Don't Plan Capacity When You Should Predict Applications

Tim R. Norton
Doctoral Candidate
Colorado Technical University
CMG97 Session 443, December 10, 1997

As application designs incorporate client/server systems, either as new functions or as front-ends for legacy systems, the capacity planning challenges increase at a staggering rate. Changing from a single system to clients and servers on different computer platforms would be hard enough, without the additional complexities of different operating systems, databases and networks. The capacity planner needs to take an enterprise view of the application and predict the performance from the user's point-of-view. Planning the capacity of client/server applications requires an understanding of each of the systems and the inter-relationships between them.

*The Simalytic™ (**Sim**ulation/An**alytic**) Modeling Technique[1] provides a modeling framework to connect the parts of an application together, using the modeling tools already in place for the individual systems. This technique uses a general purpose simulation tool as an underlying framework and the results of analytic platform-centric modeling tools to represent individual nodes in an enterprise model of the application. How to construct a Simalytic Model is discussed, including an example using an inexpensive general-purpose simulation tool, a simple queuing theory tool and Visual Basic to implement the Simalytic Function that provides the bridge between them.*

1. Introduction

Today's computing environment is changing at a very rapid pace. New applications that once would have been implemented on a single computer are now multi-platform. What were once batch applications are now on-line transaction processing client/server systems with GUI (graphical user interface) front-ends on PWS's (programmable work-stations) attached to departmental servers and mainframe repositories. Complex application designs utilize the features and services of different types of computers (mainframe, mid-range, desktop) running different operating systems (MVS, Unix, OS/2, Windows, etc.) connected by a variety of communication network techniques (RPC, DCE, NFS, FTP, SNA, APPN, etc.) (Hatheson 1995; Wilson 1994).

As applications move into this new client/server world, how do we select the right systems at each level and, once selected, how do we insure those systems are the right size? If any of them are too small the whole application will fail. If any are too big, the cost of running the application may exceed the revenue it generates. Neither is a very attractive situation.

Capacity planning has traditionally been focused on determining if a given system has "enough capacity" to service an application. Today, planning the capacity of large computer installations with multiple systems requires an understanding of not only the operating systems, the platforms, the clients, the servers, the networks, the transaction systems, etc., but also the relationships between them. Once those relationships are defined and understood, the application's performance can be assessed against the business objectives and goals. Projected business volumes are then modeled to predict the capacity required to meet those goals at future volumes. Instead of planning the capacity of individual systems, the responsiveness of the application needs to be predicted across the entire enterprise. Only then can the true capacity requirements be identified.

There are many modeling tools and techniques that address both performance and capacity for each of the systems in today's multi-platform environment (Pooley 1995; Smith 1995). The Simalytic™ (**Sim**ulation/An**alytic**) Modeling Technique provides a bridge across these existing tools to allow the construction of an enterprise level application model that takes advantage of models and tools already in place for planning the capacity of each system. The advantages of using the Simalytic Model Technique are:

- Rapid Analysis
- Spiral Methodology
- Reuse
- Distributed Model Development
- Applicable Tools

Rapid Analysis: A Simalytic application model can be quickly constructed with minimal effort for each node model. Analysis of the application using this high

[1] *Simalytic™, Simalytic Modeling™, Simalytic Modeling Technique™, Simalytic Enterprise Modeling™* and *Simalytic Function™* are trademarked by Tim R. Norton. All other trademarked names and terms are the property of their respective owners.

level model allows additional effort to be applied only to the nodes in the critical-path areas, saving time and effort by avoiding areas with minimal impact on application responsiveness.

Spiral Methodology: Simalytic Modeling provides a mechanism to promote the exchange of information between the users, the developers and the modelers. As more information becomes available, the application model is refined. Early assumptions can be replaced with the results from more sophisticated tools as more details become available. This spiral approach allows modeling earlier in the design phase as well as after implementation.

Reuse: Simalytic Modeling provides for the direct incorporation of existing tools and techniques into the high level model. The investment in time, effort and money expended for training on the tools used for existing system models is preserved. It is much easier to get someone who has built a number of models of a system, using a tool they are well trained on, to do another model, than it is to start over in different tool.

Distributed Model Development: Simalytic Modeling provides easy distribution of modeling activities to multiple people or organizations. Because of the reuse of existing modeling techniques, node models can easily be "sub-contracted" to the modelers most familiar with those tools and systems.

Applicable Tools: The ability to use the most applicable tool for each node of the Simalytic Model increases both the speed (construction and execution) and the accuracy of the application model. For example, the network component of the application can be assumed constant or modeled with greater detail if the modeler determines that the network is a major component of response time. If the simulation tool does not provide the level of complexity needed, then a specialized network modeling tool can be used to create network nodes for the whole Simalytic Model or just for a critical part of the network between two servers.

1.1 Major Topics Covered

- Section 2, *Simalytic Modeling Review*, is a general review of Simalytic Modeling and the related background topics.
- Section 3, *Steps to Build a Simalytic Model*, is a step-by-step description of how to implement a Simalytic Model for transaction based applications.
- Section 4, *Simalytic Model Implementation*, discusses the actual implementation of the relevant steps from Section 3 using some inexpensive and readily available tools.
- Section 5, *Conclusion*, discusses how Simalytic Modeling applies to actual modeling situations and the benefits of using the technique.

2. Simalytic Modeling Review

What is Simalytic Modeling? Simalytic™ Modeling (from **Sim**ulation and An**alytic**) is a hybrid modeling technique that uses a general purpose simulation modeling tool as a underlying framework and the results of an analytic modeling tool to represent the individual nodes or systems. The problem addressed by Simalytic Modeling is at the intersection of several areas: capacity planning, modeling (both simulation and queuing theory), client/server transaction processing systems, and commercial tools (both general purpose and platform-centric). The goal of a Simalytic Model is to predict the capacity requirements of an application executing on heterogeneous computer systems by creating an enterprise level application model.

