The Association of System Performance Professionals

The **Computer Measurement Group**, commonly called **CMG**, is a not for profit, worldwide organization of data processing professionals committed to the measurement and management of computer systems. CMG members are primarily concerned with performance evaluation of existing systems to maximize performance (eg. response time, throughput, etc.) and with capacity management where planned enhancements to existing systems or the design of new systems are evaluated to find the necessary resources required to provide adequate performance at a reasonable cost.

This paper was originally published in the Proceedings of the Computer Measurement Group's 2001 International Conference.

For more information on CMG please visit http://www.cmg.org

System Performance Management and Capacity Planning[ℵ]

Dr. Bernard Domanski, College of Staten Island/CUNY
John P. Pilch, Performance Capacity Solutions
E. Jeanne Diaz, Merrill Lynch

Abstract

This tutorial was designed to introduce the manager/analyst new to the field to a variety of techniques and methodologies for *capacity planning* and *performance evaluation*. In particular,

- *the system life cycle,*
- *benchmarking,*
- *modeling techniques,*
- *workload characterization* and
- *performance management*

will be discussed. Thus, the primary purpose of this tutorial is to provide an introduction for the novice and an overview for the manager.

The presentation will be divided into three sections. They are:

- A General Systems Approach to Performance Management and Capacity Planning.
- Workload Characterization and Prediction Techniques.
- Simple Performance Calculation

This paper is intended to expose some of the many facets of Computer Performance Evaluation (CPE), and augments the technical content of the presented sessions at the conference.

1. Introduction

Large computer systems incorporate many features such as multiprogramming, database management, virtual memory and complex operating systems. These features make performance management and capacity planning of these systems particularly difficult. The purpose of this paper is to outline those aspects of capacity planning and performance management that are the most critical in viewing a complex computer system. Topics to be covered include the system life cycle model, benchmarking, modeling techniques such as operational analysis and queuing theory, simulation, workload characterization and performance management. It is our hope that enough information and references will be provided so that the reader may peruse topics of interest on their own.

2. The System Life Cycle Model

A systems' life cycle can be represented in many ways. Figure 2.1 is one model of the systems life cycle.

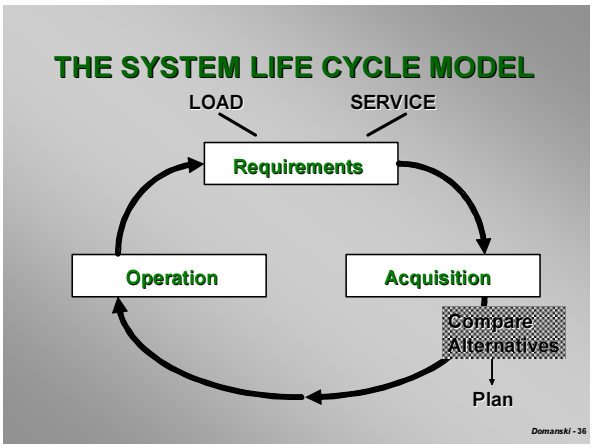


Figure 2.1

Expanding on this model, we note that during the requirements phase we need to understand the role (objectives) of the system, the load the system must support and the service the system is required to provide. During this phase, one must define more clearly the functionality of the desired system.

Once the requirements are identified, we begin to enter a phase of acquiring the needed resources to support the system requirements. Here we

usually must compare among a set of alternatives over the entire expected life of the system. That is, as the system matures, the load supported will normally increase and the resources required to support this increased load should be identified during the acquisition phase. Thus, a plan should exist to support the system requirements over its expected life cycle. In essence there are two "types" of comparisons going on here:

1. For each alternative "vendor" or set of resources, one must compare alternative changes in the set of resources (more memory, disks, etc.) over the life of the required system, thus producing a plan.

2. One must also compare alternative "vendor " plans.

After we have compared and selected/obtained the needed resources, we must operate the system including all the resources acquired to support the system. In essence, we must manage the system so that it meets its required goals in a timely fashion given the load. We, therefore, must manage the set of resources to ensure that the resources provide the required service given the load. This is sometimes referred to as Performance Management.

Figure 2.2 is an expanded view of the System Life Cycle.

