Designing Your System to Meet Your Requirements

Cary V. Millsap
Oracle Corporation

January 3, 1996

The technical architect of an application system is responsible for building a system that meets the goals of the end users. To succeed, this person must combine the right hardware components with suitable application architectures, always obeying complex functional, operational, and economic constraints. This job is complicated. A noticeable lack of good tools, methods, and experience have made it all the more difficult. Yet an inadequate technical architecture will *doom* an application.

Oracle's success in the mainframe downsizing market has begun to produce tools, methods, and experience that tremendously reduce technical architecture risk. This paper will identify factors critical to the success of a technical architecture design project. We will discuss successful methods used at the most demanding relational database projects in the world, and we will tell you how to use those methods to make your application succeed.
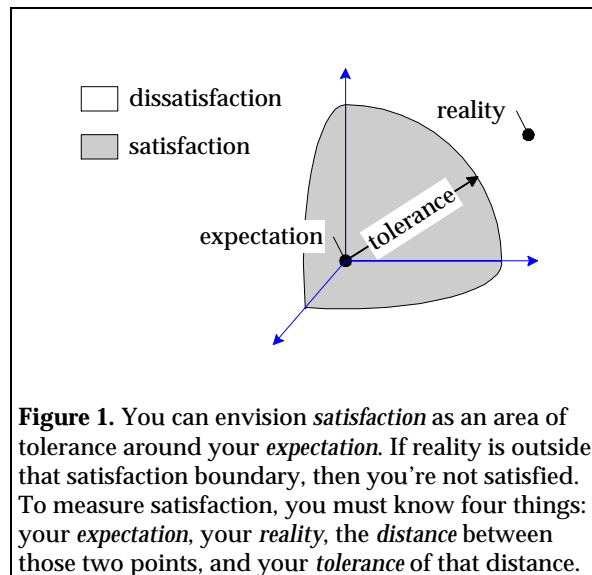
Contents

1. Introduction

Just about every vendor in the world has the words "satisfied customer" somewhere in its mission statement. The Oracle Server-based system you paid for must satisfy you. But what is *satisfaction*?

Satisfaction is something you usually just know "when you see it." However, with expensive things, like military aircraft or computer systems, "I'll know it when I see it" usually isn't a solid enough basis upon which to build a business relationship. Vendors and customers tend to see satisfaction differently when there's a lot of money involved.

Whatever satisfaction is, at the very least it's an absence of problems; and *problem* is a word that science has a grip on. A problem is a perceived difference between expectation and reality.[1] Satisfaction then is an acceptably small perceived difference between expectation and reality.

There's one troublesome word in there. *Perceived* is one of those subjective words that engineers try not to use, and that you don't usually find in contracts. Perception is the problem when you think your system is slow, and your vendor thinks it is fast. Your computer system is important enough to your business that you can't tolerate subjective measures of its success.

[1] Thank you, Dr. Ray Quiet, for giving me this tool and several others in a fall 1981 episode of introductory sociology.

Figure 1. You can envision *satisfaction* as an area of tolerance around your *expectation*. If reality is outside that satisfaction boundary, then you're not satisfied. To measure satisfaction, you must know four things: your *expectation*, your *reality*, the *distance* between those two points, and your *tolerance* of that distance.

2. Specifying Your Requirements

The way to fix this problem is to form a customer-vendor agreement up-front on what specifically constitutes satisfaction. The more *measurable* your specifications are, the less subjective your estimates of satisfaction will be. To measure satisfaction objectively, you must do the following:

1. First, identify the important attributes of your system. Perhaps for your system to succeed, it must process 50 on-line orders per minute at peak times. Perhaps your business could not tolerate an unplanned system outage of more than four hours. You must identify your system's *critical success factors* (CSFs). You probably already have an idea about what they are.

2. Second, document your expectations in precise detail for each CSF. By specifying your expectations in detail, you will be able to compare these expectations with your measurements of reality.

3. Third, specify your tolerance for deviation from these expectations. People are always more tolerant in some areas than others, so a blanket "±10%" clause, for example, probably isn't good enough.

