QSEM[SM]: Quantitative Scalability Evaluation Method

Lloyd G. Williams, Ph.D.
PerfX
2345 Dogwood Circle
Louisville, Colorado 80027
(720) 890-8116
www.perfx.net

Connie U. Smith, Ph.D.
Performance Engineering Services
PO Box 2640
Santa Fe, New Mexico, 87504-2640
(505) 988-3811
www.perfeng.com

While scalability is important to today's software applications, few organizations understand how to quantitatively evaluate their software's scalability. This paper describes the Quantitative Scalability Evaluation Method, QSEM. QSEM uses straightforward measurements to quantify the scalability of a software application. The results provide an understanding of the application's scalability that makes it possible to extrapolate behavior to larger configurations with confidence. The seven steps of the QSEM method are described and illustrated with a case study.

INTRODUCTION

Scalability is one of the most important qualities of today's software applications. As businesses grow, the systems that support their functions also need to grow to support more users, process more data, or both. As they grow, it is important to maintain their performance (responsiveness or throughput) [Smith and Williams 2002]. Poor performance in these applications often translates into substantial costs. Customers will often shop elsewhere rather than endure long waits. Slow responses in CRM applications mean that more customer-service representatives are needed. And, failure to process financial trades in a timely fashion can result in statutory penalties as well as lost customers.

Despite its importance, scalability is poorly understood and few organizations understand how to quantitatively evaluate an application's scalability. As a result, they often make assumptions about the scalability of their software. If wrong, these assumptions can be costly.

This paper describes the Quantitative Scalability Evaluation Method, QSEM[SM]. A QSEM evaluation provides the information needed to quantitatively predict the scalability of an application system. It allows you to determine whether you can meet your scalability requirements and, if there is more than one feasible alternative, provides the information you need to select the one that best meets your overall objectives.

Scalability is a *system* property; however, the software architecture is a key factor in achieving scalability. For example, if the software architecture is not able to use additional resources to increase throughput, the system will not be scalable. The choice of execution environment is also very important—the same workload executed on two different platforms can exhibit significantly different scalability properties [Williams and Smith 2004].

QSEM uses straightforward measurements of maximum throughput at different numbers of processors or nodes. The results of the data analysis provide an understanding of the application's scalability that makes it possible to extrapolate behavior to higher numbers of processors or nodes with confidence.

QSEM may be applied as a stand-alone method for evaluating application scalability. It is also employed as part of the PASA[SM] approach to the performance assessment of software architectures [Williams and Smith 2002] where scalability is a concern and the required measurements can be obtained.

We begin with an overview of scalability. This is followed by a description of the QSEM method. We then illustrate the application of QSEM with a case study.

SCALABILITY OVERVIEW

Despite its technical and economic importance, there is no generally accepted definition of scalability. In this paper, we will use the following definition [Williams and Smith 2004]:

[SM] QSEM and PASA are service marks of PerfX and Performance Engineering Services.

> Scalability is a measure of an application system's ability to—without modification—cost-effectively provide increased throughput, reduced response time and/or support more users when hardware resources are added.

An application may be scaled to provide more throughput, handle more users and/or reduce response time.

Scaling Strategies

Applications are scaled by adding resources to remove a bottleneck. The bottleneck could, in principal, be any resource: CPU, disk, network, and so on. The bottleneck could also be a software resource, such as a thread. However, the term "scaling" is most commonly used to mean adding processors.

It is possible to distinguish two basic scaling strategies:

- *Vertical scaling* (*scale-up*)—resources (CPUs, memory, disks) are added to a single server to increase capacity
- *Horizontal scaling* (*scale-out*)—additional nodes or servers are used to increase capacity

As demonstrated in [Williams and Smith 2004], these strategies can yield very different results depending on the application. They may also have very different cost functions.

Removing one bottleneck may cause another to emerge. For example, we may be able to increase capacity by adding processors to a system but discover that, at some point, the load causes another resource, such as a disk, to become the bottleneck. At this point, our scaling strategy must shift to address this secondary bottleneck. An example illustrating the effects of a secondary bottleneck is presented in [Williams and Smith 2004].

Categories of Scalability

We recognize three categories of scalability. This classification scheme is similar to that proposed by Alba for speedup in parallel evolutionary algorithms [Alba 2002].

- *Linear scalability*—the capacity (as measured by throughput) of the system with *p* processors is *p* times the capacity with one processor. That is, doubling the number of processors will double the system's throughput.
- *Sub-linear scalability*—the capacity of the system with *p* processors is less than *p* times the capacity with one processor. That is, doubling the number of processors does not double the system's throughput.
- *Super-linear scalability*—the capacity of the system with *p* processors is more than *p* times the capacity with one processor. That is, doubling the number of processors more than doubles the system's throughput.

