3. Simalytic Modeling Methodology

3.1 Overview

"Simalytic" (**Sim**ulation/An**alytic**) Modeling[TM] [2] is a hybrid modeling technique. The methodology uses a general purpose simulation modeling tool as an underlying framework and the results of an analytic modeling tool to represent the individual nodes or systems. 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.

The two key differences between the existing modeling techniques and the Simalytic Modeling technique are interoperability and reuse. The first, interoperability, is the capability 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, reuse, is the capability 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 by using the results from commercially available platform-centric tools or existing detailed application models.

3.2 Methodology

Simalytic Modeling brings together existing performance models and application information. One of the problems with queuing theory is the reliance on averages, such as average response time, average service time and average arrival rate. The goal of Simalytic Modeling is to model the application over longer periods of time to understand the appli-

[2] *Simalytic*[TM], *Simalytic Modeling*[TM], *Simalytic Modeling Technique*[TM] and *Simalytic Enterprise Modeling*[TM] are trademarked by Tim R. Norton.

cation 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[3] usually produce better model results because there is less variation in the measurement data. The node model can produce very accurate response time predictions when built with a queuing theory tool using a short data collection interval to minimize the variability in the data.

Simalytic Modeling also provides a useful technique for modeling the behavior of a single system application over very long periods of time. Because the variation in the arrival rate distribution increases as the model interval increases, such long interval models are generally implemented using simulation tools. However, the capacity planning modeling tool of choice has generally been a platform-centric queuing theory tool. Creating an entirely new model of the application in a different tool is seldom cost effective and therefore seldom done. Using the Simalytic Modeling Technique, the modeler only creates a very simple simulation framework model to drive the single node (most likely from trace data) that has been implemented using a Simalytic Function. The majority of the work in constructing the Simalytic Model would involve using the queuing theory tool already used to model the system. Thus a Simalytic Model would provide a higher level of understanding of the dynamics of an application over time that would not be cost effective using other techniques.

[3] In this context, 'interval' refers to the time period for which measurement data was collected to be used in building a model. Interval selection is the analysis of all available measurement data to determine the interval that is most representative of the application situation to be modeled.

3.2.1 Methodology Assumptions

Simalytic Modeling is not a technique for collecting data or measuring systems or applications. There are several underlying assumptions that must be true before the Simalytic Modeling technique can be used:

- The definition of each workload to be modeled must be consistent across all the models used, including all the nodes where each workload executes and the simulation model framework. Therefore, all assumptions used for each of the node level models must be consistent with all of the other node models and must be included in the simulation framework model. For example, assume the Order Entry workload on the Order Entry system is defined as transactions OE1 and OE2 from any user and that the OE2 transactions are routed to Shipping. The Order Entry workload on the Shipping system must also use the same definition; it cannot include any other transactions or exclude any users. The simulation framework model must route the OE2 transactions to Shipping. It cannot route OE1 transactions and any measurement data from the Shipping system must not contain any OE1 transactions.
- The applications to be modeled at the enterprise level must be understood at the enterprise level, which includes transaction arrival distributions. As in the above example, the application must be well enough understood to know not only the percentage of transactions that access the Shipping server, but also if those accesses are generated by only the OE1 transaction or by both the OE1 and OE2 transactions (and the respective percentages).

- A valid model must exist for each system or node to be included in the application enterprise model. The node models must be proven to produce accurate predictions, within an acceptable error range, for the application being modeled. The accuracy of the overall Simalytic Model will be reduced as the complexity or the heterogeneousness of the individual node models increases. For example, data collected for a single workload composed of very similar transactions distributed evenly over the data collection interval will produce a very accurate and reliable model of that application on that system. However, the results produced by the node model will be much less accurate if the node model must provide results for several different workloads or there is wide variation in the characteristics of the transactions in the workload.
- The simulation tool selected for the enterprise model framework must support submodels, must be able to invoke external functions and must support the modeling of individual transactions.

The model builder must understand the applications and the individual systems in the enterprise (or have access to others with such understanding) before they can be put together into an enterprise level model. Using the example from section *1.6.3 Client/Server Environment* on page 16, the relationship between the Order Entry transactions and the Shipping system must be understood and measurable. If the model builder does not have any information about which transactions send requests to the Shipping system, he cannot build a model that matches the application. Once the application is understood, the model builder must know how each system reacts to different transaction loads. How this information is developed will vary across the different nodes depending

on particulars such as the hardware, the operating system and the level of data collection. Techniques for developing a look-up table of valid response time predictions for each node include:

- collecting measurement data at different transaction rates,
- using platform-centric modeling tools or
- calculating expected response times based on benchmarks.

The fully implemented Simalytic Modeling tool would integrate a platform-centric queuing theory model into the simulation tool by replacing the submodels for individual systems with calls to the queuing theory model tool.

3.2.2 Methodology Process

Once the model builder has all the fundamental information, an enterprise level model can be constructed. The simplest way to do this is to construct a very high level simulation model of the enterprise where each system, or node, is a single server capable of unlimited parallelism. Then, instead of using a pre-defined service time, each server would use a transform function that maps the transaction arrival rates to service times. 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 for the server. The transform function will increase the node service time as the arrival rate at that node increases to have the same effect as a constant service time plus queue time. Each node in the simulation model must allow enough parallelism to avoid queuing to enter the node because such queuing is already accounted for in the submodel response time. Additional queuing in the simulation framework would not be an accurate reflection of the actual system for those nodes where the transform function has been implemented. Other model

elements, such as networks or delay servers, and servers not implemented with a transform function are implemented as standard nodes in the simulation model. This also applies to any node implemented with a transform function where the response time data used to create the transform function does not include queuing to enter the node. Such cases require extra care to insure that any queuing in the simulation model reflects only queuing not accounted for in the value returned by the transform function.

Continuing with the same example, some number of the Order Entry transactions would be routed to the Shipping server. Assume it has been determined that Shipping can provide a response time of one second when arrivals are less than three per minute and a response time of two seconds when arrivals are more than three per minute. When the Order Entry transaction arrival rate increases such that more than three per minute are sent to Shipping, the response time will jump from one to two seconds. This is a very simple example, but it illustrates the point. The increased service time at Shipping will cause the overall response time for those transactions to increase, which will be seen as a longer average response time or reduced throughput for the application.

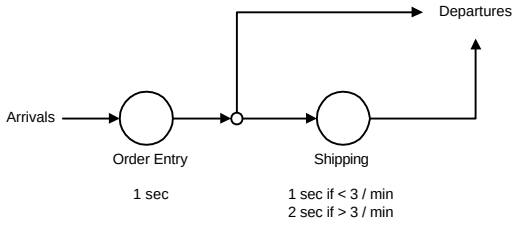


Figure 7 Simple Enterprise Model

Figure 7 Simple Enterprise Model shows a diagram of this model. The response time is measured from **Arrivals** to **Departures**, either through the **Shipping** node or around it. If there is a limit on the number of transactions that can be active in the Order Entry system at any given time, then there could be some queuing to get into the system. This would represent a user's workstation waiting to send

the transaction to the server. 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 Order Entry transactions (i.e. those that are 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 could adjust the service time in the Order Entry server by replacing the measured wait time component of the response time with the projected delay from the Shipping server. This is a level of detail beyond the initial discussion of Simalytic Modeling in this paper, but the technique lends itself to this type of extension.

The next question is "how does the Shipping server node in the model know what arrival rate to use for a single transaction?" An arrival rate must be calculated for each transaction based on the current level of activity. In this example, the load is determined by calculating the arrival rate from a moving average of the prior two transaction interarrival times, which acts as a smoothing function. This arrival rate is used in a transform formula similar to the one shown in section *3.3 Foundation* on page 61. If the average interarrival time is less than 20 seconds, then the average arrival rate to account for that interarrival time would have to be greater than three per minute. If the average interarrival time is longer than 20 seconds, then the average arrival rate would have to be less than three per second. Therefore,

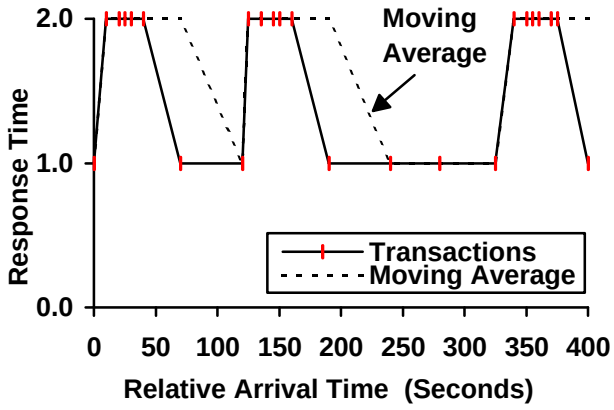


Figure 8 Shipping Transaction Arrivals

knowing the average interarrival time over a small number of transactions, the model can calculate an arrival rate at each server that represents the current activity. (In this context, current means a period of time just prior to the transaction being serviced that is relatively small as compared to the total overall time of the model interval.) *Figure 8 Shipping Transaction Arrivals* shows the actual transactions and their respective response times. When the transaction arrivals are close together the response time is high and when the arrivals are further apart the response time is low. The line labeled **Moving Average** shows the response time derived from using the moving average of the prior two interarrival times. (Two interarrival times is an extremely small sample used only for illustration. A larger sample, sized appropriately for the workload, would be used in real modeling situations.) This average tends to incorrectly predict the response time of the first transaction in each group of lower activity, but provides a better workload profile that using the average of all interarrival times (17.39), which would return a response time of 1.0 for all transactions. This smoothing allows the Simalytic Model to account for variability in the arrival rate that causes changes in the node level response times due to changes in the load at the node. Variability in the service times of devices within the nodes is accounted for in the node level models and is reflected in the different response times at the different loads.