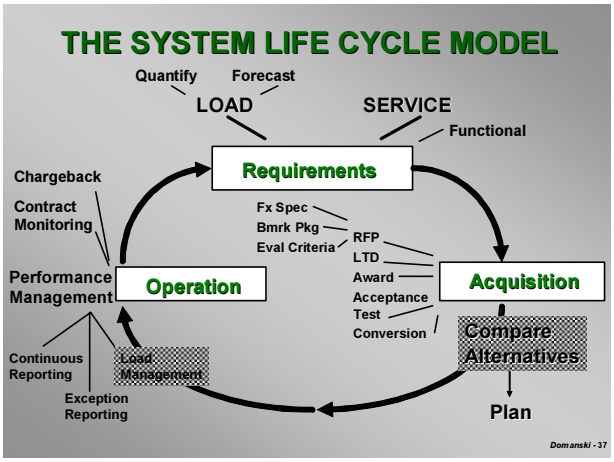


Figure 2.2

Performance Management then consists of measuring the system's service during its operating phase and comparing the observed service measurements against the planned or required service. When the observed differs enough from the planned, some action is triggered. Normally one would tune the set of resources in some way first to improve the overall system service. If tuning was not successful or possible, one would enter into a major planning activity usually to acquire additional resources to support the system.

Figure 2.3 represents the Performance Management process within the larger system life cycle model.

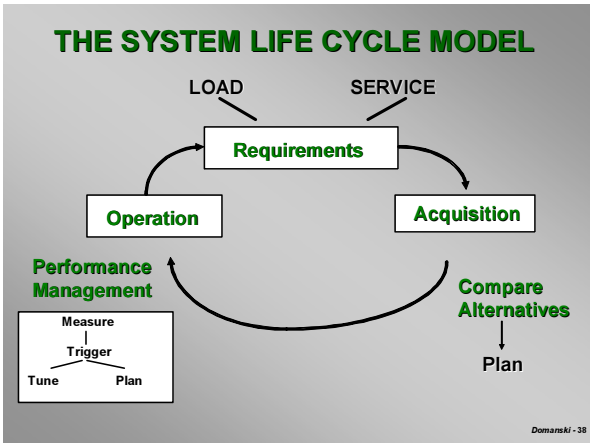


Figure 2.3

During the tuning process one needs to have several alternative optimization approaches suggested by the measurement data. At least conceptually one must compare these choices and select an alternative to implement. Also, during the triggered plan or re-plan process, one needs to have several alternative implementation plans from which to choose. Again, one must compare among a set of alternatives.

Capacity Planning looks at business requirements that need to be filled, and through understanding and analyzing the workloads to be run and the service (e.g. response time) to be delivered, details the physical resources (i.e. capacity) needed (Figure 2.4).

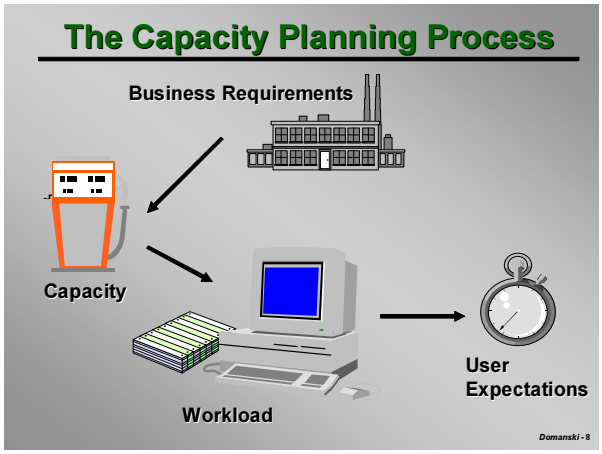


Figure 2.4

The process is straightforward - evaluate alternative configurations at different costs to determine if they can process the projected workloads while still delivering acceptable service as shown in Figure 2.5.

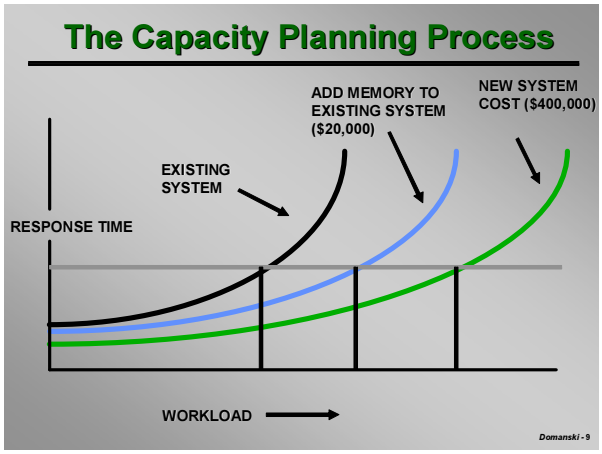


Figure 2.5

3. Benchmarking

A *benchmark* is defined by Webster as a point of reference for measurement. The act of obtaining the point or point(s) of reference is defined as benchmarking.

An important aspect of benchmarking is a clear experimental design. One needs to have identified the questions being addressed, the definition of the system boundaries of interest, a representation of the load for which the system is to respond and a means by which the load can be presented to the system under test and service data collected.