These three categories of scalability are illustrated in Figure 1.

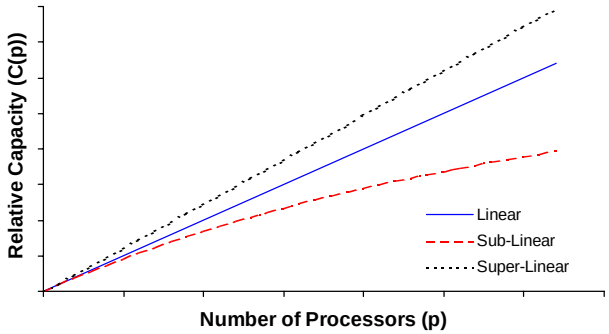

Figure 1: Categories of Scalability

Linear Scalability Linear scalability can occur if the degree of parallelism in an application is such that it can make full use of the additional resources provided by scaling. For example, if the application is a data acquisition system that receives data from multiple sources, processes it and prepares it for additional downstream processing, it may be possible to run multiple streams in parallel to increase capacity. In order for this application to scale linearly, the streams must not interfere with each other (for example via contention for database or other shared resources) or require a shared state. Either of these conditions will reduce the scalability below linear.

Sub-Linear Scalability Sub-linear scalability occurs when the system is unable to make full use of the additional resources. This may be due to properties of the application software, for example if delays waiting for a software resource such as a database lock prevent the software from making use of additional processors. It may also be due to properties of the execution environment that reduce the processing power of additional processors, for example: overhead for scheduling; contention among processors for shared resources, such as the system bus; or communication among processors to maintain a global state. These factors cause the relative capacity to increase more slowly than linearly.

Super-Linear Scalability At first glance, super-linear scalability would seem to be impossible—a violation of the laws of thermodynamics. After all, isn't it impossible to get more out of a machine than you put in? If so, how can we more than double the throughput of a computer system by doubling the number of processors?

The fact is, however, that super-linear scalability is a real phenomenon. The easiest way to see how this

comes about is to recognize that, when we add a processor to a system, we are sometimes adding more than just a CPU. We often also add additional memory, disks, network interconnects, and so on. This is especially true when expanding clusters (horizontal scaling). Thus, we are adding more than just processing power and this is why we may realize more than linear scaleup. For example, if we also add memory when we add a processor, it may be possible to cache data in main memory and eliminate database queries to retrieve it. This will reduce the demand on the processors, resulting in a scaleup that is more than can be accounted for by the additional processing power alone.

For more details on these categories of scalability and a discussion of models that exhibit these types of behavior, see [Williams and Smith 2004].

Models of Scalability

Currently, we use four models to analyze and predict the scalability of applications:

- Linear scalability
- Amdahl's Law
- Super-Serial Model
- Gustafson's Law

The Super-Serial Model is an extension of Amdahl's Law and has appeared in the context of on-line transaction processing (OLTP) systems [Gunther 2000], in beowulf-style clusters of computers [Brown 2003] and others. Amdahl's Law and Gustafson's Law were developed in the context of speedup for parallel algorithms and architectures. Amdahl's Law and the Super-Serial Model, describe sub-linear scalability. Gustafson's Law, describes super-linear scalability. Williams and Smith discuss these models and illustrate their utility for Web applications [Williams and Smith 2004].

We note that, due to their complexity, it is likely that some systems will not conform to any of these models. It is therefore important to determine—via analysis of measured data—that a given system actually follows a known model before making decisions based on predicted scalability.

Enhancing Scalability

Scalability isn't just a hardware issue. While adding processors is the most common approach to scaling an application, it may not always be the most cost-effective. It is often possible to increase the scalability of an application through simple modifications to the software architecture. For problem systems, this may be the only way to achieve scalability requirements. Software approaches to enhancing scalability include:

- reducing the demand at the bottleneck resource,
- increasing the degree of parallelism in the application,
- reducing contention for shared resources, and
- reducing the effects of interprocessor communication.

The choice of whether to use a hardware or software approach—or a combination of the two—to achieving scalability requirements will depend on several factors. A quantitative evaluation of the application's scalability provides a starting point for weighing these alternatives.

The next section provides an overview of the QSEM method. The following section illustrates the application of QSEM with a case study.

THE QSEM METHOD

QSEM is a model-based approach to quantitatively evaluating the scalability of Web-based applications and other distributed systems. The analysis uses readily-obtained data from straightforward measurements of throughput at different numbers of processors or nodes. The results provide an understanding of the application's scalability that makes it possible to extrapolate behavior to higher numbers of processors or nodes with confidence. Different configurations can be measured to study the effectiveness of different scaling strategies (e.g., vertical versus horizontal scaling). The most effective strategy can then be selected based on technical feasibility as well as other needs of the organization.