The next step is to analyze the model using the business objectives. 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 OE 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 must also query the Shipping system is assumed to be 30%. The re-

sponse time goal for the Shipping system is less than 10 seconds (because these transactions generally do not involve a waiting customer) and this long response time is acceptable. 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 Order Entry system upgrade is needed to fix the problem?"

Figure 9 Model Results Analysis shows the hypothetical results of this example model. When each of the systems are analyzed independently, neither of the response times ever approach the business goal of 10 seconds for the Shipping transactions and 1.7 seconds for the Order Entry transactions. However, when the relationship between Shipping and Order Entry is added to the chart in the form of results from a Simalytic Model, the revised Order Entry response times show that the system will need to be upgraded by year end, well within the scope of the analysis. In addition, the 'fix' to the problem is to

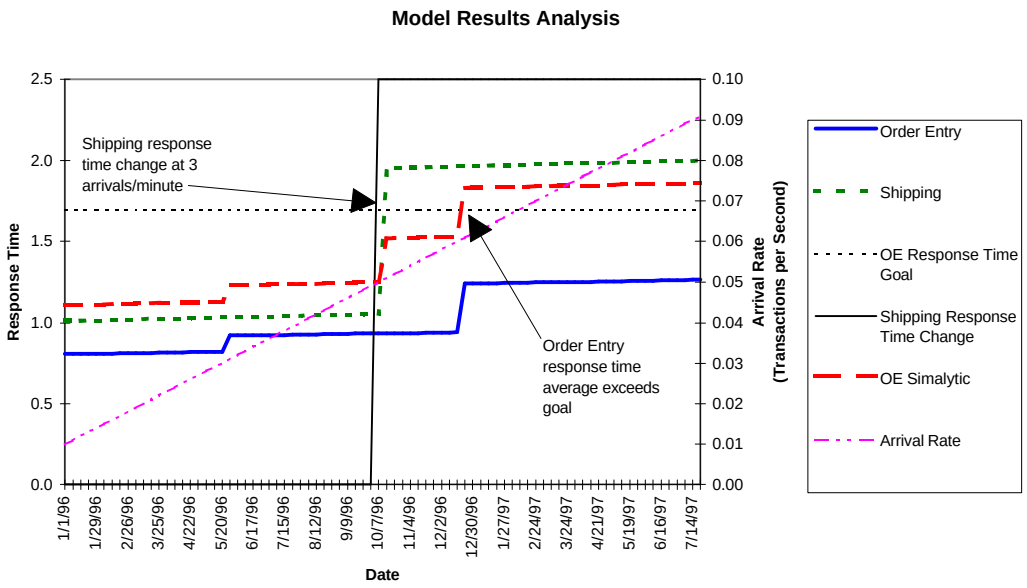


Figure 9 Model Results Analysis

upgrade the Shipping system, which never exceeds its response time goal. The Simalytic Model allows the analyst to see the impact of relationships that, although known, may not be full appreciated.

3.3 Foundation

The above example shows how simulation and analytic modeling techniques can work together, but it does not prove that the Simalytic Modeling technique is viable. To do that we need to look at the mathematical formulae behind the two techniques and how they are combined into the Simalytic Modeling formula shown in *Equation 3 Simalytic Response Time Formula* by using a transform function. As the framework for a Simalytic model is a simulation model, we start with the simulation response time formula shown in *Equation 2 Simulation Response Time Formula* on page 35, which is a summation of all of the server response times divided by the number of transactions to get the average response time. The server time from each transaction iteration (T_i) is replaced with a transform function $f(\lambda_i)$, where i is the iteration index and λ_i is an arrival rate input to the transform function for that iteration.

The arrival rate is calculated from the prior interarrival times to provide a smoothing function across the modeling interval. This allows the arrival rate used as a pa-

Equation 3 Simalytic Response Time Formula

$$T = \frac{\sum_{i=1}^{n_t} f(l_i)}{n_t}$$

where:

f = a transform function

l_i = arrivals per second as:

$$l_i = \frac{b}{(c_i - c_{i-w}) / w}$$

c_i = simulation clock value

b = simulation clock ticks per second

w = number of transactions in averaging window:
for rolling average: when $w > i$, then $w = i$
for cumulative average: $w = i$

rameter to the Simalytic Function to be an average arrival rate, which is appropriate for use with the queuing theory formula, but still localized to the current transaction, which is appropriate for use with the simulation formula. The arrival rate is calculated from the interarrival times by dividing the number of simulation clock ticks per second (*b*) by the difference in the simulation clock value for the current iteration and a prior iteration (c_i - c_{i-w}) divided by the number of transactions between the current iteration and the prior iteration (*w*) as shown in *Equation 3*. There are two special cases for *w*. When a rolling average is desired, *w* must be set equal to *i* for those iterations where *w>i*, which provides a cumulative average for the first *w* iterations. When a cumulative average is desired, *w* must be set equal to *i* for all iterations.

The transform function *f* is used to transform an arrival rate into a service time which is based either on the results of an analytic response time formula similar to the one in *Equation 1* on page 32 or on the results of a platform-centric analytic tool. The transform function is a step function whose output (response time) either remains constant or increases as the input (arrival rate) increases. The service time returned by the transform function represents the response time for that node given the current load (expressed as the arrival rate) and any other conditions at the node that impact response time. It may be a known and understood formula, as in the case of Equation 1, or it may be the results derived from proprietary algorithms implemented in a commercial tool. The use of such a tool allows a model builder to concentrate on the enterprise view of the application because the system level model has already been calibrated. The transform function can be implemented directly by invoking a submodel to calculate and return the response time based on λ_i. Or, it can be implemented indirectly by creating a look-up table of the results

of an analytic formula or tool that has been invoked at some other time for some subset of the expected values for the λ_i's. In addition, the transform function can be enhanced to utilize other information known about the node at the time the transaction arrives, such as queue lengths, resource consumption by other workloads and recent historical performance. Such a complex function would require additional inputs as it would then no longer be a function of only the arrival rate. The simulation model invokes a submodel that returns the response time appropriate for the λ passed as a parameter. In an actual implementation, the simulation tool can consider other information that is not passed as a formal parameter. Although a formula could be devised to replace the look-up table, using a table will be more practical than using an analytic tool that must be invoked for each arrival rate to be investigated. However, if the Simalytic Modeling technique becomes widely accepted, then the analytic tools could be modified to produce such a formula instead of response time reports.

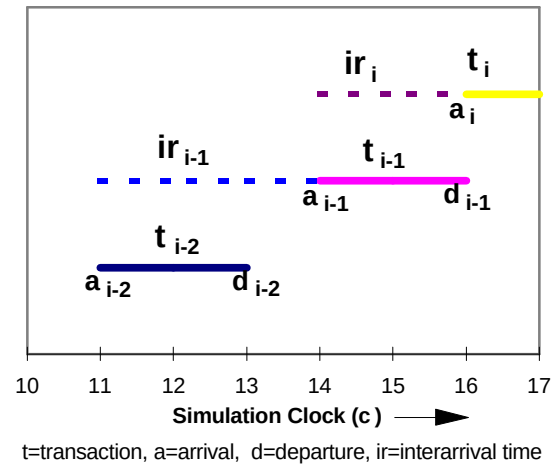


Figure 10 Interarrival Time Example

Figure 10 Interarrival Time Example shows an example of how an average arrival rate can be calculated from the interarrival times between transactions. In the figure, *t* represents a transaction, *i* represents the transaction index, *c* represents the simulation clock and *w* represents the number of interarrival times to include in the average. The i^{th} transaction (t_i) arrives at the clock value of c_i, which is considered the clock for the i^{th} transaction. The average

arrival rate is calculated over the transaction window (w) of prior transactions. Therefore, the accumulative interarrival time for all transactions in the window starts at the arrival of the w^{th} prior transaction (a_{i-w}) and is calculated as the clock value at a_i (c_i) minus the clock value at a_{i-w} (c_{i-w}). As an example, if $w = 2$, $a_i = c_i = 16$, and $a_{i-w} = c_{i-w} = 11$, then the cumulative interarrival time would be 5 ($a_i - a_{i-w} = 16 - 11 = 5$) and the average interarrival time would be 2.5 ($5/w = 5/2 = 2.5$). If there are six clock ticks per second, then the average arrival rate used for the i^{th} transaction (λ_i) would be 2.4 transactions per second ($6/2.5 = 2.4$).

