Solving Stochastic Rendezvous Networks of Large Client-Server Systems with Symmetric Replication

by

Amy M. Pan

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

Master of Engineering

Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada, K1S 5B6

5 September 2001

The undersigned recommend to the Faculty of Graduate Studies
and Research acceptance of the thesis

Solving Stochastic Rendezvous Networks of Large Client-Server Systems with Symmetric Replication

submitted by Amy M. Pan, B.Eng.
in partial fulfillment of the requirements for
the degree of Master of Engineering

Chair, Department of Systems and Computer Engineering

Thesis Supervisor

Carleton University
5 September 2001

Abstract

As industry moves towards distributed client-server systems, the performance of these systems has gained attention. Distributed systems may consist of a large number of components with functionality distributed throughout the components. Analytical methods, which use queueing theory, are suitable for the performance study of large systems. However, for practical purposes, they may be limited to the size of the system that can be solved. This thesis extends the performance analysis toolset, Stochastic Rendezvous Network (SRVN) models and the Layered Queueing Network Solver (LQNS), to handle large systems with groups of so called replicated components that are homogeneous from a performance modeling point of view. The analytical method described removes the limitation on the size of these large systems that can be analyzed by the toolset. The method includes the use of a simple model definition for large systems and a solver that is quick and memory efficient.

Acknowledgments

I would like to thank my supervisor, Professor C. Murray Woodside, for his guidance and advice throughout this thesis. I would also like to thank Greg Franks for his assistance and explanations. Thanks go to my fellow grad students and the performance group for their support, and to Curtis Hrischuk for the use of his thesis template. I am grateful to my family and friends for their moral support and encouragement.

Finally, I wish to thank Bell Sygma for funding this research.

Table of Contents

List of Figures

List of Tables

List of Symbols

Replication Notation

$\overline{F}_{AB}$	Fan-out of task A to B
F_{AB}	Fan-in of task B from A
K_A	Number of replicas of task A
S_A	Phase service time of task A

MVA

$D_{c,k}$	Service demand of chain c at device k
Nc	Number of customers in chain c
Q_k	Average number of customers at device k
$Q_{c,k}$	Average number of customers at chain c at device k
$R_{c,k}$	Residence time of chain c at device k
X_c	Chain c system throughput
Z_c	Think time of chain c

Replication Method

L_{ks}	Queue length of chain k at server s
N_k	Number of tokens for chain k
R_{km}	Residence time of chain k at station m
S'_{kt}	Modified service time of client station t for chain k
S_{kt}	Service time of client station t for chain k
V_{km}	Visits to station m by chain k
W_{km}	Waiting time of chain k at station m
X_{km}	Throughput of chain k at station m
X_m	Throughput of station m
Y_{ks}	Number of visits to server s by chain k
Z_k	Surrogate delay of chain k

Chapter 1.0 Introduction

1.1 Performance Analysis

The performance of distributed systems is gaining interest because of the trend in industry towards replacing mainframe systems with client-server systems. Client-server systems for business may contain a large number of components with functionality distributed throughout the system. Performance studies are conducted to analyze these large systems for purposes of capacity planning and software performance engineering. However, the practicality of modeling and analyzing these systems may be limited by the size of the system. The purpose of this thesis is to provide analytical means whereby very large client-server systems may be evaluated for their performance.

A client-server system consists of a client entity which is any process that issues requests to other server processes via a remote procedure call (RPC). The server only responds to requests and hides the details of the server environment from the client. The client and server may reside on separate processors and delays may occur in their communication due to network delays and overhead processing. An example of a client-server system is a network file server with the applications running on the workstations as the clients. An entity may act as a server to another entity but may itself be a client requesting service from another server. In such multi-tiered systems, the performance issues are more complex and unpredictable since the response time of an entity is not only dependent on the server it calls directly but also on any other underlying servers. An example of a multi-tiered client-server system is a distributed on-line transaction processing (OLTP) system where a transaction manager receives requests from client application tasks. The transaction manager may translate the requests and forward them on

to a database manager. In this case, the transaction manager is an intermediate layer which acts as a client to the database manager while acting as a server to the application task.

Capacity planning and software performance engineering (SPE) are two activities which study the performance of systems such as client-server systems. Capacity planning predicts whether a computer system can support the growth of existing applications, a change in system parameters and configuration, or the addition of new applications without violating user-specified service levels such as a specific response time or processor utilization [Menasce 94]. (The reader is referred to the book by Menasce et al. [Menasce 94] for a complete description of capacity planning.) Software performance engineering (SPE) is the process of constructing software systems that meet performance objectives. SPE is used throughput the software engineering life-cycle. It is used to discard unacceptable designs and to identify designs likely to yield satisfactory performance results [Menasce 94]. (The reader is referred to the book by Connie Smith [Smith 90] for a more complete description of SPE.) Both capacity planning and SPE use performance modeling techniques in the performance analysis of systems.

A performance model represents the behavior of a system in terms of its performance [Menasce 94]. It may be used to model an existing system or one that is being developed. The parameters of the model of an existing system may be obtained through measurement while for future systems the parameters must be estimated or guessed. The parameters of interest are the service demands and visit ratios of each system entity. In addition, for transaction processing systems, the parameters needed are the average transaction arrival rate and the maximum degree of multiprogramming which is the number of transactions that can be in execution at a given time. For interactive systems, the additional parameters are the average think time which is the time a user takes before issuing a command, the number of terminals, and the maximum degree of multiprogramming.

The results of solving the performance model are the performance metrics or measures. The desired metrics are response times, throughput and utilization. A response time is the time interval between a user's request to the system and when the user receives a response from the system. The throughput is the rate at which the requests are serviced by the system [Jain 91]. Typically, it is desirable for a system to have a short response time and high throughput. To improve the performance, the bottleneck of the system must often be found. The bottleneck is the component with the highest utilization. The utilization of a component is the fraction of time the component is busy servicing requests [Jain 91].

There are two main ways of solving performance models, simulation and analytical solutions. Simulation models are based on computer programs that emulate the different dynamic aspects of a system and their static structure. Simulation models require a great deal of detail thereby making them expensive to run and develop. Analytical models, on the other hand, are based on sets of formulae or computational algorithms which provide the values of desired performance metrics as a function of the set of values of the performance parameters [Menasce 94]. Although simulation models may provide more accurate results, analytical models are useful for analyzing large systems since they are computationally more efficient and their parameters are easier to obtain due to their higher level of abstraction.

In this thesis, the focus is on analytical methods for solving performance models of large client-server systems. The Stochastic Rendezvous Network Model (SRVN) proposed by Woodside [Wood 95a] and the Layered Queueing Network Solver (LQNS) [Franks 94] [Rolia 95] are the basis for solving such systems. The SRVN is used to model a client-server system, and LQNS is used to solve the SRVN model.

1.2 Motivation

This research was prompted by the performance study of a large industrial two-tiered client-server system. The system consists of several local-area networks (LANs) connected to two mainframes via a wide-area network (WAN) and backbone network. Each LAN consists of workstations which are serviced by a local server. In turn, the local servers seek service from the two mainframes. The number of workstations is in the thousands while the LANs are in the order of hundreds. Several performance issues are of interest such as where to deploy the software functionality to give the highest overall system throughput and where the bottleneck areas are if configurations are changed (e.g. more workstations are added). Because of the large size of the system, an analytical method is deemed to be more suitable than simulation in evaluating the performance of the system. This particular system is just one example of a new and widespread class of systems, as already described. The research applies to the broader class as well, including multi-tiered systems.

Analytical tools are available for studying the software performance of client-server systems, but their scope is limited by the size of the system. One such toolset is the Stochastic Rendezvous Network (SRVN) proposed by Woodside with the underlying Layered Queueing Network Solver (LQNS). The SRVN model is solved using queueing theory by the LQNS software tool. For studying large client-server systems (greater than 100 tasks), such as the industrial system in question, a group of client tasks or a group of server tasks may have similar performance parameters and thus similar performance metrics. It may also be possible to simplify the model by generalizing some tasks. The replication of these tasks has the advantage of simplifying the SRVN model representation of the system as well as in simplifying the computation. The thesis describes a simplified notation for replication and an algorithm that uses the replication of tasks to save on the

computational time and memory used by the LQNS. In simplifying the SRVN model, the subsequent presentation of performance metrics is also reduced.

1.3 Contributions to Research

The contributions made by this thesis are:

- The details of a model for simplifying the representation of large systems for performance analysis, suggested by Professor C. Murray Woodside [Wood 95b], are developed. Fewer entities and interactions are required in the model.
- A method, based on the method of surrogate delays, that enables the solving of the performance model of large systems is proposed. The method removes the size limitations on the LQNS solver.
- A Newton-Raphson iteration technique was developed to improve the convergence of the iteration step of the method.
- The solution strategy of the method was evaluated by examples and a case study of a real system. The accuracy of the method is best using the Schweitzer approximate MVA.

1.4 Thesis Organization

The thesis first provides some background material in Chapter 2 on the SRVN model used to model systems for performance analysis. Queueing theory is reviewed since it is used to actually solve the SRVN model. Some numerical methods used in the convergence of the proposed replication algorithm are presented. The LQNS tool which is modified to apply the replication method is described in Chapter 3. Chapter 4 presents the notation for representing systems with replication, and Chapter 5 describes the new method proposed to handle the solving of large systems with replicated tasks. Chapter 5 also gives some test

results for the method on modest-sized systems. The new method is applied in Chapter 6 to a case study of the large industrial system mentioned above. Finally, some conclusions are drawn in Chapter 7.

Chapter 2.0 Background

2.1 SRVN Model

The Stochastic Rendezvous Network (SRVN), proposed by Woodside et al. [Wood 95a] [Wood 88], is the model used in the performance analysis method discussed in this thesis. The SRVN performance model is used mainly to model a system with software queueing and rendezvous, although hardware elements may also be included in the model. The model is well-suited for systems with parallel tasks running on a multiprocessor or on a network, such as a client-server system. The SRVN modeling strategy is described with its terminology and notation which are used by the Layered Queueing Network Solver (LQNS) tool discussed in Chapter 3. Modeling a software system as an SRVN model is also discussed.

2.1.1 General Description

SRVN is used to model a software system consisting of several concurrent tasks communicating with each other via the rendezvous mechanism. A client task initiates the rendezvous with a server task and is blocked until a reply from the server is received. The execution time during which the server services the request is called the first phase service time. Any operation taken by the server after sending the reply is executed in the second or subsequent phases. These phases are executed autonomously by the server task and run concurrently with the client task. In the first phase, the server may rendezvous with other tasks to service the client request. These are referred to as “included service times”. Figure 1 diagrams the interactions. Asynchronous interactions may also be modeled with the client task continuing its execution after sending the request.

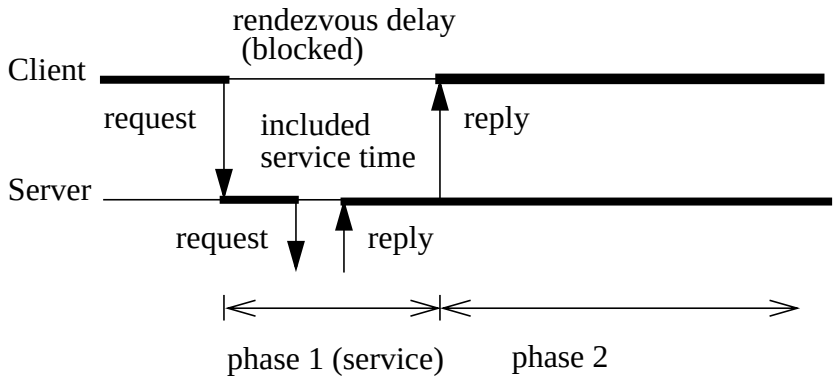


Figure 1: SRVN Phases

When one server provides more than one service, each of these services is modeled as an “entry” with its own phase service times. In fact, a task is actually a collection of entries with one message queue. Between tasks, an entry of one task visits or issues requests to the entry of another a specified number of times. A task always has at least one entry. Thus, an SRVN model may be described as a directed graph with nodes that are service entries and arcs that represent visits from one entry to another [Franks 95].

There may be several layers of tasks (represented by parallelograms) interacting with tasks from various other layers. (See Figure 3 on page 11). However, no cycles are permitted in the graph. The top layer of tasks consists of pure client tasks referred to as reference tasks. Reference tasks do not receive requests but only initiate requests. As a result, reference tasks do not have phase one service times but only phase two service times. Contrary to reference tasks, pure server tasks do not initiate requests but only receive requests. The tasks of the middle layers may act as servers and clients. In Figure 3, WS1 and WS2 are reference tasks while MF is a pure server. LS1 and LS2 are servers when interacting with WS1 and WS2 but are clients when interacting with MF.

When one task is visited by more than one client task, there is contention and a queueing delay. Each task can only visit one task at a time. That is, tasks do not have internal concurrency and may only execute one entry at a time. There is only one queue for all the entries in a task. However, software tasks may be multi-threaded. Multi-threaded tasks are modeled as tasks with multiple copies. Pure client tasks are modeled as infinite servers with no queueing delay, although they may queue for their processors.

2.1.2 Creating an SRVN Model

To model a software system using SRVN, each software entity is represented as a task in the SRVN model. Each task is associated with the processor it runs on. The processor itself is also modeled as a task, although it need not be explicitly represented. Several tasks may share a processor. In the client-server example, Figure 2, there are two LANs each consisting of two workstations and a local server. The workstations make requests to the local server which forwards some of the requests to the mainframe. Workstation 4 may also make direct requests to the mainframe. There is one task running on each of the workstations, local servers, and mainframe. Hence, there is a one-to-one mapping from the entities to the SRVN tasks shown in Figure 3. Each task runs on its own processor which is not shown in Figure 3.

Once the components of the systems are modeled, the entries of the tasks must be determined followed by the parameters such as the phase service times and the visits. In the client-server example, the local servers provide screen update information, forward database requests, and provide report generation information for the workstations. Each of the functions is mapped to an entry:

- SCR (screen update)

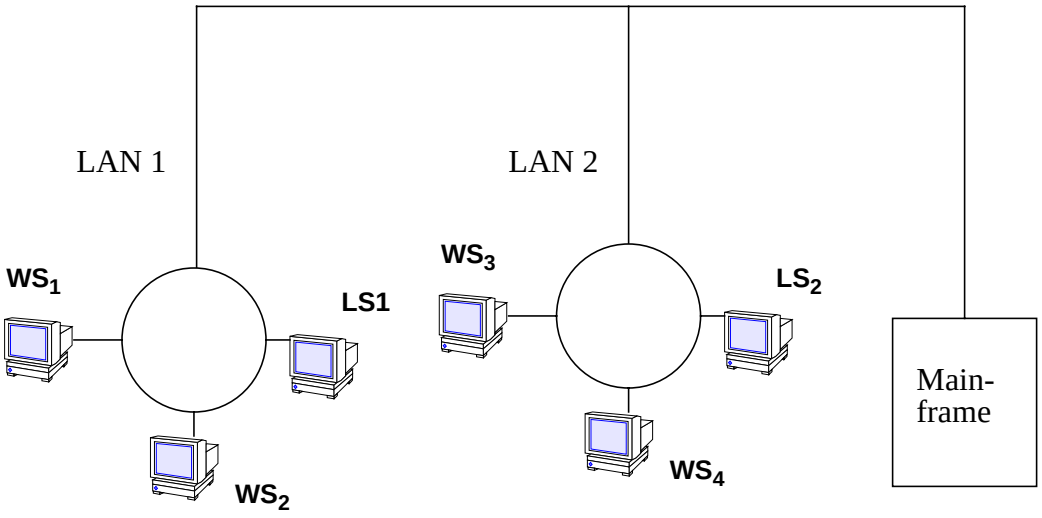
- DBA (database requests)
- RPT (report generation)

Similarly, the mainframe provides database access and report information services which are modeled as entries DB and REP. The mainframe also provides service to autonomous requests from workstation 4. This is modeled as entry OTH. When a request is received by a server, it is common for the different functions to be part of a general case statement. Each case is mapped to an entry.

Each entry has its own phase service times. The phase one service time of the local server and mainframe entries is the execution time of the code invoked in servicing the client request before a reply is sent back. The second or subsequent phase service time is the execution time of the code following the reply. For example, the mainframe can return the database information to the local server request and then execute some clean-up code. A reference task, such as the workstation, has one implicit entry and, being a pure client, only has phase two service time. The phase two service time is the execution time of the code before the rendezvous to a server is invoked.

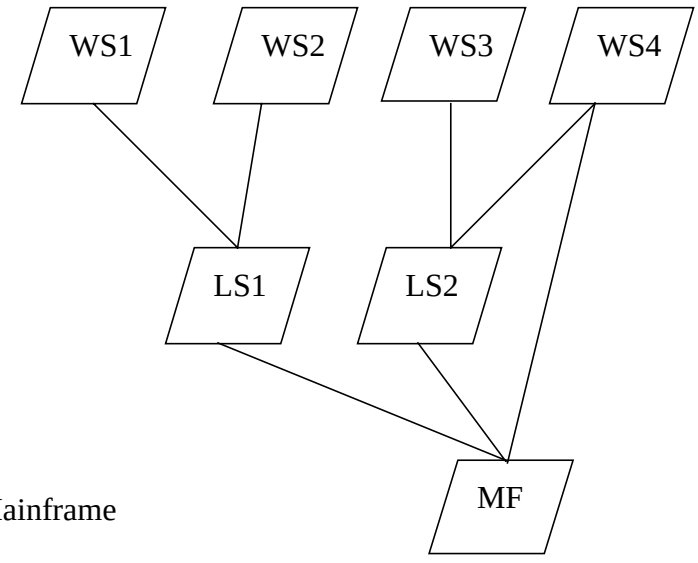
The visit ratio between entries are the number of rendezvous calls to the server. The server is assumed to be ready to receive requests and cycles infinitely. The pure client tasks continually makes requests with a think time in between requests. Tasks may have outside inputs in which case it is an open or mixed model.

The parameters may be obtained through instrumentation of the code and measurements. They may also be estimated from experience and expert judgement. Since the execution times and visits may be random, the average execution times and frequency of visits are used.



WS - Workstation
LS - Local Server

Figure 2: Client-Server System



MF - Mainframe

Figure 3: SRVN Model of Client-Server System

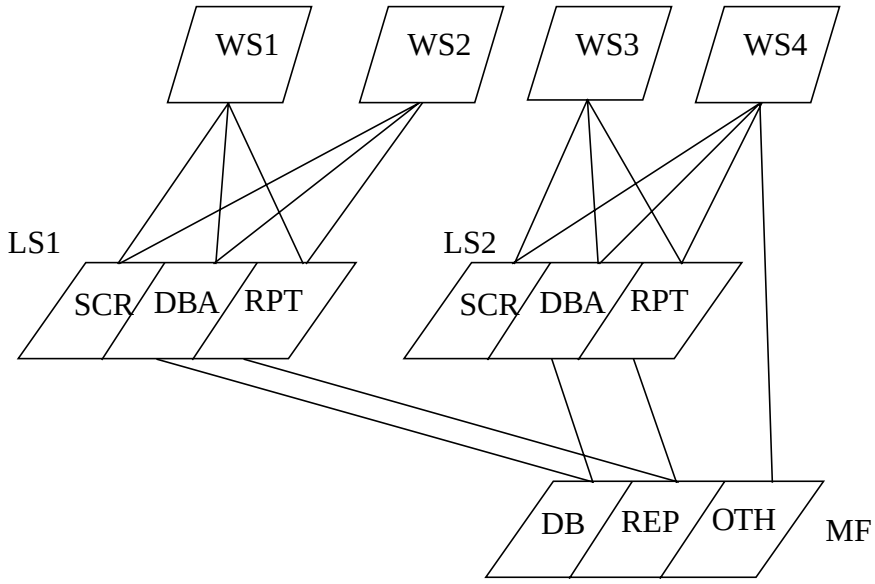


Figure 4: SRVN Model with Entries

2.2 The Method of Layers (MOL)

The method of layers [Rolia 95] [Rolia 92] is the basis of the LQNS "Merged-Layerize" strategy described in Chapter 3. MOL is used to predict the performance measures of systems by viewing a performance model as a sequence of layered queueing models (LQMs). Each LQM is divided into two complementary models, one for software and one for devices. The results of solving the two models are combined along with the results of all the LQMs to obtain the performance measures of the system. The models are solved using a modified version of Linearizer. (Linearizer is discussed in Sect. 2.3.) The LQNS "Merged-Layerize" strategy differs here in that devices are not solved in a separate model per se. However, it does follow MOL in the basic layering strategy and in representing clients as queueing network job classes.

2.3 Queueing Theory

The Layered Queueing Network Solver (LQNS) solves an SRVN model by representing each layer of the model (referred to as a layer submodel) as a network of queues and evaluating the submodels analytically. Since queueing networks are well known, this section reviews them only briefly for completeness. A queueing network consists of a collection of service centers with customers, referred to as tokens, visiting the service centers. There are two types of service centers. The queueing station is a service center at which customers may need to wait in a queue to receive service. A delay server (or infinite server) is a service center where customers receive immediate service and there is no queueing time. In most models of computer systems, computer terminals or workstations are represented as delay servers.

In an LQNS layer submodel, the client tasks are represented as delay servers and the server tasks as queueing stations. There are several parameters associated with each service center. The most obvious is the service time of each service center. That is, the time the center takes to actually execute a request. The queueing discipline is another parameter associated with the queueing stations. In the discussions following, the queueing stations are assumed to be first-come first-served (FCFS). Another parameter is the visit ratio which represents the routing probability of tokens to the service centers.

There are three types of queueing networks, open, closed and mixed. In an open queueing network, customers enter and leave the network. Therefore, the population of the network may be infinite. An example of an open network is the model for a transaction processing system. In a closed queueing network, the population is fixed. That is, customers circulate in the network but never leave it. An example is a batch processing system. A mixed queueing network consists of customers that enter and leave the network

as well as those that circulate within the network. Figure 5 shows an example of a closed queueing network with one delay server and two queueing stations.

Basically, the inputs, or known parameters, of a queueing network are the service times and visit ratios. The results that are of interest are the throughput, utilization, and the waiting times (or queueing time) at each service center.

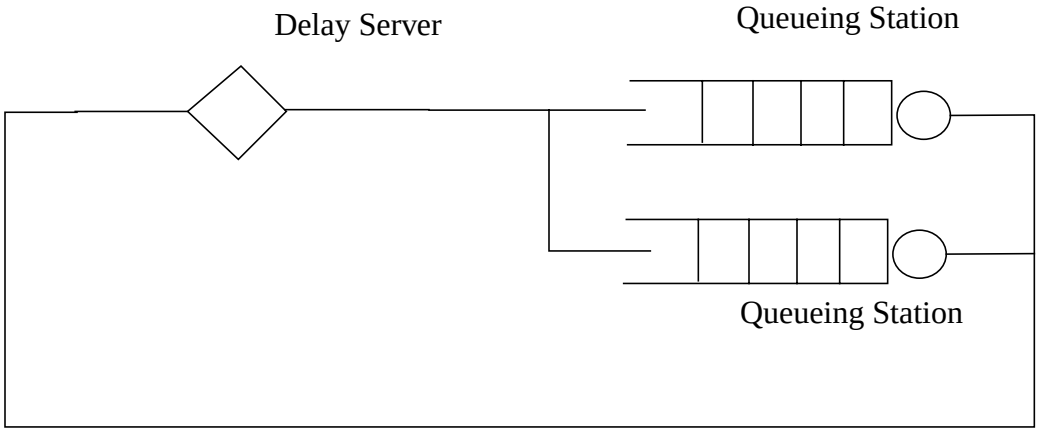


Figure 5: Closed Queueing Network

2.3.1 Chain vs. Class

The customers or tokens of a queueing network, which have similar statistical characteristics, may be grouped into chains or classes. The terms chain and class are often used interchangeably. However, in this thesis, the definition specified by Chandy and Sauer [Chandy 78] is taken. A chain specifies a routing in a queueing network. Tokens that visit the same service centers with the same frequency are grouped into the same chain. Class is

associated with a local service center. That is, the service times and distributions may be different for different classes at a service center. In the SRVN model, the concept of class is associated with the entries of a task. The tokens of the same chain may actually belong to a different class at a service center. The terms chain and class become equivalent when the tokens of the same chain belong to the same classes at all the service centers visited. Chandy and Sauer refer to this as a "global" class [Chandy 78].

2.3.2 Mean Value Analysis (MVA)

MVA is an algorithm employed to solve closed product-form queueing networks in steady state to give average performance results for delay and throughput. (Refer to [Jain 91] and [Lazow 84] for a definition of product-form.) The algorithm presented here is for a model with multiple chains or classes. The terms class and chain are equivalent in this discussion of the MVA. That is, it assumes that classes are "global" and there is one class per chain.

