Enhanced Modeling and Solution of Layered Queueing Networks

Greg Franks *Member, IEEE*, Tariq Al-Omari *Member, IEEE*, Murray Woodside *Fellow, IEEE*, Olivia Das, Salem Derisavi

Abstract—Layered queues are a canonical form of extended queueing network for systems with nested multiple resource possession, in which successive depths of nesting define the layers. The model has been applied to most modern distributed systems, which use different kinds of client-server and master-slave relationships, and scales up well. The Layered Queueing Network (LQN) model is described here in a unified fashion, including its many more extensions to match the semantics of sophisticated practical distributed and parallel systems. These include efficient representation of replicated services, parallel and quorum execution, and dependability analysis under failure and reconfiguration. The full LQN model is defined here and its solver is described. A substantial case study to an air traffic control system shows errors (compared to simulation) of a few percent. The LQN model is compared to other models and solutions, and is shown to cover all their features.

Index Terms—Modeling and Performance Prediction, Queueing Theory

1 Introduction

MANY distributed computing systems can be modeled compactly using a canonical form of extended queuing network (EQN) called *layered queueing* (LQ). When a software server calls another server and waits (blocked) for the return from the call, that is an example of layered queueing. The pattern can be repeated to any depth, and includes requests to processor servers. Layered queueing occurs in all kinds of information and e-commerce systems (e.g. Client-Server, Service Oriented Architecture, etc.), in grid systems, and in real-time systems such as telecom switches [1]. An example LQ model is shown in Figure 1 and explained below. Efficient analytical solutions can be computed for complex systems (tens of layers, hundreds of servers, thousands or millions of replicas).

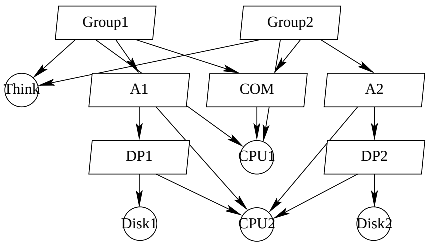


Fig. 1. A multi-tier client-server system from [2]. Tasks are represented by parallelograms. The customers are represented by the tasks Group1 and Group2. Pure servers, such as devices and think times for customers, are represented by circles.

The layered queueing model was first introduced as “Active Servers” [3], [4], describing the key property that a server may, during its service, stop for a nested request to another server. This was extended by Stochastic Rendezvous Networks (SRVN) [5], which treated waiting for each server separately, and the Method of Layers (MOL) [2] (a development of the ‘Lazy Boss’ algorithm [6]), which introduced the important concept of grouping the servers in “layer submodels”, at the cost of using a different model for software and hardware servers. From MOL and SRVN the Layered Queueing Network (LQN) model was created and evolved by adding features found in important application systems [7]–[12]. Other research on layered queueing includes:

- a model for a single (open) server with one layered service [13]
- an improved solver based on Markov Chain aggregation, for SRVN models with multi-class servers [14],
- a solver using a stronger approximation for non-exponential service times [15], and handling asynchronous messages,
- the Method Of Decompostion (MOD), developed to analyze layered software described in the UML [16]
- two different EQN solvers restricted to two-layer systems with software resources such as critical sections [17] or thread pools [18].

The modeling semantics and solution techniques of all of these models are subsumed and extended by LQN as described here, while retaining the solution efficiency and accuracy of the simpler forms. This paper describes the extensions in a unified way, and a solution technique adapted to them,

- *G. Franks and M. Woodside are with the Department of Systems and Computer Engineering, Carleton University, Ottawa ON K1S 5B6, Canada, e-mail: {greg,cmw}@sce.carleton.ca*
- *T. Al-Omari and S. Derisavi are with the IBM Toronto Lab, 8200 Warden Avenue Markham, Ontario, L6G 1C7, e-mail: {talomari,derisavi}@ca.ibm.com*
- *O. Das is with the Department of Electrical and Computer Engineering, Ryerson University, Toronto ON M5B 2K3, Canada, email: odas@ee.ryerson.ca*

Manuscript received Month XX, 2007; revised Month XX, 2008.

and implemented in the Layered Queueing Network Solver (LQNS). The extensions include common features of distributed systems, such as

- FULL-ACCESS: a server can issue requests to any server in a lower layer, rather than just to the layer below it. This is frequently the case in practice, a simple example is an application which makes requests to a database server, where they both use the same file server.
- MULTI: multithreaded and multiprocessor servers,
- ASYNC/OPEN: asynchronous messages, and open as well as closed models,
- ACTIVITY: a detailed execution graph for provision of a service, showing parallelism but also sequence, branching and loops.
- VAR: arbitrary variance of CPU demands
- SERV-PATTERN: both stochastic and deterministic patterns of requests for lower-layer service

and common performance optimizations such as

- PH2: servers with early replies and autonomous continuations, called a second phase. This reduces client blocking delays.
- PAR: parallelism in providing a service, used to represent prefetching, asynchronous remote procedure calls, and speculative computing, as well as parallelization of algorithms. This uses ACTIVITY.
- QC: consensus-based parallelism, requiring K out of N branches to complete,

Scalability of models and solutions is increased by

- REPL: explicit replicas of servers,
- REPL-BR: replicas of parallel branches,

and solver features have been introduced to improve the extended queueing network approximations:

- FAST: a fast-coupling correction for multi-class FIFO servers with different service times,
- INTERLOCK: a correction for correlated requests due to shared resources in generating arrivals.

Alone among the various LQ approaches [2], [5], [15], [17]–[20], the LQN solution algorithm handles all combinations of the above.

This paper gives a unified account of the LQN model and its solution techniques, emphasizing how the solver extensions are related. For example, servers with multiple services (described as *entries*, below) require a multiclass solver, while multiple threads use a multiserver solver, and both of these must be adapted for second phases and replicas.

2 Layered Queues

The central idea of the layered queueing (LQ) model is an Extended Queueing Network in which a service may have within it a nested service by another server, with nesting to any depth. This nested simultaneous resource possession permits an elegant compact representation. Further, the representation is designed to model directly the client-server type interactions commonly found in distributed systems, thus reducing the semantic gap between the model and the system being studied.

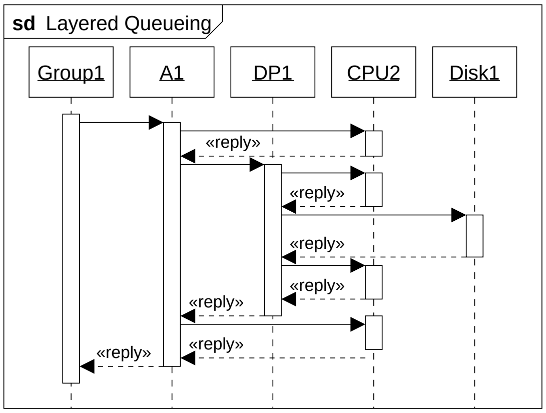


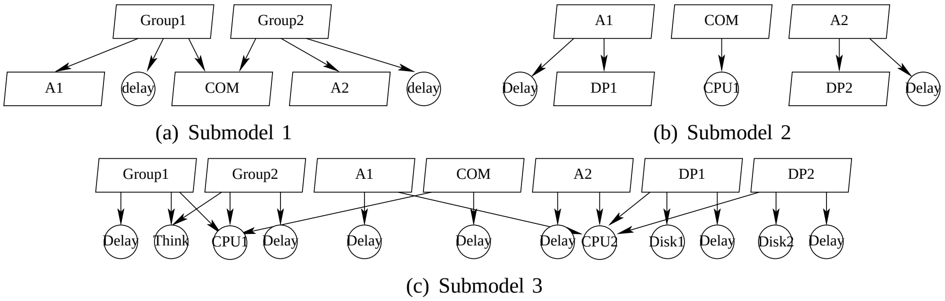
Fig. 2. A sequence diagram showing how service requests nest from Group1 to Disk1 in Figure 1. A request is blocked until its reply is received.

The example in Figure 1 (taken from [2]) is used here to describe the basic features of a layered queueing network model. The primary entities of the model consist of software servers called *tasks* shown as parallelograms, and hardware servers shown as circles. Tasks are used to represent any entity that can make requests to any other entity. For example, they can represent operating system processes, customers to the system and hardware devices such as disks. In Figure 1, the topmost tasks Group1 and Group2 are sources which make requests to servers A1, A2 and COM, which in turn make requests to lower servers and processors. Each service is a sequential process, and multiple requests are made sequentially. Servers which make no requests are called the hardware servers and behave like servers in a conventional queueing network. These servers can also supply pure delay, as shown by the infinite server Think which provides the thinking time for the users. Figure 2 shows one possible sequence of requests from Group1 to Disk1 illustrating the nesting of calls and the uniform treatment of hardware and software servers.

