APPROXIMATE AND SIMULATION BASED ANALYSIS FOR DISTRIBUTED OBJECT SOFTWARE PERFORMANCE MODELS

Panajotis Katsaros Constantine Lazos

Department of Informatics
Aristotle University of Thessaloniki
54006 Thessaloniki
Greece
{katsaros, clazos}@csd.auth.gr

ABSTRACT

In complex software systems, the effectiveness of model based performance predictions is limited by the availability of appropriate solution techniques. These techniques should allow to take into account the software components interaction effects. In distributed object systems, the main problem is the simultaneous resource possession caused by the synchronous, often nested object invocations, which block the callers, until they get the replies. This paper provides a review of the analysis techniques, which address that fact, while preserving the abstract system view, offered by a queuing network representation. Two of these techniques, were proposed for solving a general class of models, with one or more layers of software servers and a third technique was designed specifically for distributed object software performance models. The advent of an extended flow-equivalent approximation, which is also described, opens new prospects for the development of efficient solution algorithms. Finally, simulation based estimation is discussed, in respect with the applicability of the well-founded and accurate, single-run regenerative method.

KEYWORDS: synchronous object invocation, nested object invocations, simultaneous resource possession, approximate MVA, flow-equivalent approximation, regenerative method

1. INTRODUCTION

Classical queuing network solution techniques, assume that a job can only use a single resource at a time and simultaneous resource possession is not possible. Moreover, most of them may be only used for analyzing typical "flat" system representations and do not directly support the solution of excessively complex queuing networks, like the ones obtained in realistic performance models of distributed software systems.

Early contributions to the problem of analyzing simultaneous resource possession, in complex hierarchical performance models, were based either on

- iterative estimation techniques with unknown calculation time and not guaranteed convergence for the performance measures of interest or
- low-level Markov based approximations that are neither practical nor scalable enough for software engineering purposes.

Recent advances, like the advent of an extended flow-equivalent approximation ([KK99]), constitute a new theoretical basis for the development of efficient solution algorithms.

In this paper, we review those techniques that we believe, they may have an impact in the practice of distributed object systems performance modeling and we comment on the new prospects opened by the last developments.
Accurate simulation based analysis of such complex models is an absolute necessity, since it is basically the only alternative for checking the validity of the approximations to be used. Thus, we also review theoretical results on the applicability of the single-run regenerative approach, in hierarchical queuing network models.

2. ITERATIVE PERFORMANCE EVALUATION ALGORITHMS

2.1 *The Stochastic Rendezvous Networks*

The Stochastic Rendezvous Networks (SRVNs) were first introduced in [WOO89] and constitute a practical and scalable technique for the analysis of systems, where software components interact with the rendezvous mechanism. The rendezvous mechanism can be used to model different interaction types present in software, including all the types of object invocations ([WNPM95]) taking place in distributed object systems. The solution for a SRVN can be found with an iteration algorithm that computes a series of intermediate solutions, using an MVA approximation and continues until the estimated throughputs converge.
The primitive structural entity of a SRVN model is the task. Each task may represent a single software or hardware component and communicates with other tasks by messages of a request-wait-reply style that mimics the rendezvous interaction mechanism. The task, which sends the message (*client* task), requests the rendezvous and blocks until it gets the reply. The receiving task (*server*) accepts the request and executes two (or more) service phases. The client task is blocked only for the time period of the first phase. Thus, a synchronous object invocation may be modeled by a rendezvous request with zero service requirements, for its second phase and an asynchronous (one-way) object invocation, by a rendezvous request with zero service requirements, for its first phase.
A SRVN differs from a typical queuing network, in that a server may act as a client requesting service from one or more tasks of a lower layer, in any phase of a rendezvous service execution. A server may provide more than one service, modeled as separate *task entries*, with their own performance parameters. However, each task owns a single message queue with a specific service discipline. Entries correspond to distinguished object method invocations, since messages are directly addressed to the entries and not to the tasks.
There are three types of tasks, namely:

- the *pure clients*, which may only initiate requests and in this sense act as the jobs traversing a queuing network,
- the *active servers*, which may accept requests and initiate new ones and
- the *pure servers*, which may only accept requests and provide service to them.