If, for example, we were comparing two processors to answer the question, "*which processor does the move-character operations more effectively?*", then the system boundaries could be the processor and memory and the load needed to represent the move-character operations. Let us assume the load consisted of a set of programs composed of different quantities of move-character operations, then this set of operations would be presented to each of the processors and we would measure how long it took the set to complete (figure 3.1). The operations would be presented to the system as machine instructions.

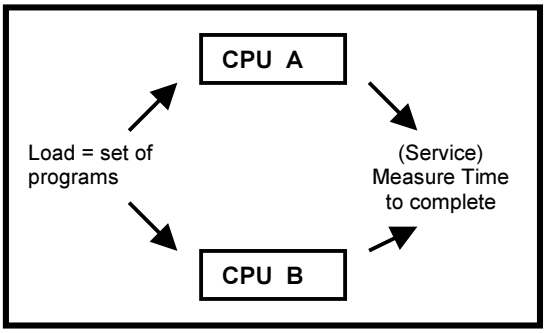


Figure 3.1

If the question were to find out which system does matrix inversions the best, then the system boundaries change and the representation of the load changes. For example, if the matrix inversion involves using a ***C++*** program to do the matrix inversion, then, we are interested in how efficient the ***C++*** compilers are on the alternative system(s). One would be inclined to take the original ***C++*** program and have each system generate the compiled object code. Then one would simulate each system in order to measure the service delivered by timing how long it took for each systems' object code to run. The differences in the observed measured data (across systems) might be due to the differences in hardware or may be due to the efficiency of the ***C++*** compiler(s) to generate good object code. As long as the original question was to determine which system did the best ***C++*** based matrix inversion, the measured data would present a reasonable estimate.

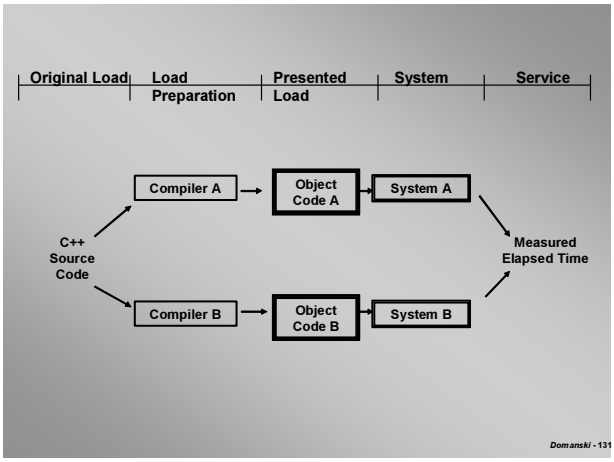


Figure 3.2

As we better understand the work that the final system is to do, we will be more able to represent the load and the means of loading and measuring the systems(s). Let us assume that we have the representative workload and we are interested in determining the configuration(s) for a timesharing system. The question becomes: What system configuration will support the load (provide acceptable service) over the systems life?

This leads us to generate a set of sub questions that come from hypotheses about the system being considered. That is, knowledge about the computer system allows us to think of alternative configurations.

Question: What initial system configuration will support the load?

Sub questions:

Initial configuration

- memory needs?
- processing needs?
- storage needs?
- communication needs?
- software needs?

From these sub questions, a set of alternative initial configurations would be generated. It is these alternative configurations that we must obtain load-service relationships to compare these alternatives. It is a result of comparing these alternatives that we really develop the computer capacity plan. We'll assume that several alternative configurations are proposed for evaluation. Figure 3.3 represents key alternatives that must be understood to select the initial configuration.

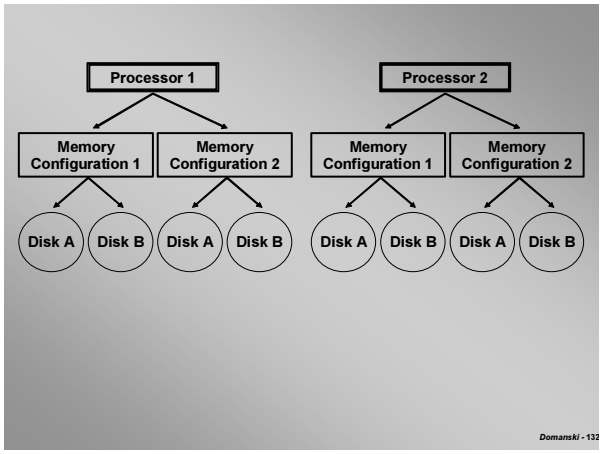


Figure 3.3

That is, to select the initial hardware configuration, we ideally need to compare the eight possible configurations shown in Figure 3.3 (where a configuration is a combination of processor, memory configuration, and disk type) using load-service relationships (curves) as shown in Figure 3.4