The MVA algorithm uses three key equations which are basically derived from Little's Law and the Arrival Instant Theorem. Little's Law states the following [Jain 91]:

Mean number of jobs in a system = mean arrival rate x mean time spent in the system

The arrival instant theorem states that, in a product-form queueing network, the queue length ($A_{c,k}$) seen by a customer of chain *c* on arrival at a center *k* is equal to the mean queue length (Q_k) there with the arriving customer removed from the network [Lazow 84]. That is,

$$A_{c,k}(\bar{N}) = Q_k(\overline{N-1_c}) \qquad \text{Eq (2.1)}$$

where $\bar{N} = (N_1, \ldots, N_c)$ is the workload intensity vector consisting of all chain population sizes (N_c), and $\overline{N-1_c}$ is the population $\bar{N}$ with one customer of chain c removed.

The three key equations used in the MVA algorithm follow.

- The service center residence time ($R_{c,k}$) for each chain (uses equation (2.1)),

$$R_{c,k}(\bar{N}) = D_{c,k}(1 + A_{c,k}(\bar{N})) \qquad \text{(queueing stations)} \qquad \text{Eq (2.2)}$$

$$R_{c,k}(\bar{N}) = D_{c,k} \qquad \text{(delay server)} \qquad \text{Eq (2.3)}$$

  where $D_{c,k}$ is the total demand of chain c at center k (= service time x visits). Note that the residence time equals the visits of chain c at center k ($Y_{c,k}$) times the waiting time ($W_{c,k}$). That is, $R_{c,k} = Y_{c,k} W_{c,k}$.

- Applying Little's law to the queueing network as a whole, the throughput (X_c) for each chain is,

$$X_c(\bar{N}) = \frac{N_c}{Z_c + \sum_{k=1}^{K} R_{c,k}(\bar{N})} \qquad \text{Eq (2.4)}$$

  where Z_c is the think time for chain c and N_c is the number of tokens for chain c.

- Applying Little's law to each service center, the mean queue length ($Q_{c,k}$) for chain c at center k as well as the total mean queue length (Q_k) at center k are,

$$Q_{c,k}(\bar{N}) = X_c(\bar{N}) R_{c,k}(\bar{N})$$
$$Q_k(\bar{N}) = \sum_{c=1}^{C} Q_{c,k}(\bar{N}) \qquad \text{Eq (2.5)}$$

Basically, the algorithm consists of finding an arrival-instant queue length ($A_{c,k}$) and using this queue length to find the residence time (equation 2.2). The residence time is then used to derive the throughput (equation 2.4). Finally, from this throughput a new queue length may be found (equation 2.5).

There are two approaches in evaluating the three equations, exact and approximate, which differ in the way the arrival instant queue lengths ($A_{c,k}$) are computed. In the exact method, applicable only to product-form queueing network models, equation (2.1) is evaluated exactly. The trivial solution of the network for population $\overline{0}$ ($Q_k(\overline{0}) = 0$ for all centers *k*) is used and applied to equations (2.2) to (2.5). From equation (2.5) the queue length for the next large population with one more customer in one chain is obtained. The computation proceeds recursively over increasing populations until the target population $\overline{N}$ is reached. The exact MVA algorithm is given in Figure 6. The exact MVA requires an evaluation at every possible population less than the target population $\overline{N}$. The computational complexity increases with the number of chains and centers.

The approximate method often becomes the more practical solution since it does not require the evaluation of equations (2.2) to (2.5) for all populations from zero to the full population. Instead, the arrival instant queue lengths ($A_{c,k}$) are estimated based on the time averaged queue lengths at the service centers with the full customer population ($\overline{N}$), and iteration is used to improve the estimate. The algorithm is given in Figure 7 [Lazow 84]. Many different functions may be used to estimate the arrival instant queue length ($A_{c,k}$). The Bard-Schweitzer approximation assumes that $A_{c,k}(\overline{N})$ is proportional to $Q_{c,k}(\overline{N})$. The function used is:

$$A_{c,k}(\overline{N}) = Q_k(\overline{N-1_c}) \approx \left(\frac{N_c - 1}{N_c} Q_{c,k}(\overline{N})\right) + \sum_{j=1;(j \neq c)}^{C} Q_{j,k}(\overline{N}) \qquad \text{Eq (2.6)}$$

Inputs:

C = Number of chains

K = Number of service centers

N_c = Number of customers in chain c

Z_c = Think time of chain c

$D_{c,k}$ = Service demand of chain c at device k

Outputs:

X_c = Chain c system throughput

Q_k = Average number of customers at device k

$Q_{c,k}$ = Average number of customers of chain c at device k

$R_{c,k}$ = Residence time of chain c at device k

Initialization:

FOR k = 1 to K DO $Q_k(\bar{0}) = 0$;

Iteration:

FOR n = 1 to $\sum_{c=1}^{C} N_c$ DO

FOR each feasible population $\bar{n} \equiv (n_1, \ldots, n_c)$ with n total customers DO

BEGIN

FOR c = 1 to C DO

FOR k = 1 to K DO

$R_{c,k}(\bar{n}) = D_{c,k}$; (for delay servers)

$R_{c,k}(\bar{n}) = D_{c,k}(1+Q_k(\overline{n - 1_c}))$;(for queueing stations)

FOR c = 1 to C DO

$$X_c(\bar{n}) = \frac{n_c}{Z_c + \sum_{k=1}^{K} R_{c,k}(\bar{n})} \quad ;$$

FOR k = 1 to K DO

FOR c = 1 to C DO

$Q_{c,k}(\bar{n}) = X_c(\bar{n})\, R_{c,k}(\bar{n})$;

$$Q_k(\bar{n}) = \sum_{c=1}^{C} Q_{c,k}(\bar{n})$$

END;

Figure 6: Exact MVA Algorithm

Inputs and Outputs: Same as Figure 6 for exact MVA

Initialization:

delta = 0;

FOR c = 1 to C DO

FOR k = 1 to K DO $Q_{c,k}(\bar{N}) = \frac{N_c}{K}$; $Q^{old}_{c,k}(\bar{N}) = 0$

Iteration:

WHILE delta > ε DO %Convergence not reached

BEGIN

FOR c = 1 to C DO

FOR k = 1 to K DO

% $A_{c,k}(\bar{N}) = Q_k(\overline{N - 1_c})$

% $= h_c(Q_{1,k}(\bar{N}), \ldots, Q_{C,k}(\bar{N}))$

$A_{c,k}(\bar{N}) = \left(\frac{N_c - 1}{N_c} Q_{c,k}(\bar{N})\right) + \sum_{j=1;(j \neq c)}^{C} Q_{j,k}(\bar{N})$ %Bard-Schweitzer

FOR c = 1 to C DO

FOR k = 1 to K DO

$R_{c,k}(\bar{N}) = D_{c,k}$ %for delay servers

$R_{c,k}(\bar{N}) = D_{c,k}(1+A_{c,k}(\bar{N}))$ %for queueing stations

FOR c = 1 to C DO

$$X_c(\bar{N}) = \frac{N_c}{Z_c + \sum_{k=1} R_{c,k}(\bar{N})}$$

FOR k = 1 to K DO

FOR c = 1 to C DO

$Q_{c,k}(\bar{N}) = X_c(\bar{N}) R_{c,k}(\bar{N})$

delta = max{delta, $Q_{c,k}(\bar{N}) - Q^{old}_{c,k}(\bar{N})$ }

$Q^{old}_{c,k}(\bar{N}) = Q_{c,k}(\bar{N})$

END;

Figure 7: Approximate MVA Algorithm (Bard-Schweitzer)

Another approximate MVA method is Linearizer [Chandy 82]. This method improves on the Bard-Schweitzer approximation of the queue length at each device. The Linearizer considers the fact that there is a fractional change in the queue length with the removal of a token. That is, the fraction of tokens of a chain that are queued at a device changes with the change in the number of tokens in the system. However, the method does assume that the fractional change is linear. Linearizer uses the basic Bard-Schweitzer algorithm to obtain queue length estimates at population $\overline{N}$ and $\overline{N\text{-}1}_c$. From the two sets of queue lengths, the change in fraction of tokens at each device for each chain is calculated. The change in fraction is then applied to the next iteration for new queue length estimates.

All solution schemes, including the "exact MVA", of the LQNS include approximations to the waiting time calculation for service time effects and second phase.

2.3.3 Method of Surrogate Delays

The method of surrogate delays is a key concept in the solving of replicated models and is also employed in the solving of the SRVN models as a whole. The method, proposed by Jacobson and Lazowska [Jacob 82], is an approximate solution technique for queueing network models which have resources that are accessed simultaneously or have an overlap in possession. Basically, the queueing network is split into multiple models. In each model, the queueing delay encountered at one of the resources is represented as a delay server. The queueing delay is obtained from the model in which the resource is explicitly modeled with the other resources represented by delay servers. The method iterates the queueing delay estimates between the models until convergence. Figure 8 gives an example used by Jacobson and Lazowska.

In the example, the disks in an I/O system compete for use of a channel to complete their data transfer to and from memory. The queueing network is broken into two models.

In the first model,Figure 9(a), the channel is represented by a delay server while the disks are represented by queueing stations. In the second model, Figure 9(b), the disks are represented by a delay server and the channel by a queueing station (actually a flow-equivalent service center).The two models are solved with the queueing delays in one model used in the other. The delay value for the delay server in one model is set equal to the delay (queueing and service) computed in the other model. Initially, the delays are set to the service time only. Then, they are computed iteratively until convergence.

If the MVA algorithm is inspected, it becomes apparent that basically the residence times of the stations are being estimated using a fixed-point iteration technique, the Gauss-Seidel iteration. Each submodel expresses equations of the form $x_i = F_i(x_i)$. In fact, the approximate MVA is itself a Gauss-Seidel iteration.

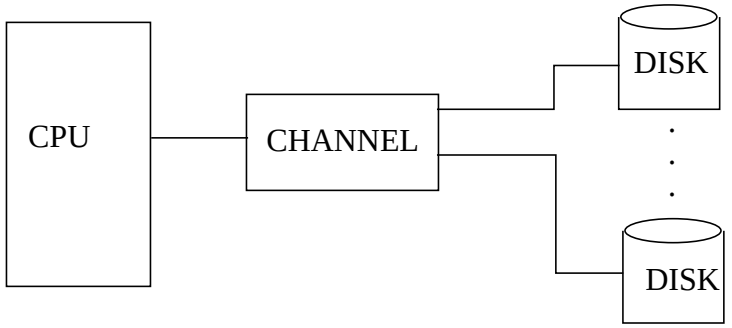


Figure 8: I/O Subsystem

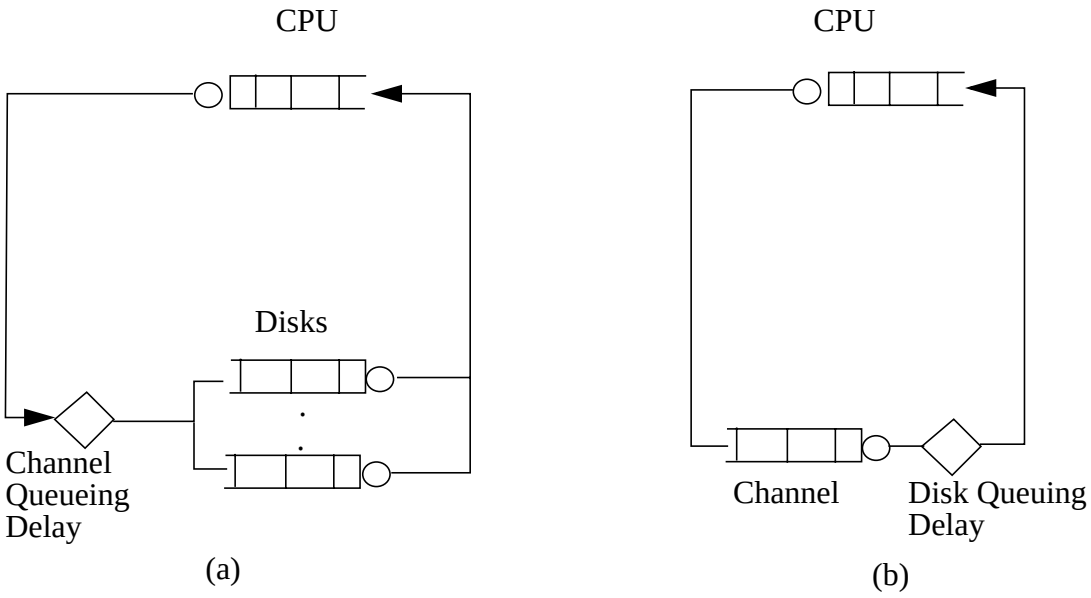


Figure 9: Models for Solving the I/O System

2.4 Fixed-Point Iterative Methods

Iterative methods may be used to solve a system of non-linear equations, with the general form given below, where elimination is usually not feasible.

$$\begin{aligned} f_1(x_1, x_2, \ldots x_k) &= 0 \\ f_2(x_1, x_2, \ldots x_k) &= 0 \\ &\ldots \\ f_k(x_1, x_2, \ldots x_k) &= 0 \end{aligned} \qquad \text{Eq (2.7)}$$

The Gauss-Seidel iteration and the Newton-Raphson method are two such iterative methods discussed in the next two sections. Both methods are used in solving replicated systems as proposed in this thesis.

2.4.1 Gauss-Seidel Iteration

To solve equation (2.7) using the Gauss-Seidel method, the equation is rearranged into the following form:

$$\begin{aligned} x_1 &= F_1(x_1, x_2, \ldots x_k) \\ x_2 &= F_2(x_1, x_2, \ldots x_k) \\ &\ldots \\ x_k &= F_k(x_1, x_2, \ldots x_k) \end{aligned} \qquad \text{Eq (2.8)}$$

Iterative methods which solve the equation in the form of equation (2.8) are referred to as fixed-point iterative methods.

Successive estimates of the solution may be computed with the following Gauss-Seidel iteration [Davis 86]:

$$x_i^{n+1} = F_i(x_1^{n+1}, x_2^{n+1}, \ldots x_{i-1}^{n+1}, x_i^n, x_{i+1}^n, \ldots x_k^n) \qquad \text{Eq (2.9)}$$

where the subscript *n* denotes the iteration in which the value was computed.

Starting with an initial estimate and continuing, the solution may converge. However, convergence is not guaranteed. Figure 10 illustrates graphically the sequence of values calculated by a Gauss-Seidel iteration for a single equation $x = F(x)$. The fixed-point, S, is at the intersection of *y=x* and *y=F(x)*. Figure 10 (a) shows a case where the iteration converges to the solution S, whereas, (b) shows a case where the iteration diverges. Convergence occurs if $|F'(S)| < 1$. In the case of a system of equations, S is a vector (S_1, S_2, . . . S_k).

The iteration used in the method of surrogate delays discussed in Section 2.3.3 is actually a Gauss-Seidel iteration. Each submodel computes a delay which is one of the components of *x*, and the other submodel depends on this delay.

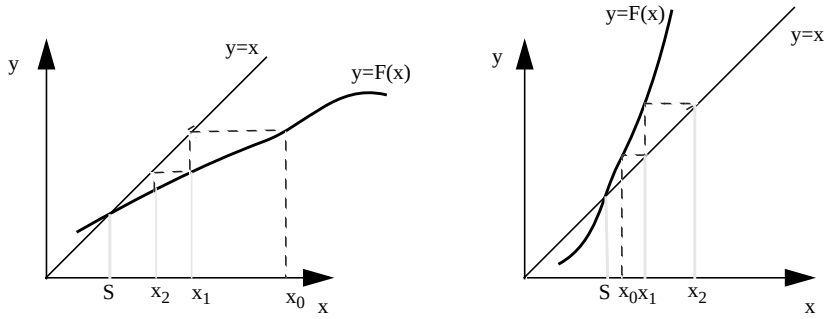


Figure 10: Convergence of the Gauss-Seidel Iteration

2.4.2 Newton-Raphson Method

The Newton-Raphson method may be used to solve a system of non-linear equations with a more rapid convergence than the Gauss-Seidel method. In this method, the next iteration value is obtained by taking the derivative of the function at the estimation point. In this section, the Newton-Raphson method is derived solving the system of equations given in the form of equation (2.8). That is, a fixed-point Newton-Raphson method is derived as opposed to other derivations ([Davis 86] [Pearson 86]) where the method solves the system of equations given in the form of equation (2.7). Figure 11 graphically illustrates the method for a single equation $x = F(x)$. The iteration formula is:

$$x^{n+1} = F(x^n) + F'(x^n)(x^{n+1} - x^n) \qquad \text{Eq (2.10)}$$

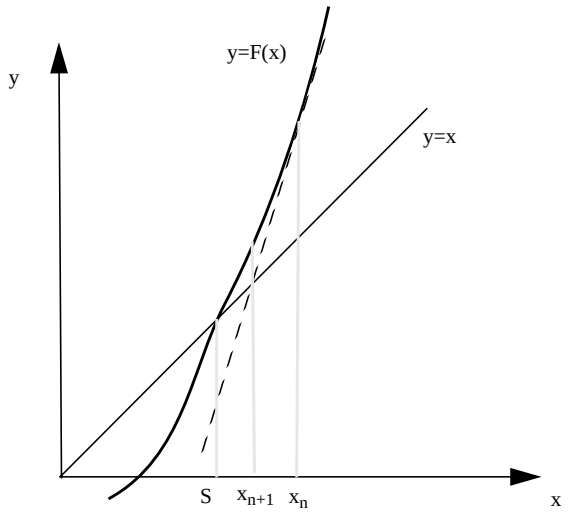


Figure 11: Newton-Raphson Method

Taking the same examples (Figure 10) for the Gauss-Seidel iteration and applying the Newton-Raphson method, it can be seen that the convergence is quicker. In the case of Figure 10 (b) the Newton-Raphson version of the iteration actually converges. The convergence for the Newton-Raphson method is, however, not guaranteed. Refer to the references by Davis and Pearson ([Davis 86] [Pearson 86]) for a detailed explanation of the quadratic convergence of the Newton-Raphson Method.

Extending the Newton-Raphson method to a system of equations gives the following iteration formula:

$$x_i^{n+1} = F_i(x_1^{n+1}, x_2^{n+1}, \ldots x_k^n) + \sum_{l=1}^{k} \left(\left(\frac{\partial}{\partial x_l} F_i(x_1^{n+1}, x_2^{n+1}, \ldots x_k^n) \right) (x_l^{(n+1)} - x_l^{(n)}) \right) \quad \text{Eq (2.11)}$$

where the subscript *n* denotes the iteration in which the value was computed as in equation (2.9).

It is apparent from equation (2.11) that there is a great deal of work per iteration. At each iteration all of the functions must be evaluated as well as the derivatives of each function with respect to each variable. Then, a system of equations must be solved.

The method may be simplified to reduce the work per iteration. In the simplified Newton-Raphson method, each equation in the system is considered to be an equation for just one of the unknowns. The resulting simplified iteration is:

$$x_i^{n+1} = F_i(x_1^{n+1}, x_2^{n+1}, \ldots x_k^n) + \left(\frac{\partial}{\partial x_i} F_i(x_1^{n+1}, x_2^{n+1}, \ldots x_k^n) \right) (x_i^{(n+1)} - x_i^{(n)}) \quad \text{Eq (2.12)}$$

With this simplified method, fewer functions need to be evaluated and a system of equations does not need to be solved. Again refer to the references by Davis and Pearson for a detailed discussion of the simplified method.

In summary, the Newton-Raphson method can be used if the derivative of the function is easily calculated. The method may be used to increase the speed of convergence and, in some cases, converges where the Gauss-Seidel fails.

Chapter 3.0 Layered Queueing Network Solver (LQNS)

3.1 General Overview of the LQNS Tool

The Layered Queueing Network Solver (LQNS) is the computational tool that was modified in this research for use in the performance study of large client-server systems. The LQNS is a software package that can solve the "Stochastic Rendezvous Network" (SRVN) which models client-server systems as shown in Figure 3 in Chapter 2. This chapter gives a general understanding of the LQNS tool which is needed to incorporate the replication modifications discussed later in Chapter 5. Only the aspects of the tool which are relevant to the modifications are discussed. For example, the LQNS package may be used to solve SRVN models using several different layering methods. However, the discussion in this chapter pertains only to the 'Merged-Layerize' layering strategy which is used by the replication modifications.

The LQNS software is the solver engine behind the user interfaces Timebench and lqndef [Hubbard 95]. Timebench, a graphical user interface, and lqndef are used to define the parameters of the SRVN model, such as service time and visits. Both interfaces generate a description file, referred to as the input file, from the user supplied description of the SRVN model. The input file contains information about the processors, tasks, entries and the visits between entries. (See [Petriu 95] for a full description of the input file.) The user interfaces invoke the LQNS solver which reads the data in the input file and generates its own internal description of the SRVN model. That is, it stores the SRVN model information in its own database. The LQNS then solves the SRVN model layer by layer by

mapping each layer to a queueing network. The results of solving the SRVN model, such as the throughput and wait times of tasks, are written to an output file. The output file data is then processed by the user interface, either Timebench or lqndef, and displayed accordingly.

It should be noted that the LQNS solver may be invoked independently of the user interfaces. The input file may be generated manually in the specified format [Petriu 95]. The LQNS solver is then invoked via a command line and the output file generated may be inspected for the results. The overall mechanism is shown in Figure 12.

3.2 Solving the SRVN Model

The 'Merged-Layerize' strategy used in LQNS solves the SRVN model by utilizing the method of layers [Rolia 92] and the method of surrogate delays [Jacob 82]. The SRVN model is divided into submodels with each submodel representing a layer of the model. Subsequently, each submodel is mapped to a product-form queueing network and is solved by the MVA algorithm. The station delays are iterated between the submodels until convergence. The following sections provide the details of the submodels and the iteration between the submodels.

3.2.1 Mapping A Layer To a Queueing Network

Figure 13 shows the submodels for the example SRVN model shown in Figure 3 in Chapter 2. The submodels correspond to the layers of the SRVN model. The first submodel consists of the client and server tasks of the first layer of the SRVN model. The second submodel, however, not only consists of the tasks in the second layer but also client task WS4 of the first layer. This task is included in the second submodel since it visits a task

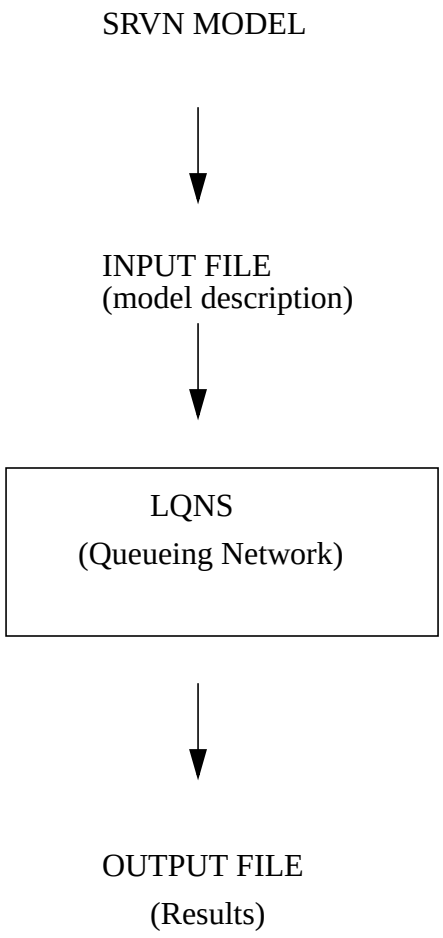


Figure 12: LQNS Context

(MF) which is in the second layer. The submodels also contain the processors (P1 to P7) used by each task. The processor allocations are defined in the input file but usually are not represented in the SRVN model as seen in Figure 3. In the example, each task has its own processor, but it is possible for tasks to share processors. The processors are represented and treated as tasks in the LQNS solver

The LQNS reads in the SRVN model description from the input file and stores the information internally having broken the model up into submodels or layers as described

above. The submodels are stored in terms of SRVN parameters. That is, the service time and visits are stored in reference to client and server task entries. However, these parameters and submodels must be translated into queueing network terms to solve the submodels via MVA. Each submodel is mapped to a product-form queueing network. The clients in each submodel are represented as delay servers and the server tasks are represented as queueing stations. As an example, Figure 14 shows the queueing network associated with submodel 2.