2.1 The Method of Layers (MOL)

The approach of MOL [2] will be used to describe the solution of Basic LQs. The service relationships are decomposed into a set of ordinary queueing networks, which are two-layer submodels showing clients in the upper layer requesting service from the lower layer, as shown in Figure 3 for the model of Figure 1. Each task appears as a server in one submodel, and again as a client in the next lower submodel. As a server, it is modified to include a surrogate delay (labeled Delay) representing the nested services in even lower layers. As a client, it has a surrogate delay to capture the delays between the requests it makes. The surrogate delays are calculated by a set of *import relationships* which are the core of the method (see [2]). The bottom layer submodel is constructed to include all the hardware servers.

MOL solves the separate submodels using the Linearizer approximate MVA algorithm [21]. Each submodel is a conventional separable queueing network in which the servers form the service centers and the clients form the customers. The



(a) Submodel 1

(b) Submodel 2

(c) Submodel 3

Fig. 3. Submodels generated by the Method of Layers for the model in Figure 1.

solver iterates between submodels by updating the surrogate delays with the import relationships.

In MOL, requests between tasks are strictly layered (i.e. they can only be made to the next layer down, except for processors; additional pseudo-tasks can be introduced to overcome this). This and other limitations are overcome by the LQN model.

2.2 The Layered Queueing Network (LQN) Model

The LQN model has gradually evolved to add features found in real systems, as listed in the Introduction [7]–[12], and is best described in the User Guide [22]. Some of the added features are illustrated in the example shown in Figure 4, based on [23]. It represents tasks and services in an Air Traffic Control (ATC) center in the US National Air Space infrastructure [24] for the airspace away from airports where aircraft normally fly at high altitudes. For example, the task labeled DM (data management) services the user consoles UI and makes requests to CR (conflict detection) which makes requests in turn to SP (signal processing) and Radars (operations of the radar sets). The notation will be introduced below with the features that are described.

2.2.1 Multiple classes of service at a server (MULTI)

Hardware and software servers are treated uniformly in LQN. Some software servers in Figure 4 offer more than one kind of service, indicated by small parallelograms nested inside a task (in LQN these are called *entries*). For example the FPM1 application has entries FPM1get and FPM1modify, which can have in general different CPU demands and different requests to lower servers. Since software servers usually have a FIFO discipline, this requires solving a multiclass FIFO queue.

In Figure 4, the stacked parallelograms indicate multi-threaded software servers, (which may run on multiprocessors, note that the processors are indicated by the dashed boxes in Figure 4). MOL also supports these multiservers, but only for a single class of service.

2.2.2 Asynchronous Messages and Open Arrivals (OPEN/ASYNC)

All of the requests between tasks shown in Figure 4 are synchronous, or blocking. The model also supports asynchronous requests, which do not block the caller and do not return information. Entries can also accept *open arrivals* with a Poisson arrival process.

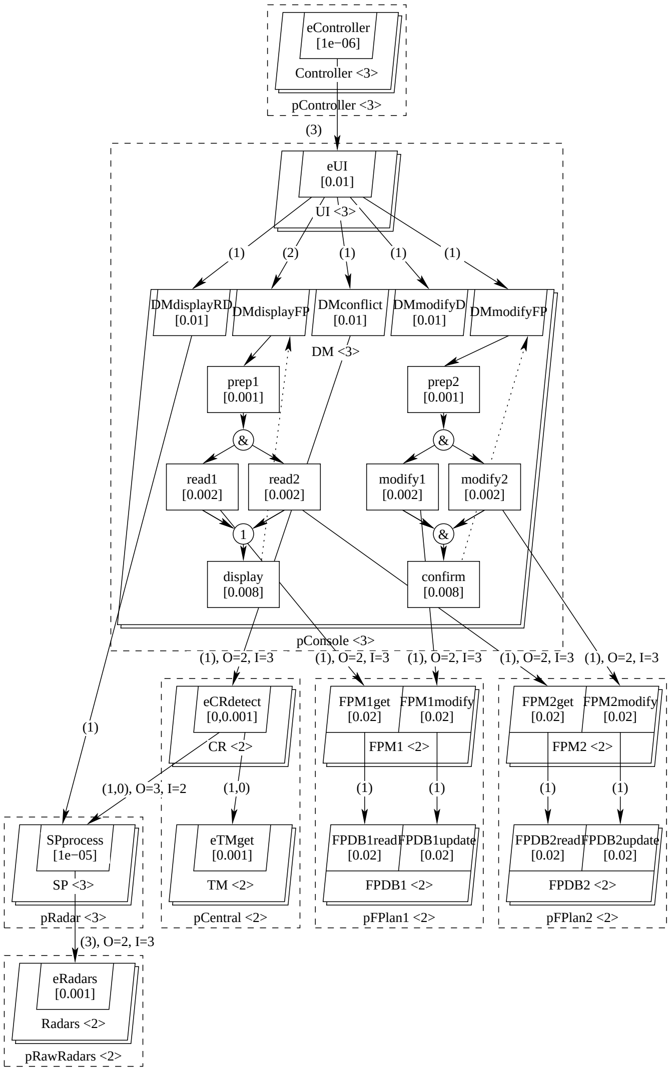


Fig. 4. A Layered Queueing Network (a model of an Air Traffic Control System studied in Section 6)

2.2.3 Second Phases (PH2)

An entry can send an *early reply* to its requester, and then continue to execute (called a second phase) [9]. In this case the requester and server execute in parallel for a time. Figure 5 shows timing detail for the execution of the eCRdetect entry of task CR. On a diagram it is indicated by making the CPU demands and service requests a pair of numbers, as shown for the eCRdetect entry. Early replies are often useful in real systems to improve performance, provided the server is not saturated [9]. Early replies also provide a modeling construct for shared buffers, for example in a file system [10].

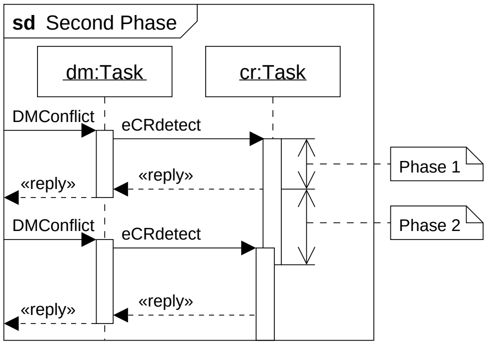


Fig. 5. First and second phases at a serving task. In this figure, the client, DM, sends a second request to the server, CR, before the server finishes processing the first request. This request will be queued if the server cannot start a second thread.

2.2.4 Parallel Activities (PAR) and Activity Detail

The large DM task has five entries, two of which are executed in parallel, as indicated by a small *activity graph* drawn inside the task. For instance entry DMmodifyFP invokes activity prep2, which then forks two parallel sub-threads for activities modify1 and modify2 which update different databases FPM1 and FPM2. The following join labeled '&' indicates that both paths must complete (fork-join parallelism). The dashed arrow back to the entry indicates the point at which the reply to the original request is generated (if there are subsequent activities they are part of a second phase).

The parallel branches for entry DMdisplayFP end in a join labeled '1', which indicates that only one of the two must terminate; it takes the first result. This is an example of Quorum Consensus, described further below.

Even without parallelism an activity graph can be used to build up an entry behaviour from a more detailed description, providing an execution graph for the entry [25]. At the level of an LQN model, an activity is the basic unit of behaviour. It includes CPU processing and nested service requests. An entry without explicit activity detail has by default one activity (or two, if there is a second phase).

2.2.5 Replication of Servers (REPL)

Many large systems have identical or nearly identical subsystems, which can be exploited for scalable representation and analytic solution with each replica represented only once. Solution effort becomes independent of the number of replicas [12]. Replication of an entity is a deeper form of multiple servers, in which replicas execute independently of each other.

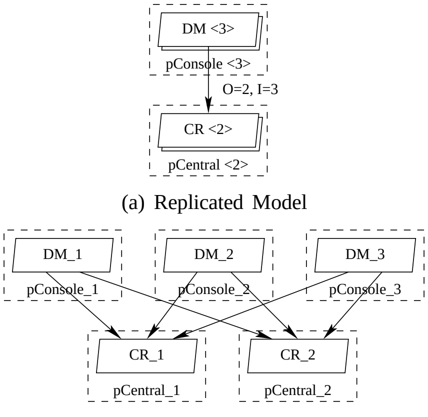
A replication notation, illustrated by the example in Figure 4, was developed in [11], [12] to exploit the symmetry in an LQN model. The interactions between DM and CR are expanded in Figure 6 to show the replicas explicitly. The notation adds three new elements:

- each replicated task and processor is represented once with a replication count r given in angle brackets, as $< r >$,
- each arc representing an interaction has a fanout count O, giving the number of target replicas for each source replica, and
- a fan-in count I giving the number of separate source replicas there are for each target replica.

These elements have default values of one.

In Figure 4 all tasks are replicated as well as being multithreaded, shown by the integer in angle brackets (e.g. $< 3 >$). In LQN the interpretation of a request to a replicated server is that one request is sent to one replica, chosen randomly. For a replicated server there is a subset that forms a pool used by each client replica, of size O=*fan-out number*, and similarly the set of client replicas that may make requests to each server replica has size I=*fan-in number*. An example is the request from entry DMconflict to entry eCRdetect, where all three DM replicas fan in to the two FPM1 replicas).

