Efficiency Improvements for Solving Layered Queueing Networks

Greg Franks
greg@sce.carleton.ca

Lianhua Li
lianhua@sce.carleton.ca

Department of Systems and Computer Engineering, Carleton University
Ottawa, ON Canada K1S 5B6

ABSTRACT

Layered Queueing Networks (LQN) have been used successfully by numerous researchers to solve performance models of multi-tier client server systems. A common approach for solving a LQN is to split the model up into a set of submodels, then employ approximate mean value analysis (AMVA) on each of these submodels in an interactive fashion and using the results from the solution of one submodel as inputs to the others. This paper addresses the performance of the layered queueing network solver, LQNS, in terms of submodel construction and in terms of changes to Bard-Schweitzer and Linearizer AMVA, in order to improve performance. In some of the models described in this paper, there is a difference in four orders of magnitude between the fastest and slowest approaches.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques

General Terms

Performance

Keywords

Performance Analysis, Approximate Mean Value Analysis

1. INTRODUCTION

The Layered Queueing Network (LQN) model exploits a key property that a task may, during its service, stop while making a nested request to another server. This type of interaction, a remote procedure call, is very common in the software of today's distributed systems. Conventional product-form queueing networks cannot be used to solve these types of models because of the blocking. The call to the lower level server creates a form of simultaneous resource possession because the customer is occupying the client and it's server during the remote procedure call.

Figure 1 shows a *Layered Queueing Network* (LQN) using a condensed notation. The parallelograms in the figure represents *Tasks*. Tasks are concurrent entities and can make requests to other tasks in the model. Tasks in layer 1 in the figure are called *Reference Tasks* and are the customers in the model. Tasks at lower layers in the model, called *active servers*, can both accept requests and make requests to other tasks and are used to model actual tasks in the system, non-processor hardware devices such as disks, and passive resources such as critical sections. The circles in the figure represent *processors* and are *pure servers* are used to consume time. Both tasks and processors can be *multiservers*. For reference tasks, a multiserver represents a population of customers greater than one.

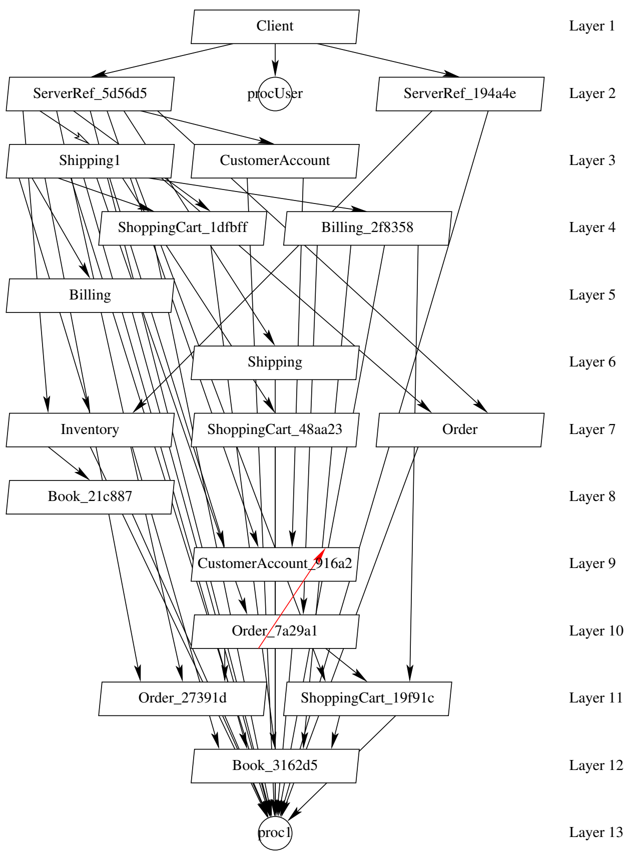

Figure 1: Bookstore model (from [4]).