A chain is generated in the queueing network for each client task of the SRVN submodel. The number of tokens in each chain corresponds to the number of copies of a client task. A chain traverses all the servers visited by a client. Thus, only one chain may traverse each client (or delay server) but a server (queueing station) may be traversed by several chains. (This method of constructing the chains is modified for replicated tasks described later in Chapter 5.) The visit ratio of the tokens of each chain to the queueing stations corresponds to the visits made by the client to its servers in the SRVN submodel. The service time of each entry of the server task of the submodel corresponds to the service time of each class of the queueing station. For the queueing network of submodel 2 shown in Figure 14, where each task has one entry and one phase, the chains are generated as follows (N_k = number of tokens in chain k; V_{km} = visits to station m by chain k tokens):

Chain 1: $N_1 = 1$

$V_{1,LS1}$; $V_{1,P5}$; $V_{1,MF}$

Chain 2: $N_2 = 1$

$V_{2,LS2}$; $V_{2,P6}$; $V_{2,MF}$

Chain 3: $N_3 = 1$

$V_{3,WS4}$; $V_{3,MF}$

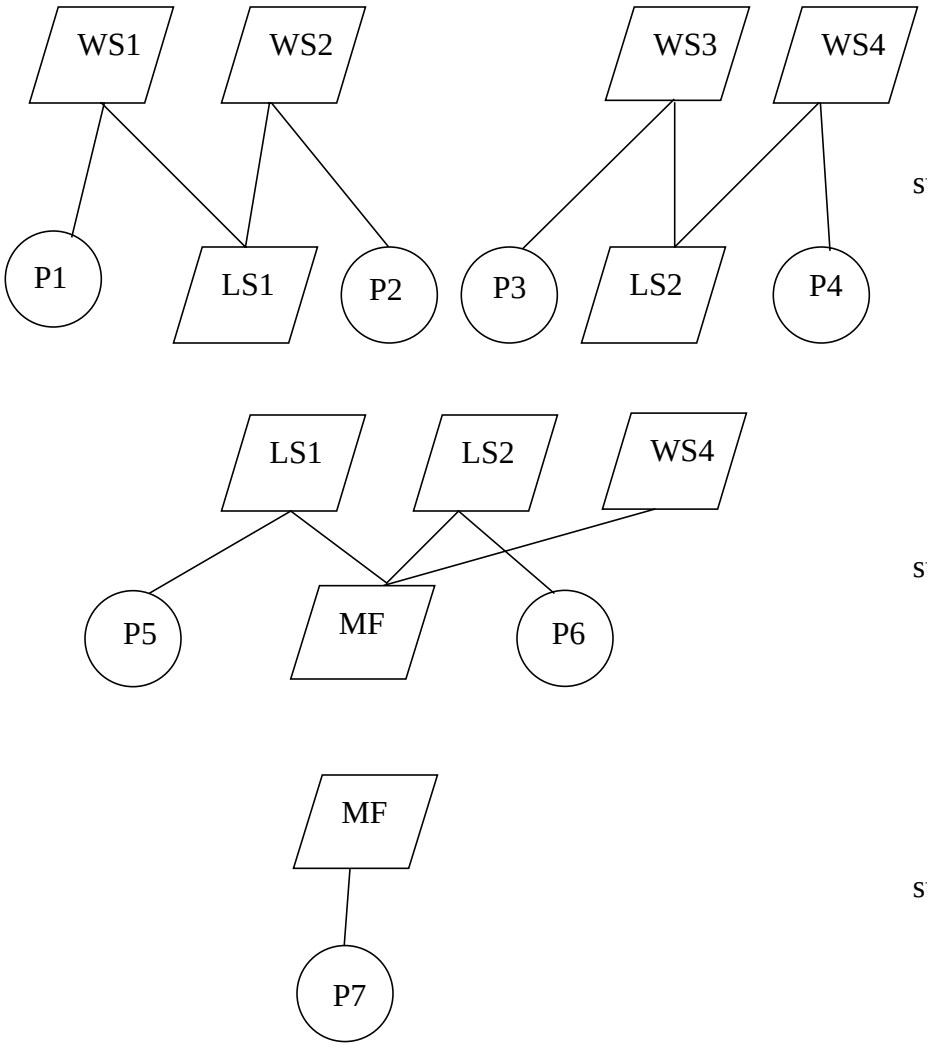


Figure 13: Submodels for Figure 3

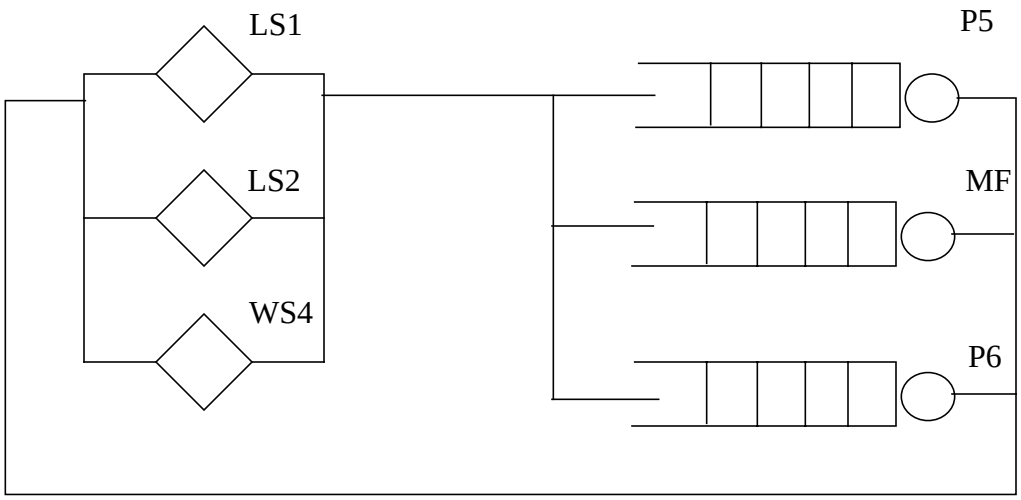


Figure 14: Queueing Network for Submodel 2 of Figure 13

3.2.2 Solving Between Layers

Since the SRVN model consists of several interacting layers, the parameters of each submodel, representing a layer, must somehow relate to the layers or submodels it interacts with. The method of surrogate delays [Jacob 82] is employed. The delay that a task encounters in one layer (or submodel) is represented as a surrogate delay in another layer (or submodel) that also contains that task. For example in Figure 13, the residence time calculated for client task WS4 in submodel 1 is used as the service time of client task WS4 in solving submodel 2. Likewise, once submodel 2 is solved, its residence time for task WS4 is used as the service time when solving submodel 1. The service time of a task as a client is calculated from the residence time or service time of the task at all layers except its own layer. For server tasks, the service time of the task is obtained by adding all the delays of that task in all layers. For example, the service time of server task LS1 in

submodel 1 is added to the residence time of the same task calculated in submodel 2 to obtain the task LS1 service time to be used in solving submodel 1. The idle time (or think time) of a client task is calculated from the utilization of that task when it acts as a server.

The SRVN model is solved by initially solving the submodels from the top layer down. The service time values are used between submodels. A Gauss-Seidel iteration is employed, and the submodels are solved until all the service time values reach convergence. That is, the differences between all the values are less than a defined value usually in the range of 10^{-6}.

3.3 Implementation Structure

The LQNS tool has been designed using an object-oriented approach and is implemented in C++. Two models are employed within the LQNS. One model is the SRVN model and the other is the MVA queueing network model. The components of each model are mapped to object classes. The class hierarchy diagram for the two models is shown in Figure 15 [Franks 94]. The notation of Rumbaugh is used [Rumb 91].

3.3.1 Classes

A. Classes for Model Building

These classes are used to store and manipulate the parameters of the SRVN model. The major components of the SRVN model are tasks and processors. Classes *Task* and *Processor* define these two components respectively. Since processors are essentially viewed as tasks in the SRVN model, processor objects have some attributes and operations that are common with task objects. The common features are grouped in class *Entity*, the abstract superclass of subclasses *Task* and *Processor*. (The notation for this relationship is the triangle.) Since tasks must run on a processor (task entities are actually clients to the

processor entities), there is a many-to-one association between the two classes. The solid ball at class *Task* indicates that many task instances may be associated with one processor instance, thus modeling the fact that many tasks can run on one processor but a given task may only run on a single processor.

The entries of the SRVN model are modeled as a separate class. An aggregation relationship exists between the class *Entry* and class *Entity* since a task (or processor) is essentially a collection of entries in the SRVN model. The diamond notation is used to denote that entries are part of an entity. The arcs between entries are modeled by class *Call*. An entry may visit or call many other entries. A call is made up of two entries, one from a client task and one from a server task.

B. Classes for the MVA Solver

These classes are used for solving the queueing network associated with one SRVN layer submodel. The class *Server* represents the stations of a queueing network. It is an abstract superclass of subclasses that represent the different types of stations such as a delay server and a first-come first-served server. The *Server* class is associated with the SRVN model class *Entity* since the parameters of a queueing network station are mapped from the SRVN entity parameters. Instances of class *Server* are used by the class *MVA* which solves the queueing network. Class *MVA* is also an abstract superclass. Its subclasses are the main types of MVA solvers, exact, Linearizer, and approximate (Schweitzer).

The class *Layerize* divides the SRVN model into layers or submodels that are then solved by class *MVA*. This class builds the submodel from the SRVN model classes and also translates the results of the submodels back to the SRVN model components. The overall solution of the whole SRVN model is handled by class *Layerize*. It is the superclass

of the different layering solution strategies used in LQNS, namely, “Merged-Layerize” and Rolia’s method of layers [Rolia 95].

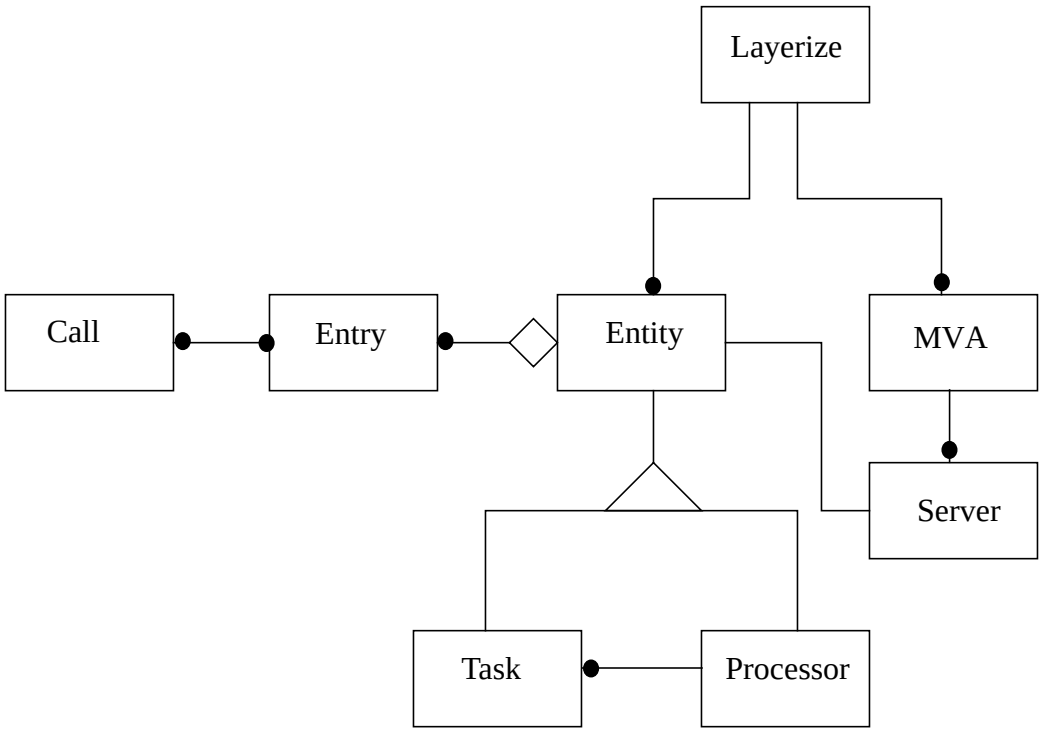


Figure 15: Class Hierarchy Diagram

Chapter 4.0 Representation of Replicated Systems

4.1 Introduction

In some large complex systems, it may be possible to find a large number of identical or nearly identical components. For example, in a transaction processing system, the client requesters are typically from similar devices and from similar applications. From a modeling point of view, they may be treated equally if the system they are part of is symmetric. In the SRVN model, the components are represented as tasks. Tasks that have the same characteristics and parameters, such as the number of phases, the phase service times, the entries, the number of visits to servers, may be defined as replicated tasks. Figure 16 gives an example of a system with replicated tasks. In this system, each task has one entry. The service times for tasks A1, A2, A3, and A4 are equal as are their think times. As these are pure clients, they have only phase 2 service times. Similarly, the service times for tasks B1 and B2 are equal. These two tasks may have more than one phase with the corresponding phase service times. Since the tasks are replicas, their phase service times are the same. The number of visits between tasks A1 and B1 are equal to the number of visits between any of the A tasks and task B1. Similarly, the number of visits between the A tasks and task B2 are equal. Since each task has the same parameters, it can be shown that their performance characteristics are also equal.

This chapter introduces the notation and model for systems with replicated tasks as proposed by Woodside [Wood 95b]. The model and notation provides a compact simplified way of representing large systems with replicated components by representing the replicas

by one component. The connections between replicated components are also represented once. The model will be referred to as the replicated model or simplified model.

4.2 Notation

To define the notation of the replicated model representation, an example is used. Figure 17 shows the replicated model representation of the system in Figure 16. For simplicity, it is assumed that each task has only one entry. The parallelograms represent a set of tasks (A*, B*) instead of one task (A1, B1 etc.) as in Figure 16. The number of replicated tasks are defined by K. In Figure 17, parallelogram A* represents the four identical tasks (A1, A2, A3, A4) by defining $K_A = 4$. Likewise, parallelogram B* represents the two identical tasks B1 and B2 by defining $K_B = 2$. The arcs between the parallelograms represent the request/reply interaction between task set A* and task set B* in the example system of Figure 16. However, the arcs may also connect task sets with single tasks, as is seen in later examples. The visit rate defined on the arc of the full representation of the system corresponds to the visit rate along the single representative arc in the replicated model representation. For example, the visit rate from task A1 to task B1 (Y_{A1B1}) of Figure 16 is equal to the visit rate defined for the arc between task A* and B* (Y_{AB}) of Figure 17.

There are two additional parameters associated with each arc, the fan-in and fan-out, which define the model uniquely. The fan-in at a task set represents the number of members of the requesting task set that make requests to each member of the task set. In the example, the fan-in, shown in Figure 17 and denoted by F_{AB}, equals 4 since four member tasks (A1, A2, A3, A4) of task set A* visit each member (B1, B2) of task set B*. The fan-out, $\overline{F}_{AB}$, defines the number of members of a task set visited by another task or task set. In the

example, $\overline{F}_{AB}$ equals two since each member task of task set A* (e.g. A1) visits two members of task set B* (B1, B2). As can be seen, the fan-out can be derived from the fan-in by the following equation:

$$K_A \times \bar{F}_{AB} = K_B \times F_{AB}$$

In addition, since the fan-out $\overline{F}_{AB}$ must be greater than or equal to one, and the number of replicas K_A must also be at least one, then

$$K_B \times F_{AB} \geq K_A$$

In defining the replicated model, it is desirable to define both the fan-in and fan-out. However, since one may be derived from the other, the model is still uniquely defined if only one is given. The fan-in and fan-out are integers in a perfectly symmetrical system, but they may be fractional in which case they would represent average fan-out and fan-in values.

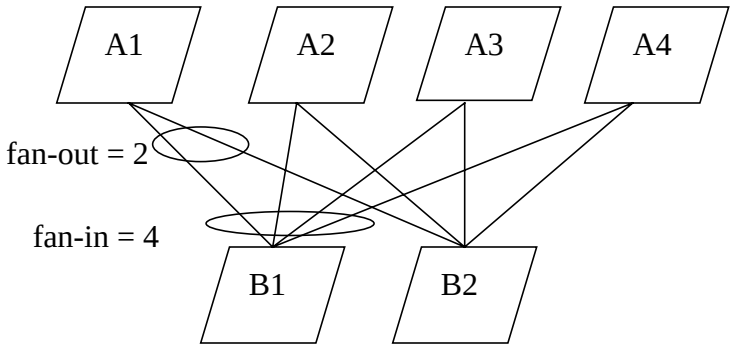


Figure 16: System with Replicated Tasks

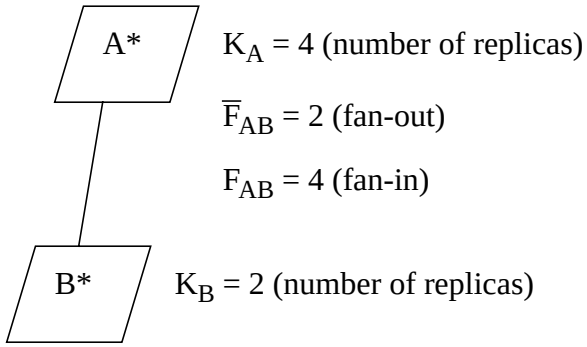


Figure 17: Replicated Representation of Figure 16

4.3 Replicated Systems

The replicated model may represent many types of systems. One type may consist of a system that is referred to as an all fan-in system. In this type of system, only the fan-in of

tasks may be greater than one. The fan-outs are all equal to one. Figure 19 gives the replicated representation of the all fan-in system shown in Figure 18. In this system, the fan-out of task A to C is one. The fan-out of task B to C is also one as is the fan-out from task B to D. Figure 19 also illustrates a system with more than one tier. The replicated model may represent multi-layer systems. The replication of tasks may occur at any or at all layers. The replicated model may consist of a mixture of replicated and non-replicated components as seen in Figure 18. In this system, the A tasks and B tasks are replicated while tasks C, D, and E are single tasks. In general for an all fan-in system, since $\overline{F}_{AB}$ is equal to one, then $K_A \geq K_B$.

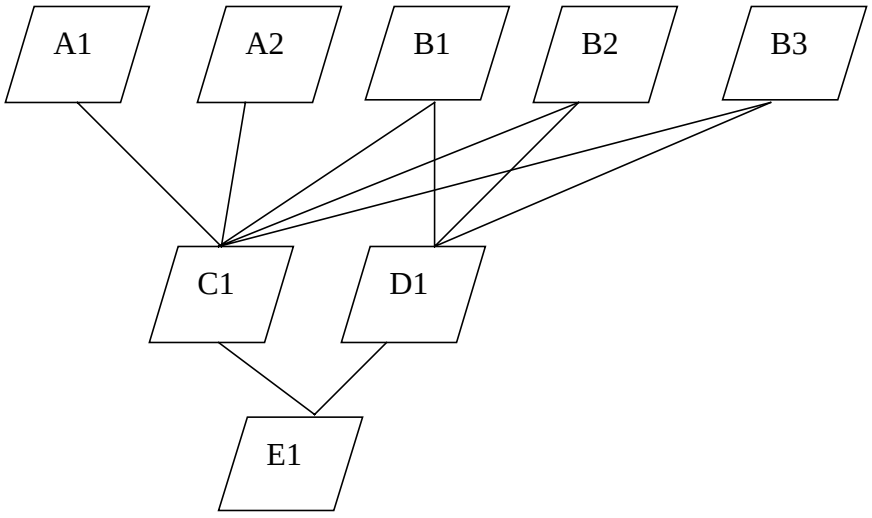


Figure 18: All Fan-In System

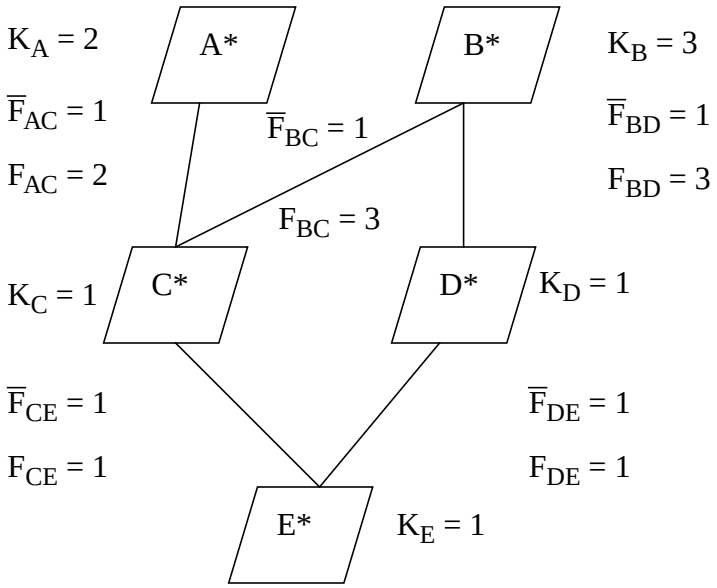


Figure 19: Replicated Representation of All Fan-In System (Figure 18)

Similarly, it is possible to have a system with only fan-outs. In the case of the all fan-out system, the fan-outs are all greater than one and the fan-ins are all equal to one. Figure 20 and Figure 21 illustrate such a system. In Figure 21, the fan-in to tasks B and C are one while the fan-outs of tasks A and B are 2. Again, in general, since the fan-in, $F_{AB,}$ is equal to one, then $K_B \geq K_A$.

A general system consists of both fan-in and fan-out. Figure 22 and Figure 23 illustrate such a system with various fan-in and fan-out values. Figure 22 is also another example of a multi-layered, mixed replicated system. That is, the top two layers consist of replicated tasks while the last layer consists of a non-replicated task.

In the examples considered, it has been assumed that all tasks have only one entry. However, it is possible to represent replicated tasks with multiple entries. The entries for a task or task set may have different fan-ins and fan-outs as shown in Figure 24.

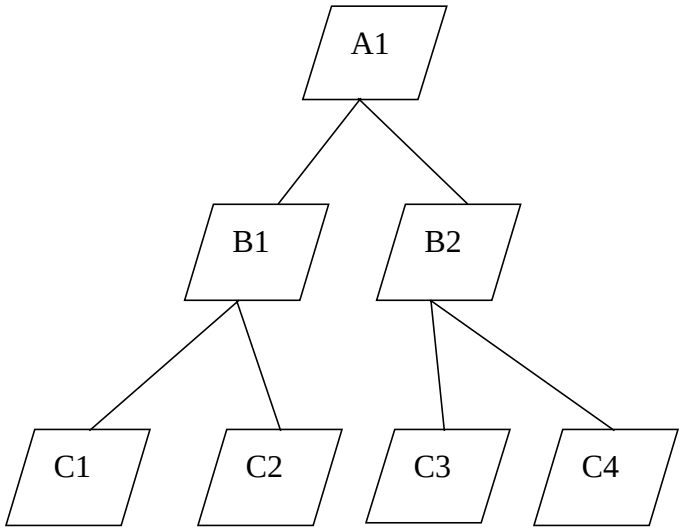


Figure 20: All Fan-Out System

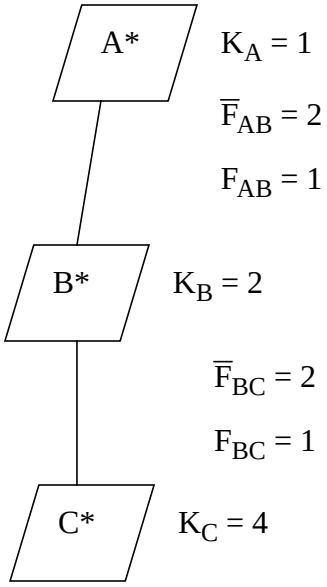


Figure 21: Replicated Representation of All Fan-Out System (Figure 20)