Once the arrival rate is calculated, the transform function (f) is called with the arrival rate input for that transaction (λ_i). The transform function provides a step function approximation of the results of the queuing theory formula *Equation 1 Analytic Response Time Formula* on page 32 to a greater or lesser degree, depending on the complexity designed into f. *Figure 11 Simalytic Function* shows a hypothetical example of the relationship between this transform function (the short broken line) and the results of *Equation 1 Analytic Response Time Formula* on page 32 (the solid line) and the results of *Equation 2 Simulation Response Time Formula* on page 35 for each of a series of simulations at different arrival rates (the long broken line). The relationship between these three techniques is evident by how well they track each other. Even with only a single simulation execution for each arrival rate and only five values in the transform function look-up table, the graph of the re-

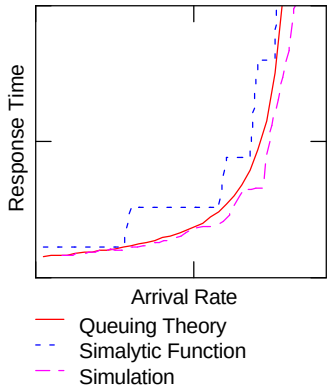


Figure 11 Simalytic Function

sults appears to show a high correlation between the three techniques, which provided the initial encouragement for this research.

The transform function is a key principle to Simalytic Modeling because it provides the bridge between the two types of models (simulation and queuing theory). The calculation of an arrival rate provides the input for the transform function which returns a response time output based on a queuing theory submodel. One of the assumptions for Simalytic Modeling (see section *3.2.1 Methodology Assumptions* on page 54) discusses the importance of using only results from validated node models in the Simalytic Model. The ability to use arrival rates calculated from subsets of the interarrival times of all transactions is valid because the transform formula for each node returns results that have already been proven to produce acceptable predictions for the loads represented by those arrival rates. The arrival rate calculated from a subset of the interarrival times is really a way of segmenting the model interval and obtaining the arrival rate that would cause the given response time in a steady state over each segment. Each arrival rate will already be validated as part of the construction of the node level model.

Even assuming a best case situation where the interarrival times of the actual transactions are uniform (no variation), there would still be variation in the response times due to variation in the workload (different transactions, different calculations, different logical path, etc.) and due to variations in the device service times (CPU cache and pipeline, disk cache and rotation, interference from higher priority workloads, etc.). When using any queuing theory model, an assumption is accepted that the predicted results represent the workloads being investigated even though they are averages based on the average of each of the parameters over the data collection interval. This is similar to the FESC

(flow-equivalent service center) decomposition technique discussed in (Menascé, Almeida, and Dowdy 1994), used for solving complex queuing theory models. The system to be modeled is divided into parts that can each be analyzed and solved independently. Each part is then replaced with a single server that is representative of the flow through that part of the system. One of the main points in each of the discussions (Menascé, Almeida, and Dowdy 1994, 163-7 and 236-9) is that the FESC must be solved independently and must maintain the 'flow' of the overall model (i.e. the behavior of each FESC must be almost indistinguishable from the subsystem it replaces). The subsystem model is solved to obtain the subsystem throughputs as a function of multiprogramming level (for closed models) or the subsystem response times as a function of arrival rate (for open models). The FESC is then substituted for the subsystem in the overall model using the function to describe the subsystem's behavior. (Menascé, Almeida, and Dowdy 1994, 236) cites (Chandy and Sauer 1978) that the flow-equivalent method yields exact results when applied to closed single class product form models and cites (Cortois 1975) that little error is introduced if the transition rate within the submodel is much greater than the interaction rate between the submodel and the overall model (which will necessarily be the case when the submodel represents an entire independent system or node).

Equation 3 only calculates the response time for a single workload on a single server. The response times for additional workloads and servers would be calculated the same way. The average system response time could then be calculated by adding the response times together based on the probability of each workload visiting each server. As can be seen by this simple example, the calculations will quickly grow out of hand. To avoid this, the Simalytic Modeling Technique uses existing simulation and queuing theory

tools together to implement a Simalytic Model. In addition, the simulation tools allow transactions to be assigned attributes that can contain application design information not available in the measurement data.

3.4 Development of Foundation

The discussions in sections *2.2.1 Analytic Queuing Theory* on page 30, *2.2.2 Simulation* on page 34, and *3.2 Methodology* on page 52, addressed only single server situations. This section expands on the formulae presented in those sections to show that the technique is viable for additional servers. Several different enterprise system topologies are investigated and the mathematical formulae to calculate the overall system response time is developed for each topology:

- The single server system as discussed in the earlier sections (reproduced here for the convenience of the reader).
- A two server system where all transactions visit both servers.
- A two server system where transactions are forked and each image of the transaction visits each server in parallel.
- A two server system where transactions are routed to a server based on some probability.
- A three server system where transactions are routed to a server based on some probability.
- A four server system where transactions are routed to a server based on some probability.
- A generalized n-server system where transactions are routed to a server based on some probability.

It is assumed that these scenarios (single server, series of servers, fork/join and probability routing) represent all of the most likely scenarios needed to model an application at the enterprise level and that any other scenarios can either be simplified to one of these or discounted as extremely unlikely. The percentages of transactions routed to each server in each of the routed scenarios are all positive values such that their sum is equal to 1 (100%). Each individual percentage must be in the range of 0 to 1 (0% to 100%). The arrival distribution at each server is assumed to be Poisson. Only the servers will be addressed in these scenarios because it has been shown that, based on Poisson distributions, the arrival rate at a server from multiple sources (i.e. workstations) is the sum of the arrival rates generated by each source and that the arrival rate at one of multiple servers from a single source is the original rate times the probability that server will be selected (Kobayashi 1981, 103-5; Pooch and Wall 1993, 342-3).

This section also shows that each of these scenarios can be represented by the formulae used for either a series or a multi-server system with probability routing. Because the other scenarios are equivalent to one of these formulae, the number of scenarios investigated in the following sections, *4 Simalytic Model Development* on page 97 and *5 Investigations into Simalytic Modeling* on page 127, are greatly reduced.

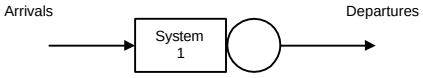
3.4.1 Mathematical Formulae

Each of the figures in this section shows a graphical diagram for one of the above scenarios with the queuing theory response time formula (top), the simulation response time formula (middle) and the Simalytic response time formula (bottom) for that scenario. The following legend applies to all of the figures in this section:

s	=	The total number of nodes (servers) in the system.
x	=	Any one of the s nodes.
T	=	The average response time for all transactions in the system.
T_x	=	The average response time for all transactions that visit node x. The sub-script i denotes the response time for the i^{th} transaction to be serviced by that node, such as: T_{2_i} would be the response time of the i^{th} transaction at node 2.
S_x	=	The service time for node x. This is the total time to process a transaction at the server, or node, without any queue time.
n_t	=	The total number of transactions processed by the system.
p_x	=	The probability that a transaction will take the route to server, or node, x. In all cases p must meet the conditions: $0 \le p_x \le 1$ and $p_1 + p_2 + \ldots p_s = 1$. This insures that all of the transactions visit one of the servers and that no transactions visit any server more than once.
l	=	The arrival rate of transactions into the system.
$f_x(l_{x_i})$	=	The transform function for node x as discussed in section *3.3 Foundation* on page 61 which always returns a value equivalent to T_{x_i}.
MAX	=	A function that returns the largest of the parameters.

3.4.1.1 Single Server System

The first formula to be discussed is for the single server system shown in *Figure 12 Single Server System*. All three of the formulae are the same as those presented earlier and are shown here in the same format that is used to discuss the other formulae. In this case, the system response time is the same as the server response time.



$$T = T_1 \qquad T = \frac{S_1}{1 - l\, S_1}$$

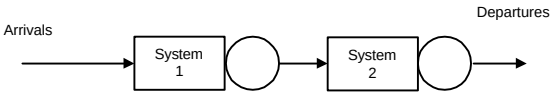
$$T = T_1 \qquad T = \left(\frac{\sum_{i=1}^{n_t} T_{1i}}{n_t} \right)$$

$$T = T_1 \qquad T = \left(\frac{\sum_{i=1}^{n_t} f_1(l_{1i})}{n_t} \right)$$

Figure 12 Single Server System

3.4.1.2 Two Server System

Figure 13 Simple Two Server System shows a simple system consisting of two servers, or nodes, where each transaction visits each node in series. The overall response time is the sum of the response times of each of the servers. Using the concept of a FESC (flow-equivalent service center) decomposition technique (Menascé, Almeida, and Dowdy 1994) discussed earlier in this section, a two server system can be simplified to a single server system like the one shown in *Figure 12*. This simplification allows any series of servers to be represented by a single FESC server because the series can be simplified, two servers at a time, until only a single server remains. Therefore, no additional formulae for other systems consisting of a series of servers are developed in this section.



$$T = T_1 + T_2 \qquad T = \frac{S_1}{1 - l\, S_1} + \frac{S_2}{1 - l\, S_2}$$

$$T = T_1 + T_2 \qquad T = \left(\frac{\sum_{i=1}^{n_t} T_{1i}}{n_t} + \frac{\sum_{i=1}^{n_t} T_{2i}}{n_t} \right)$$