This section provides a general overview and review of Simalytic Modeling. Additional detailed information about Simalytic Modeling is available in other papers published and presented by the author. A detailed description of the technique and a discussion of the industrial and environmental changes that provided the impetus for Simalytic Modeling are available in (Norton 1996). Details of the mathematical foundation and validation of the technique are available in (Norton 1997a) and (Norton 1997b).

Before discussing Simalytic Modeling, it is important to quickly review the background topics that support Simalytic Modeling. It is assumed that the reader is somewhat familiar with each of these topics and the review will provide for a common basis in the discussion of the implementation. Additional information on the following topics is available in the works mentioned above and in the references cited in those works:

- General capacity planning.
- Transaction based applications.
- Client/Server environments.
- Modeling capacity projections.
- Modeling tools.
- Simalytic Modeling methodology.

2.1 Capacity Planning

The capacity of a system can be measured many different ways, depending on the business the system supports. Generally, the way a system is measured centers around the performance of one or more of the applications it supports. The system "has enough capacity" if everything is getting done when it is needed. Capacity planning is making decisions about the resource requirements of a given computer system based on the forecasting of future application performance using the goals and expectations of the business. What do we have to buy and when do we have to buy it to make sure that the applications that

run the business perform at the level required to insure that the business succeeds?

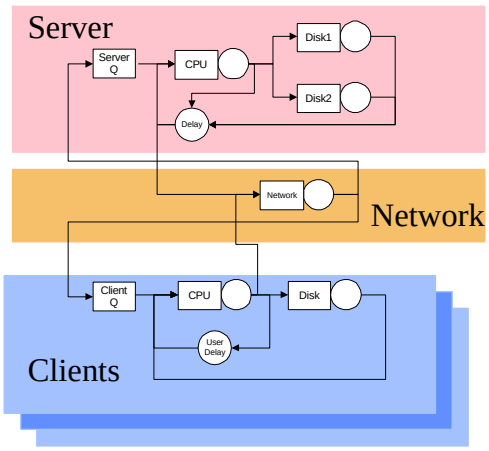
2.2 Transaction Based Applications

Although there are still many important batch applications, this discussion will center around transaction based applications. Transaction processing systems, often referred to as OLTP (On-Line Transaction Processing), allow the end-user to enter a relatively small independent unit of work into the system and receive some information as a response in near real-time. Transactions include entering an order at a terminal (business transaction), an SQL command (database transaction), some keystrokes followed by a carriage-return (interactive transaction)—whatever is meaningful from the end-user's point-of-view. Transactions can be counted to establish load (e.g. arrival rate) and measured to establish performance (e.g. response time). The responsiveness of the transactions associated with an application determine if that application meets the needs of the business. Projected business volumes are then modeled to predict the capacity required to meet the business goals at those volumes.

2.3 Client/Server Environments

When modeling application performance, which techniques and products are chosen to implement the application are of concern only to the extent that some provide better modeling data than others (i.e. SMF, RMF, CMF, etc.). The interrelationships between the systems, however, must be fully understood. It is important to understand that any of the client applications on any of the PWS's can, and eventually will, send transactions to several of the legacy applications to provide the end-user a screen of complete and interrelated information.

Modeling in this environment is a challenge because each of the systems requires a different knowledge base and expertise (Gunther 1995; Hatheson 1995). None of the systems can be modeled independently because the transaction arrival rate for one system may be dependent on the response times of the others. *Figure 1 An Enterprise Model* shows a very simplistic model for each of the major areas of a client/server application and, although it only shows a single server, the interdependence is evident. The responsiveness of one part of the model (server, client or network) will have an impact on the other two. Each of the models in *Figure 1* can be well defined, but when taken as a whole, the complexity rises quickly as additional servers and clients are included in the model. Each server is often part of a design focused on one application that seldom coordinates platform selection with other applications. The use of different data collection utilities and different modeling tools on each of the different platforms greatly increases the complexity of modeling the application.



Reproduced with permission from Dr. Connie Smith, Performance Engineering Services.

Figure 1 An Enterprise Model

2.4 Modeling Capacity Projections

The use of models to assess and predict the performance of computer systems is not new. Capacity planning has always relied to some extent on modeling because of the need to predict future requirements. A capacity planner can analyze the workloads and make predictions based on experience or simple trending. Models can also be constructed to understand how an application functions without any intention to predict future performance. The area of interest here is the intersection of the two fields; models used to predict capacity requirements based on performance expectations.

2.4.1 Response Time Modeling

The key to the capacity planning methodology discussed so far is the ability to predict the performance of a future workload, given a desired system configuration. As applications move towards being transaction based, the definition of application performance becomes centered around transaction response time. Modeling the response time then becomes crucial to the ability to predict the future performance of that application.

There are two basic modeling techniques used for computer performance modeling: simulation and analytic queuing theory (Kobayashi 1981; Menascé, Almeida, and Dowdy 1994). Either of these techniques will build a model that represents the major components of the computer system to be modeled. A third technique, hybrid modeling, is the combination of both simulation and analytic techniques in a single model (Kobayashi 1981).

2.5 Modeling Tools

In addition to the choice between analytic and simulation tools, the capacity planner or performance analyst has the choice between platform-centric and general purpose tools. The basic difference between

these two groups is which problem set the tools were designed to address.