4. Fourth, measure reality for each CSF. The most reliable measures of reality are actual hands-on measurements of a real-live system. However, if we haven't built our
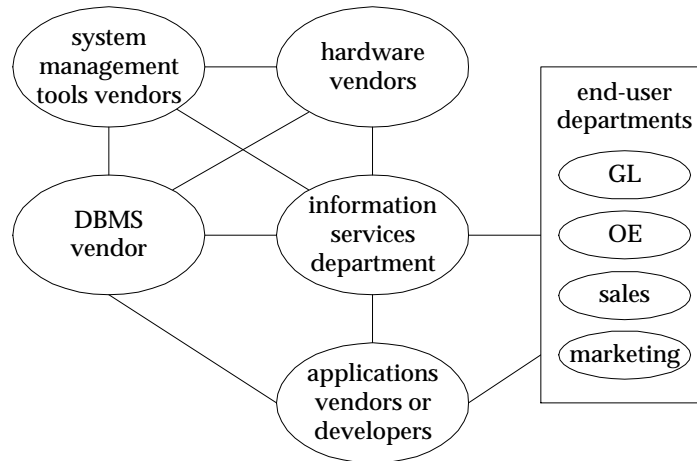
ORACLE

Figure 2. The relationships among providers and consumers of information make your information services department the arbitrator among the groups involved in your system. One function of this department will be to understand user pain and then adjust your technology cost-benefit trade-offs to remove that pain without creating intolerable pain elsewhere. If your users do not understand the costs of technology benefits, then the job is all the more difficult. The SLA construction process benefits your business by educating you and your users early about those trade-offs.

system yet, we must rely on mathematical models, prototypes, and measurements of existing systems resembling the one we hope to build.

5. Fifth, measure the reality-expectation difference for each CSF (step 4 minus step 2). This step is simpler if you've stated your expectations in the same units reported on by your system management tools used to produce your measurements of reality.

6. And last, compare your reality-expectation difference (step 5) to your satisfaction tolerance (step 3). If your system meets all of your requirements, then you are in good shape; otherwise, you either have to tune something, relax a requirement, or buy more stuff.

2.1 Service Level Agreement (SLA)

A *service level requirement* is a formally specified end-user expectation about your system. A *service level agreement*, or *SLA*, is a collection of service level requirements that have been negotiated and mutually agreed upon by your information providers and your information consumers (Figure 2).

A useful SLA is a contract with three attributes:

- *Structure*—Your SLA mustn't leave anything out; no service level requirement that will be important either to the service organization or its customers may be omitted.

- *Precision*—To measure satisfaction objectively, each service level requirement should be expressed in the same units that will be used to measure the real system.

- *Feasibility*—The collection of requirements must obey the physical laws of nature; service level requirements must not contradict one another or lie outside the constraints of your technology.

2.1.1 Structure

Structure allows both the service organization and its customers to manage risk. Most risk comes in categories that the designers of a system just never considered. For example, Oracle customers who never consider their system up-time and downtime requirements before they buy their hardware later have to rely on luck to meet their "availability requirements."

Your computer system requirements and constraints come from three directions:

- *Functional requirements*—End-users specify functional requirements. These include specification of the kinds of tasks the sys-

tem must perform, how long those tasks can take, and at what times the system must be on-line to perform them.

- *Operational constraints*—Your system managers specify operational constraints like the schedule of planned downtime, the methods for requesting repair of unexpected functional or operational errors, and limits on throughput required to meet user response time demands.

- *Economic needs*—When the functional requirements and operational constraints don't meet, your economic needs determine the trade-offs. Economic needs are usually negotiated at the executive level where company strategy and spending authority meet [Boar].

2.1.2 Precision

Your SLA will be a valuable system design tool if you specify your expectations in units that you can measure and compare to vendor spec sheets. For example, "50 orders per minute" at your company doesn't equal "50 orders per minute" at my company because we use different application setup options, and we have different numbers of lines per order. Hence there's no spec sheet that will reliably tell you how big a computer to buy to process "*n* orders a minute." However, if you can convert your "*n* orders per minute" to disk read and write calls per second, bytes of memory per user, and Pentium CPU milliseconds per order, then you have information to help design a system.

