This article appeared in the Proceedings of the First Int. Workshop on Software and Performance (WOSP98), pp. 120-130, Santa Fe, October 1998.

Performance of Multi-Level Client-Server Systems with Parallel Service Operations

Greg Franks Murray Woodside

Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6
{greg,cmw}@sce.carleton.ca

Abstract

Parallel execution can enhance the performance of distributed client-server systems, but the enhancement may be less than expected. Evaluations of such designs must include the complex effects of overheads, heterogeneous parallel branches, contention by the parallel parts for servers in lower levels, and simultaneous resource possession effects. A "compensated complementary delay" approximation is described which exploits layered queueing approximations for layered resources which occur in client-server architectures, based on synchronization delay estimates and adjusted levels of contention. The new approximation uses the overlap of parallel branches and a new fast calculation of join delays. It gives acceptable errors (averaging about two percent), and has an enormously lower computational cost compared to the competing approach based on decomposition. The new approximation may moderate a conclusion made by Heidelberger and Trivedi, that decomposition gives greatly superior accuracy to the much cheaper complementary delay approach; these delay approximations are only a little less accurate.

1. Introduction

Processes in a distributed system use services over a network, and may have serious delays when blocked waiting for a service to complete. To reduce the impact of blocking a process that needs several operations could request them in parallel, by forking parallel threads that then make the service requests. Figure 1 illustrates a database application with this kind of internal parallelism, sending requests to three separate databases and then combining the results. The right hand side shows how a user request is broken into three requests at the fork and sent to the three databases. At the join the application waits for all the replies. These *server systems with parallelism* may occur in business data processing, world wide web servers [1], and in geographic information systems, among others. Notice that in a multi-level system there can be parallel steps in servers at any level, so for example there might also be a parallel operation within one of the databases, issuing parallel requests to disks or file servers (such an example will be described later).

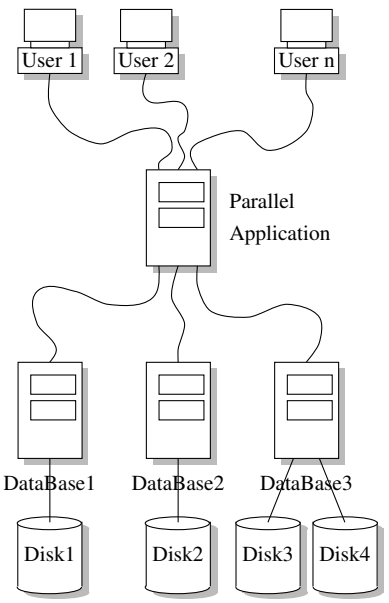
Other performance optimizations are logically equivalent to parallel threads even if the threads are hidden from the programmer. For instance for prefetching a file, a file access thread is forked at the prefetch and joined when the data is used.

Since parallelism is a performance optimization, a way to estimate the performance benefit would be helpful. We would like to employ analytic modelling for speed, but the models are complex to create because they combine parallelism effects with resource contention and (in many cases) with simultaneous resource possession that arises when a server or thread blocks to wait for a reply. Furthermore, practical systems may be quite large, so the method must scale up well. The approach taken here is to add the parallelism effects to an existing modelling framework, the layered queueing model or "rendezvous network" [2], [3], which is scalable and which assists the model-builder in representing the resource contention and the simultaneous resources. Layered models have been validated against simulations on hundreds of cases and in a variety of practical and laboratory applications [4], [5], [6].

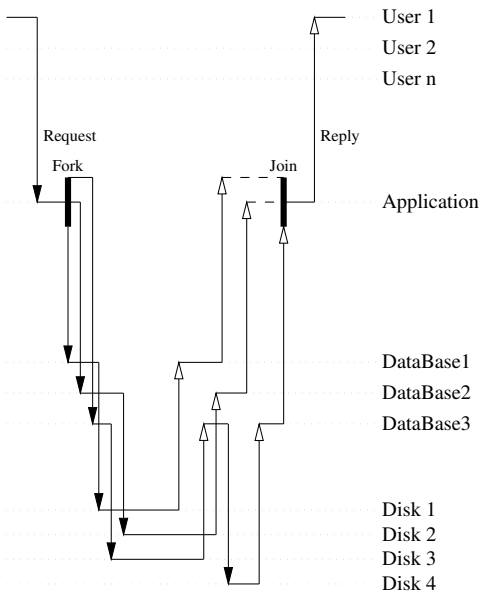
The common characteristics of service systems with parallelism are:

- random unsynchronized requests to a server from multiple clients,
- heterogeneous operations on the parallel paths, including perhaps requests to other servers,
- multiple layers of servers having resource constraints at all layers, such that a request may hold resources at higher layers while executing a low level service,

Copyright 1998 by the Association for Computing Machinery. Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.



(a) System Architecture



(b) Interaction Timeline

Figure 1. An example representing a business client-server system. The client workstations each make requests to a server running a parallel application. The application, in turn, makes requests to the three lower-level database servers. Figure 1(b) shows a sample interaction between User 1 and the parallel application.

- a substantial element of uncertainty or randomness in the duration of the parallel paths,
- no system-wide synchronization between workload components. Rather, parts of the workload are due to separate users who compete for resources. This is different from massively parallel computing in which the parallel paths all execute the same instructions and there may be frequent global synchronizations.

Several authors have developed analytic models for parallel service systems. Heidelberger and Trivedi [7] considered a simple computational loop executed by each user, alternating between a sequential section and a parallel section. They did consider contention between the parallel subtasks, but only one layer of resources. They compared two modelling approaches, called "decomposition" and "complementary delays" (CD). Decomposition requires a separate performance model for each possible state of parallelism, as the number of independent users increases the algorithm suffers combinatorial explosion. It does not scale up to large systems. CD determines an equivalent delay for the synchronization at the join point and scales up well.