2.5.1 Platform-Centric

Platform-centric means the tool contains detailed information about the platform, but does not allow more than one platform to be modeled at a time. For example, they would include information about the number and type of processors for each system in the model (e.g. an IBM 982 running MVS 5.2 with EMC Symmetrix 5500 disk subsystems). Platform-centric models are generally easier to build because they are made of "building blocks" already defined to the tools, and the relationships between them are fully understood by the model. However, these tools cannot be used to model an environment not built into the tool. Although many platform-centric tools allow the user to define new servers with new performance characteristics, they generally do not provide large libraries of device and system definitions dramatically different from the supported platform. Platform-centric tools are *generally* implemented using analytic, or queuing theory, modeling techniques and process performance and configuration data collected from existing running systems.

2.5.2 General Purpose

General purpose means the tool contains the features to allow the user to model almost anything, but with little or no "built-in" understanding of any given computer platform. These tools are used to model more than just the hardware, including application design, traffic flow and communications protocols. System components are modeled using either a submodel to implement the underlying architecture or a pre-determined delay value. Although many general purpose tools provide libraries of sub-models for a variety of systems and devices, they generally do not provide the required level of granularity, being either too general or too detailed for the situation. Building the relationships between the submodels is part of the overall model construction and may require an in-depth understanding of all of the submodels used, some of which are provided by the tool vendor in executable–only formats. General purpose tools are *generally* implemented using simulation modeling techniques.

2.6 Simalytic Modeling Methodology

There are two key differences between the existing modeling tools and the Simalytic Modeling methodology. The first is the ability to use the results from not only a different tool, but a different modeling technique altogether, as a submodel within an enterprise model. The second is the ability to use the results from tools or techniques already being used to model individual nodes in the system. These differences reduce the time and effort to build an enterprise level model of an application by using the results from commercially available platform-centric tools or existing detailed application models.

Simalytic Modeling brings together existing performance models (usually platform-centric analytic models) and application information (best expressed as simulations). The queuing theory models rely on averages, such as average response time, average service time and average arrival rate. These models are generally more efficient to execute than simulation models, but, because of the use of averages, their accuracy generally decreases as the data collection interval increases due to variability in the data. Simalytic Modeling allows the application to be modeled over longer periods of time to understand the application dynamics without increasing the error due to greater variation in the data items used for the above averages.

When using commercial queuing theory tools, it is generally understood that shorter intervals (the time period for which measurement data was collected to use in building a model) usually produce better model results because there is less variation in the measurement data. Trace-driven models are the most common in capacity planning and performance modeling because of the focus on existing systems. As an additional benefit, the trace data provides the transaction arrival distributions, which is often a major issue in model construction.

Simalytic Modeling is based on a hybrid technique that allows the models to use the best features of each tool. Submodels allow some part of the model to be replaced with a different model, using a different technique, as long as it provides appropriate functionality and results; similar to the FESC (flow-equivalent service center) decomposition technique discussed in (Menascé, Almeida, and Dowdy 1994). A valid model (proven to produce accurate predictions) must exist for each system or node to be included in the application enterprise model. The application details must be understood, and consistently defined, at the enterprise level.

2.6.1 Methodology Process

An enterprise level model is constructed by starting with a very high level simulation model of the application, where each system is a single server. Then, instead of using a pre-defined service time, each server uses a transform function, the Simalytic Function, that maps the transaction arrival rates to service times. As the simulation model is run, the service time dynamically adjusts at each node depending on the combination of transaction arrival rate for the application and the other work at the node.

3. Steps to Build a Simalytic Model

As with any modeling effort, creating a Simalytic Model requires more than just putting the pieces together in some modeling tool. A substantial amount of information is required about the applications and

systems involved. Although this section presents all of the steps to build a Simalytic Model, a full discussion of some of them is beyond the scope of this paper. The most critical step, Workload Analysis, is a very complex and involved process, and only some of the issues involved, those that relate directly to the construction of a Simalytic Model, are discussed here. Other areas, such as calibration techniques for queuing theory tools and features of simulations tools are assumed to be covered in the training and documentation for the specific tools

The major phases to creating a Simalytic Model are:

1. Workload Analysis.
2. Node Models.
3. Simulation Model.
4. Simalytic Model.
5. Model Analysis.

Each of these phases is discussed in detail in the following sections. This list is not meant to be comprehensive for all of the phases. Once the Simalytic Model has been created, it must be used productively. Generating the speculations, planning the scenarios and analyzing the results in terms of application impact are all activities the reader should be very comfortable with within the context of the modeling tools already being used. Simalytic Modeling requires the same level of analysis and presentation once the model has been completed and calibrated.

3.1 Workload Analysis Phase

In the Workload Analysis Phase the modeler collects information about the application to be modeled. This includes identifying, defining, documenting and measuring the application. This phase includes the same type workload analysis done for system level modeling efforts, but it must be done for all of the systems supporting the application. It also includes collecting additional information about the application from the enterprise point-of-view.

Identify: Identify the workload. Identifying the workload to be modeled is often the most difficult step of any modeling activity. Because Simalytic Modeling takes an enterprise view of the application, the identification process is even more difficult. Not only does the application need to be identified for each system, but it must also be identified at a global level. How a workload is identified will differ between platforms and depending on what database and middle-ware was used. A CICS transaction using DB2 provides different data collection than a Tuxedo transaction using Informix. A totally in-house developed application may provide better or worse data collection, but most certainly different. Although workload identification is done on each platform, it cannot be done independently. How the workloads are correlated across the platforms must be considered during identification.

Start by identifying what the end-users think of as the business transaction. This is no longer a single CICS screen, but it is often a series of prompts and replies that, taken together, make an single activity such as entering an order. Regardless of the modeling technique used, workload analysis is very complex and requires substantial effort. The point of this step is to understand the objective, which is to define business activities, such as orders entered, in terms of measurable work elements, such as transactions A, B and C. The workload projections and response time measurements are at the business level. The models are built at the IT (Information Technology) transaction level. There must be a valid mapping between these two levels.

