Proceedings of the 2001 Winter Simulation Conference
B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, eds.

A CAPACITY PLANNING TOOL FOR THE TUXEDO MIDDLEWARE USED IN TRANSACTION PROCESSING SYSTEMS

Tayfur Altiok
Wei Xiong

Department of Industrial Engineering
Rutgers University
Piscataway, NJ 08854, U.S.A.

Mesut Gunduc

Staff Engineer
BEA Systems Inc.
Liberty Corner, NJ 07938, U.S.A.

ABSTRACT

In this paper, we present a brief overview of Tuxedo middleware system (BEA Systems) and introduce an object-oriented computer simulation template developed for the purpose of capacity planning and performance analysis of Tuxedo application environments. Arena/Siman (Rockwell Software) simulation software is chosen and a *CP_Tool* template specific to Tuxedo environment is developed. The template consists of a number of modules representing client and server nodes, network nodes and other critical components of the system. Any Tuxedo environment can be created using the modules from the *CP_Tool*. The paper discusses the tool and its capabilities.

1 INTRODUCTION

The Tuxedo System provides a three-tier Client/Server application programming framework that enables the construction, execution, and the administration of high performance distributed applications. The Tuxedo system renders a scalable, standards-based software architecture supporting a variety of hardware platforms, operating systems and networks (Andrade et al 1996). Over the past decade, the changes in business needs and environments have given a tremendous boost to the client/server technology. Today's businesses need responsive, flexible, integrated and comprehensive applications to support the complete range of business processes. Clearly, demand and user expectations on the system bring issues of capacity, availability and performance. As new applications are added or additional load is experienced due to increased demand, response times tend to increase as a result of system bottlenecks. The current state in the business world necessitate the use of performance analysis tools to answer questions of "What is the impact of a 20% increase in customer transactions on the response time of our system?", or "If we were to allocate service i to server node k, how much would the response time be reduced?" or "What good would it do to move an RM operation to a remote. node?" These questions are among the many other similar questions one would ask while designing a new system or while managing an existing system. Capacity planning is the activity that responds to these questions. More formally, capacity planning is the process of predicting system performance for a given set of resource requirements and in turn using this information to decide on the resources to achieve a desired level of performance. In a client server system, probably the most important measure of system performance is the response time. Capacity planning effort predicts the response time and unearths the bottlenecks in the system (Menace and Almeida 1998).

Also, server performance is critical for E-commerce applications, and is becoming even more important as Web protocols are applied to performance-sensitive applications (Comer 1997). For instance, electronic imaging systems require servers to perform computation-intensive image filtering operations. Likewise, database applications based on Web protocols (such as Alta Vista Search by Digital) require complex queries that usually generate a higher number of disk I/O operations. As the Web grows with its current pace, it will be inevitable that every new Web application will have to be designed with the help of a capacity-planning tool.

In this paper, we introduce a tool, *CP_Tool*, designed to perform capacity planning in Tuxedo application environments. Tuxedo is a middleware system developed by BEA Systems, Inc., and currently being used as the middleware in transaction processing type client/server systems worldwide (Andrade et al 1996). *CP_Tool* is an object-oriented software tool developed based on a combined approach using simulation and queueing theory. Client/Server systems involve a number of complex queueing problems both on the client side and the server side, that are not yet mathematically studied. In this paper, we will describe some of these queueing problems and incorporate them into the *CP_Tool*.

2 CLIENT/SERVER ENVIRONMENT

In a typical client/server environment, there are a number of client nodes, a number of server nodes (that usually form a cluster), a network, possibly a database server (resource manager) and a number of disks local to any of the server nodes. In transaction processing applications, transactions are service requests that are initiated by the client nodes, processed by the server nodes and eventually are returned to the initiating client nodes. Middleware, known as the TP monitor in transaction processing parlance (Orfali 1996) is the software that provides a unified view of the server cluster to the clients (transparency). First, it coordinates the co-existence of server nodes in the cluster. Among many others services, middleware provides location and migration transparency for the application software running on the system. Second, it gets control of every transaction submitted to the system and guarantees its proper completion and return to the initiating client. The TP monitor technology is promising to be the most effective solution for client/server architectures for systems with thousands of users.

