Performance Modeling of Replication Techniques in Parallel and Distributed Layered Service Architectures

by

Tariq Al-Omari

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of the requirements of the degree of

Doctor of Philosophy

Ottawa-Carleton Institute for Electrical and Computer Engineering

Faculty of Engineering

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

May 22, 2007

Your file Votre référence
ISBN: 978-0-494-27087-5
Our file Notre référence
ISBN: 978-0-494-27087-5

Abstract

Replication is a technique that maintains and allows access to copies of data and services on multiple servers. It is usually used in multi-tier architecture systems, as found in distributed software systems, to increase performance, reliability, and availability.

The growing complexity of modern software systems, especially very large distributed systems, increases the difficulty of achieving performance objectives. A designer needs a high-level analytic tool to consistently compare the performance patterns of different replication techniques for a very large multi-tier replicated system. Analytic tools enable the designer to rapidly compute the relevant performance measures at an early stage of the design. This enables the designer to gain a competitive advantage by delivering his product in time, and at a lower cost.

The Layered Queueing Network (LQN) is the formalism selected in this thesis, as it is designed for performance modeling of software and hardware systems with a multi-tier (layered) architecture. The Layered Queueing Network Solver (LQNS) is an analytic tool used for performance analysis. This thesis describes new analytic algorithms implemented in the LQNS to compute relevant performance measures for parallel and distributed systems with replication. The contributions of the thesis are as follows. First, replication performance patterns are identified and classified. Second, a general quorum pattern is incorporated in the LQN formalism to model and compute the performance measures of a replication model with a quorum consensus protocol. Third, the accuracy of the LQNS is improved by introducing new closed-form formulas for the thread delay distribution. The

distribution is used to compute the service time of a client when the client has a quorum pattern. Finally, algorithms that exploit the symmetry in a system with replicated components are designed and implemented to hasten model solution time. Those replication components can contain internal parallelism.

The results of this research are used to study the performance of an Air Traffic Control system and an Industrial Information Management system. The LQNS solutions for these large systems are rapid, and the accuracy of the solutions is adequate for most purposes during the early stage of system modeling.

Dedication

I dedicate this thesis to my parents.

Acknowledgments

I would especially love to express my appreciation to my supervisor Dr. Greg Franks, for investing plenty of time and effort to make my thesis a success. I would like to express my gratitude to my supervisor, Dr. Murray Woodside, for support, and for generously providing good advice at all stages of my research.

There certainly exists no words that could possibly express the extent of gratitude I owe my loving parents and my caring and supportive brothers and sisters. Their kind words of advice and perpetual encouragement were essential to my success.

I am indebted to my devoted wife Bayan, and my wonderful and loving daughters Jenna and Juman. They have helped and inspired me to do the things that I only dreamed of.

I would like to thank Dr. Salem Derisavi and Hussein Al-Zubaidy for their invaluable comments and discussions during the course of my PhD study, and Dr. Susan Logie for her invaluable feedback on the final version of my thesis.

Contents

List of Figures

List of Tables

Glossary

- **BNS**: behavioral near symmetry
- **CDF**: cumulative distribution function
- **CTMC**: continuous time Markov chain
- **DTMC**: discrete time Markov chain
- **Execution demand**: the mean time a service center takes to process one job
- **FIFO**: first in first out
- **LQN**: layered queueing network
- **LQNS**: layered queueing network solver
- **MOL:** method of layers
- **MVA**: mean value analysis
- **PDF**: probability distribution function
- **RAID**: Redundant Arrays of Inexpensive Disks
- **RPC**: remote procedure call
- **PS**: processor sharing
- **RV**: random variable
- **Service time**: is the time a job waits in a queue, plus the time it takes for the server to process the job
- **SNS**: structural near symmetry
- **SRVN**: stochastic rendezvous network

- **Thread delay**: is the service time of a thread when executing on its host processor plus the service time due to making requests to remote server(s)
- **TDA**: task directed aggregation
- **TMR**: triple modular redundancy
- **UML**: unified modeling language
- f_X: probability density function of the RV X
- F_X: cumulative distribution function of the RV X
- $\mathrm{E}[X]$: expected value of the RV X
- θ_c: expected value of the RV associated with the service time of each local processing of the client c
- θ_s: expected value of the RV associated with the service time of each request to the server s
- $OS(J, \{X_i\}_{i=1}^{N})$: RV associated with the J^{th} order statistic of a statistical sample of size N, defined as the J^{th} smallest value out of N
- X_{Quorum}: RV associated with the quorum delay. X_{Quorum} is the J^{th} order statistic of the set of N thread delays $X_{Thread,1} \cdots X_{Thread,N}$
- X_{App}: RV associated with the service time of task App, which is the delay from accepting a request until App is ready to accept another
- $R_{Local,i}$: RV associated with the host processor service time for thread i
- $R_{Remote,i}$: RV associated with the blocking delay for thread i
- $S_{OHLocal}$: RV associated with the total host demand for overhanging execution
- $X_{OHLocal}$: RV associated with the local host processor service time for the activities executed by the overhanging threads after the quorum-join event, and totaled for all overhanging executions

- $X_{OHRemote}$: RV associated with the total blocking time for remote requests, for all overhanging executions
- $< K >$: a replication count K for each replicated task and processor
- O: fanout count of an arc, showing how many separate target tasks there are for each source replica
- I: fanin count for an arc, showing how many separate source tasks there are for each target replica

CHAPTER 1

Introduction

1.1. Motivation

Hardware and software systems are vital parts of the infrastructure on which information technology is built. As the demand for faster, smaller, and cheaper systems grows, the desired improvements often translate into more complicated system designs and models. The growing complexity, especially in very large distributed systems such as the Google search engine, increases the difficulty of achieving quality of service (QoS) objectives. Quality includes performance measures that are usually specified by the following **[104]**:

(1) **Service time**: the time a server takes to respond to a request, which includes queueing time and the time to process the request.