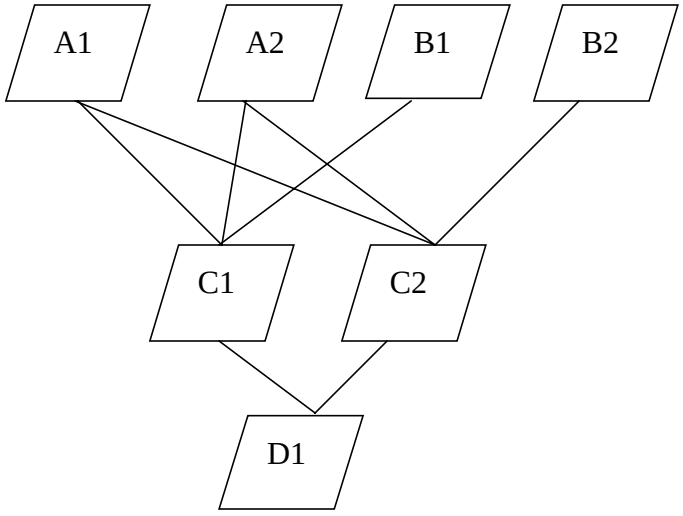


Figure 22: System with Fan-in and Fan-out

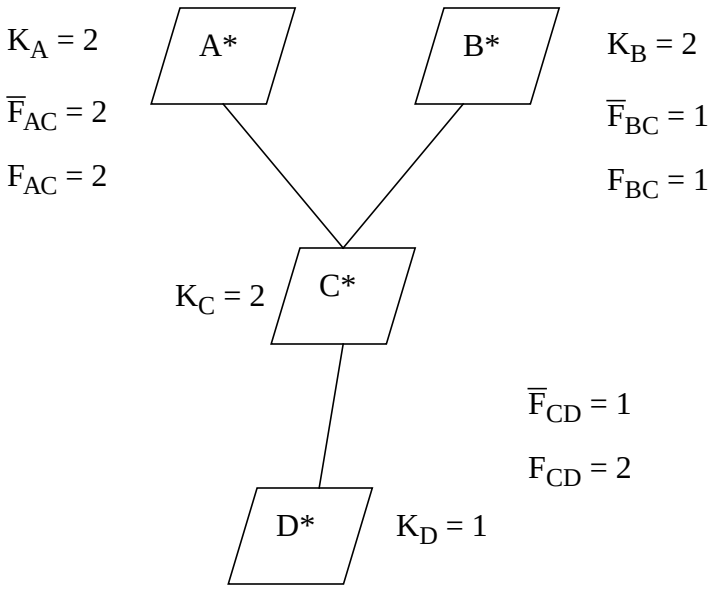


Figure 23: Replicated Notation for Figure 22

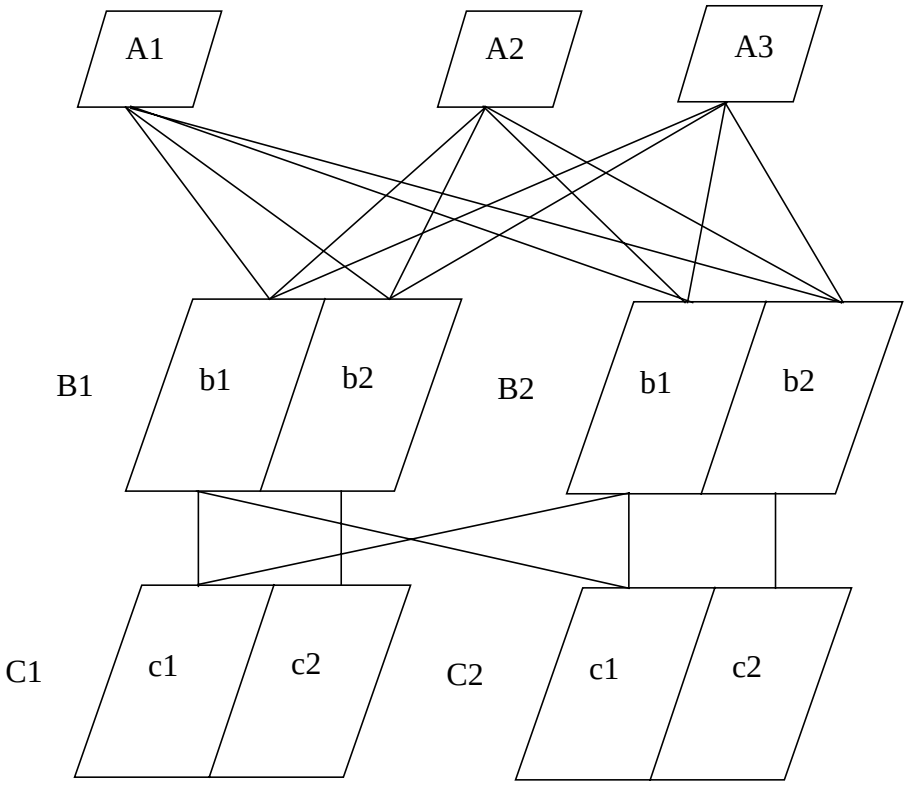


Figure 24: System with Entries

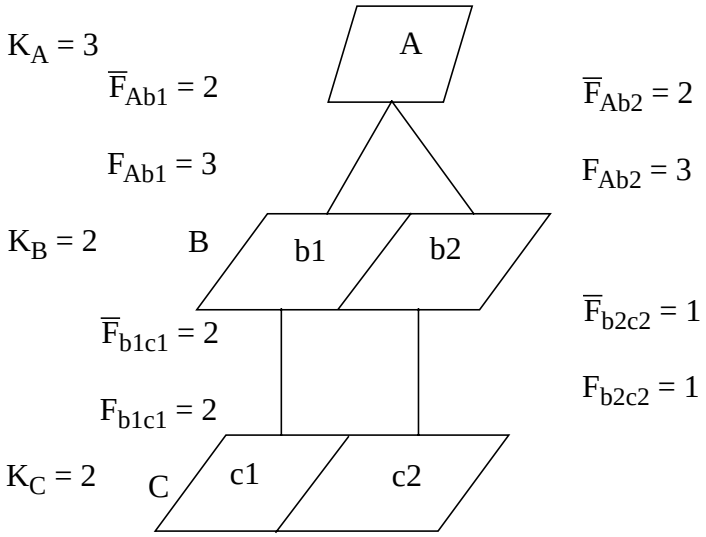


Figure 25: Replicated Representation of System with Entries

4.4 Identifying Replication

The symmetry of the whole system must be considered when defining tasks as being replicated. For example, in Figure 26, the tasks A1, A2, A3 and A4 have the same service times and visit ratios to their servers B1 and B2. The B1 and B2 tasks have the same service times but are not replicated tasks since they visit different tasks C and D. The delays encountered at tasks B1 and B2 are different, and thus in solving for the whole system, the updated service times for B1 and B2 are different. Because B1 and B2 are not replicated tasks, tasks A1, A2, A3, and A4 cannot be considered a replicated group because they in fact visit different servers. (However, tasks A1 and A2 are replicas as are task A3 and A4.)

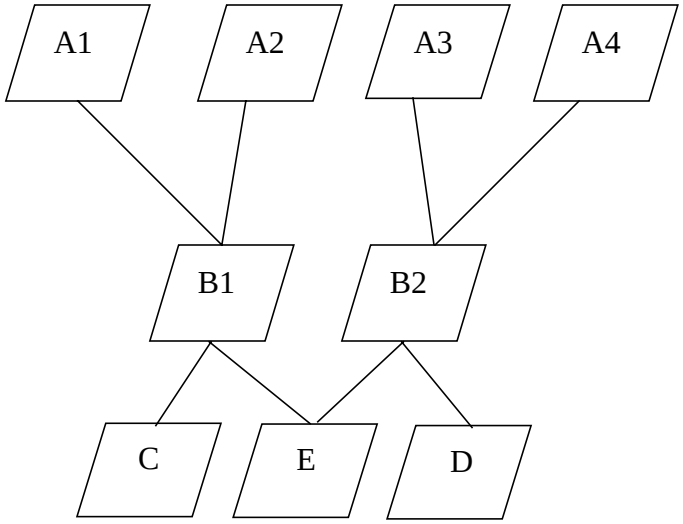


Figure 26: Deceptive Replication

4.5 Concentrated vs. Diffused Replication

Given the parameters for a replicated model, such as the number of replicas, fan-in, and fan-out, it is possible for some systems to have two interpretations of the replicated model. The two types of replications are defined as concentrated and diffused. The replicated representation of such a system is shown in Figure 27 [Wood 95b]. The corresponding concentrated and diffused interpretations of the parameters are shown in Figure 28 and Figure 29 respectively.

The performance results for both models are the same since the contention for tasks is the same. The concentrated model consists of two or more submodels. Solving for one submodel would be sufficient to get the performance results for the whole system. Therefore, it is reasonable to recommend that the submodel of the concentrated interpretation for a system be used for analysis since it represents a simpler model.

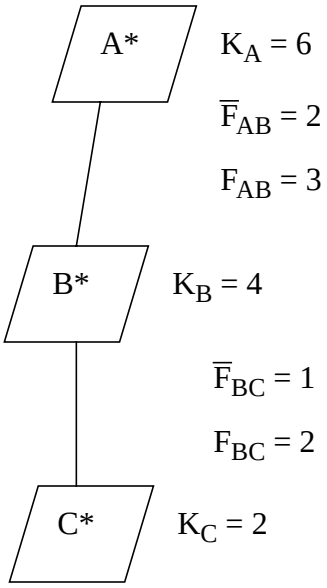


Figure 27: Replicated Model

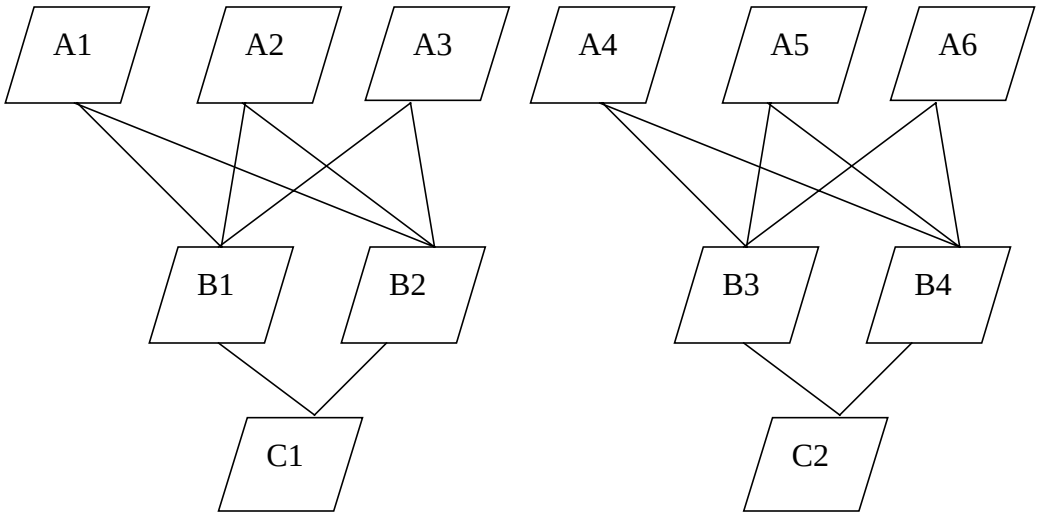


Figure 28: Concentrated Replication Interpretation of Figure 27

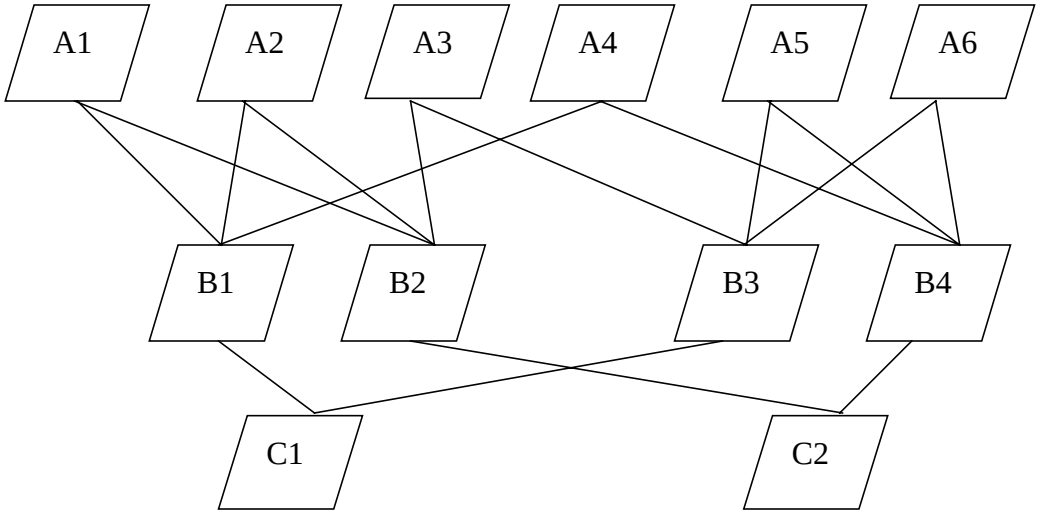


Figure 29: A Diffused Replication Interpretation of Figure 27

Chapter 5.0 Solving Models With Replication

5.1 Introduction

The notation presented in Chapter 4 greatly simplifies the model representation of a system with replicated components. In order to also simplify the solving of the model, a new simplified solution is introduced. This replication algorithm is applied by modifying the Layered Queueing Network Solver (LQNS).

The LQNS has limitations on the size of the system it can solve because of the computational time involved and the memory limitations. The LQNS can handle systems of up to 100 server tasks, 100 processors, and 100 client tasks. The computational time of the LQNS algorithm increases as the number of tasks (clients and servers) in the SRVN model increases. This occurs since each client or server, in each layer of the SRVN model, is represented as a station in the representative queueing network to be solved. The MVA algorithm used by LQNS takes longer to solve with an increase in the number of stations and chains. In addition, more memory is required to store information about each station. With more tasks, the LQNS also has more layer submodels to solve.

The replication algorithm proposed enables the LQNS software to solve large systems with replicated clients and servers in a timely manner with reduced memory requirements. The algorithm accelerates the solution by analyzing only one of the tasks in a group of replicas and by using the results for the one task to represent the others. In other words, a simplified replicated model of the system is solved thereby reducing the number of stations in the queueing network presented to the MVA algorithm.

In this chapter, the problem of solving the replicated model is presented. The way to construct chains for the model and the solution algorithm are described. The convergence of the algorithm is discussed followed by some implementation details. Finally, the results of using the algorithm and the limitations of the algorithm are highlighted.

5.2 Problem Statement

The LQNS solves a series of "layer submodels" by representing each submodel as a queueing network. It is desired to solve each submodel with a substitute queueing network in which the replicated stations are represented only once, and still obtain the same results (or nearly the same results) as if the queueing network for the full submodel had been solved. (This substitute queueing network will be referred to as the replicated queueing network.) First, constructing the chains for the replicated queueing network poses some problems. Secondly, in the replicated queueing network only one station of a set of replicated stations is used. Therefore, the delays that would be encountered at the other stations have to be accounted for. The example system shown in Figure 30 and Figure 31 is used to illustrate the problems of solving a "layer submodel".

Figure 30 represents an SRVN model of the system with replicated tasks. Tasks A1 and A2 have identical parameters such as the same phase service times and visits to tasks C1, C2, and C3. Likewise, tasks B1 and B2 are replicas as are tasks C1, C2, and C3. Task D1 and D2 are also identical. For simplicity, again, it is assumed the tasks have only one entry and one phase. Figure 31 shows the same system using the replication notation presented in Chapter 4.

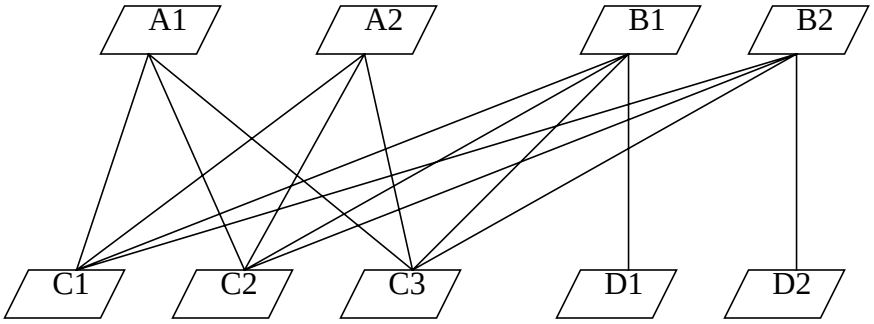


Figure 30: Full Model

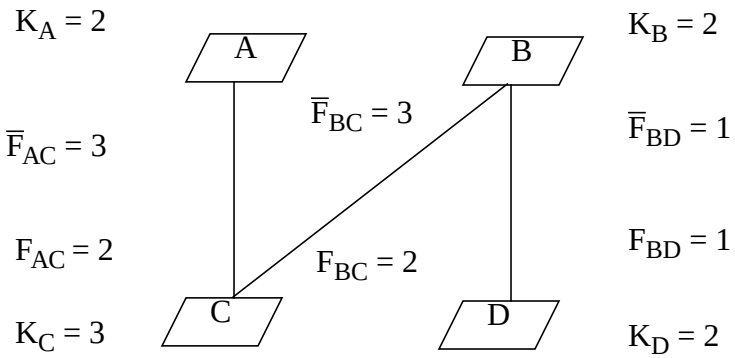


Figure 31: Replicated Model of Figure 30

To solve the full model of Figure 30, the queueing network that represents this model must be solved. The queueing network associated with the full model is shown in Figure 32. Note that the example system has only two layers of tasks and therefore, has only one layer submodel. The client tasks A1 and A2 are mapped to the delay servers A1 and A2 respectively. Similarly, tasks B1 and B2 are mapped to the delay servers B1 and B2. The five server tasks are each represented by a queueing station C1, C2, C3, D1, and D2. The SRVN service times for each task are mapped to the queueing network service times of their corresponding stations. The service times for the C stations are equal as are the service times of the D stations. There are also queueing network service times associated with the delay servers, S_{A1}, S_{A2}, S_{B1}, and S_{B2}. The service time relationships are specified below:

$S_{A1} = S_{A2}$; $S_{C1} = S_{C2} = S_{C3}$

$S_{B1} = S_{B2}$; $S_{D1} = S_{D2}$

In the terminology being used, the service time of a delay server equates to the service time of the second phase of the client task in the SRVN model.

The queueing network associated with the replicated model is shown in Figure 33. Again, the client tasks A and B of Figure 31 are represented as delay servers in the queueing network. The server tasks C and D are queueing stations. The service times of the stations are given below in relation to the service times of the full model:

$S_A = S_{A1}$; $S_C = S_{C1}$

$S_B = S_{B1}$; $S_D = S_{D1}$

The problem is to be able to solve the replicated queueing network of Figure 33 and obtain the same results as if the full queueing network of Figure 32 had been solved. If the full queueing network is solved, the results for each of the tasks in a group of replicas are the same. For example, the throughput for station C1, C2 and C3 are equal as are their

utilizations. The replicated model takes advantage of the symmetry. It is obviously advantageous to solve for the simpler queueing network of the replicated model (Figure 33) which has fewer stations. In addition, as the number of replicas increases, the replicated queueing network maintains the same number of stations.

5.3 Chain Construction

The queueing network of the full system as seen in Figure 32 is incomplete since the chains and corresponding visit ratios have not been specified to correspond to the SRVN model (Figure 30). The visit ratios of the queueing network correspond to the SRVN visit ratios between client and server tasks. The chains may be constructed for the full model as follows with the visit ratios specified below (N_k = number of tokens in chain k; $V_{k,m}$ = visits to station m by chain k tokens):

Chain 1: (server set = {A1, C1, C2, C3})
$N_1 = 1$
$V_{1,A1} = 1$; $V_{1,C1} = V_{1,C2} = V_{1,C3}$

Chain 2: (server set = {A2, C1, C2, C3})
$N_2 = 1$
$V_{2,A2} = 1$; $V_{2,C1} = V_{2,C2} = V_{2,C3}$

Chain 3: (server set = {B1, C1, C2, C3, D1})
$N_3 = 1$
$V_{3,B1} = 1$; $V_{3,C1} = V_{3,C2} = V_{3,C3}$; $V_{3,D1}$

Chain 4: (server set = {B2, C1, C2, C3, D2})
$N_4 = 1$
$V_{4,B2} = 1$; $V_{4,C1} = V_{4,C2} = V_{4,C3}$; $V_{4,D2}$

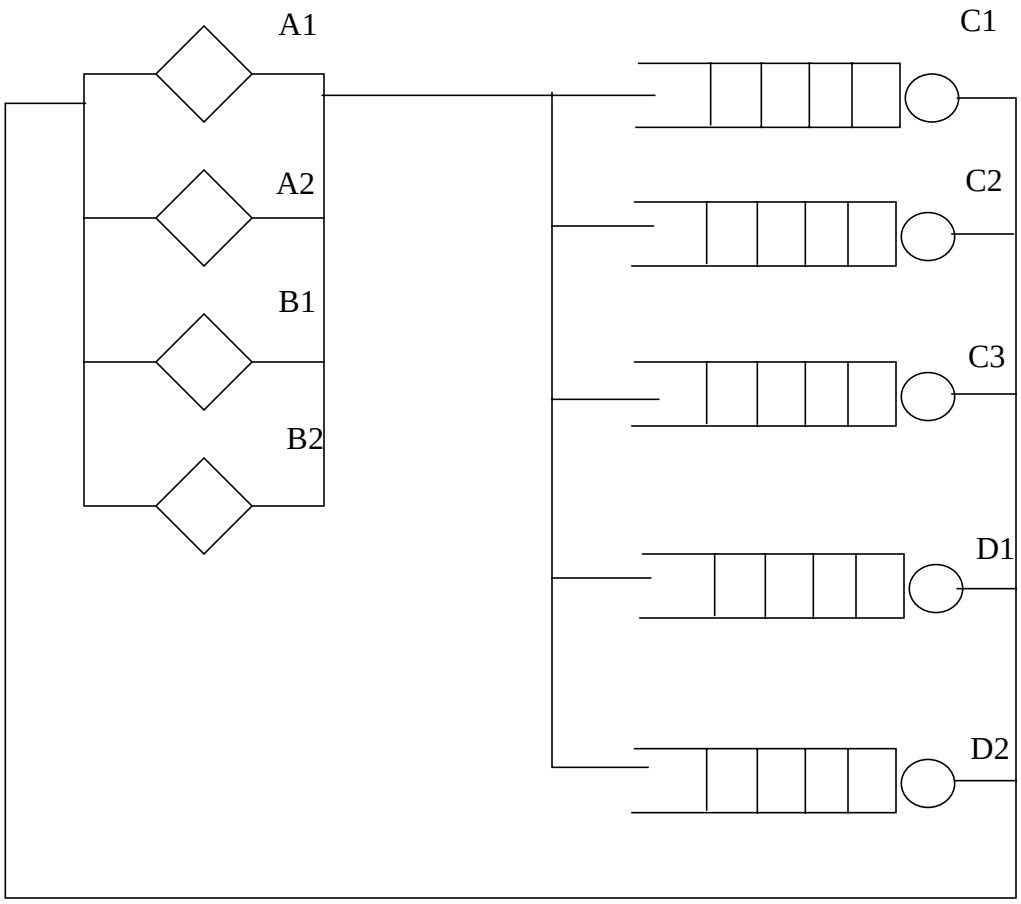


Figure 32: Queueing Network for Full Model (Figure 30)

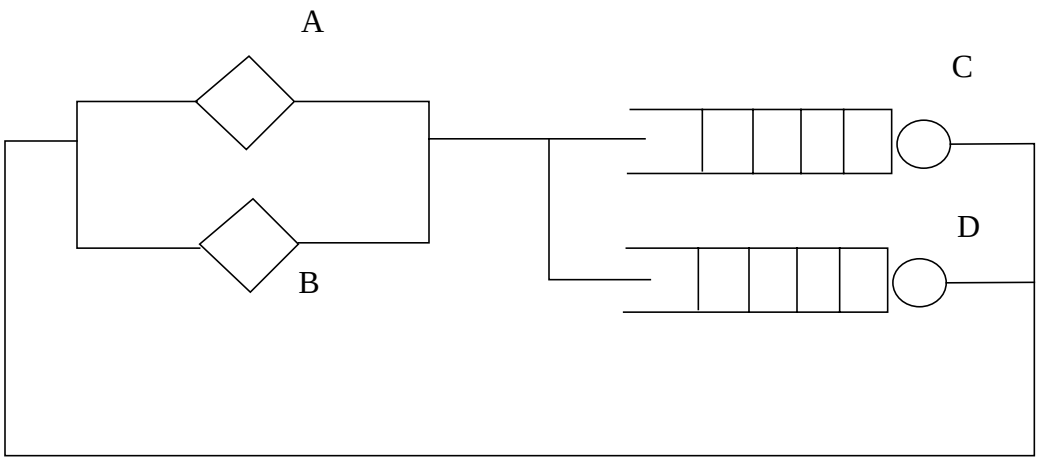


Figure 33: Queueing Network for Replicated Model (Figure 31)

The chains are constructed from the point of view of the client tasks. That is, a chain with one token is associated with each client. The token of chain 1 visits the delay server A1 once and visits stations C1, C2, and C3 an equal number of times. The token of chain 2 visits delay server A2 once and also visits stations C1, C2, and C3 an equal number of times. (Since the tokens for chains 1 and 2, associated with clients A1 and A2, visit the same stations with the same visit ratios, these two chains are equivalent and could be merged into one chain with two tokens (N = 2)). The chains associated with clients B1 and B2 cannot be combined into one chain since task B1 visits task D1 but task B2 visits D2. Therefore, chain 3 ($N_3 = 1$) consists of one token which visits delay server B1 once, stations C1, C2, C3 an equal number of times, and station D1 once. Similarly, chain 4 ($N_4 = 1$) consists of one token which visits delay server B2 once, stations C1, C2, C3 an equal number of times, and station D2 once. It can be seen that there is contention at stations C1,

C2, and C3 where altogether four tokens may visit. There is no contention at stations D1 and D2 since only one token visits these stations. There is also no contention at the delay servers which have no queues. The queueing network with the defined parameters and chains does indeed represent the corresponding SRVN model of the system (Figure 30).

For the replicated queueing network (Figure 33), the chains must be constructed so that the number of tokens contending for each resource or station is the same as in the full system. That is, there must be four tokens that visit station C, and one token that visits station D. The chains cannot be constructed in relation to the client as can be seen from this example. If the chains were constructed as in the full queueing network, chain 1 would consist of two tokens which visit station A once and station C once. This is consistent with the SRVN model (Figure 31). However, if chain 2 is allocated with two tokens (since $K_B = 2$), two tokens would visit station C which is consistent with the SRVN model, but two tokens would also visit station D, which is not consistent with the SRVN model. This chain could be split up into two chains with one token each. However, this would defeat the purpose of simplifying the full model. If task B represented a large set of replicas, a large number of chains would be needed. This would again slow down the MVA computation.