In later papers the decomposition approach was extended. Thomasian and Bay [8] developed a way to apply it to larger task graphs with a general structure of parallel and sequential parts. Petriu et. al. [9] applied it to layered systems with any number of layers of servers (that is, with complex simultaneous resources). In both cases the cost is prohibitive if the number of independent users is even moderately large.

The CD approach has also been extended. Chu Sit and Leung [10] modelled the join delay more carefully by considering the distribution of delay on each parallel path, and included one additional upper-layer resource (equivalent to a middle layer of service). Mak and Lundstrom [11] improved the accuracy by accounting for the overlap in time of synchronized parallel sub-paths. However their development was limited to one set of forked jobs at a time, and also did not include simultaneous resources. Both these papers assumed that the branches of a parallel operation have an exponentially-distributed delay, to calculate join delay based only on mean value.

Another class of models for these systems is based on Markov Chains derived from Petri nets or Process algebras. Models in this class represent all the states of parallelism in a single model, together with the contention in each state. However they suffer even worse from state explosion than the decompositions above, so we must regard them as unscalable in their present form.

If decomposition and state-based approaches do not scale up, then we must see if the delay-based approach

can be made more accurate. To do so, this paper extends the calculation of the overlap in contention given in [11], which makes it apply to layered systems with multi-threaded servers. It also exploits the variance of the delay on the parallel paths, with a "three-point" approximation for the join delay. This gives a "Compensated Complementary Delay" (CCD) approach described in section 3..

The CCD analysis is applied in Section 4. to several examples, showing that:

- the approach scales up to systems with many servers having parallel sections and to many independent users,
- approximation errors are limited to a few percent,
- from the 228 examples studied in [7], the CCD errors are about three times as large as errors from decomposition (CD errors are about ten times as large).

It is concluded that the delay approximation combines usable accuracy with scalable complexity.

2. Example as a Layered Model

Figure 2 shows a simplified version of the system of Figure 1 (i.e., without the disks) as a layered queueing model, showing parallel requests from the Application Server in the middle to the Database servers at the bottom. The parallelism is defined by the little *activity graph* nested inside the ParApp service in the Application Server. An activity graph defines precedence among its activities.

In a layered model, each task is a resource; a multi-threaded task is a set of M resources with a single queue. At the top of Figure 2 is a set of N Client tasks which cycle, executing a unit of work in each cycle. In each cycle there is 1.0 seconds of CPU demand on the client's local processor, α requests for the first service, labelled *SeqApp*, and one request for the second service, labelled *ParApp*, which has internal parallelism. Each request for service is blocking, so the Client waits for it to be completed. The SeqApp service of the Application Server is a typical sequential program. It demands 0.1 seconds of CPU time on its processor, and makes an average of 1.0 and 0.2 requests to Databases 1 and 2, respectively. The number of requests is geometrically distributed.

The service ParApp of the Application Server makes requests in parallel to the three Database servers, and this is represented in the model as follows. When a request arrives at the Application Server for ParApp, it triggers the first activity at the top of the activity graph (indicated by a rectangle named a1 in Figure 2 and labelled with a CPU demand of 0.3 units). It then forks three parallel threads, indicated by the arcs coming out of the bottom of the first rectangle and the

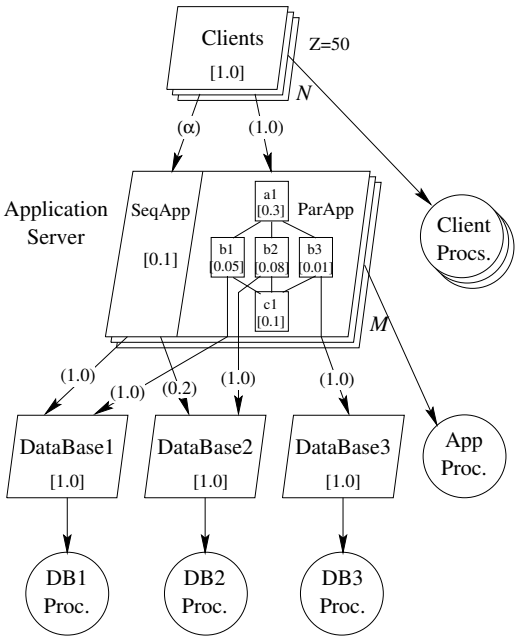


Figure 2. Layered Queueing Network for the business client-server system shown in Figure 1. The large parallelograms are tasks and the circles are processors. A stacked icon denotes a set of processors, tasks or threads. The Application task has two different service types, labelled *SeqApp* and *ParApp*. The ParApp service type consists of a single thread of control that forks into three independent threads, then joins again. Numbers within parentheses, '( )' represent request rates. Numbers within brackets, '[ ]', represent CPU demand.

rectangles labelled b1, b2 and b3 with CPU demands of 0.05, 0.08, and 0.01 respectively; these threads make requests to the three Database servers, indicated by the arrows going from the boxes to the servers. When the parallel parts are completed they join and execute the last activity, labelled c1 with 0.1 unit of CPU demand.

The Application Server is also multi-threaded in the sense that it can initiate additional services for other clients, while its busy threads are waiting for responses from Databases, up to a limit of M concurrent threads. These M main threads all run on the single AppProc processor, and execute whichever service (SeqApp or ParApp) is requested next. The Database servers are assumed here to be single threaded servers on a single processor each, which is simplistic but focuses attention on the parallelism issue; the model can instead accommodate any configuration of multithreaded, parallel and multiprocessor database servers as well, as will be shown in Section 4..