Processors are pure server pseudo-tasks that provide hardware resource service to one or more active server tasks. The entries of the tasks, assigned to it, request service one slice at a time and each request and service are modeled in the SRVN as a rendezvous between the requesting task and the processor pseudo-task.
The SRVN solution algorithm, proposed in [WNPM95], is summarized in the following steps:

1. Carry out the software model transformation to create tasks that represent processors, which correspond to co-allocated tasks.
2. Construct the so-called *task request graph*, with a node for each task and a directed arc from node *i* to node *j*, if the entry *e* in task *i* sends requests to any entry *d* in task *j*. This graph is used to determine the order of computation in the next steps of the algorithm. If there is a cycle in the task request graph it is not possible to determine such an ordering and the analysis cannot proceed to the next steps.
3. Initially, set 0 the mean waiting times seen by all entry requests, when sending messages to other entries.
4. Calculate all the mean service times (including the queuing delays at tasks and processors) and the throughputs, in the order found at step 2, by using the following inputs:

the parameters specifying the arrival rates, if there are external streams of requests directed to specific entries
the parameters specifying the mean number of entry requests directed to other entries, during each phase
the parameters denoting the mean total service time provided by each entry, during each phase
the mean round-trip communication delays from task to task, if any

5. Estimate the mean waiting times for all entry requests directed to other entries, based on the MVA expression provided in [WNPM95] and the alternative approximations proposed, in order to determine the task queue properties, at the instant of the rendezvous request arrival.
6. If throughputs are all sufficiently close to the previous iteration, then stop. Otherwise go to step 4.

The results obtained include throughputs, mean waiting times, entry service times and entry and task utilizations. The described solution technique is closely related to the Bard-Schweitzer approximate MVA algorithm [SCH79] and the heuristics provided in [WNPM95] support either FIFO or Preemptive Priority scheduling and stochastic or deterministic phase types, with possibly varied coefficients of variation, in the second case.
An alternative task-directed aggregation algorithm [PET94] replaces most of the arrival-instant probability estimates of the described algorithm, with estimates derived from the analysis of an aggregated Markov chain model of the task interactions. The modified algorithm produces more accurate results for multiple entry tasks, with high service time variation across entries and large differences in request inter-arrival times. It can be only used under FIFO scheduling and it does not provide support for external streams of requests.
Finally, the SRVN based solution variants cannot be applied to models with multi-threading tasks.

2.2 *The Method of Layers*

In the method of layers [RS95], the system performance model is viewed as a sequence of layers, where processes request service only from one level lower, in the hierarchy. Processes with statistically identical behavior form a *group* or a *class of processes*. To identify the level of each group, in the hierarchy, a topological sort of the process groups may be first developed. Thus, each group is placed at exactly one level and the resulted graph is assumed not to contain any cycles. If requests for service span more than one level, virtual flow-equivalent groups modeling the requests and their service rates have to be introduced, in all intermediate layers (this is not required for the SRVNs based solution techniques).
Each pair of successive levels, in the hierarchy, defines a submodel. The response time of a process that is considered as a client in one submodel defines its service time, when it is considered as a server in its alternate submodel. The performance estimates for the submodels are found using a modified version of the Linearizer algorithm [CN82]. The solution of all of the submodels represents an iteration of the algorithm. The algorithm is applied iteratively, until the changes in estimated mean response times for processes between successive iterations is below a specified tolerance.
When solving the hierarchy of models only software contention is taken into account. After the algorithm terminates, a second model, in which software contention is ignored, is created. Each process is included in a single hardware resource contention queuing network and the amount of time that a process does not spend competing for devices is used as a think time in the model.
With new estimates of device responsiveness, another solution for the software contention model is found. The method of layers alternates between software and hardware contention models, until the estimates for mean response times of non-serving processes differ by less than some tolerance. This approach is similar to the method of complementary delays ([HT83]) that proceeds by using closed queuing networks, augmented by a virtual delay service center.
The method of layers supports the use of *multiple entries* FIFO rendezvous servers, as in the case of the SRVNs based solution techniques. Additionally, the use of the so called *multi-servers* permit the representation of multiple thread servers (as opposed to the SRVNs based solution techniques).
The combination of the use of the Linearizer, which often provides superior accuracy, when compared to the Bard-Schweitzer algorithm and the simultaneous solution of all servers in a layer yields, in some