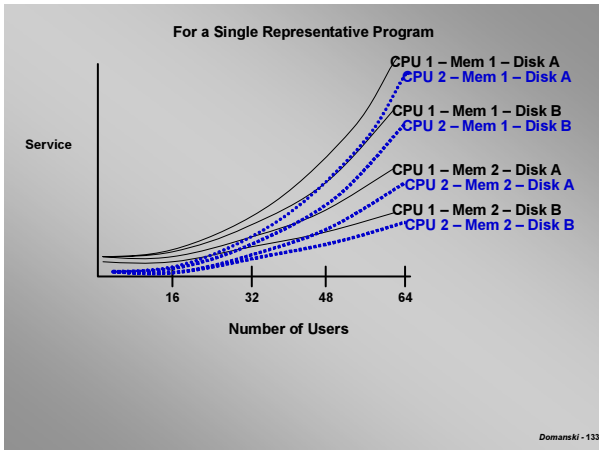


Figure 3.4

To create the load-service relationships, we must collect response time values for selected user levels and hardware configurations. From Figure 3.5 we see that for a single representative program that 32 "tests" might be performed to collect the desired data to make the comparison.

	Processor 1				Processor 2			
	Memory Config 1		Memory Config 2		Memory Config 1		Memory Config 2	
# of Users	Disk A	Disk B	Disk A	Disk B	Disk A	Disk B	Disk A	Disk B
16								
32								
48								
64								

Figure 3.5

If we decide to collect this data from a live system, we might need about 32 hours of test time. We

would also need a way to present the load to the system configurations under test and measure the time it took to process the load. This is usually done with an *RTE* (Remoter Terminal Emulator).

Remote Terminal Emulation is an approach to the testing and evaluation of computer systems in which a *driver* program is implemented (usually externally but not necessarily) to and independent of the system being tested. The driver, called the Remote Terminal Emulator (*RTE*), applies a specified workload to the system such that it appears to the System Under Test (*SUT*) that it is connected to actual users. The objective of remote terminal emulation is to drive a SUT in a way that is controllable, repeatable, economical and realistic. Figure 3.6 is a schematic of the flow of information in a remote terminal emulation experiment.

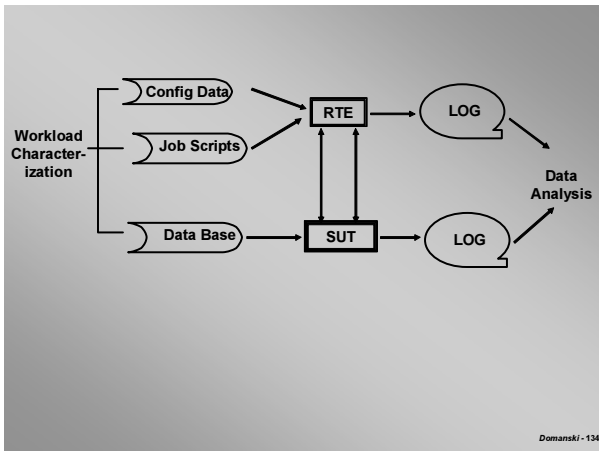


Figure 3.6

In general the workload or representative program(s), etc., must be represented in some way to the RTE so that the system under test (SUT) believes it is the real world presenting the load. This needs to be done by creating *scripts* that contain message text and control information. Figure 3.7 is one example of what would be included in a script for a UNIX® environment for someone checking their email.

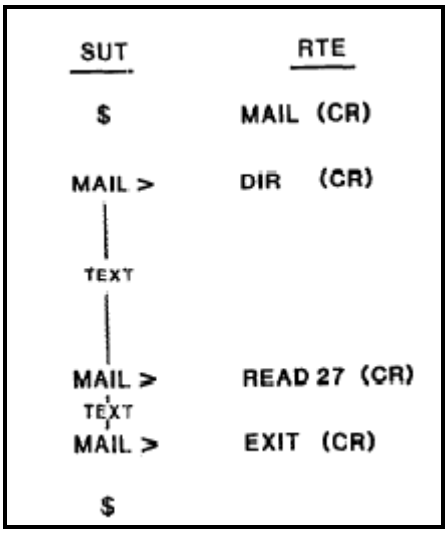


Figure 3.7

In general, the script has the information about what needs to be sent to the system under test and what it should expect. All the interactions with the SUT are time-stamped and put on a log file for later reduction and analysis. If Figure 3.7 represented a *mail* user and we ran an experiment where there were 16 such users, our RTE log file might look like Figure 3.8 if it were sorted by terminal number and time of day.