The QSEM method consists of seven steps. The steps are typically performed in the order given. In some cases, however, the order may be varied and some steps can be conducted concurrently. For example, some measurement planning steps do not require that all the previous steps be complete to begin. Also, discovery of new information in one step often requires revisiting a previous one, so iteration is possible.

The seven steps of the QSEM method are:

1. *Identify critical Use Cases*—Identify the externally visible behaviors of the software that are critical to responsiveness or scalability.
2. *Select representative scalability scenarios*—For each critical Use Case, identify the scenarios that are important to scalability.
3. *Determine scalability requirements*—Identify precise, quantitative, measurable scalability requirements.

4. *Plan measurement studies*—Identify the bottleneck resource, plan measurements, develop load generator scripts, determine what parameters to measure, identify needed measurement tools, and document the test plans.
5. *Perform measurements*—Conduct the measurement experiments, collect data, and document the results.
6. *Evaluate data*—Evaluate the measurement data to determine whether the scalability requirements can be met and select the best scaling strategy.
7. *Present results*—Present results and recommendations to stakeholders.

The first three steps are concerned with gathering the information needed to perform the measurement studies and evaluate their results. The next two steps address planning and carrying out the required measurements. The last two steps focus on evaluating the measured data and presenting the results to stakeholders.

Once the results have been presented, stakeholders can use this information to select the scaling strategy that best meets their needs.

The following sections describe each step in more detail.

Step 1: Identify Critical Use Cases

Use Cases describe externally visible behaviors of the software. From a scalability perspective, the critical Use Cases are those that describe the most common, or typical, uses of the application. Critical Use Cases may also include functions that are executed infrequently but have high resource demand [Smith and Williams 2002]. The critical Use Cases will be a small subset of the possible Use Cases.

Step 2: Select Representative Scalability Scenarios

Each Use Case consists of a set of scenarios that describe the sequence of actions required to execute the Use Case. Not all of the scenarios belonging to a critical Use Case will be important from a scalability perspective. We focus on the scenarios that are executed frequently in typical uses of the application (and/or have high demand) and thus contribute to the dominant workload.

For example, in a customer relationship management (CRM) application there may be many ways to retrieve customer account information such as: account number search, name search, zip code search, and so on. If one of these possible paths is executed only infrequently, it will not be important to scalability unless it has very large resource demand. In this case, we use the product of the probability of execution and resource demand to determine if the scenario is important to scalability. If account number is used almost exclusively to retrieve customer accounts, only this scenario will be important to scalability. If a name search is also typically used, we will need to include the scenarios for each and know the relative frequency or probability of using each alternative.

Scenarios provide the basis for writing load generator scripts. To ensure that these scripts are an accurate reflection of the behavior of the application, scenarios should be documented in as much detail as possible.

We use UML sequence diagrams [Booch, et al. 1999], [Smith and Williams 2002] to document scenarios. In an object-oriented system, a sequence diagram describes the objects (individual objects, components, or subsystems) that cooperate to perform a function and the sequence of interactions between them. For non-object-oriented systems, a sequence diagram documents the major software units that perform a function and their interactions. An example of a sequence diagram appears in the case study (Figure 2).

To construct *representative* load generator scripts, we also need quantitative information about typical execution paths. Examples of parameters that are important for constructing representative scripts are: think times between user requests, typical database queries, the probability of taking different execution paths, and so on. For example, for a CRM application we might need to know:

- the probability of executing the account number search versus the name search
- the time between steps in the scenario, such as RetrieveCustomerAccount, ViewPayHistory, and DocumentContact requests
- the characteristics of the database accesses and the amount of data displayed

We will also need to know the workload mix if multiple scenarios contribute to the dominant workload—the functions that are executed most frequently and account for most of the resource usage [Smith and Williams 2002].

These parameters may be obtained from performance measurements of the application, from measurements of user behavior (e.g., from productivity measures), and/or from interviews. We usually develop initial end-to-end scenarios through interviews with application specialists and users, then supplement this information with measurements of some of the required parameters.

Many times, particularly with legacy systems, Use Cases and scenarios have not been documented. Then, the scalability team must work with the application team and end-users to identify the important uses of the software and detail the processing steps that are executed in order to develop representative load generator scripts. The process used for eliciting this information is similar to that used for performance walkthroughs, as discussed in [Smith and Williams 2002].