The fan-in/fan-out values in Figure 4 are artificially introduced here to illustrate the notation and the use of the solver. In an actual ATC system the replicas are used differently, for fault tolerance, as described in Section 6.



(a) Replicated Model

(b) Expanded Model

Fig. 6. Subset of the model in Figure 4 showing how the compact "replicated" notation is used to represent the conflict resolution subsystem consisting of three display managers and two conflict resolution tasks.

2.2.6 Quorum Consensus (QC)

In some systems with parallel execution, it is not necessary for all branches to complete. This is particularly true for voting or Quorum Consensus systems, in which N identical requests are made in parallel but K out of N replies (with values that agree) are sufficient to proceed. The LQN notation for QC is to label the parallel join node with the size K of the quorum, as in the node preceding the display activity in task DM in Figure 4. The analytic solution is discussed below.

2.3 LQN Meta-Model

The meta-model for a Layered Queueing Network, shown in Figure 7, is the formal model used to describe the information that goes into a LQN. An LQN model consists of a set of *processors* which contain *tasks*. Processors are used to consume "time", and often represent the actual CPUs in a distributed system. A processor is a *pure server* in that it can only receive requests for service from the tasks that it contains. A processor may have a *multiplicity* in which case it is a multi-server. If the multiplicity is *infinity*, then the server becomes a pure delay. A processor can be shown as a dashed rectangle enclosing its tasks, as in Figure 4, or as a circle with arcs attaching it to its tasks, as in Figure 3.

Tasks can represent different kinds of objects, i.e.:

- clients to the network,
- actual processes or threads in a system,
- non-processor devices such as disks,
- critical sections, and
- resources such as buffers.

The same task can act both as a *client* that makes requests and as a *server* that accepts requests. Tasks which do not accept any requests represent load-sources or users and are called *pure clients* or *"reference tasks"*. They correspond to customers in closed chains of conventional queueing networks. Tasks and processors have a multiplicity, which for a reference task gives the number of sources or customers, and for other tasks represents the resource multiplicity (e.g. the number of homogenous threads of control, or the number of buffers). An infinite multiplicity makes a task or processor a delay server.

Tasks receive requests in a single FIFO queue. Classes of service identified by *entries*. For consistency, reference tasks also have entries even though they do not accept requests. Once an entry accepts a request, actual processing is performed by *activities*, the lowest level of detail in the performance model. Activities are combined by *Pre* and *Post* precedence connectors expressing sequence, and *"Or"* and *"And"* forking and joining. Or-forks have probabilities, and Post-nodes can invoke a subset of the graph a random number of times with a given mean – the equivalent of a subroutine call – to define looping. Activities:

1) consume time by making requests to the processor associated with the task. Service time demands are shown on Figure 4 with labels in square brackets. This time demand is divided into *slices* between requests, as shown by the UML Sequence Diagram in Figure 8. It gives the details of the phase-1 activity of the entry SProcess in Figure 4), which alternates between slices of processing by the pRadar processor, and requests to entry eRadars. The mean number of slices is always $1 +$ (total Requests).

   By default, the demand of a *slice* is assumed to be exponentially distributed [5], but a variance may be specified.
2) make requests to other tasks through *Requests*. Requests are made to entries on other task and can be either *Synchronous* or *Asynchronous*. The mean number of requests per entry execution is shown in parentheses attached to the request arcs. By default the number of request is geometrically distributed with the specified mean [5]. The number can also be deterministic, though the order of requests is not defined (they can be invoked by separate activities if the order is significant).
3) reply to synchronous requests, shown using the dotted line within a task from the activity to the entry. The entry can either reply to the originating task, or *Forward* the request with some priority to one or more other entries.
4) invoke other activities through *Precedence*.

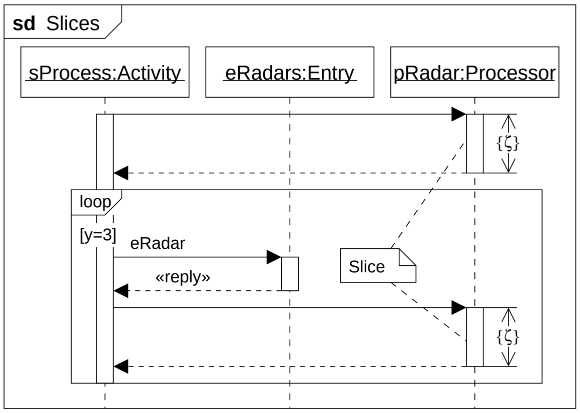


Fig. 8. Slices of CPU time between requests

The remaining tasks in Figure 4 use an abbreviated notation where one or two activities are invoked implicitly by an entry. The first activity implicitly replies for the entry. Service demands for this case are shown as a list of one or two items within square brackets, e.g. [0, 0.001] in entry eCRdetect.

3 Analytic Solution of LQNs

Algorithm 1 shows the overall algorithm used to solve layered queueing networks. The overall model is represented by a set of related submodels, each of which is solved using the Linearizer algorithm [21] of Mean Value Analysis (MVA) [26] with modifications to handle any two-phase servers [9], [10]. The sections that follow describe how the submodels are constructed, how they are solved, and finally, how the process is modified if replication is involved.

Algorithm 1 LQNS Algorithm

```
Load Model
Extend Model
Topological Sort
Layerize (create and initialize layer submodels)
repeat
    Solve the layer submodels using Linearizer MVA
until convergence or iteration limit
Save results
```

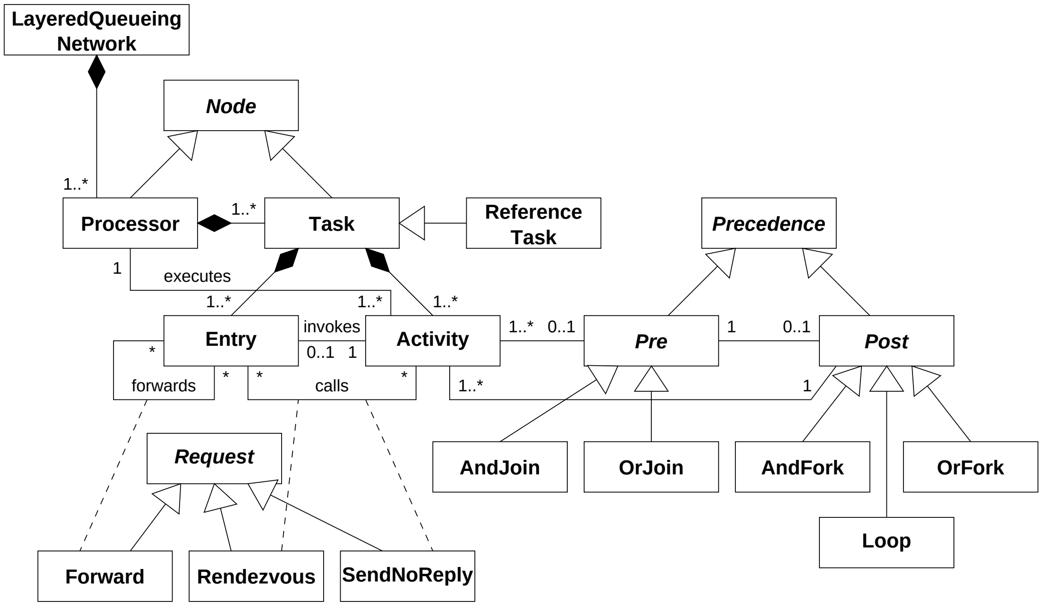


Fig. 7. Meta-model for Layered Queueing Networks

3.1 Submodel Construction

The topological sort identified in step 3 of Algorithm 1 assigns a nesting depth or layer s to each of the nodes in the input model. The layerizing step uses the nesting depth to generate submodels consisting of a set of servers and a set of clients. Submodels are created by layer. Submodel s is created with all of the tasks and processors at layer $s + 1$ as the servers, and all of the tasks that make requests to these servers as the clients. Figure 9 shows the submodels that arise from Figure 4. Notice that the processors shown as boxes in Figure 4 belong to the layer next below the lowest task in the box. Other layering strategies are possible. For example, the Stochastic Rendezvous Network (SRVN) Solver forms a submodel for each server in the model [5], whereas the Method of Layers (MOL) solver is similar to LQNS except that all of the processors in the model are grouped together in the lowest layer [2].

The routing chains created for MVA submodels depend on whether replication is present or not. When a submodel contains no replicated components, a chain is created for each client in the model. The number of customers in each chain is the lesser of the multiplicity of the task, or the number of clients of the task when it is acting as a server. Figure 10(a) shows the queueing network for Submodel 1 shown in Figure 9(a) assuming that there are no replicas.

When a submodel contains replicated components, a chain is created for each server in the submodel. Splitting the customer chains, according to the server they visit, is necessary if different fanout values can be applied to different server tasks in the LQN, since there is one server center in the layer submodel for each server task in the LQN. Figure 10(b) shows the queueing model for this case, with the flows labelled by their chain identifiers.