2.1.3 Feasibility

The final necessary attribute of an SLA is feasibility. Building a feasible SLA requires commitment of both the information providers and the information consumers to engage in the cost-benefit analysis.

Without technology advice, users tend to over-constrain their requirements. For example, users commonly specify without flinching that their database system must have full "7 x 24" availability (up 7 days a week, 24 hours a day), not realizing that even 99.8% of true 7 x 24 access would require geographically replicated multi-way clusters of perfectly designed applications with three or more very smart and very expensive system administrators working 8-hour shifts through the night at each site using

very expensive custom-designed monitoring tools. They don't realize that sacrificing just one down weekend night each week would save probably 95% of the cost.

On the other hand, when you put technologists in a room without users, they tend to over-engineer systems. I once sat in a four-hour user-free meeting in which a bunch of us smart guys added over a million dollars to proposed system's price trying to meet a strict end-user throughput requirement to run several batch programs in a peak time window. After three and a half hours, our user manager joined us and said that if that's what it would cost, he'd run the batch load at night. End of meeting.

Your business and your technology will impose constraints upon your system. Creating a feasible SLA requires time from the people who know how to negotiate the trade-offs among those constraints.

2.2 Making Your SLA

SLA construction is a negotiation process, not so much among groups of people as between one team of people and the immutable conservation principles of physical law that require a cost for every benefit. In the process, you try to sacrifice only those things that you don't really need so that you can afford all those things you really do need. People who buy groceries have to do this every day. So do computer system architects.

2.2.1 Participants

The SLA is a binding contract between information providers (like your information services department and your application development departments) and information consumers (like your end users' departments). To create an SLA that works, you need dedicated participation from the following people:

- *System architect*—The project leader for SLA construction is usually an experienced system architect, either from your staff or from an outside consulting firm. This leader is responsible for managing the SLA construction project or sub-project.

- *Operations manager(s)*—These participants must have the authority to make cost-benefit compromise decisions for the entire information services department. They must understand the economic implications

Technical response commitment	What the end-users must expect	What the IS department must expect
Fast responsive-ness	Lowest possible cost by intelligent use of resources	Funding for sufficient high-performance hardware
	Response times guaranteed to fall below negotiated limits, as a function of required throughput	Resource consumption rates guaranteed to fall below negotiated limits, as a function of response time requirements
High availability	Uptime performance guaranteed to meet or exceed negotiated limits, as a function of business need	Realistic uptime requirements that allow for scheduled maintenance Funding for fault-resilient hardware, software, procedures, and people
	Downtime guaranteed to fall below negotiated limits, as a function of business need	Realistic downtime requirements that allow for unscheduled repairs Transaction durations guaranteed to fall below a negotiated limit

Figure 3. A tool to assist you in constructing your SLA is a worksheet like the one shown here. The format may remind you of a T-account worksheet if you have some accounting in your background. The principal idea of this format is to encourage the matching of complementary expectations among the departments involved. You may need another worksheet like this to help construct the service level requirements between your IS department and your application development department, another for your IS department and your hardware vendors, and so on.

of user requirements upon the operational complexity of the system.

- *End-user manager(s)*—These participants must have the authority to make cost-benefit compromise decisions for the entire end-user community. They must understand the economic implications of system operational restrictions on the business.

- *Oracle Server technician(s)*—These participants must understand the power and limitations of the Oracle Server system upon which your applications are made. They must understand the economic implications of your proposed throughput, availability, volume, load, and data locality requirements on the server.

- *Application technician(s)*—These participants must understand the technical architecture of the applications being considered. These could be vendors, end-users, or people from your own applications development department. They must understand the economic implications of your requirements upon the design (or redesign) of the applications.

- *Hardware technician(s)*—You may or may not require dedicated hardware specialists for the duration of the process, but throughout the SLA construction project, you will require information about CPU, disk, memory, bus, and network capacities of any hardware being considered.

Building an SLA will require dedicated time from some of your company's busiest and most important people. Your return on investment will be the information you need to optimize your trade-off decisions as you design your system. A good SLA gives you the specific, realistic specifications and tolerances you need to measure your information services department's success.