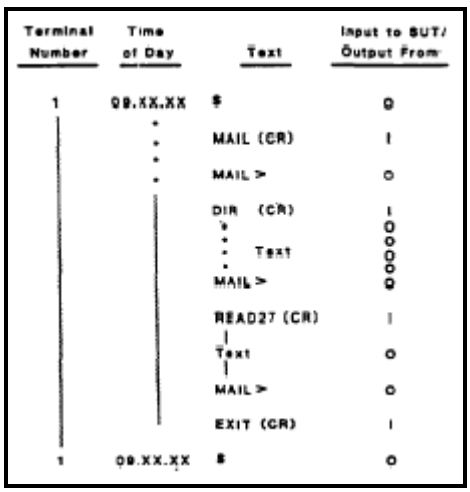


Figure 3.8

Much work has been done using RTEs that dates back nearly 20 years (Domanski[1]) and the reader is referred to past CMG proceedings - search on 'benchmarks' - for an extensive list that spans many different platforms.

[1] Domanski, B., "*Load Driving A System*," Computer Performance, Vol. 3, No. 4, Dec 1982

Another approach to obtaining the load service relationship is via *modeling* the actual system. The next section will present an overview of Modeling.

4. Modeling

A model of a system is a representation of the system that consists of a certain amount of information about the organization of the system, and is built to study it. Since different kinds of questions can be asked about a system, several different types of models can be constructed. They all represent the same system, but look at the system from different purposes or they contain varying amounts of detail.

A model can be viewed as a system (here system is not a abbreviation of a computer system). Various models correspond to partitioning the system into different components, representing the interactions between components and its operation environment. These conceptual models play a fundamental role in evaluating computer system performance. They consist of all the information the evaluator has about the behavior and structure of the actual system. The deeper the knowledge of a system, the better the conceptual model and the easier and more successful the study is likely to be.

Characterizing the behavior of a system is often done using the concepts of state transitions.

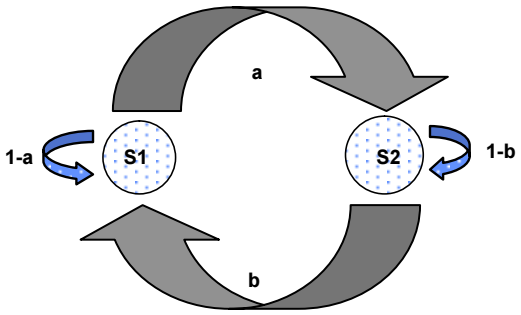


Figure 4.1

The state of the system at time t is defined as the values of the system parameters of interest at time t. Any variation in these values is viewed as a transition to another state, i.e. a state transition. A state transition model associates probabilities with each state transition.

A simulation model reproduces the behavior of the actual system over time. Its behavior in time, under stimuli that represents the actual systems environment, is observed and performance data is collected. In this way, the simulation model is treated just like the actual system. Thus, designing experiments for simulation models is the same as designing experiments for the actual system.

An analytic model consists of a set of mathematical equations that are solved by following the evolution of their solutions over time. Computer systems are seen as discrete systems; that is, they evolve by discontinuous state transitions called events. These models include equations expressing the logical conditions under which these events occur.

Queuing Concepts and Operational Analysis

Queuing models are popular analytic models. As an example to illustrate the concepts underlying queuing models, consider a barber shop.

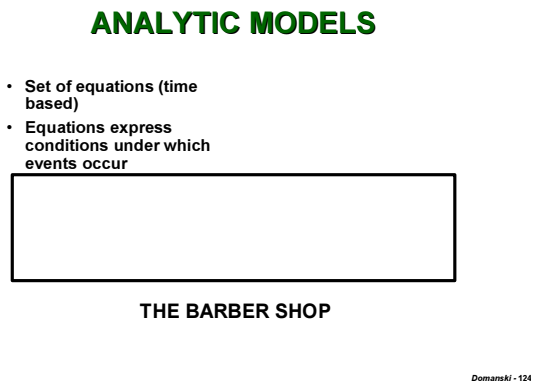


Figure 4.2

Here customers enter the barber shop, and wait until the barber is available to cut their hair. It takes some amount of time to cut their hair, after which they leave the barber shop. This is known as a single server queuing system, and is formally denoted by the following diagrams.

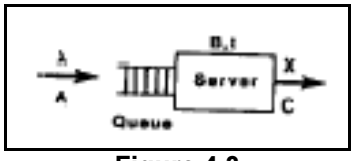


Figure 4.3

Problems analyzing these systems are all derived by making assumptions, which are testable, about the queuing model. This simple approach is called *Operational Analysis*, developed by Buzen & Denning[2].

In both the benchmarking and modeling approaches to obtaining load-service relationship there is a need to characterize and forecast the load. The next section presents an overview of load characterization and forecasting.

5. Load Characterization and Forecasting

The objective in load characterization is generally to represent in some way the workload that is required to obtain a load-service relationship. That is, represent the load in enough detail to used in either a benchmarking or modeling approach to obtaining the load-service relationship.