A client/server application is primarily a network of message queues. Transactions experience delays in client node queues, in network queues, and in the various server node queues. Response time of a transaction starts at the moment it is initiated at the client node and includes the time until its response is processed back in the client node. This time includes all time segments that are delays in process queues, processing times and the I/O related times (Cady and Howarth 1990). Due to inherent complexities in maintaining a middleware on top of the operating system at each node, it is quite difficult to identify every time segment on the path of a transaction in the system. No matter how detailed it may be, every approach to describe the sample path of a transaction turns out to be an approximate one. In this paper, we report on a detailed analysis of server nodes in Tuxedo application environments. Experience shows that the time spent on the client side and in the network are significantly lesser that the times spent on the server side. Therefore, most of our effort in the development of *CP_Tool* concentrated on the server side, even though *CP_Tool* models the entire client server system.

3 TUXEDO ENVIRONMENT

Tuxedo is a message-based middleware for client-server systems. Client and server processes communicate by sending messages to each other. Tuxedo offers a rich set of communication paradigms including request/reply, connection-oriented events and persistent-storage based message queues. Among them, request/reply messaging is the most common. With this type of communication paradigm, application clients issue service requests against some services offered by the server processes hosted on Tuxedo domain server nodes. Once a request is serviced, a reply message is constructed by the server (serving the request) and is sent back to the client who initiated the request. In the meantime (while the request is being processed), the client is blocked, waiting for a reply. This is the synchronous version of request/reply communication type. There is also an asynchronous version of it in which the client process sends a service request and does not get blocked while waiting for a reply, and thus proceeds to do other things, i.e. issue other service requests.

In a typical Tuxedo application domain, client nodes host Tuxedo client processes that generate requests (i.e. tpcall()) against services via Tuxedo ATMI, as shown in Figure 1. Services are provided by the server processes residing on the server nodes. While processing a request, a server process may communicate with a Resource Manager (RM), i.e. a RDBMS, or another Tuxedo domain. When processing a request is completed, the server process constructs a reply message and sends it to the client via an appropriate Tuxedo ATMI, i.e. tpreturn().

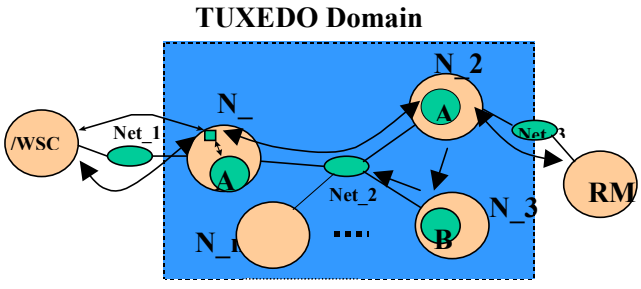

Figure 1: A Typical Tuxedo Application Environment

It is also possible that the client may decide not to receive a reply, in which case no reply message is generated. Client processes may in fact reside on one or more of the server nodes together with Tuxedo administrative and application server processes. The scenario depicted in Figure 1 indicates that a client residing at the /WSC node requests the execution of Service A. This request arrives at node N_1 via network NET_1. It is then sent, via NET_2, to Node N_2 where Service A is offered. While being processed at N_2, the server process providing Service A requests data base operations at a remote RM node via NET_3. Eventually, processing of the request is completed and the server process produces a tpreturn(). This reply goes back to N_1 and finally to /WSC via networks NET_2 and NET_1. The entire time the request for Service A spends in the system includes waiting times at the local client and server nodes, processing time at these nodes, time spent in the RM node and the network transmission times. While designing Tuxedo application domains or while measuring the performance of existing systems, the most critical performance measure to look at is the response time that encapsulates all these time delays requests ex-

perience. Naturally, users want the system to be responsive with short response times. Clearly, there are cost considerations also, thus making the design problem a challenge. Other critical measures are the waiting times and utilization at integral components of the system, i.e. the server processes.