The performance is governed by a complex combi-

nation of resource contention and parallelism. When the Application Server is multithreaded its threads may contend for access to the Databases; threads executing SeqApp make a different pattern of Database accesses, than those of ParApp. Conversely, when it is single threaded, the Application Server is held as a simultaneous resource during a Database server access. A larger value of α increases the contention for Databases 1 and 2, so the duration of parallel branches b1 and b2 increases also through contention. This delays the synchronization, and keeps the Application Server threads busy longer, waiting.

3. Solution of Queueing Networks with Layered Resources

The service time of a software server runs from when it begins to work on a request, until it sends the reply. The time includes parts from the server's own processing delay and parts from lower level servers which it uses. These lower servers may in turn make requests to other still lower servers, with contention effects which are propagated upwards.

The entire system of queues and delays can be solved by separating it into several "layer submodels", such that one submodel contains the servers at a given call depth or request depth from the clients [2], [3]. In the layer submodel there is a customer for each entity that can make a request to one of its servers. A single-threaded higher-level task gives rise to a separate chain with one customer; a multi-threaded task gives a chain with M customers, one for each thread. Each layer submodel is an ordinary closed queueing model, and can be solved by one of the usual algorithms such as Linearizer [12].

The parameters of a layer submodel are partly found from other layers:

- The service times of its servers are calculated from waiting times and service times in lower layers.
- The time between requests from the customers of a chain is a surrogate delay or "think time" for the chain, which is determined by the frequency of requests and other behaviour of the corresponding upper layer entity.

The underlying assumptions of the model are as follows [2]:

- Messages arriving at a task are queued with a first-come, first first served discipline.
- The CPU demand at a task in response to a message is divided up into exponentially distributed slices between requests to lower level servers.
- Calls to lower level services are geometrically distributed with the specified mean.
- The call graph with tasks as nodes and requests as arcs is acyclic (cyclic graphs may deadlock).

The web of relationships by which results are found in one layer and used in another layer is resolved by a fixed-point iterative calculation, which has been described in other papers [13], [2], [3], [14]. The overall strategy has been applied to many models and generally converges well (sometimes under-relaxation must be used), with approximation errors of a few percent.

3.1. Solutions for Parallelism

Parallelism affects the solution of the layered queueing model in two ways. First, if a server has parallel sub-paths, its service time includes the delay from the fork to the join, including the synchronization delay waiting for the last sub-path to finish. Second, in the layers below the parallel server, the parallel part is represented by an additional customer entity for each sub-path. The response times of these entities provides the sub-path delays used to calculate the join synchronization delay. If we just solved these layered models using standard queueing algorithms, we would have the CD approach of [7]; this technique is called CD$'$ in results described later.

There is a contention effect that was not considered in the CD method of [7], which has motivated the "Compensation" aspect of Compensated Complementary Delays (CCD). The sibling parallel sub-threads of a single server task are all launched at once at a fork operation, and this increases their contention at the beginning of the operation. Then as they complete one by one the contention level drops until the last one has no contention from its siblings. The probability of having to contend with for one of these sibling sub-threads will be multiplied by a compensation factor reflecting the probability that sub-threads are active at the same time. Since Compensation only considers the siblings of a particular sub-path, forked from the same parent, it is probably most important in analyzing a system with parallelism in a single-threaded server, or one with very few threads. For more server threads, most of the contention of sub-path "customers" is with customers from other server threads, for which there is no special interaction.

3.2. Contention Generated by Parallel Activities

Figure 3 shows the activities and their interaction timeline of the ParApp service from Figure 2 along with their temporal relationship to the chains and active threads used by the MVA solution.

When an application service forks a set of parallel branches, each branch is represented by a separate customer chain in the lower layer submodel. For example, Figure 3(c) shows chain 1 representing the main thread,