(2) **Throughput**: the number of requests that can be served by an application per time unit.

(3) **Resource utilization**: the measure of how much server and network resources are consumed by an application. Resources include CPU, memory, disk, and network.

(4) **Workload:** the total number of users, concurrent active users, data volumes, and transaction volumes.

Performance of a software application is affected by different factors, including blocking, batch processing, caching, scheduling, and contention. Figure 1.1 shows common performance factors across application layers **[104]**.

The system performance should be considered in designing, building, testing, maintaining, or managing stages of the software product development. System designers should

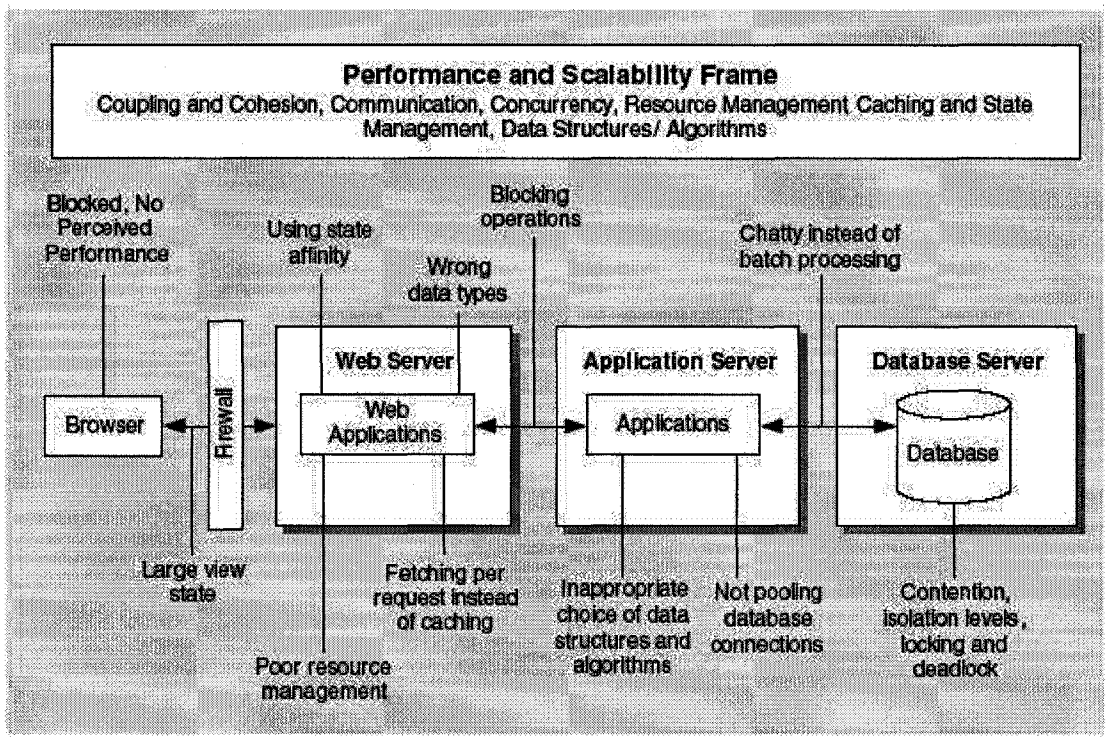


Figure 1.1: Common performance factors across application layers [**104**].

understand their performance objectives to achieve them, and to make their product a success [**104**]. Performance is especially important to the designers of large software systems for two main reasons. First, customers of these systems always desire rapid responses to their requests when using the system. Second, builders of these systems always want to build their products rapidly, and to minimize the total cost of the final product to gain a competitive advantage. These two objectives can often conflict, so designers need a reliable tool to determine the performance of their system as early as possible in the life cycle of the product.

"Software Performance Engineering (SPE) represents the entire collection of software engineering activities and related analysis used throughout the software development cycle,

which are directed to meeting performance requirements" [**168**]. To predict whether a design fulfills its performance requirements, a performance model is produced and analyzed. A performance model captures the structure, and the behavior of the system under study, and defines how system operations use the available software and hardware resources. Developing better tools for software modeling is one direction of software performance engineering [**168**]. Having rapid and cheap modeling techniques is a desired objective for performance diagnosis at an early stage of the software product development. Balsamo et al. [**12**] survey model-based approaches for the prediction of performance in the software product development stages. Smith [**149, 150**] produces a performance model at an early stage of the product life cycle and uses performance measures to change the design and architecture of the software product to fulfill performance requirements.

To improve the performance of a system, to fulfill the performance requirements, multithreading of customer processes can be used to minimize the blocking time of customers. At the server, performance can be improved in different ways. The server may reply to the client before the request is completed, thus allowing the client to resume processing. In addition, the performance can be improved by speeding up the hardware resources, reducing the demand on the system resources, or replicating the server processes. The system performance will be improved only if that server process is the bottleneck, which is the component of the system with the highest utilization. These choices may have different costs for a given system.