All the nodes in a Tuxedo application environment are interconnected via various LAN/WAN networks, FDDI, Ethernet, ATM, token ring, etc. Tuxedo is transparent to the underlying network, provided that it supports TCP/IP protocol stack. Tuxedo also provides server process location transparency to clients. In other words, client processes are not aware of the location of the server processes that provide a specific service. Each server node is associated with a set of client nodes in such a way that it is responsible for selecting the server process (anywhere in the Tuxedo domain) for the requests coming from these client nodes. Server node constructs a message containing the request and ships it to the queue of the server process it selects. Inter-server nodes communication is handled by a group of Tuxedo administrative processes, called BRIDGE processes, one per server node. Given an incoming request, Tuxedo's TP Monitor selects the target server process by the following algorithm: A local idle server process is selected if it provides the service requested. Otherwise, the server process with the smallest load factor (user assigned), anywhere in the domain, providing the requested service is selected. Below, we discuss how transactions, client, network and server nodes are viewed in Tuxedo environment.

3.1 Transactions

Transactions are identified through their attributes. These attributes include transaction type (e.g., request/reply or request only), service requested, message sizes for requests as well as replies and service priority. Note that there may be several transactions requesting the same service but with different message sizes.

3.2 Client Node

A client node is a computer that may host a number of client processes each with possibly several instances as shown in Figure 2. Transactions (or service requests) arrive at client process queues according to their types and arrival frequencies.

Active client processes reside either in the CPU queue or on the CPU initiating the transaction. For a transaction of request/reply type, the response time starts at the moment the request arrives at a client process queue, and it is recorded when processing of the reply is complete at the same client process.

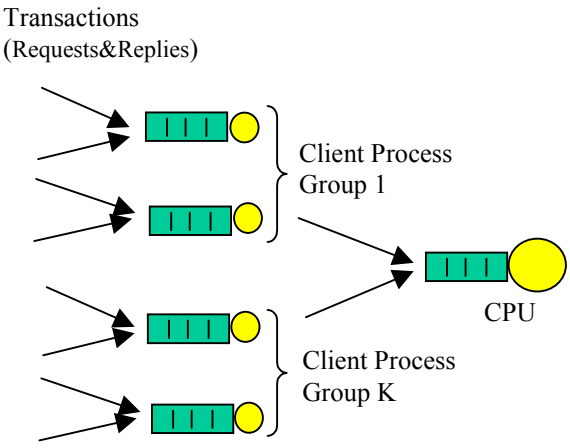

Figure 2: /WSC node traffic model

3.3 Network

The network node is viewed as a single server queue as shown in Figure 3, where transactions (requests and replies) wait upon their arrival if the network is busy, and receive service on a FIFO basis. Service in the network node is simply the transmission of the message. Transmission time depends on the message size. Naturally, larger messages take longer to transmit. The transmission rate of a network is its bandwith capacity. In an existing system, the bandwidth capacity can be obtained using observed data for network utilization and the throughput. In new systems where data is lacking, the projected bandwidth capacity can be obtained using the published value for network capacity and the anticipated message transfer efficiency (proportion of time the network will be available for message transmission).

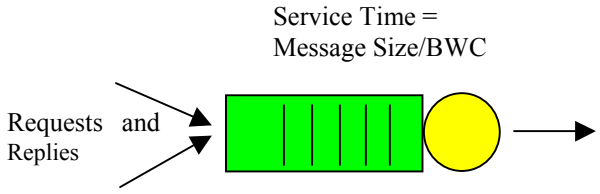

Figure 3: Network node traffic model

3.4 Server Node

A Tuxedo server node is viewed as a pool of highly interdependent resources of finite capacity, i.e. CPUs, memory, or disks, as shown in Figure 4. There are possibly a number of CPUs executing server processes providing services for transactions, work station handlers getting hold of entering transactions, bulletin board used to decide to which server process to send the entering transactions, bridge process that manages the communication between server

nodes in the cluster, and a number of other processes in order to process Tuxedo transactions.