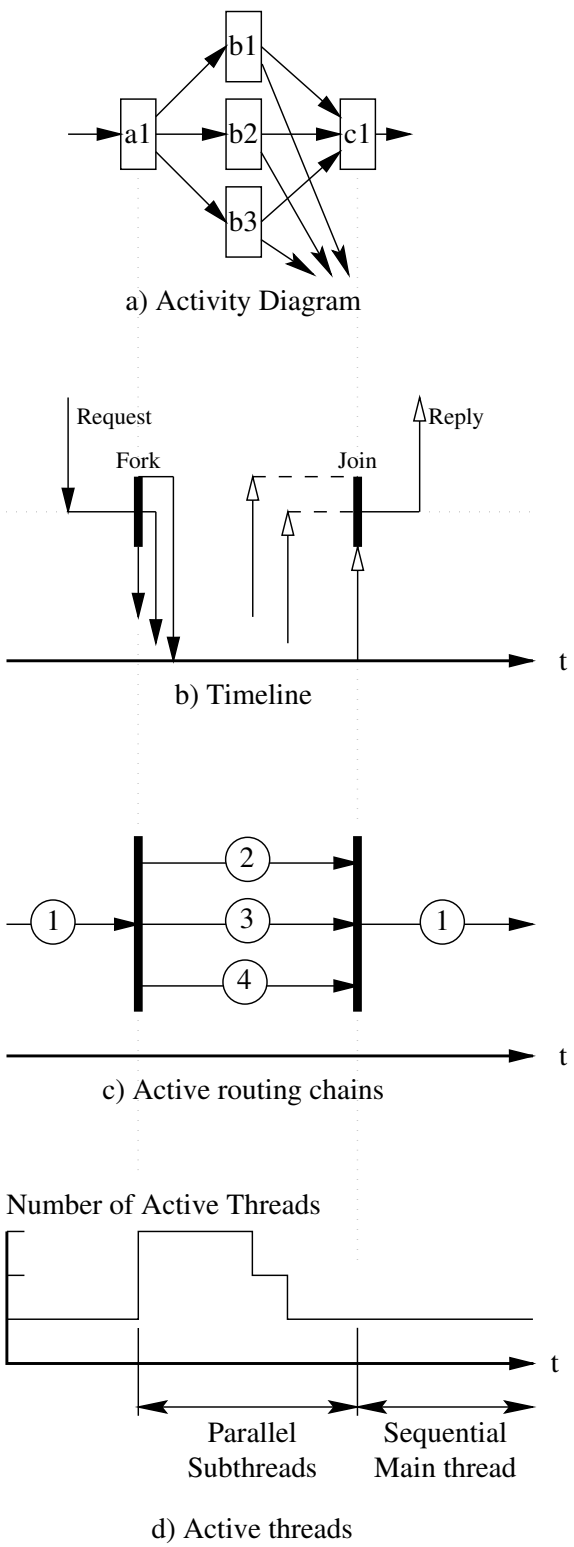


Figure 3. Activity graph - customer chain relationship for the parallel application from Figure 2. Note that even though there are four distinct chains, at most three are active at any time.

which executes activities a1 and c1 of the ParApp service and also executes the SeqApp service when requested, together with chains 2, 3, and 4 representing the threads that fork and execute activities b1, b2 and b3. The delay time (think time) of these chains is the mean time from the join to the next fork, for each main thread instance. This is a complementary delay. The chain populations are the same as the population of chain 1, which is the number M of main threads in the server.

The mean-value equations of contention by members of chains 2, 3 and 4 at queues in a lower level of servers must be adjusted to account for their synchronization at the fork. Mak and Lundstrom did this in [11] by finding the overlap between the active phases of chains 2, 3, and 4. The *overlap fraction* θ_{kj} is the probability that, when chain k is active, *sibling* chain j is active also. This fraction is

$$\theta_{kj} = d_{kj}/W_j \tag{1}$$

where W_j is the mean active lifetime of sub-path j and d_{kj} is the mean overlap time from the fork point, given by

$$d_{kj} = \frac{1}{W_j^{-1} + W_k^{-1}}. \tag{2}$$

When there are two related fork-joins, one nested in the other, then a branch of one also has an overlap with a branch of the other. This is calculated by determining the probability p_{kj} of any overlap at all, given by [11]:

$$p_{kj} = 1 - \Pr(E_j < S_k) - \Pr(E_k < S_j). \tag{3}$$

This calculation is based on the delay from the common synchronization point at the outermost fork; S_j denotes the time when j can start executing and E_j denotes the time when j finishes. Then

$$\theta_{kj} = \frac{p_{kj} d_{kj}}{W_j}. \tag{4}$$

Mak and Lundstrom considered only a queueing network with at most one customer in each chain. Since this work has servers with multi-threaded servers, there can be additional members of chain j which are not siblings of the member of chain k. For these others, the overlap is different. For example, a particular active member of chain 2 from Figure 3 is synchronized only with one member of chain 3 and one of chain 4, its own siblings that were forked with it from the same chain-1 parent. Other chain-1 customers fork at unrelated moments and spawn members of chains 1, 2 and 3 which have no special temporal relationship to this particular member of chain 2. Assuming the forking times of

non-sibling chains j and k are independent, the *unsynchronized overlap fraction* for members of chain j and k, which we can call θ^*_{kj}, is just the fraction of time chain j is active:

$$\theta^*_{kj} = \lambda_k W_k \tag{5}$$

where λ_k is the throughput for chain j.

Over the N_j members of chain j, the average overlap fraction is found from 4 and 5.

$$\overline{\theta_{kj}} = \frac{\theta_{kj} + (N_j - 1)\theta^*_{kj}}{N_j}. \tag{6}$$

In MVA [15] a mean number L_{mj} of chain j customers is found in queue m at an arrival instant of a chain k customer. MVA assumes no temporal relationships between thread activations, and therefore corresponds to an unsynchronized overlap probability of θ^*_{kj}. If the average overlap is $\overline{\theta_{kj}}$ instead, we assume that the mean number of customers increases in proportion to $\overline{\theta_{kj}}$; the mean number L_{mj} then is replaced by $(\overline{\theta_{kj}}/\theta^*_{kj})L_{mj}$. With multiple chains $j = 1..K$ we obtain

$$W_{mk}(\mathbf{N}) = D_{mk}\left[1 + \sum_{j=1}^{K} \frac{\overline{\theta_{kj}}}{\theta^*_{kj}} L_{mj}(\mathbf{N} - \mathbf{e}_k)\right] \tag{7}$$

where $W_{mk}(\mathbf{N})$ is the total residence time per response cycle of chain k at station m, in a system with a chain population vector $\mathbf{N}$, D_{mk} is the total service demand of chain k at m, and $L_{mj}(\mathbf{N} - \mathbf{e}_k)$ is the mean number of chain-j customers found at an arrival instant (as computed by the usual MVA method [15]). L_{mj} is further approximated in the solver in the Linearizer algorithm.

Finally, there are pairs of customers that have zero overlap such as the parent customer in chain 1 with any of its children in chains 2, 3 and 4. For these pairs of customers $p_{kj} = 0$, and thus $\theta_{kj} = 0$.

3.3. Approximation for Join delays

The difficulty with finding join delays is that an accurate calculation requires the full probability distribution for the branch lifetimes, which is not available from a mean value analysis (MVA). The usual approximation, made for instance by Mak and Lundstrom [11] and by Heidelberger and Trivedi [7], is that branch delays are exponential, in which case there is a simple analytic formula for the join delay which depends only on mean values. However this may give quite large errors. To eliminate these, Chu, Sit and Leung [10] demonstrated a gamma-distribution approximation which requires fitting several distributions and then integrating numerically over the final distribution to obtain the mean and variance. This works well but is computationally expensive.