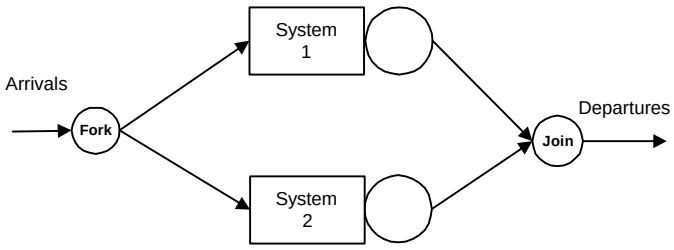
$$T = T_1 + T_2 \qquad T = \left(\frac{\sum_{i=1}^{n_t} f_1(l_{1_i})}{n_t} + \frac{\sum_{i=1}^{n_t} f_2(l_{2_i})}{n_t} \right)$$

Figure 13 Simple Two Server System

3.4.1.3 Fork/Join System

Figure 14 Two Server System with Fork/Join Routing shows a system consisting of two servers, or nodes, where each transaction visits each node in parallel. The transaction is reproduced or split in such a way that identical copies are sent to each server at the same time. The overall response time is the greater of the response times of each of the servers. Because the individual transaction identity is not preserved in a queuing theory model, fork/join routing cannot be calculated using queuing theory. A queuing theory model can only approximate the response time in this situation by calculating the overall response time independently for each server and using the larger of the response times. In addition, the fork/join routing system cannot be simplified to a single node because of the relationship between the service times of the parts of a forked transaction.

Fork/join systems are not considered further for two reasons. First, the formulae



T= *Cannot be calculated with queuing theory*

$$T = \frac{\sum_{i=1}^{n_t} MAX(T_{1_i}, T_{2_i})}{n_t}$$

$$T = \frac{\sum_{i=1}^{n_t} MAX(f_1(l_{1_i}), f_2(l_{2_i}))}{n_t}$$

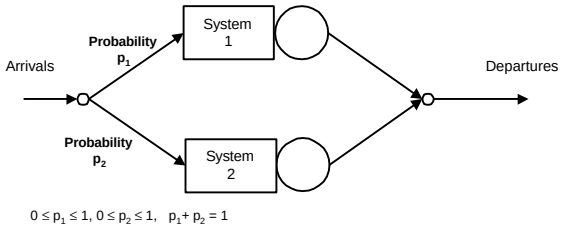
Figure 14 Two Server System with Fork/Join Routing

shown in *Figure 14* cannot be cross-checked against each other because there is no queuing theory solution. Second, the use of fork/join routing is assumed to be an unusual situation in actual client/server implementations. Therefore, investigations into the behavior of fork/join routing systems is deferred as an area for future research.

3.4.1.4 Two Server Routing

Figure 15 Two Server System with Probability Routing shows a system consisting of two servers, or nodes, where each transaction visits one of the servers depending on some probability. The overall response time is the sum of the response times of each of the servers weighted by the probability of visiting that server. This type of system is usually thought of as a routing system where each transaction visits one or the other of the servers.

The remaining sections briefly examine adding additional servers to produce a three server (section *3.4.1.5*) and a four server (section *3.4.1.6*) system. The final section (*3.4.1.7*) shows a generalized multi-server system with probability routing that can be used to represent all probability routing systems, including the two server system shown in *Figure 15*.



$$T = p_1 T_1 + p_2 T_2 \qquad T = \frac{p_1 S_1}{1 - p_1 l S_1} + \frac{p_2 S_2}{1 - p_2 l S_2}$$

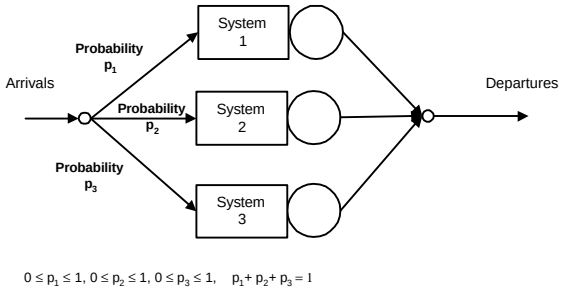
$$T = p_1 T_1 + p_2 T_2 \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} T_{1i}}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} T_{2i}}{p_2 n_t} \right)$$

$$T = p_1 T_1 + p_2 T_2 \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} f_1(l_{1_i})}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} f_2(l_{2_i})}{p_2 n_t} \right)$$

Figure 15 Two Server System with Probability Routing

3.4.1.5 Three Server Routing

Figure 16 Three Server System with Probability Routing is similar to *Figure 15* but it shows a system consisting of three servers, or nodes, where each transaction visits one of the servers depending on some probability. The overall response time is the sum of the response times of each of the servers weighted by the probability of visiting that server. Therefore, the addition of the third server consists of adding the proportional response time component for the new server to the proportional response times of the other servers.



$$T = p_1 T_1 + p_2 T_2 + p_3 T_3 \qquad T = \frac{p_1 S_1}{1 - p_1 l S_1} + \frac{p_2 S_2}{1 - p_2 l S_2} + \frac{p_3 S_3}{1 - p_3 l S_3}$$

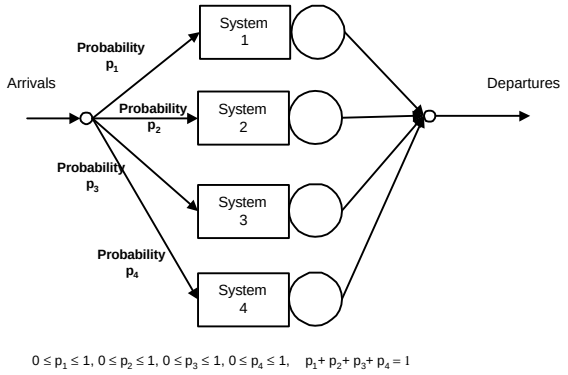
$$T = p_1 T_1 + p_2 T_2 + p_3 T_3 \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} T_{1i}}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} T_{2i}}{p_2 n_t} + \frac{\sum_{i=1}^{p_3 n_t} T_{3i}}{p_3 n_t} \right)$$

$$T = p_1 T_1 + p_2 T_2 + p_3 T_3 \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} f_1(l_{1i})}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} f_2(l_{2i})}{p_2 n_t} + \frac{\sum_{i=1}^{p_3 n_t} f_3(l_{3i})}{p_3 n_t} \right)$$

Figure 16 Three Server System with Probability Routing

3.4.1.6 Four Server Routing

Figure 17 Four Server System with Probability Routing is similar to *Figure 15* but it shows a system consisting of four servers, or nodes, where each transaction visits one of the servers depending on some probability. The overall response time is the sum of the response times of each of the servers weighted by the probability of visiting that server. Therefore, the addition of the fourth server consists of adding the proportional response time component for the new server to the proportional response times of the other servers.



$$T = p_1 T_1 + p_2 T_2 + p_3 T_3 + p_4 T_4 \qquad T = \frac{p_1 S_1}{1 - p_1 l S_1} + \frac{p_2 S_2}{1 - p_2 l S_2} + \frac{p_3 S_3}{1 - p_3 l S_3} + \frac{p_4 S_4}{1 - p_4 l S_4}$$

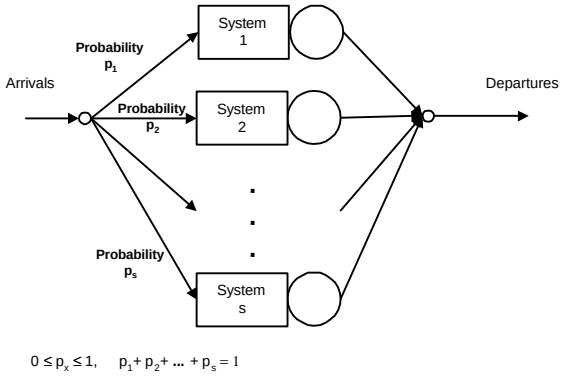
$$T = p_1 T_1 + p_2 T_2 + p_3 T_3 + p_4 T_4 \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} T_{1i}}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} T_{2i}}{p_2 n_t} + \frac{\sum_{i=1}^{p_3 n_t} T_{3i}}{p_3 n_t} + \frac{\sum_{i=1}^{p_4 n_t} T_{4i}}{p_4 n_t} \right)$$

$$T = p_1 T_1 + p_2 T_2 + p_3 T_3 + p_4 T_4 \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} f_1(l_{1i})}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} f_2(l_{2i})}{p_2 n_t} + \frac{\sum_{i=1}^{p_3 n_t} f_3(l_{3i})}{p_3 n_t} + \frac{\sum_{i=1}^{p_4 n_t} f_4(l_{4i})}{p_4 n_t} \right)$$