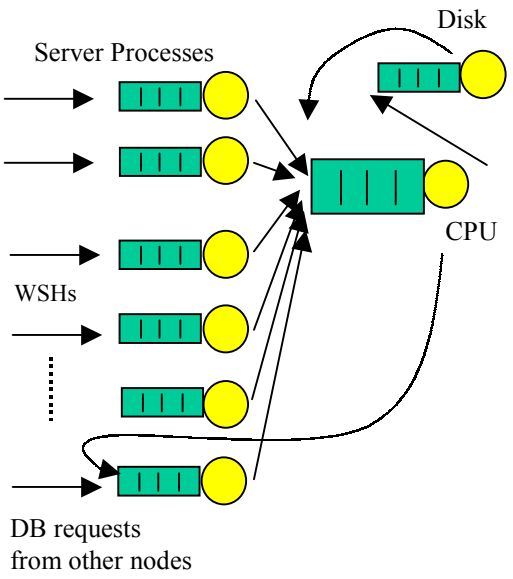

Figure 4: Traffic model of a Tuxedo server node

At the higher level, one can visualize the server node environment as a pool of server/client processes with the associated queues of request/reply messages awaiting to be served. However, a server/client process is not a real resource. The real resource is the underlying processor, i.e. the CPU which executes the instructions. So, at the physical level, there is a processor with a queue of runnable server/client processes awaiting to be processed as depicted in Figure 4.

Service time of a given transaction is not static as it depends on

- the associated server's locality of reference characteristics,
- hit ratios,
- speeds and sizes of the underlying CPU cache and RAM,
- speed of the underlying disk subsystem,
- current load on the CPU.

The delay time of a transaction at a server process queue is only partly due to its actual processing time. As indicated in Figure 4, this delay also includes the following delays (Hennesy and Patterson 1996):

- The queue delay incurred by the request/reply as it waits in the queue for the server/client process
- The dispatch time incurred while the server/client process is waiting for the CPU
- The processing time of the server/client process including time spent in resources e.g. disk, RM, etc.
- The contention delay for the processor with the underlying operating system as it handles interrupts and other system activities, i.e. context switches, preemption by higher priority processes, IPC
- File I/O is probably the most important delay factor in majority of the commercial application environments.

Many of the production Tuxedo systems are database intensive, that is, the service requests make heavy use of databases during processing. Databases usually reside on one or more disk devices and are typically managed by a database manager (a set of cooperating processes responsible for efficient access to data, its consistency and its recovery, among other things). Database managers in turn interact with the file system component of the underlying operating system layer for basic file I/O operations (e.g., open, close, read and write.) After all, a database is a mere set of files residing on a group of disks. Due to slow disk speed relative to the system's RAM, most operating systems today provide a buffer cache component that functions much like the CPU cache. The purpose is to keep the most recently used disk data in memory with the anticipation that it will be retrieved again, while in cache. They are most effective when files are accessed randomly. A given service request typically generates a sequence of file I/O operations which in turn result (with a buffer cache-hit probability) in a sequence of disk I/O operations against the files in a database. The following factors should be taken into account in predicting the time involved in a typical file I/O operation:

- disk rotational time,
- average disk access time (average seek time + average rotational latency),
- number of disk read operations required,
- number of disk write operations required,
- average buffer cache hit ratio,
- file system service time,
- file organization on disk,
- in case of write, whether it is write-through or cached,

Next, we present a brief overview of the *CP_Tool*.

4 *CP_Tool*

CP_Tool is a simulation model built over Arena/Siman simulation software as a template specific for Tuxedo application environment. Arena is a simulation modeling tool based on Siman simulation language. It abstracts basic Siman language constructs as basic building blocks for

simulation modeling of real applications. These basic building blocks are called modules and relevant modules are put together to create templates tailored for specific applications. It offers an object-oriented approach for building simulation models. Modules are essentially the objects of the real application being modeled, and they have logic associated with them. The basic objects which move through the system (from one module to another) are called entities that correspond to transactions being issued by clients. The logic of a module defines the steps of operation an arrival entity is subjected to.

CP_Tool consists of 8 modules, namely Services, Transaction, /WSC, Network, Server, Disk, RM Node, and Simulate, as shown in Figure 5, along with their user views. The module names clearly describe the association between the module and the component each module is modeling. These objects are connected to each other according to the scenario being modeled and simulated using Arena's simulation engine to produce performance measures such as response times, CPU, server process and disk utilizations and queue delays. These are essential measures to point out bottlenecks and reasons for long delays and hold ups in the system.

