On System Scalability

Charles B. Weinstock
John B. Goodenough

March 2006

Performance-Critical Systems

Technical Note
CMU/SEI-2006-TN-012

This work is sponsored by the U.S. Department of Defense.

The Software Engineering Institute is a federally funded research and development center sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

For information about purchasing paper copies of SEI reports, please visit the publications portion of our Web site (http://www.sei.cmu.edu/publications/pubweb.html).

Contents

List of Figures

Abstract

A significant number of systems fail in initial use, or even during integration, because factors that have a negligible effect when systems are lightly used have a harmful effect as the level of use increases. This *scalability* problem (i.e., the inability of a system to accommodate an increased workload) is not new. However, the increasing size (more lines of code, greater number of users, widened scope of demands, and the like) of U.S. Department of Defense systems makes the problem more critical today than in the past.

This technical note presents an analysis of what is meant by scalability and a description of factors to be considered when assessing the potential for system scalability. The factors to be considered are captured in a *scalability audit*, a process intended to expose issues that, if overlooked, can lead to scalability problems.

1 Introduction

This technical note is the result of a Carnegie Mellon® Software Engineering Institute (SEI)-funded Independent Research and Development (IR&D) project on scalability conducted during 2004 and 2005. When proposing the IR&D, we observed that a significant number of systems fail in initial use, or even during integration, because factors that have a negligible effect when systems are below a certain level of use have a harmful effect as the level of use increases. Although this scalability problem is not new, the increasing size of U.S. Department of Defense (DoD) systems (increasing code size, number of users, scope of demands, and the like) makes the problem more critical today than in the past. The purpose of the IR&D project was to determine whether there were common patterns and issues that could be addressed to avoid scalability problems. To gather information about such problems, we surveyed the literature and conducted interviews with people who had experience in dealing with large systems.

This note presents our findings: an analysis of what is meant by scalability and a description of factors to be considered when assessing the potential for system scalability. The factors to be considered are captured in a *scalability audit*, a process that is intended to expose issues that, if overlooked, could lead to scalability problems.

1.1 Defining Scalability

Our review of the literature showed two main uses of the term *scalability*:

1. Scalability is the ability to handle increased workload (without adding resources to a system).
2. Scalability is the ability to handle increased workload by repeatedly applying a cost-effective strategy for extending a system's capacity.

For both definitions, the term system usually refers to the combination of computing hardware and software. Because the human effort to administer a system can become significant as systems get very large, we consider humans to be a component in the systems under consideration. We'll discuss both definitions; however, the second presents more interesting issues and will be the focus of this note.

® Carnegie Mellon is registered in the U. S. Patent and Trademark Office by Carnegie Mellon University.

1.1.1 The Informal Concept: Ability to Handle Increased Workload

Our first definition reflects a limited, rather informal, use of the term scalability, in which the concern is only that a system continues to perform adequately as its workload grows. Typically when the term is used in this way, no particular attention is given to how a system will have to be modified when the load grows too big. In our review of research on scalability, we found that the first definition seemed to be what people had in mind when they were using the term informally, although sometimes even a formal definition seemed to reflect just the concern that a system perform well as its workload increases. We found usage consistent with our first definition in 4 papers out of the nearly 20 papers we examined.

For example, in a paper on simulation languages, Boesel said, "The speed and scalability of [the simulation language] SLX are essential to the approach, which would otherwise be impractical" [Boesel 03]. Boesel argued that the language was scalable because a simulation model containing 500,000 objects could run in 9 seconds on a 2 gigahertz (GHz) personal computer (PC) when 250,000 objects were active. The only evidence for scalability was the size and speed of the example model (i.e., "good" performance on a large problem was evidence of scalability).

In two papers concerned with processes for administering databases, Rosenthal used scalability in the following ways:

> [Each person making security policy decisions gets information from well-known providers.] … The tight coupling [between information receivers and information providers] inhibits scalability and change [Rosenthal 03].
>
> *Scalability* refers here to the *data administration* difficulties in creating and maintaining large systems (not to runtime performance) [Rosenthal 01].

In the first paper ([Rosenthal 03]), Rosenthal examined how a certain technical approach would decrease the administrative workload of updating a database as the database grew. A more efficient way of exchanging and updating security information would allow each administrator to do more work and, therefore, handle a larger workload. He argued, in essence, that a more efficient process improved the scalability of the task. Rosenthal's examination is an example of the first definition: the administrative process would permit an administrator to handle a bigger workload; hence, the system would be scalable.

In the second paper ([Rosenthal 01]), Rosenthal and Seligman used scalability to characterize how nonsecurity-related processes for administering a database could become unwieldy as the number of databases and administrators grows. They proposed a refinement to make administrative processes and related software support more efficient. These authors, too, argued that reducing the administrative workload for certain actions would make the system more scalable.

Another example of the first definition of scalability can be found in "A Metric to Predict Software Scalability" [Weyuker 02]. In that paper, Weyuker and Avritzer described a process

for predicting when (i.e., at what workload level) a system's performance would become unsatisfactory. Given such a prediction, system developers could monitor the actual workload growth and increase the system's capacity before that level was reached. Weyuker and Avritzer gave the following definition:

> [Scalability is the ability] to continue to function with acceptable performance when the workload has been significantly increased [Weyuker 02].

The authors characterized an interval of demand in which the system would perform acceptably. Their implication was that if the interval were sufficiently large (i.e., covered a significantly wide range of workloads), the system would be scalable. They did not address how capacity should be increased or whether, say, doubling the number of processors would allow the system to handle double the workload.

Informally, people say things such as, "This system is scalable. It could handle double the workload (or a very large workload) with no problem." The term scalable is used in the sense of Definition 1 when it means the system can handle an extremely heavy workload with its current configuration. Because this usage is common, we have captured it in Definition 1.

1.1.2 A More Complex Concept: Scalability by Extension

Our second definition addresses more issues:

> Scalability is the ability to handle increased workloads by repeatedly applying a cost-effective strategy for extending a system's capacity.

This definition focuses on the strategy to be used in adding capacity. We're not interested in a one-time increase in capacity; rather, we're interested in the strategy for adding capacity and the number of times that strategy can be applied. We want to explore the way to address questions such as "If more processors are added, what is the method for coordinating work among the added processors?" If the coordination method used takes too many processing cycles, the full benefit of the added processing capability won't be obtained. Eventually, adding processors will not be cost-effective for increasing system capacity.

By far, our second definition is more commonly found on Web sites and in papers that discuss scalability issues. (For details, see Appendix B on pages 32–50.) However, in many papers, close reading is required to understand that the topic is a strategy for modifying a system to accommodate an increased workload. It is also important to keep in mind that the system being discussed sometimes includes human administrators as well as computer hardware and software.

The notion that it is possible to apply the scaling strategy repeatedly is essential to our definition and to an understanding of what causes scalability failures. For example, replacing an $O(n^2)$ sort with an $O(n \log n)$ sort increases the efficiency of a computation in the sense that a larger workload can be processed in the same amount of time. Thus, the algorithmic replace-

ment makes the system more scalable in the sense of Definition 1. But once the most efficient algorithm is in place, we are at the end of the algorithmic replacement strategy. Algorithmic replacement—or more broadly, performance improvement—is a valid strategy for increasing a system's capacity, but it is not a strategy that can be applied repeatedly.

1.2 Convention Used to Delineate Scalability Definitions

Because our definitions of scalability are significantly different and common usage does not clearly distinguish which meaning is intended, we will use subscripts in our text at times to call attention to which definition is being used.

- We use scalability_1, scales_1, scalable_1, or scaled_1 to refer to Definition 1 (the ability of a given system to handle increased workload).
- We use scalable_2 or scalability_2 to refer to Definition 2 (in which the strategy for expanding system capacity is the key).
- We use scale_{12}, scalable_{12}, or scalability_{12} to draw attention to both meanings.
- We leave the terms un-subscripted when we don't care which definition is meant.

Also, we will refer to our definitions as we have in this section: Definition 1 and Definition 2.

1.3 Document Overview

In Section 2, we'll take a closer look at scalability. In Section 3, we'll formalize some of the concepts. In Section 4, we'll introduce the concept of a scalability audit: those factors to be considered when assessing whether a system is likely to be scalable. Finally, two appendices include information from our interviews and literature survey.

2 Good Systems Gone Bad

In the introduction, we said that scalability is the ability to handle increased workload (1) without adding resources to a system or (2) by repeatedly applying a cost-effective strategy for extending a system's capability. Many people speak of a system being scalable (or not), but that's an oversimplification. Most systems scale in some sense (usually in the sense of our Definition 1), and no system is infinitely scalable.

Consider Figures 1 and 2. Figure 1 shows illustrative response versus demand curves for *System A* and *System B*. Clearly for any given demand level, the response metric is larger (worse) for B than for A. If there is a maximum tolerable value for the response metric, System B will reach it before System A. Using our first definition of scalability, it's clear that System A $scales_1$ better than System B.[1] If both curves remain monotonically increasing, it is clear that without some intervention, and given ever-increasing demand, both will eventually reach a point where the demand for resources exceeds their availability and the response metric becomes unsatisfactory. At that point (different for each system), the system will have reached its limit of scalability.

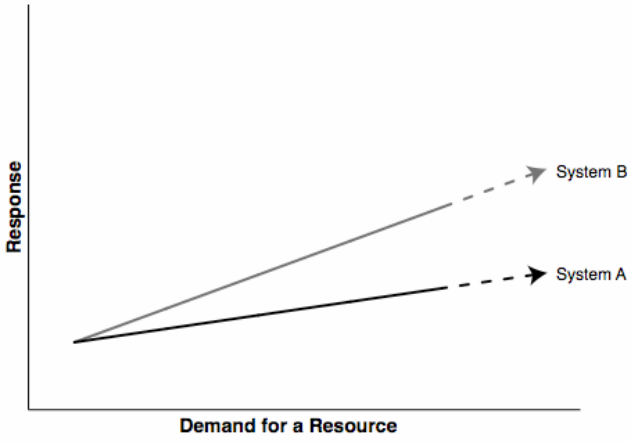

Figure 1: Response versus Demand

Figure 2 shows an imaginary system that tolerates increased demand until it comes up against a significant design limitation. Note that this curve (dashed portion) may well be the tail end of the curves shown in Figure 1. The further to the right the knee of this curve occurs, the more the system can be $scaled_1$. A system operating close to the knee of the curve is no longer $scalable_1$ because virtually any increase in demand for the resource will result in an excessive response metric (i.e., the system no longer $scales_1$).

[1] Reminder: We are using subscripts at times to call attention to which of our two definitions of scalability is being used. See Section 1.2.

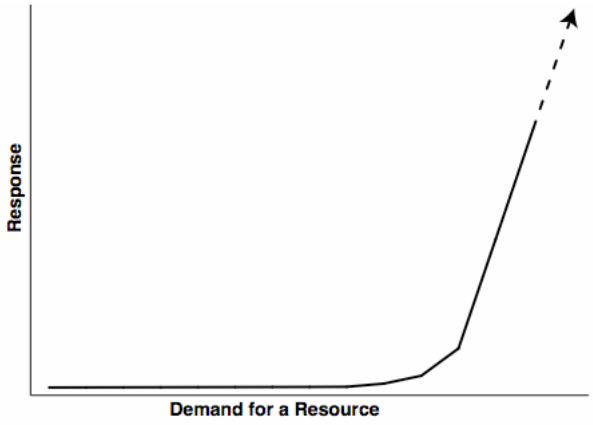

Figure 2: A Scalability Failure

Of course, the examples in Figure 1 and Figure 2 are idealized and are related to our Definition 1. More interesting real systems exhibit a hybrid behavior for which Definition 2 applies. Figure 3 illustrates this circumstance. When the system is running in the bottom zone, it is responding well. As it begins to run in the middle zone, the response starts to become unacceptable. Then, when it crosses into the top zone, the response is unacceptable (labeled "Unusable" in Figure 3). In the example we see that the administrators of the system were able to take quick action twice to keep the response acceptable, while once they required more time (and perhaps more effort). Eventually, the system reached a point where no reasonable action by the administrators kept the response metric from taking on an unacceptable value.