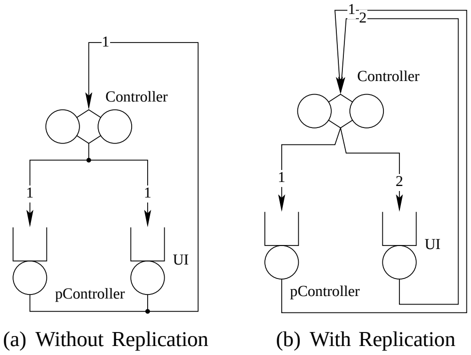
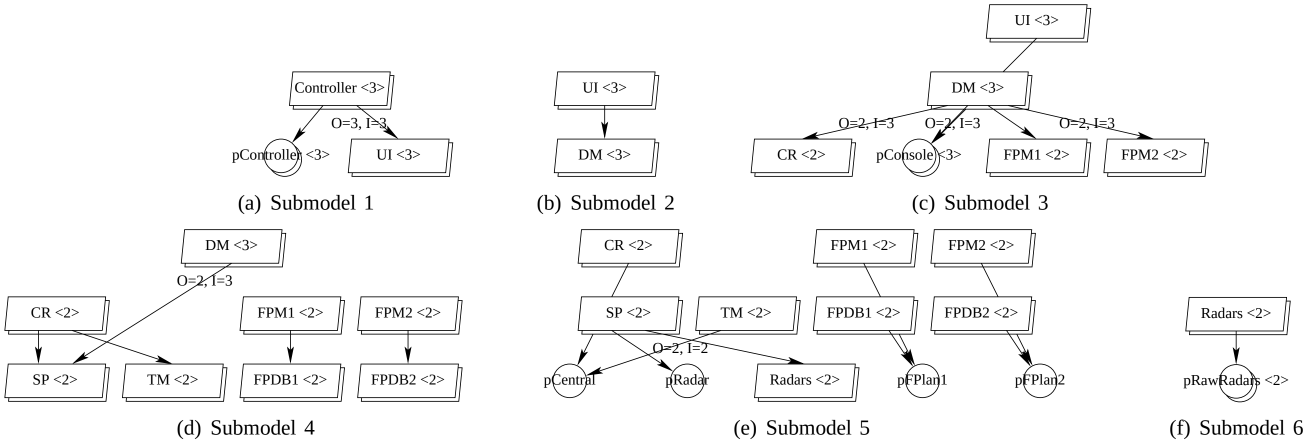


Fig. 10. MVA model for submodel 1 in Figure 9.

3.2 Submodel Parameterization

Service demands and think time parameters for each submodel are found from the results obtained in other submodels. The service time for a *client* in a submodel is found by summing up the waiting times (queueing time plus service time) to all of the tasks and processors it calls, that are outside the current submodel. The service time for a *server* in a submodel is found by summing up the waiting times to all of the tasks and processors it calls, including calls to entities in the the current submodel. For example, consider the task UI in Figure 4. In Submodel 1, UI acts as a server; its service time is found by summing up the waiting time for the requests it makes to the task DM, and to its processor, pConsole. In submodel 2, UI acts as a client to task DM. Its service time is the waiting time to its processor pConsole. Finally, in submodel 3 UI is a client to processor pConsole. Here, its service time is found by taking the sum of the waiting times to task DM. Note that in submodel 3, task DM is also acting as a client.

The other parameter that must be calculated from the solution of other submodels is the think time for each chain representing a client task. This value is derived from the



(a) Submodel 1

(b) Submodel 2

(c) Submodel 3

(d) Submodel 4

(e) Submodel 5

(f) Submodel 6

Fig. 9. Submodels for the model in shown in Figure 4. The objects in the bottom row of each submodel form the servers in the corresponding MVA queueing model. All the other objects form the clients.

throughput and utilization of the task when it is behaving as a server in a submodel, using Little's result. For example, the utilization and throughput for task UI found from the solution to submodel 1 is used to set the think time for this task in submodels 2 and 3.

Using this approach, service times for submodels are found starting from the bottom layer and working up, and think times are found top down. Deeper models require more iterations of the outer loop to solve than shallow models because of the need to propagate results from one layer to another in both directions.

Algorithm 2 Solve Layer Submodel

```
for all Clients do
    Calculate imported service and think times.
end for
for all Servers do
    Calculate imported mean and variance of service times.
end for
solve submodel using mixed-model MVA.
```

3.3 Submodels with Replication

The semantics of replication are illustrated by *flattening* a small part of the ATCS model (Figure 4) in Figure 6, to show each replica separately. When a client with rc replicas and fanout O requests an interaction with a server with rs replicas, the flattening allocates the $rc \times O$ flattened interactions sequentially to the server replicas, modulo rc. In passing, we note the constraint that $rc \times O = rs \times I$.

The LQN solution algorithm represents each replicated server (task or processor) by a single server. The layer submodels are adapted as follows:

- each surrogate delay in any layer submodel, representing the response time of a visit to a server, is replaced by $(\text{fan-out}) \times (\text{response time})$. This applies to surrogate delays in source chains, and in service times.
- each class of service in a submodel has a source chain for each replicated client, with population equal to the fan-in of the interaction. It includes a special delay term for visits to other replicas of the same server and class, equal to $(\text{fan-out} - 1) \times (\text{response time})$.

The latter change means that some service times in a layer submodel depend on results of the same submodel, which was resolved by iteration. An approximate multivariate Newton-Raphson iteration was used [27] for these variables.

3.4 Servers With Variance

Fixed-rate queueing stations for the MVA submodels are solved using servers which allow for variance using the approximation from [28].

3.4.1 Random Phases

The variance σ_i^2 at activity i with geometrically distributed requests to other servers j is the sum of a random number of random variables and is given by:

$$\sigma_i^2 = \sigma_\zeta^2 + \sum_j \left(\overline{y_{ij}}(\sigma_{ij}^2 + \sigma_\zeta^2) + \sigma_{y_{ij}}^2(\overline{\zeta} + \overline{x_j}) \right) \qquad (1)$$

where $\overline{\zeta}$ and σ_ζ^2 are the mean and variance of the service time of one slice, σ_{ij}^2 is the variance of the waiting for the request from activity i to entry j, $\sigma_{y_{ij}}^2$ is the variance in the number of requests from i to j, and $\overline{x_j}$ is the mean service time at entry j.

3.4.2 Deterministic Phases

The variance σ_i^2 at activity i with deterministically distributed requests to other servers is the sum of variances of all of the requests to other servers and is given by:

$$\sigma_i^2 = Y_i \sigma_\zeta^2 + \sum_j \left(y_{ij} \sigma_{ij}^2 \right) \qquad (2)$$

where the terms of the equation are the same as those for Eq. 1 and $Y_i = 1 + \sum_j y_{ij}$.

3.5 Servers with Two Phases

Special approximations are needed to solve queueing models which contain a two-phase server, shown earlier in Figure 5, because the second phase effectively creates a new customer in the queueing network, violating the conditions of product form queueing. Two new effects must be accounted for. First, a request from a client may find a server busy processing an earlier request made by that very client. This event is called *overtaking* and is shown in Figure 5 by the overlapping execution occurrence for Task CR. Second, the second phase makes the server work longer; this demand must be accounted for.

3.5.1 Overtaking Probability

The overtaking probability Γ_{ij} is the probability that a request made by a client i to a server j finds the server busy servicing the previous request made by client i. A simple approximation, used in [2], [5], approximates this probability as a race between two exponentially distributed random variables: the "return time" between requests to server j from client i, of mean τ_i, and the mean time server j's second phase takes to finish, s_{j2}. Thus:

$$\Gamma_{ij} = \frac{s_{j2}}{\tau_i + s_{j2}} \tag{3}$$

This expression works well provided that the client makes requests from only one phase, to one and only one server. If this restriction is violated, then the return time is not exponentially distributed and large errors can occur [9].