cases, more accurate results, compared to the ones obtained by the SRVN based solution variants. In the sequel, the basic method of layers estimation is outlined:

```
1. Initialize the response time estimates for process groups, with the mean service times,
   assuming no hardware or software contention. The initial response times are computed level
   by level in a bottom-up manner.
2. WHILE group response times have not reached a fixed point DO
           WHILE group response times have not reached a fixed point DO
                   FOR each software submodel selected in a top-down manner DO
                           solve the submodel using Linearizer with the following
                           residence time expressions ([RS95]):
                                   FIFO, rendezvous, multiple entry, multi-server and delay
                   END FOR
           END WHILE
           solve the hardware resource contention submodels using Linearizer with the
           following residence time expressions ([RS95]):
                   PS, FIFO, LIFO, priority preemptive resume, and delay
           Update the group response time estimates
   END WHILE
```

2.3 *The Method of Decomposition*

The method of decomposition, introduced in [KAH00], was designed specifically for distributed object software performance models. It includes approximations, which overcome the limitation of not being applicable (as in the case of the methods already described) in models with cyclic graph dependencies. Cycles may be introduced, when two objects are allowed to make synchronous invocations to each other, as for example in the case of the callback interaction pattern. This type of interaction is sometimes used as a means to avoid extensive blocking times that may be caused by the usual single synchronous object invocations. Recursive invocations to the same object are also allowed, as this is possible in real world applications.

In the sequel, we provide a description of the core solution technique. The overall approach lies on the well-known method of surrogate delays ([JL82]), in order to decompose the model into multiple queuing networks, so that the primary and the secondary (software or hardware) resources of a blocking access are always in different networks. As a result, we obtain a set of multi-chain product form queuing networks.

For each case of simultaneous resource possession, the primary network is created by removing all secondary resources and all accesses to them and then by including a surrogate delay resource. The following transformations are carried out:

- for each access to the primary resource, its service demand is increased by the sum of the response times obtained for all blocking invocations to the secondary resources
- for each chain (class), the service demand of the surrogate delay server is defined to be the sum of the response times for all non blocked accesses of the secondary resources used by the chain

For each primary network, a set of secondary networks is created. These networks contain the secondary resource itself and an auxiliary delay server, for modeling the time the jobs are spending elsewhere in the system. Each chain in the original queuing network generates at most two secondary chains:

- a closed one, if there are blocking accesses, with its population defined as the maximum number of accesses that can reach the secondary resource in parallel (multithreading server) and
- a secondary chain that corresponds to a chain with non blocking accesses to the resource, with the same parameters (equal arrival rates in case of open chains and equal populations in case of closed ones)

The service demands for all non blocking and blocking secondary resource accesses remain the same, unless there are nested blocking accesses to other resources. In this case, the service demand is increased by the sum of the response times obtained for the nested accesses. For closed secondary

chains the service demand imposed to the auxiliary delay server is calculated through the Little' s rule, by using the throughput of the calling resource, in the corresponding primary network. For open secondary chains there is no need to specify a service demand for the auxiliary delay server.
The resulted networks are to be solved by an approximate Bard-Schweitzer MVA algorithm, modified appropriately (by the load concealment transformation, as described in [AGR85]) for the simultaneous solution of the open and the closed chains. Input parameters for some networks require the existence of a solution for some other networks and it is even possible to obtain cyclic dependencies. Thus, the solution is carried out iteratively and in each repetition the input parameters of the networks are adjusted to the outcome of the previous repetition, until the throughputs of the secondary resources are sufficiently close to the throughputs of the corresponding primary ones.
To conclude, the method of decomposition does not support priority scheduling, as opposed to the SRVNs based solution, proposed in [WNPM95] and the method of layers.

3. EXTENDED FLOW-EQUIVALENT APPROXIMATION

The standard decomposition or flow-equivalent method ([COU77] and [CHW75]) is a well known approximation, based on the replacement of a subset of the model's queuing centers, with a single, flow-equivalent center, characterized by a calculated state-dependent service rate.
Flow-equivalent based performance estimation techniques have been used in various simultaneous resource possession models ([SAU81]). However, these methods could not be applied to models, where jobs that possess different types of passive resources (i.e. different sets of blocked object servers, as in the case of synchronous nested object invocations) compete with other jobs for the same active resources.
Recently, in [KK99], the authors introduced an extended flow-equivalent approximation, for overcoming this particular restriction. The proposed approximation is described in the frame of a general two-layer queuing network, where the upper layer represents software resource contention and the lower layer, the caused hardware resource contention (Figure 1).