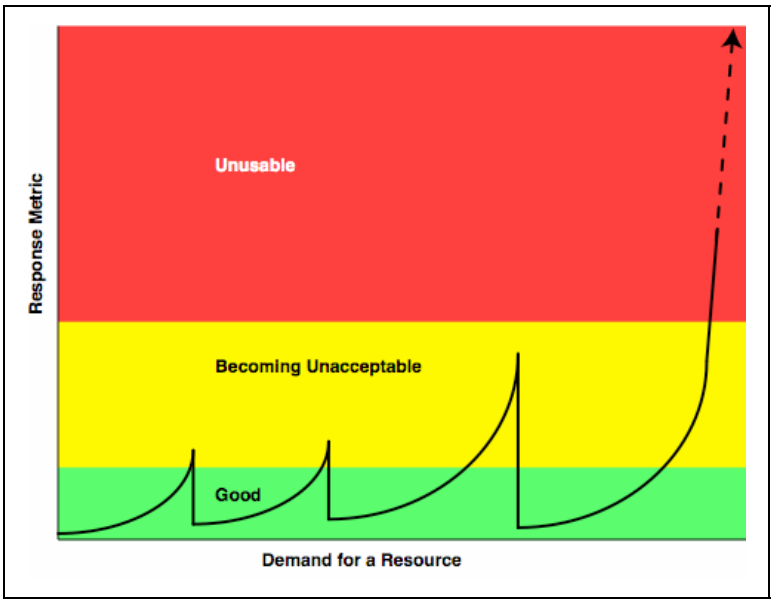

Figure 3: Response Metric vs. Demand for a Resource

To make this example less abstract, imagine that the administrators needed only to add a memory chip to bring response back into the bottom zone on the first two occurrences. The

next time response slowed, administrators had to replace the motherboard to accommodate more memory. The last time, however, when response passed into the middle and ultimately into the top zone, there was no cost-effective solution to the problem.

So, is this system scalable? The answer is "It depends." If the owner of the system can afford to continue to meet the higher levels of demand, then the system is scalable$_2$ up to the point where no cost-effective action (or possibly no action at all) can alleviate the response metric problems. Conversely, if the owner of the system spends all of its money on the initial system build and lacks the funds to purchase the first additional resource (and did not expect to have to), the system is *not* scalable$_2$.

System designers make informed guesses based, presumably, on user requirements and their own experience about the needed resources and the demands on those resources. They also try to anticipate future resource demand and usage patterns. Good system design accounts for anticipated changes—perhaps by assuring that all the resources needed are available from the beginning or by supporting the dynamic addition of resources when they are needed. A superior design will include some margin to allow for even more significant changes than those foreseen. By contrast, a poor design won't even take current resources and usage profiles into account. The lack of an accounting for current resources and usage profiles in system design happens more than it should and was one of the initial motivations for this report.

2.1 Causes of Scalability Failure

Scalability$_1$ failures occur when increased demand causes some resource to become overloaded or exhausted. This result can be seen in examples where

- Available address space is exceeded.
- Memory is overloaded.
- Available network bandwidth is exceeded.
- An internal table is filled.

Scalability$_2$ failures occur when some resource is overloaded or exhausted and adding capacity to the resource does not result in a commensurate ability to handle significant additional demand. For example, adding a processor may not allow a system to meet the additional demand if adding the processor also increases overhead significantly.

Scalability$_{12}$ problems surface as an apparent need for more resources than are available. Sometimes these resource limits are obvious (e.g., CPU, memory, disk, or network bandwidth). Other times, they are not (e.g., how much work an operator can accomplish or the amount of screen real estate). A system designed to be operated by humans with no alternative means of operation such as autonomic computing[2] will have a serious scalability$_2$ problem. In general, if a system is running short on some resource (virtually *any* resource), it has

[2] *Autonomic computing* deals with the design of computing systems that don't require much human intervention because they manage themselves [Kephart 2003].

a scalability$_1$ problem. If the shortage cannot easily be alleviated, the system has a scalability$_2$ problem.

Here's a real-world example of a scalability$_1$ failure. Comair is the commuter affiliate of Delta Airlines. During the Christmas travel season in 2004, Comair stranded thousands of airline passengers because its 10-year-old crew scheduling system crashed.[3] The reason for the crash was that the system was only capable of accommodating 32,000 (probably 32,767) crew schedule changes in a month. This limit was by design and was presumably acceptable—until it wasn't. Comair was able to cobble together a fix involving separate databases for pilots and flight attendants reasonably quickly but not before spoiling the holiday plans of a significant number of travelers. It took several months before a new system was ready to go online. Notice that this is different from the increasing demand/response trend shown in Figure 1. In the Comair failure instance, there likely was no indication that a problem was imminent—things didn't run slower and slower before they just stopped. One minute, there was no problem; the next, there was. This example is very similar to what might have happened if the Y2K problem had not been anticipated.

Resource exhaustion, however, is not always an indication of a scalability problem. If a system regularly runs out of memory, is that because the system requires a larger memory footprint or because it is leaking memory? Leaking memory is a bug—one that adding memory won't really solve. If the need for a bigger memory footprint can be easily and relatively cheaply met with another bank of memory (and if that action solves the problem), then the system does not have a scalability$_2$ problem. However, if it is really difficult or impossible to provide more memory or to make good use of the added memory, then the system likely does have a scalability$_2$ problem.

Notice that it is exceptionally difficult to anticipate all of the resources that will be affected by an unexpected increase in demand. An extreme example, anecdotally attributed to IBM founder T. J. Watson, is this: "I think there is a world market for maybe five computers." That statement is equivalent to a system designer predicting "There will never be more than 5,000 simultaneous users of the system"—and then discovering that the entire city of Chicago wants to order Bruce Springsteen tickets at the same instant.

2.2 The Interdependency of Resources and Scalability

When thinking about scalability, it is easy to fall into the trap of looking only at a single resource. In fact, for many complex systems, there are a number of different resources, and the exhaustion of any one of them can cause the system to fail to scale$_{12}$. When considering whether a system will scale over time, it is important to identify resources that may not appear to be relevant to scaling initially but will become more significant as particular kinds of demand grow.

[3] Read more at http://www.usatoday.com/money/biztravel/2004-12-28-comair-usat_x.htm?csp=34.

The previous section of this report discussed the idea that all systems are scalable to a point (and that no system is infinitely scalable.) That discussion focused on the behavior of some (unspecified) response metric as demand for some (unspecified) resource increased. In the real world, a complex system can fail to scale on multiple response metrics as the result of the increase in multiple kinds of demand. A Web site may become unusable (too slow) as the number of simultaneous users increases. It might also become unusable (again, too slow) as the number of entries in the underlying database increases. If both demand levels (number of users or entries) are near their respective knees on their scalability curves, taking action to allow a higher number of users to be served by the Web site may have any of the following effects on the system.

- In the best case, this change will alleviate the problem and the system will be back to normal.
- The increased number of users may lead to additional database entries causing the system to remain too slow—but for a different reason.
- The action taken to increase the capacity of the Web site (e.g., an increase in memory allocated to the Web server) may decrease the capacity of the database (e.g., reduce the memory allocated to holding frequently accessed tables), again causing the system to remain too slow for a different reason.

The preceding example illustrates that it is not reasonable to say that a system is scalable without specifying one or more demand measures with an allowable range of values and one or more response measures with an allowable range of values. Demand and response measures expressed in those terms might be stated "The Web site shall be able to handle at least 10,000 simultaneous users with response delays never exceeding 1 second" or "The database shall be capable of handling at least 1 million entries without causing response delays in excess of 1 second." The example also suggests that there are complex couplings between individual demand measures: will a fix to one problem exacerbate another?

2.3 Tradeoffs: "To achieve scalability I have to give up *what*?"

Building a system to be scalable (in either sense) almost always requires making tradeoffs with other attributes of the system. In order to achieve ever-higher levels of operation, it may be necessary to give up degrees of performance, usability, or another important attribute or to pay a big monetary price. In this section, we'll look at some of the issues involved.

Performance vs. Scalability

This tradeoff always comes to mind when thinking about scalability. In a non-scalable system, performance will often degrade when the demand (e.g., the number of users) or system complexity (e.g., the number of nodes) increases beyond a threshold. It may be possible to achieve higher levels of usage without a noticeable loss of performance simply by giving up some performance at the lower levels of usage. That is, instead of designing the system for the best-case performance given the currently expected levels of usage or system complexity,

the system designer will sacrifice some performance now for the ability to support higher levels of usage or complexity later. This powerful tradeoff can be used to support either definition of scalability, although the designer needs to be careful regarding how much performance to give up.

Cost vs. Scalability

Designing a system to be scalable$_{12}$ may entail upfront costs that are higher than those of a less scalable version of the same system. The adage "Pay me now, or pay me later" applies here. If the less expensive, less scalable design is chosen and the system reaches the point where it needs to grow but can't without heroic effort, the cost of modifying it will be extreme—perhaps even impractical. On the other hand, if demand never reaches a critical level, the incremental cost for the scalable design will have been wasted. With contracts going to the lowest bidder, program managers must understand that it may be hard to justify additional up-front cost unless there are

- scalability requirements[4] in the RFP
- a way of evaluating the life-cycle cost for different scenarios of demand growth

Operability vs. Scalability

As systems get larger and larger (especially distributed systems or integrated systems of systems), it becomes difficult for humans to operate them. One result of this phenomenon is the growing interest in autonomic computing. The tradeoff between operability and scalability may involve a loss of fine-grain human control of the system to achieve levels of scalability$_1$ that would otherwise be unmanageable (i.e., beyond human capabilities).

Functionality (Usability) vs. Scalability

The tradeoff between functionality and scalability recognizes that it may be possible to handle additional users or demands on the system if the system provides less service overall. Designing a system to provide differing levels of functionality may be one way of achieving a scalable$_1$ system.

One study analyzed real-world data from a top e-commerce site. The researchers spent time understanding the user community and the loads placed on the system by different kinds of visitors. To achieve maximal scalability$_1$, they suggested caching large quantities of static data for quicker delivery as well as shedding "personalization"[5] data as the load on the system increased [Arliltt 2001].

[4] A scalability requirement should specify not only possible growth in system load but also some restriction on the incremental cost to handle increased load.

[5] Commercial Web sites such as http://www.amazon.com often provide user-specific information to their customers. This is called personalization.

Replica Consistency vs. Scalability

One of the common goals of a distributed system is to achieve replica consistency in which the data on any one processor is identical to the data on any other processor at all times (other than possibly during an atomic transaction). Yu and Vahdat suggest that higher levels of scalability$_1$ are obtainable at a cost of having more network partitions and, therefore, a reduced guarantee of replica consistency [Yu 00].

2.4 Prediction of Scalability

A significant roadblock to predicting the scalability of a system is that scalability is a moving target—the pace of technological change can easily outstrip any particular scalability prediction. Consider, for example, how the growth in the number of households having broadband access to the Internet has fundamentally changed the kinds of data streams that are being served up from Web sites. Before widespread broadband access, the Web was predominately text-based with static images. Now, it is rare to come upon a Web site that doesn't have flash movies and other "gimcracks" that use up lots of bandwidth. A server scaled$_1$ in the mid-1990s could not keep up with the demands of today.

Another, albeit trivial, example is the way conferences used to give attendees access to the Internet. Not too long ago, conference sponsors would set up a room with a small number personal computers and dialup lines for use by the attendees. Over time, they began to provide a few workstations with high-speed connections. Although attendees had to queue up to use either means provided, the waiting period wasn't usually too long. In recent years, the trend has been to provide WiFi access throughout the conference venue. Suddenly the bandwidth demand has skyrocketed. Yet some venues still have only a single broadband connection and cannot deal with this massive wireless demand.

Finally, Satyanarayanan writes about his experience with the Coda file system project from the late 1980's to the 1990's. In the late 1980's, the ratio of disk size to main memory in a computer system was a significantly lower than it is today. Designs that optimized for the situation in the 1980's (e.g., Network File System [NFS]) would not scale as well in 2006 as those optimized for the ratio in 2000. Yet, it would be extremely unfair to fault the designer of the 1990 system [Satya 02].

There has been some progress towards predicting scalability. Weyuker and Avritzer developed the Performance Non-Scalability Likelihood (PNL) metric. The goal of the metric is to determine whether a system is likely to be scalable$_1$—which the authors define as being able to continue to function with acceptable performance when the workload has been significantly increased. The PNL value is then used to help plan for the eventual need to upgrade server capacity in a manner that is transparent to the users [Weyuker 02].

In order to compute the PNL value, it is first necessary to collect significant amounts of field data to derive the operational distribution, including an operational profile of the system un-

der study. The researchers claim that the cost of gathering and analyzing this data can be modest and is offset by the benefit derived from being able to predict when performance problems might become problematic. Because it requires the collection of field data, the PNL process is not a suitable prediction measure for a new system.

Liu et al. developed a performance prediction framework for component-based systems. The framework begins with a model that identifies the main components of the system and notes where queuing delays may occur. The model is calibrated by mapping an architectural choice to a set of infrastructure components and their communication pattern. The components are aggregated into computing modules. From this activity, a mathematical model is derived with parameters characterizing those modules.[6] The result of this modeling is an equation that predicts the performance of the specific system. Some of the parameters of this equation are tunable; others reflect internal details of black box components. To test the equation, an application with minimal business logic and a simple architecture is implemented and its performance is measured. Solving the model corresponding to this simple application allows the determination of the required parameter values, called the performance profile of the platform. These parameters can be substituted into the performance model of the desired application, giving the required prediction [Liu 04b].

[6] Characterization of the modules involves determining how often each component is executed.

3 A More Formal Look at Scalability

In this section we will be looking more formally at scalability.