A more robust approximation for the overtaking probability, described in [9] is used by LQNS. A transient Markov chain is analyzed starting at state at the moment of the return, called state S_r. It gives the absorption probability $\Pr(\mathrm{OT}_p|S_r)$ for the overtaking of a client i in phase p, while its server j is executing in phase s while completing an earlier request from the client i in phase r. After considerable manipulation one obtains a relatively simple product-form expression for $\Pr(\mathrm{OT}_p|S_r)$ [9]. The overtaking probability for client i with P phases calling server j is then given by:

$$\Gamma_{ij} = \sum_{p=1}^{P}\sum_{r=0}^{P}\frac{\lambda_i y_{ijr}}{\lambda_{ij}}\Pr(\mathrm{OT}_p|S_r) \tag{4}$$

where:

$$\begin{aligned}
\lambda_i &= \text{Total throughput at client } i\\
\lambda_{ij} &= \text{Throughput from client } i \text{ to server } j\\
y_{ijp} &= \text{Mean requests from } i \text{ in phase } r \text{ to } j
\end{aligned}$$

3.5.2 Delay at Fixed-rate Servers

The usual MVA expression for the waiting time, W_{mk} at a FIFO server m in class k with non-exponential service times [28] (i.e. the first three terms in Eq. 5) is modified by adding two additional terms [9]. The fourth term accounts for overtaking while the final term accounts for the effect of the customer created by the phase-two service.

$$\begin{aligned}
W_{mk}(\mathbf{N}) &= s_{mk} + \sum_{j=1}^{K} s_{mj}Q_{mj}(\mathbf{N}-\mathbf{e}_k)\\
&+ \sum_{j=1}^{K} r_{mj}U_{mj}(\mathbf{N}-\mathbf{e}_k)\\
&+ \sum_{j=1}^{K} s_{mj2}\Gamma_{mj}(\mathbf{N}-\mathbf{e}_k)\\
&+ \sum_{j=1}^{K}(1-\Gamma_{mj}(\mathbf{N}-\mathbf{e}_k))\\
&\times \left[s_{mj1} + \frac{s_{mj2}^2}{s_{mj}}\right]U_{mj}(\mathbf{N}-\mathbf{e}_k)
\end{aligned} \tag{5}$$

3.5.3 Delay at Load-Dependent (multi-) Servers

In [2], a very simple expression was derived for finding the waiting time at a multi-server which did not involve the computation of the marginal probabilities. This expression was modified for LQNS by extending it to multiple classes, and second phases [10]:

$$\begin{aligned}
W_{mk}(\mathbf{N}) &= s_{mk1} + \frac{U_m^{(1)}(\mathbf{N}-\mathbf{e}_k)^M}{J_m}\\
&\times \sum_{j=1}^{K} s_{mj}\left[L_{mj}(\mathbf{N}-\mathbf{e}_k) + U_{cj2}(\mathbf{N}-\mathbf{e}_k)\right]\\
&+ \frac{\Gamma_{mk}}{J_m}\cdot s_{mk2}
\end{aligned} \tag{6}$$

4 Models with Internal Parallelism

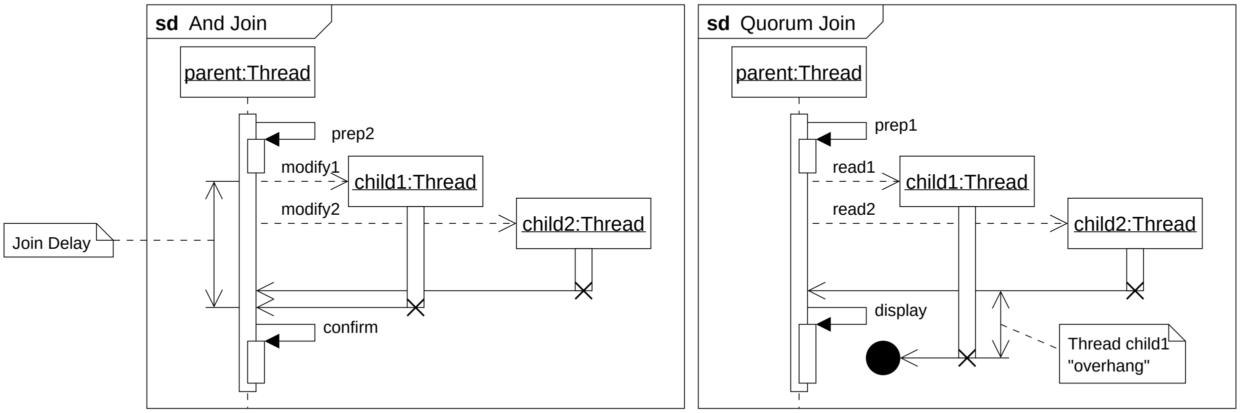
Models with internal parallelism arise when a task has internal activities which fork into separate threads which join at some later time, illustrated by the Task DM in Figure 4. Two cases exist, depending on whether some or all of the threads join or not, and are shown in Figure 11. For the case where all of the threads join (Figure 11(a)), the solution algorithm must be augmented to account for the *join delay* and to account for the additional customers in the underlying queueing network caused by the threads. For the QC join, where only a subset of the threads join, further approximations are required.

4.1 MVA Solution

The underlying strategy for solving a queueing network submodel containing parallel sections (including QC sections) is the complementary delays technique [29], with the accuracy improvements of [30]. The parameters for stations acting as servers and those acting as clients are calculated differently, described next.

4.1.1 Servers with Heterogeneous Threads

The service time for a task with internal parallelism acting as a server in a submodel is computed by aggregating the service times and variances of all of the internal activities into one or two phases, depending on the location of the reply. First, the



(a) And Join

(b) Quorum Join

Fig. 11. Task behaviour at joins

reduction finds all subgraphs of activities without any fork or join and reduces these to a single composite activity with a mean and a variance. Second, the overall delay over the set of composite activities between a fork and its corresponding join is calculated using the method described below. Finally, the fork, the join, and the corresponding branches are reduced to a single composite activity. This process is repeated until all of the forks and joins are removed.

4.1.2 Clients with Heterogeneous Threads

A different reduction process is applied to a task with heterogeneous threads acting as a client. For this case, each branch of a fork is represented as a distinct routing chain in the underlying queueing network and these routing chains are distinct from the routing chain of the parent. The service time for each branch of a fork is found by aggregating all of the activities between the fork and its corresponding join into a single composite activity. Similarly, all of the activities corresponding to the parent thread of the task are aggregated into a single composite activity for the parent's routing chain.

In [30], the probabilities of routing chains contending with each other (named *Overlap probabilities*) are used to modify the MVA waiting time expression to remove contention when routing chains cannot interfere with each other. For example, since a parent thread is blocked while the threads associated with each of the branches of its fork execute, a customer in the parent's routing chain cannot contend with any of the customers in the routing chains corresponding to the branches of the fork. The number of customers in the routing chains of the branches of a fork is inherited from the number of customers in their parent. While it is not possible for a customer in a branch chain to interfere with its corresponding customer in the parent, it can interfere with other customers in the parents chain, so the overlap probability is adjusted by $(N-1)/N$, an extension of the approach in [30].

4.2 Estimating Join Delays

The join delay depends of the entire probability distribution of the delays on the branches. For *AND-joins* the delay is the maximum of the branch values and the mean depends heavily on the distribution tails; the *three-point* approximation described in [31] was found to be highly effective [8]. However in a QC join where only K out of N branches need to complete, the details of the distributions are more important, and a better approximation to the branch delay is essential.

4.3 QC Delay

Most performance studies of QC delay use simulation [33], [34], however a rapid analytic approximation has advantages for exploring alternatives. The QC delay for a K-out-of$N$ quorum is the K^{th} out of N *order statistic* [32], $X_{\text{Quorum}} = \text{OS}(K, \{X_{\text{Branch},i}\}_{i=1}^{N})$.

The branch delay distribution was approximated in two ways. Where the number of requests to lower-level servers is deterministic, a Gamma distribution using the first two moments of $X_{\text{Branch},i}$ is used. When the number of requests to lower-level servers is distributed geometrically, a closed-form expression is used NEEDS REF TARIQ PAPER.

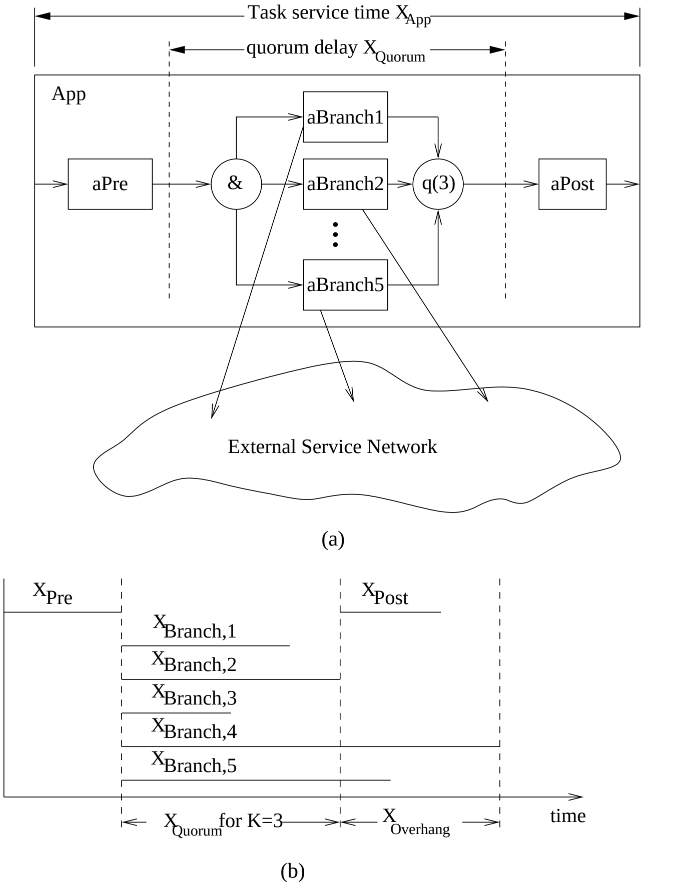
The QC behavior studied here is illustrated in Figure 12. We assume:

- a maximum of one QC section per software process
- that the QC section follows the pattern shown in Figure 12(a), but the activities aPre and aPost may be replaced by arbitrary activity subgraphs
- the QC task waits for the delayed branches (called the overhang) before becoming free.