2.2.2 Format

As you build your SLA, remember the give-and-take, trade-off orientation of a cost-benefit analysis. For each category of service provided to your end-users, ask the questions:

- *What must the end-users expect of the information services department?*

- *What must the information services department expect of the end users?*

The worksheet in Figure 3 shows some of the trade-offs that should be communicated in the SLA.

There is a comprehensive SLA document description and outline in [Kern and Johnson]. The technical topics you will see in this paper fall predominantly into the "Technical Response Commitments" section of that outline.

3. Understanding Your Technology

To create a feasible SLA, you must understand your technology. Is it feasible to "require" that your system be as fast when a thousand users are logged in as when you're the only one on the system? Maybe, maybe not. Is it feasible to "require" that your Oracle Server system never be down longer than an hour when one of your disks crashes? Maybe, maybe not. The following sections will help you understand the factors involved in determining whether requirements like these are feasible for your system.

3.1 Functional Requirements

People who build or buy software generally understand very well that some of the most important selection criteria have to do with whether the application *does* the things it's supposed to *do.* An accounts payable system has to write checks properly or you wouldn't even consider it. An electronic mail system has to handle unexpected system outages without losing messages, or you wouldn't consider it.

We will not discuss these kinds of functional constraints in this paper, but we will focus on *performance*, a very important functional requirement that is often overlooked because forecasting performance is a difficult technical challenge.

3.1.1 Performance

An on-line user measures interactive performance as *response time,* the time required to see your information after pressing the "enter query" key, or the time it takes to regain control of your application after pressing the "commit" key. A user has different response time expectations for different application functions. For example, interactive validation of an Oracle Accounts Payable accounting flexfield should happen almost instantaneously, but a General Ledger trial balance report may perform acceptably well if it prints within a half hour of when you submit it.

Total response time for any system component is the sum of the time consumed by that component to execute your request (called the *service time*) plus the time spent waiting (called *queueing delay*) for that component's attention. For example, if you request to read a byte on a disk drive that is busy serving another request, your request must wait for that request to finish. Your total response time is the time spent doing your work, plus the time spent waiting. The waiting gets worse as the popularity of the resource increases.
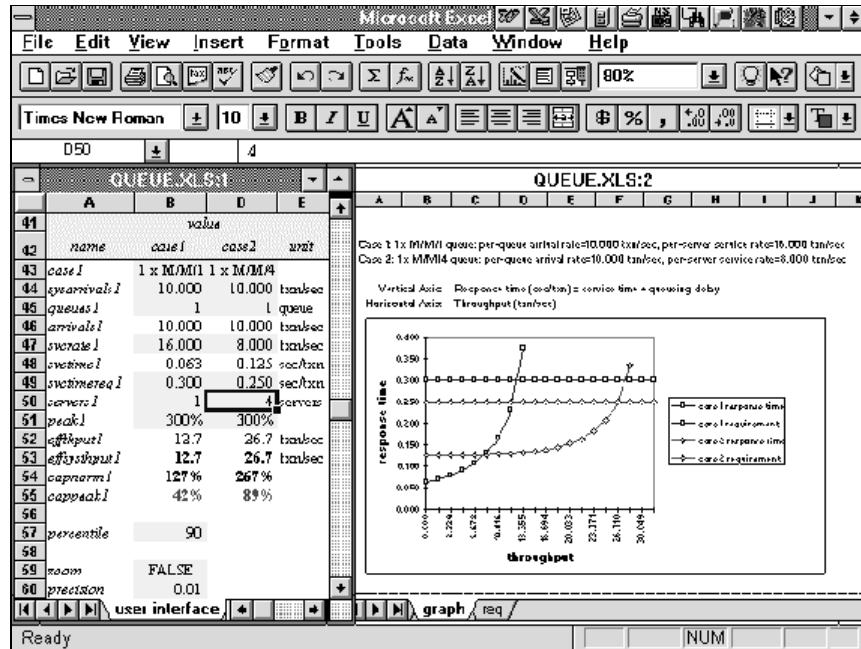
ORACLE

Figure 4. This is a screen shot from the queueing model that Oracle consultants use to forecast response time as a function of throughput. Model parameters include the unloaded service time and the response time requirement for the component being modeled.