Figure 17 Four Server System with Probability Routing

3.4.1.7 Generalized Multi-Server Routing

Figure 18 Generalized Multi-Server System with Probability Routing is similar to *Figure 15* but it shows a system consisting of an unspecified number of servers, or nodes, where each transaction visits one of the servers depending on some probability. The overall response time is the sum of the response times of each of the servers weighted by the probability of visiting that server. Therefore, as shown in the above sections (*3.4.1.5* and *3.4.1.6*), the addition of another server consists of adding the proportional response time component for the new server to the proportional response times of other servers. Any number of servers can be added by starting with the two server formulae shown in *Figure*



$$T = p_1 T_1 + p_2 T_2 + \ldots + p_s T_s \qquad T = \frac{p_1 S_1}{1 - p_1 l S_1} + \frac{p_2 S_2}{1 - p_2 l S_2} + \ldots + \frac{p_s S_s}{1 - p_s l S_s}$$

$$T = p_1 T_1 + p_2 T_2 + \ldots + p_s T_s \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} T_{1i}}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} T_{2i}}{p_2 n_t} + \ldots + \frac{\sum_{i=1}^{p_s n_t} T_{si}}{p_s n_t} \right)$$

$$T = p_1 T_1 + p_2 T_2 + \ldots + p_s T_s \qquad T = \left(\frac{\sum_{i=1}^{p_1 n_t} f_1(l_{1i})}{p_1 n_t} + \frac{\sum_{i=1}^{p_2 n_t} f_2(l_{2i})}{p_2 n_t} + \ldots + \frac{\sum_{i=1}^{p_s n_t} f_s(l_{si})}{p_s n_t} \right)$$

Figure 18 Generalized Multi-Server System with Probability Routing

15 and adding one more server until the desired number of servers is represented in the system. This can be viewed as the opposite of simplification using FESC where a single server is considered a submodel and decomposed into its server components.

The formulae shown in *Figure 18* can be used to represent all of the formulae presented above other than a series of systems, which is represented by *Figure 13*. The probability is set to zero for all servers not present in the system being examined. When a transaction can be routed between a group of *x* servers, the sum of the probabilities for that group is always equal to one (100%) because all transactions must visit one, and only one, of the servers in the group.

3.4.2 Formulae Significance

The formulae discussed in section *3.4.1 Mathematical Formulae*, on pages 69-78, are important to the proof of the hypothesis because they show both a formal relationship between the different modeling techniques and a formal relationship between the formulae and the concept of a FESC (flow-equivalent service center). The relationship between the techniques is evident in the simple examples provided, where each of the models can be solved by each of the techniques, because each of the formulae calculate the same value, the average response time, for each model. The following section, *3.5 Validation*, expands on the ideas presented in this section by implementing these formulae using a mathematical language and shows the validity of this relationship with actual formulae results. Although not intended as a proof of the FESC decomposition technique (Menascé, Almeida, and Dowdy 1994), this discussion also shows how this well-established technique can be applied to each of the three modeling techniques. The formulae provide a solid foundation for the development and presentation of the remaining results.

3.5 Validation

3.5.1 Validation Overview

Simalytic Models must be validated at two levels. First, each model used for a system or node in the enterprise must be validated and calibrated for that system. This means not only that the response time versus load (arrival rate) relationship is valid for all arrival rates seen, but that it also holds across all of the possible arrival rates for all workloads likely to be generated by the enterprise model. If a submodel does not accurately represent the "knee" in the response time curve of one of the systems, the enterprise model will not accurately predict beyond that point. Second, because the Simalytic Modeling includes a model of the relationships within the applications being modeled, measurement data must be collected to validate those relationships. For example, if the model from the earlier example shows 30% of the Order Entry transactions are routed to Shipping, the model builder must collect the data to support that assumption.

Neither of these validation issues is easy. The first requires more effort than just modeling individual systems because each system can influence the others. Other workloads on any of the systems can impact the applications being modeled. One system can host more than one application being modeled at the enterprise level, which would require the submodels to have information about how to calculate the response times for all such applications. The second issue requires a level of understanding and documentation of both the systems and the applications that many organizations simply do not have available. Most applications do not collect information about execution paths, spawned transactions and client/server requests. In addition, many of the client/server systems today do

not collect enough, or sometime even any, detailed information about business transactions even if they do collect information about database or OLTP transactions.

Simalytic Model validation requires more time and effort than the validation of single system models. Until better cross-platform application measurement tools are in place, the most promising validation technique appears to be the "model the past to predict the present" approach. Simply stated, this technique is applied after all other issues are thought to be resolved. A model is built from measurement data collected from a different enterprise environment, generally at a lower transaction volume. Growth is then applied to the model to see if it predicts what can be currently measured. If it can, then there is confidence in the predictions of the future. If it cannot, the model must be revised. Until measurement data is improved for many of the systems used in today's client/server environments, many situations will require the modeled changes to be implemented and measured before they can be validated. Buzen discusses some of these issues of client/server model validation and determines that shortcomings in the ability to collect detailed measurement data does not preclude the usefulness of a client/server model. A client/server model can be validated at whatever level is supported by the measurement data and it is usable as long as it is only used to predict system performance at that same level (Confrey 1996). His example illustrates that a valid model can be built using system utilization when transaction measurement data is not available, but such a model cannot then be used to predict transaction response times because it cannot be validated at that level.

3.5.2 Verification/Validation Approach

Because of the issues discussed above (individual system models, relationships between systems and measurement data collection), the approach used to show the validity of Simalytic Modeling is a comparison of the results of the Simalytic technique to the results of the simulation technique rather than to a collection of measurement data. Verifying that the results produced by this approach are consistent with the results produced by pure simulation shows the approach to be valid for this type of problem. This approach is similar to those used in (Ahn and Kim 1994) and (Thomasian and Gargeya 1984). The comparisons use two strategies. The first is to compare the results of the mathematical formulae and the second is to compare the results of models of representative hypothetical systems implemented with actual modeling tools. This allows a set of assumptions to be used for all of the comparisons of mathematical formulae results and a different, but consistent, set of assumptions to be used for all of the comparisons of commercial simulation tool results. The series of test cases was selected to show that the modeling techniques produce equally acceptable enterprise model results within the bounds of reasonable usage with either a theoretical (mathematical) or a practical (commercial tool) implementation. By verifying the consistency of the results across what is assumed to be the entire range of practical situations, Simalytic Modeling is shown to be a valid technique for solving that set of problems.

3.5.2.1 Validation Approach Limitations

There are inherent limitations in the implementations of both a theoretical (mathematical) strategy and a practical (commercial tool) strategy. Both the tools used to implement each strategy, and the assumptions required to simplify each strategy enough to

allow a practical implementation, limit one's ability to generalize the results of the strategy to all practical client/server situations. However, the limitations of each strategy are addressed in the implementation of the other and it is therefore assumed that a consistency of results across both strategies provides sufficient evidence to allow for such a generalization.

Sections *3.4.1.1* through *3.4.1.7* use mathematical formulae to show how a system can be simplified or decomposed using the concepts of submodels and FESC's. The major difficulty with using these mathematical formulae is that sub-components must be solved in advance to allow the proper FESC's to be constructed. For example, when simplifying a fork/join subsystem, the response time of both servers must be known to allow the larger one to be selected. In addition, this selection is based on some simplification of the results, such as the average of all response times for each server. The reality of the situation may be that each server has the longer response time for some number of the transactions. The actual response time of the subsystem may be longer that the average of either server just as 5 (the average of 5 and 5) is larger than 4 (the average of 7 and 1) but the subsystem average is 6; calculated as (7+5)/2=6. (The larger of the first two numbers, 7, is averaged with the larger of the second two numbers, 5. The larger number is always selected as the response time for a fork/join subsystem). The working assumption is that the formulae represent the average of situations from a very large sample size and that the positive and negative differences will be equally distributed within the sample.

Probability routing systems have a similar limitation with the assumptions regarding the values to use for the probabilities. The actual probabilities observed over short periods of time may not be the same as the average for the entire sample period. Indeed,

the probabilities may start weighted toward one server and shift to be weighted toward the other. The average will not represent such dynamics in the system. Again, the working assumption is that the formulae represent the average of situations from a very large sample size and that the positive and negative differences will be equally distributed within the sample.

These examples show the limitations of using a purely mathematical approach and establish the need for the additional support of actual simulation models. These limitations are easily corrected when implementing the model with commercial simulation tools, which allow a level of control over the events within the simulation model comparable to that of a programming language. Unfortunately, the flexibility and complexity inherent in any programming language reduces the ability to provide a mathematical proof of correctness that is desired to establish the conceptual foundation upon which to build a methodology. Such a foundation is provided by the mathematical formulae even with the limitations discussed above. Section *3.5.3 Validation of the Mathematical Foundation* on page 86 analyzes the results of the mathematical formulae and section *4 Simalytic Model Development* on page 97 continues the analysis using commercial simulation tools.

3.5.2.2 Validation Approach Justification

The justification for using this approach to show the validity of Simalytic Modeling is based on the success of the approach in other works investigating hybrid models. They also show their techniques valid through the verification of consistent prediction compared to simulation. Ahn and Kim compared the results of their hybrid modeling methodology to the results of a discrete event simulation model and concluded "that there was little difference between the two approaches." (Ahn and Kim 1994, 5). They do not attempt to

prove that either model is an accurate representation of any real situations and assume that the simulation model was valid.