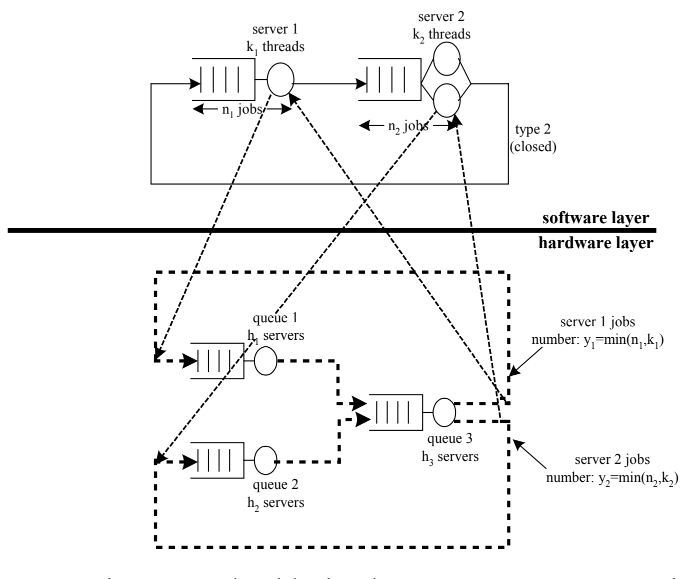

Figure 1 A two-layer queuing network with simultaneous resource possession

The existence of a single upper layer does not provide adequate support for the representation of nested object invocations, where multiple object servers may be blocked at the same time. However, the advent of such an extended flow-equivalent approximation opens new prospects for the development of efficient non-iterative estimation techniques and for this reason, we proceed to the description of it.

Let us consider a queuing model, where the upper layer consists of R FIFO software servers labeled as 1, 2, . . . , R and the lower layer consists of N queues representing hardware resources that are labeled as 1, 2, . . . , N. Each software server owns k_i (≥1) threads, $i \in 1, \ldots, R$ and one queue.
A job (object invocation) that arrives at server i will immediately be routed to the lower layer upon its arrival, if there is an available thread at the server. The time period in which a job passes to and returns from the lower layer is referred to as the server's service time. Each queue $j \in 1, \ldots, N$ in the lower layer owns h_j (≥1) servers and one queue. Each job routed to the lower layer from software server i will travel through the queues at that layer, according to a specified Markovian routing chain, associated with the software server i. Subsequent routing of a job (object invocation) through the various software servers, in the upper layer, can be conducted according to a number of possible (open or closed) routing chains, that represent the desired software functionality. In this article, our description, for the upper layer, is restricted to the simple case of a single closed routing chain, with v_i denoting the relative frequency for a job to visit the software server i.
Let $x_{j(i)}$ be the total number of jobs of software server i, placed at the lower layer queue, j. The number of jobs of software server i, in the lower layer is given as,

$$y_i = \sum_{j=1}^{N} x_{j(i)}$$

and the aggregate queue occupancy vector is defined as

$$\mathbf{y} = (y_1, y_2, \ldots, y_R)$$

If n_i is the total number of jobs (object invocations) at the software server i, then $y_i = \min(k_i, n_i)$.
For any job arriving at the queue j, the required service time will depend on the particular software server the job comes from and is subject to an exponential distribution with parameter $\mu_{j(i)}$. The service discipline at each lower layer queue can either be FIFO, PS (Processor Sharing) or LIFOPR (LIFO Preemptive Resume). If the discipline is FIFO, the parameters $\mu_{j(i)}$ are the same for any i, i.e. $\mu_{j(i)} = \mu_{j(h)}$, for any $h \neq i$.
The extended flow-equivalent method, proposed in [KK99], is summarized in the following steps:

1. Calculate the numbers of jobs $\mathbf{y} = (\min(n_1, k_1), \ldots, \min(n_R, k_R))$ in the lower layer, for all possible software server vectors, $\mathbf{n} = (n_1, n_2, \ldots, n_R)$.
2. Each **y**-job based model is created from the original two-layer model, by disregarding the behavior of jobs in the upper layer and setting a short circuit at each software server, so that each job arriving at the server from the lower layer returns immediately to it. Thus, each short-circuited server can be effectively replaced by a state-dependent virtual service center, whose service rate is calculated in the following manner:
    a. Solve the shaped **y**-job based models, for all **y**, by using the convolution algorithm, in order to obtain the normalization constants $G(\mathbf{y})$.
    b. Approximate the service rate $\mu_i^*(\mathbf{n})$ at the software server i as

$$\mu_i^*(\mathbf{n}) = v_i \cdot \frac{G(\mathbf{y} - \mathbf{e}_i)}{G(\mathbf{y})}$$

where $\mathbf{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$, with the unit placed in the i-th position.
In [KK99], the authors also propose a Markov based approximation and an alternative closed form approximation, for the upper layer, which we think are inappropriate for the performance evaluation of nested object invocations. Thus, we omit their description.
The extended flow-equivalent approximation of [KK99] may be used as is, in the hybrid simulation setting, in the way described in the next section. On the other side, we continue working to take advance of it, in order to provide a novel estimation technique that will successfully address the needs for efficient performance prediction, under the existence of synchronous, nested object invocations, in the software layer. The software servers' workload derivation procedure, to be used, will be based on the one introduced in [VHS93].

4. SINGLE-RUN SIMULATION BASED PERFORMANCE EVALUATION

The development of simulation-based queuing models for object based distributed software systems is discussed in [KL02]. Alternatively, [BBR02] introduces the use of XML Petri Nets (PNML), as a means for structured Petri-net model specification, of general distributed heterogeneous systems. In this section, we are restricted to the single-run output analysis of complex hierarchical model configurations, like the ones described.

We propose the use of the so-called regenerative method for the sequentially controlled estimation of steady-state performance measures, for the following reasons:

- it is based on a sound theoretical foundation,
- it is not bound to the problem of the "system initialization bias",
- it is characterized by appealing asymptotic properties,
- its accuracy has been tested both in "flat" ([KL03]), as well as, in hierarchical queuing network models ([BDHL85]) and
- it is also applicable in the hybrid simulation setting, with state-dependent service rates ([SHE93]).

Although this method is not in widespread use, due to an inherent difficulty in identifying regeneration system states, there is significant theoretical progress towards this direction, in the last few years. The most notable advances rely on a generalized semi-Markov (GSMP) system representation and conclude to general results that may be utilized, for the regeneration state selection, at the more abstract level of a queuing network model ([SHE93]). In [KL03], we have experimentally evaluated the effectiveness of alternative selection strategies in the sequentially controlled setting, in terms of both the obtained accuracy and observed efficiency behavior.

5. CONCLUSIONS AND FURTHER RESEARCH WORK

This work is focused on the available approximate solution methods that we believe they may have an impact in the practice of distributed object systems performance modeling. They are all based on iterative estimations that scale well in large software performance models, but are also characterized by unknown calculation time and not guaranteed convergence, for the measures of interest.

The SRVNs based solution techniques lack support for cyclic object invocation dependencies, recursion and multithreading servers, since they were initially designed for the performance analysis of general software systems. The method of layers allows the use of multithreading servers, but requires that processes request service only from one level lower, in the hierarchy. Both approaches offer support for synchronous, as well as asynchronous object method invocations, by the use of the two-phase rendezvous type interaction mechanism and multiple entries software servers.

Priority scheduling is mainly required for the modeling of real-time distributed object systems and is also included as an open possibility for the two aforementioned methods. On the other side, it is not supported by the method of decomposition, which, as opposed to the two other alternatives, allows cyclic object invocation dependencies (callbacks) and recursion.

Although there are still great challenges for improvements in the existing core estimation techniques and the specialized approximation heuristics used by them, the advent of the extended flow-equivalent approximation, described in section 3, opens new prospects for the development of new, efficient solution approaches.

Significant progress has been also carried out in the field of object based model parameter capture ([CHA03] and [VHS93]).