This work used a new "three-point" approximation due to Jiang and Woodside [16] to compute the mean and variance of the join delays, based on the mean and variance of the branch delays. The distribution for each branch delay is approximated by three point probabilities, and the rest of the analysis is exact. The algorithm for determining the locations t_{ij} of the three points and the values $a(t_{ij})$ for the probabilities (from the mean and variance of a delay t_i) is given in [16]. Figure 4 illustrates the point probabilities $a_i(t)$ for one branch and the approximation $A_i(t)$ for the probability distribution function for that branch. For the overall delay it is $A(t)$, the product of the $A_i(t)$. The probabilities $a(t_k)$ for the fork-join delay are found for the points t_k where $A(t)$ jumps, and the mean and variance follow. The discrete approximation makes the calculation fast.

The three-point approximation was tested in [16] with hundreds of randomly generated test cases. When the squared coefficient of variation of each of paths involved in the join was less than or equal to one, the mean absolute error in overall delay was less than one percent, and when the squared coefficient of variation was between one and nine, the mean absolute error was less than four percent. In larger task graphs the errors tended to be smaller. This accuracy is adequate for practical predictions

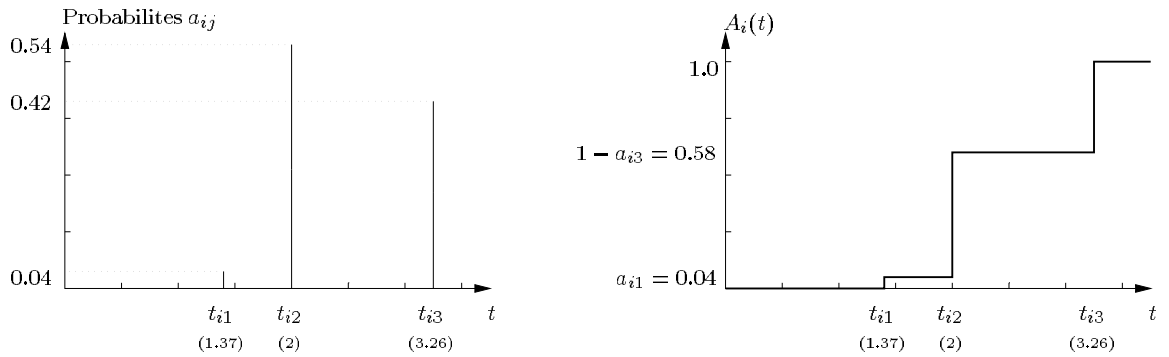
3.4. Complexity

The layered model algorithms are described as "scalable" because their computational demands and storage go up as low-order polynomials in the size parameters of the model, such as the number of tasks, services and request arcs [2]. Each of the layer submodels is solved using the Linearizer approximate MVA algorithm, therefore the dominating complexity parameter is cubic in the number of chains [12] in a given submodel.

With parallel services there is an additional parameter B, the largest number of parallel branches in any one server. B enters the time complexity through the overlap and the branch and join delay calculations, (which are linear in B) and through the increase in the number of chains in some submodels by a number proportional to B. Similar arguments apply to the space requirements. Thus the layered solution algorithm incorporating the fork-join approximations remains scalable.

4. Results

A useful robust approximation must be sufficiently accurate (a few percent error), and also be scalable to large systems. This section considers three examples, first to illustrate accuracy using a small example, second an example with greater system complexity and third to compare the accuracy to [7]. Each



(a) Approximate distribution for a branch i.

(b) Distribution Function

Figure 4. Three-point approximation for a distribution with $\overline{t_i} = 2.0$ and $\sigma_i^2 = 0.4$.

of the systems were compared to detailed simulations which modeled the semantics of the layered queueing networks.

4.1. Application Server Example

The example given in Figure 2 was analyzed with one thread and four threads in the Application Server, and with variations in the relative frequency of requests to the sequential service SeqApp and the parallel service ParApp, as determined by the SeqApp request rate α. The results in Tables 1(a) and 1(b) show the client throughput values by simulation, with confidence intervals, and the predicted throughput values and errors by CD′ and CCD methods. The prediction error is measured by the relative error in the predicted client throughput, $\epsilon = \frac{\text{sim.}-\text{approx.}}{\text{sim.}} \times 100$.

With only one thread in the server, CCD is roughly four to eight times better. The advantage becomes large as the traffic levels increase (α increases) and the Application Server becomes more of a bottleneck. However with four threads, CCD is only a little better and only for small α. The difference is that with one thread all the sub-paths in the Application Server are synchronized and have overlap factors which are different from the default, while with four threads the unrelated and unsynchronized sub-paths dominate. Also, as α increases, a bottleneck appears at Database1 and this dominates the system solution.