This step must be done in conjunction with both the application developers and the end-users. It is a series of trade-offs between what the end-users would like to use and what is realistic, considering how the application works. For example, if new orders and inquiries of existing orders are done from the same screen using the same transaction, it may not be possible to separate them into different workloads.

Document: Document the application topology. What transactions are routed where under what conditions? Are the IT transactions (i.e. CICS or Tuxedo) serialized or are some executed in parallel? Is the client/server architecture 2-tier, 3-tier, a combination or something altogether different? The documentation technique used should be whatever best supports the application and has the support of the users and developers, who must maintain the documentation.

The objective of this step is to produce a topology description of the application that can be easily and accurately translated into a simulation model. When this step is completed, the modeler should be able to track a business transaction from the originating workstation through the entire environment (including all splits, protocol translations, routing decisions, etc.) back to the same workstation.

Measure: Measure the workload. One of the key enablers for Simalytic Modeling is the ability to measure the application from both the business point-of-view as well as at the system level. The overall Simalytic Model can be calibrated only if the responsiveness of the business transactions can be measured. The node level model can be calibrated only if the IT transactions can be measured. The measurement of the IT transactions is generally already implemented for the node level models currently being done. However, in today's client/server environments, the measurement of the business transaction is very difficult.

In this step, the modeler must determine the ability to measure the application and workloads at each system and from the end-user point-of-view (sometimes referred to as end-to-end response time). It is assumed here that the node level measurement data are readily available. At the very least, the modeler must have additional information about the number, frequency and response times of the business transactions. If the application or the system does not collect the data, the modeler may need to observe the application users and collect the data manually. Although far from providing the quality of data most modelers have come to expect from today's systems, manually collected data will allow the modeler to produce enough results to hopefully encourage the application designers to generate the required data on an on-going basis.

The objective of this step is to determine the feasibility of the modeling effort. If adequate measurement data cannot be collected then the modeler must determine if sufficient interest will be generated by the effort to increase the quality of the measurement data. Another possible approach would be to use the Simalytic Model to establish reasonability bounds for each node based on overall application performance. However, how well such results will be accepted will vary by organization and company.

Correlate: Correlate the workload across systems. The final step of Workload Analysis is to determine the correlation between the workloads at each system. Does the definition for a workload at one system really mean the same thing as that workload at another system? There cannot be any additional or missing transactions. For example, if workload W1 is defined as three transactions, A, B and C, then the measurement of W1 at system S1 must include all of the transactions A, B, and C that are routed to S1. In addition, it cannot include any other transactions that run on S1 not included in the overall definition of W1. This appears to be a straight-forward requirement, but it becomes very complex as the client/server environment grows and applications attempt to reuse functions. An existing legacy application transaction might be invoked by more than one client/server business transaction to look up customer information. This transaction would then need to be included in multiple workloads, which would then cause error in the model. This situation cannot be resolved without some additional application modification to enable collecting data about which workload invokes each transaction.

The objective of this step is to insure the consistency of the workloads across the entire enterprise model. Problems, such as the one mentioned above, need to be identified, documented and resolved with data collection, application changes, model changes or simplifying assumptions.

3.2 Node Models Phase

In the Node Models Phase, the modeler models all of the systems supporting the application. This phase includes the same type modeling done for system level modeling efforts, but coordinates the node level models to integrate with the additional information about the application from the enterprise point-of-view.

Build: Build a model of each node. Building a model of each node used by the application is not significantly different from any existing system level modeling efforts. Whatever tool is currently used to model each system should be used for that node in the overall model. The major difference is that the workload definitions used in the node models are those developed in the prior Workload Analysis phase. From a realistic stand-point, only the application of interest should be modeled as an identifiable workload with response time predictions. All other activity at each node should be included only to understand resource usage and correctly influence the workload of interest.

The objective of this step is to build a model of each system, but to also take advantage of any existing modeling efforts. Although the workload definition may change, the processes already in place to collect measurement data and calibrate the models, and possibly some of the actual models, for any of the nodes can be effectively reused.

Calibrate: Calibrate the model of each node. The node level models must be valid (calibrated and verified) before continuing. This can be time consuming for a complex application if there are a large number of nodes involved. There must be a high degree of confidence in the predictive nature of each of the node models. Because the Simalytic Model will connect the nodes together using the workload definitions, an error or poor results from any one node model can impact the accuracy of the Simalytic Model for the entire application.

The calibration techniques used are dependent on the modeling tools and are much too complex to discuss here. In addition, care must be taken to insure that steps taken to calibrate one node do not contradict assumptions made in a different node model.

The objective of this step is to have a solid predictive model for each node that presents a consistent view of the application across all nodes.

Run: Run the models. Develop a profile of the application by running each of the node models for a series of arrival rates from very low (i.e. .01 or .001) to very high (either the model saturates or the 'knee' of the response time curve is well established). The actual arrival rates used will depend on the application and should make sense to the actual users of the application. The arrival rate increment should be

as fine as is practical, considering the time and resources required for each execution of the model. If the arrival rate range is .01 to .09 then the increment should be something like .005 or .001. If the arrival rate range is .1 to 10.5, then the increment should be larger, like .05 or .1. The increment does not have to be uniform across the range; use a larger increment when there is little change in the response times between arrival rates and use a smaller increment when there is a large change.

The objective of this step is to establish a response time curve that can be used to extrapolate the response time when presented an arrival rate not modeled.

Create: Create a model results table. Create a table of response times and arrival rates for each system for the workload of interest. Other workloads on the system will not be modeled in the Simalytic Model but they are still accounted for in the node level system models. A key assumption here is that the other workloads provide a consistent load on the system and thus a consistent level of interference to the application workload being modeled. If this is not true, then the table must be extended to include some external parameters, such as time of day. The response time values must then be based on the combination of those parameters and the arrival rate.