Thomasian and Gargeya use a similar approach to validate their hybrid technique for analysis of the performance of slotted ring multiprocessors (Thomasian and Gargeya 1984). They assume that the simulation model produces correct results and compare the results of their technique to the results of the simulation model. They also use bounding assumptions to limit the comparisons to a maximum of 32 processors, which is a reasonable upper bound given the scope of their inquiry.

3.5.2.3 Validation Approach Assumptions

The following assumptions were used for determining that the series of test cases selected are within the bounds of reasonable usage:

- The response time from any single node would be between 0.1 and 100 seconds. Values outside this range, although possible, are not common in modern transaction-based client/server systems. Transactions which complete in less than 0.1 seconds are normally considered trivial transactions and generally do not constitute a significant portion of a client/server workload because of the small amount of productive work that can be accomplished in such a short time. Transactions which complete in more than 100 seconds are normally considered pseudo-batch transactions and are generally avoided because of the business requirements in applications supporting end-users interactively. All service times are assumed to be exponentially distributed.
- The mean arrival rate of transactions for an individual server is assumed to be between an upper and lower limit set for each scenario based on the relation-

ship between the service time and the interarrival time. The upper limit is when the mean interarrival time approaches the service time. Therefore, the arrival rate upper limit is set by calculating the inverse of the service time times 0.99 to avoid a divide-by-zero condition. The lower limit is when the mean interarrival time is enough below the service time to cause minimal queuing and a value of ten times the service time is assumed to meet this condition. Therefore, the arrival rate lower limit is set by calculating the inverse of ten times the service time. These bounds cover the range from 10% to 99% server utilization, which is an acceptable range for a capacity planning model.

- The arrival rate bounds for the entire system in a given scenario should be set to a reasonable range. The lower bound should be equal to the lowest bound for any server in the system because anything lower would not produce any significant queuing. The upper bound should be set no higher than the lowest high bound for any server in the system because overall model results will not be usable once a server saturates.
- Simalytic Model results within 10% of the results of the simulation model will be considered equivalent. The generally accepted rule-of-thumb for capacity planning modeling is that a model is a valid predictor of the capacity requirements of a system if its results are within 10% to 20% of the measured data. Given this reasoning, a worst case of worst cases could produce a model between 72% and 132% of the measured data, which would still be considered a usable, although not a good, capacity planning model. Even at the 20% deviations level, a capacity planning model would be considered a high level ap-

proximation, such as server load, and not used for more detailed assessments, such as transactions service time analysis. Additional modeling effort would be required, regardless of the technique used, if there was a business need for greater accuracy.

- The upper limit for the number of servers used to implement a single client/server application is assumed to be eight servers. This assumption is based solely on the author's personal experience and opinion of what are practical client/server application implementations. Advances in client/server application design techniques that are not foreseeable today will allow larger numbers of servers to be used. Then, as with any capacity planning technique, additional testing and verification will be required. This assumption is made only to set the scope of test cases and does not limit the concepts or theories underlying Simalytic Modeling.

3.5.3 Validation of the Mathematical Foundation

The mathematical foundation discussed in section *3.3 Foundation* on page 61 is shown to be valid by comparing the results of each of the three formulae for each of the scenarios discussed in that section. The formulae are implemented in MathCAD (MathSoft 1995) and the results of each set plotted together to show the degree of convergence. First, however, the scope of this analysis can be reduced by some simple investigations into the relationship between arrival rate and service time. To examine this relationship further, a visual representation was created using three-dimensional surface plots.

Figure 19 Queuing Theory Single Server Formula Plot shows a surface area plot of the results of the queuing theory formula from *Figure 12 Single Server System* on page 70 for the range of values discussed in the assumptions in section *3.5.2 Verification/Validation Approach*. Although the assumptions seem quite reasonable, it is readily apparent from *Figure 19* that the majority of arrival rate/service time combinations exceed the 100 second response time limit and, in fact, represent saturated servers. Further investigation shows that there is actually a relatively small area where the response time is acceptable.

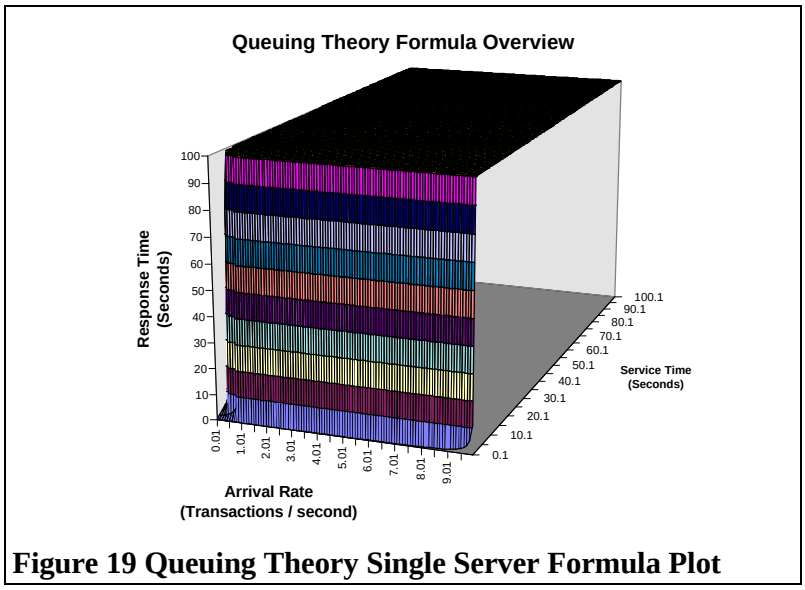


Figure 19 Queuing Theory Single Server Formula Plot

Figure 20 Queuing Theory Partial Results shows a close-up view of this area at a higher granularity. The arrival rate is in the range of 0.01 to 0.5 transactions per second and the service time is in the range of 0.1 to 3.0 seconds. Rather than attempt to model arrival rate / service time combinations that are known to saturate the server, the set

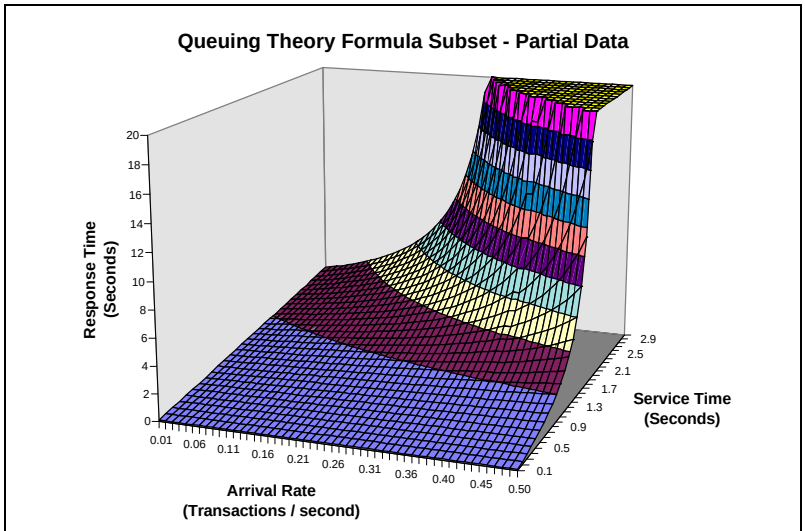
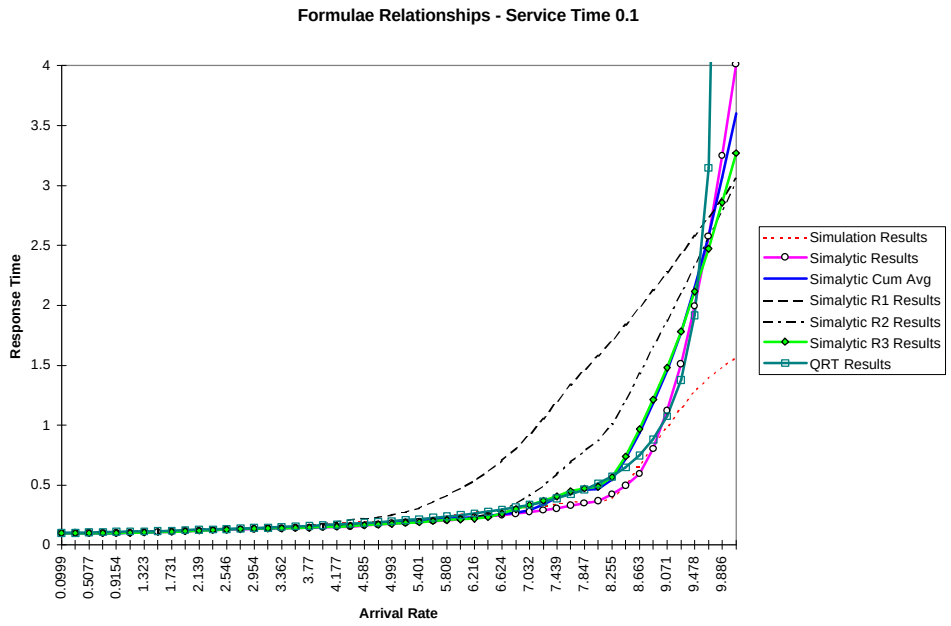


Figure 20 Queuing Theory Partial Results

of models for each service time used arrival rates from slightly greater than zero to slightly less than the inverse of the service time (because a model is known to be saturated when the average interarrival time is equal to or greater than the service time). The service times modeled were 0.1 to 5.0 in increments of 0.5 and 25.0, 50.0, 75.0 and 99.0. These surface plots show a smooth transition between data points as either the arrival rate or service time increases, which indicates that a reasonable sampling of arrival rate / service time combinations is appropriate. The MathCAD worksheet used to generate the data for these plots is listed in *7.4 Appendix D: MathCAD Queuing Formulae* on page 178.