To construct the chains for the replicated model, the chains are constructed in relation to the server. In this way, the correct contention to each station is maintained and the number of chains required may be reduced. The chains for the example (Figure 31 and Figure 33) are constructed as follows:

Chain 1: $N'_1 = F_{AC} = 2$
$V'_{1A} = 1$; $V'_{1C} = V_{1,C1}$

Chain 2: $N'_2 = F_{BC} = 2$
$V'_{2B} = 1$; $V'_{2C} = V_{3,C1}$

Chain 3: $N'_3 = F_{BD} = 1$

$V'_{3B} = 1$; $V'_{3D} = V_{3,D1}$

As can been seen, a chain is constructed for each replicated group of clients that visits a server. The number of tokens for that chain is the fan-in of that server. For example, chain 1 consists of two tokens since the fan-in of task set C from task set A is two ($F_{AC} = 2$). There are two replicated tasks A that actually visit each task C. Another chain (chain 2) is allocated for the two tokens that visit one of the tasks of task set C from the tasks in task set B. From the chain allocation, it is verified that there are four tokens contending for station C and one token contending for station D as desired.

From this simple example, the method for constructing the chains for the replicated model does not seem to reduce the number of chains greatly (from 4 chains for the full model to 3 for the replicated model). It would even seem that for the full model only three chains are actually needed since chain 1 and 2 are equivalent. However, the reduced number of chains becomes very apparent if, in the example, task B were replicated by more than 2. For example, if task B were replicated by 20 times with the corresponding number of replicated tasks D (shown in Figure 34), the full model would require a chain for each B task. Figure 35 shows the corresponding queueing network with the chains specified. For the replicated model, however, the same three chains specified previously could be used with the number of tokens for chain 2 increased to 20 (see Figure 36 and Figure 37). The simplified replicated model and queueing network are virtually unaffected by the increase in the number of replicas. Only the parameters are affected. In addition, the approximate MVA computational time would not increase since it is not affected by the number of tokens in a chain but only by the number of chains and queueing stations.

In general for the replicated model, one chain is constructed for each client that visits a server. Each chain visits one client and one server. Therefore, a server and client may be traversed by many chains. The number of tokens in a chain is equal to the fan-in of the client to that server. (The fan-in reflects the number of client replicas that actually visit the server in the full model.) In essence, the replication of clients is represented by the number of tokens in the chain.

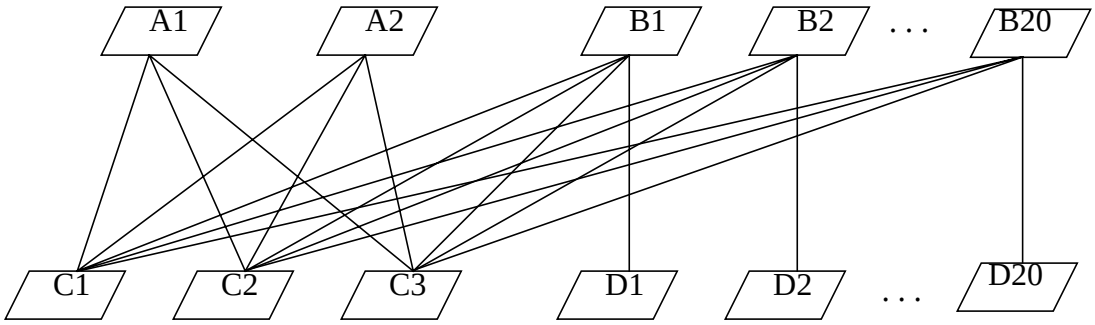


Figure 34: Full Model with Task B Replicated 20 Times

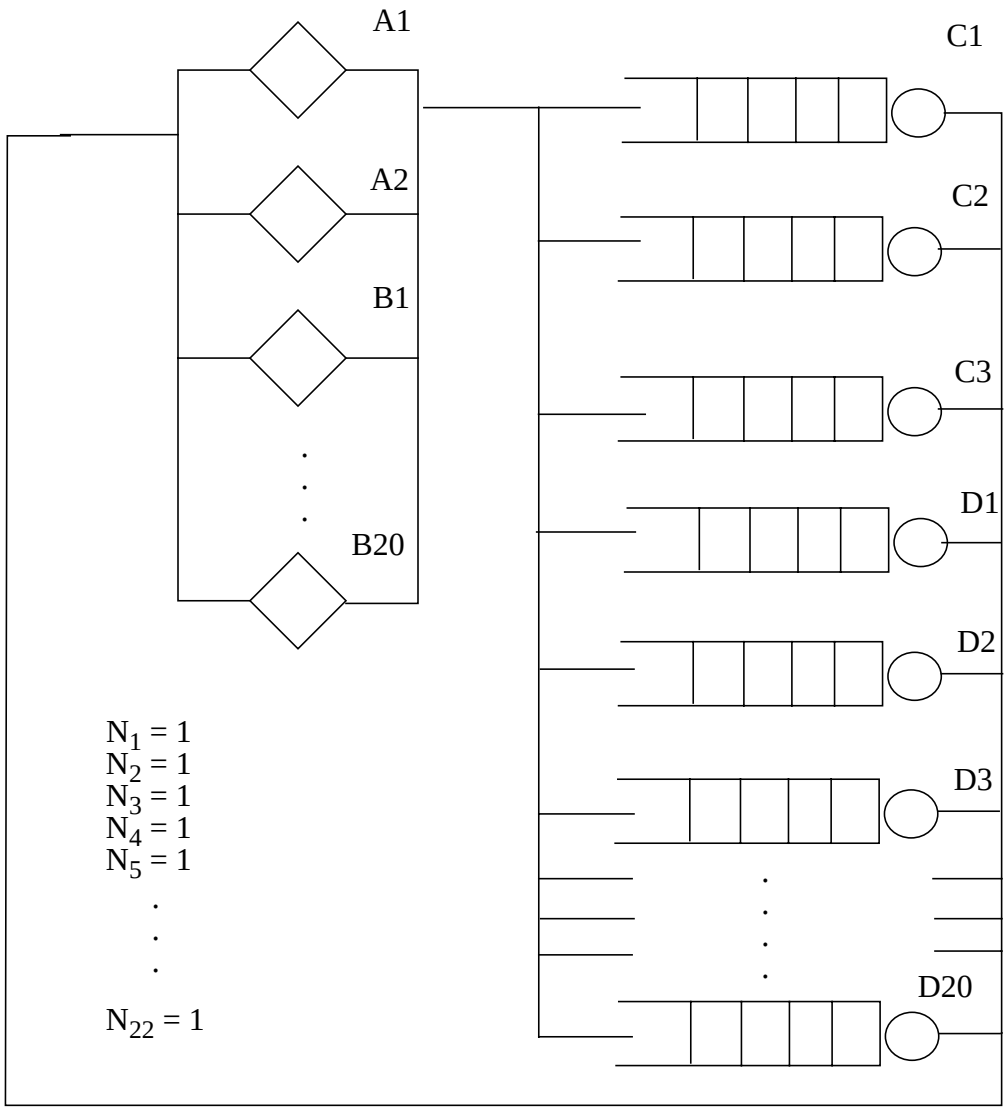


Figure 35: Queueing Network for Full Model of Figure 34

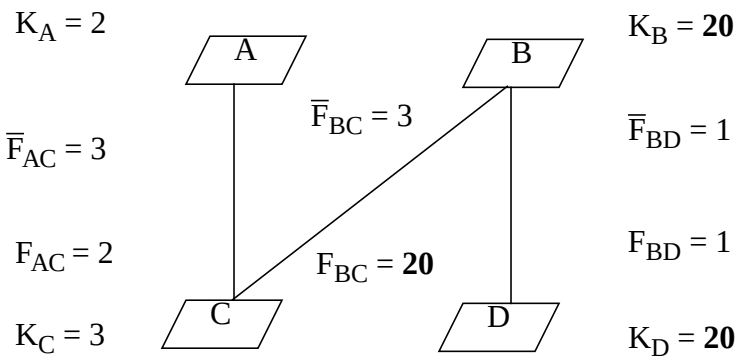


Figure 36: Replicated Model of Figure 34

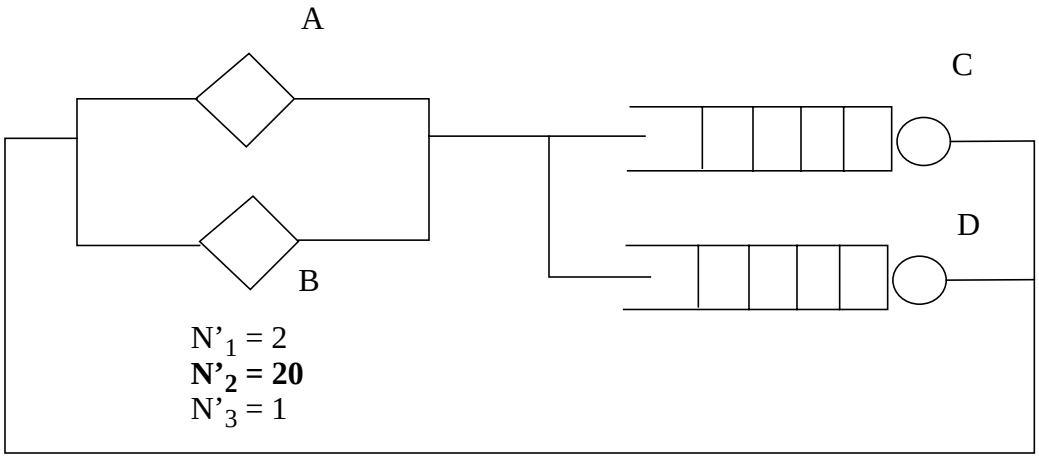


Figure 37: Replicated Queueing Network for Figure 36

5.4 Service Time Calculation

In assigning the chains and parameters of the queueing network for the replicated model (Figure 31), the contention at each station is equivalent to that of the queueing network for the full model. That is, the number of tokens visiting a station in the replicated model is the same as the corresponding station in the full model. However, the two queueing networks are still not equivalent in their solutions since there are delays at the clients for the full model that must be accounted for in the simplified replicated model.

To account for the delays of the 'missing' servers in the replicated model, the delays that would be seen at these stations are added to the service time of the delay servers (client tasks). In essence, the method of surrogate delays is employed. The delays of the 'missing' stations are added to the service time of the delay servers that also represent the client tasks.

In the replication example discussed (Figure 31), there is only one station C in the queueing network of the replicated model (Figure 33), whereas, there are three stations C1, C2 and C3 in the queueing network of the full model (Figure 30). The replicated queueing network leaves out two replicated C tasks.To compensate for the delay at the two C stations seen by the tokens of chain 1 visiting delay server A of the full network, twice the delay (R_{1C}) calculated at station C is added to the service time of delay server A. The queueing network is solved again with this new service time for delay server A. This is repeated until the delay at station C converges. In essence, the two stations (C2 and C3) for the full network, are represented as additional delays in the replicated queueing network, shown as additional shaded delay servers in Figure 38. The modified service time of station A for chain 1 is given as:

$$S'_{1A} = S_{1A} + (\bar{F}_{AC} - 1)R_{1C} \qquad \text{Eq (5.1)}$$

Where:

S'_{1A} = Modified service time of station A for chain 1

S_{1A} = Service time of station A for chain 1

R_{1C} = Residence time of chain 1 at station C = $Y_{1C}W_{1C}$

Y_{1C} = Number of visits to station C by chain 1

W_{1C} = Waiting time of chain 1 at station C (service time plus queueing time)

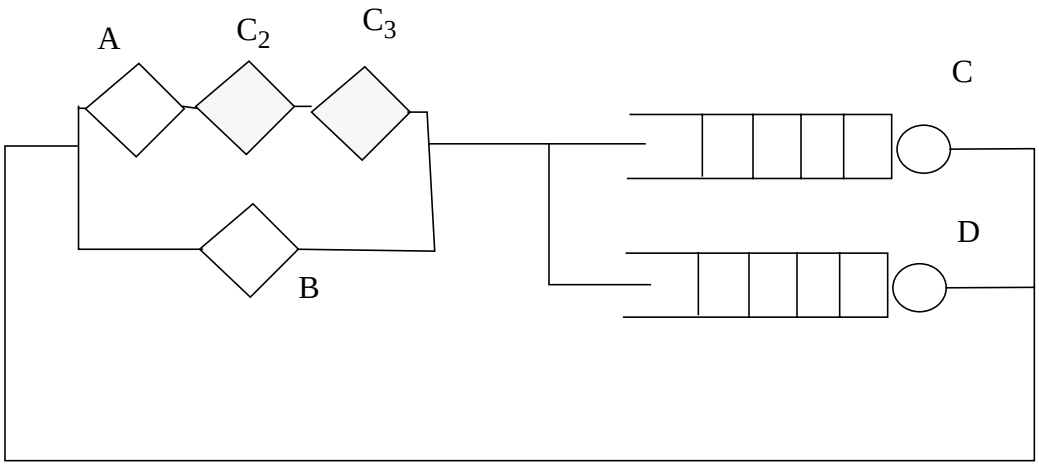


Figure 38: Replicated Sub-Queueing Network for Chain 1

The compensated delays for the 'missing' C stations are also needed for the tokens of the chains 2 and 3 that visit delay server B. For chain 2, the delay that would be seen at station D must also be accounted for. This delay may be gotten from the delay of chain 3. Similarly, for chain 3, the delay seen for the 'missing' C stations are gotten from the delay

of chain 2. The delays are added to the service times of delay server B for each chain. The modified service times for delay server B are as follows:

$$S'_{2B} = S_{2B} + (\bar{F}_{BC} - 1)R_{2C} + \bar{F}_{BD} \times R_{3D} \quad \text{Eq (5.2)}$$

Where:
- S'_{2B} = Modified service time of station B for chain 2
- S_{2B} = Service time of station B for chain 2
- R_{2C} = Residence time of chain 2 at station C
- R_{3D} = Residence time of chain 3 at station D

$$S'_{3B} = S_{3B} + (\bar{F}_{BD} - 1)R_{3D} + \bar{F}_{BC} \times R_{2C} \quad \text{Eq (5.3)}$$

Where:
- S'_{3B} = Modified service time of station B for chain 3
- S_{3B} = Service time of station B for chain 3
- R_{2C} = Residence time of chain 2 at station C
- R_{3D} = Residence time of chain 3 at station D

In essence the sub-queueing networks of Figure 40 and Figure 41 are solved with the results (delay at station C, $R_{2C,}$ and delay at station D, R_{3D}) exchanged between the two. The modified service times for delay server B are used in the next iteration for solving the queueing network. This iteration in solving for the queueing network produces new values for the delays at station C and D (i.e. R_{2C} and R_{3D}). The iteration is repeated until the delay value for station C converges and the delay value for station D converges.

By using the method of modifying the service times of the client delay servers to account for the delays at unrepresented replicated stations, the sub-queueing networks of Figure 38, Figure 40, and Figure 41 are effectively being solved. However, these sub-networks are not solved separately but are solved together in one invocation of the MVA solver for the replicated queueing network (Figure 33). The sub-networks are only presented to explain the method being employed which is essentially the method of surrogate delays.

In general, the equation to be employed in modifying the client service times for the replicated model is:

$$S'_{kt} = S_{kt} + (\overline{F}_{tm} - 1)R_{km} + \sum_{\forall M} \overline{F}_{tM} \times R_{KM} \qquad \text{Eq (5.4)}$$

Where:

- m = Server visited by chain k
- t = Client visited by chain k
- S'_{kt} = Modified service time of client station t for chain k
- S_{kt} = Service time of client station t for chain k
- $\overline{F}_{tm}$ = Fan-out of client task t to server m (SRVN model)
- R_{km} = Residence time of chain k at station m (m is visited by chain k)
- $\overline{F}_{tM}$ = Fan-out of client task t to server M (SRVN model)
- R_{KM} = Residence time of chain K at station M
- K = All other chains besides k that visit client t
- M = All other servers besides m invoked by client t

Note: Each chain visits only one source station (client) and one server.

That is, given tasks i, j, l as shown in Figure 39, equation (5.4) may be expressed as:

$$S'_{(ij)i} = S_{(ij)i} + (\overline{F}_{ij} - 1)R_{(ij)j} + \sum_{l \neq j} \overline{F}_{il} \times R_{(il)l} \qquad \text{Eq (5.5)}$$

Where: (*ij*) = Chain that represents client *i* requesting service from server task *j*.

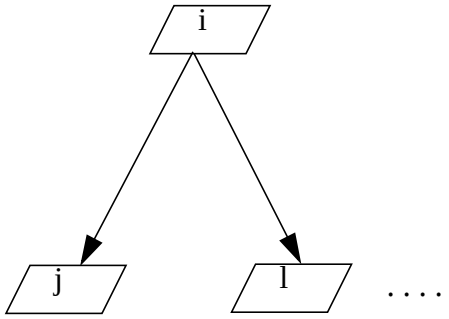


Figure 39: General Tasks

5.5 Interpretation of Results

The throughput results obtained from solving the replicated queueing network with the modified service times must be interpreted to obtain the throughput at each client task. The throughput of the client for a particular chain must be divided by the number of tokens in that chain. For example, in the replicated example (Figure 31 and Figure 33), the throughput for station or task A, X_A, is given as follows: $X_A = X_{1A}/N_1$ where X_{1A} is the throughput of chain 1 at station A. The throughput of any chain that visits station A may be used to obtain the throughput of station A. The same throughput should be obtained. For example, for client B in the replication example, the following holds true: $X_B = X_{2B}/N_2 = X_{3B}/N_3$. The throughput result is modified because the throughput obtained for a chain is the throughput of the chain which has the replicated number of tokens

representing the number of replicated clients. Therefore, to obtain the throughput of one replica client, the throughput of the chain must be divided by the number of tokens.

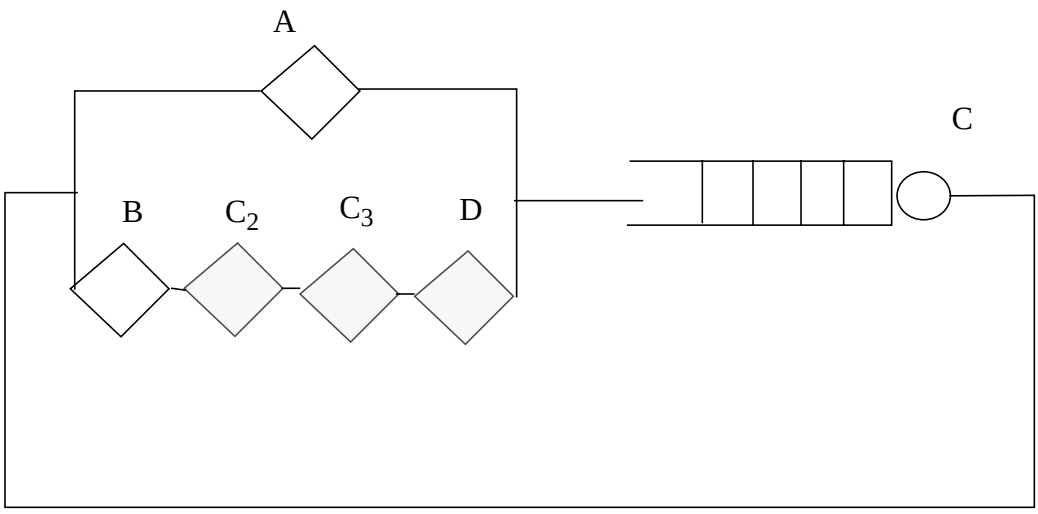


Figure 40: Replicated Sub-Queueing Network for Chain 2

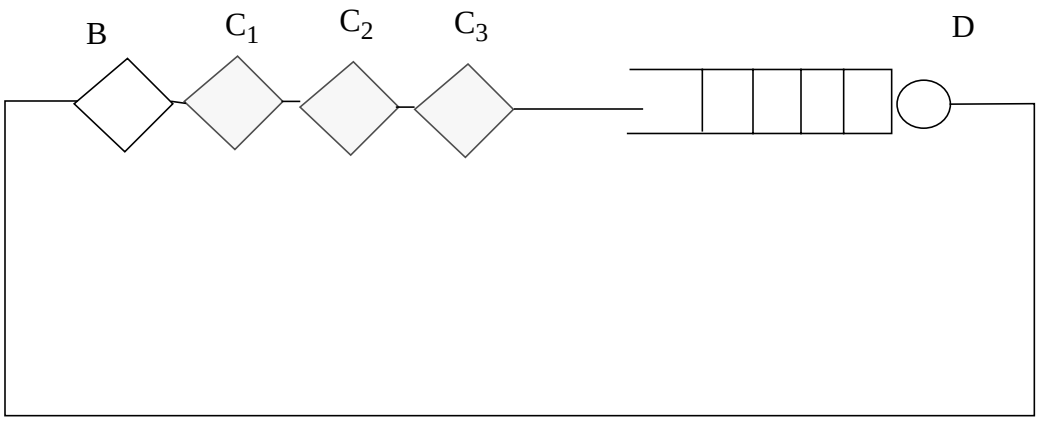


Figure 41: Replicated Sub-Queueing Network for Chain 3

5.6 Convergence

The method of surrogate delays, employed by the replication algorithm, is basically a Gauss-Seidel iteration (See Section 2.4). The surrogate delay (Z_k) of a chain k at a delay server is calculated from the waiting times of all chains that visit that delay server (W_{km}, where m is the server visited by chain k). Note that for the particular chain set up used in the replication algorithm, each chain visits one delay server and one queueing server/ station. From inspection of the MVA algorithm (Refer to Section 2.3.2), it is evident that the waiting times, W_{km}, are functions of the surrogate delays (Z_1, Z_2 ... Z_K) where K is the maximum number of chains that visits the delay server. (Note that $R_{km} = Y_{km}W_{km}$ where Y_{km} equals the number of visit to server m by chain k.) Thus, the Gauss-Seidel function for the replication algorithm can be written as:

$$Z_k^{(n+1)} = \sum_{l=1;\forall m}^{K} C_{lm} W_{lm} = f_k(Z_1^{(n)}, Z_2^{(n)}, \ldots, Z_K^{(n)}) \qquad \text{Eq (5.6)}$$

where C_{lm} = (Constant Fan-Out Term ($\overline{F}$)) x Y_{lm} = TotalVisits from chain l to server m.

Since k ranges from 1 to K, equation (5.6) represents a set of non-linear equations to be solved. The Gauss-Seidel iteration is used to solve this set of non-linear equations, as shown below:

$$\begin{aligned} Z_1^{(n+1)} &= f_1(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)}) \\ Z_2^{(n+1)} &= f_2(Z_1^{(n+1)}, Z_2^{(n)}, \ldots Z_K^{(n)}) \\ &\ldots \\ Z_K^{(n+1)} &= f_K(Z_1^{(n+1)}, Z_2^{(n+1)}, \ldots Z_K^{(n)}) \end{aligned} \qquad \text{Eq (5.7)}$$

The convergence of the Gauss-Seidel iteration is not guaranteed. In fact, for some cases of heavy utilization of a queueing server, the iteration oscillates or converges very slowly. (This was actually observed in the example case described in Section 5.3 with twenty B tasks shown in Figure 36.) To counter the convergence problem, the implementation of the algorithm provides an option whereby a simplified Newton-Raphson method may be employed instead of the Gauss-Seidel iteration. The Newton-Raphson method is chosen since it may converge where a Gauss-Seidel iteration fails to converge and also converges more rapidly than the Gauss-Seidel method (See Chapter 2). ([Chow 83] and [Souza 84] discuss the use of the Newton-Raphson method to accelerate convergence. [Eager 84] and [Patti 90] also discuss convergence issues.) However, unfortunately, the Newton-Raphson method also does not guarantee convergence, In fact, in some cases, it may even cause an iteration to diverge. It is for this reason that the method is used only as an option when the Gauss-Seidel iteration fails to converge.