Replication, a technique used for improving performance, is performed by using identical (symmetrical) replicas of the system to distribute accesses to them. Moreover, replicas of services can be distributed geographically close to users to reduce access latencies. Replication usually appears in multi-tier client server systems.

Distributed software systems usually have multi-tier (or layered) client server architectures in which an application is executed by more than one distinct software component. The performance measures in multi-tier client server systems are more complex to predict than those measures in two-tier client server systems. The complexity is due to the fact that the response time of an entity depends not only on the server it communicates with directly, but also on any other underlying servers. Overall system performance may be limited by the performance of intermediate servers that could be the bottleneck. The system may be saturated even though none of the hardware devices are fully utilized, because of the possible intermediate software severs bottlenecks.

One of the broader goals of this research is to improve the performance evaluation of multi-tier software distributed systems. This goal poses four major challenges:

(1) Selecting the appropriate modeling formalism
(2) Constructing models from requirements, or designs
(3) Analyzing (or solving) models to compute performance measures
(4) Giving feedback and possible design modifications to designers

In this research, the Layered Queueing Network (LQN) is selected as the modeling formalism. The LQN is a form of an extended queueing network designed to model software and hardware systems with multi-tier architectures. Layering appears in a system when a client makes a request to an intermediate server, which in turn makes requests to servers at even lower layers.

Three main methods are used to analyze the performance of a proposed software system: actual system construction, simulation of the model, and analytic solution of the model. Of the three methods, analytic analysis usually provides rapid performance evaluations, which allows designers to examine a larger variety of potential design alternatives

in a shorter period of time. In this work, analytic analysis is used to address the third challenge (performance measures computation). The second and the fourth challenges cited above are out of the scope of this thesis.

Much research has been done to come up with new replication techniques, but there has been little work done to consistently compare their performance. It remains difficult to evaluate the claims of performance improvements, made by designers of replicated systems. The difficulty is because most designers use simulations to measure the performance of some replication techniques [**45, 8, 99, 98, 47, 87**]. Simulations are very detailed, which usually makes the comparison of the performance improvements of different replication techniques unfair and inconsistent, because different levels of details can impact the performance evaluation. System designers need a high-level analytical tool to evaluate and compare the performance of different replication techniques, and to decide on the level of replication that fulfills the performance requirements economically. Then, different replication techniques can be modeled and their performance measures computed rapidly using one high-level tool, which allows to conduct a fair performance comparison.

In this thesis, well-known replication techniques, in parallel and distributed systems, are identified. Replication patterns from a performance perspective are developed and classified. The patterns are: Master-slave, Master-slave snapshot, Master-slave cascading, Consolidation, Read-One-Write-All, Quorum, Master-master, and Select. Replication patterns are then modeled using the LQN formalism to measure their performance efficiently. Designers can use these patterns to come up with the best performance solution for their domain of software application. The performance of different replication patterns can be compared more consistently than simulation, because they can be modeled using the LQN, which is a high level abstraction modeling formalism. To use LQN to model most of the

classified replication performance patterns, extensions to the LQN formalism are required, as described below.

In an LQN task (a software process), internal parallelism occurs when a main thread of control is forked into N concurrent heterogeneous threads. The main thread is said to be AND-forked. The execution of the main thread of control is suspended until J out of the N forked threads complete execution; this is called a quorum-join in which $J \leq N$. An AND-join is the special case of the quorum-join in which $J = N$.

Replication systems can use the quorum (voting) consensus protocols in which an operation is allowed to proceed to completion when a subset of replicas responds to requests made by that operation. The quorum pattern is introduced in the LQN formalism to model and compute the relevant performance measures of a replication model with a quorum consensus semantic. A thread within a process in the LQN formalism can have complex behavior as it sends requests to lower layer servers which in turn send requests to other lower servers. In an LQN model, to compute the relevant performance measures of a model with a quorum pattern, one should be able to directly convert the LQN model with a quorum to a Queueing Network (QN). Based on this study, this is intractable to do. For this reason, a method that converts an LQN model with quorum to another without quorum, that behaves approximately like the original one, has been developed.

In order to analytically compute the service time of a client with a quorum pattern, the J^{th} order statistic of the N random variables (RVs) associated with the delay distributions of the forked threads needs to be calculated. To do so, the cumulative distribution function (CDF) of each thread has to be evaluated or approximated. Prior to this thesis, the CDF was approximated using a distribution that provided high errors in many model cases in which $J \ll N$. In the models tested in this thesis, the percentage error in the performance measures can be as high as 270% when using a previous method. In this thesis, closed-form

formulas for the distribution functions of the thread delay are derived **[113]**. It is assumed that the number of requests the thread makes to lower layer server(s) is deterministic or geometrically distributed. The new closed-form formulas produce a significantly better approximation with a mean for the absolute value of the percentage error of 10% in the performance measures in all cases tested when $J < N$. In addition, the accuracy of the existing AND fork-join solutions is improved, as the new closed-form formulas are also used in the AND fork-join delay approximations. This error is acceptable for performance prediction at an early stage of software product development.

Large distributed systems often contain symmetrical components that can have complex behaviors such as parallel threads. The modeling power of the LQN formalism is extended and the LQNS is modified to model, and compute the relevant performance measures of replicated LQN tasks with AND-fork and AND-join internal parallelism in closed models. Symmetry in these systems is exploited in order to facilitate their modeling through a compact notation for replicated components, and to expedite the solution time. The approach iteratively computes the relevant performance measures for only one replica, then distributes the results appropriately to other replicas. The challenge is to compensate for the performance effects of other replicas, which are not part of a solution iteration. The approach is scalable, in terms of solution time, to the number of replicas in a model **[116, 115]**, and can efficiently compute waiting times and throughputs for models with tens of thousands of nodes and processes. This approach was used to evaluate the performance of a mobile network **[114]**.