The branch of mathematics called *queueing theory* gives us some of the tools we need to compute expected wait times. Sophisticated formulas in excellent texts like [Jain] and [Menascé et al.] can forecast your response times if you can feed them your service times and your throughput requirements. Oracle consultants use these formulas with our system measurement tools in our system design and capacity planning projects (Figure 4).

An application end-user response time is the sum of several component response times, including:

    presentation management
        CPU service time
        I/O service time
        queueing delays (for multi-user clients)
    network
        service time
        queueing delay
    transaction processing monitor
        service time
        queueing delay
    database server
        CPU
            service time
            queueing delay
        I/O
            service time
            queueing delay

Understanding this list will help you create feasible performance requirements in your SLA.

- *Presentation management*—Will your end-users' PCs have fast enough CPUs and I/O subsystems to run your application front-end? The easiest way to predict this is to test and measure a prototype application.

- *Network*—Will your network give you fast enough communication between the application front-end and the database server? Will your network have sufficient throughput capacity to meet your response time requirements as concurrency hits peak levels?

- *Transaction processing monitor*—Using a TP monitor will increase the number of instructions that your system must execute for each transaction (your *code path*). Will the resulting service times grow your total response times beyond your tolerances? You may have to make up the difference by buying faster network or database server hardware.

ORACLE

- *Database server*—Will your database server be fast enough to meet your response time requirements? Different CPU architectures are optimized for different requirements. For example, fast single-processor systems execute large single-threaded transactions quickly, but symmetric multiprocessing (SMP) computers generally handle higher OLTP throughput. What about I/O? Will your disk drives be fast enough to meet single-user response time requirements? Will you have enough controllers and drives that performance won't degrade under peak loads?

Application design trade-offs pervade each of these areas. You will save money on client hardware if you use a simpler character-based interface instead of a graphical user interface (GUI)—but your best economic benefit may be to spend money to improve user productivity. You may be able to save money on network hardware by configuring your application to generate less network traffic—but your best economic benefit may be to ensure fast, reliable remote access to data. You may be able to save money on database server hardware by tuning your application to reduce code path length and I/O call counts—but new hardware may cost less than tuning your application.

If you've ever tried to compress a water balloon with your hands, then you have a good mental image of how you can squeeze here or you can squeeze there, but you can't squeeze everywhere at once. You have to live within your technology's constraints. You can't compress water, and you can't get more out of your system than its peak capacity. You've got to decide what your goals are, understand your trade-offs, and invest intelligently.

3.2 Operational Constraints

It's easy to list the things you *want*—it's harder to list what you can *afford.* Operational realities of technology constrain your functional requirements. Accurately documented operational constraints will keep the functional requirements in your SLA from raging beyond what your information services department can actually provide.

To identify operational constraints accurately, you need experienced people who understand the complex operational requirements of Oracle applications. In this section we will discuss some of the operational constraints you must know to construct a feasible SLA.

3.2.1 Availability

An operations manager measures system availability with two quantities, *MTTF* and *MTTR. Mean time to failure (MTTF)* is a measure of how long a system or component is on-line and ready for action. *Mean time to repair (MTTR)* is a measure of how long it takes to repair and restart a system or component once it goes down. MTTF is average uptime, and MTTR is average downtime.

To specify structured, precise, and feasible availability requirements, you must understand all the ways your system can fail [Gray and Reuter].[2] For example:

    environment
        fire, flood, network, electrical failure
    hardware
        physical device failure (CPU, disk, etc.)
    maintenance
        hardware repair errors
    operations
        configuration, administration errors
    software
        O/S, DBMS, and application bugs
    process
        data entry error, labor disputes, etc.

For each of the faults that can bring your system down, you must understand the process required to recover from that fault before you can create a feasible downtime requirement. And you must understand the MTTF for each of your system's components before you can create a feasible uptime requirement.