Before proceeding, we need to define some terms and usage. First, we'll assume that we are dealing with a software-based system for which there is some set of response metrics defined. Let *M* (possibly an *n*-tuple) be the value for the metric that must be met by the system. For example, if the metric of interest was simply response time, *M* might be 0.1 second. Any response time under 0.1 second is tolerable; anything over is not. We'll initially assume *M* to be constant.

Second, the system is subject to a demand. The instantaneous demand on the system will be represented by *d*. The maximum demand that the system can meet while still satisfying the metric, *M,* is represented by *D*, also known as the capacity of the system. In Figure 3 on page 6, *D* is the point on the X-axis where the curve crosses into the top portion.

Suppose there is a system running on a uniprocessor at demand level $d \leq D$, a level that was expected and specified in the requirements. Since *d* does not exceed *D*, the system is, by definition, satisfying *M*. At this point, we have neither enough information about this system to say whether it can handle demand level *D* nor any information regarding its scalability.

3.1 How Capacity Can Be Increased

If we ask whether a system is scalable, we could be asking whether it can handle a "large" workload (Definition 1) or whether the system can be (repeatedly) augmented to handle increasing workloads (Definition 2). So if we ask how to *increase* the scalability of a system, the answer depends on whether we are considering scalability_1 or scalability_2. Increasing scalability_1 means increasing a system's capacity to handle a larger workload without adding resources to the system. Typically this is done by tuning the system in some way (e.g., by improving an algorithm, modifying a data structure, or reducing the amount of work the system needs to do by decreasing the services it provides when the system is heavily loaded).

Increasing scalability in the sense of Definition 2 means improving the *strategy* for repeatedly adding capacity to a system, where such a strategy is more or less feasible depending on its cost implications. For any system to be scalable_2, the capacity addition must be cost-effective. Cost-effectiveness has to be determined situationally. A large company may not have a problem going to a 32-bit event-ID because the server the system is running on has plenty of disk space. A small nonprofit with limited funds may be unable to purchase a larger capacity disk.

3.2 A Model of Scalability$_2$

In this section, we'll look at adding hardware resources incrementally to increase capacity.[7] We'll think in terms of processor units, P, and assume that the system can consist of one or more processors added incrementally. The maximum capacity of a single processor is D_1 demand units. That is, a single processor running alone will satisfy the metric M as long as $d \leq D_1$. Theoretically doubling the number of processors would result in increasing capacity to $2D_1$, but this will seldom be the case due to increased overhead.

This relationship is illustrated in Figure 4. The lighter, dashed line in the figure represents the maximum tolerable value of M, or the threshold. Notice that there are two potential overhead issues. First, doubling or tripling the number of processors does not double or triple system capacity (e.g., the 2-unit system is shown as having about 1.9 times the capacity of the single unit system, not twice the capacity). Second, adding a processor may decrease performance under low demand (the 2-unit and 3-unit curves start higher up the Y-axis than the 1-unit curve). Notice, too, that this penalty is paid only at very low loading levels.

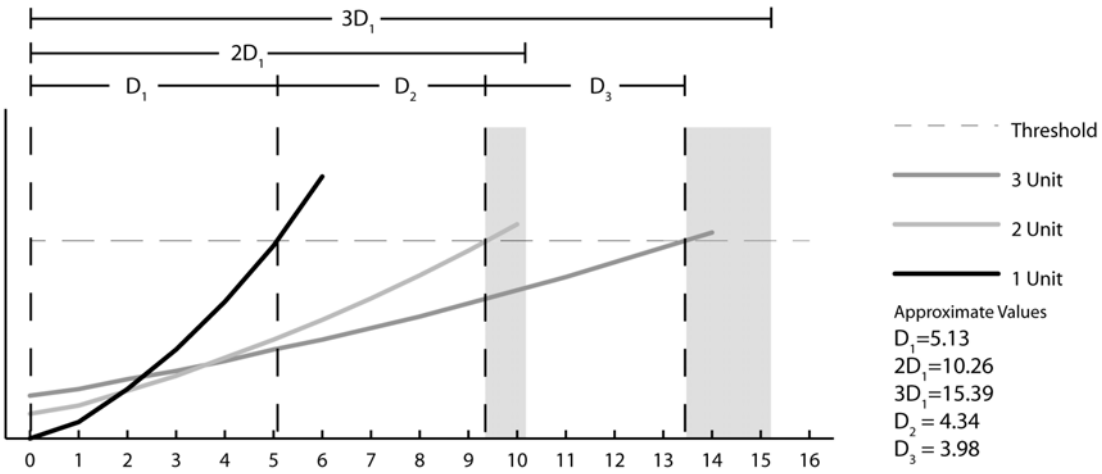

Figure 4: Capacity of Multiple Units

Let D_2 be the capacity added by the second processor. Then the total capacity of the system is $D_1 + D_2$. The efficiency, E_2, of the two processor system is $E_2 = (D_1 + D_2)/2D_1$ (i.e., the actual capacity available in the two-processor system when overhead is taken into account, divided by the capacity that would be available if overhead were ignored). In general:

$$E_n = \frac{\sum_{i=1}^{n} D_i}{nD_1}$$

We can use efficiency to measure the relative scalability$_2$ of a system. A system architecture in which efficiency falls off slowly as more capacity is added is more likely to continue to be

[7] Our explanation of a scalability model borrows from *Analytical Modeling of Parallel Systems* [Grama 03].

scalable (i.e., modifiable at reasonable cost) than one in which efficiency falls quickly. The efficiency curve for the system shown in Figure 4 is shown in Figure 5.

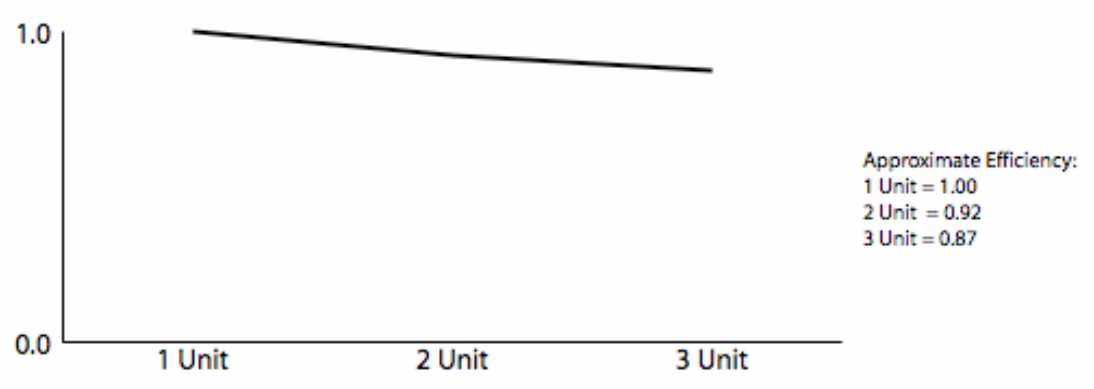

Figure 5: Efficiency Curve for Multiunit System

Because of the increasing overhead visible in Figure 4, this efficiency curve slopes downward. If this curve marks the beginning of a trend, then at some point in the future adding a unit will reduce efficiency to an unacceptable level (i.e., little additional usable capacity will be available).

Recall that our second definition of scalability is this:

> Scalability is the ability to handle increased workloads by repeatedly applying a cost-effective strategy for extending a system's capacity.

The discussion of Figures 4 and 5 ignores cost—but in reality cost can't be ignored. However, formalizing the concept of cost-effectiveness is tricky because it is a subjective concept. Having said that, here's an attempt to make some sense of the topic of cost-effectiveness.

Let C_i be the cost of extending a system's capability by adding the i^{th} unit. C_i is actually made up of two pieces: (1) a fixed one-time cost (e.g., the cost of a new processor) and (2) a variable continuing cost (e.g., the incremental costs of operating the additional processors, such as the extra electricity consumed).

Now, we have a cost value; but what about an effectiveness value? A surrogate for effectiveness is the efficiency value, E_i, as defined earlier in this section. At identical cost, a solution with higher efficiency is more cost effective than one with lower efficiency. At identical efficiency, a solution with lower cost is more cost effective than one with higher cost.

Given that calculation, we can define a cost-effectiveness metric K_i as some function—$f(C_i,E_i)$. When two strategies have different values of K, it is always more effective to pick the one with the higher value when adding the i^{th} unit. However, notice that this is a local optimization. It may be that choosing a less cost-effective strategy for adding the i^{th} unit would lead to a much more cost-effective strategy for adding the j^{th} unit.

Evaluating the cost-effectiveness of scalability$_2$ is a complex topic. Much more work is needed on this subject.

3.3 The Scalability Sweet Spot

An idealized scalable system will be able to start with a small footprint and grow into the indefinite future, remaining cost effective all the way. Figure 6 illustrates such a system.

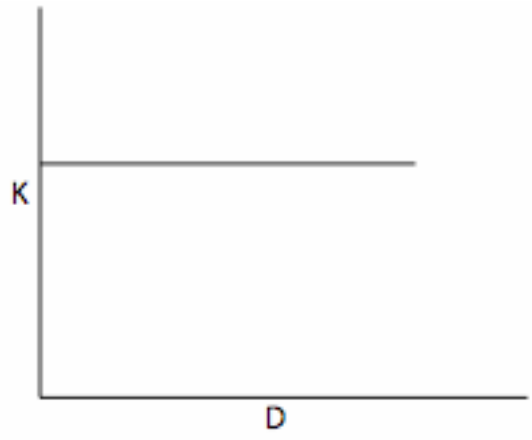

Figure 6: Idealized Scalability

This idealized system is unachievable—at least to the extreme. The system will always start to run into some scarce resource that cannot be increased in a cost-effective manner. When that point is reached, the curve will start to rise. For such a system, we would say that the region of scalability$_2$ ends where the curve begins to rise.

But a region of scalability$_2$ does not have to start when the system is deployed. Consider Figure 7, which shows a system whose cost-effectiveness does not start out very well and gradually improves as demand grows. As with all systems, it will eventually run into a point where it is no longer possible to add resources cost effectively to solve some limitation, but it does not reach that point for a while. We call the region where the system is highly scalable$_2$ its "sweet spot."

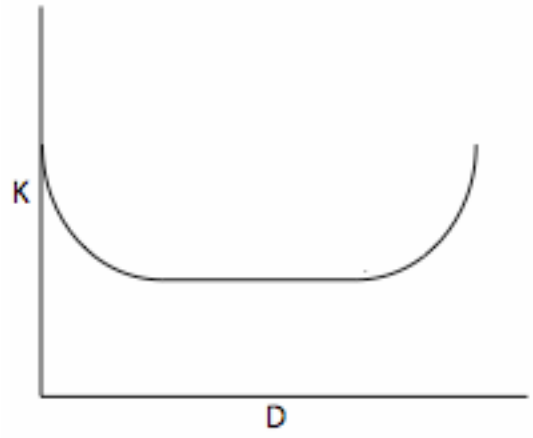

Figure 7: A System with a Sweet Spot

How might a curve such as that in Figure 7 occur in real life? All systems have fixed and variable costs. The system has to pay the fixed costs no matter how much demand there is. It only has to pay variable costs that are proportional to the demand. The curve above indicates a system with a lot of fixed cost that needs to be spread across an initially small demand. As capacity grows, the variable costs start to outweigh the fixed costs. At some point, variable costs dominate so much that the system ends up being very scalable from that point forward—until it isn't. This situation is not uncommon. Often systems are designed for demand levels that will not be achieved during early deployment. Alternatively, the system may be designed for and achieve a level of demand in the sweet spot from the beginning. Either way, a curve similar to that in Figure 7 will occur.

4 A Scalability Audit

In this section, we will discuss factors to be considered when assessing whether a system is likely to be scalable. If scalability$_2$ (i.e., the ability to increase a system's capacity cost-effectively) is a concern in a system's design, then the designers should examine any proposed strategy for increasing system capacity to see if all significant factors have been addressed correctly. We call this examination a *scalability audit*.

As a result of our interviews and literature survey, we see several recurring factors that must always be considered when developing a scalability strategy. This section suggests questions for evaluating such a strategy. We have not yet used these questions and concepts to conduct a review. Instead, the questions are a way of making operational the various insights we have formed in the course of the study.

A scalability audit can be applied to evaluate whether the architectural design of a system (1) can be easily augmented to handle a significantly higher workload over its anticipated lifetime or (2) is robust enough to accommodate an unexpected, significant increase in workload where failure to handle the increased workload would be disastrous or very severe for the system.

In other words, we see two situations in which it would be useful to conduct a scalability audit.

1. The system design is intended to accommodate a significantly increasing load over its lifetime.
2. The system is critical in some sense and its failure would be significant. In this case, you need to evaluate how likely the system is to fail when presented with unexpected load conditions.

In either situation, we recommend that the scalability audit be performed before the system is implemented or whenever considering a significant redesign.

