Including the Operating System Overhead into Analytical Models

Mariela J. Curiel[1]
Grupo de Sistemas Paralelos y D.
Departamento de Computación y T.I
Universidad Simón Bolívar,Caracas Venezuela.
curiel@ipc4.uib.es

Ramón Puigjaner
Departament de Ciències Matemàtiques i I.
Universitat de les Illes Balears, España
putxi@ps.uib.es.

Abstract

Our research looks at reducing the computing time in large simulation models, by offering the possibility of replacing the whole system or some of its parts by analytical models. A good candidate for replacement is the server (CPU with its disks). It can be replaced by a load dependent server whose characteristics will be obtained by the application of some analytical modelling techniques. In order to build realistic server models, they should include the operating system overhead. Two ways of including the operating system overhead in analytical models are proposed. However, the main difficulty is the data describing the overhead as a function of the workload. We propose a methodology to collect the overhead information from the workload using software monitors.

Keywords: Performance Evaluation, Queuing Networks, Analytical Models, Operating Systems.

1. Introduction

The goal of performance modelling is to estimate the performance of some system when the total system or some part of it - such as hardware, software or load- does not exist. In this case it is impossible to measure the performance using monitors. A solution to this problem is to make some mathematical models based on queueing networks. From models of this type we can estimate the performance behaviour of the system under study.

The main techniques to process queueing network performance models are analytical, simulation and numerical. With analytical techniques, models are solved in short execution times by the application of some closed formula. Their main limitations come from the set of conditions that the queuing network must satisfy. Simulation techniques do not have any theoretical limitation but the execution times can be quite long to obtain enough significant results.

Our research is oriented to obtain a significant reduction of the computing time in large simulation models. It will be accomplished by offering the possibility of reducing the simulation complexity and duration of such models by estimating the behaviour of system's parts by means of the application of analytical methods to them. One of the more susceptible elements to be studied is the server. The proposal is to replace parts of the global system by load dependent servers, whose characteristics have been obtained by applying some analytical technique.
There are well known models to accurately analyse the behaviour of a server. From the basic structure of the closed central sever model several improvements can be introduced. One of them is to introduce the operating system overhead.

[1] On leave at the UIB as graduate student working in the PhD thesis.

In the present document two ways of introducing the operating system overhead in analytical models are presented. We establish comparisons between both in order to determine which one can better represent the required overhead. However, to build these models is not the drawback. The main difficulty comes from the data describing this overhead as a function of the workload. A measurement methodology is proposed to obtain the required data from the workload or a representative benchmark, using software monitors.

This research is part of ESPRIT project 22354, HELIOS[2], devoted to the development of a tool for evaluating the performance of database (ORACLE) and object oriented (CORBA) client-server systems.

The remainder of this document is organised as follows: in Section 2, we start by reviewing the main techniques to develop performance models. In Section 3, the basic server models are presented. Sections 4 and 5 depict improvements to these basic server models by adding the memory management constraint and the operating system overhead, respectively. In Section 5, two analytical models that include the system overhead are described. The above models are compared and we present the problems that arise in the calibration step in Section 6. In Section 7, we expose a methodology to take measurements able to collect overhead information. Finally, in Section 8, we present a conclusion based on the results obtained from the comparison of models.

2. Evaluation Techniques.

Queueing network modelling is a particular approach to computer system modeling in which the computer system is represented as a network of queues. A queueing network is a collection of servers or service centres, which represent the computer resources, and customers, representing the users or the transactions. In sections 3 to 5 several models of server -from basic to more realistic models- will be studied. In the next paragraphs, we are going to present the different methods to evaluate queueing networks.

Two of the main techniques to extract performance estimations from a system represented in terms of a queueing network are: analytical and simulation techniques.

Computer based simulation involves writing a computer program that "imitates" the behaviour of the system, enabling us to "observe" it and to collect performance statistics. The simulation has no theoretical limitation, but the effort and time spent to develop and to debug a simulation program can be considerable. The execution time can also be quite long, if we whish, to obtain significant results.