Calls in Figure 1 are shown as directed arcs from one task to another. Calls can either be synchronous or asynchronous; replies from synchronous requests can be deferred to lower layer servers through forwarding. Layering arises from analyzing the call graph of the requests from the customers in layer one, to the servers (usually processors) at the deepest layer in the graph. A complete description of the model can be found in [3].

Fast analytic solutions for solving a Layered Queueing Network were developed in [9], with Stochastic Rendezvous Networks (SRVN), and in [7] with the Method of Layers (MOL). Stochastic Rendezvous Networks treated each server independently and used new approximations for calculating the residence times at servers. The Method of Layers introduced the important concept of grouping servers in "layer submodels", then solved the submodels using Linearizer approximate Mean Value Analysis (AMVA). Blocking delays caused by synchronous calls are treated as surrogate delays [5] and are imported from other submodels. The Layered Queueing Network (LQN) model is an extension of both MOL and SRVN and incorporates features found in important application systems [3].

Layer submodels can be constructed in a variety of different ways. This paper examines five different layering techniques, ranging from a single server per layer, like the SRVN method in [9], to the extreme opposite case where there is only one layer containing all of the servers in the model. Approximate MVA is used to solve each of the layer submodels so the approximate MVA algorithm was modified to reduce the amount of iteration. The modified algorithms are compared against the original implementation for improvements in performance and variations in the results.

2. LAYERING QUEUEING NETWORK SOLVER

The layered queueing network solver (LQNS) [3] is used in this work. Algorithm 1 lists the major steps used to solve a model. The algorithm iterates over each of the submodels and uses successive substitution to converge on a solution. The algorithm will terminates when the root of the sum of mean difference in utilizations of the entities in the model is less than a user-defined amount or some pre-defined iteration limit is reached.

Algorithm 1 LQNS Algorithm

```
 1: Read input and create LQNS model.
 2: Create $S$ submodels based on layers.
 3: repeat
 4:    for $s \leftarrow 1$ to $S$ do
 5:       Solve submodel $s$ using approximate MVA.
 6:       Set waiting times for submodel $s$.
 7:       Set think times for submodel $s + 1$.
 8:    end for
 9: until convergence or iteration limit
10: Save results.
```

2.1 Submodel Creation

Submodels are constructed by first finding a set of *servers* (starting from layer 2), then finding all of the callers to these servers regardless of layer and treating these entities as *clients*. A task or processor can appear as a *server* in exactly one submodel. However, a task can appear as a *client* in multiple places.

Five submodel construction strategies are studied in this paper:

SRVN: Each submodel consists of exactly one server (see [9]).

Batched: Each submodel treats of all of the tasks and processors at a given layer as servers.

MOL: *Software* submodels are constructed by treating only the tasks at a given layer as servers. A single *hardware* submodel is constructed using all of the processors as servers (see [7]).

HwSw: Two submodels are used: a *software* submodel is constructed by treating all non-reference tasks as servers and a *hardware* submodel like MOL. A task may appear as both a client and a server with this approach.

Squashed Only one submodel is used where all non-reference tasks and processors are treated as servers. Non-reference tasks will appear as both a client and a server.

2.2 Submodel Solution

Submodels are solved by converting a submodel into an ordinary queueing network model then solving this model either Bard-Schweitzer proportionate estimation [1, 8] or Linearizer [2] AMVA. Inputs to the AMVA solver are think and service times. Outputs are waiting (or residence) times, throughputs and utilizations. The clients in a submodel map to the chains in the queueing model and represented using delay centers. The service time for a client in a submodel are found by summing up the residence times to all the client's servers, except for those servers that are in the same submodel. Think times, Z, for clients in a submodel are set using the input parameters for reference tasks, and derived from the throughputs, λ, and the utilizations, U, for non-reference tasks. The servers in the submodel make up the queueing centers in the queueing model. The queueing discipline of the center depends on the task or processor type in the input model.

Once the queueing network for a submodel has been solved, the waiting time components corresponding to the submodel are updated for all of the submodel's clients, and the throughputs and utilizations are updated for all the submodel's servers. This process is repeated over all the submodels in the model until the convergence criteria has been met.