The audit questions are divided into these four, somewhat overlapping, subgroups that are explained in the following sections:

1. potential resource bottlenecks and the strategy for identifying and relieving them
2. incorrect assumptions about a scaling strategy and how they can be revealed
3. general scaling strategies and their strengths/weaknesses
4. scalability assurance methods for monitoring confidence in a scalability strategy

4.1 Resource Bottlenecks

In a sense, all scalability issues revolve around identifying and mitigating bottlenecks. Performance tuning is typically an after-the-fact way of improving the scalability of a system in the sense of Definition 1, since tuning (in its simplest form) is aimed at using existing resources more efficiently. Substituting an $O(n \log n)$ algorithm for an $O(n^2)$ algorithm will certainly improve the amount of load that a system can process acceptably (at least as long as the replaced algorithm has an important role in satisfying the load). A tuning approach is perhaps most productively used once an implementation exists because without an actual workload and actual measurements, it's hard to know where to focus the tuning efforts.

Nonetheless, after-the-fact tuning analysis is not our focus. If a scalability audit is performed prior to a system's implementation, these are the types of questions that should be explored to identify possible resource bottlenecks:

- administrative bottlenecks
  - As system workload increases, what effect does it have on administrative workload? Will administrative staff have to grow faster than the workload? Has any task analysis been done of the workload associated with administering the system?
- capacity limits
  - What "hard-coded" capacity limits are contemplated for the system? For example, consider the system that assumes there will be no more than 32,000 entries of some kind. What is the impact if this limit is exceeded by the workload and to what extent will the system have to be re-architected if the capacity limit is exceeded? The example related to this issue is the Comair crew scheduling system failure previously discussed in Section 2.1.
- user interface
  Consider the amount of information that must be presented to a user through some interface. There are several potential issues here:
  - Will the anticipated way of communicating and receiving information from a user be appropriate if the amount of information grows by an order of magnitude or more? What methods are being considered for helping the user deal with a significantly increased communication workload? What would the impact of significantly increased workload be on the system architecture?
  - As the information to be communicated grows, will there be a way for the user to terminate the communication early (i.e., before all information has been communicated)? In many cases, a user will want to reframe a request because too much information is being generated and the user doesn't want to wait before submitting a refined request. This may not be a problem initially when there is a relatively small amount of information to be presented (or the time to process a poorly phrased request is short), but as the amount of information being dealt with grows, this can become an important issue [Anderson 99].

- algorithmic performance
  - What is the inherent performance profile of key problem-centric algorithms (e.g., computing lines of sight can be $O(n^2)$ for certain applications)? As the problem space grows, computing capacity could become a problem. Are there different approaches that could be considered? What range of n is likely to be accommodated for the contemplated architecture, and what is the risk if requirements grow larger?
- centralized control
  - Centralized control leads to resource bottlenecks. Allowing for decentralization helps to eliminate or reduce resource bottlenecks. Do the processes and overall approach determine if the number of points of centralized control has been minimized?

4.2 Revealing Scaling Assumptions

Factors that are easy to overlook when a system is relatively small become significant as the system grows. We are not talking here about the impact of workload growth per se on the performance of a system; rather, these factors show how the *expansion* of a system can reveal new problems that need to be considered by a scaling strategy.

- As a system expands, information about the system's configuration (both hardware and software) is likely to contain errors. These errors will be increasingly difficult to uncover unless an appropriate strategy is in place. If the strategy for maintaining or updating a system contains (implicit) assumptions that the configuration information is correct or that configuration skew is within certain limits (e.g., that all patches will be only one version out of date), there may be a significant problem as system growth begins to encounter these incorrect assumptions.
  - What are the assumptions about the correctness of configuration and usage information? What would be the impact on operational procedures if this information turned out to be incorrect?
- As a system expands or as workload increases, "rare" failures become more frequent because the conditions under which they occur become more frequent. For example, a race condition may show up only once in a million transactions. As a system's workload increases to the level of a million transactions per day, this "rare" failure will occur daily, on average. In particular, hardware failures will become more frequent as the amount of hardware increases, just because the failure rate is applied to a larger population.
  - In light of system and workload growth, what reliability assumptions are being made to determine the actions or support needed as certain types of failures become more frequent?
  - Are there procedures in place to monitor failures and failure trends and provide an early warning of certain kinds of failures that will increase as the system grows? With early warning, it may be possible to put appropriate procedures in place before the number of failure occurrences becomes intolerable.

- As a system grows larger, it's important to design the system so that failure effects are localized. In a small system, certain types of failure may take down the entire system, but a failure should not be able to take down the entire system when it has grown. Alternatively, if a failure occurs, there should be methods in place to ensure that a user does not lose a significant amount of work that has to be redone when the system comes back up.
  - Has a Failure Mode, Effects, and Criticality Analysis (FMECA) been done to determine the impact of a localized failure on the overall system?[8]
  - Are there recovery strategies in place to ensure that work is not lost when failure occurs? (These strategies become more important as the number of users grows, since the impact of a failure may affect a larger number of people.)
- As a system expands, it can become complex and difficult to understand unless appropriate strategies are in place to reduce the number of interactions that contribute to system behavior.
  - In what ways will the scaled-up system be more difficult to understand? What are the potential sources of complexity (e.g., what aspects that are readily understood when a system is small could become very complicated to understand as the system grows and interactions become more complex)?

4.3 Scaling Strategies

In this section, we'll discuss how to evaluate whether a scaling strategy has shortcomings.

- When a system has a long life, initial assumptions about how the system will be used will have to be changed. The scaling strategy may assume that certain patterns of use will continue.
  - How sensitive are these assumptions to possible changes in usage patterns, and how likely are those changes?
  - Is there a process in place to track usage patterns so that reevaluation of the scaling strategy can be based on factual data?
  - How do changes in types of use affect the nature of the capacity that needs to be added?
  - What resources are stressed by changes in type and volume of use? Is the strategy robust against a variety of possible changes in usage patterns?
- The scaling strategy may depend on users not taking advantage of knowledge about how the system is implemented (e.g., rebalancing load is much easier if applications have made minimal assumptions about where particular code is executing).
  - To what extent is the scaling strategy sensitive to users being able to "look behind the curtain" to take advantage of supposedly hidden information?
- Over the life of the system, assumptions about aspects such as equipment capacity (size, speed) and support tools will change. Notice that changes in hardware capabilities can

[8] For more about the FMECA method, go to http://en.wikipedia.org/wiki/FMECA.

lead to changes in usage patterns. For example, changes in disk capacity can lead to changes in what the capacity is used for; in turn, alternate usage can put different computational demands on a system (e.g., increased disk size and speed make multimedia applications possible).

  - Has the scaling strategy been evaluated against assumptions about hardware trends?

- If an objective of the scaling strategy is to support a large user base, the system may need to be architected to fit in a variety of user environments. System portability may also be relevant if the systems' software environment (operating system, middleware, and database technology) is changed.
- Look for "infinite" assumptions in the system (i.e., assumptions that some bound on demand will never be exceeded). Every system has some upper limits on specifications such as file size or number of database fields. When these limits begin to be approached, the system may be stressed and begin to fail more often or totally. Some examples of such assumptions are as follows: a certain type of file can be completely processed in memory; there will never be more than *X* transactions processed in a certain time period; or no file will ever need to be more than *X* bytes. Various problems have been experienced by real systems when these (usually unexamined) limits were exceeded.
  - What fixed capacity limits exist? How are those limits related to usage and usage growth assumptions?
  - What are the decentralized claims on resources that are assumed to have very large capacities? If everyone assumes that the total capacity is available, the capacity limits can be reached very quickly.

4.4 Scalability Assurance Methods

You can't really test (in the traditional sense) to see if a system is scalable. It is usually impossible to generate a realistic load—much less to actually add computing capacity to see if the planned scaling strategy will work smoothly. Scalability assurance addresses the mechanisms put in place to give greater confidence that a planned scaling strategy will work or, if it won't work, to give an early warning that problems are on the horizon.

- Look at performance curves for various resources as demand varies. How can the curves be characterized—as $O(n)$, $O(n^2)$, or $O(n \log n)$ for example? How do these curves change as capacity is added under particular scaling strategies?
- Identify mechanisms in place to reveal bottlenecks or where scaling design assumptions are beginning to be violated. What methods exist for assessing trends relevant to design decisions?
- Perform a SWOT[9] analysis on the scalability strategy:
  - **S**trengths—the kinds of growth the strategy is designed to handle

[9] Strengths, weaknesses, opportunities, threats (SWOT) analysis is a strategic planning tool used in evaluating projects and business ventures. We've proposed it here as a means to analyze scalability. For more information on the SWOT technique, go to http://en.wikipedia.org/wiki/SWOT_Analysis.

- **W**eaknesses—growth that would not be handled well by the strategy
- **O**pportunities—possible changes in workload or technology that the strategy would be able to exploit really well
- **T**hreats—possible changes in the workload or technology that would put the strategy in doubt

5 Summary and Conclusions

The ability or inability of systems (or systems of systems) to deal with issues of scale is becoming increasingly important in DoD and other systems. In studying this problem, we discovered that there was no uniform definition of what constituted a scalable system—an omission that we've attempted to rectify in this report.

We've shown that there are actually two common uses of the word scalability:

1. Scalability is the ability to handle increased workload (without adding resources to a system).
2. Scalability is the ability to handle increased workload by repeatedly applying a cost-effective strategy for extending a system's capability.

We've also shown how those definitions relate to real-world systems, both informally and formally.

We were helped in this effort by the results of both a literature survey and interviews with three individuals who have first-hand knowledge of scalability issues of differing magnitude.

The synthesis of our investigation is the concept of the scalability audit presented in Section 4. A scalability audit can be applied to evaluate whether the architectural design of a system (1) can be easily augmented to handle a significantly higher workload over its anticipated lifetime or (2) is robust enough to accommodate an unexpected, significant increase in workload where failure to handle the increased workload would be disastrous or very severe for the system.

We have not yet used the scalability audit to conduct a review of a real system development effort, and that would be the next logical step for our study of scalability.

Appendix A Interviews

This appendix contains a summary of the interviews that we conducted with three individuals who have had extensive experience with real-world distributed or embedded systems scalability issues. Conducted during February and March of 2005, the interviews elicited the experiences that Bruce Maggs, Mahadev Satyanarayanan, and David Farber—all faculty members at Carnegie Mellon University—had in constructing what they considered to be scalable systems. Each of them highlighted different aspects of the scalability problem.

Please note: We've summarized their comments below. Although the summaries are written in the style of a transcript, the content is a reconstruction from our notes of what they said. Our comments on the interviews are given in boxes interspersed with the text.

Interview with Bruce Maggs

> ***Bruce Maggs*** *is a professor in the School of Computer Science and the Department of Electrical and Computer Engineering at Carnegie Mellon. He's also vice president of research at Akamai Technologies, an organization that provides distributed computing solutions and services. Akamai delivers static Web pages and images for its customers though an on-demand distributed computing platform with more than 14,000 servers in 1,100 networks in more than 65 countries. We talked in his Carnegie Mellon office on February 11, 2005.*

Maggs: You notice that you've failed to scale when the system grinds to a halt because you have overloaded the CPU, run out of memory or disk space, or filled a network pipe. These are symptoms of some underlying scalability problem. Perhaps the underlying problem is a poor choice of algorithm. For instance, the bubblesort sorting algorithm works wonderfully up to a point. If the system never reaches that point, then perhaps bubblesort is the right algorithm to be using. However, if it is likely to exceed that point someday, then perhaps quicksort or another alternative would be a better choice.

Discussion: This use of *scale* is consistent with scalability Definition 1, because "failure to scale" refers only to the exhaustion of a resource; it does not address the problems faced when resources are added.

Maggs: A system fails to scale when you need one more unit of something and you are unable to provide it without using up significantly more than one unit of some resource.

Discussion: This comment is related to scalability Definition 2, because it concerns the overhead involved as resources are added.

Maggs: Small systems can be managed and dealt with by small numbers of people. Large systems require many more people, and truly large systems may require more people than can be hired and trained fast enough. As labor costs go up, profits may also not scale. These kinds of problems are behind the development of so-called autonomic computing.[10]

Discussion: This comment refers to an aspect of scalability that is sometimes overlooked—the growth in human administrative costs as systems get larger.

Maggs: This may seem obvious, but you can't test for scalability.

Discussion: You can't test for scalability in the sense that it may be impossible even to simulate the effect of increased workload and increased capacity. Some other approach is needed to gain confidence in whether a scaling strategy will work.

Maggs: If you attempt to simulate heavy workload in order to assess how a system will perform, the accuracy of the simulation can be a problem. A small measurement error in a simulation can give misleading results (e.g., the simulation may say you've only loaded the CPU to 70%, but in fact you've loaded it to 101%).

At any one time, Akamai expects a significant number of components across their network to be in a failure state. Needing a robust method for mitigating this condition, Akamai has turned to scalable fault tolerant strategies.