A simplified version of the Newton-Raphson is implemented since the full method requires a considerable amount of work per iteration and is thus less practical. This will become apparent as the equations for the Newton-Raphson iteration are developed below.The full Newton-Raphson method is applied to equation (5.6) giving the new iteration function:

$$Z_k^{(n+1)} = f_k(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)}) + \sum_{l=1}^{K} \left(\left(\frac{\partial}{\partial Z_l} f_k(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)}) \right) (Z_l^{(n+1)} - Z_l^{(n)}) \right) \qquad k = 1, \ldots K \qquad \text{Eq (5.8)}$$

Equation (5.8) represents the following set of non-linear equations to be solved:

$$\begin{aligned}
Z_1^{(n+1)} &= f_1(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)}) + \sum_{l=1}^{K} \left(\left(\frac{\partial}{\partial Z_l} f_1(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)}) \right) (Z_l^{(n+1)} - Z_l^{(n)}) \right) \\
Z_2^{(n+1)} &= f_2(Z_1^{(n+1)}, Z_2^{(n)}, \ldots Z_K^{(n)}) + \sum_{l=1}^{K} \left(\left(\frac{\partial}{\partial Z_l} f_2(Z_1^{(n+1)}, Z_2^{(n)}, \ldots Z_K^{(n)}) \right) (Z_l^{(n+1)} - Z_l^{(n)}) \right) \\
&\ldots \\
Z_K^{(n+1)} &= f_K(Z_1^{(n+1)}, Z_2^{(n+1)}, \ldots Z_K^{(n)}) + \sum_{l=1}^{K} \left(\left(\frac{\partial}{\partial Z_l} f_K(Z_1^{(n+1)}, Z_2^{(n+1)}, \ldots Z_K^{(n)}) \right) (Z_l^{(n+1)} - Z_l^{(n)}) \right)
\end{aligned} \qquad \text{Eq (5.9)}$$

The Newton-Raphson method involves setting a set of non-linear equations and finding the partial derivative of each function with respect to each variable (Z_1, Z_2, ... Z_K). Evidently the work for each iteration is enormous especially if there are many chains which increases the number of equations to be solved. The simplified Newton-Raphson method may be used to produce more manageable equations. In this method, the derivative is taken with respect only to one variable. That is, each equation is regarded as though it were an equation for just one of the unknowns [Davis 86].With the simplified Newton-Raphson method, a set of non-linear equations need not be solved per iteration. The simplified Newton-Raphson function, shown below, is used instead of equation (5.8):

$$Z_k^{(n+1)} = f_k(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)}) + \left(\frac{\partial}{\partial Z_k} f_k(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)})\right)(Z_k^{(n+1)} - Z_k^{(n)}) \qquad k = 1, \ldots K \qquad \text{Eq (5.10)}$$

The expanded form is:

$$\begin{aligned}
Z_1^{(n+1)} &= f_1(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)}) + \left(\frac{\partial}{\partial Z_1} f_1(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)})\right)(Z_1^{(n+1)} - Z_1^{(n)}) \\
Z_2^{(n+1)} &= f_2(Z_1^{(n+1)}, Z_2^{(n)}, \ldots Z_K^{(n)}) + \left(\frac{\partial}{\partial Z_2} f_2(Z_1^{(n+1)}, Z_2^{(n)}, \ldots Z_K^{(n)})\right)(Z_2^{(n+1)} - Z_2^{(n)}) \\
&\ldots \\
Z_K^{(n+1)} &= f_K(Z_1^{(n+1)}, Z_2^{(n+1)}, \ldots Z_K^{(n)}) + \left(\frac{\partial}{\partial Z_K} f_K(Z_1^{(n+1)}, Z_2^{(n+1)}, \ldots Z_K^{(n)})\right)(Z_K^{(n+1)} - Z_K^{(n)})
\end{aligned} \qquad \text{Eq (5.11)}$$

Substituting in the value for $f_k(Z_1^{(n)}, Z_2^{(n)}, \ldots Z_K^{(n)})$ from equation (5.6) into equation (5.10) gives:

$$Z_k^{(n+1)} = \sum_{l=1;\forall m}^{K} C_{lm} W_{lm} + \left(\frac{\partial}{\partial Z_k} \sum_{l=1;\forall m}^{K} C_{lm} W_{lm}\right) \cdot (Z_k^{(n+1)} - Z_k^{(n)}) \qquad \text{Eq (5.12)}$$

It seems probable that the major effect of a change in Z_k will be on its own chain performance, i.e. on W_{km}. Therefore, only the W_{km} term in the second summation in equation (5.12) is retained. This also simplifies the calculation. The resulting iteration function is used:

$$Z_k^{(n+1)} = \sum_{l=1;\forall m}^{K} C_{lm} W_{lm} + \left(C_{km} \cdot \frac{\partial W_{km}}{\partial Z_k}\right) \cdot (Z_k^{(n+1)} - Z_k^{(n)}) \qquad \text{Eq (5.13)}$$

This is the Gauss-Seidel iteration with an additional term. The partial derivative of the waiting time (W_{km}) may be evaluated as follows using the Schweitzer approximation to the arrival instant queue length (Refer to Section 2.3.2):

$$W_{km} \cong \frac{N_k - 1}{N_k} \cdot \bar{n}_{km} S_{km} \qquad \text{Eq (5.14)}$$

$$\frac{\partial W_{km}}{\partial Z_k} \cong \frac{N_k - 1}{N_k} \cdot \frac{\partial \bar{n}_{km}}{\partial Z_k} \cdot S_{km} \qquad \text{Eq (5.15)}$$

Where: $\bar{n}_{km}$ = Average number of tokens for chain *k* at server *m*

N_k = Total number of tokens for chain *k*

S_{km} = Service time of server m for chain *k*

To evaluate equation (5.15) further, an assumption is made that a change in Z_k, ΔZ_k, changes the number of tokens of chain *k* at the delay station proportionally. That is,

$$\begin{aligned} n_{zk} &= X_k \cdot Z_k \\ \Delta n_{zk} &= X_k \cdot \Delta Z_k \end{aligned} \qquad \text{Eq (5.16)}$$

Where: X_k = Throughput of chain *k*

The $-\Delta n_{zk}$ tokens that are taken away from the delay server are distributed to the server queue in proportion to those already there. Therefore,

$$\Delta \bar{n}_{km} = -\Delta n_{zk} \times \frac{\bar{n}_{km}}{N_k - 1} = -\left(X_k \cdot \Delta Z_k \cdot \frac{\bar{n}_{km}}{N_k - 1} \right) \qquad \text{Eq (5.17)}$$

Substituting into equation (5.15) gives the following:

$$\frac{\partial W_{km}}{\partial Z_k} = -X_k \frac{\bar{n}_{km}}{N_k} \cdot S_{km} \qquad \text{Eq (5.18)}$$

Returning to equation (5.13), the simplified Newton-Raphson function may be evaluated as follows:

$$Z_k^{(n+1)} = \sum_{l=1;\forall m}^{K} C_{lm} W_{lm} + C_{km} \cdot \left(-X_k \frac{\bar{n}_{km}}{N_k} \cdot S_{km} \right) \cdot (Z_k^{(n+1)} - Z_k^{(n)}) \qquad \text{Eq (5.19)}$$

Let,

$$\alpha = C_{km} \cdot \left(X_k \frac{\bar{n}_{km}}{N_k} \cdot S_{km} \right) \qquad \text{Eq (5.20)}$$

and solving for $z_k^{(n+1)}$ gives the final update function:

$$Z_k^{(n+1)} = \frac{\sum_{l=1;\forall m}^{K} C_{lm} W_{lm} + \alpha \cdot Z_k^{(n)}}{1 + \alpha} \qquad \text{Eq (5.21)}$$

In terms of the SRVN model and the LQNS replication implementation, the service time is modified as follows from equation (5.4):

$$S_{kt}'^{(n+1)} = S_{kt}^{(n+1)} + Z_k^{(n+1)} \qquad \text{Eq (5.22)}$$

The modification is similar to equation (5.4) used for the Gauss-Seidel iteration except with the additional terms in the Z_k expression. The SRVN terms to evaluate Z_k are:

$$\sum_{l=1;\forall m}^{K} C_{lm} W_{lm} = (\bar{F}_{ts} - 1) R_{ks} + \sum_{(\mu,\sigma) \in (K,S)} \bar{F}_{t\sigma} \times R_{\mu,\sigma} \qquad \text{Eq (5.23)}$$

$$\alpha = (\bar{F}_{ts} - 1) \cdot \left(\frac{X_k L_{ks} S_{ks} Y_{ks}}{N_k} \right)^{(n)} \quad \text{Eq (5.24)}$$

Where:

- s = Server visited by chain k
- t = Client visited by chain k
- S'_{kt} = Modified service time of client station t for chain k
- S_{kt} = Service time of client station t for chain k
- $\bar{F}_{ts}$ = Fan-out of client task t to server s (SRVN model)
- R_{ks} = Residence time of chain k at station s (s is visited by chain k)
- $\bar{F}_{tS}$ = Fan-out of client task t to server S (SRVN model)
- $R_{\mu, \sigma}$ = Residence time of chain μ at station σ
- K = Set of all other chains besides k that visit client t
- S = Set of all other servers besides s invoked by client t
- X_k = Throughput of chain k (MVA queueing network parameter)
- L_{ks} = Queue length of chain k at server s (queueing network parameter)
- S_{ks} = Service time of chain k at server s (queueing network parameter)
- Y_{ks} = Number of visits of chain k to server s (queueing network parameter)
- N_k = Number of tokens for chain k (queueing network parameter)

Note: Each chain visits only one client (delay server) and one server.

5.7 Implementation

The replication algorithm with the simplified Newton-Raphson method was implemented in the LQNS software. As discussed in Chapter 3, the LQNS is presented with an SRVN model with possibly several layers. Each layer is transformed into a queueing network submodel and solved via the MVA solver. The method of surrogate delays is used to account for the interaction between the layers. Thus, there is a Gauss-Seidel iteration between the results of the submodels.

The replication algorithm is applied at the submodel level and does not involve the iteration between submodels. The MVA solver code that solves each submodel was not changed. Only the input parameters presented to the MVA solver were manipulated to get the required results. The parameters were obtained by modifications mainly to the chain construction and service time calculation areas of the software.

Although the replication algorithm does not involve the inter-submodel iteration, referred to as the “outer” iteration, the algorithm does include an iteration of its own within the submodel, referred to as the “inner” iteration. The “inner” iteration is the Gauss-Seidel iteration or Newton-Raphson method discussed in the previous sections in which the service times for delay servers are modified. The check for the convergence of the iteration occurs after each invocation to the MVA solver. If convergence has not occurred, the service time values for delay servers (or client tasks) are modified and the MVA solver is invoked again. The pseudo-code for method Layerize::solveLayer shown in Figure 42 illustrates the “inner” iteration. Note that the inputs to this method are SRVN model parameters such as client task objects and server entity objects.

```
Layerize::solveLayer(clients, servers, layer number, validity flag)
BEGIN
    Initialize values;
    MakeChains;                 %Create chains and associate them with clients and servers
    Create the clients for the MVA model;
    Create the servers for the MVA model;
    DO replication iteration
        Initialize values;
        Set validity flag to false;
        IF first iteration
            IF layer has replicated tasks
                ModifyClientServiceTime for each client;
            ELSE
                Set validity flag to true;
                Set iteration count to limit;    %Layer has no replicated tasks.
                Set convergence to false;        %Execute loop only once.
            ENDIF;
        ELSE
            ModifyClientServiceTime for each client;
        ENDIF;

        IF convergence
            Set validity flag to true;
            Exit iteration loop
        ENDIF;

        Generate MVA model;                      %Open and closed
        Solve Model;.
        Store results from MVA model to SRVN model;
    WHILE (iteration limit not reached);
    Cleanup;
    RETURN validity flag;
END
```

Figure 42: Pseudo-Code for "Inner" Iteration

The chains for the MVA model are constructed in method Layerize::makeChains. The chains are allocated differently for replicated and non-replicated client tasks. If a certain client task is not replicated, one chain is constructed for all tokens that visit that client and

all its servers. If the client task is replicated, one chain is constructed for each server path. The pseudo-code is shown in Figure 43.

```
Layerize::makeChains(clients, servers, customers vector, thinktime vector, priority vector,
                     clientChains vector, serverChains vector)
BEGIN
    Initialize values;
    FOR all clients                %Create chains and associate them with clients and servers
        Get all servers for this client;
        IF client is NOT replicated
            Increment chain number k;
            Add chain number k to clientChain vector;
            Set thinktime to client idle time;
            Set number of customers and priority;
            Add chain number k to all servers of this client;
        ELSE
            %Replicated case
            FOR all servers of this client
                Increment chain number k;
                Add chain number k to serverChains;
                Add chain number k to clientChains;
                Set thinktime, number of customers, priority;
            ENFOR;
        ENDIF;
    RETURN number of chains;
END
```

Figure 43: Pseudo-Code for Chain Construction

The method Task::modifyClientServiceTime is applied to each replicated client task to modify the service time to account for the “missing” replicated tasks. A service time is associated with each chain that visits each entry of the client task. As a consequence, there are two loops; a loop through all entries, and an inner loop through all chains to an entry.

```
Task::ModifyClientServiceTime
BEGIN
        Initialize values;
        FOR all entries of this client task
                IF first iteration
                        initialize;
                ENDIF;

                FOR all chains of this entry
                        FOR all phases of this entry
                                Set the service time to Entry->waitExceptChain;
                        ENDFOR;
                ENDFOR;
        ENDFOR;
        RETURN delta value;
END
```

Figure 44: Pseudo-Code for ModifyClientServiceTime

The method Entry::waitExceptChain is where the service time is actually modified. The method is called for each entry with an input parameter of the chain. To modify the service time of the entry for a particular chain, the resident times of all entries that the entry calls is obtained. The service time is modified according to equation (5.4) and according to equation (5.22) if the Newton-Raphson method is used.

The input file format for the LQNS solver, shown in Figure 46, is used to specify the SRVN model with replicated tasks. The last lines specifies the fan-in and fan-out with the notation ‘i’ for fan-in and ‘o’ for fan-out. The first grouping, the paragraph beginning with ‘P’, specifies the processors which may also be replicated. The replication is specified with

```
Entry::waitExceptChain
BEGIN
        Initialize values;
        FOR all calls from this entry
                IF chain k does not visit the task called
                        sum = sum + (delay to this task) (fanout of this entry to called task);
                ELSE
                        sum = sum + (delay to this task) (fanout-1 of this entry to called task);

                        IF first iteration
                                Initialize
                        ELSE
                                Calculate the Newton-Raphson factor F;          %Equation (5.20)
                        ENDIF;
                        Calculate delta values;
                        Store service time values;
                ENDIF;
        ENDFOR;
        Repeat above code for processor task.
        sum = sum + F(oldsum)
        sum = sum/(1+F)                                                         %Equation (5.22)
        RETURN sum;
END
```

Figure 45: Pseudo-Code for waitExceptChain

an ‘r’ followed by the replication factor. The same notation is used for tasks which are shown in the paragraph beginning with ‘T’. (Refer to [Petriu 95] for a detailed description of the input file.)

```
# SRVN Model Description File

G "" 0.000010 50 10 0.900000 -1

P 4                                 # 4 processors
 p A1 f r 2                         # r2 for 2 replicas
 p B1 f r 2
 p C1 f r 3
 p D1 f r 2
-1

T 4                                 # 4 tasks
 t A1 r A1 -1 A1 0 r 2              # r2 for 2 replicas
 t B1 r B1 -1 B1 0 r 2              # Each task has an entry of the same name
 t C1 n C1 -1 C1 0 r 3
 t D1 n D1 -1 D1 0 r 2
-1

E 4
 s A1 0.000000 2.000000 -1
 y A1 C1 0.000000 2.000000 -1
 s B1 0.000000 4.000000 -1
 y B1 C1 0.000000 3.000000 -1
 y B1 D1 0.000000 4.000000 -1
 s C1 3.000000 0.000000 -1
 s D1 5.000000 0.000000 -1
 i A1 C1 2 -1                       # i specifies fan-in for entry A1 to entry C1
 o A1 C1 3 -1                       # o specifies fan-out
 i B1 C1 2 -1
 o B1 C1 3 -1
 i B1 D1 1 -1
 o B1 D1 1 -1
-1
```

Figure 46: Input File for Replication Example

5.8 Tests

To test the solution method for replicated systems, several models in their simplified replicated forms and their corresponding full model forms were solved using LQNS. The results from the replication algorithm were compared with the results of solving the full model using exact MVA and the Bard-Schweitzer approximation. Five cases are presented below. Case 1 is the example system discussed consisting of two layers of tasks with both fan-in and fan-out. Case 2 is a mixed model with both replicated and non-replicated tasks.

Case 3 verifies that the replication method can solve for tasks with multiple entries. Case 4 is a multi-layered system with replicated tasks. Finally, in Case 5, the example of Figure 36, which failed to converge with the Gauss-Seidel iteration, is solved using the Newton-Raphson method in the solution method.

The example system discussed, Case 1, in its full model form (Figure 30) as well as its replicated model form (Figure 31), was solved using the LQNS tool with the replication modifications. The figures do not show processor allocation. However, the processor allocation is specified in the corresponding input files (Figure 46) which describe the models. In the models solved, one task is assigned one unique processor. The following input parameters were used:

Full Model	Replicated Model
$S_{A1} = S_{A2} = S_{A3} = 2$	$S_A = 2$
$S_{B1} = S_{B2} = 4$	$S_B = 4$
$S_{C1} = S_{C2} = S_{C3} = 3$	$S_C = 3$
$S_{D1} = S_{D2} = 5$	$S_D = 5$
$V_{1,A1} = 1$	$V_{1,A} = 1$
$V_{1,C1} = V_{1,C2} = V_{1,C3} = 2$	$V_{1,C} = 2$
$V_{2,A2} = 1$	$V_{2,B} = 1$
$V_{2,C1} = V_{2,C2} = V_{2,C3} = 2$	$V_{2,C} = 3$
$V_{3,B1} = 1$	$V_{3,B} = 1$
$V_{3,C1} = V_{3,C2} = V_{3,C3} = 3$	$V_{3,D} = 4$
$V_{3,D1} = 4$	
$V_{4,B2} = 1$	
$V_{4,C1} = V_{4,C2} = V_{4,C3} = 3$	
$V_{4,D2} = 4$	

Three different queueing network solvers were used: the Schweitzer approximation, Linearizer, and the exact MVA. The results for solving the full system and for solving the simplified replicated system are shown in Tables 1 to 4. The cycle time result includes the service time of a task plus the rendezvous delay time it encounters with its servers. The results of solving the full model using the exact MVA are used to compare the replication results. It is seen that the replication algorithm works best with the Schweitzer approximation with an error of less than 5%. The replication algorithm using the exact MVA and the Linearizer provide fairly high errors, up to 12%.

The higher errors for the exact MVA and Linearizer may be explained by inspecting the MVA algorithm. (Refer to Section 2.3.2.) Modifying the service time of the client delay server with the delay at the 'missing' stations is essentially estimating the *R* values (resident times of chains at the stations) in the MVA algorithm. However, in the case of the exact MVA and Linearizer, the *R* value for different populations are required in the MVA iteration. For the exact MVA, the *R* value for populations from $\overline{0}$ to $\overline{N}$ is required. For the Linearizer, the *R* values for population $\overline{N}$ and $\overline{N\text{-}1}_c$ are required. By modifying the service time of the client, an estimation of *R* for a population of $\overline{N}$ is used, which is fixed throughout the iteration. That is, it is used even though for the exact MVA an *R* value for the range of populations $\overline{0}$ to $\overline{N}$ is needed. The estimation for *R* is incorrect and therefore produces big errors in the exact MVA and Linearizer.

The error for the Schweitzer approximation is low since, in this algorithm, only the delays for population $\overline{N}$ are needed. In this case, the estimated *R* is correct, or nearly so. The error in the results is due to the Schweitzer approximation itself. The Schweitzer approximation results for the full models are very close to their corresponding replicated model results also using Schweitzer. In addition, the error for the cycle time result is increased since the cycle time is obtained by multiplying the calculated delay at the client

by the number of visits to a server. In other words, the error from the replication algorithm appears in the delay result of the client (delay server) which is magnified in the client cycle time result by the number of visits.

Table 1: Cycle Time Results for Replication Example (Case 1)

Task Name	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Schweitzer)		%Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	**Exact MVA**	**Schweitz**
A	0	33.092	0	33.753	0	33.748	2	-0.01
B	0	73.564	0	74.014	0	74.007	0.6	-0.01
C	3	0	3	0	3	0	0	0
D	5	0	5	0	5	0	0	0

Task Name	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	33.092	0	33.753	0	31.443	0	31.466
B	0	73.564	0	74.014	0	71.370	0	71.419
C	3	0	3	0	3	0	3	0
D	5	0	5	0	5	0	5	0

Table 2: Throughput Results for Replication Example (Case 1)

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	%Error	
				Exact MVA	**Schweitzer**
A	0.0302184	0.0296272	0.0296307	-1.6	0.01
B	0.0135935	0.0135109	0.0135120	-0.6	0.01
C	0.2024350	0.1995740	0.1995950	-1.4	0.01
D	0.0543741	0.0540435	0.0540480	-0.6	0.01

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.0302184	0.0296272	0.0318026	0.0317800
B	0.0135935	0.0135109	0.0140113	0.0140016
C	0.2024350	0.1995740	0.2112780	0.2111300
D	0.0543741	0.0540435	0.0560451	0.0560064

Table 3: Task Utilization Results for Replication Example (Case 1)

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Schweitzer)		%Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Exact MVA	Schw
A	0	1	0	1	0	0.9999	0	0
B	0	1	0	1	0	0.9999	0	0
C	0.60730	0	0.59872	0	0.5987	0	-1.4	0
D	0.27187	0	0.27021	0	0.2702	0	-0.6	0

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	1	0	1	0	0.999	0	0.999
B	0	1	0	1	0	0.999	0	0.999
C	0.60730	0	0.59872	0	0.63383	0	0.63338	0
D	0.27187	0	0.27021	0	0.28022	0	0.28003	0

Table 4: Processor Utilization Results for Replication Example (Case 1)

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	%Error	
				Exact MVA	Schweitzer
A	0.0604367	0.0592544	0.0592613	-2	0.01
B	0.0543741	0.0540435	0.0540481	-0.6	0.01
C	0.6073040	0.5987220	0.5987850	-1.4	0.01
D	0.2718700	0.2702180	0.2702400	-0.6	0.01

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.0604367	0.0592544	0.0636053	0.0635599
B	0.0543741	0.0540435	0.0560452	0.0560066
C	0.6073040	0.5987220	0.6338350	0.6333890
D	0.2718700	0.2702180	0.2802260	0.2800320

The replication algorithm may be used to solve several different configurations of SRVN models. It can handle a model with a mix of replicated and non-replicated tasks,

models with multiple entries, and multi-layered models. An example of a mixed model, Case 2, is shown in Figure 47 and Figure 48 with the results of solving the system shown in Tables 5-8. A model with multiple entries, Case 3, (Figure 49) and a model with multiple layers, Case 4, (Figure 50) were also solved with the results shown in Tables 9-16. Finally, the example of Figure 36, Case 5, which has twenty B tasks and does not converge using the Gauss-Seidel iteration, is solved using the simplified Newton-Raphson method. It converges using the Newton-Raphson option but diverges otherwise.

As with Case 1, the Schweitzer approximation gives the closest results to the full system. The multi-layered example, Case 4, produces higher errors than the single-layered examples since the errors encountered at each submodel layer tend to propagate to the other submodel layers. Despite the errors, the advantages of the replication are obvious when comparing the calculation time, the ease of reading the results, and the simplification of the input (input file format) between the full model and the replicated model. The summary of results for the replicated system is much more compact than for the streams of data for the full system. But mainly it can be seen that the SRVN model with its input file are simple for the replication model as compared to the full model. If more replicas are added, the change to the replicated model is trivial but is quite cumbersome for the full model.

Since the Schweitzer MVA approximation gives the best results, the space and time complexity relative to this algorithm is discussed. The space requirements for Schweitzer is proportional to the product of the number of chains, *C*, and the number of stations, *N*, i.e. *O(CN)*. The time requirement per iteration of the algorithm is also proportional to this product. The replication algorithm reduces the number of chains and the number of stations needed, thereby reducing the space and time requirement for each Schweitzer iteration by $O(\sum_{m=1}^{M}(K_m - 1))$, where K_m is the number of replicas at server *m* and *M* is the total number

of replicated task sets ($N = \sum_{m=1}^{M} K_m$). The replication iteration introduced for solving each submodel increases the time by an unknown factor. Finally, the time complexity for one iteration of the LQNS inter-layer submodel solution (i.e. the "outer iteration" mentioned in Section 5.7) is $O(LN^2)$ or $O(L(\sum_{m=1}^{M} K_m)^2)$, where L is the number of layers [Rolia 92]. (This is derived from the time complexity of one iteration of Schweitzer which is $O(CN)$.) Since the replication algorithm reduces the number of stations by representing a set of replicas by one station, the time complexity of one LQNS iteration is reduced to $O(LM^2)$, or by a factor of $\left(\frac{N}{M}\right)^2$.

In executing the test cases, the times for solving the replicated models were found to be shorter than for the full models. All test cases were run on an HP 9000/735 workstation. For Case 5, with around 20 tasks, the time difference was especially visible with the replication model taking less than a second to solve in contrast to the full model which took around 9 minutes.