Analytical techniques allow the modeller to compute the performance metrics by applying some kind of algorithm or closed formula (exact or approximate). Their limitations come from the set of conditions (type of stations, service and inter-arrival time distributions, etc.) that the queueing network must meet. Its advantages are simple debugging and short computing times. Exact analytical methods are based on the BCMP theorem [1]. It describes the set of conditions that a

[2] This work is partially supported by the UE Esprit project 22354 HELIOS. Mariela Curiel is supported in part by a CONICIT (Consejo Nacional de Investigación Científica y Tecnológica de Venezuela) and AECI/ICI (Agencia Española de Cooperación Internacional/Instituto de Cooperación Iberoamericana) scholarship.

queueing network must satisfy in order to have the state probability in product form. However, the BCMP theorem does not allow features such as: priorities, class dependent service times at FIFO exponential stations, finite buffers, FIFO non-exponential stations, etc. Queueing networks with such features are important in modelling computer and communication systems. Due to the difficulty -even imposibility- to obtain an exact solution of such networks, they must be approximately analysed. The different types of approximations can be grouped in three classes of methods: decomposition-aggregation, diffusion and iterative [7].

3. Server Models

The central model server is the basic queue network scheme to study computer systems performance. It can take the form of an open queue (Figure 1) or a closed (Figure 2) queueing network. In basic models only the critical components of the system are considered. These elements are the CPU, the disks and the workload intensity, defined by the arrival rate in an open system or by a set of terminals and the thinking times in a closed system.

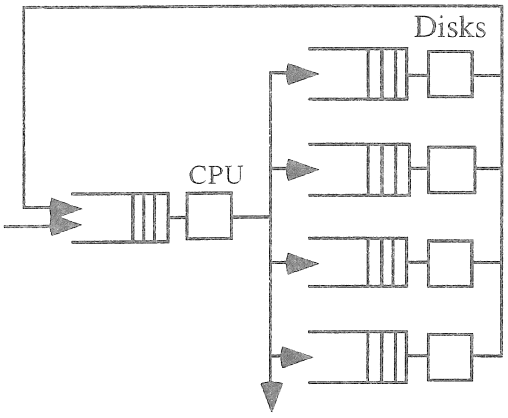

Figure 1: Open System

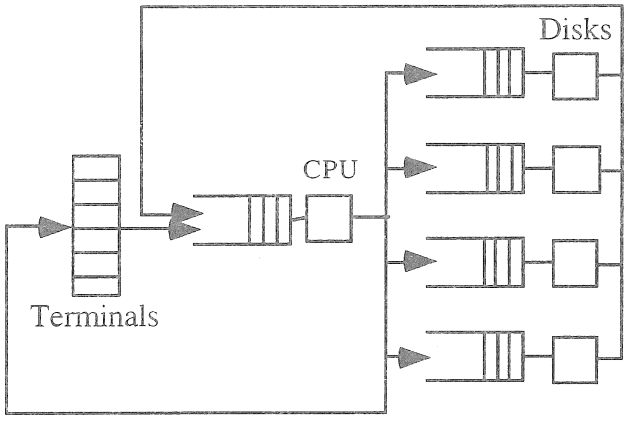

Figure 2: Closed System

Exist many choices to model the CPU. Let us see:

- *Batch Processor Model*: It is the simplest model. A FIFO queue dispatches the jobs and each job owns the CPU until the service is finished.
- *Processor Sharing Model*: In this model the jobs are executed during an uninterrupted quantum q while they own the CPU. If the service is not completed, the job returns to the queue in order to obtain a new quantum of service. It is desirable that when there are N jobs in the system, each one receives $1/N$ of the available time; so, the quantum` lenght should be reasonably short.
- *Interruptible Processor Model*: in this management policy, an arriving job can interrupt the running job and send it to a LIFO queue while it finishes its execution or is interrupted by another arriving job. Once the latter process finishes its work, the job at the top of the stack is resumed.

All these policies can be easily represented in analytical and simulation models.

With respect to the disk models, these could be represented by a FIFO queue without taking into account the type of path going to the main memory. To take into account the conflicts in the use of the common path to the main memory an iterative model can be used, allowing the representation of the service time load dependence.

4. Representing the Throughput-Limiting Effect of Memory Constraint

To simplify a server model some assumptions can be made, for example in a close model with a terminal workload we assume that the system has a fixed number of interactive users, and that enough memory exists to accommodate all the users requesting memory concurrently. Although these models are easy to solve, these simple assumptions are not always adequate. Taking into account that the systems resources have limited capacity it is very important to study the case where the memory constraints are sometimes but not continuously reached. In this section we show how to solve in analytical form a model with memory constraints.