Discussion: Scalable here refers to the use of methods for which the administrative and computational burden does not increase faster than the number of components. By pushing out recovery procedures to the servers and routers themselves, Akamai ensures that the router/load balancer nearest to the failed server is responsible for reallocating its traffic in a transparent manner, reducing network and administrative load. In terms of scalability Definition 2, this is a strategy for adding network components in such a way that the costs of coping with component failures do not increase faster than the number of components.

Maggs: Upgrading software poses a nonintuitive problem that becomes significant as a system grows, because it becomes impossible to guarantee accurate knowledge of what versions are running where. Akamai has servers all around the world and does not expect all of them to be running the same versions of their various systems. Therefore, the various systems must be backward compatible—at least to a point. Further, it must be possible to revert fairly easily to a prior edition if that is what it takes to keep the system running.

[10] *Autonomic computing* implies a reduction in human involvement at least. For more information, see "The Vision of Autonomic Computing" [Kephart 03].

Discussion: The problem posed by software upgrading is more significant in large systems than in small ones. In small systems, you have more certainty about the system configuration than you can have in very large systems. If your procedures do not take into account the increased likelihood of incorrect system information, you will have problems in updating your system. In addition, you will face increased costs due to coping with growing configuration variances and trying to minimize them. These administrative costs become more significant because of the size of a system.

Maggs: A common mistake that is exacerbated by large numbers of servers in a widely distributed system is the failure to keep accurate configuration records. The configuration data is as important and serious an issue as new software and should be treated as such. Servers with different configurations still need to be able to work together. Akamai did an audit on their servers and discovered that all of them had the same parameters—such as the same version of the same operating system or the same memory—true 99% of the time. As systems grow in unbounded directions, you can't let that 1% kill you.

Discussion: That discussion highlights scalability as a management problem. Again and again, we see that distributed systems can become hard to manage efficiently using traditional techniques as the systems grow beyond a certain size. Traditional techniques assume an accuracy of knowledge that cannot be attained in large systems. You need to assume you will act on inaccurate information, and you must have ways of recovering from such mistakes.

Maggs: Everything we've discussed has bitten Akamai at one time or another, even when they were prepared for a particular problem; even little problems can cascade out of control.[11] Consequently, they tend to favor low-technology, well-tested mechanisms wherever possible. For instance, Akamai uses e-mail to send billing logs. Akamai's needs include a conservative process for adding capacity, engineering to avoid the need for operator intervention, and techniques to deploy new code safely as needs change.

Discussion: For the Akamai example, at least, there are two classes of scalability issues. The first type revolves around the ability to add servers as computing resources get stressed. The second, and seemingly more vexing to them, is the management and administrative issues and costs that develop as the number of servers increases.

[11] Dave Farber (see below) noted that in large-scale systems defects that are rare can be manifested on a daily basis (i.e., what is a minor problem in a small-scale system can become a major problem at a large scale).

Interview with Mahadev Satyanarayanan

Mahadev Satyanarayanan is the Carnegie Group Professor of Computer Science in the School of Computer Science at Carnegie Mellon and the founding director of Intel Research Pittsburgh. He was a lead architect and implementer of the Andrew File System as well as the principal investigator of the research that led to the Coda file system [Satya 02]*. We talked in his Carnegie Mellon on February 21, 2005.*

Satyanarayanan: When a project or a system has a long life, world assumptions may change. For example, during the development of the Coda file system, the size of disk drives and the size of memory changed significantly in relation to each other. Disk sizes today are much larger (in comparison to memory sizes) than in the late 1980s. Consequently, initial architecture decisions that assumed a lower ratio were no longer optimal. The changed ratio also changed how users used the system. For instance, multimedia was not practical when disk drives were smaller and slower. Multimedia applications are commonplace now, and they put stresses on a system that it was not designed to handle.

Discussion: The implication here is that it's fine to design a system to accommodate growth in certain kinds of demand. But if the assumptions underlying the design change in unforeseen ways, the system may require an unexpected and costly redesign. Scalability, as used here, is the ability of the system to perform well as more storage capacity is added, the number of users increases, and the applications the system supports change how disk storage is being used (e.g., for multimedia applications).

Satyanarayanan: Scalability is important enough that a scalability audit should be a standard and required practice for any large system development. The audit should look at the architectural and implementation levels and should consider things such as the data structure layout and assumptions (implicit or explicit) about what capacity will need to be added and how it will be added. The audit will help to make assumptions explicit, and this may lead to better design decisions for accommodating growth. The audit should also identify mechanisms to be used to determine when key assumptions are becoming less tenable. Such mechanisms should identify trends to determine at what point certain assumptions may be violated.

Discussion: This comment was the source of our section on Scalability Audits (beginning on page 18). In addition, notice that the audit is meant to address how capacity will be added—that is, the concern is with the strategy for adding capacity. This concern is consistent with Definition 2.

Satyanarayanan: One reason that systems don't scale is that people start with small configurations and build up. Since they are not sure that the system will be successful, they do not invest the time, money, and energy in designing to make it easy to add capacity to service certain kinds of increased demand. Scalable designs inevitably increase the initial system's

complexity, development cost, use of computational resources, and operational costs. And there is no guarantee that paying the price now will lead to any benefit at all in the future. Yet, failure to design for growth in demand can be very unsatisfactory when the system actually needs to be scaled up.

Discussion: The discussion talks about the conflict between designing for today's demand and designing to meet much higher demand than is cost-effective to support at the start. Consider the added costs of designing, implementing, and operating a system that is initially just a handful of nodes in the same room versus the costs of designing and implementing a system to be efficiently operated when it consists of hundreds of thousands of nodes spread geographically throughout the world in some indefinite and possibly unlikely future. There is a counterpart to this problem in the world of building and construction: the desire to keep the initial cost of a building low may lead to a lifetime of higher maintenance costs.

Satyanarayanan: Network File System (NFS) is an example of a file system that did not scale well. As the number of users/files increased, a variety of band-aid solutions (e.g., automounter software) were required to make NFS manageable at large scale. In contrast, both the Andrew File System [Howard 88] and Coda [Satya 02] thought ahead with respect to scalability and avoided putting server names or explicit mount points in the file names. Server anonymity was a desired property. This made it easy to add new servers and to expand the user base.

Discussion: The use of the word *scale* here is somewhat different. The system designed with anonymous files made it relatively easy to add large numbers of servers and users. Files could be transparently moved from server to server to balance load. In contrast, load balancing on NFS was not transparent to users because server names were included in file references. In that it highlights a usability problem brought on by scaling up, this situation is related to but different from the administrative scalability issue discussed previously.

Satyanarayanan: Scalability is a matter of identifying bottlenecks, right? Well, not necessarily. "Infinite" assumptions can create resource usage problems that cause scalability issues. For instance, consider the tragedy of the commons: each user takes a little bit of a resource for individual use. Pretty soon you have a problem. For a real-time system, the commons is time and the locking strategy. To help identify these issues, look for decentralized claims on resources.

Discussion: Resource exhaustion reveals bottlenecks that are sources of performance problems. When you have a river that flows into a pipe and you fill the pipe, you've got a performance problem that can be solved by having a bigger pipe, slowing the arrival of water, or storing the backed up water until it can be handled by the pipe.

Satyanarayanan: Finally, scalability is not a scalar, it's a vector. We need to understand scalability dimensions to identify regions within which a system will operate satisfactorily.

Interview with Dave Farber

Dave Farber is the Distinguished Career Professor of Computer Science and Public Policy in the Institute for Software Research International at Carnegie Mellon. Additionally, Professor Farber was one of the pioneers behind the Arpanet, a precursor to the Internet. We talked with Professor Farber on March 3, 2005.

Farber: As application complexity increases, rare bugs are manifested more often, making a usable system all of a sudden unusable. As an example, in the early days of the Internet we had a protocol failure that showed up once in a while. It worked fine in the testbed and when the Internet consisted of just a few nodes, but as more and more nodes were added, things all of a sudden stopped working. The problem was a race condition that occurred once in a million times for four nodes. But with hundreds of nodes the problem manifested itself at least once a day and it became impossible to do large file transfers. This is indistinguishable from a scalability problem to the outside observer.

Discussion: In the preceding two interviews, we observed two types of scalability issues—one had to do with resource demands and the other with managerial issues (operating costs). Farber's example seems like a third type—a "law of large numbers" kind of scalability. As a system gets bigger and bigger, defects in the system manifest themselves more frequently. Rare defects become frequent because the unusual (rare) conditions under which they are manifested occur more often. The population of condition occurrences is so much larger per unit time.

For example, if a failure occurs once per million file transfers, the failure rate is low. But if the system load is such that there are a million transfers per day, there will be about one system failure per day due to file transfers. A rare event when measured against the number of events becomes frequent when we consider how many of these events occur in a given time interval. This becomes a scalability issue when growth in workload causes an increase in the number of events subject to a low failure rate.

Farber: As another example, consider a system the size and complexity of OS/360. In that system, bugs were fixed and the number of reported bugs decreased over time—until there came a day when bug fixes began introducing additional bugs. This is because the ramifications of a bug fix were not clear given the size and structure of the system. Again, though, is this a scalability problem or a complexity problem, and are they different? Similarly, requirements creep leads to scalability issues. Eventually the system gets too big and can't cope with the new requirements.

Discussion: The implication here might be that a system can become more difficult to understand as it grows in size. Consequently, changes to the system can have unexpected effects. From a scalability viewpoint, perhaps the issue is the effect of the scaling strategy on system understandability. If the addition of capacity makes the system harder to understand and fix, the ability to continue with the strategy will become less and less.

Another possible inference to draw from the OS/360 example is that reported bugs eventually become very subtle to fix because the repair involves understanding more and more of the system and how the system is used. The lack of understanding is only revealed when the bug fix is found to be incomplete or partially wrong. This implies that if growth in complexity (meaning diminished understanding of how a system works) is a consequence of increasing the capacity of a system, new methods of understanding system interactions will be needed in order to keep the system operational.

Farber: It's not just bugs and system complexity that cause a system to behave poorly as the system grows. There a have been CAD tools that fail when they could not fit their diagrams on one screen (at a usable size.)

Discussion: This form of growth in demand effectively combines resource issues and usability issues. The resource in question is screen space. The assumption that screen space is always big enough to display a diagram becomes an assumption that limits the size of problem the system can handle (i.e., the assumption becomes a limit on the system's scaling strategy).

Farber: We need to ask, "Which negligible problems today become big at a higher scale factor on any one of the scale dimensions?" The electric power system runs fine until there is an overload, at which point the whole distribution system can collapse. Are there lessons to be learned?

Discussion: One lesson to be learned is that in large systems, it's important to isolate failure effects. If part of a system fails, it shouldn't bring down the rest of the system.

Farber: I was involved with two research projects designed to be scalable. The designers spent a lot of time working on issues such as the number of processors and daemons. They had to worry about performance versus scale. A message-based interprocess communication (IPC) is an example. You pay a performance price in small applications to get much better scalability as you go to multiple processors or nodes. You have to sacrifice performance in the small application to get performance in the large one.

Discussion: When does it pay to invest extra effort in building a system for large-scale use, instead of starting out to accommodate small-scale use and re-architecting as necessary?

Appendix B Books and Papers

This appendix contains details of the literature survey conducted as a part of this study. We reviewed papers and books to see how the term scalability was used and what kinds of problems were considered scalability problems. The citations are grouped into categories:

- definitions found in an Internet search
- simulation scalability
- distributed system scalability
- prediction of scalability
- parallel processing scalability

Our review showed that it was not always easy to decide what was meant when the term *scalable* was used. We began to question every use of the word to see what might be intended. This usually led to greater insight into the issue being addressed, as noted below in our comments on the papers.

Please note: The paragraphs below each referenced item represent our summary of the content of the document. The comments in boxes represent our discussion of how the document uses the term *scalability*.

Definitions Found in an Internet Search

http://searchdatacenter.techtarget.com/sDefinition/0,290660,sid80_gci212940,00.html

Two usages are defined. In the first, scalability is defined to be

> the ability of a computer application or product (hardware or software) to continue to function well when it (or its context) is changed in size or volume in order to meet a user need. Typically, the rescaling is to a larger size or volume. The rescaling can be of the product itself (for example, a line of computer systems of different sizes in terms of storage, RAM, and so forth) or in the scalable object's movement to a new context (for example, a new operating system).

A second usage is defined as

> the ability not only to function well in the rescaled situation, but to actually take full advantage of it. For example, an application program would be scalable if it could be moved from a smaller to a larger operating system and take full advantage of the larger operating system in terms of performance (user response time and so forth) and the larger number of users that could be handled [TechTarget 05].

Discussion: The first definition captures the idea that scalability is defined with respect to different versions of a system, with one version being more capable than the other. Both definitions consider a performance metric that is satisfied ("continue[s] to function well") as load is increased. Neither definition considers the ease of scaling to a larger size.

http://www.webopedia.com/TERM/s/scalable.html

Scalability refers to

> how well a hardware or software system can adapt to increased demands. For example, a scalable network system would be one that can start with just a few nodes but can easily expand to thousands of nodes [Jupitermedia 06].