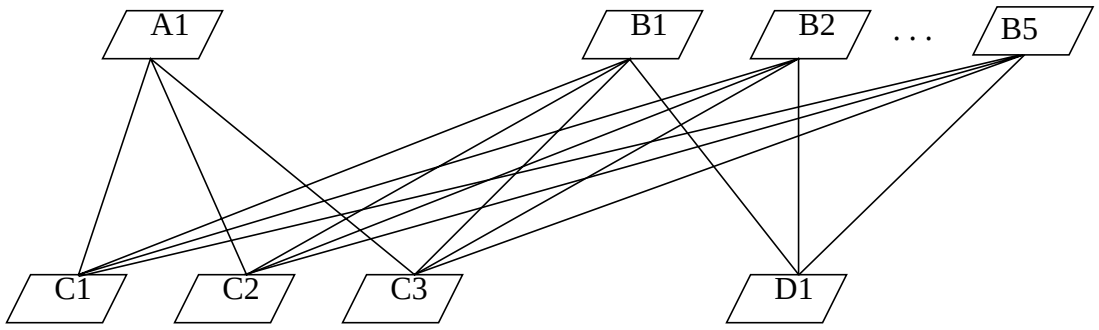


Figure 47: Full Model for Mixed System (Case 2)

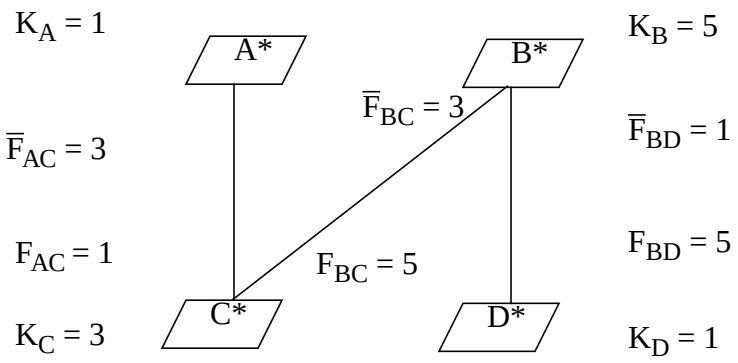


Figure 48: Replicated Model for Mixed System (Case 2)

Table 5: Cycle Time Results for Mixed System (Case 2)

Task Name	**Full Model (Exact MVA)**		**Full Model (Schweitzer)**		**Replicated (Schweitzer)**		**% Error**	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	**Exact MVA**	**Schw**
A	0	63.108	0	65.221	0	65.205	3	-0.02
B	0	188.59	0	188.98	0	188.92	0.2	-0.03
C	3	0	3	0	3	0	0	0
D	5	0	5	0	5	0	0	0

Task Name	**Full Model (Exact MVA)**		**Full Model (Schweitzer)**		**Replicated (Linearizer)**		**Replicated (Exact MVA)**	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	63.108	0	65.221	0	59.387	0	59.704
B	0	188.59	0	188.98	0	176.26	0	176.93
C	3	0	3	0	3	0	3	0
D	5	0	5	0	5	0	5	0

Table 6: Throughput Results for Mixed System (Case 2)

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	%Error	
				Exact MVA	Schw
A	0.01584580	0.0153323	0.01533630	-3	0.02
B	0.00530251	0.0052914	0.00529328	-0.1	0.03
C	0.23312500	0.2311960	0.23127400	-0.8	0.03
D	0.10605000	0.1058280	0.10586600	-0.2	0.03

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.01584580	0.0153323	0.01683880	0.01674930
B	0.00530251	0.0052914	0.00567356	0.00565196
C	0.23312500	0.2311960	0.24909100	0.24806700
D	0.10605000	0.1058280	0.11347100	0.11303900

Table 7: Task Utilization Results for Mixed System (Case 2)

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Schweitzer)		% Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Exact MVA	Schw
A	0	0.99999	0	0.999999	0	1	0	0
B	0	0.99999	0	0.999999	0	1	0	0
C	0.69938	0	0.69359	0	0.6938	0	-0.7	0.04
D	0.53025	0	0.52914	0	0.5293	0	-0.2	0.03

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	0.99999	0	0.999999	0	1	0	1
B	0	0.99999	0	0.999999	0	1	0	1
C	0.69938	0	0.69359	0	0.74727	0	0.74420	0
D	0.53025	0	0.52914	0	0.56736	0	0.56520	0

Table 8: Processor Utilization Results for Mixed System (Case 2)

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	% Error	
				Exact MVA	Schweitzer
A	0.0410406	0.0397106	0.0397210	-3	0.02
B	0.0243915	0.0243404	0.0243491	-0.2	0.03
C	0.6993750	0.6935880	0.6938210	-0.8	0.04
D	0.5302510	0.5291400	0.5293280	-0.2	0.03

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.0410406	0.0397106	0.0436125	0.0433807
B	0.0243915	0.0243404	0.0260984	0.0259990
C	0.6993750	0.6935880	0.7472730	0.7442000
D	0.5302510	0.5291400	0.5673560	0.5651960

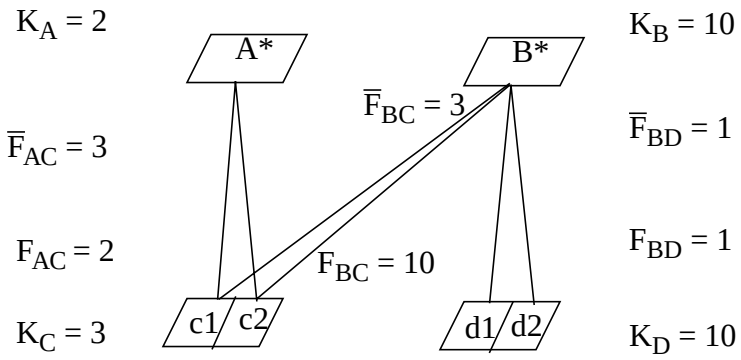


Figure 49: Replicated Model of Multi-Entried System (Case 3)

Table 9: Cycle Time Results for Multi-Entried System (Case 3)

Entry Name	Full (Exact MVA)		Full (Schweitzer)		Replicated (Schweitzer)		% Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Exact MVA	Schw
A	0	148.45	0	150.16	0	150.14	1	-0.01
B	0	267.03	0	268.96	0	268.92	0.7	-0.01
c1	3	0	3	0	3	0	0	0
c2	3	0	3	0	3	0	0	0
d1	5	0	5	0	5	0	0	0
d2	5	0	5	0	5	0	0	0

Entry Name	Full (Exact MVA)		Full (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	148.45	0	150.16	0	133.63	0	135.50
B	0	267.03	0	268.96	0	248.05	0	250.42
c1	3	0	3	0	3	0	3	0
c2	3	0	3	0	3	0	3	0
d1	5	0	5	0	5	0	5	0
d2	5	0	5	0	5	0	5	0

Table 10: Throughput Results for Multi-Entried System (Case 3)

Entry Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	% Error	
				Exact MVA	Schweitzer
A	0.00673588	0.00665915	0.00666024	-1	0.02
B	0.00374483	0.00371797	0.00371856	-0.7	0.03
c1	0.13928900	0.13817600	0.13819800	-0.7	0.01
c2	0.13928900	0.13817600	0.13819800	-0.7	0.01
d1	0.01497930	0.01487190	0.01487420	-0.7	0.02
d2	0.01497930	0.01487190	0.01487420	-0.7	0.02

Entry Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.00673588	0.00665915	0.00748333	0.00737994
B	0.00374483	0.00371797	0.00403132	0.00399326
c1	0.13928900	0.13817600	0.15087300	0.1493180
c2	0.13928900	0.13817600	0.15087300	0.1493180
d1	0.01497930	0.01487190	0.01612530	0.0159730
d2	0.01497930	0.01487190	0.01612530	0.0159730

Table 11: Task Utilization Results for Multi-Entried System (Case 3)

Entry Name	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Schweitzer)		% Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	**Exact MVA**	**Schw**
A	0	0.99999	0	0.999999	0	1	0	0
B	0	0.99999	0	0.999999	0	1	0	0
c1	0.41786	0	0.41453	0	0.4146	0	-0.7	0.02
c2	0.41787	0	0.41453	0	0.4146	0	-0.7	0.02
d1	0.07490	0	0.07436	0	0.0744	0	-0.7	0.05
d2	0.07490	0	0.07436	0	0.0744	0	-0.7	0.05

Entry Name	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	0.99999	0	0.999999	0	1	0	1
B	0	0.99999	0	0.999999	0	1	0	1
c1	0.41786	0	0.41453	0	0.45262	0	0.44795	0
c2	0.41787	0	0.41453	0	0.45262	0	0.44795	0
d1	0.07490	0	0.07436	0	0.08063		0.07987	
d2	0.07490	0	0.07436	0	0.08063		0.07987	

Table 12: Processor Utilization Results for Multi-Entried System (Case 3)

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	% Error	
				Exact MVA	Schweitzer
A	0.0134718	0.0133183	0.0133205	-1	0.07
B	0.0149793	0.0148719	0.0148742	-0.6	0.02
C	0.8357310	0.8290540	0.8291860	-0.7	0.01
D	0.1497930	0.1487190	0.1487420	-0.6	0.02

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.0134718	0.0133183	0.0149667	0.0147599
B	0.0149793	0.0148719	0.0161253	0.0159730
C	0.8357310	0.8290540	0.9052380	0.8959050
D	0.1497930	0.1487190	0.1612530	0.1597300

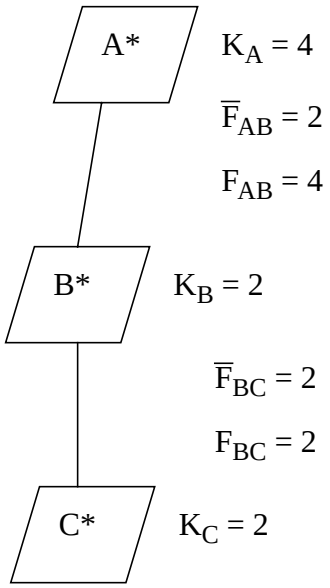


Figure 50: Replicated Model of Multi-Layered System (Case 4)

Table 13: Cycle Time Results for Multi-Layered System (Case 4)

Entry Name	Full (Exact MVA)		Full (Schweitzer)		Replicated (Schweitzer)		% Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	**Exact MVA**	**Schw**
A	0	17.805	0	18.225	0	18.224	2	-0.01
B	3.5071	0	3.5625	0	3.5627	0	1.6	-0.01
C	1	0	1	0	1	0	0	0

Entry Name	Full (Exact MVA)		Full (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	17.805	0	18.225	0	16.525	0	16.529
B	3.5071	0	3.5625	0	3.5210	0	3.5162	0
C	1	0	1	0	1	0	1	0

Table 14: Throughput Results for Multi-Layered System (Case 4)

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	% Error	
				Exact MVA	**Schweitzer**
A	0.0561653	0.0548689	0.0548741	-2	0.02
B	0.2246610	0.2194760	0.2194960	-2	0.02
C	0.4493220	0.4389510	0.4389930	-2	0.01

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.0561653	0.0548689	0.0605136	0.0604992
B	0.2246610	0.2194760	0.2420540	0.2419970
C	0.4493220	0.4389510	0.4841090	0.4839930

Table 15: Task Utilization Results for Multi-Layered System (Case 4)

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Schweitzer)		% Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	**Exact MVA**	**Schw**
A	0	1	0	1		1	0	0
B	0.78791	0	0.7819	0	0.7820	0	-0.7	0.01
C	0.44932	0	0.4389	0	0.4390	0	-2	0.02

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	1	0	1	0	1	0	1
B	0.78791	0	0.7819	0	0.85228	0	0.85092	0
C	0.44932	0	0.4389	0	0.48411	0	0.48399	0

Table 16: Processor Utilization Results for Multi-Layered System (Case 4)

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	% Error	
				Exact MVA	**Schweitzer**
A	0.0561653	0.0548689	0.0548741	-2	0.02
B	0.2246610	0.2194760	0.2194960	-2	0.01
C	0.4493220	0.4389510	0.4389930	-2	0.01

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.0561653	0.0548689	0.0605136	0.0604992
B	0.2246610	0.2194760	0.2420540	0.2419970
C	0.4493220	0.4389510	0.4841090	0.4839930

Table 17: Cycle Time Results Using Newton-Raphson (Case 5)

Entry Name	Full (Exact MVA)		Full (Schweitzer)		Replicated (Schweitzer)		% Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Exact MVA	Schw
A	0	132.23	0	132.89	0	132.82	0.4	-0.05
B	0	220.32	0	221.17	0	221.06	0.3	-0.04
C	3	0	3	0	3	0	0	0
D	5	0	5	0	5	0	0	0

Entry Name	Full (Exact MVA)		Full (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	132.23	0	132.89	0	121.90	0	122.38
B	0	220.32	0	221.17	0	206.36	0	207.01
C	3	0	3	0	3	0	3	0
D	5	0	5	0	5	0	5	0

Table 18: Throughput Results for Using Newton-Raphson (Case 5)

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	% Error	
				Exact MVA	Schweitzer
A	0.00756260	0.00752531	0.00752928	-0.4	0.05
B	0.00453883	0.00452144	0.00452359	-0.3	0.05
C	0.30258000	0.30138700	0.30153200	-0.3	0.05
D	0.01815530	0.01808570	0.01809430	-0.3	0.05

Task Name	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.00756260	0.00752531	0.00820354	0.00817116
B	0.00453883	0.00452144	0.00484581	0.00483068
C	0.30258000	0.30138700	0.32356300	0.32252500
D	0.01815530	0.01808570	0.01938320	0.01932270

Table 19: Task Utilization Results for Using Newton-Raphson (Case 5)

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Schweitzer)		% Error	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Exact MVA	Schw
A	0	0.999999	0	0.999999	0	1	0	0
B	0	0.999999	0	0.999999	0	1	0	0
C	0.90774	0	0.90416	0	0.90460	0	-0.3	0.05
D	0.09078	0	0.09043	0	0.09047	0	-0.3	0.04

Task	Full Model (Exact MVA)		Full Model (Schweitzer)		Replicated (Linearizer)		Replicated (Exact MVA)	
	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2	Phase 1	Phase 2
A	0	0.999999	0	0.999999	0	1	0	1
B	0	0.999999	0	0.999999	0	1	0	1
C	0.90774	0	0.90416	0	0.97069	0	0.96758	0
D	0.09078	0	0.09043	0	0.09692	0	0.09661	0

Table 20: Processor Utilization Results for Using Newton-Raphson (Case 5)

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Schweitzer)	% Error	
				Exact MVA	Schweitzer
A	0.0151252	0.0150506	0.0150586	-0.5	0.05
B	0.0181553	0.0180857	0.0180943	-0.3	0.05
C	0.9077410	0.9041620	0.9045970	-0.3	0.04
D	0.0907766	0.0904287	0.0904717	-0.3	0.05

Processor	Full Model (Exact MVA)	Full Model (Schweitzer)	Replicated (Linearizer)	Replicated (Exact MVA)
A	0.0151252	0.0150506	0.0164071	0.0163423
B	0.0181553	0.0180857	0.0193832	0.0193227
C	0.9077410	0.9041620	0.9706880	0.9675760
D	0.0907766	0.0904287	0.0969161	0.0966135

5.9 Limitations

From the analysis of the results in Section 5.8, it can be seen that the replication algorithm should use the Schweitzer approximation for the best results. Other limitations exist which may be removed in future work.

The current implementation of the replication algorithm can handle multiple entries for each task. However, the entries of a task must have the same fan-out and fan-in values. It may be noted that the usefulness of having entries with different fan-out and fan-in values is questionable. In other words, what kind of system would need this modeling capability keeping in mind that the symmetry of the system must be maintained.

Another limitation of the current implementation also deals with the symmetry issue. The processors associated with the tasks in the SRVN model are treated as servers. (The processors are not shown explicitly but are specified in the model in the input file.) Therefore, they have exactly the same restrictions as tasks. That is, a replicated group of tasks must be allocated either to one processor or to a replicated group of processors.

Finally, convergence of the algorithm is not guaranteed. The simplified Newton-Raphson method may be applied when the Gauss-Seidel iteration fails to converge. However, it too may not converge in some cases.

Chapter 6.0 Case Study

6.1 Introduction

As the trend towards distributed computing continues, many systems in industry have been converted to a client-server architecture with some being multi-tiered. Multi-tiered systems are systems with server entities that act as servers to some entities and as clients to others. Some systems may contain a large number of components especially if the system spans geographic boundaries. The importance of performance analysis of these systems becomes apparent and, in some cases, critical as the configurations of these systems evolve. Before any changes are made to the system, the performance of the system must be predicted to avoid any degradation or total inoperability of the system. Capacity planning studies are conducted which include performance evaluation techniques. Performance evaluation consists of defining the goals of a performance study, creating a performance model for the system, deciding on the performance metrics, measuring or estimating the performance model parameters, selecting and modeling the workload, solving the performance model with the appropriate evaluation techniques, and reporting the results.

The Layered Queueing Network Solver (LQNS) with replication modifications, as discussed in Chapters 4 and 5, may be used to predict system performance of large multi-tiered client-server systems. In this chapter, a simple study of a real industrial system is conducted to demonstrate a practical application of the LQNS tool with replication.

6.2 Capacity Planning for a Large Client-Server System

To demonstrate the solving of replicated systems, a study of a large industrial client-server system is presented. The study is based on data from the capacity planning study

conducted by Shen [Shen 96]. The structure of the system is shown in Figure 51 on page 101. The main function of the network is to provide database access to the users of the system. The large database system consists of thousands of workstations (or PC's) submitting transactions to two databases, called RF and BC, stored on two separate mainframes. The workstations are connected to a local area network (LAN) and access the databases via a LAN server (PC or workstation). The LAN server also provides some local service to the workstations. Some of the LAN's are connected to the backbone network (FDDI (Fiber Distributed Data Interface)) and others are connected to a wide area network (WAN). There may be other traffic on the network besides the database transactions.

To simplify the modeling of the database system, some assumptions are made. All the LAN's are assumed to be token ring with the same traffic factor and workload. That is, the number of workstations attached to each LAN is the same, 10. The number of LAN's attached to the WAN is the same as the number of LAN's attached to the backbone. The workstations are modeled as equivalent entities producing similar workloads. With the simplifications, the database system has symmetric properties and can be easily modeled into an SRVN model using the simplified notation as shown in Figure 52 on page 102. The workstations, or more appropriately the tasks running on the workstations, are modeled as client tasks. The LAN server is modeled as a FCFS (first-come first-served) server task. In fact, it is also a client to the database servers. The two databases are modeled as FCFS server tasks in the bottom layer. The LAN is a token ring and its delay has been modeled as a delay server. Similarly, the WAN and backbone delays have been modeled as delay servers. For purposes of this study, which is concerned with the performance of the whole system rather than individual components of the system, the front-end processor

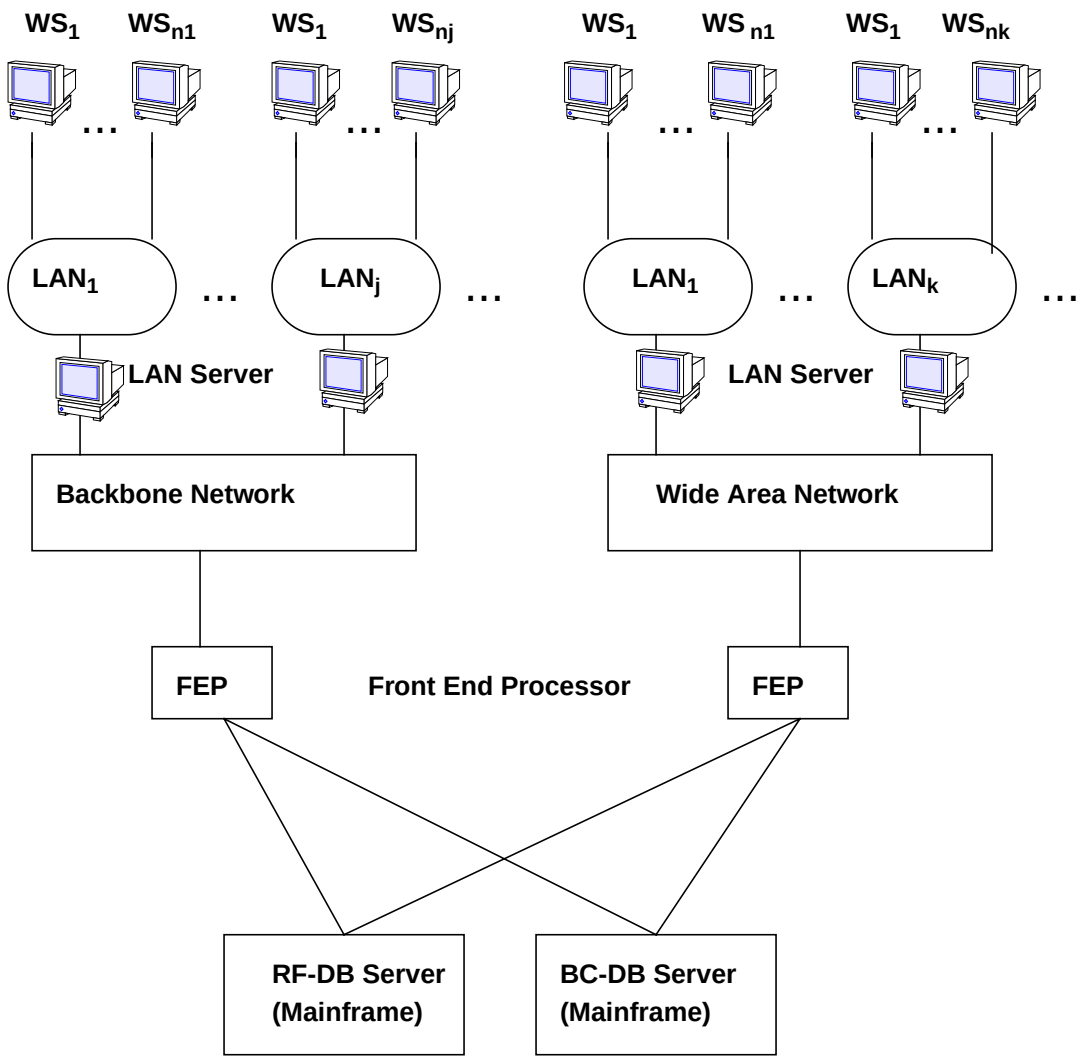
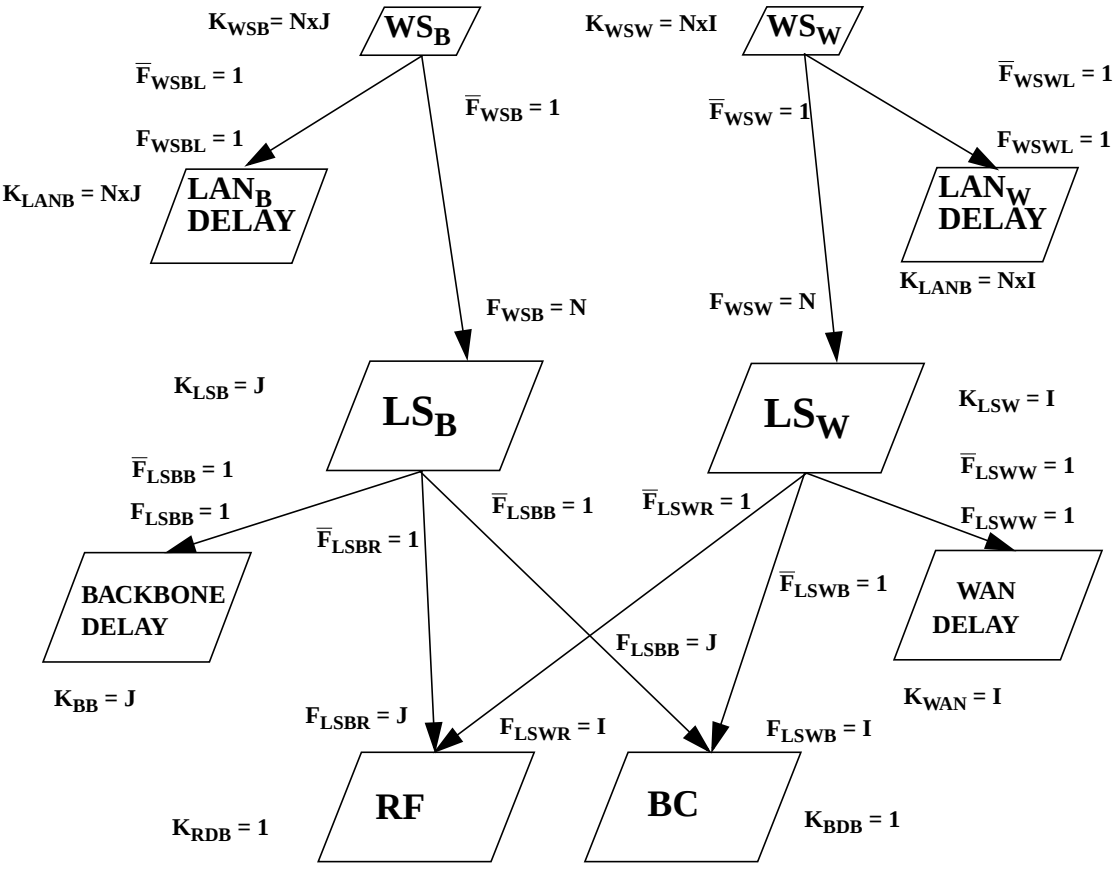


Figure 51: Database System



WS - Workstation
LS - LAN Server
I - Number of LANs on WAN
J - Number of LANs on Backbone
N - Number of Workstations per LAN

Figure 52: SRVN Model of Database System

and disks are not of great importance and have not been modeled. Their effects have been incorporated into the parameters of the database server tasks.