Let's walk through the recovery process for an example unplanned outage. Let's explore disk drive failure. How often can you expect this to occur, and what is the expected impact of each outage?

The MTTF of a non-RAID disk farm is easy to calculate once your hardware vendor tells you the MTTF rating on each of the disks. If each drive had a 200,000-hour MTTF,[3] then a 50-

---

[2] You should not embark upon designing a fault-tolerant transaction processing system without owning and studying the Gray and Reuter book.
[3] Ask your disk drive vendor about the MTTF for specific models. These figures vary dramatically for different disk drives.

ORACLE

drive system would have a 4,000-hour MTTF—roughly five and a half months.[4] This means you could expect two unplanned disk crashes a year without RAID. MTTF for the 50 disks mirrored in a RAID level 5 configuration would be over 3,000 years [Chen et al.].

Now let's calculate disk crash MTTR.[5] This requires understanding of the recovery process. Let's assume that you've wisely invested into the operational overhead of scheduled hot backups that will allow you to recover your database to your users' most recent commit. The recovery process then looks roughly like the following:

> Shut down the Oracle application and the Oracle Server.
> If the disk can be repaired, repair the disk. Otherwise, salvage all the files you can from the corrupt disk, replace the disk drive, and replace the salvaged files.
> Restore from tape the most recent hot backup of the Oracle database files that were corrupted.
> Restore all of the archived redo log files generated since that hot backup.
> Initiate the Oracle Server startup process.
> Wait for Oracle to roll forward through the archived redo logs.
> Wait for Oracle to roll back all pending uncommitted transactions.
> Notify your users that they may resume their work.

By practicing this process on a prototype system, you will learn how long each of these tasks will take in your environment, and you will learn how to perform some of these tasks simultaneously to save time. Disk failure MTTR will be the elapsed duration from the start of the first task to the finish of the last.

Note that each task in the recovery process outlined above bears risk—for example, what if one of the tapes you need for restoration is cor-

rupt? It happens. When it does, you must use an alternate recovery plan that has a much higher MTTR because it requires users to redo work they thought they'd finished. Using tape reliability (MTTF) statistics, you can calculate how often you can expect this event to happen to you. The economic impact of this possibility may motivate you to multiplex your tape backups. You must analyze each step in your recovery process for possible surprises like this.

As you study the recovery process for the disk failure event, you'll notice that MTTR depends heavily upon economic trade-off decisions. You can reduce MTTR for any event by incurring cost somewhere else. Depending on your desired level of investment, your MTTR for disk crash outages might be anything from minutes to hours.

For example, in our disk crash example, you could reduce MTTR by reducing the elapsed time of the roll-forward step: simply take hot backups more frequently. What are the costs? You would have to buy more tapes; you'd have to construct more complex tape management procedures; you'd pay increased backup process labor costs; and you would endure marginally degraded database performance more often during the more frequent backups.

Your information providers' real job is to use ingenuity to find ways to reduce the cost of your benefits. For example, you can get many of the benefits of more frequent hot backups by backing up your most heavily updated Oracle tablespaces more often than you back up your infrequently updated ones.

For each type of system failure you can encounter, have your vendors help you compute MTTF. To compute MTTR, create a miniature project plan for the repair process using a project planning tool. After you quantify your reliability (MTTF) and service interruption (MTTR) statistics, you can quantify the economic impact of the requirements that you document in your SLA. As you implement your system, test your estimates and refine your SLA to reflect the actual facts of your experience.

3.3 Economic Needs

Your users want 7 x 24 system access. Your operations staff say no way unless you're willing to spend several millions of dollars each year on the system. Does your business need real 7 x 24

---

[4] See Gray and Reuter's equation 3.5.

[5] The Oracle market trend today is to use mirrored disks, which, as we've seen, have extraordinarily high MTTF ratings. Some RAID system managers consider the reliability to be so good that they don't even formulate a disk recovery plan. Although RAID reliability is excellent, RAID systems can still crash and require Oracle Server media recovery. So even if you will use RAID, you should still analyze your disk corruption MTTR.

access? If your system is a general ledger for a company with offices in two mainland USA time zones, then probably no. If your system manages over 10 million reservations per day for a global airline with customers calling from every time zone on the planet, then definitely yes.