3. APPROXIMATE MVA CHANGES

In solving a layered queueing network using Linearizer AMVA, there are three levels of loops: the outer iteration shown in Algorithm 1, the loops in Linearizer for running Bard-Schweitzer AMVA, and the iterations within Bard-Schweitzer AMVA itself. This section briefly describes Approximate MVA and the changes to Linearizer and Bard-Schweitzer AMVA to remove most of this looping.

3.1 One Step MVA

Algorithm 2 shows one step of the main algorithm for Mean Value Analysis. The function $\texttt{wait}(m, k, \mathbf{n})$ is used to calculate the residence time at queueing station m for chain k for the population vector $\mathbf{n}$. The computational cost of one-step MVA is $O(MK)$ where M is the total number of stations in the queueing model and K is the total number of chains. Exact MVA recursively performs one-step MVA for all customer populations starting from zero, using the results from a previous solution with one customer removed from a chain j in the current iteration. It becomes prohibitively expensive for moderate numbers of chains and customers.

Algorithm 2 `step`($\mathbf{n}$)

```
 1: for $m \leftarrow 1$ to $M$ do {One-step MVA}
 2:     for $k \leftarrow 1$ to $K$ do
 3:         $R_{mk} \leftarrow v_{mk} \times wait(m, k, \mathbf{n})$
 4:     end for
 5: end for
 6: for $k \leftarrow 1$ to $K$ do
 7:     $\lambda_k \leftarrow \dfrac{n_k}{Z_k + \sum_{m=1}^{M} R_{mk}}$
 8:     for $m \leftarrow 1$ to $M$ do
 9:         $L_{mk}(\mathbf{n}) \leftarrow \lambda_k R_{mk}$
10:     end for
11: end for
```

3.2 Bard-Schweitzer AMVA

Bard-Schweitzer proportional estimation AMVA [8] breaks the recursion in Exact MVA by solving the network at $\mathbf{N}$ (the full customer population). It estimates the queue lengths L_{mk} at station m for each chain k with one customer removed from chain e_j, by assuming that $L_{mk}(\mathbf{N} - e_j)$ is proportional to $L_{mk}(\mathbf{N})$, i.e:

$$L_{mk}(\mathbf{N} - e_j) = \begin{cases} L_{mk}(\mathbf{N}) & \text{for } k \neq j \\ \frac{N_j - 1}{N_j} L_{mk}(\mathbf{N}) & \text{for } k = j \end{cases}$$

Algorithm 3 shows the Bard-Schweitzer AMVA algorithm as incorporated by Linearizer. $D_{mkj}(\mathbf{N}) = 0$ is a value used to adjust the proportion one customer contributes to the length of a queue. For pure Bard-Schweitzer AMVA, $D_{mkj}(\mathbf{N})) = 0$.

Algorithm 3 `core`($\mathbf{N}$)

```
 1: repeat
 2:     {Estimate $L_{mk}$}
 3:     for $m \leftarrow 1$ to $M$ do
 4:         for $k \leftarrow 1$ to $K$ do
 5:             $F_{mk}(\mathbf{N}) = L_{mk}(\mathbf{N})/N_k$
 6:             for $j \leftarrow 1$ to $K$ do
 7:                 $L_{mk}(\mathbf{N}-e_j) \leftarrow (\mathbf{N}-\mathbf{e}_j)_k \cdot (F_{mk}(\mathbf{N}) + D_{mkj}(\mathbf{N}))$
 8:             end for
 9:         end for
10:     end for
11:     step(N) {One-step MVA}
12:     {Termination test}
13: until $\max_{m \in M, k \in K} \dfrac{|L^I_{mk}(\mathbf{N}) - L^{I-1}_{mk}(\mathbf{N})|}{N_k} \leq \dfrac{1}{4000 + 16|\mathbf{N}|}$
```

3.3 Linearizer AMVA

Bard-Schweitzer AMVA often has large errors because the queue length estimate $L_{mk}(\mathbf{N} - e_j)$ is not a simple ratio of the customers in chain j. Linearizer [2] improves the accuracy of the Bard-Schweitzer approximation by calculating a scaling factor D_{mkj} to be used to find the populations of the queues with one customer removed. Linearizer finds the scaling factors by solving the queueing network using the Schweitzer approximation at both the full customer population $\mathbf{N}$, and $\mathbf{N} - e_j$ (i.e., one customer removed from chain j) for all K chains. Algorithm 4 shows the Linearizer algorithm.