The objective in creating the models using a series of service times is to show a consistent relationship between the different modeling techniques (simulation, queuing theory and Simalytic Modeling) for each of the service times. The surface plots show that there is a consistent relationship between service times for each of the techniques and that it is reasonable to assume that service times not modeled would show similar behavior.

Figure 21 Example Formulae Comparison shows two examples of the comparison of the three formulae, the top one at a low service time of 0.1 seconds and the bottom one at a high service time of 2.5 seconds (low and high are relative to the area shown in *Figure 20*). A complete set of charts, showing service times from 0.1 to 99.0 can be



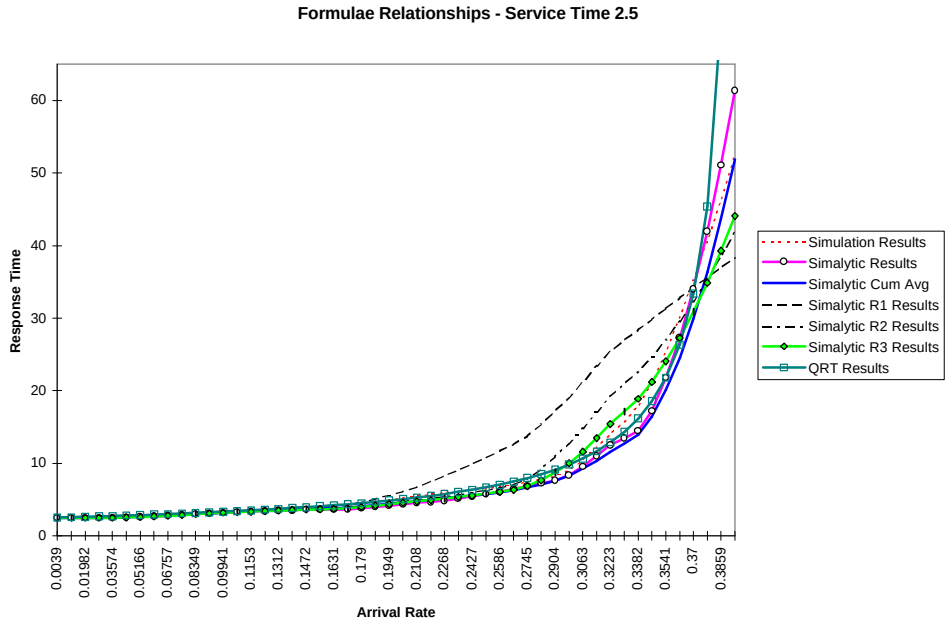


Figure 21 Example Formulae Comparison

found in *7.5 Appendix E: MathCAD Formulae Results Charts* on page *180*, including a detailed description of the chart elements and how the charts were created. The charts in *Figure 21* are reduced versions of charts from section *7.4* and are reproduced here only to provide easy reference to the shape of the curves, not the actual values and labels. The charts in *Figure 21* are representative of the results seen in all of the charts. There are seven lines in both charts defined as:

Simulation Results: the results from the simulation model.

QRT Results: the results from the queuing theory model.

Simalytic Results: the results from the Simalytic Function response time table where the function is called once for each arrival rate with the average of all interarrvial times. This is the theoretical best fit between the Simalytic Function and the queuing theory formula because the interarrival times are both the same average.

Simalytic Cum Avg: the results from a Simalytic Model using a Simalytic Function that calculates a cumulative average of all prior interarrival times each time it is called.

Simalytic R1 Results: the results from a Simalytic Model using a Simalytic Function that calculates a rolling average of the interarrival times each time it is called using the last 1.5% of the interarrival times. This percentage, and the percentages used below in the description of the **Simalytic R2 Results** and the **Simalytic R3 Results**, is calculated as the number of interarrival times included in the rolling average divided by the total number in the model interval.

Simalytic R2 Results: the results from a Simalytic Model using a Simalytic Function that calculates a rolling average of the interarrival times each time it is called using the last 5% of the interarrival times.

Simalytic R3 Results: the results from a Simalytic Model using a Simalytic Function that calculates a rolling average of the interarrival times each time it is called using the last 10% of the interarrival times.

The two lines that do not merge with the other lines are the **Simalytic R1 Results**

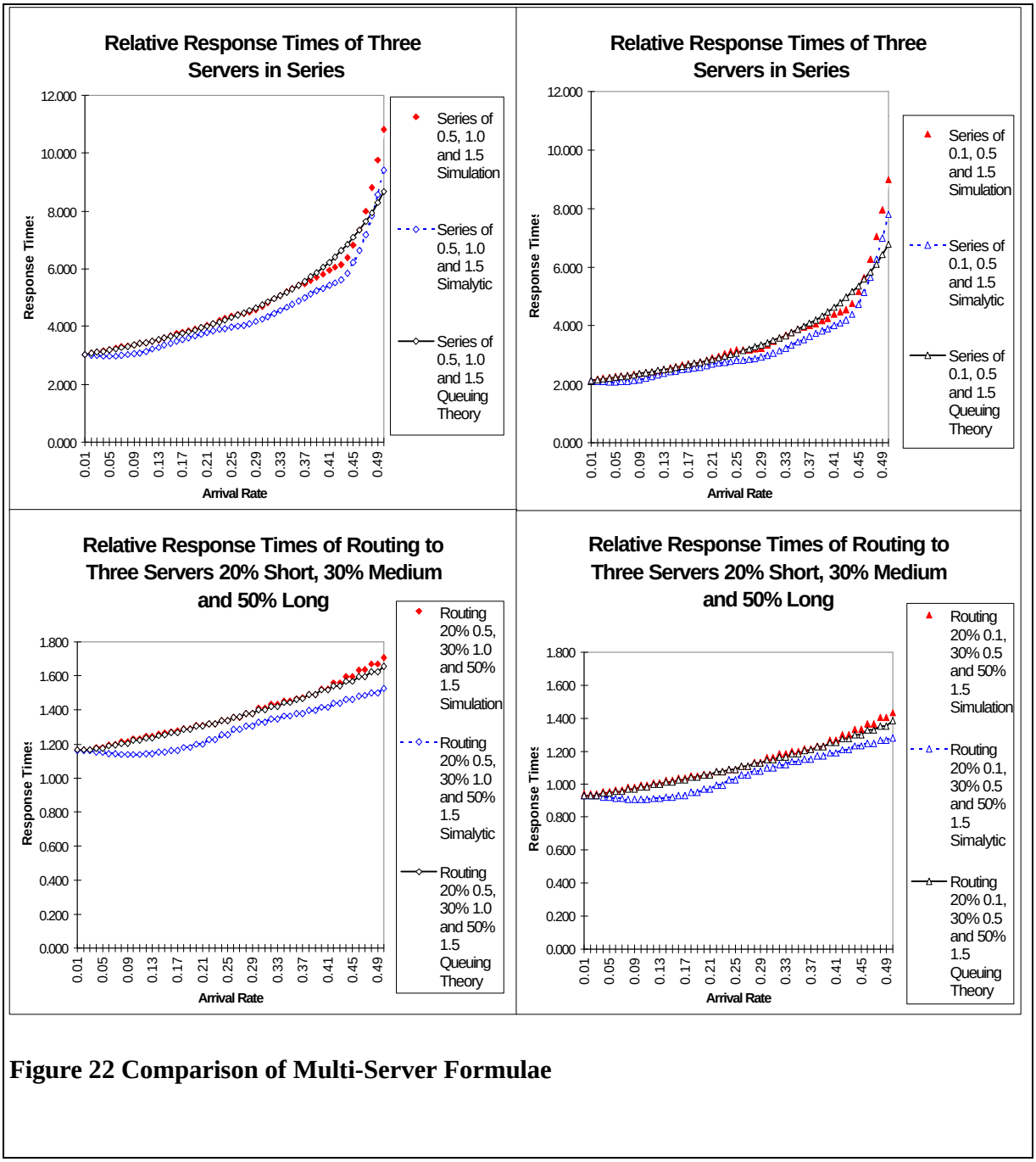


Figure 22 Comparison of Multi-Server Formulae

and the **Simalytic R2 Results**. The other implementations of the Simalytic Function produce results that consistently track the simulation results. The **R1** and the **R2** implementations did not produce acceptable results and are not used in any of the remaining analysis. Please refer to section *7.5 Appendix E: MathCAD Formulae Results Charts* on page *180* for a more detailed explanation of how these values were calculated.