4.2. Extended Client-Server Example

To show how the solution behaves in a larger client-server model, parallelism was incorporated into lower level server, Database 3, as shown in Figure 5. It now has parallel paths within it, and additional disks are accessed in parallel. This system was then analyzed

	Simulation		Analytic Approximations			
			CD′		CCD	
α	λ	$\pm 95\%$	λ	$\epsilon(\lambda)$	λ	$\epsilon(\lambda)$
1	0.48	0.0020	0.43	-10.06	0.50	2.76
2	0.55	0.0034	0.49	-11.48	0.56	1.82
4	0.62	0.0016	0.56	-9.97	0.63	1.52
7	0.67	0.0051	0.62	-7.95	0.68	0.96
10	0.69	0.0067	0.65	-6.12	0.70	1.10

(a) $M = 1$ thread

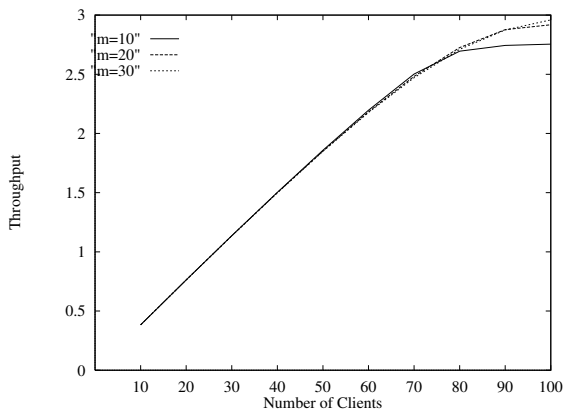
	Simulation		Analytic Approximations			
			CD′		CCD	
α	λ	$\pm 95\%$	λ	$\epsilon(\lambda)$	λ	$\epsilon(\lambda)$
1	0.90	0.0030	0.87	-4.38	0.88	-2.96
2	0.96	0.0047	0.93	-3.02	0.93	-2.58
4	0.98	0.0036	0.97	-0.76	0.97	-0.74
7	0.99	0.0019	0.99	0.37	0.99	0.37
10	0.99	0.0025	1.00	0.46	1.00	0.46

(b) $M = 4$ threads

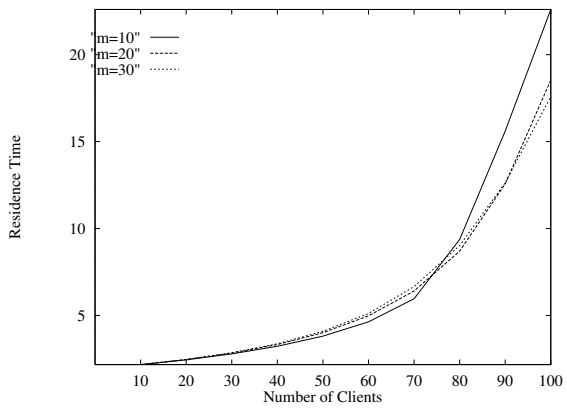
Table 1
Results for the Application Server with M threads (throughput, predicted throughput and its percentage error).

by simulation and by the CD′ and CCD solvers over a range of client population N from 10 to 100, and a range of threads in Application Server from 10 to 30. The computing time for each solution was about 20 seconds each for CD′ and for CCD (independent of the number of customers), and ranged from 20 to 140 minutes for simulation

The CCD results in Figure 6 show the Client throughput and response time rising as the number of clients is increased. The additional Application Server threads (beyond 10) make no difference until the clients approach 100. At this point, ten threads give a bottleneck at the Application Server, while 30 threads push the bottleneck down to its processor.



(a) Throughput



(b) Waiting Time.

Figure 6. Results for the extended example, with varying threads in Application Server.

When we compare the CD′ and CCD prediction errors in Table 2 we find that they are much the same. In this system with so many threads the effect of overlap is buried by the large number of unsynchronized chains due to the other threads. The errors are largest at the bottom of Table 2(a), due to errors in the fork-join delay approximation, which clearly still needs to be improved. The mean error over all the runs, -0.79% is within the 95% confidence intervals of $\pm 2\%$ of the "true" simulation results.

The value of CCD is now clear. Compared to CD′, CCD gives much better accuracy in certain cases, and in all cases, it contributes a negligible extra cost. It gives the extra accuracy when there is a small number of threads that can fork in parallel in the same server, and when this server is limiting the performance of the system.

4.3. Comparison to Decomposition

The 228 test cases described in [7] were recast as layered models to compare the new algorithm to the decomposition method of [7]. The test cases have one two five client tasks which cycle. Each client task has a single parent activity which forks either two or three child sub-paths and waits for the children to complete, then cycles again. All the sub-paths make requests to a set of disks. For the two-child test, 108 different cases were analyzed with one or two CPUs, 1, 3, or 5 clients, and different visit and service time values. For the three-child test, 120 cases were analyzed with one or three CPUs, 1, 3 or 5 clients, and different values of other parameters, as described in [7].

Table 3 shows the mean absolute percentage error, the 90th percentile and the largest sample of $|\epsilon|$, denoted by Ave, 90% and Max respectively over the 54 cases in each of the two-child (Ch = 2) reports and the 60 cases in each of the three-child (Ch = 3) reports. The columns labelled "Dcmp" and "CD" list the results given in [7] for decomposition and for CD respectively. The results in the columns labelled "CD′" and "CCD" were found by comparing simulations (with confidence intervals of 0.5% at the 95% level) to predictions made with CCD and with CD′. The comparisons between results for Ave, 90% and the maximum observed $|\epsilon|$ are very similar so they will be discussed together.

The results for CD and CD′ differ slightly because the simulations were redone, and because CD′ used the three-point approximation to find the join delays. On these tests they are about the same, with some errors larger in one column, some larger in the other.