Algorithm 4 `linearizer`($\mathbf{N}$)

```
 1: Initialize $L, \forall m, k$
 2: for $I \leftarrow 1$ to 2 do
 3:     core(N) {Step 1}
 4:     for $c \leftarrow 1$ to $K$ do {Step 2}
 5:         core($\mathbf{N} - \mathbf{e}_c$)
 6:     end for
 7:     for $m \leftarrow 1$ to $M$ do {Step 3}
 8:         for $k \leftarrow 1$ to $K$ do
 9:             $F_{mk}(\mathbf{N}) \leftarrow L_{mk}(\mathbf{N})/N_k$
10:             for $j \leftarrow 1$ to $K$ do
11:                 $F_{mk}(\mathbf{N} - \mathbf{e}_j) \leftarrow L_{mk}(\mathbf{N} - \mathbf{e}_j)/N_k$
12:                 $D_{mkj}(\mathbf{N}) \leftarrow F_{mk}(\mathbf{N} - \mathbf{e}_j) - F_{mk}(\mathbf{N})$
13:             end for
14:         end for
15:     end for
16: end for
17: core(N) {Step 1}
```

3.4 AMVA Changes

Three changes have been incorporated into the LQNS solver to speed up MVA.

1. Restart MVA from previous solution [6]: After the first iteration of Algorithm 1, values exist for the queue lengths of all stations so the initialization step at line 1 in Algorithm 4 does not need to be run again.

2. Run `core`($\mathbf{N} - \mathbf{e}_c$) in Linearizer only once instead of twice: The Linearizer algorithm in [2] runs the Bard-Schweitzer `core()` three times at the full population $\mathbf{N}$, and twice with one customer removed from chain j for all K chains. After the first run of Linearizer, the invocation of `core` at line 3 in Algorithm 4 will generate the same results as the last iteration of `core` at line 17. Therefore, for the second and subsequent invocations of Linearizer for a submodel, omit the call to `core` at line 3.

   The second performance improvement to Linearizer is to reduce the number of iterations of the main loop at line 2 in Algorithm 4 from 2 to 1. In effect, the main loop of linearizer is now being performed by the main loop of Algorithm 1.

3. Don't iterated `core`: In Algorithm 3, remove the **repeat** ... **until** loop at lines 1 and 13. Again, the main loop of Algorithm 1 can perform this step.

4. TEST CASE

In the results that follow a model of a bookstore application [4] was solved using the five layering strategies described earlier in §2.1 and using Linearizer AMVA (LIN), One-Step

MVA Solver	Layering Strategy				
	Batched	HwSw	MOL	Squashed	SRVN
Number of outer iterations.					
LIN	13	17	25	22	14
OSL	17	21	16	27	21
BSC	20	17	48	22	14
OSM	34	41	31	47	40
Calls to `wait()`$\times 10^4$					
LIN	26117	40129	87852	70247	26900
OSL	632	1036	814	1824	655
BSC	99	133	519	264	99
OSM	3	4	7	5	3
Run times (in mm:ss.cc)					
LIN	00:37	00:56	01:47	01:41	00:38
OSL	00:10	00:14	00:11	00:28	00:10
BSC	00:03	00:04	00:17	00:07	00:03
OSM	00:02	00:02	00:06	00:03	00:02

Table 1: Results for the Bookstore model.

Linearizer (OSL), Bard-Schweitzer (BSC), and One-Step MVA (OSM). Table 4 shows the number of outer iterations of Algorithm 1, the number of calls to the `wait()` function and the overall run time for each approach.