Figure 12(a) shows the execution of an application App with a QC section shown by parallel branches terminating at the node labeled $q(3)$. The QC section is preceded by a set of operations represented by an aggregate activity aPre, and followed by operations represented by activity aPost. The QC section spawns (forks) $N = 5$ branches with activities aBranch$i$ for $i = 1, ..., 5$, and requires $K = 3$ responses.

The branch completion delays $X_{\text{Branch},i}$, for $i = 1, ..., 5$, are illustrated in Figure 12(b), showing the time at which the quorum of three responses is satisfied. Branches left out of the quorum are said to *overhang*. The overall application service time, X_{App} is:

$$X_{\text{App}} = X_{\text{Pre}} + X_{\text{Quorum}} + \max(X_{\text{Post}}, X_{\text{Overhang}}). \quad (7)$$



(a)

(b)

Fig. 12. Behavior of an application with a quorum consensus section.

The branch delay distributions from [35] are used to calculate the QC delay and to approximate the overhang effect, which can have an unbounded effect on the predicted performance measures. It is important because the application App cannot accept another request until the overhang is completed.

4.4 Solution Strategy for Models with Quorum

The solution strategy is to convert a model M with the QC section to an approximate model M' without one, and apply existing mean-value solution techniques that include fork-join parallelism.

The activity graph of a task with a QC section (such as task App in Figure 12(a)) is replaced in model M' with another activity graph as shown in Figure 13. The behavior of the QC section is changed to a full parallel section (denoted by '&' in Figure 13), followed by a second parallel section for the overhanging period. It has activity aPost and surrogate activities aOHLocal and aOHRemote for the overhanging branches. The model M' is constructed in this way to account for contention of branches for resources using the existing LQN constructs described earlier.

The delay for a branch is broken down into *local* delays at the host processor of App, and *remote* delays due to blocking for services at other servers, shown in Figure 14. It is assumed that all of the branches involved in the quorum join run on a common processor, so all of the requests to the processor are serialized into the overall delay S_{OHLocal}. It is also assumed that requests to any other resources can run in parallel. This time is combined into combined S_{OHRemote}. Because the overhanging branches are logically parallel, and the local delays are sequential, the surrogate delays are treated as partly parallel and partly sequential when they are combined.

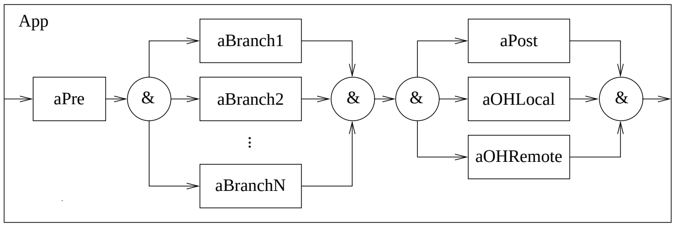


Fig. 13. Model M': the transformed activity graph for the model in Figure 12(a). M' is constructed in this way to account for contention of branches for resources.

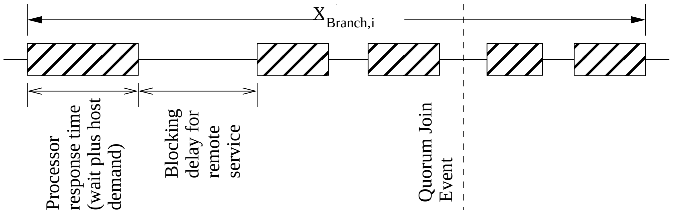

$X_{Branch,i}$
Processor response time (wait plus host demand)
Blocking delay for remote service
Quorum Join Event

Fig. 14. Demands and response times of a Branch. There are k blocking delays.

A different solution strategy would be to *decompose* each quorum construct of M for all possible combinations of active branches and then combine the results. This method is costly, because the number of models M' grows combinatorially with the number of quorum branches in each QC section and also exponentially with the number of QC sections in M.

5 Results and Analysis

In [35], the quorum delay approximation was evaluated using 110 tests covering parameter variations within six major cases, with sufficient accuracy in the vast majority of tests. The test parameters were chosen to stress the algorithm, rather than to favour it. Based on our experience with the analytic solution technique, the highest errors in almost all cases occur for the first order statistic, i.e., when $K = 1$. Two conditions may affect the accuracy of the approximate distributions:

1) The individual external service delays may not be exponentially distributed, homogeneous or independent.
2) Queueing delay on the processor that runs the quorum is assumed to be insignificant which will be satisfied if the load on the processor is light.

5.1 Scalability of Analytic Solution Time

The scalability of analytic solutions versus simulation is illustrated in Figure 15 which shows run-times (in seconds) for solving a replicated database model [35]. The application system with its two databases was replicated r times, with customers proportional to r and customer requests split equally

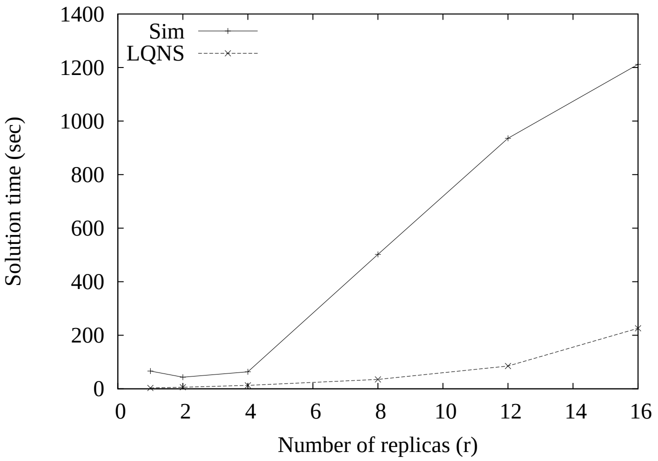


Fig. 15. Run time for simulation and LQNS analytic solutions.

among the applications. The simulation run-time was adjusted to be sufficient to give a 95% confidence interval of $\pm 5\%$. The simulation is one to two orders of magnitude slower. At larger numbers of replicas, LQNS runs out of memory due to inefficient coding for large numbers of chains, giving a space complexity of chains times entries. The algorithm should be linear in replicas.

6 Results: Air Traffic Control System (ATCS)

This section reports results for a model of a moderately large system, and extends a method for analyzing dependability, to cover all LQN features. The ATCS model shown in Figure 4 approximately represents an en route controller for a sector of airspace between airports. Each such system receives aircraft surveillance and weather data from radars and communicates with other external subsystems including other en-route facilities. Inside each ATCS, there are [23]:

1) A Display Manager DM which shows aircraft position information, takes inputs from air traffic controllers, and requests updates from SP and CR.
2) Surveillance Processing SP which receives radar data and identifies the aircraft tracks.
3) A Central subsystem including Trajectory Management (TM) and Conflict Resolution (CR) (on pCentral).
4) Flight Plan Management provided by two independent servers and databases FPM1/2 and FPDB1/2.

To illustrate the accuracy of the LQN algorithm, three versions of ATCS were analyzed:

NOREP: the replication parameters were all set to 1. This gives just one Controller and DisplayManager.

REP: the replication parameters in Figure 4 were used with fan-out and fan-in.

RECONF: the same replicas were used to provide fault-tolerance, and were reconfigured according to failure states of components; this is described below.

The read quorum for DMdisplayFP also took different values $K = 2$ (parallelism, with both branches executed) and $K = 1$ (quorum of 1). Analytic results were compared against simulations of the same model. Models with replication (shown as REP) were also *flattened* to the expanded form shown in Figure 6 as the simulator cannot solve replicated models directly. For these cases, results for the replicated and flattened analytic models were compared to the flattened simuation.

For systems without fault-tolerance, Table 1 compares the analytic solution for replication to simulation results for three cases:

Case 1: $K = 2$, NOREP. The example was solved with one replica of each task (that is, the replication values in the Figure were all set to 1), and the quorum value for the entry DMdisplayFP (the parallel graph on the left in DM) was changed from 1 to $K = 2$. This gives ordinary fork-join parallelism for reading from the two databases.

Case 2: $K = 1$, NOREP. The quorum value was set to $K = 1$, so DM only waits for the first result to be returned.

Case 3: $K = 1$, REP. The replication and fanin/fanout values shown in the Figure were used. Because fanout > 1 implies additional blocking for requests to replicas, the performance is reduced (this case however is meant to evaluate accuracy).