The load on the system can be represented in some abstract way by collecting some resource or functional oriented data and reducing it to model the load. Here, resource oriented work units would be designed to consume system resources such as CPU, memory, I/O). etc. Functionally oriented work units are those that do some predefined function such as C compiles, text-edits, email, and even database queries and updates. Representing the work units in this way is sometimes referred to as creating *synthetic* workloads. Of course, one could combine the functional and resource oriented approach in an attempt to create synthetic workloads for benchmarking.

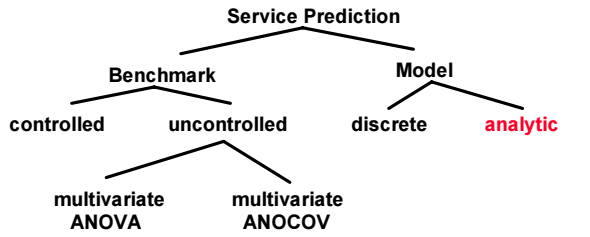


Figure 5.1

[2] The Operational Analysis of Queuing Network Models, Computing Surveys, Vol. 10, No. 3. September 1978.

Several tools can be used to help create the synthetic workload(s) including hardware/software monitors, log files, survey data and even console logs. These tools might interrelate in the following way (see Figure 5.2).

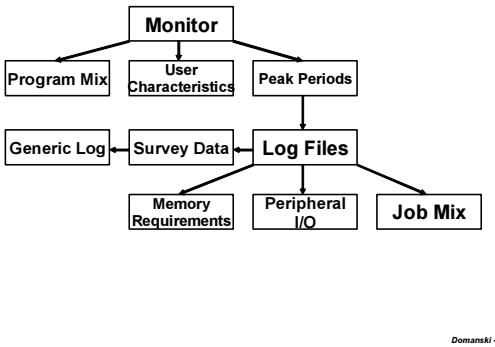


Figure 5.2

The statistical approach known as *clustering* has been used successfully to identify a small set of resources consuming work units. The reader is referred to much of the original defining work by Artis[3] in the field for more information.

6. Performance Management

Introduction

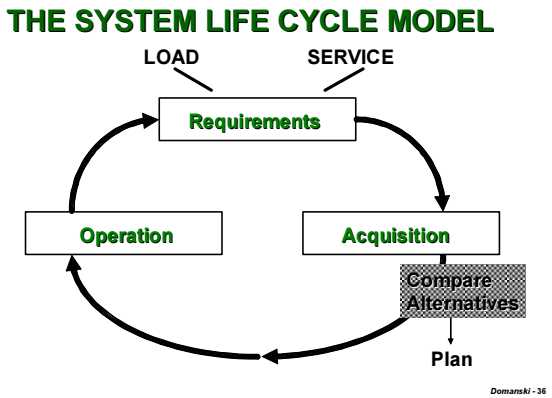


Figure 6.1

The System Life Cycle Model has previously been shown to accurately reflect the different phases of a computer system, from inception through actual modification and change. The requirements phase involves the specification of functions and perhaps

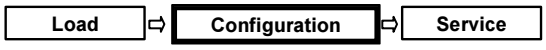
[3] Artis, H. P., "Capacity Planning for MVS Computer Systems" D Ferrari, ed., Performance of Computer Installations. North Holland Publishing, 1978

even the behavior of a proposed computer system. For example, the requirements for a time sharing based application would specify the different types of work that are to be performed, e.g. inquires about a customer, and they might also specify the expected response time for each different type of work. Let a work unit be defined as the accomplishment of some function(s) during some period of time, where a work unit is a task in a batch environment, and a work unit is a transaction in a timesharing environment. Given a list of required work units a system will do, an analyst moves onto the acquisition phase of the model.

Acquisition involves transferring the requirements into a set of hardware and software components to be obtained, which ultimately, will provide the functionality and behavior specified in the requirements.

Once the set of components are derived, an analyst goes thru a somewhat detailed phase of evaluating different alternatives (i.e. configurations) that are available, to derive that set of components that (1) is the most cost effective and (2) satisfies the initial requirements.

As a direct outgrowth of the acquisition phase, the analyst is, in essence, deriving a capacity management plan. This consists of three elements:



1.Load forecasts
2.Anticipated service levels
3.Configuration definitions over time

Domanski - 137

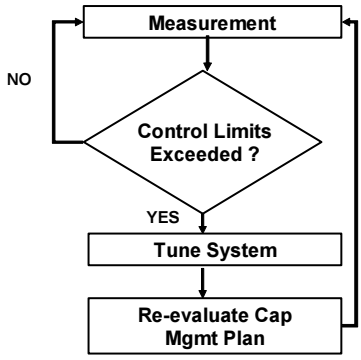
Figure 6.2

These elements address the primitive model of a computer system, namely the expected load on the system over time, the expected level of service the system provides over time, and the different configurations of both hardware and software over time that will handle the expected load.