Finally, LQN performance models for an industrial Information Management System (IMS) and an Air Traffic Control (ATC) system are constructed. The LQN model for the IMS includes replicated LQN tasks with internal AND fork-join parallelism. The LQN model for the ATC system is modeled using tasks that have AND-fork and quorum-join

internal parallelism. Those tasks are also replicated in another version of the LQN model. The performance measures for the models are computed using the new LQNS.

1.2. Contributions

This section describes the contributions of this work to the state of the art of software performance engineering. The contributions are: identifying and classifying of performance replication patterns, extending the modeling power of the LQN formalism, improving the accuracy of the Layered Queueing Network Solver (LQNS) solutions, and the case studies.

(1) Performance Replication Patterns (Chapter 4).

A UML sequence diagram and an LQN model template is developed for each pattern.

(2) Extension of the LQN fork-join internal parallelism to model the quorum pattern (Chapters **5**, and **6**)

(a) This extension requires the derivation of a cumulative distribution function of a thread delay to improve the approximation accuracy. This also improves the accuracy of the existing AND fork-join parallel threads.

(b) For the quorum pattern, the concept of overhanging (delayed) threads is introduced. In addition, a correction for the contention they introduce is developed.

(3) Extension of the LQN replication semantic to include parallel threads (Chapter **7**)

(4) Evaluation of the accuracy for different system architectures and configurations (Chapters **5**, **6**, and **7**)

(5) Case studies for applications drawn from real systems (Chapter **8**)

1.3. Thesis Organization

The remaining of the thesis is structured as follows:

- **Chapter 2** describes relevant techniques used for analytic performance modeling and analysis of software systems, especially the systems with layered architectures. The LQN formalism, and the LQNS are presented. They are the basis for the work in the thesis.
- **Chapter 3** provides a literature review of the replication techniques in parallel and distributed systems. Further, it reviews the techniques of modeling replication and exploiting symmetry in replicated systems. This literature review provides the background necessary to formalize the research problem.
- **Chapter 4** identifies and develops performance replication patterns. It proposes and justifies the solutions required for systems with replication patterns.
- **Chapter 5** presents a technique to increase the accuracy of approximating the distribution of a thread delay. Closed-form formulas for the distribution function of a thread delay are derived.
- **Chapter 6** defines the semantics of the quorum pattern, and develops its solution technique in the LQNS.
- **Chapter 7** shows the solution for solving replicated models that have AND-fork and quorum-fork behaviors.
- **Chapter 8** shows the application of the results of this thesis to two case studies: an Air Traffic Control system and an Industrial Information Management System.
- **Chapter 9** draws conclusions and summarizes the benefits of the contributions of the work. In addition, future work is addressed.

CHAPTER 2

Layered Queueing Networks and Performance Analysis

Performance studies are conducted by system designers to analyze large distributed systems for the purpose of capacity planning [**106**], and software performance engineering [**149**]. One way to study the performance of a system is by constructing a performance model. A performance model represents the structure of a system, and its behavior in terms of its performance parameters. It can be used to model an existing system or one that is being developed. The results of solving the performance model are service times, throughputs, and utilizations of all components in the model.

This chapter discusses the advantages and disadvantages of analytic solutions over simulations. It describes the Markov chains as a formalism for analytic solutions. Then, the queueing networks (QNs) formalism is introduced, which avoids the state space explosion problem of Markov chains. After that, method of surrogates which provides solutions for non-product form queueing networks, is presented. Then the Stochastic Rendezvous Network (SRVN), and the Method of Layers (MOL), used to model layered software systems, is described. Finally, the LQN, an extension to the SRVN and MOL models, formalism, and the LQNS are presented. The LQNS is the analytic tool used in this thesis.

2.1. Analytic versus Simulation Solutions

Computer systems analysts have several tools to use in evaluating systems' performance. These include queueing models, trace-driven simulations, and statistical models

derived from observed data. The selection of a tool depends on the level of accuracy needed, and/or the speed required to obtain results.

Computer center managers and systems analysts rank tools in an increasing order of credibility as follows [**30**]: queueing models, discrete-event random number simulations, trace-driven simulations, monitoring systems running synthetic jobs, and measurements [**5**] of a real workload on a real system.

Simulation models require more details, thereby making them expensive to design, code, debug, parametrize and execute. In order for it to be more accurate, the simulation needs to be rerun many times and for long periods to gain high confidence in the performance measures. The long run time can be a serious problem when evaluating different configurations of a system.

Although simulation models may provide results that are more accurate, analytical models are useful for analyzing large systems. They are computationally more efficient and their parameters are easier to obtain, because of their higher level of abstraction. Analytical models are based on sets of mathematical expressions that give the values of the desired performance measures as a function of the set of values of the performance parameters [**106**]. These expressions can usually be solved rapidly which aids in exploring the parameter space of a system.

Often, many simplifying assumptions are required to use analytical models. These assumptions may result in a model that might not represent accurately the system under study. They will result in an error in the prediction of performance measures. Nevertheless, the prediction error for service time of 10% to 30% can be acceptable for a great number of applications, as it is the case for performance modeling and analysis at an early stage of a product life cycle. Therefore, analytic tools can be more useful than simulations in some applications.