TABLE 1
Results for Response Time of the ATCS Without Reconfiguration for Fault Tolerance

Case	Simulation $\pm$ 95%	Analytic Result	Diff. (%)
1 Parallel ($K = 2$, NOREP)	2.670$\pm$0.0344	2.629	1.5
2 Quorum ($K = 1$, NOREP)	2.583$\pm$0.0700	2.524	2.3
3 Quorum and replica ($K = 1$, REP)	1.887$\pm$0.0059	1.574	16.6

These results show that for the parallel and quorum cases without replication the difference between the simulation and analytic results is just one or two percent, and is less than the statistical error at the level of 95% confidence. In the third case with replication the errors are higher, but still usable.

6.1 Model with Reconfiguration for Failures, and Dependable-LQN Analysis

The ATCS requires high reliability, which can be analyzed by an LQN extension called *Dependable-LQN* [23], [36]. Dependable-LQN has been extended here to deal with activities to define execution of an entry, and quorum computations with K $<$ N (which affect system failures). To improve reliability, server and processor replicas were used to reconfigure the system when an element fails (the RECONF cases). In ATCS, replicas were configured as follows:

- the three DM replicas were load balancing, that is the controller requests were distributed across the set of non-failed DM tasks,

- primary-standby replication was used for the Central and FPM subsystems, so each request went to one replica and the other was used only if the primary failed.
- the SP and Radar servers used LQN replication semantics (each SP server requests the raw radar data from both radars).

To reconfigure when a processor fails or a task crashes, requests going to it are diverted either to its partners (in load-balancing replication) or to the standby element.

Dependable-LQN modeling [23], [36] describes the failure and repair of tasks and processors by failure and repair rates, assuming independent failures. Dependent failures (due to servers which depend on other servers and on their processors) are captured automatically, and additional dependencies can be modeled explicitly. For ATCS, the tasks Controller, Radars and their associated processors are assumed to be fully reliable. The other failure and repair parameters are set arbitrarily as in [23] to be:

- processor mean time to failure (MTTF) is a year,
- processor mean time to repair (MTTR) is 2 days,
- software process MTTF = 30 days,
- software process MTTR = 1 hour.

Markov chain analysis of separate elements gives steady-state failure probabilities to be 0.00545 for processors and 0.00139 processes. The *failure state* of the system includes the failure status of every task and processor.

Many failure states are equivalent, in the sense that they determine the same set of usable elements (taking into account service dependencies) and give the same performance model. Each of these equivalence classes of states has an *operational configuration* which defines its performance model, and an aggregate steady-state probability. The operational configuration probabilities are found by generating and solving a non-coherent fault tree [37] (non-coherent because there can be a mixture of available and unavailable components in an operational configuration) using the Relex tool [38]. The LQN for each configuration is solved to determine its performance, and an overall average performance. The average throughput capacity now includes periods with reduced capacity due to failed servers.

Results in [23] show the analysis is both accurate and fast and that the aggregation of operational configurations in the Dependable-LQN technique reduces the number of performance models that must be solved by up to two orders of magnitude in many cases.

6.2 Extension of Dependable-LQN Analysis for Quorum Consensus

This work introduces parallel and quorum execution into the analysis. Parallel branches with K = N can be evaluated without change, but when K < N some extensions are necessary. The improvement of reliability due to the quorum section (since only K responses are needed) must be determined, Also it increases the number of configurations, and this can be countered by aggregation.

The Dependable-LQN analysis is extended here to accommodate symmetrical parallel or QC branches, as in the ATCS example. For instance, the left branch and right branches of the QC for DMdisplayFP in Figure 4 are symmetrical. On the left, activity read1 is failed if there is an unrecovered software or hardware failure in FPM1 or FPDB1, on which it depends. For each operational configuration with activity read1 operational and activity read2 failed, there is a corresponding configuration with the failures reversed, and with the same performance. These configurations are combined to give a smaller number of *aggregated operational configurations*, and thus to reduce the number of LQN solutions. We can illustrate this with four cases:

Case 4: (QC, $K = 2$ NOREP) In this simple case (which does not require the extension to Dependable-LQN described here) the replication of all tasks and processors is set to 1. Any failure causes a system failure. This is the same as Case 1 but with failures, and has one aggregated configuration.

Case 5: (QC, $K = 1$, NOREP) This is still a simple case. Dependable-LQN finds 3 operational configurations for the three failure states of the QC activities (none failed, left branch failed, right branch failed) and two aggregated configurations (no branch failed, one branch failed).

Case 6: (QC, $K = 2$, RECONF) The replication of tasks and processors is as given in Figure 4, and tasks are reconfigured for fault tolerance. There are three aggregated configurations.

Case 7: (QC, $K = 1$, RECONF) Replicas are reconfigured. Twelve aggregated configurations were found.

The overall throughput and response time results for these cases are summarized in Tables 2 and 3 respectively.

TABLE 2
Results for Throughput of the ATCS with QC and Failures

Case (All cases include QC)		Thruput Capacity /sec	Failure Prob. ($\times 10^{-3}$)	Aggreg. Configs	Total Operat'l Configs
4	$K = 2$, NOREP	1.1422	44.3	1	1
5	$K = 1$, NOREP	1.1977	41.7	2	2
6	$K = 2$, RECONF	1.7647	0.251	3	168
7	$K = 1$, RECONF	1.8238	0.249	12	1512

TABLE 3
Results for Response Time with QC and Failures

Case		sim	flat		replicated	
				% diff		% diff
4	$K = 2$, NOREP	2.346	2.401	2.37	–	–
5	$K = 1$, NOREP	1.795	1.808	0.72	–	–
6	$K = 2$, RECONF	1.951	1.639	16.01	1.701	12.82
7	$K = 1$, RECONF	1.916	1.594	16.83	1.659	13.45

The results show how replication with reconfiguration improves both performance and reliability, and the use of a quorum with K = 1 provides small additional improvements to both. The reduction in effort due to aggregating the oper-

TABLE 4
Solution Run Times for Cases with Reconfiguration

Case	Aggr	simulation		flat		replicated	
	cfgs	mean	σ	mean	σ	mean	σ
6	3	176.67	9.39	4.33	2.05	4	2.16
7	12	420.5	49.25	13.89	10.53	24.84	18.26

ational configurations into Aggregated Configurations is very worthwhile (from 1512, down to 12, in Case 7).

The use of replication for reconfiguration cannot however be compared directly with the basic replicated structure in Case 3 (with fanout) because the fanout implies multiple sequential requests, which reduce performance.

The value of the analytic solver is faster solutions. Table 4 shows the mean run times for the RECONF results shown in Table 2. The analytic solution is from 15 to 40 times faster than the simulation of the same model, when simulations are run to provide 95% confidence intervals no greater than $\pm 5\%$.

7 Conclusions

The assembly of approximations in LQNS covers more system features than any other attempt to solve layered queueing systems, as summarized in Table 5. The solution accuracy of individual LQNS features (as reported in the references where each feature was introduced), and also for the other LQ algorithms referenced in Table 5, is generally less than 10% error (and mostly less than 2% error). Two particularly accurate solvers are TDA [39] and [15]. However both these algorithms solve systems with a limited range of features compared to the LQNS algorithm.

The solution accuracy for a single model combining many features was investigated by comparing to simulations, in the ATCS case study of Section 6. Errors were less than 2% except where the replication feature, combined with a quorum, gives larger (about 15%) errors. However for the preliminary evaluation of high-level system descriptions, this accuracy is sufficient.

The algorithms are highly scalable. Our experience, not all reported here, has been that systems up to 100 tasks are solved in a few seconds in most cases. Occasionally, as in other extended queuing techniques requiring iteration, the iteration of Algorithm 1 fails to converge even when under-relaxation is applied to the iteration. The replication feature gives a computational complexity which is completely insensitive to the number of replicas of any task, making it feasible to model very large systems which combine replicas of a modest number of different tasks.

Quorum joins (which take the first K out of N parallel responses from lower level servers) are a recent addition to the feature set of LQN, and they have been included in the ATCS study. Quorum joins improve both performance and reliability. The *Dependable-LQN* technique was extended to analyze performance and failure probability in systems with symmetrical quorum joins. The scalability of the technique was improved by a new aggregation of operational configurations, reducing the number of analytic models which needed to be solved by two orders of magnitude. The results showed however that the quorum join made only small improvements to the performance and the failure probability.

Acknowledgment

This research was supported by the Natural Sciences and Engineering Research Council of Canada and by Communications and Information Technology Ontario (CITO).

References

[1] P. Maly and C. M. Woodside, “Layered modeling of hardware and software, with application to a LAN extension router,” in *11th International Conference on Computer Performance Evaluation; Modelling Techniques and Tools TOOLS 2000*, ser. Lecture Notes in Computer Science, no. 1786. Schaumberg, IL: Springer-Verlag, Mar.27–31 2000, pp. 10 – 24.

[2] J. A. Rolia and K. A. Sevcik, “The method of layers,” *IEEE Trans. Software Engineering*, vol. 21, no. 8, pp. 689–700, Aug. 1995.

[3] C. M. Woodside, E. Neron, E. D. Ho, and B. Mondoux, “An “active server” model for the performance of parallel programs written using rendezvous,” *Journal of Systems and Software*, pp. 844–848, 1986.