The objective of this step is to characterize the application performance and responsiveness. The information in this table will be used to create the Simalytic Function when the Simalytic Model is constructed.

3.3 Simulation Model Phase

In the Simulation Model Phase, the modeler builds an overall model of the application with each of the systems supporting it represented as a node or server. This phase uses the information from the Workload Analysis Phase to connect each of the systems together to provide the enterprise view of the application.

Build: Build an overall model. Using the simulation tool of choice, build an overall model of the application with a single server for each node in the system. This model is defined by the application topology documented in the Workload Analysis stage. It identifies what transactions are routed to which nodes under what circumstances. This overall model of the application can be built before any of the node level performance data has been collected. Make assumptions as to the expected or desired performance at each node and use the model to identify how sensitive the application is to changes in the performance of any given node. If large variations in service time at a node have only minimal response time impacts, then that node level model may be deferred.

The objective of this step is to build a model of the application that represents the overall application behavior across the enterprise.

Set: Set the overall model parameters. Set the service time of each node to the lowest response time in the table created in the Node Models Phase. Set each node to have enough servers so that there is ***no*** queuing at any node. How this is done will differ with each of the simulation tools. Generally, it is some type of replication factor within the node. The value must be very high so that there is never any queuing to get a transaction through the node. It is generally not a good idea to create multiple nodes because of the problems that creates with transaction routing. In the enterprise model, the service time and the response time for each server will be the same because the queue time is accounted for in the response time data for the server. (The service time of each node in the simulation model is the response time from the analytic model of that node, which is a combination of queue time and service time.)

The objective of this step is to set the simulation model such that the response time at any node can be controlled by the Simalytic Function when it replaces the static service time in a later phase. In addition, the simulation model at this stage can be used to verify the application topology and conduct sensitivity analyses of user expectations.

Calibrate: Calibrate the overall model. Calibrate the simulation model against the end-user response time for the very low arrival rate and verify that there is no queue time at any of the nodes. Because the response time from the queuing theory tool includes the queue time in the node, any queue time in the simulation model will, in effect, double count the queue time. The simulation tool is being used to control the flow and routing of transactions, not calculate the queue time. This step should insure that the topology and routing information is correct before too much effort is spent developing the Simalytic Model.

The objective of this step is to verify that the simulation model accurately reflects both the topology of the application and the response time seen by the users at very low arrival rates.

3.4 Simalytic Model Phase

In the Simalytic Model Phase, the modeler incorporates the results of the system models into the overall model of the application. This phase uses the information from the Workload Analysis Phase and the Node Models Phase to provide the predictive capabilities to the enterprise view of the application.

Create: Create the Simalytic Function. Using the table of response times and arrival rates created from the node models, create a Simalytic Function for each node. This can either be a look-up table or a formula

derived from the curve established by fitting a line to the response time data. The details of the function and how it is implemented will depend on the simulation modeling tool used for the overall model framework.

The initial function is implemented by converting the interarrival time between each pair of transactions to an arrival rate and thereby to the associated response time (Norton 1996). It is most likely that the initial function will not provide accurate enough results due to issues such as arrival distributions. This is because of the difference between the individual transaction nature of the simulation model and the averaging effects of the analytic node models.

The function will need to be enhanced to include additional information about the state of the node for each transaction. The more complex function is referred to as the Simalytic Function because it not only includes the results of the node models, but also the additional features to select the most appropriate result for each transaction visit. To do this, the arrival rate used will more than likely need to be modified by some technique. One approach is to maintain a rolling average over some number of transactions. The number needs to be small enough to maintain the responsiveness of the model to workload changes but large enough to minimize the influence of isolated instances of very small interarrival times. Another approach would be to examine the node to determine the current number of transactions being serviced or the node utilization, then select an arrival rate more consistent with that node state. The technique chosen is a trade-off between rapid development and model accuracy. Any combination of techniques can be used. Implement the simplest Simalytic Function possible and enhance it as required, and only when necessary, to achieve the desired level of accuracy.

The objective of this step is to create a function for each node that accurately reflects the application's behavior. Each Simalytic Function must return a value for the service time of the node for each visit of a transaction based on transaction interarrival time and other node state information.

Replace: Replace the static service times. Replace the service time for each node with the function created in the prior step. Again, how this is done will differ with each simulation tool. For example, some simulation tools support load dependent service times and the response time values can be entered into the load dependent service time table. However, this technique may not be viable if a more complex function is required and the load dependent server cannot implement a Simalytic Function.

The objective of this step is to implement the Simalytic Function in each node of the overall simulation model. The service time used for each transaction visit is the value returned by the Simalytic Function.

Calibrate: Calibrate the Simalytic Model. First calibrate it against the prior simulation model for the very low arrival rate to insure the overall model structure is still correct. Next, calibrate it against known end-user response times for known arrival rates. Enhance the Simalytic Function as required to get the necessary level of accuracy. The objectives of the modeling effort will determine what is an acceptable level of accuracy.

The objective of this step is to insure that the Simalytic Model provides valid application prediction within the required level of accuracy.

3.5 Model Analysis

The next phase uses the Simalytic Model to analyze the application. At this point, the Simalytic Model can be used just as any other type of model which has been calibrated. In addition, how a model is used to answer "what-if" questions is very dependent on the questions themselves. Therefore, the details of the phase will not be discussed here other than to note that all of the phases of constructing a Simalytic Model should be considered as a spiral development process. The completion of each phase may identify additional information or requirements for one of the prior phases. This provides the added benefit of allowing the modeler to implement a quick, simple Simalytic Model and then continue to refine it based on the business requirements and objectives.