Economic need—your company's unique mixture of investment aggressiveness and cost consciousness—determines the right answer to trade-off decisions. Do you go with a less expensive disk system with a 6-month MTTF, or do you buy a more expensive system with a 3,000-year MTTF? What do you do if your system is too slow? Do you tune your software, upgrade your hardware, or negotiate relaxed functional requirements? These are all economic decisions, and the ingenuity you need to make them will come from understanding your technology.

> *Understand your technology before you aspire to make optimal economic decisions about your system.*

The best way to understand your technology is to get hands-on experience with it as quickly as possible. Don't assume that you understand your system's response time performance until you've tested it under varying throughput loads. Don't dare assume that you understand your system's recovery processes until you've studied and tested them at your site. You wouldn't assume that a custom-built or pre-packaged application does what you need without studying it. Don't treat your other functional and operational specifications any less seriously.

4. Designing Your System

After you have specified your cost-benefit balanced requirements in your preliminary SLA, you are ready to begin choosing hardware. The following sections will orient you to many of the issues you must consider when selecting your system components.

4.1 Disk

Specifying disk hardware is not just an exercise in figuring out how much data you will need to store. In addition to data volume requirements, you must consider:

- How many I/O calls per second will your application generate? You must design your system to have enough disk drives and controllers that your I/O subsystem won't cause unacceptable delays as you increase the load on your system. Measure your application or a prototype to learn your throughput requirements. Ask your hardware vendor about your disk drive and controller specifications. You must buy enough devices to keep individual component utilizations at acceptably low levels [Jain].

- How much hardware fault resilience is required to meet your uptime and downtime requirements? Can you withstand the failure rate of un-mirrored disks, or do you need to purchase a more expensive RAID configuration?

4.2 Memory

Measuring memory consumption is well-documented [Loukides]. Although memory is one of the easiest system components to specify accurately, there are a few factors that make the task tricky in spots:

- All operating systems give you the ability to measure the size of a program's text, data, and stack areas. UNIX, for example, gives you the **size** command. However, most well-designed, complex applications allocate memory dynamically at run-time commensurate to memory need of a particular user's invocation.[6] You cannot forecast dynamic memory consumption using a static command like **size**—you must measure this memory usage operationally on a system in use. If your system doesn't exist yet, then you must measure using a prototype, or you're confined to making educated guesses.

- It may be difficult to get an accurate report of actual memory usage. For example, many implementations of the UNIX **ps** command do not report memory consumption accurately. Most vendors have tools unique to their systems for measuring memory consumption, but you often have to ask specifically for these tools before you get to use them.

---

[6] If you're a C programmer, we're talking here about programs that do a lot of **malloc()** function calls.

ORACLE

• You must understand your operating system's implementation of memory sharing for program text. Your hardware vendor or your Oracle consultant can help you understand how to count memory accurately without double-counting shared segments.

4.3 CPU

CPU is the most difficult system component to size. The root of the difficulty is that no useful transaction unit exists for CPU specifications. With disk drives, we can compare milliseconds-per-call response time specs and bytes-per-second throughput specs directly to your application's requirements. With networks, we have milliseconds-per-transmission response time specs and bits-per-second throughput specs. But with CPUs, all we have are benchmark statistics that allow you to compare one CPU to another. These numbers don't reliably tell you what kind of CPU you need to buy for your unique application.

Computing a system's CPU response time and throughput capacities is expensive. The lowest risk method is to measure a system's performance while it is running *your* application with *your* setup and *your* configuration. You can reduce your cost for this information with prototypes and models if you know your technology well enough minimize the risk of missing something important.

Guessing is a high-risk method that has worked successfully for 75% or more of our customers. Guessing works okay for you if you can be certain that the machine you will buy will easily outperform your requirements. If your requirements are sufficiently easy to meet, it may cost less to buy an overpowered computer than to figure out exactly how much power you really do need. But if your requirements are tough, guessing won't be good enough.