[4] C. M. Woodside, “Throughput calculation for basic stochastic rendezvous networks,” *Performance Evaluation*, vol. 9, pp. 143–160, 1989.

[5] 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. Computers*, vol. 44, no. 8, pp. 20–34, Aug. 1995.

[6] J. A. Rolia, “Performance estimates for systems with software servers: The lazy boss method,” in *VIII SCCC International Conference On Computer Science*, I. Casas, Ed. Santiago, Chile: Chilean Computer Science Society, Jul. 1988, pp. 25–43.

[7] G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, and M. Woodside, “Performance analysis of distributed server systems,” in *The Sixth International Conference on Software Quality (6ICSQ)*. Ottawa, Ontario, Canada: American Society for Quality Control (ASQC), Oct. 1996, pp. 15–26.

[8] G. Franks and M. Woodside, “Performance of multi-level client-server systems with parallel service operations,” in *Proceedings of the First International Workshop on Software and Performance (WOSP ’98)*, ACM Sigmetrics. Santa Fe, NM: Association for Computing Machinery, Oct. 12–16 1998, pp. 120–130.

[9] ——, “Effectiveness of early replies in client-server systems,” *Performance Evaluation*, vol. 36, no. 1, pp. 165–184, Aug. 1999.

[10] ——, “Multiclass multiservers with deferred operations in layered queueing networks, with software system applications,” in *Proceedings of the Twelfth IEEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2004)*, D. DeGroot and P. Harrison, Eds. Volendam, The Netherlands: IEEE Computer Society Press, Oct. 2004, pp. 239–248.

[11] T. Omari, G. Franks, M. Woodside, and A. Pan, “Solving layered queueing networks of large client-server systems with symmetric replication,” in *Proceedings of the Fifth International Workshop on Software and Performance (WOSP ’05)*, ACM Sigmetrics. Palma de Mallorca, Spain: Association for Computing Machinery, Jul. 11–14 2005, pp. 159–166.

[12] ——, “Efficient performance models for layered server systems with replicated servers and parallel behaviour,” *Journal of Systems and Software*, vol. 80, no. 4, pp. 510–527, Apr. 2007.

[13] M. L. Fontenot, “Software congestion, mobile servers, and the hyperbolic model,” *IEEE Trans. Software Engineering*, vol. SE-15, no. 8, pp. 947–962, Aug. 1989.

[14] D. C. Petriu and C. M. Woodside, “Approximate MVA for software client/server models by markov chain task-directed aggregation,” in *The Third IEEE Symposium on Parallel and Distributed Processing*. Dallas, Texas: I.E.E.E., Dec. 1991.

[15] S. Ramesh and H. G. Perros, “A multi-layer client-server queueing network model with synchronous and asynchronous messages,” in *Proceedings of the First International Workshop on Software and Performance (WOSP ’98)*, ACM Sigmetrics. Santa Fe, NM: Association for Computing Machinery, Oct. 12–16 1998, pp. 107–119.

[16] P. Kähkipuro, “UML based performance modeling framework for object oriented systems,” in *UML ’99, The Unified Modeling Language, Beyond the Standard*, ser. Lecture Notes in Computer Science, vol. 1723. Berlin: Springer-Verlag, 1999, pp. 356–371.

TABLE 5
Solver Features

Feature	LQNS	MOL [2]	SRVN [5]	TDA [39]	Ramesh [15]	MOD [16]	Fontenot [13]	SQN-HQN [17]	Kurasugi [18]	APERA [40]
FULL-ACCESS	yes	no	yes	yes	no	no	no	no	no	yes
Device Scheduling	FHPS	FPHS	FPH	F	FP	FP	F	FP	FP	FP
Task Scheduling	FH	F	F	F	F	F	F	F	F	F
Open arrivals (OPEN)	yes	no	yes	no	?	?	yes	yes	yes	?
MULTI	yes	yes	no	no	no	?	no	yes	yes	yes
Infinite-servers	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
SERV-PATTERN	SD	S	SD	S	SD	D	S	SD	SD	SD
VAR	yes	yes	yes	no	yes	no	?	yes	no	no
PAR	yes	no	no	no	no	no	no	no	no	no
REPL	yes	yes	no	no	no	no	no	no	no	?
ASYNC	yes	no	yes	no	yes	?	no	no	yes	?
Forwarding	yes	no	no	no	no	no	no	no	no	no
FAST, INTERLOCK	yes	no	yes	no	no	no	no	no	no	no

where F: FIFO, P: Preemptive Priority, H: Head-of-Line Priority, R: Random, S: Processor Sharing
S: Stochastic Phase, D: Deterministic Phase, ?: Unclear from the reference

[17] D. A. Menascé, "Two-level iterative queuing modeling of software contention," in *Proceedings of the Tenth IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2002)*, Fort Worth, TX, Oct. 12–16 2002.

[18] T. Kurasugi and I. Kino, "Approximation methods for two-layer queueing models," *Performance Evaluation*, vol. 36–37, pp. 55–70, Aug. 1999.

[19] R. G. Franks, "Performance analysis of distributed server systems," Ph.D. dissertation, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, Dec. 1999.

[20] P. Kähkipuro, "Performance modeling framework for CORBA based distributed systems," Ph.D. dissertation, Department of Computer Science, Univeristy of Helsinki, May 2000. [Online]. Available: http://citeseer.ist.psu.edu/600340.html

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

[22] G. Franks, P. Maly, M. Woodside, D. C. Petriu, and A. Hubbard, *Layered Queueing Network Solver and Simulator User Manual*, Real-time and Distributed Systems Lab, Carleton University, Ottawa, http://www.sce.carleton.ca/rads/lqn/lqn-documentation/.

[23] O. Das and M. Woodside, "Dependability modeling of selfhealing client-server applications," in *Architecting Dependable Systems II*, ser. Lecture Notes in Computer Science, R. D. Lemos, C. Gacek, and A. Romanovsky, Eds. Springer-Verlag, Dec. 2004, vol. 3069, pp. 266–285.

[24] A. M. Bayen, "Computational control of networks of dynamical systems: Application to the national airspace system," Ph.D. dissertation, Stanford University, 2003.

[25] C. U. Smith, *Performance Engineering of Software Systems*, ser. The SEI Series in Software Engineering. Addison Wesley, 1990.

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

[27] A. M. Pan, "Solving stochastic rendezvous networks of large client-server systems with symmetric replication," Master's thesis, Department of Systems and Computer Engineering, Carleton University, Sep. 1996, oCIEE-96-06.

[28] M. Reiser, "A queueing network analysis of computer communication networks with window flow control," *IEEE Transactions on Communications*, vol. COM-27, no. 8, pp. 1199 – 1209, Aug. 1979.

[29] P. Heidelberger and K. S. Trivedi, "Analytic queueing models for programs with internal concurrency," *IEEE Trans. Computers*, vol. 32, no. 1, pp. 73–82, Jan. 1983.

[30] V. W. Mak and S. F. Lundstrom, "Predicting performance of parallel computations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 3, pp. 257–270, Jul. 1990.

[31] X. Jiang, "Evaluation of approximation for response time of parallel task graph model," Master's thesis, Department of Systems and Computer Engineering, Carleton University, Canada, Apr. 1996.

[32] R. A. Sahner and K. S. Trivedi, "Performance and reliability analysis using directed acyclic graphs," *IEEE Trans. Software Engineering*, vol. 13, no. 10, pp. 1105–1114, Oct. 1987.

[33] A. Kumar, "Performance analysis of a hierarchical quorum consensus algorithm for replicated objects," *The 10th International Conference on Distributed Computing Systems*, pp. 378–385, 1990.

[34] M. L. Liu, D. Agrawal, and A. E. Abbadi, "On the implementation of the quorum concensus protocol," *Parallel and Distributed Computing Systems*, 1995.

[35] T. Omari, S. Derisavi, and G. Franks, "Deriving distribution of thread service time in layered queueing networks," in *Proceedings of the Sixth International Workshop on Software and Performance (WOSP '07)*, ACM Sigmetrics. Buenos Aires, Argentina: ACM Press, Feb. 5–7 2007, pp. 66–77.

[36] O. Das, "Dependability modelling of layered systems," Ph.D. dissertation, Carleton University, Ottawa, Canada, 2004.

[37] Y. Dutuit and A. Rauzy, "Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within aralia," *Reliability Engineering and System Safety*, vol. 58, no. 2, pp. 127–144, Nov. 1997.

[38] http://www.relex.com/products/pdfs/relex_ft_ds.pdf.

[39] D. C. Petriu, "Approximate mean value analysis of client–server systems with multi-class requests," in *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.* Nashville, TN, U.S.A.: A.C.M. SIGMETRICS, May 1994, pp. 77–86.

[40] M. Litoiu, "Application performance evaluator and resource allocation tool," http://www.alphaworks.ibm.com/tech/apera, May 2003.

[41] *Proceedings of the First International Workshop on Software and Performance (WOSP '98)*, ACM Sigmetrics. Santa Fe, NM: Association for Computing Machinery, Oct. 1998.