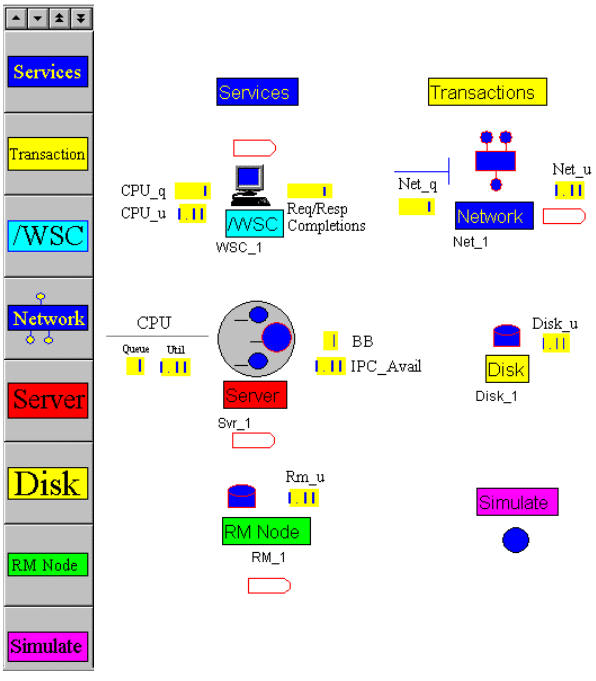

Figure 5: *CP_Tool* template consisting of modules and their user views

Below, we briefly review the dialog boxes of some of the modules. Transaction module lets the user define transactions (service requests) through their attributes. Dialog boxes associated with the Transaction module are given in Figure 6.

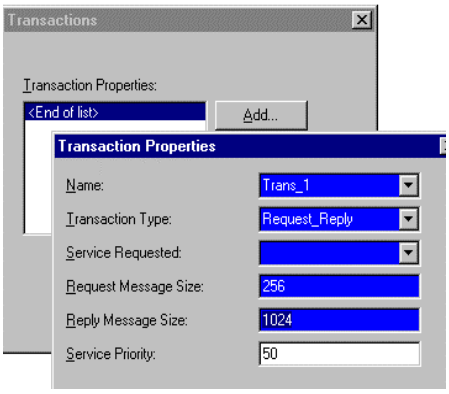

Figure 6: Dialog boxes associated with the Transaction module

Client nodes have dialog boxes to describe client processes and the transactions originating from these client processes. Client process properties include the server node that it is associated with and the network node in between. It also includes instance number to indicate the number identical processes of this type (forming a client process group) as shown in Figure 7.

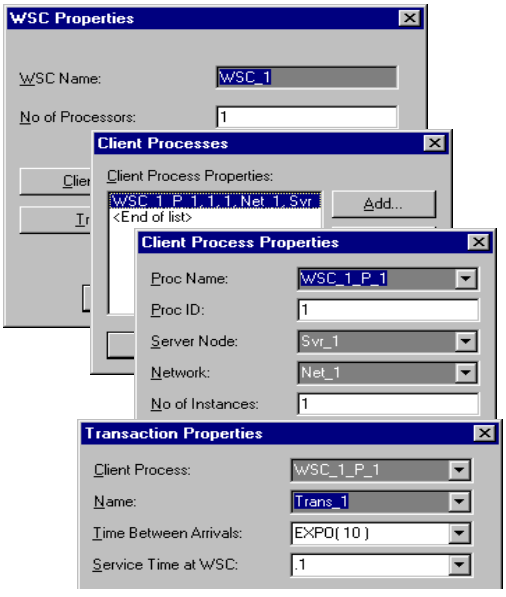

Figure 7: Dialog boxes associated with the client node

WSC Properties also includes attributes of the transactions originating from the current WSC node.

Server node dialog box allows the user to define server node properties, the attributes of the server processes, services, workstation handlers and native clients, as shown in Figure 8. It also allows the user to input disk IO operation details. Server Node Properties include parameters such as IPC message pool size, bulletin board lock hold time,

memory-page size, per page memory copy time, CPU time quantum and context switching time, among others.

