX operating system. To improve performance, reliability, and to reduce maintenance costs, it was decided to upgrade the hardware platform with mid-range systems that use multiple PA RISC processors, have large amounts of RAM, and are capable of supporting many physical I/O interfaces.

7. Performance Testing Objectives

When the new platform was purchased, the only available information regarding its required performance came from an analytical study done by the project team. This study estimated the expected number of transactions that should be processable by the new platform when it is connected to host systems on the SNA/3270 and TN3270 protocols. It determined that 56 links, each capable of

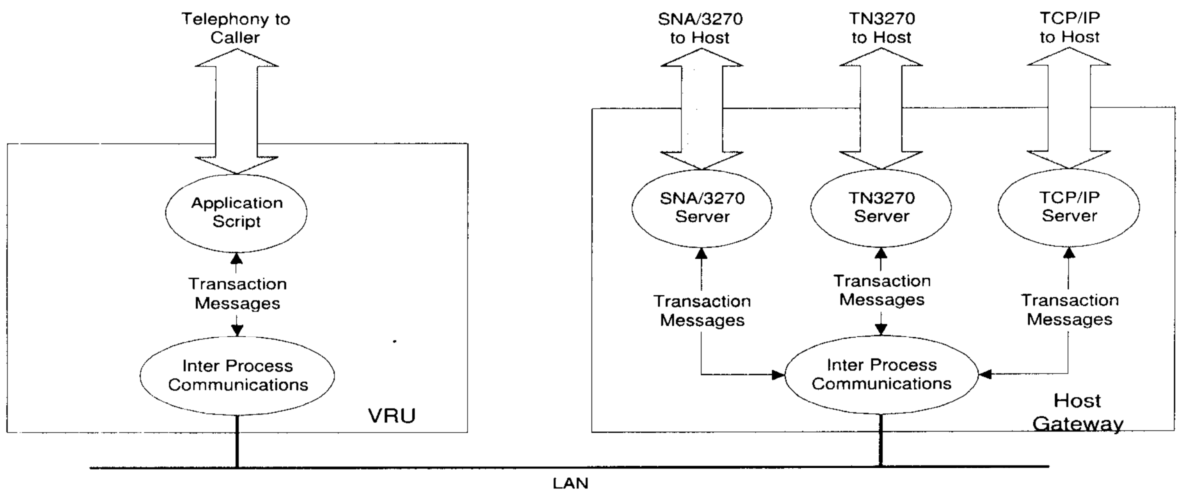

Figure 2. High Level Software Architecture

handling 56Kbps were needed, and that the system would have to function with 80% of the links active, each handling transactions averaging 1620 bytes. Although the vendor had given assurances that the new platform would be able to handle the required workload, they did not provide adequate data to substantiate these claims. What they did supply was performance data from the use of these systems as database servers. However, a gateway system provides functionality that is qualitatively different than the functionality provided by a database server and therefore we felt that the performance data supplied by the vendor would not be indicative of the behavior that we could expect in our target environment.

Given that the new platform contained new software components whose performance behavior was largely unknown, and given that the performance data supplied by the vendor was not directly applicable to a system used as a gateway, the project team decided to test the performance of the gateway prior to deployment with a configuration similar to the one that will be used in production. An important first step was to identify the objectives of the performance testing activity. Initially we thought that the objective should be to validate the requirements set by the analytical study. That is, to directly address the question of whether or not the gateway could effectively handle the expected number of transactions per second. After careful consideration, however, we decided that performance testing should help us answer the following questions:

1. Does the new gateway have enough resources to support increased end-user volume within pre-specified budgets of resource consumption and latency?
2. At what volume of end-users do these resources become inadequate?
3. What components of the hardware and software architecture of the gateway limit performance? Where are the bottlenecks?

8. Designing the Performance Test Cases

As we mentioned earlier, the project team decided that it would be best to do performance testing on a configuration similar to the one used in production. To accomplish this, we had to borrow testing facilities from the vendor. Access to these facilities was strictly limited to one week. As a result, test cases had to be carefully planned.