Simulation based analysis for hierarchical model structures like the ones used in the described analysis methods is an absolute necessity, since it is basically the only alternative for checking the validity of the used approximations. The proposed single-run output analysis method is based on a sound theoretical foundation, it is not bound to the problem of the "system initialization bias" and it is also characterized by appealing asymptotic properties. Furthermore, its accuracy has been tested in "flat" ([KL03]), as well as, in hierarchical queuing network models ([BDHL85]) and we have also theoretically verified its applicability in the hybrid simulation setting, with state-dependent service rates.

REFERENCES

[AGR85] Agrawal, S. C. (1985). Metamodeling: A study of approximations in queuing models, MIT Press, Cambridge, MA, USA

[BBR02] Brosso, I., Bressan, G. and Ruggiero, W. V. (2002). Simulation e-business applications using PNML, Proceedings of the 2nd WSEAS International Conference on Simulation, Modeling and Optimization (ICOSMO)

[BDHL85] Blum, A., Donatiello, L., Heidelberger, P., Lavenberg, S. and MacNair, E. A. (1985). Experiments with decomposition of extended queuing network models, In D. Potier (ed.): Modeling Techniques and Tools for Performance Analysis, Elsevier Science, 623-640

[CN82] Chandy, K. M. and Neuse, D. (1982). Linearizer: A heuristic algorithm for queuing network models of computing systems, *Communications of the ACM*, 25, 2, 126-134

[CHW75] Chandy, K. M., Herzog, U. and Woo, L. (1975). Approximate analysis of general queuing networks, *IBM Journal of Research and Development*, 19, 43-49

[CHA03] Chatzigeorgiou, A. (2003). Performance and power evaluation of C++ object-oriented programming in embedded processors, *Information and Software Technology*, 45, 195-201

[COU77] Courtois, P. J. (1977). Decomposability: Queuing and computer system applications, Academic Press

[HT83] Heidelberger, P. and Trivedi, K. S. (1983) Analytical queuing models for programs with internal concurrency, *IEEE Transactions on Computers*, 32, 1, 73-82

[JL82] Jacobson, P. A. and Lazowska, E. D. (1982). Analyzing queuing networks with simultaneous resource possession, *Communications of the ACM*, 25, 2, 142-151

[KAH00] Kahkipuro, P. (2000). Performance modeling framework for CORBA based distributed systems, PhD thesis, Department of Computer Science, University of Helsinki, Finland

[KL02] Katsaros, P. and Lazos, C. (2002). Structured performance modeling and analysis for object based distributed software systems, Proceedings of the 15th ISCA International Conference on Parallel and Distributed Computing Systems, Louisville, USA

[KL03] Katsaros, P. and Lazos, C. (2003). Return state selection for improved effectiveness in sequentially controlled regenerative simulation, (submitted)

[KK99] Kurasugi, T. and Kino, I. (1999). Approximation methods for two-layer queuing models, *Performance Evaluation*, 36-37, 55-70

[PET94] Petriu, D. C. (1994). Approximate mean value analysis of client-server systems with multiclass requests, Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Nashville, Tennessee, 77-86

[RS95] Rolia, J. A. and Sevcik, K. C. (1995). The Method of Layers, *IEEE Transactions on Software Engineering*, 21, 8, 689-700

[SAU81] Sauer, C. H. (1981). Approximate solution of queuing networks with simultaneous resource possession, *IBM Journal of Research and Development*, 25, 6, 894-903

[SCH79] Schweitzer, P. (1979). Approximate analysis of multiclass closed networks of queues, Proceedings of the International Conference on Stochastic Control and Optimization, Amsterdam

[SHE93] Shedler, G. H. (1993). Regenerative Stochastic Simulation, Boston, Academic Press

[VHS93] Vetland, V., Hughes, P. and Solvberg, A. (1993). Improved parameter capture for simulation based on composite work models of software, Proceedings of the 1993 Summer Computer Simulation Conference, Boston, Massachusetts, 110-115

[WOO89] Woodside, C. M. (1989). Throughput calculation for basic Stochastic Rendezvous Networks, *Performance Evaluation*, 9

[WNPM95] Woodside, C. M., Neilson, J. E., Petriu, D. C. and Majumdar, S. (1995). The Stochastic Rendezvous Network Model for performnce of synchronous client-server-like distributed software, *IEEE Transactions of Computers*, 44, 1, 20-34