One formalism used for the analytic evaluation of a performance model is Markov chains, which will be described in the next section.

2.2. Markov Chains

A Markov chain is a discrete state space stochastic process. The future state of the system depends on the current state only, this defines the Markov property. There are two kinds of Markov chains: Discrete-time Markov chains (DTMCs) in which the time, at which the system is observed, is discrete, and Continuous-time Markov chains (CTMCs) in which the time is continuous. A transition probability governs the transition of the system from the current state to another state.

In a DTMC, the value of a RV X_i represents the state of the system at time i. The transition probability is a conditional mass distribution for X_i given by:

$$P(X_i = x|X_{i-1} = x_{i-1}, X_{i-2} = x_{i-2}, \cdots, X_0 = x_0) = P(X_i = x|X_{i-1} = x_{i-1}), \quad (2.1)$$

In a CTMC, the state change can occur at any time t. If $t_0 < t_1 < t_2 < \cdots < t_n < t$; the transition probability is a conditional probability distribution given by:

$$\begin{aligned} &P(X(t) = x|X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \cdots, X(t_0) = x_0) \\ &\quad = P(X(t) = x|X(t_n) = x_n), \end{aligned} \quad (2.2)$$

where x is a future state of the system at time t.

A performance study of a client server system based on Markov chains is conducted in [74]. In general, Markov chains are the most expensive in their solution time and space requirements. They are useful for studying detailed communications among concurrent

entities. However, it is usually not possible to apply them to complex systems, because they suffer from state space explosion.

2.3. Queueing Networks

The key components of a queueing network are *service centers* and *queues*. A queue is a buffer with a queueing discipline such as FIFO, Round Robin, or LCFS preemptive resume. A service center is the component that provides service to the customers. Examples of service centers include a bank teller, hardware device like a disk, or database. Each service center has a queue that holds jobs to process. A replicated service center represents identical providers of service, which process jobs from a single queue. For example, a bank teller counter has one line of customers but many tellers. A service center with infinite replicas is called a *delay center* and may be used to model a communication delay.

Queueing network analysis can produce results both for individual queues (associated with service centers), and for the whole network. To derive performance measures for individual service centers, the execution demands and arrival rates must be known. The execution demand is the mean time the service center takes to process one job. The arrival rate is the mean rate at which jobs arrive. The execution demand and the time between job arrivals often are assumed to have exponential distributions.

Performance measures for a single queue, that can be calculated using queueing theory, include **[85, 77, 63]**:

- Utilization which is the percentage of time the service center is busy
- Mean time a job spends waiting in the queue
- Mean queue length
- Probability the queue length is n

- Drop rate at which incoming jobs are discarded because of overflow in a queue of length B
- Latency which is the time to process a job
- Throughput which is the rate at which jobs are processed
- Whether the system is stable or overloaded - in an overloaded system, the arrival rate is greater than the throughput
- Number of outstanding jobs in the most utilized service center.

Open queueing networks allow a varying number of jobs in the system. The arrival rate does not usually depend on the number of jobs already in the system. Closed queueing networks consider a fixed number of jobs in the system. That is, a completed job is immediately replaced by a new one.

Open queueing networks can be used in a database system in which the number of transactions is typically not constant for example. In certain cases, closed models provide easier solutions than open models. For instance, the performance of concurrency control methods depends on the multi-programming level. A closed model provides easier solutions, because the number of concurrent transactions is constant for a given multi-programming level.

Some systems process several types of jobs. To model jobs with different behavior, each queue is divided into one or more *job classes*. Transition probabilities are specified between job classes, instead of queues. Customers of the same type are grouped in a *chain*. In general, a chain consists of one or more classes that represent different services in the queueing network. A chain can be used to allow a customer to request two services, using two classes, from a given service center [**11**]. In this thesis, it is assumed that classes are global and there is one class per chain.

2.3.1. Mean Value Analysis.

Mean Value analysis (MVA) is an iterative algorithm used to solve closed product form queueing networks in steady state to give mean performance results for delay and throughput [**77, 91**]. The basic idea of MVA is to find the arrival-instant mean queue length $Q_m(\overrightarrow{N} - \overrightarrow{1}_k)$ and use it to find the residence time of an average customer. The residence time ($R_{k,m}$) of class k at service center m is found as follows:

$$R_{k,m} = D_{k,m} \times (1 + Q_m(\overrightarrow{N} - \overrightarrow{1}_k)) = D_{k,m} \times (1 + \sum_{j=1}^{K} Q_{m,j}(\overrightarrow{N} - \overrightarrow{1}_k)), \qquad (2.3)$$

where

- $D_{k,m}$ represents the execution demand at service center m for class k
- K is the number of job classes in the queueing network
- $\overrightarrow{N}$ is a vector of customer population $(N_1, N_2, N_3, \cdots, N_K)$ where N_k denotes the number of customers belonging to class k, for $k = 1, \cdots, K$
- $\overrightarrow{1}_k$ is the unit vector with a one in the k^{th} place and zero elsewhere
- $Q_m(\overrightarrow{N} - \overrightarrow{1}_k)$ is the mean queue length at service center m with one less customer from class k present in the network
- $Q_{m,j}(\overrightarrow{N} - \overrightarrow{1}_k)$ is the mean queue length of class j at service center m with one less customer from chain k present in the network.

The residence time is then used to derive the throughput. Finally, from this throughput a new queue length, that will be used in the next iteration, is found using Little's law [**97**]. Little's law states that the average queue length of customers in a stable system, over some time interval, is equal to their throughput multiplied by their residence time in the system.