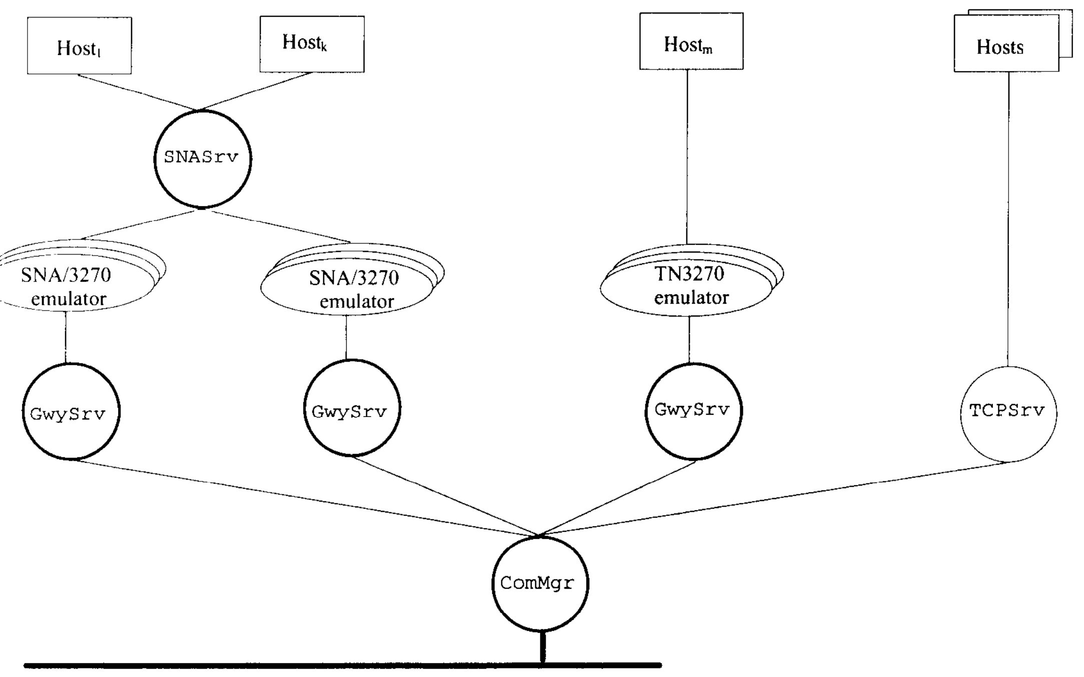

Figure 3. Gateway Software Architecture

It was decided that our performance testing had to address the questions listed in the last section for both average and peak workloads. This meant that we had to determine what these loads were. Experience told us that a usage scenario should be defined in terms of those parameters that would most significantly affect the overall performance of the system. Therefore, our immediate priority was to identify such parameters. To do this, we first looked at the software architecture of the gateway and its relationship to the VRUs and to the customer host systems. We felt that the software architecture of a system [3] could provide important insights that could be useful when designing our performance test suite.

By studying the software architecture of the gateway, we observed that its performance is closely related to the performance of three processes. The first process was a new software component that did not exist on the old platform. There is one such process on the gateway and its purpose is to communicate on one end with all 3270-based hosts and on the other end with the different SNA/3270 emulators. The second process was a modified version of one found on the old platform. Modifications were made so that this process could work with the new process mentioned above. No additional functionality was incorporated. There is one of these processes for each link to a host and each of these processes may interact with up to 25 different emulators. Each emulator supports a maximum of 10 Logical Units (LUs).

The third process interfaces directly with the VRU. There is one of these processes on the gateway and it has been ported without any modifications from the old platform. Since the first process deals with connections to customer host systems and with processes that communicate with the end-user, and the second also deals with those processes that communicate with the end-user, we reasoned that the number of simultaneous active connections to customer host systems and the number of simultaneous callers should be two of the parameters. Additional parameters that would noticeably affect the overall performance of the gateway were the number of transactions per call, transaction sizes, call holding times, call interarrival rate and traffic mixture (that is, percent of SNA vs. TN3270 vs. TCP/IP).

To determine *realistic* values for these parameters, we collected and analyzed data from the current production gateways. We used tracing to record certain types of information for each transaction. These included such things as time-stamps, response time, and LU number. We did this for several days with 6 different applications at two different sites. The data gave us a good feel for the usage levels. Based on this information and using some estimated values from past experience, we determined that:

- *Average* usage was typically about forty LUs per link, with 24 simultaneous callers, 3 transactions per call, 2.5 minute call holding time, and a 5 second call interarrival rate. This corresponds to approximately one transaction every three seconds for each link.
- *Heavy* usage was determined to be one hundred twenty LUs per link, with 120 simultaneous callers, 4 transactions per call, 3.5 minute call holding time, and a 5 second call interarrival rate. This is equivalent to approximately 2.2 transactions per second for each link.

We were unable to obtain precise data regarding transaction sizes and traffic mixture. For transaction sizes, the only available information we had was that the minimum and maximum sizes were 100 and 1920 bytes respectively. As a first approximation, our solution was to run the test cases for both usage scenarios assuming that each transaction was 1100 bytes (100 bytes sent to the host and 1000 bytes received from the host). This was the number assumed in the analytical study and, based on experience, it seemed to be a reasonable assumption. Alternatively we could have assumed a non-uniform distribution of transaction sizes such as assuming that different segments of the transaction population would have different transaction sizes. For example, for the typical usage scenario we could assume that 25% of the transactions were 100 bytes, 25% were 1920 bytes and the remaining 50% were equally distributed to represent transactions of 500 bytes, 1000 bytes, and 1500 bytes. For the heavy usage scenario we could assume that 25% of the transactions were 1000 bytes, 25% 1500 bytes, and 50% 1920 bytes.