4. Simalytic Model Implementation

An actual Simalytic Modeling effort is very complex and involves creating and calibrating multiple models. In order to focus on the model building process, a simplistic example is used to illustrate the steps presented in Section 3.

4.1 Implementation Example

This implementation of a Simalytic Model uses a hypothetical client/server environment to illustrate the process. Assume the workload of interest is an Order Entry application on one server, and a Shipping application on another server also used by the Order Entry application. The Order Entry user types in the name of an existing customer and gets not only their address, but information about any orders. This may provide better service, but it also causes transactions to be sent to another system. Defining the topology of the application identifies that some number of the Order Entry transactions are routed to the Shipping server. The measurement data provides the number and distribution of transactions. If the Shipping workload outgrows the system, it can impact the responsiveness of the Order Entry transactions. In addition, growth in the Order Entry workload will now impact the Shipping system, but only if the orders are from existing cus-

tomers. The systems cannot be modeled independently because the service time for one system is dependent on the response time of the other. When the Order Entry transaction rate increases, more transactions are sent to the Shipping server. The increased response time at Shipping will cause the overall response time for those transactions to increase, which will be seen as either longer average response time or reduced through-put for the application.

Figure 2 A Simple Enterprise Model shows a diagram of this system. The response time is measured from Arrivals to Departures, either through the Shipping node or around it. This example shows how the Simalytic Model connects what is happening in the application on the different servers. If the Order Entry system is modeled by itself, the workload representing the long transactions (those also sent to Shipping) would not reflect the increased response time due to the load at Shipping. Because of the additional application information in the Simalytic Model, it can adjust the response time in the Shipping server based on the current load, which will then be reflected in the Order Entry transactions that visit the Shipping server.

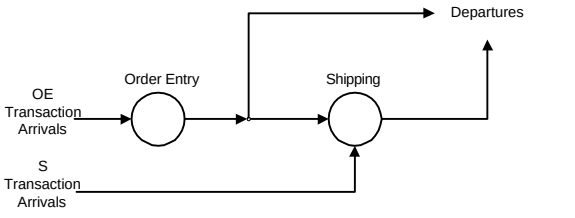


Figure 2 A Simple Enterprise Model

As with any modeling effort, there must be business objectives to analyze using the model. Assume that the manager of the Order Entry department has requested a model to determine when the Order Entry system will need to be upgraded in order to maintain the required response time of less than 1.7 seconds. The arrival rate is assumed to have a constant increase over the next 18 months (the scope of the analysis) and the percent of the Order Entry transactions that also query the Shipping system is assumed to be 30%. The response time goal for the Shipping system is less than 10 seconds (because these transactions generally do not involve a waiting customer). The objectives of the analysis are to answer two questions: “When does the Order Entry system fail to meet the business response time goal?” and “What must be upgraded to again meet the goal?”

4.2 Implementation Process

Although the implementation process for the example follows the steps presented in Section 3, many assumptions have been made about the information collection process to simplify the example. The actual implementation uses some inexpensive and readily available tools. The framework is implemented using Simul8 (Visual), a general purpose simulation modeling tool developed primarily to model manufacturing situations. The performance characteristics of the nodes are developed using the results of the OpenQN analytic tool (Menascé, Almeida, and Dowdy 1994). OpenQN is a simple Pascal program that reads an input file of workload parameters and produces a report. OpenQN was selected because it is easy to use, fast and included with Dr. Menascé’s book (Menascé, Almeida, and Dowdy 1994). Finally, Microsoft‘s Visual Basic provides a rich programming environment and an interface to Simul8 to implement the Simalytic Function™. This interface was one of the main reasons Simul8 was selected for the research in Simalytic Modeling. Either Visual Basic programs or Microsoft Excel spreadsheets can be used for either the service time or for the transaction distributions at each server. In addition, either the programs or the spreadsheets can make subroutine calls to the model to get current state information or to control the model itself. Although not as sophisticated as many of the client/server tools available, Simul8 provides a simple to use GUI interface in addition to excellent extension capabilities to implement a Simalytic Function.

4.2.1 Workload Analysis Example

Identify: For this example, there is a single Order Entry transaction, OE, and a single Shipping transaction, S. The OE transactions are the workload of interest. The S transactions need to be included only to the extent they impact the OE transactions. However, some additional information about the S transaction response times is included to illustrate the pit-fall of modeling the systems independently. The S transaction arrival rate is kept constant at 0.1 arrivals per second. Only the OE transaction arrival rate is changed to represent growth in that workload.

Document: Refer to *Figure 2 A Simple Enterprise Model*. Assume that measurement data shows 30% of the OE transactions are also routed to the Shipping server. To keep this example simple, also assume that all of the transactions that execute on either server use the same resources. This means that there is no difference on the Order Enter server between the OE transactions that route to Shipping and those that don’t. It also means there is no difference between the transactions that execute on the Shipping server (i.e. an OE transaction routed to Shipping consumes the same resources on the Shipping server as an S transaction).

Measure: Because this is a hypothetical client/server environment, there are no actual measurements. Therefore, the results of a purely simulation model of the environment will be used to represent these measurements.

Correlate: The workload correlation is assumed.

4.2.2 Node Models Example

Build: The service times for the OpenQN model of each node is shown in the following table:

Server	Order Entry	Shipping
Workload	OE	S
Device 1: CPU	0.02	0.10
Device 2: Disk1	0.06	0.70
Device 3: Disk2	0.02	0.50
Device 4: Disk3		0.40
Device 5: Disk4		0.30
Total Service Time	0.10	2.00

Calibrate: The model of each node is assumed to be calibrated for this example.