Two approaches exist for evaluating the arrival instant queue lengths, exact and approximate. The exact MVA requires an evaluation at every possible population from $(0, \cdots, 0)$

up to the target population $\overrightarrow{N}$. Eq. (2.3) requires the queue length at the service center when there is one less customer of class k present. Algorithm 2.3.1 shows the exact MVA for a single class of customers, and Algorithm 2.3.2 shows the exact MVA for a queueing network with multiple classes. The computational complexity increases with the number of classes and centers. The time complexity of the multi-class exact MVA is $\Theta(K \prod_{k=1}^{K}(N_k + 1))$, and the space complexity is $\Theta(M \prod_{k=1}^{K}(N_k + 1))$.

ALGORITHM 2.3.1. *Single-class exact Mean Value Analysis algorithm.*

- Residence time R, Throughput X, Think time Z.
- Number of service centers M, Demand at service center m D_m, Number of customers N, Queue length Q.
- FOR ($m = 1$ to M) $Q_m = 0$;
- FOR $n = 1$ to N
  - ✧ FOR ($m = 1$ to M) $R_m = \begin{cases} D_m & Infinite\,Server \\ D_m \times (1 + Q_m) & FCFS, LCFSPR, PS \end{cases}$;
  - ✧ $X = \frac{n}{Z + \sum_{m=1}^{M} R_m}$;
  - ✧ FOR ($m = 1$ to M) $Q_m = X \times R_m$;
- END FOR

ALGORITHM 2.3.2. *Multi-class exact Mean Value Analysis algorithm.*

- Residence time R, Throughput X, Think time Z.
- Number of service centers M, Number of classes K, Demand for class k at service center m $D_{k,m}$, Number of customers N, Queue length Q.
- FOR $(m = 1$ to $M)$ $Q_m(\overrightarrow{0}) = 0$;
- FOR $n = 1$ to $\sum_{k=1}^{K} N_k$
  - ✧ FOR all $|\overrightarrow{n} \equiv (n_1, .., n_K)| = n$
    - ❑ FOR $k = 1$ to K
      - ▷ FOR $(m = 1$ to $M)$

$$R_{k,m} = \begin{cases} D_{k,m} & Delay\ centers \\ D_{k,m} \times (1 + \sum_{j=1}^{K} Q_{m,j}(\overrightarrow{n} - \overrightarrow{1_k})) & Queueing\ centers \end{cases};$$

      - ▷ END FOR
      - ▷ $X_k = \frac{n_k}{Z_k + \sum_{m=1}^{M} R_{k,m}}$;
      - ▷ FOR $(m = 1$ to $M)$ $(Q_{m,k}(\overrightarrow{n}) = X_k R_{k,m})$;
    - ❑ END FOR
  - ✧ END FOR
- END FOR

The approximate methods estimate the arrival instant queue lengths $Q_m(\overrightarrow{N} - \overrightarrow{1}_k)$ based on the time averaged queue lengths at the service centers with the full customer population $(\overrightarrow{N})$, and iteration is used to improve the estimate. Bard-Schweitzer **[143, 92]** and Linearizer **[43, 29]** are two well-known approximate methods.

Bard-Schweitzer approximation is developed by replacing the $Q_{m,j}(\overrightarrow{N} - \overrightarrow{1}_k)$ term in Eq. (2.3) by:

$$Q_{m,j}(\overrightarrow{N} - \overrightarrow{1}_k) = \begin{cases} Q_{m,j}(\overrightarrow{N}) & j \neq k \\ \frac{(N_k-1)}{N_k} Q_{m,k}(\overrightarrow{N}) & j = k \end{cases}, \tag{2.4}$$

and iterating until convergence. The Linearizer algorithm estimates $Q_{m,j}(\overrightarrow{N} - \overrightarrow{1}_k)$ by using the following equations:

$$F_{m,j}(\overrightarrow{N}) = Q_{m,j}(\overrightarrow{N})/N_j, \tag{2.5}$$

$$D_{m,j,k}(\overrightarrow{N}) = F_{m,j}(\overrightarrow{N} - \overrightarrow{1}_k) - F_{m,j}(\overrightarrow{N}), \tag{2.6}$$

$$Q_{m,j}(\overrightarrow{N} - \overrightarrow{1}_k) = (\overrightarrow{N} - \overrightarrow{1}_k)_j (F_{m,j}(\overrightarrow{N}) + D_{m,j,k}(\overrightarrow{N})), \tag{2.7}$$

where

- $F_{m,j}(\overrightarrow{N})$ is the fraction of class j jobs at queue m for population $\overrightarrow{N}$
- $D_{m,j,k}(\overrightarrow{N})$ is the change in the fraction of class j at queue m due to the of the removal of one class k job
- $(\overrightarrow{N} - \overrightarrow{1}_k)_j$ is the population, which is an integer number, of class j when one class k job is removed from population $\overrightarrow{N}$. For example, assume the full population is (n_1, n_2), then if one customer of class one is removed, then $(n_1 - 1, n_2)_1 = n_1 - 1$, and $(n_1 - 1, n_2)_2 = n_2$.

Eq. (2.7) is equivalent to Eq. (2.4) when $D_{m,j,k}(\overrightarrow{N}) = 0$. Linearizer improves the accuracy of Schweitzer algorithm by computing the scaling factor $D_{m,j,k}(\overrightarrow{N})$ used to find queue lengths when one customer is removed.