Other definitions from the same source include the following: "refers to anything whose size can be changed (e.g., a scalable font is one that can be represented in different sizes)" and "when used to describe a computer system, the ability to run more than one processor."

Discussion: The definition emphasizes ease of expansion—the amount of effort that would be expended to cope with increased demand is the key factor in deciding whether a system is scalable.

http://computing-dictionary.thefreedictionary.com/scalable

> A 'highly scalable' device or application implies that it can handle a large increase in users, workload, or transactions without undue strain. Scalable does not always mean that expansion is free. Extra-cost hardware or software may be required to achieve maximum scalability [Farlex 05].

Discussion: The first part of this definition focuses on the ability to handle increased demand, without necessarily implying that the system needs to be augmented. This part of the definition agrees with our Definition 1. The latter part of the definition mentions that the cost to increase a system's capability might be a factor in deciding whether it is scalable. Such considerations are part of our Definition 2.

http://www.computeruser.com/resources/dictionary/noframes/nf.definition.html?bG9va3VwPTQ5ODY=

A system is scalable if it is

> able to be changed in size or configuration to suit changing conditions. For example, a company that plans to set up a client/server network may want to have a system that not only works with the number of people who will immediately use the system, but the number who may be using it in one year, five years, or ten years [Key 06].

Discussion: This definition is consistent with our Definition 2 because it focuses on the design of a system and the ability to change the system to meet different conditions. The implication is that ease of change is part of what makes a system scalable.

http://en.wikipedia.org/wiki/Scalable

According to the Wikipedia

> [s]calability indicates the capability of a system to increase total throughput under an increased load when resources (typically hardware) are added.
>
> A routing protocol is considered scalable with respect to network size, if the size of the necessary routing table on each node grows as O(log *N*), where *N* is the number of nodes in the network.
>
> To scale vertically or scale up means to add resources to a single node in a system, such as adding memory or a faster hard drive to a computer. To scale horizontally or scale out means to add more nodes to a system, such as adding a new computer to a clustered software application.

In addition, scalability can be measured by these aspects:

1. load scalability
   A distributed system should make it easy for us to expand and contract its resource pool to accommodate heavier or lighter loads.
2. geographic scalability
   A geographically scalable system is one that maintains its usefulness and usability, regardless of how far apart its users or resources are.
3. administrative scalability
   No matter how many different organizations need to share a single distributed system, it should still be easy to use and manage [Wikimedia 05]

Discussion: The primary definition focuses on the effect of added resources. Cost-effectiveness is implicit, as is consideration of the strategy for adding resources. This is the only definition we encountered that discusses various dimensions of scalability.

Simulation Scalability

Law, D. R. "Scalable Means More than More: A Unifying Definition of Simulation Scalability," 781–788. Washington, DC, December 13–16, 1998. *Proceedings of the 1998 Winter Simulation Conference (WSC 1998)*. New York, NY: ACM, 1998.

The author notes that in the simulation community "the word 'scalable' is often used as shorthand for 'simulates more entities.'"

Discussion: This usage is consistent with our Definition 1.

This paper presents definitions of simulation scalability that take into account performance measures of interest (e.g., how fast a simulation runs), simulation capability (e.g., the number of entities in the simulation), and architectural capability (e.g., the number of processors). The author provides various ways of characterizing simulation capability (the size of the problem being simulated) and architectural capability (the amount of computing resources available to solve the problem). Law makes the key point that scalability is a relative measure—one must identify a baseline in terms of simulation problem size and architectural capability.

The weakest definition proposed by the author is that a system is scalable if it "exhibits improvements in simulation capability in direct proportion to improvements in system architectural capability" [Law 98]. A stronger definition says that the scalability of a system is the size of the interval in which the ratio between simulation capability and architectural capability remains at least as good as it was for the specified benchmark architecture.

Discussion: In the second definition, scalability is delineated with reference to a benchmark and some measure of increased system capability. This definition is more consistent with our Definition 2, because it considers the strategy for expanding the system.

The author notes that

> [i]n the domain of parallel computation, an algorithm is scalable if the level of parallelism increases at least linearly with the problem size. An architecture is scalable if it continues to yield the same performance per processor, albeit used on a larger problem size, as the number of processors increases.

Discussion: The definition, quoted from the parallel processing community, is consistent with our Definition 2 because it considers both expansion of processing capacity and system load. The quote is a bit subtle in mentioning "level of parallelism" because this refers to the ability of the algorithm to exploit the added processing resources (i.e., for larger problems, the amount of useful computation done by each processor increases proportionally).

The author also quotes a definition used by the Defense Modeling and Simulation Office (DMSO): "the ability of a distributed simulation to maintain time and spatial consistency as the number of entities and accompanying interactions increase."

Discussion: This definition is more clearly related to our Definition 1 because it refers to the performance of a system as the load increases and does not explicitly mention any strategy for adding computational capacity.

The author then goes on to say that, in general usage, "a system [is] scalable if the customer can solve larger problems by buying more systems or more powerful systems of the same type." He notes that there is often a focus on system scalability with respect to hardware, using a performance per dollar metric.

Discussion: This definition of scalability implicitly refers to the ease (cost) of increasing system capacity as a factor in determining the scalability of a system.

The author notes that measures of performance depend on underlying problem constraints:

> The complexity of the underlying problem dictates the best performance that we can hope to achieve with any algorithm. If it is mathematically impossible to evaluate lines of sight among N tanks in less than kN^2 time, it is useful to know that before we attempt to develop algorithms to solve the problem.

Discussion: This comment refers to the ability to handle a simulation load independent of the amount of computational capacity available. Hence, is a comment consistent with our Definition 1—performance under increased load without adding resources to the system.

In line with thinking that scalability is a matter of understanding resource bottlenecks, the author notes: "scalability is a function of the application, since each application may stress different system resources."

Discussion: The last comment refers to the concept of resource bottlenecks as barriers to allowing systems to do more work. As a result, it seems to be consistent with our Definition 1—the notion of scalability that focuses primarily on how well a system responds to increases in demand without augmenting the system. Alternatively, the comment could be taken as implicitly referring to strategies for increasing the capacity of different system resources, understanding that different strategies may be needed for different resources. Consequently, strategies for making systems scalable (in the sense of Definition 2) are application-dependent.

The paper also provide an alternative definition of scalability: "the size of the interval in which the ratio between simulation capability and architectural capability remains at least as good as it was for the specified benchmark architecture" (i.e., scalability is with reference to a benchmark and with reference to some measure of system capability).

Discussion: The ratio between simulation and architectural capabilities seems similar to the notion of *efficiency* in the parallel processing community. The idea is that a system is considered scalable to the extent it is able to efficiently use the added resources on larger problems (i.e., a scalable simulation is one that exhibits improvements in simulation capability in direct proportion to improvements in system architectural capability.) This definition is very similar to our Definition 2, since scalability here is about the ability to handle an increase in demand (i.e., the size of a simulation) with no more than a linear increase in architectural capability.

Boesel, J. R., *An Air Traffic Simulation Model that Predicts and Prevents Excess Demand*, The MITRE Corp., 2003. http://www.mitre.org/work/tech_papers/tech_papers_03/boesel_ats/boesel_ats.pdf.

This paper describes a simulation model for air traffic control using the SLX[12] simulation language. That model contained 500,000 objects with up to 250,000 objects active at a given time. The simulation ran in 9 seconds on a 2 GHz PC [Boesel 03].

Discussion: Scalability seems to be used in this paper to mean the ability to handle very large numbers of objects in a simulation while producing results quickly. This is a typical use of the term in simulation applications (see [Fujimoto 95], below). Aside from the statistics concerning the speed of the simulation, scalability is used only to indicate that the performance of the simulator on a large model was considered to be impressive. No data was given to indicate the behavior of the simulator under a range of simulation load, nor was there any discussion of how simulation capacity could be increased for significantly bigger simulations.

Fujimoto, R. "Parallel and Distributed Simulation," 118–125. *Proceedings of the 1995 Winter Simulation Conference (WSC 1995)*. Arlington, VA, December 3–6, 1995. New York, NY: ACM, 1995. http://ieeexplore.ieee.org/iel2/3475/10228/00478713.pdf?isnumber=10228&prod=CNF&arnumber=478713&arSt=118&ared=125&arAuthor=Fujimoto%2C+R.M.

In parallel and distributed simulation, *speedup*—performance improvement relative to sequential execution (i.e., execution on a single machine)—is the measure of scalability [Fujimoto 95].

Discussion: In the Fujimoto paper, a distributed simulation is said to be scalable if it can be expanded to include more simulated entities executing on proportionately larger hardware configurations, with the simulator still being able to meet its stated objectives (e.g., value as a training mechanism). In turn, this condition translates to real-time performance.

[12] For more on the SLX language, go to http://www.wolverinesoftware.com/SLXOverview.htm.

Distributed System Scalability

Tran, P. & Gorton, I. "Analyzing the Scalability of Transactional CORBA Applications," 102–110. ***Proceedings of Tools 38: Technology of Object-Oriented Languages and Systems: Components for Mobile Computing.*** **Zurich, Switzerland, March, 12–14, 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.**

This paper focuses on transactional Common Object Request Broker Architecture (CORBA) applications using the Visibroker Integrated Transaction Service v1.2. The authors measure how well an architecture "scales" by measuring throughput in terms of transactions per second. The experiment shows that the CORBA transaction model "scales well" in the limited situation tested (i.e., the ability to process an increased transactional load increases proportionally with an increase in processing resources) [Tran 01].

Discussion: This is a clear example of our Definition 2, since the authors discuss the ability to handle an increased transactional load by increasing the resources available. The increased processing ability appears to be somewhat linear from the results (e.g., one server can handle 160 transactions per second [TPS], and three servers can handle about 400 TPS). There was no discussion of any problems that had to be overcome to achieve this degree of scalability.

Jogalekar, P. & Woodside, M. "Evaluating the Scalability of Distributed Systems." ***IEEE Transactions on Parallel and Distributed Systems, 11,*** **6 (June 2000): 589–603.**

This paper presents a scalability metric based on cost effectiveness, where the effectiveness is a function of the system's throughput and its quality of service. Speedup S measures how the rate of doing work increases with the number of processors k, compared to one processor, and has an ideal (linear) speedup value of $S(k) = k$. Efficiency E measures the work rate per processor [that is $E(k) = S(k)/k$] and has the ideal value of unity [Jogalekar 00].

Discussion: This paper uses scalability in the sense of Definition 2, the ability to get more work done with more processing power. The definitions of speedup and efficiency used here are those typically used in the parallel processing community.

Anderson, K. M. "Supporting Industrial Hyperwebs: Lessons in Scalability," 573–582. ***Proceedings of the 1999 International Conference on Software Engineering (ICSE 99)*****. Los Angeles, CA, May 16–22, 1999. http://ieeexplore.ieee.org/iel5/6783/18169/00841047.pdf?tp=&arnumber=841047&isnumber=18169.**

This paper describes Chimera, an open hypermedia system.

> An open hypermedia system (OHS) is typically a middleware component in the computing environment. . . . Using the services of an OHS, existing applications in the computing environment can become 'hypermedia enabled', thus supporting linking to and from information managed by the application without altering the information itself. To become 'hypermedia enabled', applications must be extended to make the hypermedia functionality available in the user interface and must be able to communicate hypermedia requests to the OHS. The term open hypermedia environment is used to cover both the OHS and the set of hypermedia enabled applications[13] [Anderson 99].

Discussion: From this definition of an OHS, it can be seen that workload increases as the amount of information to be hyperlinked increases. Coping with the increased workload presents scaling issues.

The Anderson paper focuses on meeting an industrial user's needs for a "large-scale hypermedia" application (i.e., one in which the amount of information to be linked was increased by three orders of magnitude). Although the system was able to handle hundreds of "anchors and links" in a test application, the user encountered problems in an application having tens of thousands of anchors and links and needed to develop a demonstration having approximately half-a-million anchors and links. In the process of addressing this increase, the user encountered a number of different problems.

Discussion: The measure of scalability implied by this paper is consistent with our Definition 1—in this case, the number of "anchors and links" that the system can "handle." The definition of "handle" was, at first, simply the ability to process the number of anchors and links presented to the system.

This paper addresses issues that arose when adapting the system to handle a much larger workload. The paper was, thus, a good case study of problems that should be considered in a scalability audit. The first problem was that the system could not process tens of thousands of anchors and links because the method for parsing XML required that the entire parse tree be held in memory and the size of the XML parse tree exceeded the amount of memory. To parse bigger XML files, the developer changed to a method that did not need to read an entire file into memory to parse it. The need to replace the method of parsing XML illustrates a scaling_1 problem caused by the choice of algorithm.

[13] For the OHS working group, go to http://www.csdl.tamu.edu/ohs/intro/chair.html

The author notes that scalability depended on a number of different factors—having a technique for handling an increase on one scalability dimension (the ability to parse large XML files) only revealed limitations on another scalability dimension (the user interface). Moreover, the techniques needed to cope with the limitations associated with a given scalability dimension varied across different magnitudes of scale.