Comparing alternatives in the acquisition phase of the system life cycle model will provide different load/service relationships for each unique configuration. Using these relationships, the analyst can define *control limits* for the load and service levels.

Thus, if load is increasing (or if service is degrading) at a rate faster than expected, changes to the capacity management plan should be reviewed. For example, a new CPU whose need is expected one year from the initial operation of the system may be required after only six months, because of either higher load than expected or because of a lower level of service than what is required.

By measuring the system regularly, the analyst can compare actual load/service levels with control limits established from the acquisition phase. Thus, a performance management plan is conceived, within the operations phase of the system life cycle model.



Domanski - 138

Figure 6.3

Elements of a Capacity Management Plan

Before measurement of the system, it is assumed that a capacity management plan has been defined. As stated before, it is composed of 3 elements: a load forecast, expected levels of service and a configuration definition over time. "Level" of load and service are defined as control limits. Here, the problem of identifying the data items that are used to compare against control limits is addressed.

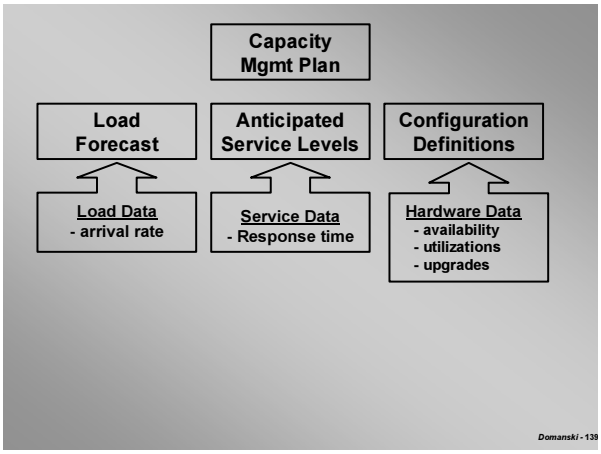


Figure 6.4

Load forecasts are composed of arrival rates for the different work units entering the system. For example, the number of C++ compiles in a given hour, or the number of arrivals of a specific database transaction in a day. Arrival rates for each unique work unit entering the system could later be compared against control limit values for these work units.

Service levels are composed of processing or throughput rates for the different work units acted on by the system. For example, the number of C++ compiles completed in an hour, or the number of completions of a specific transactions in a day. Service level data items may also address the response time of a work unit. For example, control limits for specific database transactions may have a maximum response time of ten seconds, and a minimum response time of three seconds. A natural question is posed here, namely, why worry about not achieving a minimum response time? If this is the case, one may want to delay any planned configuration changes e.g., hardware upgrades.

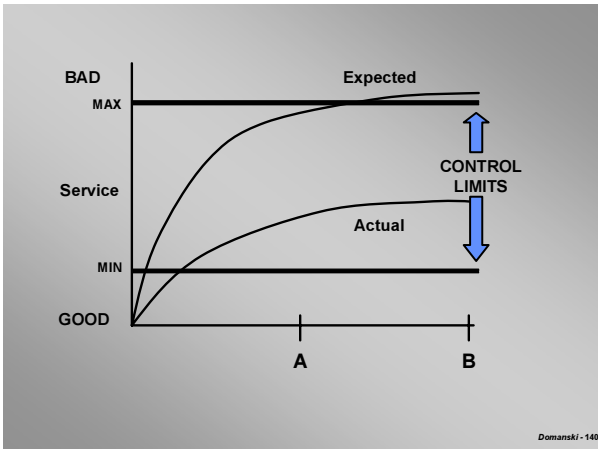


Figure 6.5

By observing that the minimal response time is being achieved later, this may imply that time A, the time at which a hardware upgrade is to be made, is too early. It may be possible to delay the upgrade until time B. All these questions are addressed in greater detail when discussing the re-evaluation of the capacity management plan.

Hardware data is composed of three components: availability information, resource utilizations, and component upgrades/replacements.

Availability information analyzes the time between component failures, and the time required to repair the failed components. Though intuitively obvious, it is worth stating that during a component repair period, little, if any use of the system will be made. Thus, it is important in defining the times at which hardware upgrades will be made, that mean time to repair and mean time between failures be carefully considered. Vendors usually supply this information by component.