Algorithm 2.3.3 shows a heuristic algorithm to approximate the mean statistics for a queueing network with population $\overrightarrow{N}$. This algorithm is called the Core algorithm[**29**]. Bard-Schweitzer approximate MVA uses the Core algorithm with input $D_{m,j,k}(\overrightarrow{N}) = 0$. In addition, the Bard-Schweitzer algorithm estimates $Q_{m,k}$, for $m = 1 \cdots, M$ and $k = 1, \cdots, K$, by uniformly distributing the jobs of each class among the queues at initialization.

The Linearizer algorithm solves the queueing network using Brad-Schweitzer approximation at both the full customer population $\overrightarrow{N}$ and $\overrightarrow{N} - 1_k$ for all k. Algorithm 2.3.4 outlines the Linearizer approximate MVA.

Approximate MVA techniques do not enumerate state spaces (population levels) to predict a system's performance behavior, so they provide solutions that are computationally more efficient than the exact MVA technique.

ALGORITHM 2.3.3. *Core: a heuristic algorithm for a queueing network with population* $\overrightarrow{N}$ *and multiple classes.*

- Residence time R, Throughput X, Think time Z.
- INPUTS: Number of service centers M, Number of classes K, Demands $D_{k,m}$, Number of customers N_k for class k, estimates of $D_{m,k,j}(\overrightarrow{N})$, estimates of Queue lengths $Q_{m,k}$.
- %There is only one population of size N=$\sum_{k=1}^{K} N_k$. The population is $(N_1, N_2, \cdots, N_k)$, for $k = 1, \cdots, K$.

function CORE($\overrightarrow{N}$):

Initialize: $I = 0$;

- DO
  - ✧ $I = I + 1$;
  - ✧ FOR ($m = 1$ to M) % Step 1: estimate $Q_{m,k}(\overrightarrow{N} - \overrightarrow{1}_j)$.
    - ❑ FOR ($k = 1$ to K)
      - ▷ $F_{m,k}(\overrightarrow{N}) = Q_{m,k}(\overrightarrow{N})/N_k$;
      - ▷ FOR ($j = 1$ to K) ($Q_{m,k}(\overrightarrow{N} - \overrightarrow{1}_j) = (\overrightarrow{N} - \overrightarrow{1}_j)_k(F_{m,k}(\overrightarrow{N}) + D_{m,k,j}(\overrightarrow{N}))$);
    - ❑ END FOR
  - ✧ END FOR
  - ✧ FOR $k = 1$ to K %Step 2: single iteration MVA.
    - ❑ FOR ($m = 1$ to M)
      - ▷ $R_{k,m} = \begin{cases} D_{k,m} & Delay\ centers \\ D_{k,m} \times (1 + \sum_{j=1}^{K} Q_{m,j}(\overrightarrow{n} - \overrightarrow{1_k})) & Queueing\ centers \end{cases}$;
    - ❑ END FOR
    - ❑ $X_k = \frac{n_k}{Z_k + \sum_{m=1}^{M} R_{k,m}}$;
    - ❑ FOR ($m = 1$ to M) ($Q_{m,k}(\overrightarrow{n}) = X_k R_{k,m}$);
  - ✧ END FOR
- WHILE $\max_{\forall k \epsilon K, \forall m \epsilon M} \frac{|Q^I_{m,k}(\overrightarrow{N}) - Q^{I-1}_{m,k}(\overrightarrow{N})|}{N_k} > \frac{1}{4000 + 16|\overrightarrow{N}|}$

ALGORITHM 2.3.4. *Linearizer algorithm for multiple classes.*

- Residence time R, Throughput X, Think time Z.
- INPUTS: Number of service centers M, Number of classes K, Demands $D_{k,m}$, Number of customers N_k for class k.
- FOR ($m = 1$ to M) %Initialization. Assume the population is uniformly distributed.
  - ✧ FOR ($k = 1$ to K)
    - ❑ $Q_{m,k}(\overrightarrow{N}) = N_k/M$;
    - ❑ FOR ($j = 1$ to K)
      - ▷ $Q_{m,k}(\overrightarrow{N} - \overrightarrow{1}_j) = (\overrightarrow{N} - \overrightarrow{1}_j)_k/M$;
      - ▷ $D_{m,k,j}(\overrightarrow{N}) = 0$;
    - ❑ END FOR
  - ✧ END FOR
- END FOR
- FOR ($I = 1$ to 3) %Step 1.
  %Use the most recent values for $Q_{m,k}(\overrightarrow{N})$ and $D_{m,k,j}(\overrightarrow{N})$ as inputs to Core($\overrightarrow{N}$).
  - ✧ Core($\overrightarrow{N}$);
  - ✧ IF (I= 3) break;
    %Use the most recent values for $Q_{m,k}(\overrightarrow{N} - \overrightarrow{1}_k)$ and $D_{m,k,j}(\overrightarrow{N} - \overrightarrow{1}_k)$ as inputs
    %to Core($\overrightarrow{N}$-$\overrightarrow{1}_k$). Linearizer assumes $D_{m,k,j}(\overrightarrow{N} - \overrightarrow{1}_k)$=$D_{m,k,j}(\overrightarrow{N})$.
  - ✧ FOR ($k = 1$ to K) Core($\overrightarrow{N} - \overrightarrow{1}_k$); %Step 2.
  - ✧ FOR ($m = 1$ to M) %Step 3.
    - ❑ FOR ($k = 1$ to K)
      - ▷ $F_{m,k}(\overrightarrow{N}) = Q_{m,k}(\overrightarrow{N})/N_k$;
      - ▷ FOR ($j = 1$ to K)
        $F_{m,k}(\overrightarrow{N} - \overrightarrow{1}_j) = Q_{m,k}(\overrightarrow{N} - \overrightarrow{1}_j)/N_k$;
        $D_{m,k,j}(\overrightarrow{N}) = F_{m,k}(\overrightarrow{N} - \overrightarrow{1}_j) - F_{m,k}(\overrightarrow{N})$;
      - ▷ END FOR
    - ❑ END FOR
  - ✧ END FOR
- END FOR