The impact of the memory and its management on system performance is twofold:

- On the one hand, there is a restriction imposed by the availability of memory: it limits the number of processes that can be active simultaneously. A memory constraint places an upper bound on the extent to which processing resources (CPU, disk, etc.) can be utilised concurrently, an thus on the throughput of the system.
- On the other hand, the establishment of the multiprogramming factor avoids the eventual system degradation due to the operating system overhead (CPU and disk consumption).

In order to study how the growth in the number of processes affects the throughput, we will limit within the model the number of processes that can be simultaneously in the memory (multiprogramming factor). In the proposed system, a client request goes from the terminals to the CPU through the memory queue (see Figure 3). This system owns a non-BCMP sub-network and can not be solved using any of the exact analytical techniques. An approximate analytical method (decomposition-aggregation) or a simulation method must be used instead.

The idea behind the decomposition is to break up the queueing network into smaller sub-systems (decomposition step), so that each sub-system can be easily analysed in isolation, and then to put together these partial solutions (aggregation step), in order to obtain the solution of the whole queueing network. The difficulty comes from finding a good way to decompose the network under study into smaller sub-networks.

In networks with a non-BCMP sub network, a possible descomposition technique is based on the so called Norton Theorem [3] that isolates the non-BCMP sub-network from the whole network. Once isolated, we study only the BCMP part and replace it by a load dependent server. Finally, we study the non-BCMP sub-network connected to the load dependent server. A load dependent server can be considered as a server whose service capacity (the number of customer leaving the station per unit time) is a function of the customer population in its queue. Normally, the aggregated network should be solved by simulation.

Inspired in the Norton theorem, a possible decomposition could be to study the computer system (CPU plus disks) with different numbers of customers inside and replace it by a load dependent server that will be connected latter to the interactive terminals.

So, we decompose the model into two parts: the central or aggregated sub-system (CPU plus disks) and the complementary network (memory queue and terminals). The central sub-system is replaced by a load dependent server which is solved for each n, where n is a number equal or less than the multiprogramming factor. This produces the throughput $X(n)$ as a function of the number of customers. $X(n)$ will be the load dependent server capacity. Now, the original network is reduced to terminals plus the load dependent server whose input queue is the memory queue (Figure 3). In this particular case, the resultant queueing network can also be solved analytically.

5. Including the Operating System Overhead

In multiprogramming systems, part of the CPU consumption is due to operating system code execution working "on behalf of" user processes. The time in which the system is executing is called *system time*. Part of this time is fixed, in that it does not depend on system load or congestion. For example, the execution of system calls' code invoked by the user programs (e.g. **write** or **read** system calls to perform I/O operations). The other part of system time is variable, and typically increases with system load (e.g. execution of code due to paging, dispatching of processes, etc.). This is the so called ***operating system overhead.*** The operating system overhead in the analytical model can be represented by:

- A fixed increment in the CPU consumption and the number of disks accesses by the user programs.
- A variable increment (linear or of higher degree) of the CPU consumption and the number of disk accesses by user programs, depending both on the number of concurrent processes.
- A process or a set of processes consuming CPU and asking for I/Os, depending both on the number of user processes active in the system.

The first proposal is rather simple, for this reason we only take into account the two last approaches and develop analytical models based on each one. To establish comparisons between both models we have introduced the same number of customers, service demands and similar quantities to represent the operating system overhead. The models were developed using QNAP2V9[10], and the throughput limiting effect of the memory management was included.

In the first model the operating system overhead is represented by a linear increment in the CPU consumption and in the number of I/O operations per user processes. The increment is proportional to the number of user programs active in the system. In the second model a set of processes, that consume CPU and ask for I/Os (like **system processes**) in random fashion, represent the system overhead. The number of such processes is a function of the user processes active in the system. The CPU and disk consumption of each one of these processes linearly increase, depending on the number of active processes. In both models we include one CPU and four disks.

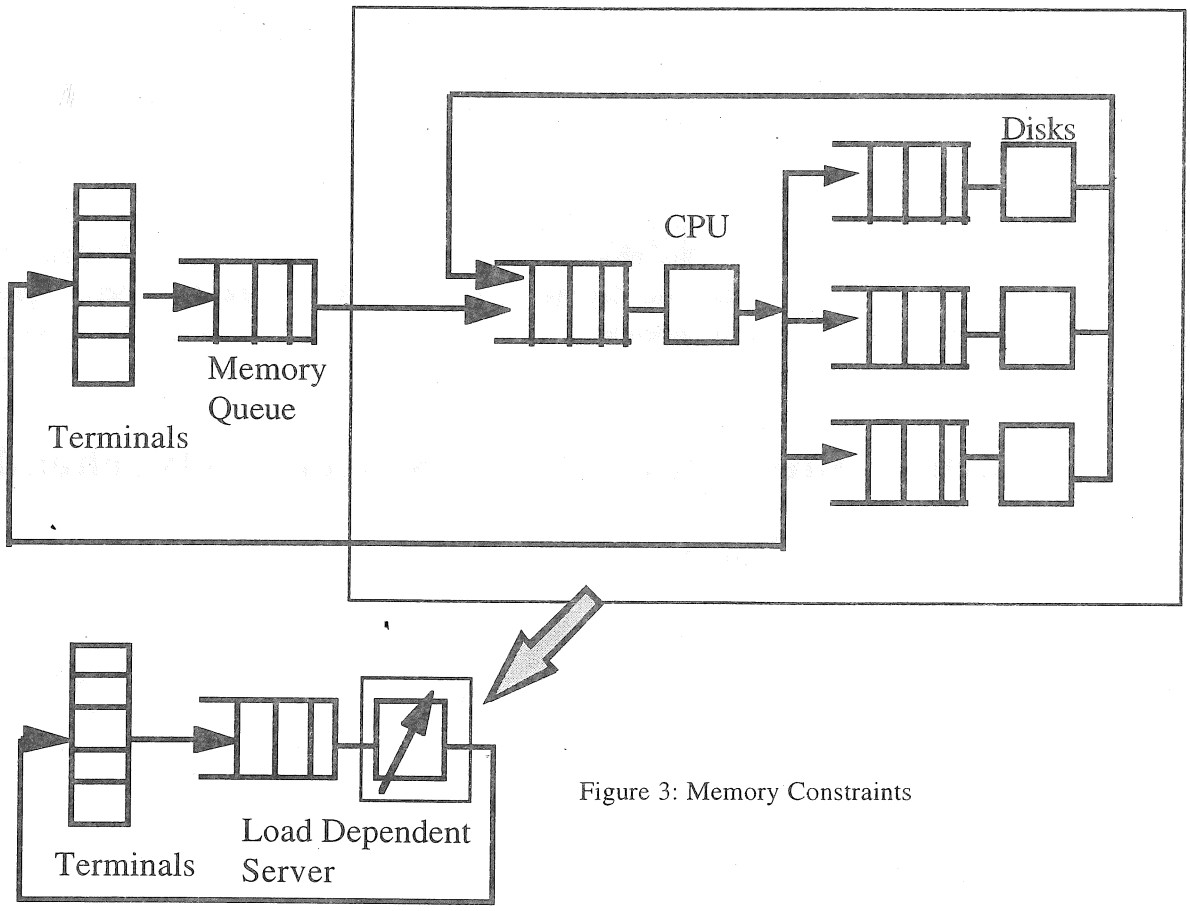

Figure 3: Memory Constraints

6. Comparison of Models

The following graphics show the CPU (Figure 4) and one disk (Figure 5) utilisation due to operating system overhead. In all the graphics the continuous lines represent the behaviour of the first model and the dashed lines represent the behaviour of the second model.

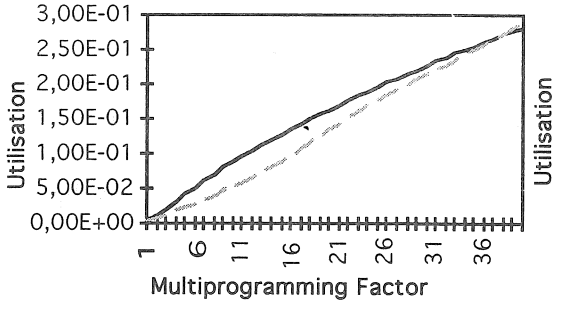

Figure 4 : CPU utilisation

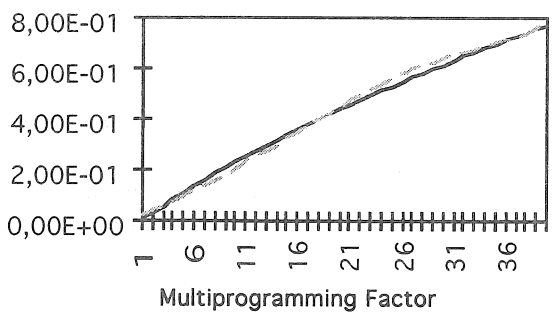

Figure 5: Disk Utilisation

The behaviour of the utilisation in all the disks is very similar, for this reason the graphics corresponding to the other disks are omitted. In all the cases the curves are practically superimposed on each other, which indicates similar overhead utilisation's. Concerning the CPU

utilisation, the differences are very small when we take into account the scale used. The above results suggest that the two models can be compared in the established range of multiprogramming factors. Graphics in figures 6 to 13 show the throughput and the response times in both models for 150, 300, 450 and 600 terminals. The multiprogramming factors are varied from 1 to 40.

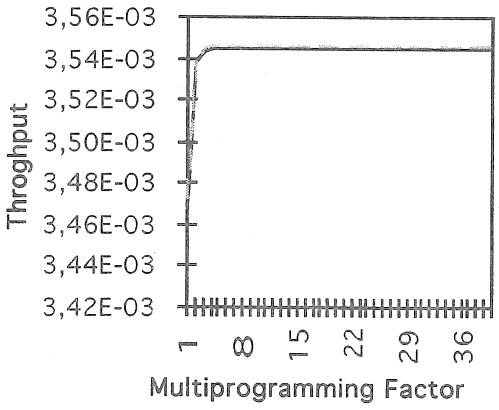

Figure 6: Throughput 150 Terminals

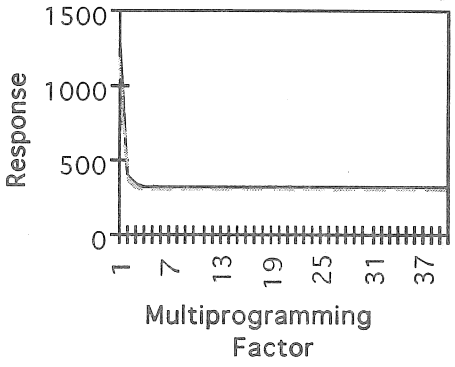

Figure 7: Response Times 150 Terminals

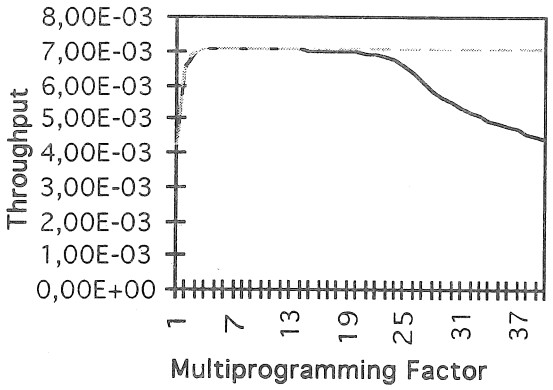

Figure 8: Throughput 300 Terminals

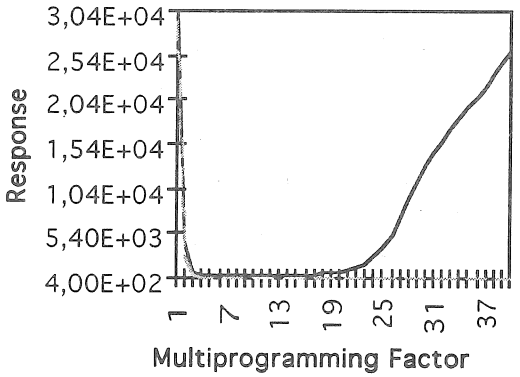

Figure 9: Response Times 300 Terminals

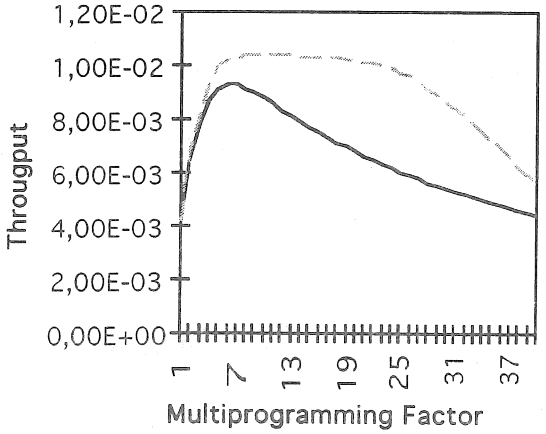

Figure 10: Throughput 450 Terminals

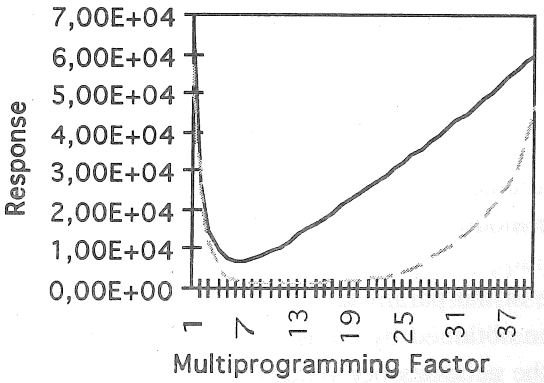

Figure 11: Response Times 450 Terminals

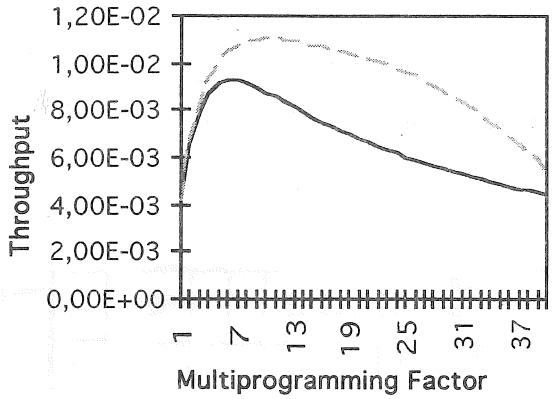

Figure 12: Throughput 600 Terminals

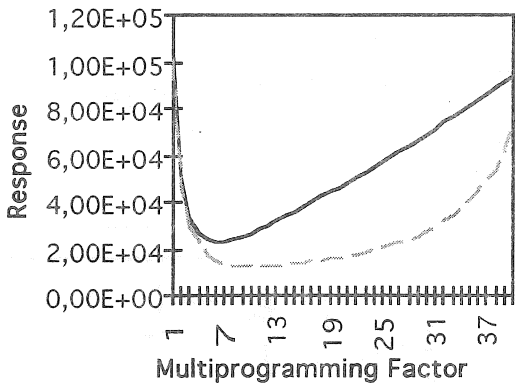

Figure 13: Response Times 600 Terminals

Concerning the throughput, we remark in most of the graphics, that the behaviour of both models is very similar and qualitatively consistent with our intuition and experience: whilst the multiprogramming factor increases the throughput increases but it is limited by the resources` capacity. For multiprogramming factors greater than or equal to a certain value the system is saturated and the throughput begins its degradation. With few terminals (150) this behaviour can not be observed in any of the models because the number of terminals is too small for submitting enough load to saturate the resources.

With respect to the response times, again for 450 and 600 terminals the desired behaviour can be remarked in both models: when the number of processes increases, the response time increases too, because of the conflicts in using the resources of the system. The operating system has to distribute the CPU, the memory and others resources among all the processes within the system; when the memory is scarce, the paging begins. The queues in the devices are long because there are a lot of processes executing I/O operations, including the paging. Remark also that in low multiprogramming factors the response times are high because the requests are blocked in the memory queue because only a short number of processes are allowed to be in it.

With 150 terminals the increment in the response time is not observed in any of the models for the reason explained above. For 300 terminals the first model reflects better the desired response and throughput behaviour. In the second model, higher values in response times and lower values in the throughput could be reached with multiprogramming factors greater than 40. In all the graphics, the response times are lower in the second model than in the first one.

However, we will not know with certainty which is the best model (i.e. which of them shows the right response times, throughput or overhead values) until we collect the information by measurement in benchmarking environments. This will enable us to introduce the right parameters to the models to improve their results.

Some of the parameters - such as the number of I/O operations per request class over each disk- can be obtained by using monitors and from our knowledge about the data distribution on disks and the accesses per request. To obtain other parameters- such as the growing pattern in CPU consumption or paging pattern- we are going to plan measurements on the workload or a representative benchmark. A methodology to perform these measurements is explained in the next section.

7. Measuring Methodology

Taking measurements over all feasible scenarios in complex systems is an impossible job. To solve this problem, commonly experimental designs - such as factorial or fractional factorial designs- can be applied [5]. The aim of the experimental design is to obtain the maximum information with the minimum number of experiments.

We propose a method to reduce the number of scenarios to be measured, based on our knowledge of the application. For example, in closed systems the existence of thinking times could make this task easier. If the thinking times are long, with respect to the system service times, the probability of having in the system as many processes as terminals is very low. Moreover, with brief thinking times, it is feasible to have a more loaded system. A way to discover which is our system's behaviour, is to build a simple model representing the workload. This model is calibrated -using the real workload or some of the components of a representative benchmark- and solved to obtain the most probable system states. In this first calibration step the overhead is not considered, the aim is to characterise the behaviour of the transactions belonging to the real or a similar fictitious system.

In the framework of the HELIOS project we are going to take measurements from the TPC-C[8] and TPC-D[9] benchmarks. The transactions founded in these benchmarks are representative of many real systems and for this reason should be easy to combine them in order to create an artificial working scenario similar to the desired one.

Once we have established the number of feasible scenarios, they can be reproduced in the system Concurrently with each scenario, software monitors can collect the requested data. Measures will be taken on a SPARC SUN-Ultra1 architecture with the Solaris 2.5 operating system [4][11] using the monitors vmstat, iostat, sar (from UNIX environments) and Proctool [2]. Variables like the CPU consumption due to the execution of system code or the number of pages "page-in" and "page-out" will be of our interest.

The information obtained will be subsequently analysed using statistical tools. Perhaps, many iterations taking measures, or other experimental designs could be required to obtain a suitable model of the overhead behaviour. The overhead information and the profile of the workload (TPC-C benchmark in the first phase) will be included in the proposed models and the model results will be compared with the measures of the real system in order to validate the proposed methodology.

8. Conclusions

Looking at the obtained results the most satisfactory conclusion we can come to is
throughput and response time behaviour can be obtained from the studied analytical models, including data representing the operating system overhead. Such behaviour is apparently (intuitively, if you wish) better reflected in the first model than in the second model. However, it is only performing measures over the workload that we are going to obtain data to calibrate and validate such models. So, we will know what model is better to represent the operating system overhead. Perhaps, some mix of both model will provide the best representation.

To take measures it is necessary to design a strategy to reduce the number of feasible scenarios. We have proposed a methodology based on the knowledge of the application that allows us to obtain more useful information with a reduced number of experiments.

Obtaining satisfactory results will permit us to develop more realistic simulation models with shorter execution times. These models will be included in a tool (HELIOS) that will allow us to predict the performance of data bases and distributed object oriented systems.

9. References

1. Baskett, T; Chandy, K; K.M; Muntz, R; and Palacios, F. " **Open, Closed, and Mixed Networks of Queues with Differents Classes of Customers**". JACM 22,2 (april 1975),
2. Cockrof Adrian. "**Sun Performance and Tunning SPARC & Solaris**". 1995. Sun Microsystems.
3. Gelenbe, E.; Mitrani, I."**Analysis and Synthesis of Computer Systems**". Academic Press 1981. 239 pp.
4. Goodheart Berny & Cox J. "**The Magic Garden Explained. The internals of Unix System V. Release 4**".1994. Prentice Hall.
5. Jain Raj. "**The Art of Computer System Performance Analysis**". Wiley Professional Computing. 1991. pp 94-100.
6. Lazowska E. Zahorjan J. Scott G. Sevcik K. "**Quantitative System Performance**". Prentice Hall 1984. pp 273-294.
7. Puigjaner Ramón, Juan J. Serrano, Alicia Moreno. "**Evaluación y Explotación de Sistemas Informáticos**". Editorial Síntesis 1995. 253 pp.
8. **TPC-C Benchmark**. Standard Specification. Revision 3.21. Transaction Processing Performance Council (TPC).1996.
9. **TPC Benchmark D.** Standard Specification. Revision 1.2. Transaction Processing Performance Council (TPC).1996.
10. **QNAP2**. Reference Manual. Simulog. June 1994.
11. Vahalia Uresh. "**Unix Internals: The New Frontiers**". Prentice Hall. Octuber 1995.