Prototype testing yields the best cost-benefit balance to reduce risk at sites who need the most from their technology. Several tools and services exist to help you accomplish this testing, including:

• Measurement and recording tools let you see the service and delay times of your application's resource requests. Many vendors supply these tools. Oracle consultants use almost all of these tools in addition to

several produced by Oracle's System Performance Group.

• Remote terminal emulators let you simulate user loads without having to have several hundred of your closest friends come use your system while you measure it.

• Queueing models allow you to predict the response time degradation effects of adding load onto a system. A portion of Oracle's queueing model tool is shown in Figure 4.

• Discrete event simulators forecast the behavior of your system given different configurations.

• And several service providers, including Oracle, can bring technology experience to your project to help you get the information you need at the lowest long-term cost.

In addition to performance considerations, you must also consider computer node fault resilience. You must analyze your CPU system MTTF and MTTR statistics, like we've done in this paper for disk drives, to determine whether your business will require redundant computer nodes to minimize the impact of unplanned CPU outages.

4.4 Network

Specifying network hardware is much like specifying disk hardware. You must consider:

• How fast must the network's response time be to supply satisfactory round-trip response times to applications making network requests?

• How many transmissions will your application require for each database transaction? You must design your network's throughput capacity to meet your response time requirements as concurrency hits peak levels.

• How much additional network hardware will you need to purchase to meet your network availability goals? Answering this requires an MTTF and MTTR analysis similar to the analysis for disk shown earlier.

Designing your network will require the services of an experienced network engineer to meet all of your networking goals. Understanding the Oracle application and database server re-

quirements upon the network will require Oracle-specific experience.

5. Conclusions

In this paper, we've developed the following line of reasoning:

- Your computer system is much too important to risk measuring your satisfaction subjectively.

- To measure your satisfaction objectively, you must specify your expectations and tolerances, and then continually measure them against reality as expectations and reality evolve.

- To understand your expectations, you should write them down in a document called your *service level agreement*—the SLA.

- Your expectations must be structured, precise, and feasible or you'll encounter an unrelenting barrage of nasty and expensive surprises.

- To create structured, precise, feasible expectations, your company's best people must participate.

- You must understand your technology before you aspire to make optimal economic decisions about your system.

In addition to these principles, we've discussed specific technical issues, including:

- You have seen an approach to structuring your SLA for Oracle applications so that you will not overlook categories of technical response commitments.

- You have seen how to evaluate the feasibility of Oracle system performance and availability requirements.

- You have seen how to begin the selection process for CPU, disk, memory, and network hardware to suit your expectations.

- You have seen references to further reading from which you can learn more about the topics presented here.

6. References

BOAR, B. *The Art of Strategic Planning for Information Technology*. John Wiley & Sons, New York NY, 1993.

CHEN, P.; LEE, E.; GIBSON, G.; KATZ, R.; PATTERSON, D. "RAID: high-performance, reliable secondary storage" in *ACM Computing Surveys*, Vol. 26 No. 2, June 1994.

GRAY, J.; REUTER, A. *Transaction Processing: Concepts and Techniques*. Morgan Kaufman, San Francisco CA, 1993.

JAIN, R. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, New York NY, 1991.

KERN, H.; JOHNSON, R. *Rightsizing the New Enterprise*. P T R Prentice Hall, Englewood Cliffs NJ, 1994.

LOUKIDES, M. *System Performance Tuning*. O'Reilly & Associates, Sebastopol CA, 1991.

MENASCÉ, D.; ALMEIDA, V.; DOWDY, L. *Capacity Planning and Performance Modeling*. P T R Prentice Hall, Englewood Cliffs NJ, 1994.

About the Author

Cary Millsap is the director of Oracle's System Performance Group, a part of the Oracle Services Advanced Technologies group. The team is responsible for building new tools and capabilities like the ones described in this paper for Oracle and its customers. The System Performance Group provides system design, capacity planning, and performance management services to customers worldwide.

Since joining Oracle in 1989, Mr. Millsap has worked with over 100 Oracle customers. He has published several papers, developed and taught internationally acclaimed courses, and he is the author of Oracle's Optimal Flexible Architecture (OFA) standard.

ORACLE