The last column describes the new CCD results. Compared to decomposition in column one, the errors are three to four times greater for CCD. While this is disappointing, CCD still has quite good accuracy for practical purposes (about 2%–3% in the Ave measure). Compared to CD′ (without the overlap compensation),

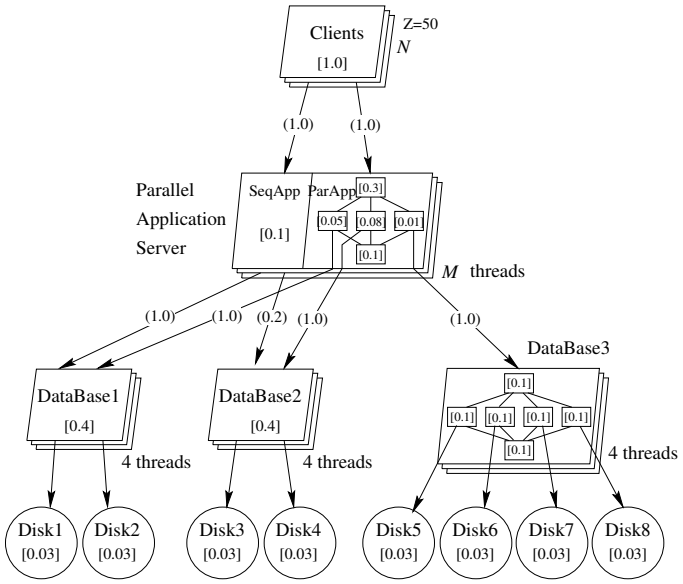


Figure 5. Layered Queueing Network for an extended business client-server system with parallelism in one of the databases.

CCD is two to three times better, and this improvement can be valuable. Thus, the compensation is a big gain.

CPU	Ch Act	Meas	From [7] Dcmp	CD	New Approx CD′	CCD
1	2	Ave	0.5	5.8	5.83	2.74
		90%	1.0	10.0	10.01	5.08
		Max	2.7	12.7	17.63	11.5
2	2	Ave	0.6	4.9	3.98	2.10
		90%	1.2	8.7	7.92	3.84
		Max	2.2	10.3	11.64	11.5
1	3	Ave	0.7	5.6	5.19	2.86
		90%	1.4	7.4	8.58	4.37
		Max	4.2	15.3	11.39	7.07
3	3	Ave	1.0	2.5	4.07	2.15
		90%	2.0	8.6	7.53	3.53
		Max	6.8	10.7	9.12	8.34

Table 3
Magnitudes of percentage throughput errors in the 228 test cases studied by Heidelberger and Trivedi [7]. Ave is the mean of the absolute relative errors, stated as percentages; 90% is a 90th percentile and Max is the maximum error in a group of samples.

The second table (3) gives more detail on the comparison between CD′ and CCD from the new results, by showing the mean and standard deviation of ϵ (without taking the absolute value). Here $\epsilon(\lambda)$ stands for errors in the throughputs calculated for all servers in the models. Now we can see that most of the throughput error is due to systematic bias which depends on the number of CPUs or parallel branches. The spread, indicated by σ, for CCD is much smaller.

CPUs	Child Activites	Measure	$\epsilon(\lambda)$ CD′	CCD
1	2	Mean	-5.84	1.84
		σ	0.21	0.04
2	2	Mean	-3.65	1.97
		σ	0.19	0.05
1	3	Mean	-4.49	1.44
		σ	0.23	0.01
3	3	Mean	-2.47	1.51
		σ	0.28	0.01

Table 4
Bias and spread of the percentage errors in throughput from the same cases as Table 3.

5. Conclusions

The examples demonstrate the power of the layered modelling framework to provide a convenient platform for defining models with parallelism. They can have parallelism in different servers and in different parts of the system, and can represent layered contention effects and the effects of multi-threading.

N cust	Simulation		Analytic Approximations CD′		CCD	
	λ	±95%	λ	$\epsilon(\lambda)$	λ	$\epsilon(\lambda)$
10	0.19	0.0007	0.19	-0.07	0.19	-0.07
20	0.38	0.0025	0.38	-0.74	0.38	-0.75
30	0.57	0.0028	0.57	-0.73	0.59	-0.73
40	0.76	0.0037	0.75	-0.93	0.75	-0.94
50	0.94	0.0012	0.93	-1.45	0.93	-1.45
60	1.12	0.0030	1.10	-1.78	1.10	-1.76
70	1.28	0.0054	1.25	-2.40	1.25	-2.35
80	1.42	0.0032	1.34	-5.12	1.35	-4.99
90	1.52	0.0065	1.37	-9.75	1.37	-9.59
100	1.55	0.0055	1.37	-11.64	1.37	-11.54

(a) $M = 10$ threads

N cust	Simulation		Analytic Approximations CD′		CCD	
	λ	±95%	λ	$\epsilon(\lambda)$	λ	$\epsilon(\lambda)$
10	0.19	0.0007	0.19	-0.07	0.19	-0.07
20	0.38	0.0030	0.38	-0.41	0.38	-0.41
30	0.57	0.0027	0.57	-0.49	0.57	-0.49
40	0.76	0.0033	0.75	-1.16	0.75	-1.17
50	0.94	0.0037	0.93	-1.61	0.93	-1.60
60	1.12	0.0022	1.09	-2.53	1.09	-2.52
70	1.28	0.0025	1.24	-3.10	1.24	-3.07
80	1.42	0.0039	1.36	-4.01	1.36	-3.96
90	1.51	0.0040	1.44	-4.81	1.44	-4.73
100	1.55	0.0044	1.46	-6.25	1.46	-6.15

(b) $M = 20$ threads

N cust	Simulation		Analytic Approximations CD′		CCD	
α	λ	±95%	λ	$\epsilon(\lambda)$	λ	$\epsilon(\lambda)$
10	0.19	0.0007	0.19	-0.07	0.19	-0.07
20	0.38	0.0010	0.38	-0.36	0.38	-0.36
30	0.57	0.0029	0.57	-0.74	0.57	-0.74
40	0.76	0.0028	0.75	-1.28	0.75	-1.28
50	0.94	0.0039	0.92	-1.68	0.92	-1.68
60	1.12	0.0047	1.09	-2.56	1.09	-2.56
70	1.28	0.0012	1.24	-3.44	1.24	-3.43
80	1.42	0.0016	1.35	-4.48	1.35	-4.47
90	1.51	0.0019	1.44	-5.09	1.44	-5.04
100	1.55	0.0044	1.48	-4.88	1.48	-4.82

(c) $M = 30$ threads

Table 2
Throughput and mean relative error results for the extended example, with varying threads in Application Server. Simulation results had 95% confidence intervals of ±2%.