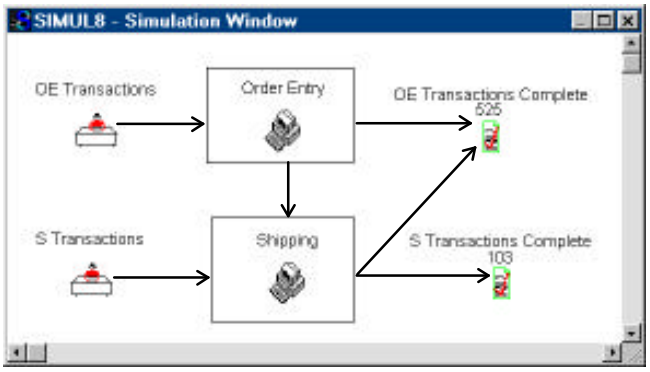
Run: An OpenQN model was run for each node.

Create: Partial results of the OpenQN model of each node are shown in the following table (not all data points are shown in the table to save space). Notice that the model was not run for the Order Entry server for a number of arrival rates because there was no significant change in response time. Also notice that the arrival rate step was reduced from .05 to .02 to better define the knee of the response time curve for Shipping:

Server:	Order Entry	Shipping
Arrival Rates	Response	Times
0.01	0.10	2.01
0.50		2.70
1.00	0.10	4.54
1.10		5.43
1.20		6.98
1.25		8.33
1.30		10.65
1.35		15.76
1.40		38.21
1.42		119.96
2.00	0.11	
10.00	0.20	
15.00	0.66	
15.75	1.15	
16.00	1.56	
16.25	2.46	
16.50	6.06	

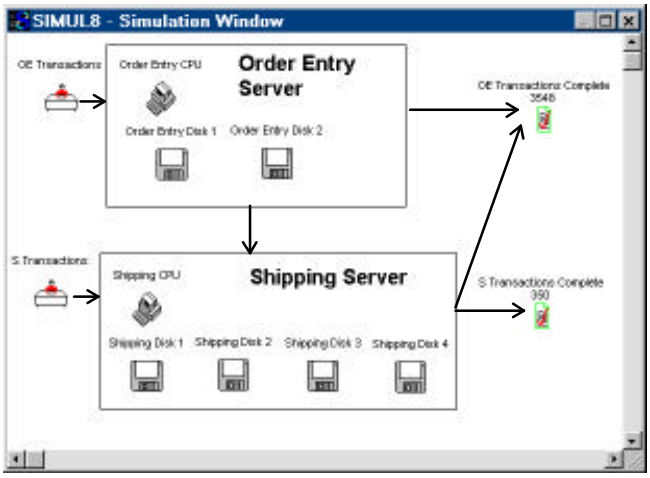
4.2.3 Simulation Model Example

Build: The overall simulation model was built using Simul8 as shown below:



Set: The service times of both the Order Entry and Shipping servers are set to the lowest response times from the table above. The replication factor for each server is set to 100 to avoid queuing. The average response times for that model run (ten trials) are 0.56 for OE and 2.01 for S. These response times are very close to the expected values of 0.7 and 2.00, respectively (the expected OE response time is 0.7 because of the routing to Shipping: 0.1+0.3*2.0=0.7) The OE response time is slightly lower because the routing was slightly lower than 30% in most of the trial runs due to the small number of transactions at the low arrival rate.

Calibrate: The results of the above model are compared to a pure simulation model, also built in Simul8, to calibrate the model. The service times are the same as shown in the table in section 4.2.2. The simulation model is shown below:



Selected results from multiple runs of this model are shown in the table below:

OE Arrival Rates	OE Response Times	S Response Times
0.01	0.677	2.093
0.10	0.730	2.119
0.20	0.749	2.151
0.50	0.787	2.273
1.00	0.857	2.498
2.00	1.069	3.155
3.33	1.777	5.495
3.70	2.332	7.487
4.00	3.317	10.674
5.00	59.415	215.970

To reduce the impact of arrival distributions, each data point is the average of ten trials for each arrival rate. The heading Arrival Rate refers to the arrival rate for the OE transactions. The S transaction arrival rate is kept at a constant value because it is not the workload of interest. Each trial was for 3600 simulation seconds (one simulation hour) with a 100 second warm-up period.

4.2.4 Simalytic Model Example

Create: Create the Simalytic Function. The Simalytic Function was created using Microsoft's Visual Basic. For this example, it is a very simple function that calculates the rolling average of the interarrival times for each workload and looks up the corresponding response time in a table.

Replace: Replace the static service times. The table below shows the results of the model trials after the Simalytic Function was implemented:

OE Arrival Rate	OE Response Time	S Response Time
0.01	0.661	2.080
0.10	0.716	2.123
0.20	0.728	2.126
0.50	0.754	2.166
1.00	0.780	2.292
2.00	0.907	2.696
3.33	1.622	5.085
3.70	2.036	6.468
4.00	2.838	9.062
5.00	11.492	38.144

Calibrate: The results of the above model are compared to a pure simulation model to calibrate the model as shown in *Figure 3 Response Time Comparison*. The Simalytic results track the simulation results. The slight under predicting is consistent with the simple implementation of the Simalytic Function and is fully explained in the author's ongoing research.

4.3 And The Answer Is…

How do the these results relate to the questions asked in section 4.1? *Figure 3* shows the answer. When the business volume grows to 3.33 OE transactions per second the response time will exceed the goal of 1.7 seconds. Furthermore, the way to keep OE transactions under the goal is to upgrade the Shipping server rather than the Order Entry server because the OE response time is directly related to the longer Shipping transactions.