This model has a software bottleneck at task `Server-Ref_5d56d5`. The throughput of the model is the same to four digits of precision regardless of the layering strategy or whether the one-step approximation was used (Linearizer and Bard-Schweitzer do produce different results, which is not unexpected). Using the One-Step Linearizer (OSL) approximation reduces the number of calls to `wait()` by up to two orders of magnitude and using One-Step MVA (OSM) improves the performance by four orders. Run times of the solver are also improved significantly from the worst to best case.

The one-step approximation increases the number of outer iterations of Algorithm 1 for all of the layering strategies excepting MOL. The MOL algorithm solves the software submodel to convergence prior to solving the hardware model. The extra precision at the early stages of the iteration are not beneficial so the one-step variant converges more quickly overall.

5. CONCLUSIONS

This paper has examined the performance effects on the solution speed of the Layered Queueing Network Solver from using five different layering strategies and from changes to the iterative structure of approximate MVA. A performance model is solved by iterating among a set of submodels, each of which is an ordinary queuing network. The layering strategies ranged from a set of submodels, each consisting of exactly one serving station, to the opposite extreme of exactly one submodel. The case study considered in this paper showed that the final result produced by the solver was relatively insensitive to the layering approach chosen. However, the approaches that created submodels with least number of customer chains (SRVN and batched) were the least costly.

This paper also looked at improvements to the iterative structure of the queueing model solution. Three changes were made. First, rather than re-initializing the MVA solution from scratch each time a submodel was solved, the iteration was started from the previous solution of the submodel. Second, the number of iterations in Linearizer's main loop were reduced, partly because of the "warm start" introduced earlier, and partly because the outer iteration of the overall algorithm was going to run multiple times. Finally, the third change was to stop iterating the "core" Bard-Schweitzer algorithm at the heart of Linearizer. Much of the work obtaining "accurate" solutions during early iterations of the outer-most loop is wasted because the parameters for a given submodel are going to change anyway due to solutions to other submodels. Using One-Step Linearizer reduced the computational effort by about two orders of magnitude. Using One-Step MVA further reduced the cost by another two orders of magnitude.

The improvements to Approximate Mean Value Analysis described here are not limited to solutions to layered queueing networks. Rather, they can be incorporated into any solution that iterates between multiple queueing models.

Acknowledgments

This research was supported by a grant from NSERC, the Natural Sciences and Engineering Research Council of Canada.

6. REFERENCES

[1] Y. Bard. Some extensions to multiclass queueing network analysis. In M. Arato, A. Butrimenko, and E. Gelenbe, editors, *Performance of Computer Systems.* North-Holland, Amsterdam, 1979.

[2] K. M. Chandy and D. Neuse. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Commun. ACM*, 25(2):126–134, Feb. 1982.

[3] G. Franks, T. Al-Omari, M. Woodside, O. Das, and S. Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Trans. Softw. Eng.*, 35(2):148–161, Mar.–Apr. 2009.

[4] T. Israr, M. Woodside, and G. Franks. Interaction tree algorithms to extract effective architecture and layered performance models from traces. *J. Syst. and Soft.*, 80(4):474–492, Apr. 2007.

[5] P. A. Jacobson and E. D. Lazowska. Analyzing queueing networks with simultaneous resource possession. *Commun. ACM*, 25(2):142–151, Feb. 1982.

[6] M. Mroz and G. Franks. A performance experiment system supporting fast mapping of system issues. In 4th *International Conf. on Performance Evaluation Methodologies and Tools*, Pisa, Italy, Oct. 20–22 2009.

[7] J. A. Rolia and K. A. Sevcik. The method of layers. *IEEE Trans. Softw. Eng.*, 21(8):689–700, Aug. 1995.

[8] P. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proc. International Conference on Stochastic Control and Optimization*, Amsterdam, 1979.

[9] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Trans. Comput.*, 44(8):20–34, Aug. 1995.