With applications for which Use Cases and scenarios have not been documented, this step is often one of the most valuable outcomes of a QSEM evaluation. Documenting scenarios:

- provides developers with an end-to-end view of the application that is otherwise difficult or impossible to obtain, especially on large projects,
- facilitates validation of load generator scripts and collection of representative parameters
- aids in explaining results of the scalability evaluation to stakeholders
- helps identify modifications that will improve scalability

Step 3: Determine Scalability Requirements

Precise, quantitative scalability requirements provide a rigorous basis for evaluating various scaling strategies and selecting the one that most closely meets your needs. Scalability requirements are often determined by combining performance requirements, current workload volume, and projected growth.

For example, if our current capacity is 50 end-user transactions per second (tps) and we expect our business to grow by 10% per year over the next five years, at the end of five years we will need a capacity of:

$$C = 50(1 + 0.1)^5$$

$$C = 80.5 \; tps$$

We may plan to meet that requirement by purchasing all of the required capacity now or by adding capacity annually or at some other interval.

Scalability requirements may be expressed in several different ways, including response time, throughput, or constraints on resource usage. Sometimes, a combination of these is needed. For example, we may specify a throughput of 500 typical transactions per second with an average response time of 1 second. We might also require that CPU utilization be less than 50% to provide for failover. Note that a “typical” transaction must also be defined. The data required to do this is gathered in Step 2.

In each case, the requirement should be quantitative and measurable. Vague statements such as “the system shall handle as many users as possible” are not useful. There is no way that you can ever be sure that you have met a requirement like this. A requirement such as “the CRM system must handle at least 1,000 end-to-end CustomerPaymentInteractions per second” is much more useful for a quantitative evaluation of scalability.

It is also important to specify the conditions under which the required performance is to be achieved. These conditions are typically expressed in terms of the workload mix (when there are multiple scalability scenarios) and the expected intensity.

Step 4: Plan Measurement Experiments

This discussion assumes that you are already familiar with the fundamentals of conducting performance measurements, such as determining whether you need an isolated target platform versus sharing it with other work, installing the proper releases of software, using a representative database, etc. If you are unfamiliar with such topics, you can find this information in [Smith and Williams 2002], [Ferrari, et al. 1983], or [Jain 1990].

As mentioned earlier, the measurements that we need are maximum throughput as a function of the number of processors or nodes. We also need measurements of the utilization of significant resources (e.g., CPUs, disks, network) in the system at maximum throughput to identify potential secondary bottlenecks and their effect on scalability.

Identify the Bottleneck Resource We begin by identifying the bottleneck resource. This information drives the measurement studies. For example, if the bottleneck resource is the application server CPU, then we measure the maximum throughput as a function of the number of application server processors or nodes. On the other hand, if the bottleneck resource is the network, we don't gain much by performing experiments that vary the number of application server CPUs. We need to remove this bottleneck before proceeding, for example by upgrading the network.

It is also important to make sure that the bottleneck resource is not a software resource, such as a thread. If, at maximum throughput, the utilization of every hardware resource is low, it is likely that there is a software bottleneck such as a One-Lane Bridge antipattern [Smith and Williams 2002]. Again, this bottleneck must be removed before proceeding.

Plan Measurements To provide the best possible data for later analysis, measurements of maximum throughput should be made for at least four configurations (number of processors or nodes) for each scaling strategy under consideration. For example, you may plan to make measurements for 1, 2, 4 and 8 processors for a vertical strategy or 1, 2, 3, and 4 nodes for a horizontal strategy.

For each configuration, the test plan should include the initial number users, the increment in users, and the time interval between increments.

Develop Load Generator Scripts Use the scalability scenarios identified earlier to develop a load generator script for a representative workload mix. Important considerations in developing scripts include:

- Use representative, randomly-generated user think times (constant interarrival times will skew the results).
- Avoid using think times of zero. While this may speed up the measurements, it will result in underestimation of the number of users that the system can support.
- Scripts should reflect all of the processing steps on the resources that determine scalability (e.g., the application and database servers are typically key resources, the firewall usually is not).
- The measurement database should reflect the size and content of the production environment for the scalability horizon. In particular, it is important to access different portions of the database to avoid having the data in cache.

Determine What Parameters to Measure For each configuration to be measured, we need to know:

- the maximum throughput versus the number of processors (for vertical scaling) or the number of nodes (for horizontal scaling)
- the utilization of the CPUs and other critical resources in the system

You may want to measure other performance metrics (e.g., response times, disk I/Os, virtual memory paging, etc.), particularly when you are looking for opportunities to improve the application software.

We have also found it useful to record the number of users and throughput at several points while ramping up the load for each configuration studied. This information can be useful in validating capacity planning models.

Identify Needed Measurement Tools Once you have decided what to measure, determine the measurement tools that provide the metrics you need. Your load driver may measure all of the data you may need directly. If not, it will be necessary to run one or more other measurement tools independently to collect the missing data. If multiple tools are needed, determine the start up and shut down procedures that you need to get meaningful measurements for (only) the duration of the experiment.

It is also important to plan how you will correlate the results provided by the different tools. For example, if you are using an independent tool to measure CPU utilization, make sure that timestamps with the appropriate granularity are available so that you can accurately determine the CPU utilization at maximum throughput.

Document the Test Plans Finally, document the measurement plans and procedures so that you and your colleagues will know how to repeat the experiments without re-discovering the entire process.

Step 5: Perform Measurements

In this step you execute the measurement experiments documented in the plans and procedures formulated in the previous section, collect the data, and document the results—including the date and time—so they can be clearly identified in the evaluation step. In some volatile environments, versions of the software, the middleware, and the platform itself may vary between experiments. While it is preferable to more closely control these aspects of the environment, it is sometimes unavoidable and you should at least document the configuration information for each experiment.

Several additional considerations are related to the running of the experiments:

- Make sure the “ramp up” of users is steady and controlled and that you can correlate the number of users with data from any other tools that you use.
- Determine the maximum throughput by increasing the number of users and measuring the throughput at each point. When the throughput no longer increases (perhaps even decreases), you have reached maximum throughput.
- Run each experiment long enough to reach steady state and stay there for several measurement intervals (don’t just run it until new users are rejected).
- Repeat several experiments and measure several data points to confirm that they are repeatable, and if not identify and correct problems.

Conduct at least one trial experiment before performing the actual scalability measurements to ensure that your procedure will work and will provide the data you need for the evaluation.

Step 6: Evaluate Data
To evaluate the measured data, we use regression analysis to determine which of the scalability models best describes the application's observed behavior and estimate the model's parameters. We then use the model to predict the scalability of the application. The regression analysis is described in more detail in [Williams and Smith 2004].

The result of the analysis is a prediction, of maximum throughput versus number of processors or nodes. However, we do not really want to operate at or near maximum throughput. Or, we may have a requirement to support a specific number of users. Once the results of the scalability analysis are known, standard capacity planning models (e.g., queueing network models) can be used to size the final configuration. The case study illustrates one possibility for this type of analysis.

Step 7: Present Results
The results include:

- Use Cases and scenario documentation
- descriptions of measurements (including scenarios, workload intensities, configurations, scripts)
- summary of measured data
- summary of data analysis
- identification of feasible scaling strategies
- capacity projections for feasible strategies

If there is more than one technically feasible alternative and economic data that allows comparison of their costs is available, this comparison can also be included in the report.

These results may be in the form of a presentation, a written report, or a combination of the two.

Stakeholders then use this information to select the scaling strategy that best meets their needs.

CASE STUDY

In this section, we illustrate the QSEM process with an example. The application has been disguised to preserve confidentiality. Some details have also been simplified for presentation.

The example is a Web Service that provides credit authorization services for e-commerce sites. Customers select items on-line—possibly using a shopping cart service—and, when ready, make their purchase using a credit card. This Web Service accepts transaction authorization requests and determines whether the request is accepted or declined.

The execution environment consists of an application server and a database server. Two alternatives for the application server were considered:

- a 1.6 GHz platform capable of hosting up to 16 processors for vertical scaling
- two-way 1.6 GHz servers which are replicated for horizontal scaling

The database server is an 16-way 1.3 GHz server with RAID disks.

Step 1: Identify Critical Use Cases
There is one critical Use Case for this application—AuthorizeTransaction. A request for authorization arrives from a client e-commerce site. The request is checked against information in the database and the authorization is either accepted or rejected.

Step 2: Select Representative Scalability Scenarios
An authorization request includes card information (card number and security data) and transaction information such as amount and vendor. Card data is then retrieved from the database and authorization rules are applied. The transaction may be authorized or declined. If the transaction is authorized, it is posted to the database and an authorization code is returned to the client. Figure 2 shows the sequence diagram for this scenario.

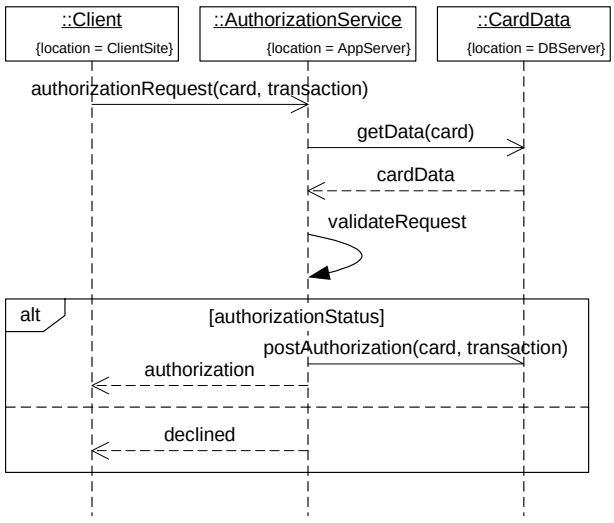

Figure 2: Credit Authorization Scenario

Note that we do not include scenarios that are not executed frequently, such as exception handling for missing or erroneous data. These scenarios do not impact scalability.

To construct a representative measurement script, we need to know the probabilities for approving and rejecting authorization requests. For the purposes of this

case study, they are 0.91 for approval and 0.09 for rejection.

Each client has multiple shoppers making purchases and therefore submits frequent authorization requests. From the point of view of the Web Service, each client appears to be a single user with a "think time" of two seconds. This is the effective interarrival time from all clients.

Step 3: Determine Scalability Requirements

Currently, the system is supporting approximately 100 clients with a maximum throughput of 35.8 authorization transactions per second. Marketing projections indicate that the system will need to scale to support 500 clients with a throughput of at least 200 transactions per second within two years. At that load, the response time must be 0.1 second or less.

Step 4: Plan Measurement Studies

To identify the bottleneck resource, the utilization of the application server and database server CPUs as well as the database server disk were measured as a function of load. Collecting this data required use of another tool and correlation of the utilization measurements with those of the load driver. The results confirmed that the application server CPU is the bottleneck resource.

It was decided to use a workstation running a commercial load driver to generate virtual users. Scripts were constructed to produce virtual users executing the representative scenario described above. A separate tool was used to measure utilizations.

The measurement plans called for investigating both horizontal and vertical scaling options. Four configurations were specified for each study. These are described in the following section.

Step 5: Perform Measurements

The results of the vertical and horizontal scaling studies are described in the following sections.

Table 1: Vertical Scalability Measurements

Number of Processors	1	2	4	8
Max tps	18.1	35.6	71.1	136.8
Appserver CPU Utilization at Max tps	.99	.99	.98	.99
DB Server CPU Utilization at Max tps	.04	.09	.18	.34
DB Server Disk Utilization at Max tps	.03	.05	.11	.21

Vertical Scaling To evaluate the vertical scaling characteristics of this application, measurements of maximum throughput (transactions per second) were obtained for a single application server with 1, 2, 4, and 8 processor configurations. This server is a 1.6 GHz platform capable of hosting up to 16 processors. Utilizations for the application server and database server CPUs as well as the database server disk were also measured. Table 1 shows the results of these measurements.

Horizontal Scaling To evaluate the horizontal scaling characteristics of this application, measurements of maximum throughput (transactions per second) were obtained for 1, 2, 3, and 4 application server nodes. Each node is a two-way 1.6 GHz platform. Again, utilizations for the application server and database server CPUs as well as the database server disk were also measured. Table 2 shows the results of these measurements.

Table 2: Horizontal Scalability Measurements

Number of Nodes	1	2	3	4
Max tps	31.8	65.1	92.6	122.9
Appserver CPU Utilization at Max tps	.99	.99	.99	.99
DB Server CPU Utilization at Max tps	.08	.16	.23	.31
DB Server Disk Utilization at Max tps	.05	.09	.14	.19

Step 6: Evaluate Data

The measured utilizations indicate that the application server CPU is the bottleneck resource in both the vertical and horizontal scaling studies.

The measured data for maximum throughput was used to construct graphs of maximum throughput versus number of processors (vertical scaling) or nodes (horizontal scaling). Regression analysis determines which of the scalability models—if any—best describes the data. Details of the analysis are discussed in [Williams and Smith 2004]. The analysis provides model parameters which are then used to extrapolate the behavior to higher numbers of processors.

Vertical Scaling Regression analysis indicates that Amdahl's Law provides the best fit to the measured data (r^2 = 0.9373). The vertical scalability of this application is therefore described by Equation 1 [Williams and Smith 2004]:

$$X_{max}(p) = \frac{X_{max}(1) \times p}{1 + \sigma(p-1)} \quad (1)$$

where: p is the number of processors

$X_{max}(1)$ is the maximum throughput with 1 processor

$X_{max}(p)$ is the maximum throughput with p processors

σ is the fraction of the workload that is performed sequentially.

The value of σ obtained from the regression analysis is 0.0082.

Figure 3 shows the measured data and the throughput modeled using Amdahl's Law with the value of σ obtained from the regression analysis.

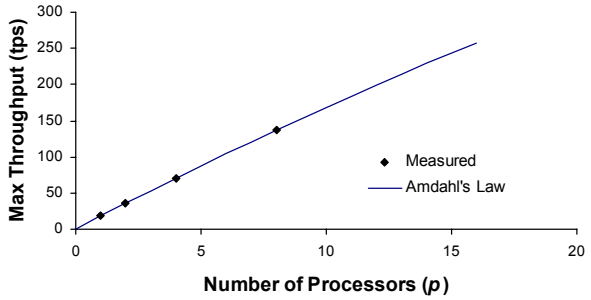

Figure 3: Measured and Modeled Data for Vertical Scaling

The Amdahl's Law extrapolation indicates that the maximum throughput with 12 processors would be 199 tps and the maximum throughput with 13 processors would be 214 tps. Thus, the required throughput of 200 transactions per second can be achieved with 13 processors.

However, at 214 tps, the system is operating at nearly one-hundred percent utilization on the application server CPU. This results in a response time of 0.43 second which is greater than the required 0.1 second.

To provide lower response times and ensure that there is some headroom for transient loads, we will size the system to operate at the "knee" of the throughput curve [Jain 1990]. This point occurs at the intersection of the vertical and horizontal asymptotes of the curve as illustrated in Figure 4.

The number of users corresponding to this point is given by :

$$N^* = \frac{D_T + Z}{D_b} \qquad (2)$$

where: N^* is the number of users at the knee of the curve

D_T is the total demand at all resources

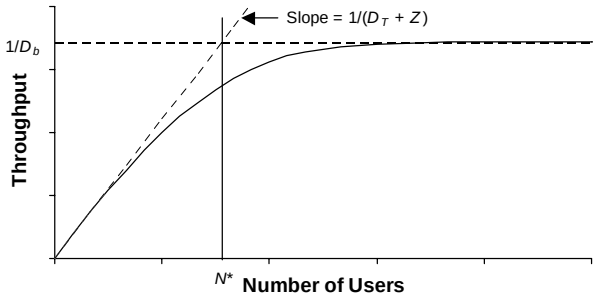

Figure 4: "Knee" of Throughput Curve

D_b is the demand at the bottleneck resource

Z is the think time.

Table 3 shows the value of N^*, throughput, and response time for various numbers of processors.

Table 3: Number of Users—Vertical

Number of Processors	Number of Users N^*	Throughput (tps)	Response Time (sec)
13	430	206	.085
14	459	220	.082
15	489	235	.081
16	517	249	.079

Table 3 indicates that, to support 500 users at a minimum of 200 tps and a response time of 0.1 second or less, we will need 16 processors.

With 16 processors, 517 users executing 249 transactions per second will yield an application server CPU utilization of 0.96. The database server CPU and disk utilizations will be 0.62 and 0.37 respectively. Thus, a secondary bottleneck is not a consideration at this time.

Horizontal Scaling Regression analysis indicates that the horizontal scaling data is best described by Linear scalability (r^2 = 0.9974). The horizontal scalability of this application is therefore described by Equation 3:

$$X_{max}(p) = X_{max}(1) \times p \qquad (3)$$

Where p, $X_{max}(1)$, and $X_{max}(p)$ have the same meaning as in Equation 1.

Figure 5 illustrates the measured data and the throughput modeled using the Linear model.

The linear model predicts that the maximum throughput with six nodes will be 186 tps and the maximum throughput with seven nodes will be 217. Thus, the

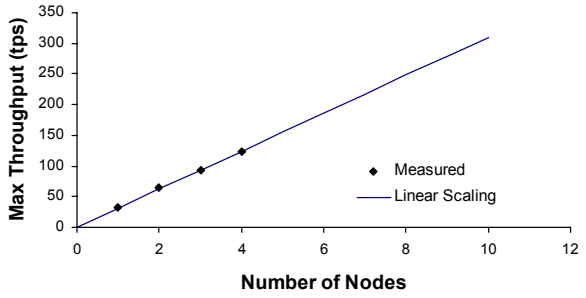

Figure 5: Measured and Modeled Data for Horizontal Scaling

required throughput of 200 tps can be achieved with seven nodes.

At 217 tps, we would be operating at near-saturation on the application servers and the response time would be 0.43 second. To meet the 0.1 second requirement, we will again size the system to operate at the knee of the throughput curve. Table 4 shows this point for various numbers of nodes.

Table 4: Number of Users—Horizontal

Number of Nodes	Number of Users *N**	Throughput (tps)	Response Time (sec)
7	436	209	.085
8	498	239	.080
9	560	269	.076

Table 4 indicates that, to support 500 users at a minimum of 200 tps and a response time of 0.1 second, we will need nine nodes.

With nine nodes, 560 users executing 269 transactions per second will yield an application server CPU utilization of 0.96. The database server CPU and disk utilizations will be 0.67 and 0.40 respectively. Thus, a secondary bottleneck is not a consideration with horizontal scaling either.

Overall Evaluation The results of the data analysis indicate that both the vertical and horizontal scaling strategies are capable of meeting the requirement of supporting 500 users with a throughput of at least 200 transactions per second. Since both scaling strategies can be used to meet the scalability requirements, the choice of strategy will depend on other criteria.

One way to distinguish between these strategies would be cost. For the vertical strategy, the hardware cost for the platform with all 16 processors is approximately $67,000. For the horizontal strategy, the cost of nine servers is approximately $72,500. Based on the hardware costs only, the vertical strategy is less costly. Costs for software licenses, system administration, and facilities are likely to increase the gap between the two strategies.

If future expansion is a consideration, we might prefer the horizontal strategy, however. With 16 processors, the vertical platform is at maximum capacity. While the current server could be replaced by a larger one or a second server of the same size could be added, the incremental cost would be high. In this case, the horizontal strategy might be preferable.

A discussion of software alternatives for improving the scalability of this application is beyond the scope of this paper. However, we note that, if the demand on the application server can be reduced substantially, the number of processors or nodes required to achieve our scalability requirement would also be reduced. In particular, for the vertical case, this would mean that fewer than 16 processors would be needed, leaving room for future expansion on that platform. Thus, if future expansion is a requirement, we might evaluate whether a combination of vertical scaling and software improvements is better than a pure horizontal scaling strategy.

The particular choice of strategy is not important for this case study. What is important is that the QSEM analysis provided the information needed to make an informed choice based on technical feasibility as well as other needs of the organization.

Step 7: Present Results

A preliminary presentation summarized the results of the analysis. This was followed by a written report that included scalability scenarios, load driver scripts and measurement procedures, a summary of the measurements and data analysis, and a discussion of the results.

Summary and Conclusions

Scalability is one of the most important qualities of today's software applications. Despite its importance, however, scalability is poorly understood and few organizations understand how to quantitatively evaluate an application's scalability. As a result, they often make assumptions about the scalability of their software. If wrong, these assumptions can be costly.

This paper has described QSEM, a quantitative approach to evaluating the scalability of applications. QSEM may be applied as a stand-alone method for evaluating application scalability. It is also employed as part of the PASA^{SM} approach to the performance assessment of software architectures [Williams and

Smith 2002] where scalability is a concern and the required measurements can be obtained.

QSEM is a model-based approach to quantitatively evaluating the scalability of Web-based applications and other distributed systems. The analysis uses readily-obtained data from straightforward measurements of throughput at different numbers of processors or nodes. The results provide an understanding of the application's scalability that makes it possible to extrapolate behavior to higher numbers of processors or nodes with confidence. Different configurations can be measured to study the effectiveness of different scaling strategies (e.g., vertical versus horizontal scaling). The most effective strategy can then be selected based on technical feasibility as well as other needs of the organization.

The QSEM process consists of the following steps:

1. Identify critical Use Cases
2. Select representative scalability scenarios
3. Determine scalability requirements
4. Plan measurement studies
5. Perform measurements
6. Evaluate data
7. Present results

These steps insure that the scalability measurements are made under conditions that are representative of the way in which that application is used and that the results of the evaluation are reproducible.

A case study illustrated the data needed for a QSEM evaluation and the types of results that are obtained. Additional details of the data analysis technique are presented in [Williams and Smith 2004].

REFERENCES

[Alba 2002] E. Alba, “Parallel Evolutionary Algorithms Can Achieve Super-Linear Performance,” *Information Processing Letters*, vol. 82, pp. 7-13, 2002.

[Booch, et al. 1999] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Reading, MA, Addison-Wesley, 1999.

[Brown 2003] R. G. Brown, “Engineering a Beowulf-style Compute Cluster,” Duke University, 2003, www.phy.duke.edu/resources/computing/brahma/beowulf_book/.

[Ferrari, et al. 1983] D. Ferrari, G. Serazzi, and A. Zeigner, *Measurement and Tuning of Computer Systems*, Englewood Cliffs, NJ, Prentice-Hall, 1983.

[Gunther 2000] N. J. Gunther, *The Practical Performance Analyst*, iUniverse.com, 2000.

[Jain 1990] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, New York, NY, John Wiley, 1990.

[Smith and Williams 2002] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*, Boston, MA, Addison-Wesley, 2002.

[Williams and Smith 2004] L. G. Williams and C. U. Smith, “Web Application Scalability: A Model-Based Approach,” *Proceedings of the Computer Measurement Group*, Las Vegas, December, 2004.

[Williams and Smith 2002] L. G. Williams and C. U. Smith, “PASA^{SM}: An Architectural Approach to Fixing Software Problems,” *Proceedings of the Computer Measurement Group*, Reno, December, 2002.