With respect to the traffic mixture, our primary concern was the SNA/3270 protocol. All the traffic on the existing gateway used the SNA/3270 protocol. Support for the TN3270 was new and the system engineers anticipated that, at least initially, only a very small number of customer host systems would be using that protocol. Furthermore, both the TN3270 and the SNA/3270 used some of the same processes, so we did not expect any performance surprises from TN3270 traffic. As for the TCP/IP protocol, its implementation does not require significant resources on the gateway and we already had data that indicated that the effect of the TCP/IP protocol on the overall performance of the gateway was negligible. Thus, we decided to focus our testing effort on the SNA/3270. Had the sensitivity of the traffic distribution between SNA/3270 and TN3270 been significant, we would have created different test cases by varying the distributions by starting with SNA/3270 at 90% and TN3270 at 10% and decreasing and increasing the distributions by 10% until SNA/3270 was at 10% and TN3270 at 90%. These nine tests, ordered in decreasing order of importance, would be run on both the average and high usage scenarios, and would provide us with valuable insights on the role that each protocol plays in the consumption of resources as well as the overall effect on the performance of the gateway.

9. Performance Testing and Results

To implement the test cases representing these scenarios, the project team developed two programs to simulate callers. These programs send transaction request messages to the gateway server process and receive replies. Both programs simulate a number of simultaneous callers, performing transactions of a size specified on the command line. One program sends transaction requests as quickly as possible, i.e., as soon as the reply to a request is received, another request is sent out. The other program more closely simulates human users by accepting parameters for call duration and number of transactions per call, and using them to add "think time" and simulate realistic call arrival rates. For a host application, we used an existing simple application that had been used in the past for functionality testing. Since all customer applications require similar processing cycles by the gateway, the choice of customer application was not important for our testing.

Testing was conducted on a hardware configuration similar to the one used on the field and revealed some very surprising results regarding the performance of the gateway. We learned that for both the average and heavy usage scenarios, the gateway could handle only a small fraction of the transactions estimated to be required by the analytical study. This was disturbing since we had been assured by the vendor that the hardware was capable of handling this load. Luckily, we were able to quickly identify three problems that caused very high CPU utilization and contributed to the poor performance of the gateway. The first problem was related to the polling frequency of the gateway by the host. The second problem was related to the algorithm used by the gateway server process to search the various LUs for the arrival of new data. The third problem was related to the inefficient implementation of some library modules used by the SNA server process. The first two problems were resolved fairly quickly and we were able to observe substantial improvement in the performance of the gateway. The solution to the third problem is much more subtle and involves software modifications that must be carefully designed. It is expected that when completed, these modifications will further improve the performance of the gateway.

10. Conclusions

For many industrial software systems, performance requirements play a central role in determining the overall quality of the system. Many project teams expend a great deal of resources testing the functionality of the system, but spend little or no time doing performance testing. The issues that have to be addressed when doing performance testing differ in a number of ways from the issues related to functionality testing. In this paper we have presented many of these issues and discussed possible approaches for dealing with them.

We have also described our experience doing performance testing of an existing gateway system that was being redeployed on a new platform. We saw clearly that doing performance testing before the system was released to the user community was a wise investment of resources. We were able to uncover and correct several software faults that substantially compromised the performance behavior of the system. We also saw the need to clearly identify the objectives of performance testing before beginning the actual test case selection. This will allow testers to identify unrealistic or untestable goals. We found that the stated performance objectives had to be refined and achieved this through many discussions with the project team.

We used the software architecture of the system whose performance we wanted to evaluate, as a way of identifying the parameters that affected the performance most directly. This allowed us, for example, to save significant time by focusing our testing effort on the SNA/3270 traffic. We concluded that this was acceptable because we had determined that almost all of the traffic used the SNA/3270 protocol, rather than the TN3270 protocol. In addition, we had determined that the software for the two protocols were almost identical. Since we had very limited access to the test platform, optimizing the test suite was essential.

Finally, we defined usage scenarios by assigning realistic values to the parameters that most directly affected performance. For most parameters we were able to collect and analyze data from systems currently in use in the field. For a few parameters we made assumptions based on our experiences with the existing system. We benefited greatly from the availability of a production-like test facility and the tools provided by the vendor to collect and analyze performance data. The primary lesson learned, however, was that it was far better to uncover performance problems before the system was deployed with customers dependent on it, than finding out about them when users' needs cannot be met.

11. Acknowledgments

Ron DeBlock helped us understand the application and the software architecture of the gateway. He implemented the caller simulation programs, executed the test cases, and collected the performance results.

12. References

1. Avritzer, A., and E.J. Weyuker. "Deriving Workloads for Performance Testing", *Software -- Practice and Experience*, Vol. 26, No. 6, June 1996, pp. 613-633.
2. Avritzer, A., and E.J. Weyuker. "Monitoring Smoothly Degrading Systems for Increased Dependability", *Empirical Software Eng. Journal*, Vol. 2, No. 1, 1997, pp. 59-77.
3. Shaw, M., and D. Garland. Software Architecture - Perspectives on an Emerging Discipline. Prentice-Hall, 1996.
4. Smith, C. U. Performance Engineering of Software Systems. Addison-Wesley, 1990.