Resource utilizations are measures of how much (little) hardware components are being used. For instance, CPU 70% busy, disk drive 1 is 20% busy are examples of resource utilizations. The resources of a computer system fall into three classes: CPU, memory and I/O. These utilizations are traditionally used to spot "bottlenecks" in system throughput, resulting in a degradation of service. Control limits for resource utilizations are usually established by "*rules-of-thumb*"; that is, intelligent guesses are made based on previous observations of similar systems using these components.

Data used in a capacity management plan can be partitioned into two classes: low versus high levels of resolution. Low resolution data items correspond to viewing load, configuration and service with respect to the entire system. For example, low resolution load data would be number of tasks arriving to the configuration in a certain time, perhaps being further classified by type of work unit. Examples of low resolution data items are shown in Figure 6.6.

Low Resolution Data		
Load	**Configuration**	**Service**
# of C++ compiles	**System MTTR, MTBF**	**4000 transactions / day**
# of file edits	**70% CPU busy**	**6 second average response time**

Figure 6.6

Given appropriate measurement tools and reports, the performance management process is summarized in Figure 6.7. Measurement tools like software monitors, hardware monitors, accounting files and system "*snapshots*" provide the necessary measurements. Graphic as well as tabular reports are necessary to analyze the status of a system. Non-traditional data analysis techniques have been successful in problem determination such as Boolean data analysis, Neural Networks, Rule-based Expert systems, and Data Mining.

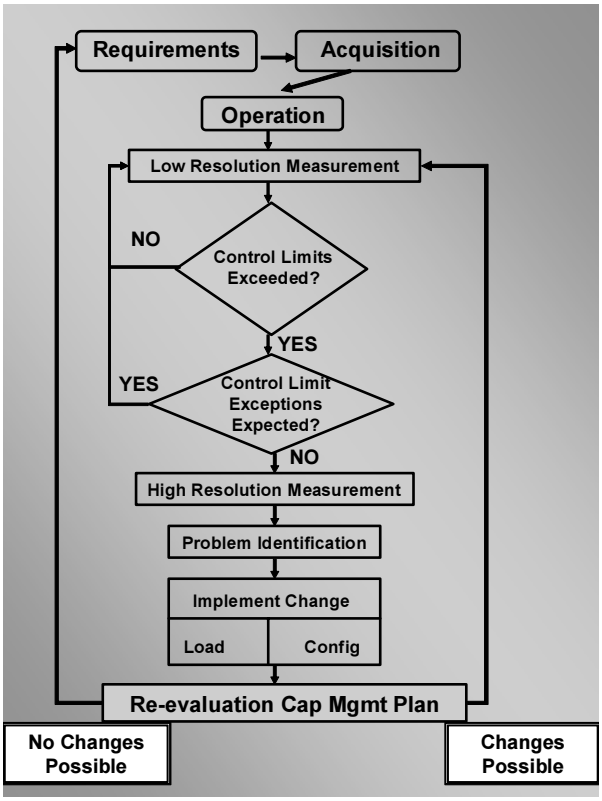


Figure 6.7

7. Summary

Our goal for this tutorial was to introduce the reader to a variety of techniques used in the capacity planning / performance evaluation process. Where possible, references were given to direct the reader to more detailed sources of information. We cannot stress enough the importance of past CMG[4] proceedings as the preeminent source of performance and capacity information.

8. Bibliography

It has been our experience that at the conclusion of the presentation, the audience requests additional information about many of the topics presented. Rather than providing a detailed bibliography, we have attached a list of papers that we believe would be beneficial to the interested reader. These papers are being presented at CMG 2001 and their session numbers are included as well.

Session #	Title of Session	Author
3202	Thinking Outside the Cubicle	Glenn R. Anderson
3204	Introduction to Wireless Networking	Laura J. Knapp
3209	Tutorial on Performance Management of Oracle Using Oracle Enterprise Manager	Herschel DeCouto
3303	Trending for Capacity Planning: A Tutorial	Ray Wicks
3304	System Performance Management and Capacity Planning	Bernard Domanski
3505	Introduction to Software Performance Evaluation	Connie U. Smith
4106	Is IT Chargeback Still Relevant? Was it Ever? A Tutorial Answer	Sidney Finehirsh
5103	Software Capacity Planning 2001	Ivan L. Gelb
5107	TCP/IP Basics for the IS Professional	Michael S. Recant
5203	Storage Networking Primer for OS390 and Open Systems	Greg P. Schulz
6103	Performance Miracles via Disk IO Activity Tuning	Ivan L. Gelb
6104	Understanding IPv6	Laura J. Knapp
6105	Workload Characterization for OS/390-z/OS Capacity Planning	Gregory V. Caliri

[4] Computer Measurement Group, http://cmg.org, P.O. Box 1124, 151 Fries Mill Rd., Suite 104, Turnersville, NJ 08012 , 800-4-FOR-CMG, cmghq@cmg.org