> After expending a certain amount of developmental effort on reaching one order of magnitude [increase on a scalability dimension], a comparable level of effort may be required to reach the next. . . . [Moreover,] as subsystems are scaled [i.e., allowed to process more data, and thereby perhaps producing more data to be processed by other components of a system], they impact the ability of other subsystems to function effectively.

Discussion: This is one of the few papers we encountered to discuss different dimensions of scalability, and it is the only paper mentioning the need for different techniques at different points on a given scalability dimension. Moreover, this paper is the only one to observe that different techniques (and significant amounts of effort) may be needed when moving to different points on a given scalability dimension. Finally, this is the only paper that mentions how solving a problem on one scalability dimension may just reveal a problem on another dimension. For the Chimera system, solving the problem of parsing large XML files only revealed a user interface problem.

Another important observation is that "as developers apply techniques to scale their system [i.e., to address limitations that are encountered as demand grows], they experience . . . unanticipated interactions of the techniques being employed."

Initially Chimera "always kept its data structures in memory and used simple file management routines to manage the persistence of this information. This approach scaled to [i.e., was sufficient for] hyperwebs containing thousands of anchors and links, but had problems scaling any further [i.e., had problems for larger hyperwebs]." The developers turned to the use of database technology.

They hadn't used database technology originally because they wanted their system (a form of middleware) to be usable in a wide variety of contexts. Linking it to a particular database technology would have created an entry barrier since users would have had to obtain and configure a database.

Discussion: This is an interesting example of how a technical decision made to increase the marketability of a system wound up conflicting with the ability of the system to deal with large loads. This might be considered a "range of applicability" dimension of scalability (i.e., this dimension is concerned with reducing the factors that limit the potential range of users of a system).

Changing their system to make use of a relational database cost a lot of effort but was essential to enable its use with large hyperwebs. The developers wanted to choose a free or low-

cost database, so they settled on an SQL product. But they discovered that each database vendor provided implementation-dependent SQL extensions that had to be used to achieve the needed performance. Portability among vendors was not possible (at least as they saw the issue at that time), but it was important to attain the "market scalability" objective (i.e., to allow Chimera users free choice of an underlying database product).

Discussion: The problem of being able to write portable code so that a product can interface with a variety of other products represents a scalability constraint on the size of the user base that could be developed.

In distributed computing applications, it is necessary to minimize the amount of network traffic to achieve acceptable performance at high loads. The Chimera developers noticed that a straightforward implementation of query operations would require one network call for each hyperweb link. A test showed that this approach was unacceptable for large hyperwebs; it would take more than 24 hours to process 33,000 links. As a result, the developers took an approach that required only a single network operation to gather all the needed information. On the same test problem, this approach took only 5 minutes.

For our purposes, the key point in the paper is that the developers didn't realize their algorithmic choice was wrong until they had access to large data sets. The author concludes: "[the developers'] experience points to a need for augmenting [the design-to-implementation] stage of software development with guidelines to help engineers prepare their implementations for large-scale contexts."

Discussion: An analysis of how an algorithm performs under increasing load is clearly one aspect that software engineers should consider. We call this the "algorithmic" dimension of scalability.

Anderson points out that developing a system that could handle much larger hyperwebs required the use of techniques from a variety of technical disciplines: "compiler construction, distributed computing, databases, software engineering, [information retrieval, and human-computer interaction]."

As they worked with large hyperwebs, the developers found that the default user interface behavior was not suitable—it took too much time to display large hyperwebs. They created an interface that allowed the user to see how long it would take to display all the requested information, so that the user could cancel the operation or revise the request to reduce the amount of information that needed to be displayed.

Discussion: This issue of user interface can be a scalability dimension. If the amount of information provided to a user grows with the "size" of the problem the system is solving, a human interface limit will be reached and a redesign of the interface may be needed. The user will perceive improved responsiveness if results of a long-running operation are provided incrementally. But that approach may require a significant architectural change, if the initial assumption was to initiate a display only when the full set of information is available.

The paper points out that different resource bottlenecks (i.e., different scalability problems) are encountered depending on whether Chimera is used by a single user on a single PC or by groups of users on networked PCs.

> Before, issues of large-scale were more closely associated with mainframe environments or high-performance systems. Now, we must more directly consider scalability in the design of small- to medium-scale systems [in networked PC environments].

Discussion: Since the developers wanted their system to be operable in a variety of network configurations (such as single machines, LANs, and WANs), they needed to use something like a product line approach in designing their system. But the different configurations present different resource bottlenecks, requiring possibly different technical solutions to achieve appropriate response.

Arlitt, M.; Diwakar, K.; & Rolia, J. "Characterizing the Scalability of a Web-Based Shopping System." ***ACM Transactions on Internet Technology, 1,*** **1 (August 2001): 44–69.**

The authors were given access to five days of workload data from a major online shopping system. In their paper, they discuss the architecture of the system and characterize the load from various sorts of users. From this analysis, they were able to determine issues affecting the performance of the system as the load increases. However, they do not provide a measure or a prediction of scalability *per se*. They advocate caching large amounts of (static) data to allow for quick responses under high loads and the shedding of personalization when the load is high [Arlitt 01].

Discussion: This paper is more concerned with how to deal with massive loads (our Definition 1) than with the strategy for increasing system capability in a repeatable fashion (Definition 2). The load-shedding strategy is a one-time fix for extending the capacity of the existing system.

Rosenthal, A. ***Toward Scalable Exchange of Security Information.*** **The MITRE Corporation. http://www.mitre.org/work/tech_papers/tech_papers_03/03_0214/index.html.**

The paper addresses the problem that security policy decisions require information from multiple sources (e.g., user authentication might be provided from an application server, clearance from a directory, and alert status from a Web service). In today's systems, each policy *decider* takes this information and determines what can be released, but that person is partnered with well-known providers and can rely on the information they provide. This tight coupling "inhibits scalability and change."

Discussion: By "inhibits scalability," Rosenthal apparently means that the decision process will become more complex and take longer as the number of policy deciders and information providers grows.

The paper also describes a decentralized way of managing an extensible set of security attributes. The proposed method would make it "easier to create new deciders" and it would also become possible "without recoding the deciders . . . to add attribute types or change where an attribute is obtained" [Rosenthal 03]. The goal apparently was to present a design that accommodates growth in certain types of workload by reducing the amount of work required.

Discussion: The paper defines scalability as the ability to handle increased workload—specifically, new providers, new attribute types, and additional request instances. Although scalable appears in the paper's title and is mentioned as the impetus for the paper, there is no analysis of how the proposed architecture would improve the ability to add new users and content to a system without requiring recoding and, presumably, without having a major impact on performance.

The paper is typical of ones we scrutinized in that it uses the word *scalable* as a generally well-understood term, without giving a clear analysis of what is meant by the term. The paper focuses on the proposed architecture; no analysis was given of how much additional workload could be processed using the new approach.

Rosenthal, A. & Seligman, L. "Scalability Issues in Data Integration." ***Federal Database Colloquium and Exposition*****. San Diego, CA, August 28–30, 2001. http://www.mitre.org/work/tech_papers/tech_papers_01/rosenthal_scalability/index.html.**

This paper discusses the difficulties of integrating new databases into a distributed collection of databases. It is unusual in that it addresses administrative rather than system performance, (i.e., whether the amount of manual action could become unreasonably large as database sources are added). The research question is how to organize the database and its services so the administrative burden of adding new elements is relatively independent of the size of the database.

Discussion: This is a Definition 1 approach to scalability because the focus is on improving how a particular activity performs as workload increases.

In explaining this problem, the authors of the paper note

> [The *Global as View* (GAV) approach] scales poorly, because GAV requires that administrators write and maintain views that reference many sources. Adding, deleting, or changing any of these sources requires inspecting a large view. If one has 2000 attributes and 1% change per month, then one has one change every working day, and each change may affect multiple views. The toughest part, for scalability, concerns knowledge of pairs of sources, [e.g., which source] is more accurate and how one can match (say) individual taxpayer identities (when TaxID numbers might be lacking or incorrect) across them [Rosenthal 01].

The authors report that the Local as View (LAV) approach has scalability$_1$ advantages.

> LAV's great advantage is in scalability of administration: each source profile is independent of the others. As result, when a new source is added or removed, or a source database schema (or its population) changes, administrators edit only the relevant source profile without having to change any of the others.
>
> The approach is also very tolerant of additions to the mediated schema – the profile of a given source remains sound though it may no longer be complete. All previous capabilities are intact; however, the system does not yet know if that source contains information relevant to the additions].

The paper has a section on "Scalability," which the authors define as follows:

> *Scalability* refers here to the *data administration* difficulties in creating and maintaining large systems (not to run-time performance): How much do you need to know to add a source, or delete one? Can one look at just the mediated schema and the source? Perhaps add just one or two neighbors for each table? Or does one have a giant view that performs a join across 30 different Foreign Tax Authority sources? Another important dimension is how much does one need to read and change if there is a small change, e.g., to a single attribute. Does the model allow you to edit just that attribute and how it relates to its federated version, or does one need to edit that same 30-table join?

In this section, the authors describe why their proposed approach is scalable$_1$ (i.e., why it reduces the administrative load as attributes are added to the database). The approach is asserted to be scalable because of the independent nature of various steps. By reducing the need for coordination with other administrators or steps in the process, the scalability achieved is related to the amount of information added to the schema rather than proportional to the number or size of the schemas in the various distributed databases.

The authors say

> [o]ur process is scalable for several reasons. First, one can edit the descriptions provided at each step, separately. There is no effect on descriptions from earlier steps, and often none on later descriptions. Automated generation can then proceed. Research is needed into algorithms that determine which descriptions are invalidated, help administrators update the information, and then regenerate the affected view information. Second, use of LAV at the initial stage partitions the problem by sources. . . . The amount of new metadata needed to introduce a new source is constant, independent of the number of existing sources. Each source table probably has a knowledgeable administrator; cross-source knowledge is rarer. We have arranged our process to maximize the amount of metadata that is unary.

They finally note some unsolved scalability problems: “If adding a source requires assessing its credibility against all existing sources, this will lead to high costs as the system scales up. Fortunately, one needs to compare only against related sources.”

Discussion: This paper is unusual in that it addresses administrative (human) scalability issues. It explains in a qualitative way why the proposed approach should reduce the administrative burden to a minimal amount, but it doesn’t project what the net gain might be for an actual application.

Prediction of Scalability

Weyuker, E. J. & Avritzer, A. “A Metric to Predict Software Scalability,” 152–161. *Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS ’02)*. Ottawa, Ontario, Canada, June 4–7, 2002. Los Alamitos, CA: IEEE Computer Society Press, 2002.

This paper introduces the Performance Non-Scalability Likelihood (PNL) metric. The goal of the metric is to determine whether a system is likely to be scalable—which the authors define as being able to continue to function with acceptable performance when the workload has been significantly increased. The measure is then used to help plan for the eventual need to upgrade server capacity in a manner that is transparent to the users [Weyuker 02].

In order to compute the PNL metric, it is first necessary to collect significant amounts of field data to derive the operational profile of the system under study. The authors claim that the cost of doing this data collection and analysis can be modest and that it is offset by the benefit derived from being able to predict the loads at which performance problems might become significant. Because it requires the collection of field data, PNL is not a suitable prediction approach for a brand new system.

Discussion: The definition of scalability is that an increased workload can be handled without a decrease in performance; this is consistent with our Definition 1.

This paper is concerned with an interval of demand in which the system performs acceptably. This interval is the region where the system performs well without any changes in things like architecture. The system is usable without modification in this interval; so some would consider it to be scalable. In this paper, the authors use models based upon measurements of field data to predict the end of this interval; they also anticipate when that point will be reached in actual operation. They then use that information to plan a seamless upgrade, presumably in advance of that point actually being reached.

Liu, Y.; Gorton, I.; Liu, A.; & Shiping, C. "Evaluating the Scalability of Enterprise JavaBeans Technology," 74–83. Gold Coast, Australia, December 4–6, 2002. ***Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC '02)*****. http://doi.ieeecomputersociety.org/10.1109/APSEC.2002.1182977.**

The kind of scalability addressed in this paper is "the performance of a system as both the client load and component infrastructure are scaled up" [Liu 02]. Clearly, the meaning of scalability for client load means that the load increases. Scaling up the component infrastructure meant varying the number of server instances on a machine, the number of machines on which server instances run, and certain JavaBean parameters. In short, the scalability relationship is that between increased demand (client load) and increased capacity (server instances, machines, and JavaBeans tuning).

Discussion: This scalability notion is consistent with our Definition 2.

The authors used the scalability metrics introduced by Jogalokar and Woodside (previously discussed in this section), with the ultimate metric being transactions per second versus dollars invested in hardware and licensing fees. The baseline was a single machine with a single server. The performance of the system (in terms of average response time and transactions per second) was then compared with a variety of other configurations, with the "scalability metric" being above 1.0 if at a given load a given configuration provided better "productivity" than the baseline configuration (i.e., the metric looked at the relative "value" of different configurations at given loads).