A characterization of the workload and the parameters of the model, such as the service times of each task and the number of visits between tasks, must be obtained to use the model for performance prediction. Both may be obtained from measurements and observations of the actual system in operation. From measurements collected on the response time of different transactions at the workstations, the workload is characterized into four types: heavy transactions to the RF database, light transactions to the RF database, heavy transactions to the BC database, and light transactions to the BC database [Shen 96]. These workload types have been represented in the SRVN model by different entries in the database tasks. The workload that only requires the LAN server is represented as an entry in the LAN server task. From the measurements, the think time at the workstations was determined to be 15 seconds. However, since other measurement data was scarce, the rest of the parameters for the model are very rough educated estimates. The following parameters are used:

Service Time of LAN Server for Local Service	= 0.03 seconds
Service Time of LAN Server for Database Service	= 0.01 seconds
Service Time of RF Database Light Transactions	= 0.04 seconds
Service Time of RF Database Heavy Transactions	= 0.08 seconds
Service Time of BC Database Light Transactions	= 0.08 seconds
Service Time of BC Database Heavy Transactions	= 0.16 seconds
Service Time of LAN delay	= 0.00039 seconds
Service Time of WAN delay	= 0.04 seconds
Service Time of Backbone delay	= 0.02 seconds

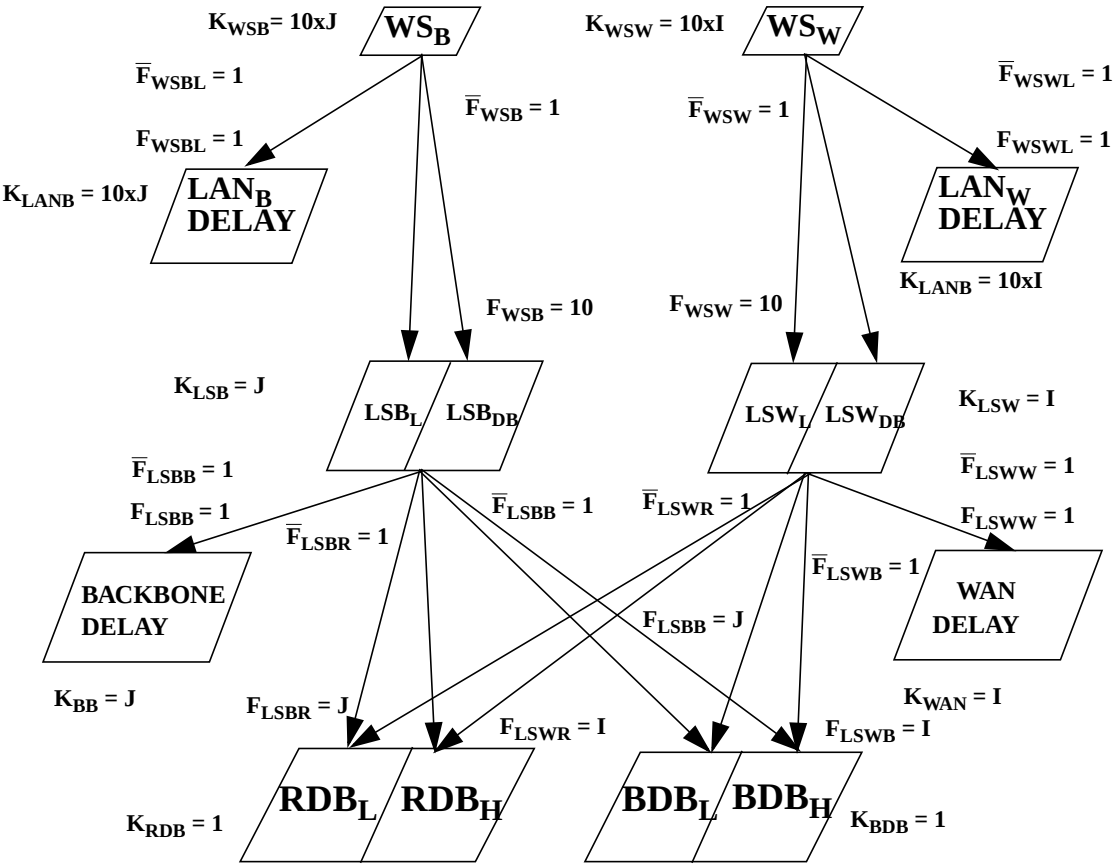
The mainframe for the BC database also receives work from other non-database applications and takes up 50% of its CPU time. The service time given above is the effective service time which is the actual service time divided by 0.5 (effective service time = service time/(1 - utilization of non-database applications)). The LAN, WAN, and backbone delays have been calculated with the equations in Shen's thesis (Equation 12 of [Shen 96]) with the same parameter assumptions and traffic factors. The WAN rate is assumed to be 50,000 bits/second and the backbone FDDI rate is assumed to be $1x10^8$ bits/second.

The visits between tasks are estimated from observations of the system. There are two visits to the LAN, WAN, and backbone servers. One delay is encountered when requesting a service and one delay is encountered for the response. It is assumed that the workstation on the average makes two requests to the LAN server for database service for every one request that only involves the LAN server. Therefore, the visit ratio is 2/3 to entry LS_{DB} and 1/3 for entry LS_L. Similarly, it is estimated that there are 5 requests to the RF database for every request to the BC database. The request for a light or heavy transaction is equal. Therefore, the visit ratio to the BC light entry or heavy entry is (1/6)(1/2)=1/12. The visit ratio to the RF database is in total 5/6.

The completed SRVN model, as shown in Figure 53, may be used to study the performance of the system under varying configurations and parameters. In the first analysis, the change in the response time at the workstations is studied when the number of workstations in the system is increased. There are ten workstations attached to each LAN. The number of workstations is increased by attaching new LAN's to the network. The SRVN model is solved several times by varying the number of LAN servers. Figure 55 shows a graph of the change in response time as seen at the client workstation. As expected, the response time increases with the number of workstations. The response time increases dramatically at more than 300 clients since the RF database saturates between 300 and 400

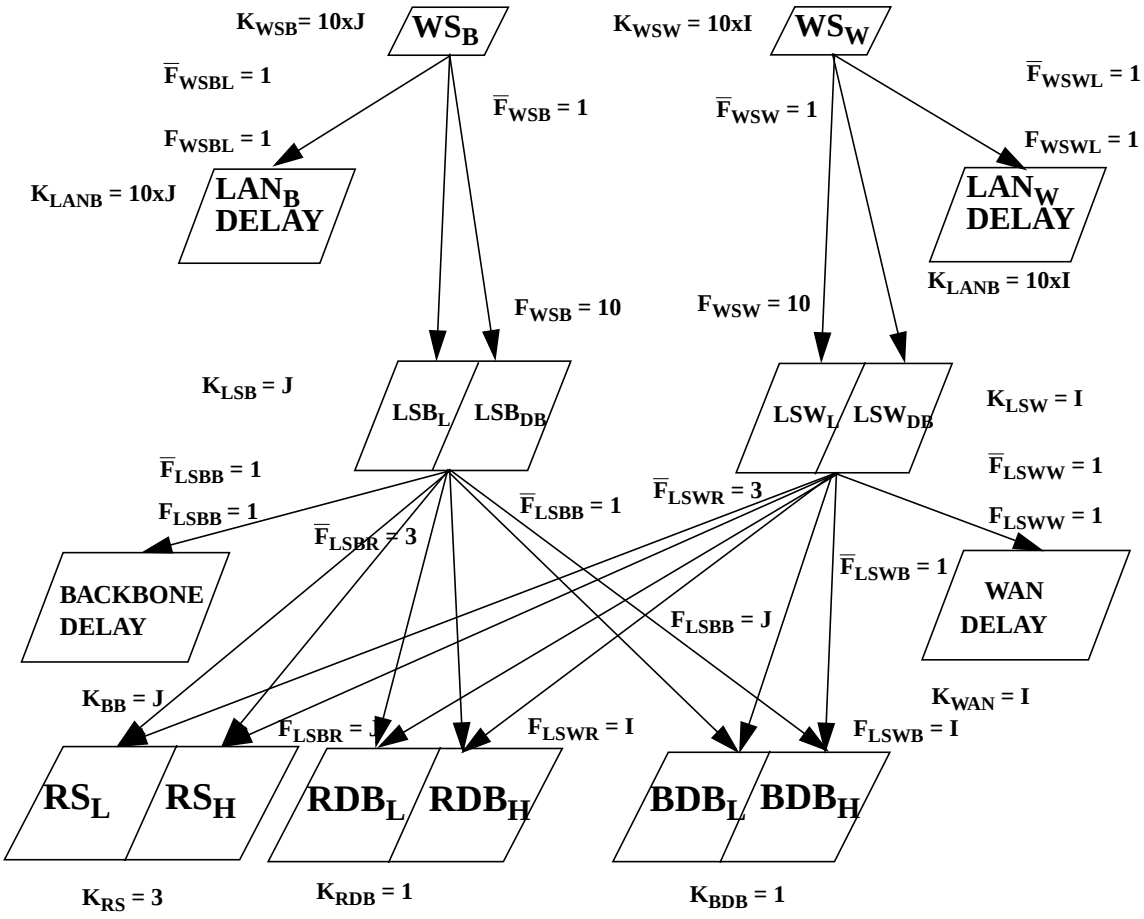
clients. The RF database is the first component to saturate and is the bottleneck of the system. After saturation, the response time continues to increase linearly.

The second analysis seeks to determine the effect of regional servers to off-load the work at the RF database. The regional servers are given the same parameters and entries as the RF database. The visit ratios to the regional servers and RF database are determined by the fraction of RF database requests that are routed to the regional servers. The SRVN model with three regional servers is shown in Figure 54. Figure 55 shows the increase in the response time seen at the client with 20% of the RF database traffic routed to the regional servers. Clearly, the response time at the client is improved and the RF database saturates between 400 and 500 clients. Finally, the effect of changing the fraction of traffic going to the regional servers is studied. Figure 56 shows the response time at the client for a system with 600 workstations when the fraction of off-loaded traffic is varied. The response time decreases with the increase in the fraction of RF database traffic going to the regional servers.



WS - Workstation (10 attached to each LAN)
LSB_L - LAN Server Entry for Local Service
LSB_{DB} - LAN Server Entry for Database Access
RDB_L - Light Transactions for Remote Forms Database
RDB_H - Heavy Transactions for Remote Forms Database
BDB_L - Light Transactions for Billing and Collections Database
BDB_H - Heavy Transactions for Billing and Collection Database
I - Number of LANs on WAN
J - Number of LANs on Backbone

Figure 53: SRVN Model with Entries



WS - Workstation (10 attached to each LAN)
LSB_L - LAN Server Entry for Local Service
LSB_{DB} - LAN Server Entry for Database Access
RDB_L - Light Transactions for Remote Forms Database
RDB_H - Heavy Transactions for Remote Forms Database
BDB_L - Light Transactions for Billing and Collections Database
BDB_H - Heavy Transaction for Billing and Collection Database
RS_L - Light Transaction at Regional Server
RS_H - Heavy Transaction at Regional Server
I - Number of LANs on WAN
J - Number of LANs on Backbone

Figure 54: SRVN Model of Database System with Three Regional Servers

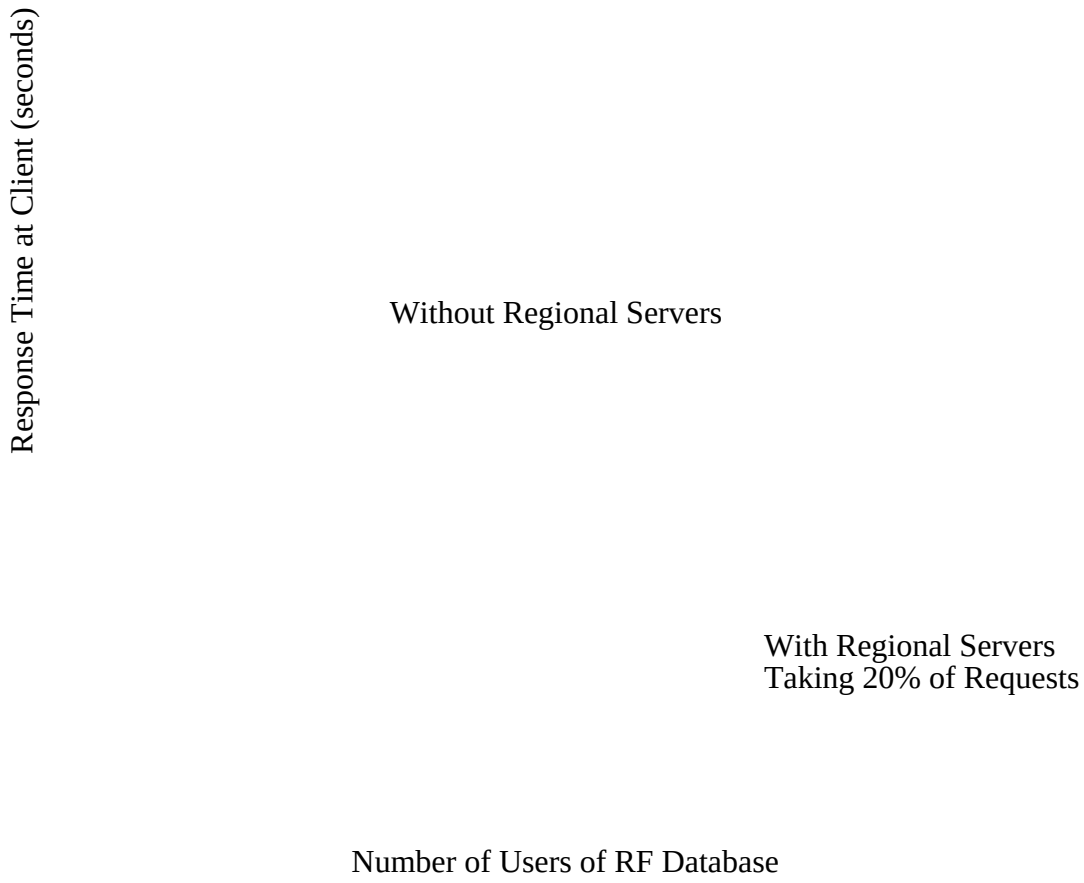


Figure 55: Performance of Database System

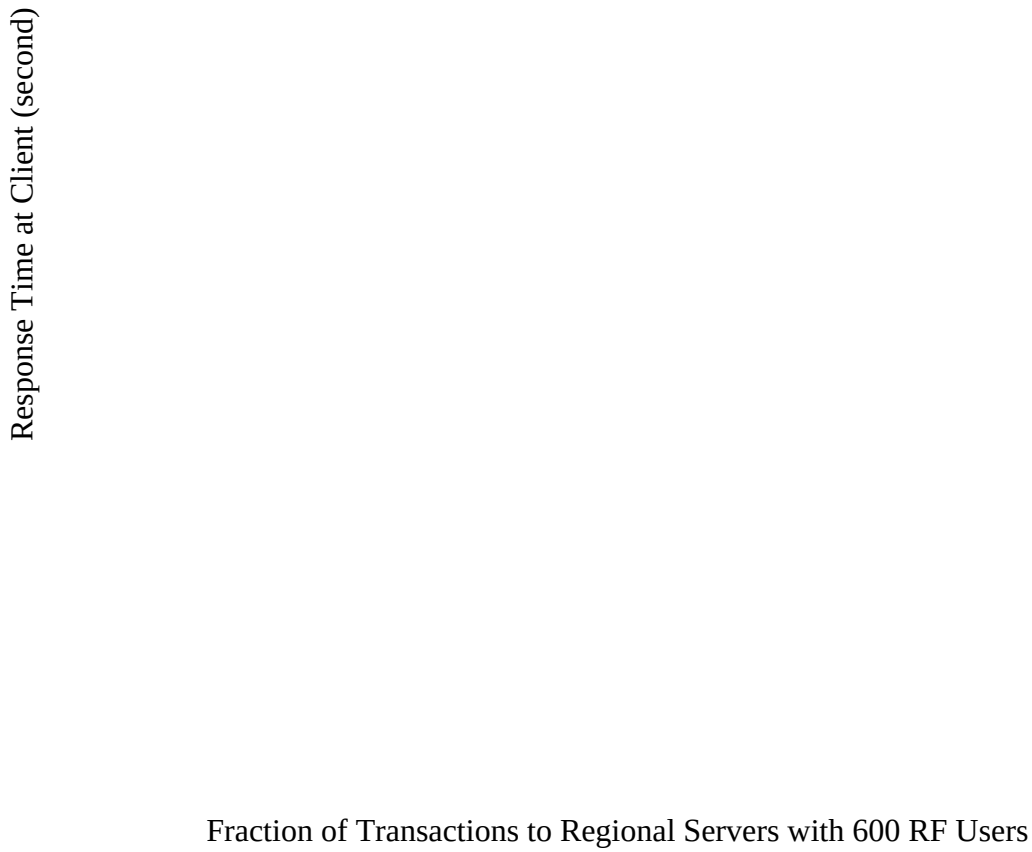


Figure 56: Effect on Performance of Off-Loading to Regional Servers

Chapter 7.0 Conclusion

7.1 Research Summary

As distributed computer systems, such as client-server systems, increase in size and complexity, the need to evaluate the performance of these systems becomes critical. There often exist components in these systems which are similar from a performance modeling point of view. These components, referred to as being replicated, have the same performance parameters. Consequently, their performance measures predicted from the performance models are also equal.

This thesis has presented a method that takes advantage of the replication of components to simplify the modeling of large systems and to solve the performance models for these systems. The method modifies the Stochastic Rendezvous Network (SRVN) and Layered Queueing Network Solver (LQNS) performance analysis toolset to enable the toolset to analyze large systems in an efficient manner.

One of the big advantages, presented in the thesis, is a simplified notation representing large systems with replicated component. A set of replicated components is represented by one entity with a notation representing the number of replicas. The connection between replicated components is also represented once with a fan-in and fan-out notation. The proposed representation thereby reduces the number of entities and connections needed to define a large system. The graphical model is simplified as well as the text description file of the model. The results of solving the SRVN model are also compacted since the performance measures for one entity is presented instead of a repetition of the same results for replicated components.

The LQNS, which solves the SRVN model, has been modified to enable the solving of large models with any combination of replicated and non-replicated components. The underlying MVA solver for queueing submodels, which is in LQNS, has not been touched. Instead, the queueing network presented to the MVA solver is modified to take full advantage of the replication. The chains for replicated components are assigned with respect to the server rather than with respect to the client as in non-replicated components. Each chain visits one client and one server for replicated components. A set of replicated components is represented by one station in the queueing network. The method of surrogate delays is employed to account for the delays of the non-represented replicated components. The method is implemented in the form of modifying the service time of the client entity given in equation (5.4) in Section 5.4.

The question of convergence of the method which is basically a Gauss-Seidel iteration has been addressed. There were some difficulties getting the iterative solution to converge for some cases. A simplified Newton-Raphson method was applied instead of the Gauss-Seidel iteration in cases where the latter fails to converge. The new update equation for the service time is equation (5.22) in Sect 5.6. This was effective in all cases that were tested. Unfortunately, convergence is not theoretically guaranteed even with the Newton-Raphson method.

Several models with replicated components were solved using the full representation and the simplified representation. The results were compared. It is evident that using the Schweitzer MVA approximation to solve the queueing submodels, with the replication method, provides the best results. The reason is found in the nature of the method of surrogate delays where the service time of the delay centre representing the ‘missing’ station is obtained for a full population $\overline{N}$. The Schweitzer method uses the estimated times for population $\overline{N}$ whereas the exact MVA and Linearizer require values for other

populations beside $\overline{N}$. Thus, an error is introduced. This source of error has not been identified in other research based on surrogate delays. The results were excellent for single-layered models but are less accurate for multi-layered models. The reason is the errors produced in the solving of one layer submodel propagates to the solution of other layer submodels.

It is hoped that the work carried out in this thesis may be beneficial to performance engineers studying large systems with replicated components, in particular distributed client-server systems. The replicated method was applied to a real industrial system to illustrate its use. The strategy presented enables large systems to be analyzed using a simple model definition and a solver (LQNS) that provides relatively quick and accurate results.

7.2 Future Work

Further work may be done to eliminate the limitations of the replication method as described in Section 5.9. The most important research is studying the convergence of the replication method. The simplified Newton-Raphson method implemented does not guarantee convergence. The full Newton-Raphson method may be implemented but requires a great deal of work per iteration. The calculation would require matrix manipluation for solving a set of equations. Even the full Newton-Raphson method does not guarantee convergence. Other methods of convergence could be considered such as those suggested in [Zahor 88], where the mean queue length is used as the metric by which errors are measured, [Bard 81], and the standard numerical methods described in [Davis 86] and [Pearson 86].

REFERENCES

[Bard 81] Y. Bard. “A Simple Approach to System Modeling.” North Holland, *Performance Evaluation*, 1(1981) 225-248.

[Chandy 78] K. Mani Chandy and C. H. Sauer. “Approximate Methods for Analyzing Queueing Network Models of Computing Systems.” *Computing Surveys*, Vol. 10, No. 3, Sept. 1978.

[Chandy 82] K. Mani Chandy and Doug Neuse. “Linearizer: A heuristic algorithm for queueing network models of computing systems.” *CACM*, 25(2):126–134, February 1982.

[Chow 83] W. Chow. “Approximations for Large Scale Closed Queueing Networks.” North Holland, *Performance Evaluation*, 3 (1983) 1-12.

[Davis 86] G. de Vahl Davis. *Numerical Methods in Engineering and Science*. Allen Unwin (Publishers) Ltd, London, 1986.

[Eager 84] D. Eager, K. Sevcik. “An Analysis of an Approximation Algorithm for Queueing Networks.” North Holland, *Performance Evaluation*, 4 (1984) 275-284.

[Franks 94] G. Franks. “Layered Queueing Network Solver Software Design.” Department of Systems and Computer Engineering, Carleton University, 1994.

[Hubbard 95] A. Hubbard, G. Franks, S. Majumdar, J. Neilson, D. Petriu, J. Rolia, C. M. Woodside. “A Toolset for Performance Engineering and Software Design of Client-Server Systems.” North Holland, *Performance Evaluation*, 24 (1995) 117-136.

[Jacob 82] P.A. Jacobson and E.D. Lazowska. “Analyzing queueing networks with simultaneous resource possession.” *Communications of the ACM*, 25(2):142–151, February 1982.

[Jain 91] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, New York, New York, 1991.

[Lazow 84] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *Quantitative System Performance*. Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1984.

[Menasce 94] D. Menasce, V. Almeida, L. Dowdy. *Capacity Planning and Performance Modeling.* Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1994.

[Patti 90] Kr. R. Pattipati, M. M. Kostreva, J. L. Teele. “Approximate Mean Value

Analysis Algorithms for Queueing Networks; Existence, Uniqueness, and Convergence Results." *Computing Machinery*, Vol. 37, No. 3, July 1990, pp. 643-673.

[Pearson 86] C. E. Pearson. *Numerical Methods in Engineering and Science*. Van Nostrand Reinhold Company, Inc., New York, 1986.

[Petriu 95] D. Petriu, G. Franks, A. Hubbard. "SRVN Input File Format." Department of Systems and Computer Engineering, Carleton University, 1995.

[Rolia 92] J.A. Rolia. *Predicting the Performance of Software Systems*. PhD thesis, University of Toronto, January 1992.

[Rolia 95] J.A. Rolia and K.C. Sevcik. "The Method of Layers." *IEEE Trans. on Software Engineering*, vol. 21, no. 8, Aug. 1995.

[Rumb 91] J. Rumbaugh, M. Blaha, W. Premerlarni, F. Eddy, W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1991.

[Shen 96] Y. Shen. "A Capacity Planning Model for a Large Client-Server System." Masters Thesis, Carleton University, 1996.

[Smith 90] C. Smith. *Performance Engineering of Software Systems*. Addison-Wesley Publishing Co., New York, NY, 1990.

[Souza 84] E. deSouza e Silva, S. S. Lavenberg, R. R. Muntz. "A Perspective on Iterative Methods for Approximate Analysis of Closed Queueing Networks." North Holland, *Mathematical Computer Performance and Reliability*, 1984.

[Wood 88] C. M. Woodside, "Throughput Calculation for Basic Stochastic Rendezvous Networks." North Holland, *Performance Evaluation*, 9 (1988/89) 143-160.

[Wood 95a] C. M. Woodside, J. E. Neilson, D. C. Petriu, and S. Majumdar. "The Stochastic Rendezvous Network Model for Performance of Client-Server-Like Distributed Software." *IEEE Trans. on Computers*, vol. 44, no. 1, Jan. 1995.

[Wood 95b] C. M. Woodside. "Replicated Components in LQNS (Layered Queueing Network Solver)." Department of Systems and Computer Engineering, Carleton University, March 1995.

[Zahor 88] J. Zahorjan, D. Eager, H. Sweillam. "Accuracy, Speed, and Convergence of Approximate Mean Value Analysis." North Holland, *Performance Evaluation*, 8 (1988) 255-270.