Compared to decomposition, CCD has adequate (though worse) accuracy combined with a manageable cost which remains feasible for large numbers of threads and branches, while the cost of decomposition rises exponentially. We can describe this as a robust combination of accuracy and cost.

The CCD approximation has been found to give quite good accuracy for practical systems, typically with errors of a few percent. It was tested over many hundreds of examples with different parameters. CCD is intermediate in accuracy between decomposition and a CD approximation which is uncompensated for chain overlap. Unlike decomposition, it can be economically scaled up to large systems, in the sense that its time and storage demands are low-order polynomials in the size parameters of a model. CCD is robust in cost and accuracy, in that it avoids both the larger errors of complementary delay methods and the exponential cost escalation of decomposition.

Compensation in CCD was found to give an error reduction of up to a factor of two in the throughput prediction over CD′. The greatest advantage occured in those cases where the parallel server has few threads and is a bottleneck to the system.

In other cases complementary delays (CD′) is almost as good. Therefore it may also have a place as a technique giving useful accuracy in many practical cases.

The examples do not exhaust the possible applica-

tions of the CCD approximation. For instance they do not include asynchronous RPCs and prefetches, but it should be clear how these can be modelled, by forking one thread to execute the remote request while the main computation goes on in a second parallel child thread, up to a join where the result is needed. Similarly, the models do not consider network latencies, but these delays can be added easily.

Acknowledgments

This research was supported by the Telecommunications Research Institute of Ontario (TRIO) and the Natural Science and Engineering Research Council of Canada (NSERC).

References

[1] James C. Hu, Sumedh Mungee, and Douglas C. Schmidt, "Techniques for developing and measuring high performance web servers over high speed networks," in *INFOCOM '98*, San Francisco, CA, Mar. 1998, IEEE.

[2] C. Murray Woodside, John E. Neilson, Dorina C. Petriu, and Shikharesh Majumdar, "The stochastic rendezvous network model for performance of synchronous client-server-like distributed software," *IEEE Transactions on Computers*, vol. 44, no. 8, pp. 20–34, Aug. 1995.

[3] J. A. Rolia and K. A. Sevcik, "The method of layers," *IEEE Transactions on Software Engineering*, vol. 21, no. 8, pp. 689–700, Aug. 1995.

[4] Fahim Sheikh and Murray Woodside, "Layered analytic performance modeling of a distributed database system," in *17th International Conference of Distributed Computing Systems (ICDCS '97)*, Baltimore, ME, May 1997, pp. 482–490, IEEE Computer Society Press.

[5] C. M. Woodside and C. Schramm, "Scalability and performance experiments using synthetic distributed server systems," *Distributed Systems Engineering*, 1996.

[6] J. Dilley, R. Friedrich, T. Jin, and J. A. Rolia, "Measurement tools and modeling techniques for evaluating Web server performance," in *Proc. 9th Int. Conf. on Modelling Techniques and Tools*, vol. 1245 of *Lecture Notes in Computer Science*, pp. 155–168. Springer-Verlag, 1997.

[7] Philip Heidelberger and Kishor S. Trivedi, "Analytic queueing models for programs with internal concurrency," *IEEE Transactions on Computers*, vol. 32, no. 1, pp. 73–82, Jan. 1983.

[8] Alexander Thomasian and Paul Bay, "Queueing network models for parallel processing of task systems," in *Proceedings of the 1983 International Conference on Parallel Processing*, H. J. Siegel and Leah Siegel, Eds., Columbus, OH, Aug. 1983, pp. 421–428, IEEE Computer Society Press.

[9] Dorina C. Petriu, Shikharesh Majumdar, Jing-Ping Lin, and Curtis Hrischuk, "Analytic performance estimation of client-server systems with multi-threaded clients," in *Proceedings of the Second International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'94)*, V. Madisetti, E. Gelenbe, and J. Walrand, Eds., Durham, NC, Jan. 1994, pp. 96–100, IEEE Computer Society Press.

[10] Wesley W. Chu, Chi-Man Sit, and Kin K. Leung, "Task response time for real-time distributed systems with resource contention," *IEEE Transactions on Software Engineering*, vol. 17, no. 10, pp. 1076–1092, Oct. 1991.

[11] Victor W. Mak and Stephen F. Lundstrom, "Predicting performance of parallel computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 257–270, July 1990.

[12] K. Mani Chandy and Doug Neuse, "Linearizer: A heuristic algorithm for queueing network models of computing systems," *Communications of the ACM*, vol. 25, no. 2, pp. 126–134, Feb. 1982.

[13] C. Murray Woodside, "Throughput calculation for basic stochastic rendezvous networks," *Performance Evaluation*, vol. 9, pp. 143–160, 1989.

[14] Greg Franks, Alex Hubbard, Shikharesh Majumdar, Dorina Petriu, Jerome Rolia, and Murray Woodside, "A toolset for performance engineering and software design of client-server systems," *Performance Evaluation*, vol. 24, no. 1–2, pp. 117–135, Nov. 1995.

[15] M. Reiser and S.S. Lavenburg, "Mean value analysis of closed multichain queueing networks," *Journal of the ACM*, vol. 27, no. 2, pp. 313–322, Apr. 1980.

[16] Xianghong Jiang, "Evaluation of approximation for response time of parallel task graph model," M.S. thesis, Department of Systems and Computer Engineering, Carleton University, Canada, Apr. 1996.