Discussion: This is one of the few papers we reviewed that explicitly considered a cost metric.

Liu, Y.; Fekete, A.; & Gorton, I. "Predicting the Performance of Middleware-based Applications At The Design Level," 166–170. ***Proceedings of the Fourth International Workshop on Software and Performance (WOSP 2004)*****. Redwood Shores, CA: January 14–16, 2004. New York, NY: ACM, 2004.**

This paper discusses predicting the performance of middleware-based applications at the architecture level. The prediction depends on modeling the waiting and resource usage aspects of the design. Measurements are taken on the middleware using a simple benchmark implementation. The model is then built and extrapolated to the full-up system [Liu 04b].

Liu, Y. & Gorton, I. "Accuracy of Performance Prediction for EJB Applications: A Statistical Analysis, Software Engineering and Middleware," 185–198. ***Proceedings of the Software Engineering and Middleware Fourth International Workshop (SEM 2004)*****. Linz, Austria, September 20–21, 2004. Berlin, Germany: Springer-Verlag, 2004.**

This paper is a follow-on to the previous one. The authors propose a performance prediction framework for component-based systems. The framework begins with a model that identifies

the main components of the system, noting where queuing delays may occur. The model is calibrated by mapping an architectural choice to a set of infrastructure components and their communication pattern. The components are then aggregated into computing modules. From this, a mathematical model is derived with parameters characterizing those modules. Characterization involves determining how often each component is executed. The result is an equation that predicts the performance of the specific system. Some of the parameters of this equation are tunable, but others reflect internal details of black box components.

A simple application was implemented with minimal business logic and a simple architecture and its performance was measured. Solving the model corresponding to this simple application determined the required parameter values, called the performance profile of the platform. These parameters could be substituted into the performance model of the desired application, giving a prediction. A series of experiments were conducted to determine the accuracy of the predictions. The results were encouraging [Liu 04a].

Discussion: In the three papers with the same first author (Liu), the definition of scalability, more or less, is "scalability is a measure of the flexibility of a system to adapt readily to intensity changes of usage or demand while still meeting the predefined business objectives" [Liu 02].[14] The phrase *adapt readily* refers to certain architectural changes that can be made easily; this is an example of our Definition 2.

These papers are unique in that they present the results of varying certain architectural parameters and loads to measure how a system performs. The papers focus on defining scalability in terms of performance relative to a baseline configuration and load, although they provide no analysis of what contributes to scalability improvement other than the data given by the experiments. One point made explicitly is that to be scalable, the system must continue to meet its objectives as load increases.

These papers together form the most interesting and promising work encountered during this study with respect to the measurement and prediction of the performance of a system as loading increases. Liu's thesis is on this subject as well.

[14] This definition paraphrases one first given in *Design Scalability—An Update* [Chiu 01].

Parallel Processing Scalability

Discussion: The papers in this subsection deal with scalability of parallel computing on multiprocessor platforms. The use of the term scalability in these papers tends to revolve around being able to utilize more processing power efficiently, i.e., $scalability_2$.

Luke, E. A. "Defining and Measuring Scalability," 183–186. *Proceedings of the Scalable Parallel Libraries Conference.* Starkville, MS, October 6–8, 1993,. Los Alamitos, CA: IEEE Computer Society Press, 1993.

The author points out that the definition of scalability of parallel systems is unsatisfying because it is subjective: provided there is reasonable performance on a sample problem, a problem of increased workload can be solved with reasonable performance, given a commensurate increase in computational resources. A more satisfying definition would be to define scalability as the ability to maintain optimal cost effectiveness as workload grows. The author proposes a definition of scalability based on cost effectiveness by considering the subtle relationship between processor and workload increases [Luke 93].

Discussion: This paper is interesting because of its explicit consideration of cost-effectiveness.

Vetter, J. S. & McCracken, M. O. "Statistical Scalability Analysis of Communication Operations in Distributed Applications," 123–132. *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP-01*). Snow Bird, UT, June 18–20, 2001. New York, NY: ACM, 2001.

This paper discusses parallel scalability as the ability of a parallel algorithm on a parallel architecture to effectively utilize an increasing number of processors. The authors are interested in helping users understand scalability phenomena in terms of decisions that the user makes while designing the application. They propose an automated process that uses statistical techniques to direct a user's attention to poorly scaling communication operations (i.e., communication operations that perform poorly as the number of required communications increases) [Vetter 01].

Discussion: Poorly designed communication strategies impose increasing overheads as the number of processors grows. Controlling the growth in overhead as processor size and problem size increases is one key to improving $scalability_2$ in the parallel processing world.

The process comprises of two stages. In the first stage, the user performs multiple experiments by varying the number of tasks and, possibly, the problem size. Specific timing information about call sites for all communications operations is collected. The second stage merges all of the experiment data and harvests timing information about each call site. A ratio of the aggregate time in each call site to the aggregate communication time is calculated. The

ratio and the number of tasks for each experiment are "rank transformed" (using their rank ordering instead of the actual numbers) and the correlation between the ranked ratio and the ranked number of tasks is determined. This provides an accurate and stable indicator for identifying communication operations that scale poorly (i.e., that have a significant effect on overhead as the processing load and resources increase.)

Discussion: This paper is interesting because it focuses on diagnosing sources of overhead as computational capacity and problem size increase (i.e., it is focused on the viability of the strategy for expanding computational capacity and what can be done to make the strategy more effective).

References

URLs are valid as of the publication date of this document.

Anderson 99 Anderson, K. M. "Supporting Industrial Hyperwebs: Lessons in Scalability," 573–582. *Proceedings of the 1999 International Conference on Software Engineering (ICSE 99)*. Los Angeles, CA, May 16–22, 1999. http://ieeexplore.ieee.org/iel5/6783/18169 /00841047.pdf?tp=&arnumber=841047&isnumber=18169.

Arlitt 01 Arlitt, M.; Diwakar, K.; & Rolia, J. "Characterizing the Scalability of a Web-Based Shopping System." *ACM Transactions on Internet Technology, 1,* 1 (August 2001): 44–69.

Boesel 03 Boesel, J. R., *An Air Traffic Simulation Model that Predicts and Prevents Excess Demand*, The MITRE Corp., 2003. http://www.mitre.org/work/tech_papers/tech_papers_03/boesel_ats /boesel_ats.pdf.

Chiu 01 Chiu, W. *Design Scalability—An Update*. IBM. http://www-128.ibm.com/developerworks/websphere/library/techarticles /hipods/scalability.html (September 2001).

Farlex 05 Farlex, Inc. "Scalable." *The FreeDictionary: Computer Dictionary*. http://computing-dictionary.thefreedictionary.com/scalable (2005).

Fujimoto 95 Fujimoto, R. "Parallel and Distributed Simulation," 118–125. *Proceedings of the 1995 Winter Simulation Conference (WSC 1995)*. Arlington, VA, December 3–6, 1995. New York, NY: ACM, 1995. http://ieeexplore.ieee.org/iel2/3475/10228/00478713.pdf?isnumber=10228& prod=CNF&arnumber=478713&arSt=118&ared=125&arAuthor=Fujimoto %2C+R.M.

Grama 03 Grama, A.; Karypis, G.; Vipin, K.; & Anshul, G. *Introduction to Parallel Computing.* Harlow, Essex, UK: Pearson Education, Ltd., 2003.

Howard 88 Howard, J. H.; Kazar, M. L.; Menees, S. G.; Nichols, D. A.; Satyanarayanan, M.; Sidebotham, R. N.; & West, M. J. "Scale and Performance in a Distributed File System." *ACM Transactions on Computer Systems, 6*, 1 (February 1988): 51–81.

Jogalekar 00 Jogalekar, P. & Woodside, M. "Evaluating the Scalability of Distributed Systems." *IEEE Transactions on Parallel and Distributed Systems, 11*, 6 (June 2000): 589–603.

Jupitermedia 06 Jupitermedia Corporation. "Scalable." http://www.webopedia.com/TERM/s/scalable.html (2006).

Kephart 03 Kephart, J. O. & Chess, D. M. "The Vision of Autonomic Computing." *Computer, 36*, 1 (January 2003): 41–52.

Key 06 Key Professional Media. "Scalable." *High-Tech Dictionary*. http://www.computeruser.com/resources/dictionary/noframes/nf.definition.html?bG9va3VwPTQ5ODY= (2006).

Law 98 Law, D. R. "Scalable Means More than More: A Unifying Definition of Simulation Scalability," 781–788. Washington, DC, December 13–16, 1998. *Proceedings of the 1998 Winter Simulation Conference (WSC 1998)*. New York, NY: ACM, 1998.

Liu 02 Liu, Y.; Gorton, I.; Liu, A.; & Shiping, C. "Evaluating the Scalability of Enterprise JavaBeans Technology," 74–83. Gold Coast, Australia, December 4–6, 2002. *Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC '02)*. http://doi.ieeecomputersociety.org/10.1109/APSEC.2002.1182977.

Liu 04a Liu, Y. & Gorton, I. "Accuracy of Performance Prediction for EJB Applications: A Statistical Analysis, Software Engineering and Middleware," 185–198. *Proceedings of the Software Engineering and Middleware Fourth International Workshop (SEM 2004)*. Linz, Austria, September 20–21, 2004. Berlin, Germany: Springer-Verlag, 2004.

Liu 04b Liu, Y.; Fekete, A.; & Gorton, I. "Predicting the Performance of Middleware-based Applications At The Design Level," 166–170. *Proceedings of the Fourth International Workshop on Software and Performance (WOSP 2004)*. Redwood Shores, CA: January 14–16, 2004. New York, NY: ACM, 2004.

Luke 93 Luke, E. A. "Defining and Measuring Scalability," 183–186. *Proceedings of the Scalable Parallel Libraries Conference.* Starkville, MS**,** October 6–8, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

Rosenthal 01 Rosenthal, A. & Seligman, L. "Scalability Issues in Data Integration." *Federal Database Colloquium and Exposition*. San Diego, CA, August 28–30, 2001. http://www.mitre.org/work/tech_papers/tech_papers_01/rosenthal_scalability/index.html.

Rosenthal 03 Rosenthal, A. *Toward Scalable Exchange of Security Information.* The MITRE Corporation. http://www.mitre.org/work/tech_papers/tech_papers_03/03_0214/index.html (2003).

Satya 02 Satyanarayanan, M. "The Evolution of Coda." *ACM Transactions on Computer Systems, 20*, 2 (May 2002): 85–124.

TechTarget 05 TechTarget. "Scalability." http://searchdatacenter.techtarget.com/sDefinition/0,290660,sid80_gci212940,00.html (2005).

Tran 01 Tran, P. & Gorton, I. "Analyzing the Scalability of Transactional CORBA Applications," 102–110. *Proceedings of Tools 38: Technology of Object-Oriented Languages and Systems: Components for Mobile Computing.* Zurich, Switzerland, March, 12–14, 2001. Los Alamitos, CA: IEEE Computer Society Press, 2001.

Vetter 01 Vetter, J. S. & McCracken, M. O. "Statistical Scalability Analysis of Communication Operations in Distributed Applications," 123–132. *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP-01*). Snow Bird, UT, June 18–20, 2001. New York, NY: ACM, 2001 (ISBN: 1-58113-346-4).

Weyuker 02 Weyuker, E. J. & Avritzer, A. "A Metric to Predict Software Scalability," 152–161. *Proceedings of the Eighth IEEE Symposium on Software Metrics (METRICS '02).* Ottawa, Ontario, Canada, June 4–7, 2002. Los Alamitos, CA: IEEE Computer Society Press, 2002.

Wikimedia 05 Wikimedia Foundation, Inc. "Scalable." *Wikipedia*. http://en.wikipedia.org/wiki/Scalable (2005).

Yu 00 Yu, H. & Vahdat, A. "Building Replicated Internet Services Using TACT: Toolkit for Tunable Availability and Consistency Tradeoff," 75–84. *Proceedings of the Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS2000*), Milpitas, CA, June 8–9, 2000. Los Alamitos, CA: IEEE Computer Society Press, 2000.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	March 2006	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
On System Scalability	FA8721-05-C-0003

6. AUTHOR(S)

Charles B. Weinstock, John B. Goodenough

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2006-TN-012

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

A significant number of systems fail in initial use, or even during integration, because factors that have a negligible effect when systems are lightly used have a harmful effect as the level of use increases. This *scalability* problem (i.e., the inability of a system to accommodate an increased workload) is not new. However, the increasing size (more lines of code, greater number of users, widened scope of demands, and the like) of U.S. Department of Defense systems makes the problem more critical today than in the past.

This technical note presents an analysis of what is meant by scalability and a description of factors to be considered when assessing the potential for system scalability. The factors to be considered are captured in a *scalability audit*, a process intended to expose issues that, if overlooked, can lead to scalability problems.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Scalability, system scalability, simulation scalability, parallel processing, scalability prediction, software architecture	62

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL