The Method of Layers

J.A. Rolia and K.C. Sevcik

***Abstract*—Distributed applications are being developed that contain one or more layers of software servers. Software processes within such systems suffer contention delays both for shared hardware and at the software servers. The responsiveness of these systems is affected by the software design, the threading level and number of instances of software processes, and the allocation of processes to processors. The Method of Layers (MOL) is proposed to provide performance estimates for such systems. The MOL uses the mean value analysis (MVA) Linearizer algorithm as a subprogram to assist in predicting model performance measures.**

***Index Terms*—Management of distributed applications, performance of systems, queueing network models, software performance engineering, software performance models.**

I. Introduction: Systems With Software Servers

Software systems are being developed in which software processes share resources and cooperate to accomplish overall system goals. Examples include applications developed in the Open Software Foundation's Distributed Computing Environment (DCE) [12] and the Object Management Group's Common Object Request Broker Architecture (CORBA) [21]. In such systems there is a likelihood that processes will incur delays due to contention for both hardware and software resources. The resulting system performance behaviour often defies intuition. The purpose of this paper is to describe an analytical performance modelling technique that can be used to help predict the performance of such systems and to give examples of how the technique can be applied. It is assumed that the reader has some familiarity with queueing network models [23], [17], [16], [7], [18].

The performance issues considered are of particular importance in distributed and multiprocessor systems. In such systems, software processes can act as both customers and *software servers* while sharing *hardware servers*. The requests for service amongst processes and for devices can be described as a *software process architecture*. This architecture is important, and often dominant in determining a system's performance. Changes in requests for service between processes, the number of instances of processes, the internal level of concurrency within processes, and the placement of processes on processors can all have a significant impact on system performance.

Measurement studies can be used to help understand the characteristics of architectures. However, prototyping and performance measurement can require a significant effort. Furthermore in some cases, the target environment needed for measurement may not yet exist. Instead we consider the use of predictive models to gain insights into system behaviour.

Performance modelling is a method of discovering which of a system's many features constrain performance by representing them in a predictive model. For the systems that are considered here, each process's *resource demands* such as processor time requirements, and requests for service from other processes, are quantified and used to define a model. The method of layers estimates contention for resources and process throughputs. The performance measures that are predicted are a superset of those for closed queueing network models and include:

- process throughputs and response times,
- process utilizations, and the average number of processes queued at each serving process,
- device utilizations, and average queue lengths.

A change in model parameters permits the prediction of system performance under modified circumstances. Using the model, a range of system scenarios can be examined without actually altering the system under study.

The method of layers can be used to determine whether a software server becomes fully utilized. If so, the software server has become the system bottleneck. Increasing the power of hardware can increase the throughput of a system, but it may be more economical to alter the software or assignment of processes to processors. Such techniques can also be used as part of a performance engineering methodology to choose a software process architecture that will meet performance goals that are part of the specification [28], [9].

Other work in this area has been carried out by Agrawal and Buzen [1] and Woodside and coworkers [33], [19], [20], [11], [34]. Agrawal and Buzen offer a model for serialization delays. Woodside et al. developed a more general technique named SRVN for predicting the performance of stochastic rendezvous networks. These networks are similar to the models presented in this paper. Their solution technique differs from the approach presented here in terms of the overall performance prediction algorithm, the types of software interactions that have been considered, and in the mean value analysis algorithm that is used. In this paper, a tailored version of the Linearizer algorithm [10], [27] is used, while an approach more like the Bard-Schweitzer approximate MVA has been implemented with SRVN. Linearizer offers accuracy superior to the Bard-Schweitzer algorithm in some circumstances [10].

In Section II, models are described whose performance measures can be estimated using the method of layers. The types of software interactions and hardware interactions that can be represented in the models are given. Section III fully defines the model parameters and presents the method of layers algorithm. Analysis for a Multiserver interaction is pre-

Manuscript received December 1993.
J.A. Rolia is with the Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, K1S 5B6.
K.C. Sevcik is with the Department of Computer Science, University of Toronto, Toronto, Canada, M5S 1A4.
IEEECS Log Number S95028.

sented in Section IV. This software interaction permits the replication of a serving process in models, as is exemplified in the sample application considered in Section V. Finally, conclusions are offered in Section VI.

II. Layered Queueing Models

In this paper, the models are called *layered queuing models* (LQMs). Processes that can be said to have statistically identical behaviour form a group or class of processes. Groups use devices but they can also request services from and provide services to other groups. The notion of a group has been introduced to enforce the notion that processes can both provide and receive service. In general, it is assumed that a serving group provides service in a first-come-first-served (FCFS) manner, with callers waiting until service is complete before being released and continuing with their own execution. Unless otherwise specified, serving groups are assumed to contain one process.

Model input parameters for LQMs include those of standard queueing network models. In addition, the average number of visits a process makes to other processes, and process scheduling disciplines must also be specified. A process's scheduling discipline is determined by the nature of the service it provides to other processes.

Performance models for individual software processes and components have been considered by Beizer [5], [6], Smith [28], [29], Qin [22], and Vetland [31]. Based on estimates of conditional branching probabilities a process can be analysed to discover the expected number of times each of its statement blocks will be executed. This information can be used to estimate average service demands at devices and the average number of visits a process makes to other processes for service. Measurement based characterizations of components can also be appropriate for estimating demands and visits [31]. These demands and visits are used as input parameters for the corresponding group in the LQM. If the system being modelled already exists, system measurement tools might also be used to obtain the values of input parameters for the LQM.

Techniques for representing several types of client-server process interactions have been developed recently. Complete descriptions are available [26]. They can be used in conjunction with the MOL to model the behaviour of software systems. Each type of process interaction is implemented as a type of server with a particular scheduling discipline. The Linearizer MVA algorithm is modified and extended to include appropriate residence time expressions for each new type of server. The new residence time expressions are for the Rendezvous, Multiple-Entry, Multiserver, and SYNC servers.

The Rendezvous server is a FCFS process with two phases of service. A caller is released after the first phase, but the server cannot begin to provide service again until it completes its second phase of service. The Multiple-Entry server is a FCFS Rendezvous server that can be used to represent processes that provide services with significantly different service times. The Multiserver permits many identical serving processes to share a single queue of customers, and the SYNC server is used to introduce synchronization between two groups of processes into the models. The Multiserver is described in Section IV and is used in the sample application modelled in Section V.

Any of the device scheduling and service time distributions considered in the MVA literature are permitted when describing devices in a system. The MVA techniques can be used without modification. They include:

- Delay (DELAY), Round Robin Processor Sharing (PS), Last-In-First-Out (LIFO), and, with exponential service times, First-Come-First-Served (FCFS) [4].
- First-Come-First-Served scheduling with general service time distributions (HVFCFS) [24].
- Priority-Preemptive-Resume servers at which groups are assigned fixed priorities and queue for service in order of priority. Processes with the same priority are served in a first-come-first-served manner (PPR) [8].

The MOL can be used to predict the performance of software systems that can be described in terms of the types of servers that are available.

A sample LQM is shown in Fig. 1. Each parallelogram is a group of one or more processes and each circle is a device. Directed arcs indicate requests for service from the calling group to the serving group or device. The requests for service from processes and devices are represented in separate diagrams for the purpose of clarity.

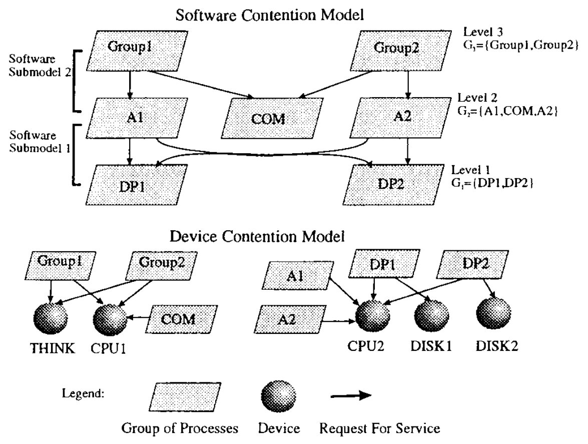


Fig. 1. A sample software process architecture.

The requests for service amongst processes can be described as a directed graph. Processes that do not request service from other processes are defined to be at the lower levels and other processes are at higher levels. A process that requests service from another process is an ancestor of the serving process. Each group is at exactly one level. Groups that use hardware devices but call on no other software servers are at level 1. It is assumed that there are no cycles in the graph so that the model's requests for service are acyclic. The groups form a hierarchy with $|L|$ levels, $1 \ldots L$. A topological sort [2] can be used to identify the level of each group. Models with cyclic request graphs have not been considered.

The basic method of layers solution technique only permits groups at level l to visit groups at level $l - 1$. However if requests for service span more than one level, throughput equivalent groups that model the requests and their service rates can be introduced at the intermediate levels in the LQM to overcome the restriction [26].

III. The Method of Layers

The method of layers is used to predict the performance measures of systems represented using layered group models. The MOL divides an LQM into two complementary models, one for software and one for devices, and combines the results to provide performance estimates for the system under study.

The software contention model describes the relationships in the software process architecture and is used to predict software contention delays. The software contention model is mapped onto a sequence of $L - 1$ two-level submodels that are used to estimate the software contention between successive levels of groups. The average response times of groups at one level of the model are estimated by viewing the groups at the next lower level as servers in a closed separable queueing network model. Groups that request service at the higher level are considered as customer classes, and groups that provide service at the lower level are represented as servers. The response time for process communication via the remote procedure call primitive has been estimated as the residence time at a first-come-first-served server [25]. This representation captures the possible queueing delays incurred by the groups requesting service if the serving group is busy doing work on behalf of another customer.

There are only $L - 1$ submodels because groups at level 1 do not visit other groups, so they are never considered as customer classes in a software contention submodel. The submodels are ordered from $L - 1$ down to 1, with groups at level l representing customer classes in the $l - 1$st submodel. The software submodels are solved in submodel order $L - 1$ down to 1. See Fig. 1 for an illustration of the relationship between submodel number and software level number.

Another queueing network model called the device contention model is used to determine queueing delays at devices. In this model, each group in the system is represented as a customer group, and each device as a server. The results of the two models are combined to provide performance estimates for the system. As in the method of complementary delays [14], the two models are solved alternately, with the solution of one determining some of the input parameters of the other.

Groups that do not provide service to other groups are defined as non-serving groups, other groups are defined as *serving* groups. Each serving group and each device is represented as a server in exactly one submodel. All the callers of the server are also represented in the same submodel, along with the callers' other servers.

A sketch of the MOL algorithm is given in Fig. 2, followed by a detailed description.

The input parameters for an LQM are a superset of those required for closed separable queueing network models and are:[1]

- G, K Set of groups and devices.
- L The number of software levels in the LQM hierarchy(*).
- G_n $\forall$ $n \in 1 \ldots L$ The set of groups at level n of the hierarchy(*).
- N_g $\forall$ $g \in G$ Population of group g.
- $V_{g,h}$ $\forall$ $g,h \in G$ The average number of visits from group g to group h(*). Group g is assumed to be one level higher than group h.
- $V_{g,k}$ $\forall$ $g \in G$ $\forall$ $k \in K$ The average number of visits from group g to device k.
- $S_{g,k}$ $\forall$ $g \in G$ $\forall$ $k \in K$ The average service time of a visit by group g at device k.
- Z_g $\forall$ $g \in G$ The think time for group g.
- ψ_j $\forall$ $j \in G \cup K$ The queueing discipline of group(*) or device j.

```
The Method of Layers
Initialize the response time estimates for groups assuming no device or serving
group contention.
WHILE successive group response time estimates have not reached a fixed point DO
    WHILE successive group response time estimates have not reached a fixed point DO
        FOR software submodel $l = L - 1$ downto 1 DO
            Solve the submodel using Linearizer with the following residence
            time expressions:
                FCFS, Rendezvous, Multiple-Entry, Multi-Server,
                SYNC, and DELAY.
            Update the submodel's group response times and utilizations.
        END FOR
    ENDWHILE
    Solve the device contention model using Linearizer with the
    following residence time expressions:
        DELAY, PS, FCFS, LIFO, HVFCFS, and PPR.
    Update the group response time and device utilization estimates.
END WHILE
End
```

Fig. 2. Sketch of the method of layers algorithm.

The think time of a group is the average time between the completion time of a process in the group and its next starting time, including periods of duration zero caused by contiguous computations. For serving groups, the think time Z_g is assumed to have value zero.

In addition to the metrics provided by MVA for device performance, some of the values that are computed by the MOL are:

- R_g $\forall$ $g \in G$ The average response time of a group g process.
- $R_{g,h}$ $\forall$ $g, h \in G$ The average residence time of a group g process when visiting group h.
- U_g $\forall$ $g \in G$ The utilization of a group g process.
- R_g^{GRP} $\forall$ $g \in G$ The average time a group g process spends acquiring service from other groups.

1. An asterisk marks those parameters that are extensions beyond separable models.

- R_g^{DEV} $\forall\ g \in G$ The average time a group g process spends acquiring service from devices.
- R_g^{IDL} $\forall\ g \in G$ The average time a group g process spends idle between invocations.

Note that $R_g = R_g^{GRP} + R_g^{DEV}$ for all groups, $R_g^{IDL} = Z_g$ for nonserving groups, and R_g^{IDL} is computed for serving groups. The throughput of one of the group g processes is denoted as X_g^1 and is its average number of completions per unit time:

$$X_g^1 = \frac{1}{\left(R_g^{GRP} + R_g^{DEV} + R_g^{IDL}\right)} = \frac{1}{\left(R_g + R_g^{IDL}\right)}.$$

The superscript of 1 indicates that the value corresponds to one customer of the group. The same notation is used for utilization as well. The average idle period for a process of group g is its average intercompletion time less its average response time and is simply (by rewriting the expression for X_g^1)

$$R_g^{IDL} = 1 / X_g^1 - R_g.$$

Using the utilization law [17], the utilization of each process in group g is $U_g^1 = X_g^1 R_g$. Thus, the average idle time of a group g process can be expressed as follows

$$R_g^{IDL} = R_g / U_g^1 - R_g.$$

Note that R_g^{IDL} and U_g are not input parameters for an LQM. Initially, the R_g^{IDL} values are only known for nonserving groups.

The purpose of the MOL is to find a fixed point where the predicted values for R_g^{IDL} and U_g are consistent with respect to all of the submodels. At that point, the results of the MVA calculations approximate the performance measures for the system under consideration. Intuitively, this is the point at which predicted group idle times and utilizations are balanced so that each group in the model has the same throughput whether it is considered as a customer class or as a server class (the rate of completions of the serving group equals the rate of requests for its service), and the average service time required by callers of the group equals its average response time.

In the following paragraphs, the software contention submodels are discussed, the device contention model is described, and then the relationship between the models is explained. The parameters of the queueing network model are described in terms of the LQM parameters.

Initially, it is assumed that there is no device or software contention in the models. The response time of a group is simply the sum of the resource demands required by the group. The initial response time estimates for the groups of the LQM are computed software level by software level in order $l = 1$ to $l = L$. Note that G_0 is the empty set because there are no groups at level zero.

$$\begin{aligned} R_g^{DEV} &= \Sigma_{k \in K} V_{g,k} S_{g,k} && \forall g \in G_l \\ R_g^{GRP} &= \Sigma_{h \in G_{l-1}} V_{g,h} R_h && \forall g \in G_l \\ R_g &= R_g^{DEV} + R_g^{GRP} && \forall g \in G_l. \end{aligned}$$

Consider the software submodel $l - 1$ in which groups g in G_l are considered as customer classes, and groups h in G_{l-1} are servers. In the submodel's corresponding queueing network model, the think time Z_g is the time that the group's processes do not spend competing for other groups,

$$Z_g = R_g^{DEV} + R_g^{IDL}.$$

The R_g^{DEV} are obtained from the device contention model. The values R_g^{IDL} are input parameters of non-serving groups of the LQM, and are computed using the relation

$$R_g^{IDL} = R_g / U_g^1 - R_g$$

for serving groups. The parameters N_g, $V_{g,h}$, and ψ_h are given in the LQM's input parameters. To complete the queueing network model's description, the service time of group g per visit to group h is required. $S_{g,h}$ is the average service time requested by a customer of group g at server h. $S_{g,h}$ is not an input parameter to the model. It is estimated using intermediate results from the MOL.

$$S_{g,h} = R_h^{GRP} + R_h^{DEV} = R_h.$$

Linearizer is used to predict the software contention between the two levels represented by the submodel. The results provide new estimates for R_g^{GRP} for all g in G_l and U_h for all h in G_{l-1}. The utilization U_h is used to predict R_h^{IDL} and hence the think time for the group h when it is considered as a customer class in the next submodel.

The set of submodels are solved repeatedly, in submodel order $L - 1$ down to 1, until the differences in successive nonserving group response time estimates differ by less than some tolerance. At this stage, the software model is assumed to have reached a fixed point. At least $L - 1$ iterations are used to ensure that the performance measures of submodel 1 affect those of submodel $L - 1$.

The device contention model is a closed queueing network model in which each group is represented as a customer class, and each device as a server. In this queueing network model, the think time Z_g of group g is time that the group's processes do not spend competing with other groups for devices,

$$Z_g = R_g^{GRP} + R_g^{IDL}$$

The values R_g^{GRP} and R_g^{IDL} are obtained from the software contention model. The remaining parameters required to complete the device contention queueing network model's description are N_g, $V_{g,k}$, $S_{g,k}$, and ψ_k and are obtained directly from the LQM's parameters.

Furthermore, each serving process that uses devices does so on behalf of its callers, and their callers. When a caller requests service, it waits in its server's queue until the service is complete. Recursively, if the caller is a server, its caller waits in its queue until the service is complete. The server's average queue lengths at devices that are incurred while an ancestor is waiting for service to complete should not be included in the ancestor's arrival instant queue lengths at the devices.

Let group a be an ancestor of group h. A process in group a can not compete both for service at a process in group h and at devices, at the same time. Define $F_{a,h}$ as the fraction of a serving group h's average queue length that is caused by group a requests for service. It estimates the portion of work that each process in group h does on behalf of each ancestor in group a. This portion of the server's work is removed from the ancestor's arrival instant queue length at devices.

$$F_{a,h} = \frac{Q^1_{a,h}}{Q^1_h}.$$

Using Little's Law, $Q^1_{a,h}$ can be computed for each ancestor-decendant pair of processes in a straightforward manner. The values are computed prior to the solution of the device contention model using MVA.

The performance measures for the device contention model are predicted using a modified Linearizer algorithm in which the arrival instant queue length of a group a at a device m does not include the portions $F_{a,h}$ of each of the serving group h's average queue length at the device. In Linearizer [27], the change was made to the arrival instant queue length calculation in the core subroutine. Using the notation of de Souza e Silva and Muntz:

$$\begin{aligned} L_{m,a}(\vec{N}-\vec{e}_j) = \; & L_m(\vec{N}) - L_{m,j}(\vec{N})/N_j \\ & + D'_{m,j}(\vec{N}) - D_{m,j,j}(\vec{N}) \\ & - \Sigma_{h \in G} F_{a,h}\, L_{m,h}(\vec{N})/N_h . \end{aligned}$$

In the expression, $L_{m,a}(\vec{N}-\vec{e}_j)$ is the arrival instant queue length of a customer in class a arriving at server m with one customer of class j removed from the model. $D'_{m,j}(\vec{N})$ and $D_{m,j,j}(\vec{N})$ are values defined and computed by Linearizer that take into account the change in contribution of each customer class to server m's queue length caused by the removal of a customer of class j.

Finally, the response time estimates for groups at devices are averaged over successive solutions of the device contention model to speed up convergence to a fixed point. Empirically, the fixed point for the entire model does not change, however fewer solutions of the device contention model are required.

Now consider the relationship between the software and device contention models. The software contention model provides estimates for R^{GRP}_g and R^{IDL}_g. These values are used to specify group think times in the device contention model, which is then solved to provide new estimates for R^{DEV}_g for all g in G. Once this is done response time estimates for the groups in the software contention model can be computed as follows. The computation is done by software level, in order $l = 1$ to $l = L$.

$$\begin{aligned} R^{DEV}_g &= \Sigma^K_{k=1} V_{g,k}\, R_{g,k} && \forall g \in G_l \\ R^{GRP}_g &= \Sigma_{h \in G_{l-1}} V_{g,h}\, R_h && \forall g \in G_l \\ R_g &= R^{DEV}_g + R^{GRP}_g && \forall g \in G_l . \end{aligned}$$

The MOL alternates between software and device contention submodels until the differences in non-serving group response time estimates, after solving the device contention model, are less than some tolerance. At this stage, the performance metrics for the LQM have reached a fixed point. This completes the description of the MOL.

```
Algorithm Method of Layers
Parameters

        • Model Inputs: an LQM.
        • Model Outputs: Performance measures: $R_g$, $U_g$, $R_{g,h}$,
          $U_{g,h}$, $R_{g,k}$, $U_{g,k}$.
        • Model Intermediate Results: $Step4R_g$ and
          $Step5R_g$ are copies of non-serving group response time es-
          timates for the previous iteration. The step 4 loop is used
          to find a fixed point for the software submodels. Step 5 is
          used to find a fixed point for the combined software and
          device contention models.
        • Notation: $\downarrow$ and $\uparrow$ indicate input and output parameters,
          respectively.
          $\Delta(R, StepR)$ compares two arrays of response time esti-
          mates, returns the maximum relative error with respect to
          $R$, and copies $R$ to $StepR$. $\tau$ is the required tolerance for
          convergence of device and software contention loops. (The
          models in this paper have been solved with $\tau = 0.005$.)
Begin
Step 1.
    Initialize intermediate result vectors
    $Step4R_g$, $Step5R_g \leftarrow 0 \; \forall$ non-serving groups $g \in G$
    $G_0 \leftarrow \emptyset$
Step 2.
    Initialize assuming no device contention, solve in order $l = 1$ to
    $l = L$

            $R_{g,k} \leftarrow$      $V_{g,k}\, S_{g,k} \; \forall g \in G_l, \; k \in K$
            $R^{DEV}_g \leftarrow$    $\sum_{k \in K} V_{g,k}\, S_{g,k} \; \forall g \in G_l$
            $R^{GRP}_g \leftarrow$    $\sum_{h \in G_{l-1}} V_{g,h}\, (R^{DEV}_h + R^{GRP}_h)$
                          $\forall g \in G_l$
            $R_g \leftarrow$          $R^{DEV}_g + R^{GRP}_g \; \forall g \in G_l$
            $R^{IDL}_g \leftarrow$    $Z_g \; \forall$ non-serving groups $g \in G_l$

Loop Device Contention
    $iter \leftarrow 0$

  Begin Loop Software Contention
  Step 3.
      $iter = iter + 1$
      For $l \leftarrow L$ downto 2 do
        Apply MVA (
            $\downarrow$Customer classes:
                  $g \in G_l$,
            $\downarrow$ Customer populations:
              $N_g$ for $g \in G_l$,
            $\downarrow$Customer think times:
              $Z_g = R^{DEV}_g + R^{IDL}_g$ for $g \in G_l$,
            $\downarrow$Servers:
              $h \in G_{l-1}$,
            $\downarrow$Visits:
              $V_{g,h}$ from customers $g$ at level $G_l$ to
              servers $h$ at level $G_{l-1}$,
            $\downarrow$Service times:
              $R_h = R^{DEV}_h + R^{GRP}_h$ of groups $h \in G_{l-1}$,
            $\downarrow$Scheduling Disciplines:
              $\psi_h$ of groups $h \in G_{l-1}$,
            $\uparrow$Residence times:
              $R_{g,h}$ of groups $g \in G_l$ at servers $h \in G_{l-1}$,
            $\uparrow$Utilizations:
              $U_h$ of groups $h \in G_{l-1}$)
        $R^{GRP}_g \leftarrow \sum_{h \in G_{l-1}} V_{g,h}\, R_{g,h} \; \forall g \in G_l$
        $R_g \leftarrow R^{DEV}_g + R^{GRP}_g \; \forall g \in G_l$
        $R^{IDL}_h \leftarrow R_h / U^1_h - R_h \; \forall h \in G_{l-1}$
      End For.
```

```
Step 4. Exit when
    $\Delta(R_g, Step4R_g) < \tau \; \forall$ non-serving groups $g \in G$
    and $iter > (L-1)$.
End Loop Software Contention
Step 5. Exit when
    $\Delta(R_g, Step5R_g) < \tau \; \forall$ non-serving groups $g \in G$
Step 6. Compute the $F_{g,h} \; \forall g, h$ used in Step 7.

Step 7.

    $R^{OLD}_{g,k} \leftarrow R_{g,k} \; \forall g \in G, \; k \in K$
    Apply MVA(
        ↓Customer classes:
         $g \in G$,
        ↓Customer populations:
         $N_g$ for $g \in G$,
        ↓Customer think times:
         $Z_g = R^{GRP}_g + R^{IDL}_g$ for $g \in G$,
        ↓Servers:
         $k \in K$,
        ↓Visits:
         $V_{g,k}$ from customers $g \in G$ to servers $k \in K$,
        ↓Service times:
         $S_{g,k}$ of groups $g \in G$ while at $k \in K$,
        ↓Scheduling Disciplines:
         $\psi_h$ of servers $k \in K$,
        ↓Fractions:
         $F_{g,h} \; \forall g, h \in G$,
        ↑Residence times:
         $R_{g,k}$ of groups $g \in G$ at servers $k \in K$,
        ↑Utilizations:
         $U_k$ of device $k \in K$)
    $R^{DEV}_g \leftarrow \sum_{k \in K} V_{g,k} (R_{g,k} + R^{OLD}_{g,k})/2 \; \forall g \in G$
    $R^{GRP}_g \leftarrow \sum_{l=1}^{l=L} \sum_{h \in G_{l-1}} V_{g,h} (R^{DEV}_h + R^{GRP}_h)$
        $\forall g \in G_l$
    $R_g \leftarrow R^{DEV}_g + R^{GRP}_g \; \forall g \in G$
End Loop Device Contention
End
```

Consider the time complexity of each iteration of the method. Let $|G|$ be the number of groups, and $|K|$ be the number of devices. When Linearizer is used as the basis of solution, the time complexities of steps 1 through 7 are as follows: $O(|G|)$, $O(|G|(|G| + |K|))$, $O(L|G|^3)$, $O(|G|)$, $O(|G|)$, $O(|G|^2)$, and $O(|G|^3|K|)$. The time complexity of Linearizer in step 7, the device contention model, includes the effort to update the arrival instant queue lengths of requesting processes.

The value of L is a function of the system that is described; it does not depend upon the number of groups $|G|$. Thus, the total complexity is considered to be $O(max(L|G|^3, |G|^3|K|))$.

IV. The Multiserver

The Multiserver is a service center at which a single queue of customers shares one or more identical servers. The modelling technique that is presented can be used when the servers are either software processes, threads within a single process [13], [3], or hardware servers such as multiple processors.

In software, a Multiserver can be used to provide multiple instances of a service simultaneously. This is particularly useful when a process acts as an agent for client processes. If a process becomes a bottleneck, the number of instances of the process can be increased thereby decreasing the customer's queueing delay at the software server and shifting contention to other software servers or devices.

In the approach that has been developed, the Multiserver is a server with two entries, each with a queue, but does not actually provide any service. The first entry is used by customers who request access to one of the multiple servers. Its second entry is used by servers that provide service to the customers. The servers are permitted to visit both devices and other software servers. The population of the serving group defines the number of instances of the serving process in the model. The Multiserver itself always has a population of one.

Each entry of the Multiserver has its own residence time expression. The residence time expressions for the entries estimate the time required for a caller to receive D units of service, and for a free server to be matched with a customer in order to provide D units of service, respectively.

Let the number of servers, S, be greater than one, and the number of customers that share the server be N. Server utilization is determined by the utilization of the Multiserver s and is defined as follows. Let $X_{g,s}(N)$ and $X_{g,s}(N-1)$ be the throughput of group g processes at Multi-Server s at populations N and $N-1$, respectively. The utilization of the Multiserver s at population $N-1$ is:

$$U_s(N-1) = X_{g,s}(N-1)\; D.$$

The utilization of the Multiserver s at population N is:

$$U_s(N) = X_{g,s}(N)\; D.$$

And finally, the servers in group h provide service on behalf of the Multiserver s. The utilization of each of the servers in group h at population $N-1$ is:

$$U_h^1(N-1) = U_s(N-1)/S.$$

Define $P[Q \geq S]$ as the probability that there are S or more customers at the Multiserver. An arriving customer at the Multiserver will only suffer a queueing delay if all the servers are busy serving other customers. This must be taken into account in the Multiserver's residence time expression.

The $M/M/m/K$ model is a closed Multiserver model with a Poisson arrival process, exponentially distributed service times, m servers, and K customers [15]. It provides the exact value of $P[Q \geq S]$ for the Multiserver. Unfortunately, it requires too much computation to use with the Multiserver residence time expression. Two estimates for $P[Q \geq S]$ have been considered. They are the Erlang C formula [15] for an open model of the Multiserver, and another estimate called the independent uncorrelated server (IUS) estimate that is based on each server's utilization.

The Erlang C formula is:

$$P[Q \geq S] = \frac{\dfrac{(S\rho)^S}{S!}\dfrac{1}{1-\rho}}{\Sigma_{k=0}^{S-1}\dfrac{(S\rho)^k}{k!} + \dfrac{(S\rho)^S}{S!}\dfrac{1}{1-\rho}}.$$

where $\rho = U_h^1(N)$ is the utilization of each of the servers.

The IUS estimate is based on the assumption that the times at which the servers are busy are independent and uncorrelated. It is:

$$P[Q \geq S] = U_h^1(N)^S.$$

$U_h^1(N)$ raised to the power S is the joint probability that all servers in group h are busy.

In Table I the exact value of $P[Q \geq S]$ is compared with the

Erlang C formula and the IUS estimate for a range of models. In the table, N is the number of customers sharing the Multiserver, S is the number of servers at the Multiserver, and $U_h^1(N)$ is the utilization value of each of the S servers. Both estimates are good at very high utilizations and when the number of customers is at least twice as large as the number of servers. The Erlang C formula is always pessimistic, and the IUS estimate always optimistic. Since the IUS estimate is more accurate when the number of customers is close to the number of servers, of comparable accuracy elsewhere, and easier to compute than the Erlang C formula, it has been chosen for use in the solution algorithm.

TABLE I
A COMPARISON OF $P[Q \geq S]$ APPROXIMATIONS

N	S	$U_h^1(N)$	Exact $P[Q \geq S]$	IUS $P[Q \geq S]$	Erlang C $P[Q \geq S]$
5	4	0.1	0.0003	0.0002	0.001
		0.3	0.012	0.007	0.033
		0.5	0.085	0.058	0.165
		0.7	0.29	0.24	0.42
		0.9	0.68	0.64	0.78
		0.98	0.89	0.88	0.93
10	4	0.1	0.0009	0.0002	0.0015
		0.3	0.03	0.01	0.05
		0.5	0.15	0.07	0.18
		0.7	0.39	0.26	0.45
		0.9	0.74	0.66	0.79
		0.98	0.93	0.91	0.95
10	7	0.1	0.0000	0.0000	0.0000
		0.3	0.001	0.0002	0.006
		0.5	0.03	0.008	0.08
		0.7	0.17	0.08	0.30
		0.9	0.59	0.48	0.72
		0.98	0.91	0.88	0.95
10	9	0.1	0.0000	0.0000	0.0000
		0.3	0.0001	0.0000	0.0023
		0.5	0.005	0.002	0.046
		0.7	0.07	0.04	0.24
		0.9	0.49	0.43	0.71
		0.98	0.83	0.81	0.92

The utilization value is used to estimate the probability that a customer will have to wait for a free server. Errors in the utilization estimate affect the accuracy of the residence time expression most when there is a small number of servers and they are near saturation.

In the residence time expression, the probability that a customer arriving at server s will find that all servers in group h are busy is defined as:

$$B_s(N-1) \leftarrow U_h^1(N-1)^S \; S>1.$$

In this closed model, the $N-1$ population vector is used because an arriving customer will only suffer a delay if all the servers are busy serving other customers. When the number of servers equals one, the residence time expression of a FCFS server should be used for entry 1.

The customer's residence time expression introduces into the analysis the fact that when a server is available it will not be necessary to suffer a queueing delay. Also, when all servers are busy the expected time to the next completion is assumed to be D/S. Thus the residence time equation for a customer of class g at entry 1 of Multiserver s, $R_{1,g,s}(N)$, is:

$$R_{1,g,s}(N) \leftarrow D\left(1+\frac{Q_{1,g,s}(N-1)}{S} B_s(N-1)\right).$$

The arrival instant queue length for a customer at a Multiserver, $Q_{1,g,s}(N-1)$, includes customers receiving service and those queued for service. If there is a free server, then the arriving customer will not be delayed.

Each of the class h servers has a residence time at entry 2 of the Multiserver s, $R_{2,h,s}$, that is a function of its utilization. It is the time that a server in class h spends idle (that is, not serving a customer) which is the server's cycle time minus the service time.

$$R_{2,h,s}(S) \leftarrow D/U_h^1(N)-D.$$

The following equations summarize the Multiserver:

$$\begin{aligned} B_s(N-1) &\leftarrow U_h^1(N-1)^S \; S>1 \\ R_{1,g,s}(N) &\leftarrow D\left(1+\frac{Q_{1,g,s}(N-1)}{S} B_s(N-1)\right) \\ R_{2,h,s}(S) &\leftarrow D/U_h^1(N)-D. \end{aligned}$$

In Linearizer, $B_s(N-1)$ is evaluated prior to each assignment of $R_{1,g,s}$. Also note that when evaluating $R_{1,g,s}(\vec{N})$, successive estimates for $R_{1,g,s}(N)$ are averaged to assure convergence, though in some cases it may increase the number of iterations. This is desirable because small changes in $U_h^1(N-1)$ can cause large changes in $B_s(N-1)$ and hence $R_{1,g,s}(N)$. Other weights for averaging are also possible.

$$\begin{aligned} prev &\leftarrow R_{1,g,s}(N) \\ R_{1,g,s}(N) &\leftarrow D\left(1+\frac{Q_{1,g,s}(N-1)}{S} B_s(N-1)\right) \\ R_{1,g,s}(N) &\leftarrow \left(R_{1,g,s}(N)+prev\right)/2. \end{aligned}$$

V. A SAMPLE APPLICATION

We now consider two examples in which the method of layers is used to predict the performance of a system supporting a complex software application. Several configurations of the system are considered that would be of interest to the system's performance analyst or manager. Performance estimates for the application are compared with simulated measures to demonstrate the accuracy of the technique.

The system supports two classes of users that request information from a database with data that spans two disks. Its software process architecture is shown in Fig. 1. Each user is supported by a user interface process in either *Group*1 or *Group*2, respectively. These processes request services from application processes in groups $A1$ and $A2$, and, send information to an external system via a communications process in group *COM*. Processes in $A1$ and $A2$ perform the users' requested operations on the database. Disk process groups $DP1$ and $DP2$ manage access to the data stored on the disks. The *COM*, $DP1$, and $DP2$ groups each contain a single process. Each of the serving groups serves its customers in a first-come-first-served manner. In general, our purpose is to find a system configuration that keeps *Group*1 and *Group*2 average response times as low as possible.

In the first example, we consider ten class 1 users and between five and ten class 2 users. The number of class 2 users depends on the time of day and distinguishes six performance periods for the system. Thus the number of *Group*1 processes N_{Group1} is ten and the number of *Group*2 processes N_{Group2} is between five and ten with a separate model for each of the population levels.

We investigate the impact of several changes in process allocation on application performance. At first we consider a two processor system with each process allocated to a specific processor. By altering the allocation of processes to processors to balance the load, we show that application performance can be improved. Next, the number of application processes in group *A*1 is increased. This provides users in class 1 with greater access to the system's resources. Naturally, this degrades the performance of class 2 users. Lastly, each of the *Group*1 and *Group*2 processes is allocated to its own dedicated processor. This is done to predict the impact on performance of evolving the system to a more distributed environment, such as a local area network.

In the second example, *Group*1 and *Group*2 each have ten processes. We investigate the impact of altering the number of group *A*2 processes. Here the MOL is used to help decide how many group *A*2 processes should be supported by the application. If there are too few, *Group*2 response times become too high, and, if there are too many, *Group*1 response times become too high. The software process architecture for example 2 is shown in Fig. 3. Device contention models for the examples are illustrated in Fig. 4.

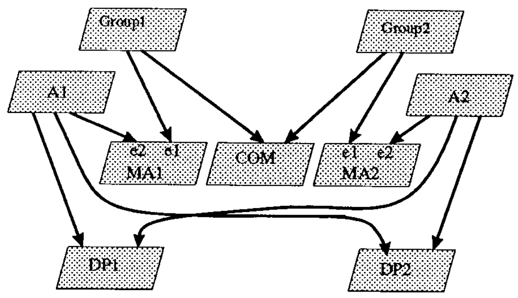


Fig. 3. A multiserver software process architecture for example 1 scenarios 3 and 4 and example 2.

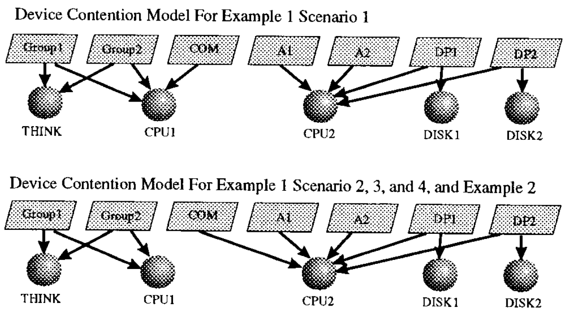

Device Contention Model For Example 1 Scenario 1
Group1
Group2
COM
A1
A2
DP1
DP2
THINK
CPU1
CPU2
DISK1
DISK2
Device Contention Model For Example 1 Scenario 2, 3, and 4, and Example 2
Group1
Group2
COM
A1
A2
DP1
DP2
THINK
CPU1
CPU2
DISK1
DISK2

Fig. 4. Device contention models for examples 1 and 2.

In the first example, the processors are highly utilized. Changes in the software process architecture have a significant impact on the performance of the system. In the second example, the processors have lower utilzations. Changes to the software process architecture have an even more dramatic effect on system performance.

The relative error in estimated performance values with respect to the simulated performance values is used as the measure of error in predicted performance measures for the models in this paper:

$$\text{error}\,\% = \frac{\text{simulated value} - \text{estimated value}}{\text{simulated value}} \times 100.$$

All simulated average response times have a 95% confidence interval within ± 5% of the reported result.

The initial model parameters for example 1 are given in Table II. They describe *scenario 1* and determine the performance measures for the six population levels of *Group*2. The following successive changes are made to these parameters.

- *scenario 2* The *COM* process is moved from *CPU*1 to *CPU*2. The average number of visits and service time remain the same.
- *scenario 3* The number of *A*2 processes is changed from one to three.
- *scenario 4* *Group*1 and *Group*2 processes are allocated to their own dedicated processor.

TABLE IIA
EXAMPLE 1: GROUP AND PROCESSOR SCHEDULING DISCIPLINES AND DESCRIPTIONS

Entity g	Scheduling Discipline ψ_g	Population N_g	Description
*Group*1	Non-Serving	10	Customer Class 1
*Group*2	Non-Serving	5 – 10	Customer Class 2
*A*1	FCFS	1	Application 1
COM	FCFS	1	Communications Process
*A*2	FCFS	1	Application 2
*DP*1	FCFS	1	Disk Process 1
*DP*2	FCFS	1	Disk Process 2
*CPU*1	PS	1	Processor 1
*CPU*2	PS	1	Processor 2
*Disk*1	PS	1	Disk 1
*Disk*2	PS	1	Disk 1
Think	DELAY	1	Customer Think Time

TABLE IIB
EXAMPLE 1: LEVEL 3 DEMANDS

Entity	V_{A1}	V_{A2}	V_{COM}	V_{CPU1}	S_{CPU1}	V_{Think}	S_{Think}
*Group*1	1	0	2	5	0.05	1	3
*Group*2	0	2	3	7	0.30	1	4

TABLE IIC
EXAMPLE 1: LEVEL 2 DEMANDS

Entity	V_{DP1}	V_{DP2}	V_{CPU1}	S_{CPU1}	V_{CPU2}	S_{CPU2}
*A*1	0.3	0.4	0	0	1.7	0.1
COM	0	0	1	0.05	0	0
*A*2	0.5	0.6	0	0	2.1	0.2

TABLE IID
EXAMPLE 1: LEVEL 1 DEMANDS

Entity	V_{CPU2}	S_{CPU2}	V_{Disk1}	S_{Disk1}	V_{Disk2}	S_{Disk2}
*DP*1	2	0.1	1	0.1	0	0
*DP*2	2	0.05	0	0	1	0.15

The average response times of the *Group*1 and *Group*2 processes are shown for each scenario in Fig. 5 and Fig. 6, respectively. The relative error of each estimated response time with respect to simulation is also illustrated in the figures. Similarly, the utilizations of *CPU*1 and *CPU*2 are given in Fig. 7 and Fig. 8.

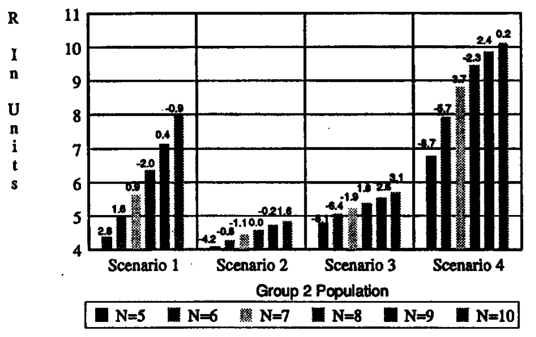


Fig. 5. Example 1: Estimated Group1 average response times and percentage relative error with respect to simulation.

In scenario 1, *CPU*1 is fully loaded, while the utilization of *CPU*2 is around 0.60. This prompts us to shift a process from *CPU*1 to *CPU*2 to balance the load. *COM* is chosen because its utilization of *CPU*1is approximately 0.20. *Group*1 and *Group*2 use more than 0.30 of *CPU*1, shifting them would cause another imbalance. Shifting the *COM* process to *CPU*2 leads us to scenario 2 and is accomplished by letting $V_{COM,CPU1} = 0$, $S_{COM,CPU1} = 0$, $V_{COM,CPU2} = 1$, and $S_{COM,CPU2} = 0.05$.

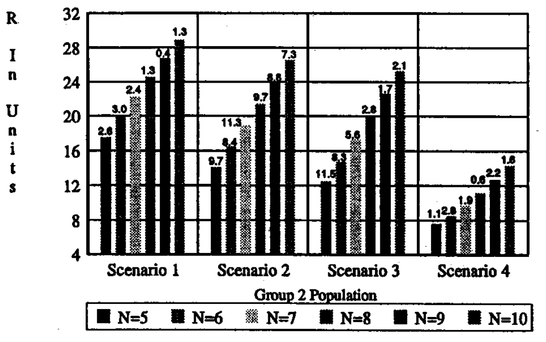


Fig. 6. Example 1: Estimated Group2 average response times and percentage relative error with respect to simulation.

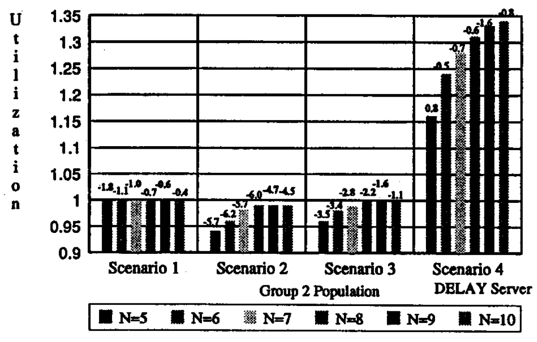


Fig. 7. Example 1: Estimated Processor1 utilization and percentage relative error with respect to simulation.

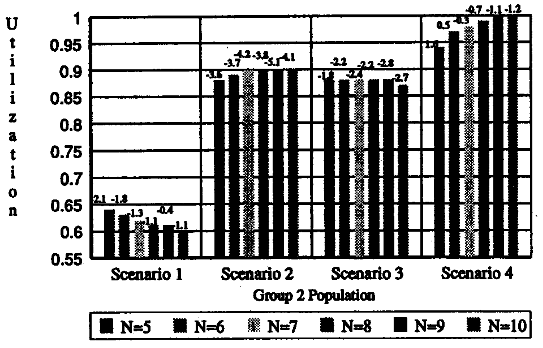


Fig. 8. Example 1: Estimated Processor2 utilization and percentage relative error with respect to simulation.

In scenario 2, the utilizations of *CPU*1and *CPU*2 are more balanced, and the average response times of the *Group*1 and *Group*2 processes benefit. *Group*1 processes are less sensitive to the increase in the number of *Group*1 processes than in scenario 1. Scenario 2 better meets our goal of achieving low *Group*1 and *Group*2 average response times.

An attempt is made to improve the response times of *Group*2 processes in scenario 3. Multiservers are introduced for application processes *A*1 and *A*2 and the number of group *A*2 processes is changed from one to three. The new software process architecture for the model is shown in Fig. 3. The changes in model parameters from scenario 2 are given in Table III.

TABLE III
EXAMPLE 1: SCENARIO 3 CHANGES TO MODEL PARAMETERS FROM SCENARIO 2

Entity	Population	$V_{MA1,e1}$	$V_{MA1,e2}$	$V_{MA2,e1}$	$V_{MA2,e2}$	V_{A1}	V_{A2}
Group1	n/a	1	0	0	0	0	0
Group2	n/a	0	0	1	0	0	0
MA1	1	0	0	0	0	0	0
MA2	1	0	0	0	0	0	0
A1	1	0	1	0	0	0	0
A2	3	0	0	0	1	0	0

The *Group*2 response times improve between 5% and 15%. But this improvement is at the expense of *Group*1 response times which degrade consistently by 15%. These changes occur even though the processor utilizations do not change by more than a few percent. Thus the Multiserver can be used to influence the allocation of system resources to different workloads. This is most manageable when each Multiserver serves a specific workload.

Scenario 4 is similar to scenario 3 except that we allocate each *Group*1 and *Group*2 process to its own processor. The change in the model is accomplished by making *CPU*1 a DELAY center. This represents the offloading of *Group*1 and *Group*2 processing to user workstations, while treating the remaining portion of the model as a serving system. The result is an almost 50% drop in *Group*2 response times. However, the *Group*1 response times increase and become much more sensitive to increases in *Group*2 population levels. This is because *Group*1 processes face more competition from *Group*2 processes in the serving subsystem than in scenario 3.

In example 2, scenario 4's average service demands and think times are altered to achieve lower processor utilizations. The new parameters are shown in Table IV. The number of *Group*1 and *Group*2 processes is set at ten, and the number of application *A*1 processes is increased from one to ten. Assuming that we wish to decrease *Group*1 response times, the method of layers can be used to find an appropriate choice for the number of *A*1 processes in the system. The average response times of the *Group*1 and *Group*2 processes for these ten cases are given in Fig. 9. As expected, as the number of *A*1 processes is increased, the average response times of *Group*1 processes decreases, and the average response times of *Group*2 processes increase. In Fig. 10, we see that the utilization of the most heavily loaded device changes by only 6% over the ten cases but group response times change by a factor of three. An appropriate choice for the number of application *A*2 servers for the system in example 2 would appear to be five. Adding more *A*1 processes does not significantly lower *Group*1 response times, but degrades *Group*2 response times needlessly.

TABLE IVA
EXAMPLE 2: GROUP AND PROCESSOR SCHEDULING DISCIPLINES AND DESCRIPTIONS

Entity g	Scheduling Discipline ψ_g	Population N_g	Description
*Group*1	Non-Serving	10	Customer Class 1
*Group*2	Non-Serving	10	Customer Class 2
*MA*1	FCFS Multi-Server	1	Customer Class 2
*MA*2	FCFS Multi-Server	1	Customer Class 2
*A*1	Server for Multi-Server	1 – 10	Application 1
COM	FCFS	1	Communications Process
*A*2	Server for Multi-Server	3	Application 2
*DP*1	FCFS	1	Disk Process 1
*DP*2	FCFS	1	Disk Process 2
*CPU*1	DELAY	1	Processor 1
*CPU*2	PS	1	Processor 2
*Disk*1	PS	1	Disk 1
*Disk*2	PS	1	Disk 1
Think	DELAY	1	Customer Think Time

TABLE IVB
EXAMPLE 2: LEVEL 3 DEMANDS

Entity	V_{A1}	V_{A2}	V_{COM}	V_{CPU1}	S_{CPU1}	V_{Think}	S_{Think}
*Group*1	1	0	2	5	0.05	1	8
*Group*2	0	2	3	7	0.30	1	30

TABLE IVC
EXAMPLE 2: LEVEL 2 DEMANDS

Entity	V_{DP1}	V_{DP2}	V_{CPU1}	S_{CPU1}	V_{CPU2}	S_{CPU2}
*A*1	0.3	0.4	0	0	1.7	0.1
COM	0	0	1	0.05	0	0
*A*2	0.5	0.6	0	0	2.1	0.2

TABLE IVD
EXAMPLE 2: LEVEL 1 DEMANDS

Entity	V_{CPU2}	S_{CPU2}	V_{Disk1}	S_{Disk1}	V_{Disk2}	S_{Disk2}
*DP*1	2	1	1	2	0	0
*DP*2	2	0.5	0	0	1	2.2

In this example the processors were not saturated, increasing the number of servers not only altered the allocation of resources, but increased the processor utilizations within the system as well.

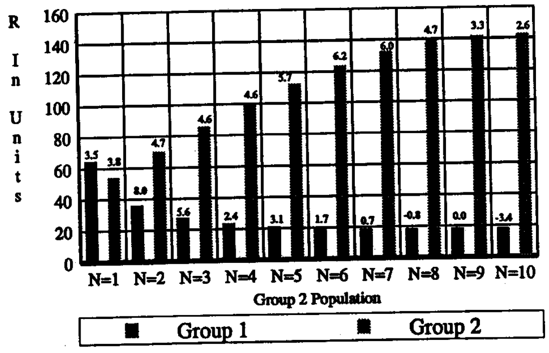


Fig. 9. Example 2: Estimated group average response times and percentage relative error with respect to simulation.

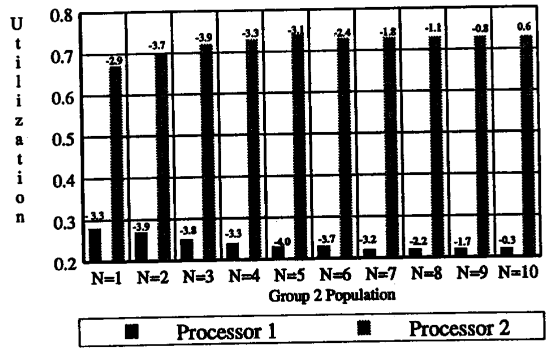


Fig. 10. Example 2: Estimated processor utilization and percentage relative error with respect to simulation.

VI. CONCLUSIONS

The MOL can be used to predict the performance of systems that contain software servers. In the MOL, the LQM submodels are solved using a version of Linearizer that has been enhanced to support several new residence time expressions. Linearizer is a well known MVA based algorithm and the MOL benefits from its good accuracy and convergence properties. In addition, standard MVA software permits customers with statistically identical behaviour to be represented using a single customer class. This makes it possible to represent non-serving processes that have statistically identical behaviour as a single group in an LQM.

Several new techniques for representing process interactions have been developed [26]. They can be used in conjunction with the MOL to model the behaviour of software systems. Each of the techniques is implemented as a type of server that addresses a specific interaction present in software systems. The MVA algorithm is modified and extended to include appropriate residence time expressions for each new type of server. The MOL can be used to predict the performance of software systems that can be described in terms of the servers that are available.

A Multiserver has been described that permits many serving processes to share a single queue of customers. The Multiserver has two significant effects. It influences the allocation of processing resources to workloads without the use of priorities. Secondly, it increases the level of concurrency in a system and can increase the total utilization of resources. Two examples with Multiservers were used to demonstrate the technique. The examples also show that some changes to a software process architecture have little or no impact on processor utilizations, yet, these same changes can have a significant impact on process response times. This implies that distributed application performance management systems cannot rely on device metrics alone to assess the performance of a system.

The estimate $F_{a,h}$ that is used to modify the arrival instant queue length in the method as presented is optimistic when a server is highly utilized. Too much of the arrival instant queue length is removed. A better approximation would be based on the change in the serving process's utilization caused by the removal of a client's removal from the network. This is the subject of future work.

ACKNOWLEDGMENT

This work has been supported by the Natural Sciences and Engineering Research Council of Canada.

REFERENCES

[1] S.C. Agrawal and J.P. Buzen, "The aggregate server method for analyzing serialization delays," *ACM Trans. on Computer Systems*, vol. 1, no. 2, pp. 116-143, May 1983.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*. Reading, Mass.: Addison-Wesley, 1983.

[3] G.R. Andrews and F.B. Schneider, "Concepts and notations for concurrent programming," *Computing Surveys*, vol. 15, no. 1, pp. 3-43, Mar. 1983.

[4] F. Baskett, K.M. Chandy, R.R. Muntz, and F.G. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *JACM*, vol. 22, no. 2, pp. 248-260, Apr. 1975.

[5] B. Beizer, *Micro-Analysis of Computer System Performance*. New York: Van Nostrand Reinhold, 1978.

[6] B. Beizer, "Software performance," C.R. Vicks and C.V. Ramamoorthy, eds., *Handbook of Software Engineering*. New York: Van Nostrand Reinhold, pp. 413-436, 1984.

[7] *Best/1 User's Guide*. Waltham, Mass.: BGS Systems, Inc., 1982.

[8] R.M. Bryant, A.E. Krzesinski, M.S. Lakshmi, and K.M. Chandy, "The MVA priority approximation," *ACM Trans. on Comp. Systems*, vol. 2, no. 4, pp. 335-359, Nov. 1984.

[9] R.J. Buhr, G.M. Karam, C.J. Hayes, and C.M. Woodside, "Software CAD: A revolutionary approach," *IEEE Transactions on Software Engineering*, vol. 15, no. 3, pp. 235-249, Mar. 89.

[10] K.M. Chandy and D. Neuse, "A heuristic algorithm for queueing network models of computing systems," *CACM*, pp. 126-133, Feb. 1982.

[11] G. Franks, A. Hubbard, S. Majumdar, D. Petriu, J. Rolia, C.M. Woodside, "A toolset for performance engineering and software design of client-server systems," *Performance Evaluation J.*, June 1995.

[12] *Introduction to OSF DCE*. Englewood Cliffs, N.J.: Prentice Hall, 1992.

[13] *IEEE Portable Operating System Interface (POSIX), Lightweight Threads*, IEEE std. P1003.4a.

[14] P.A. Jacobson and E.D. Lazowska, "Analysing queueing networks with simultaneous resource possession," *CACM*, pp. 141-152, Feb. 1982.

[15] L. Kleinrock, *Queueing Systems*, vol. 1: *Theory*. New York: John Wiley & Sons, 1975.

[16] S.S. Lavenberg, "A perspective on queueing models of computer performance," *Performance Evaluation*, vol. 10, no. 1, pp. 53-76, 1989.

[17] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Englewood Cliffs, N.J.: Prentice-Hall, 1984.

[18] *MAP User's Guide*. Seattle, Wash.: Quantitative System Performance, Inc., 1982.

[19] J.W. Miernik, C.M. Woodside, J.E. Neilson, and D.C. Petriu, "Throughput of stochastic rendezvous networks with caller-specific service and processor contention," *Proc. of IEEE InfoCom*, pp. 1,040-1,049, 1988.

[20] J.W. Miernik, C.M. Woodside, J.E. Neilson, and D.C. Petriu, "Performance of stochastic rendezvous networks with priority tasks," Technical Report. Carleton University, SCE-89-02. Presented at the Int'l Seminar on Performance of Distributed and Parallel Systems, Dec. 1988, Kyoto, Japan.

[21] Object Management Group and Xopen, *The Common Object Request Broker: Architecture and Specification*. Framingham, Mass., and Reading Berkshire, UK: Object Management Group and Xopen, 1992.

[22] Bin Qin, "A model to predict the average response time of user programs," *Performance Evaluation*, vol. 10, pp. 93-101, 1989.

[23] M. Reiser and S.S. Lavenberg, "Mean value analysis of closed multichain queueing networks," IBM Research Report RC 70 23, Yorktown Heights, N.Y., 1978.

[24] M. Reiser, "A queueing network analysis of computer communication networks with window flow control," *IEEE Transactions on Communications*, pp. 1,201-1,209, Aug. 1979.

[25] J.A. Rolia, "Performance estimates for multi-tasking software systems," Master's Thesis, Univ. of Toronto, Canada, Jan. 1987.

[26] J.A. Rolia, *Software Performance Modelling*, CSRI Technical Report 260, PhD Dissertation, Univ. of Toronto, Canada, Jan. 1992.

[27] E. de Souza e Silva and R.R. Muntz, "A note on the computational cost of the linearizer algorithm for queueing networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 840-842, June 1990.

[28] C.U. Smith, "The prediction and evaluation of the performance of software from extended design specifications," PhD Dissertation, Univ. of Texas at Austin, University Microfilms Pub. No. KRA81-00963, Aug. 1980.

[29] C.U. Smith, *Performance Engineering of Software Systems*. Reading, Mass.: Addison Wesley, 1990.

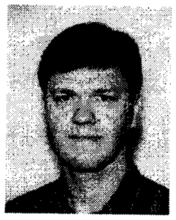
[30] K.S. Trivedi, *Probability and Statistics with Reliability, Queueing, and Computer Science Applications*. Englewood Cliffs, N.J.: Prentice-Hall 1982.

[31] V. Vetland, "Measurement-based composite computational work modelling of software," PhD thesis, Norwegian Inst. of Technology, Univ. of Trondheim, Sept. 1993.

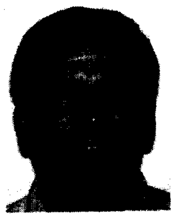
[32] C.M. Woodside, *Throughput Calculation For Basic Stochastic Rendezvous Networks*, Technical Report, Carleton Univ., Ottawa, Canada, Apr. 1986.

[33] C.M. Woodside, "Throughput calculation for basic stochastic rendezvous networks," *Performance Evaluation*, vol. 9, 1989.

[34] C.M. Woodside, E.M. Hagos, E. Neron, and R.J.A. Buhr, "The CAEDE performance analysis tool," *Ada Letters*, vol. 11, no. 3, Spring 1991.



Jerome A. Rolia holds computer science degrees from Carleton University (BCS, 1985) and the University of Toronto (MSc 1987, PhD 1992). He is an assistant professor in the Department of Systems and Computer Engineering at Carleton University in Ottawa. His research interests include performance measurement and modeling, midware and distributed application systems, the performance management of distributed application systems, and software design for performance. Dr. Rolia is a project leader in the software thrust of the Telecommunications Research Institute of Ontario and a principal investigator in the Management of Distributed Applications and Systems (MANDAS) project supported by IBM Canada and the Natural Sciences and Engineering Research Council of Canada.



Kenneth C. Sevcik holds degrees from Stanford (BS, mathematics, 1966) and the University of Chicago (PhD, information science, 1971). He is a professor of computer science with a cross-appointment in electrical and computer engineering at the University of Toronto. He is director of the Computer Systems Research Institute and past chairman of the Department of Computer Science. His primary area of research interest is in developing techniques and tools for performance evaluation and applying them in such contexts as distributed systems, database systems, local area networks, and parallel computer architectures. Dr. Sevcik served for six years as a member of Canada's Natural Sciences and Engineering Research Council, the primary funding body for research in science and engineering in Canada. He is coauthor of the book, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, and codeveloper of MAP, a software package for the analysis of queueing network models of computer systems and computer networks.