Server processes and Services let the user to introduce each server process and each service with service time and tpforward() information. WS Handler Properties allows the user to introduce workstation handlers and associated service time information. Native Client Properties require information that is quite similar to client node attributes. NonRM Disk IO Operations and database RM Operations provide further dialog boxes for the user to input detailed information on file I/O and data base operations. For each physical disk I/O operation, size (size of data to read or to write), no of instances and target disk information is requested from the user. Similar information is provided by the user for each RM operation and the resulting physical disk I/O operation.

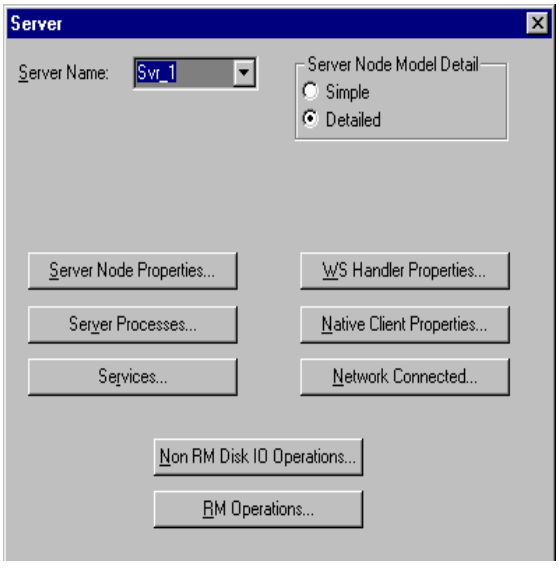

Figure 8: Dialog box associated with the client nodeServer node

CP_Tool provides the user with two approaches for modeling server nodes. The Simple approach asks for the elapsed time as the processing time. Elapsed time includes all the time required to process a service request. It has the CPU time, disk time and all the delays in the associated queues. Clearly this simplifies the use of *CP_Tool*. The Detailed approach asks for a more detailed input data on CPU time I/O times, sizes and frequencies. The advantage of the Detailed approach is that it provides disk information whereas the Simple approach bundles all the CPU and disk information into server process utilization.

Simulate is the module to declare the replication parameters such as run length, number of replications and the warm-up period. Each module has a set of dialog boxes to input values of the associated system attributes. Using these modules, one can create a simulation model for any given Tuxedo application topology. During a simulation run, *CP_Tool* replicates the operation of the system, that is creation of service requests, sending them to server nodes, processing them, queueing of transactions and sending replies back to clients. In this process, it collects observations on how long each transaction spends at each stage of its response cycle and the entire response time. Observing idle/busy periods of each resource in the system, *CP_Tool* also produces resource statistics which are instrumental in bottleneck analysis.

REFERENCES

Andrade, J., M. Carges, T. Dwyer, and S. Felts. 1996. *The Tuxedo System*, Addison Wesley, MA,.

Cady, J. and B. Howarth. 1990. *Computer System Performance Management and Capacity Planning*, Prentice Hall, NJ.

Comer, D.E. 1997. *Computer Networks and Internets*, Prentice Hall, NJ.

Hennesy, J. and D. Patterson. 1996. *Computer Architecture, A Quantitative Approach*, MK Publishers, CA.

Menace, D. and V. Almeida. 1998. *Web Performance, Metrics, Models, and Methods*, Prentice Hall, NJ.

Orfali, R. D. Harkey, and J. Edwards. 1996. *The Essential C/S Survival Guide*, Wiley, NY.

AUTHOR BIOGRAPHIES

TAYFUR ALTIOK is a Professor of Industrial Engineering at Rutgers University. He received his Ph.D. from NCSU in 1982. His research interests are in capacity planning and performance analysis of transaction processing systems, manufacturing systems and bulk port operations, queueing networks and simulation. He has co-authored the recent book *Simulation Modeling and Analysis using Arena*. His email address is <altiok@rci.rutgers.edu>.

WEI XIONG is a Ph.D. student in the Department of Industrial Engineering At Rutgers University. He graduated from Tianjin University in China in BBB. His research deals with performance analysis of middleware in distributed computing environments.

MESUT GUNDUC is a staff engineer at BEA Systems, Inc. in Liberty Corner, NJ. He received his M.Sc. in Computer Science from University College, London University in 1980. His interests include performance analysis and high availability in distributed computing environments. His email address is <mgunduc@bea.com>.