Figure 22 Comparison of Multi-Server Formulae on page 91 shows four examples of the resulting response times when servers are combined as discussed in sections *3.4.1.1* through *3.4.1.7*. A complete set of charts showing the 32 scenarios from *Table 1 Mathematical Formulae Results* on page 93, including both series and routing systems, can be found in *7.5 Appendix E: MathCAD Formulae Results Charts* on page 180. The appendix includes a detailed description of the chart elements and how the charts were created. The charts in *Figure 22* are simplified versions of charts from section *7.4* and are reproduced here only to provide easy reference to the shape of the curves, not the actual values and labels. The important information in this figure is that each chart shows that the Simalytic technique results track both the simulation results and the queuing theory results for all four of the three server scenarios (two series and two routing).

Table 1 Mathematical Formulae Results lists three different views of the relationships for each scenario: percent outside bounds, percentile, and percent difference. These percentage difference values are calculated by: (simulation results - Simalytic results) / simulation results.

The **Percent Outside Bounds** column shows the percentage of the results for each scenario that are greater than ±10% different. Using the top two scenarios in the table as examples, the first shows 0% outside of the bounds and the second shows 18% outside of the bounds. The minimum and maximum difference percents for the first scenario are only 1% and 8% respectively, both within ±10% different, thus 0% outside of the bounds. The minimum and maximum difference percents for the second sce-

Test Scenario	Percent Outside Bounds	Percentile		% Difference	
		10th	90th	Min	Max
Single Server of 0.1	0%	2%	4%	1%	8%
Single Server of 0.5	18%	2%	11%	-2%	12%
Single Server of 1.0	28%	2%	11%	-1%	12%
Single Server of 1.5	44%	5%	13%	1%	14%
Series of 0.1 and 0.5	8%	2%	10%	-1%	10%
Series of 0.5 and 1.0	0%	4%	10%	-1%	10%
Series of 0.1 and 1.5	38%	5%	13%	1%	14%
Series of 0.5 and 1.5	32%	6%	12%	1%	13%
Series of 0.1, 0.5 and 1.0	0%	4%	9%	-1%	10%
Series of 0.5, 1.0 and 1.5	18%	6%	11%	0%	13%
Series of 0.1, 0.5 and 1.5	30%	6%	11%	1%	13%
Series of 0.1, 0.5, 1.0 and 1.5	18%	6%	10%	0%	13%
Routing 80% 0.1 and 20% 1.0	0%	0%	7%	0%	7%
Routing 80% 0.5 and 20% 1.5	26%	1%	11%	0%	12%
Routing 80% 0.1 and 20% 1.5	12%	1%	10%	1%	10%
Routing 80% 0.1 and 20% 0.5	0%	0%	4%	0%	4%
Routing 20% 0.1 and 80% 1.0	10%	2%	10%	-1%	11%
Routing 20% 0.5 and 80% 1.5	40%	5%	12%	1%	13%
Routing 20% 0.1 and 80% 1.5	44%	5%	13%	1%	13%
Routing 20% 0.1 and 80% 0.5	20%	1%	11%	-1%	11%
Routing 50% 0.1 and 50% 1.0	12%	1%	10%	-1%	11%
Routing 50% 0.5 and 50% 1.5	6%	5%	10%	1%	11%
Routing 50% 0.1 and 50% 1.5	22%	5%	11%	1%	11%
Routing 50% 0.1 and 50% 0.5	6%	0%	10%	-1%	10%
Routing 70% 0.1, 20% 0.5 and 10% 1.0	0%	-1%	4%	-1%	5%
Routing 70% 0.5, 20% 1.0 and 10% 1.5	0%	0%	9%	-1%	10%
Routing 70% 0.1, 20% 0.5 and 10% 1.5	0%	1%	6%	1%	7%
Routing 20% 0.1, 30% 0.5 and 50% 1.0	0%	1%	9%	-1%	10%
Routing 20% 0.5, 30% 1.0 and 50% 1.5	6%	4%	9%	0%	11%
Routing 20% 0.1, 30% 0.5 and 50% 1.5	2%	4%	10%	1%	11%
Routing 34% 0.1, 33% 0.5 and 33% 1.0	0%	-1%	9%	-1%	9%
Routing 34% 0.5, 33% 1.0 and 33% 1.5	0%	0%	9%	0%	9%
Routing 34% 0.1, 33% 0.5 and 33% 1.5	0%	1%	9%	1%	9%
Routing 70% 0.1, 15% 0.5, 10% 1.0 and 5% 1.5	0%	0%	3%	0%	4%
Routing 10% 0.1, 20% 0.5, 25% 1.0 and 45% 1.5	0%	2%	9%	0%	9%
Routing 25% 0.1, 25% 0.5, 25% 1.0 and 25% 1.5	0%	0%	9%	0%	9%

Table 1 Mathematical Formulae Results

nario, however, are -2% and 12% respectively. In this case, the 18% outside of the bounds shows how many of the data points in the scenario are between 10% different and 12% different. The closer this value is to zero the better the correlation between the simulation and the Simalytic results.

The **Percentile (90th and 10th)** columns show the 90th percentile percentage difference for each scenario (90% of the differences are less than or equal to the percentage shown) and the 10th percentile percentage difference for each scenario (90% of the differences are greater than the percentage shown).

Finally, the columns labeled **% Difference** shows the maximum and minimum differences for each scenario. Both *Table 1* and *Figure 22* clearly show that the Simalytic function tracks closely to the simulation results. The large **Max % Difference** and the small **Min % Difference** show that the Simalytic function is also consistently under predicting the system response time, which is due to some of the issues discussed in section *3.5.2.1 Validation Approach Limitations* on page 81. It can be seen from *Table 1* that even when there is a large number of results outside the boundaries (±10% different), both the **90th Percentile** and the **Max % Difference** are relatively close to the upper boundary (+10% different). Further analysis of these results shows that the Simalytic function returns clearly delineated (i.e. precise) predictions although they are not accurate in terms of exactly matching the simulation results. However, the results from the Simalytic function can be adjusted by a simple constant multiplier (1.05 for this specific data, determined by experimentation) to greatly increase the accuracy. This adjustment, also called model calibration, is referred to as *percent difference*, and is one of several commonly used calibration techniques (Menascé, Almeida, and Dowdy 1994 309). The significance of this

factor is not the value used, but that the same value can be applied for all of the models. Not only does this improve the correlation between the two for this investigation, but it also shows that the prediction capabilities of the technique in real-world situations can be improved to the required level of accuracy. Because the Simalytic function provides consistently precise results, that is, results within a narrow range for the given input values, the accuracy can be improved by refining the implementation of the function during model calibration using techniques such as the simple constant multiplier discussed above.

Test Scenario After Adjustment	Percent Outside Bounds	Percentile		% Difference	
		10th	90th	Min	Max
Routing 25% 0.1, 25% 0.5, 25% 1.0 and 25% 1.5	0%	-5%	4%	-5%	4%
Results before adjustment	0%	0%	9%	0%	9%
Routing 70% 0.1, 20% 0.5 and 10% 1.5	0%	-4%	2%	-4%	3%
Results before adjustment	0%	1%	6%	1%	7%
Routing 20% 0.1 and 80% 1.5	0%	0%	8%	-4%	9%
Results before adjustment	44%	5%	13%	1%	13%
Routing 80% 0.1 and 20% 0.5	0%	-5%	-1%	-5%	-1%
Results before adjustment	0%	0%	4%	0%	4%

Table 2 Mathematical Formulae Results After Adjustment

A sample of scenario results calculated after such an adjustment is shown in *Table 2 Mathematical Formulae Results After Adjustment*. Only four of the scenarios were adjusted to show that the adjustment is reasonable for a variety of the scenarios. Those selected are the worst and the best differences and two spaced between those extremes. These results after adjustment are much more centered around zero. The **Routing 20% 0.1 and 80% 1.5** scenario appears to be a greater discrepancy than it really is because the **Percent Outside Bounds** value is very high (44%). However, the percent drops quickly as the bounds are increased because the results are clustered very close together just greater than the 10% delta bounds. At an 11% delta the value is 26%, at 12% delta the value is 12%, at 13% delta the value is only

8% and it is zero at 14% delta. Thus the multiplier has the effect of moving an already precise and well contained cluster of results to more accurately match the simulation results. Although such a multiplier could easily be incorporated into the Simalytic function implemented in MathCAD, the commercial tools used for the Simalytic Modeling framework provide much greater visibility into the dynamics of the system being modeled. These tools allow logical analysis of the state of the system at every transition and can therefore adjust the response time at any node based on additional information such as current queue length and server utilization. At this point, the speculation is that the under prediction is a result of the granularity of the steps used in the Simalytic Function (this was verified using a simulation tool as discussed in section *4.2.5.5 Simalytic Model Example* on page 121 and in *Figure 31 Simalytic Function Comparison* on page 123.)

3.5.4 Validation of the Simulation Framework

A similar approach is also used to validate the Simalytic Modeling Methodology using commercial simulation tools for the simulation framework. A number of scenarios are implemented using the commercial tools and the results are analyzed in a similar manner in sections *4 Simalytic Model Development* on page 97 and *5 Investigations into Simalytic Modeling* on page 127.