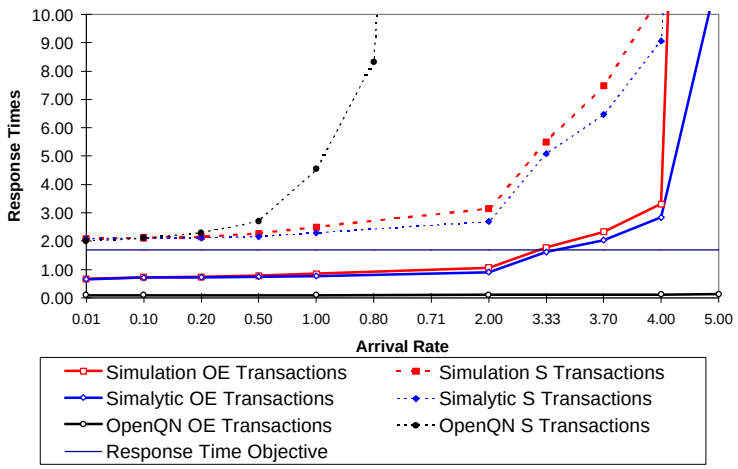


Figure 3 Response Time Comparison

Figure 3 graphs the response times for a number of different models. The response times are shown in pairs; one response time for the OE transactions and one for the S transactions. The OpenQN lines represent only the workload at the respective servers. All of the others include OE transactions sent to Shipping in the OE workload. The Simulation lines represent the pure simulation model and the Simalytic lines represent the Simalytic Model results. Because the S transaction workload arrival rate was kept constant, the Arrival Rate axis is the OE arrival rate except for the OpenQN S Transactions line, which shows what happens at the Shipping server. In all other cases, the increase in S transaction arrival rate is due to the transactions sent to Shipping from Order Entry.

From *Figure 3* we can make the following observations: OpenQN shows the Order Entry System response time to be flat and does not predict a response time of 1.7 until almost 16 transactions per second. The simulation and Simalytic Model results are very close and both show the OE workload exceeding the goal at 3.33 transactions per second .

Which is the best approach to use? The queuing theory tool greatly underestimates the OE workload response time because it does not account for the impact from the Shipping system. The results of a simple single-server simulation model (not shown) greatly overestimate the OE workload response time because of the rapid queue buildup at a single server. Only the full simulation model and the Simalytic Model represent the actual workload behavior. Which is the best is determined by the complexity of the modeling effort. The Simalytic Model is more attractive when the nodes are too complex to be easily modeled with a general simulation tool.

5. Conclusion

The traditional view of planning the capacity of a system is evolving because of the desire to predict the performance of the application. Applications designed to exploit a client/server architecture greatly increase the complexity of both the computer system configurations and the applications themselves. Predicting the responsiveness of those more complex applications requires a more complex modeling methodology. But adding complexity to a modeling effort also adds time, effort and cost. There are many techniques and tools that are beginning to address this evolution, but none of them can provide the desired level of detail for every situation and every application.

By following these steps for implementing a Simalytic Model, the modeler can rapidly produce an application model at the level of detail needed to make business decisions. Combining different modeling techniques (simulation and analytic queuing theory) and different modeling tools (platform-centric and general purpose) reduces the time, effort and cost of developing an enterprise application model. As more detailed results are required, more sophisticated tools can then be used to increase the understanding of critical sections of the model.

This level of analysis provides insight into the application's future performance that would not otherwise be available. Using the Simalytic Modeling Technique both improves the understanding of the application as well as identifies which systems require more detailed analysis. It protects the investment an organization has made in acquisition and training of existing tools. It allows the most appropriate tools to be used for each modeling effort.

Capacity planning is still fundamental to business success. But just as application designs are moving away from single system solutions, modeling for capacity planning must move away from single system analysis and begin predicting the application across the enterprise.

6. Acknowledgments

The author would like to thank Mr. Rick Lebsack and Mr. Larry Kayser for their interest and in-depth critiques of the early versions. A special thanks is also expressed to Dr. John Zingg, Dissertation Committee Chair, for his insight and assistance.

7. References

Gunther, Neil J. 1995. Performance Analysis and Capacity Planning for Datacenter Parallelism. In Proceedings. Computer Measurement Group: CMG, Inc.

Hatheson, Amanda. 1995. Two Unix Client/Server Capacity Planning Case Studies. In British Proceedings. Computer Measurement Group: CMG, Inc.

Kobayashi, Hisashi. 1981. Modeling and Analysis: An Introduction to System Performance Evaluation Methodology. The Systems Programming Series. Reading, MA: Addison-Wesley Publishing Company.

Menascé, D., V. Almeida, and L. Dowdy. 1994. Capacity Planning and Performance Modeling: from mainframes to client-server systems. Englewood Cliffs, New Jersey: Prentice Hall.

Norton, Tim R. 1996. Simalytic Enterprise Modeling: The Best of Both Worlds. In Proceedings. Computer Measurement Group,: 1-12. San Diego, CA: CMG, Inc.

Norton, Tim R. 1997a. Simalytic Hybrid Modeling: Planning the Capacity of Client/Server Applications. In Proceedings. 15th IMACS World Congress,: 1-6. Berlin, Germany: International Association for Mathematics and Computers in Simulation.

Norton, Tim R. 1997b. Simalytic Modeling: A Hybrid Technique for Client/Server Capacity Planning. In Proceedings. Summer Computer Simulation Conference,: 1-6. Arlington, Virginia: Society for Computer Simulation.

Pooley, Rob. 1995. Performance Analysis Tools in Europe. Informationstechnik und Technische Informatik 37 : 10-16.

Smith, Connie U. 1995. The Evolution of Performance Analysis Tools. Informationstechnik und Technische Informatik 37 : 17-20.

Visual. Simul8 . Visual Thinking International Limited, 141 St James Rd., Glasgow, UK G4 0LT.

Wilson, Gregory L. 1994. Capacity planning in a high-growth organization. In Proceedings. Computer Measurement Group. Orlando, FL: CMG, Inc.