2.3.2. Surrogate Delays Method.

In computer systems, remote procedure calls raise a phenomenon called simultaneous resource possession. Simultaneous resource possession occurs when two or more resources, such as memory and CPU, must be used by a customer at the same time. Models

with simultaneous resource possession cannot be solved directly using product form solution techniques, because they violate the independence requirement of the product form networks, in which the service time at a center depends on other centers in the queueing network. If the effects of simultaneous resource possession are ignored in the Mean Value Analysis solution, the throughput value obtained from a performance model will be overestimated, because the time needed to acquire resources is not considered.

Jacobson and Lazowska [**76**] developed the *method of surrogates* to overcome this problem of overestimation. Using the method of surrogates, 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 among the models until convergence.

De Souza e Silva and Muntz [**42**] generalize the method of surrogates to handle resources by customer chains and to handle nesting of resources. Jenq [**78**] demonstrates the technique for a substantial distributed transaction testbed system with the effects of the concurrency control protocol, the transaction recovery protocol, and the commit protocol all modeled and validated against measurements.

2.4. Stochastic Rendezvous Networks

The SRVN is a form of an extended queueing network designed to model software systems with nested resource requests, a phenomenon commonly found in multi-tier client-server systems. Nested resource requests occur when several concurrent tasks are communicating with each other by the send-receive-reply (rendezvous) mechanism. A client task initiates the rendezvous with a server task and is blocked until a response from the server is received. Layering takes place when a client makes a request to an intermediate server,

which in turn makes requests to servers at even lower layers. While a lower layer server is processing a request, all of the intermediate servers between this server and the client are blocked. This layered interaction is a form of simultaneous resource possession where the order of acquisition and release is strictly hierarchical.

The service time for one class of service is not a constant parameter but is determined by its lower servers. In software systems, delays and congestion are greatly affected by synchronous interactions such as remote procedure calls (RPCs) or rendezvous. The layered model captures these delays by incorporating the lower layer queueing time and service time into the service time of the upper layer server. This "*active server*" feature distinguishes the layered queueing networks from conventional queueing networks.

The SRVN **[166, 164, 93, 167, 163]** formalism models tasks, which consist of clients and servers, that interact using the RPC paradigm. The tasks represent hardware devices and software processes. The model is solved by constructing a submodel for each server. Each submodel contains all the clients and surrogate delays for the server. Each client is modeled as a chain in the underling queueing network of the model. The number of customers in that chain is equal to the multiplicity of the client task. The underlying queueing network for each submodel is solved using a MVA technique, and the whole LQN model is solved using fixed point iteration between the submodels. Tasks in SRVN models cannot have internal parallelism.

2.5. Method of Layers

The *Method of Layers* **[133, 135]** solves a layered client-server queueing network by decomposing the network into a set of two-layer MVA submodels and then solving each of these models using the Linearizer approximate MVA algorithm **[29]**. Each submodel forms a conventional product form queueing network where the servers form the service centers

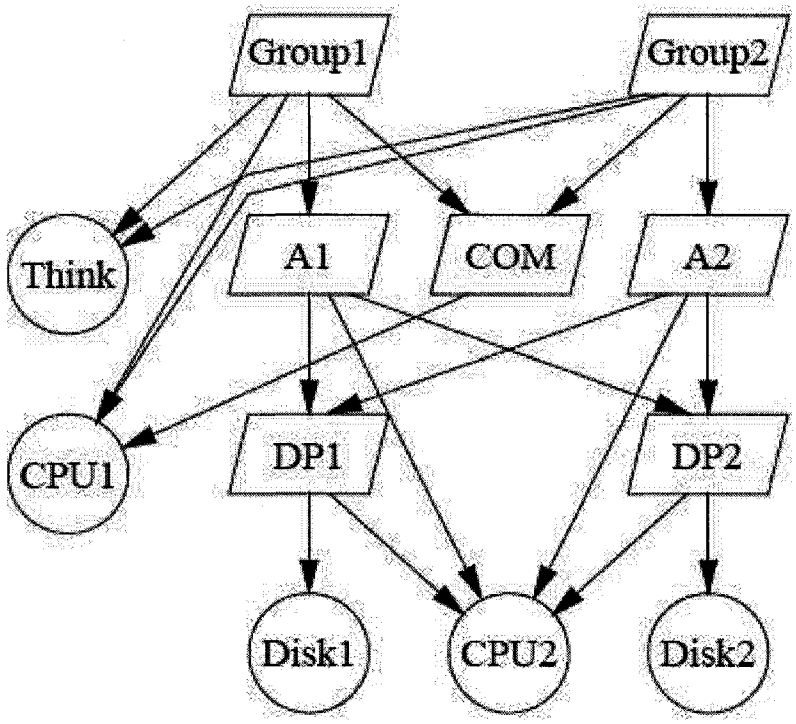
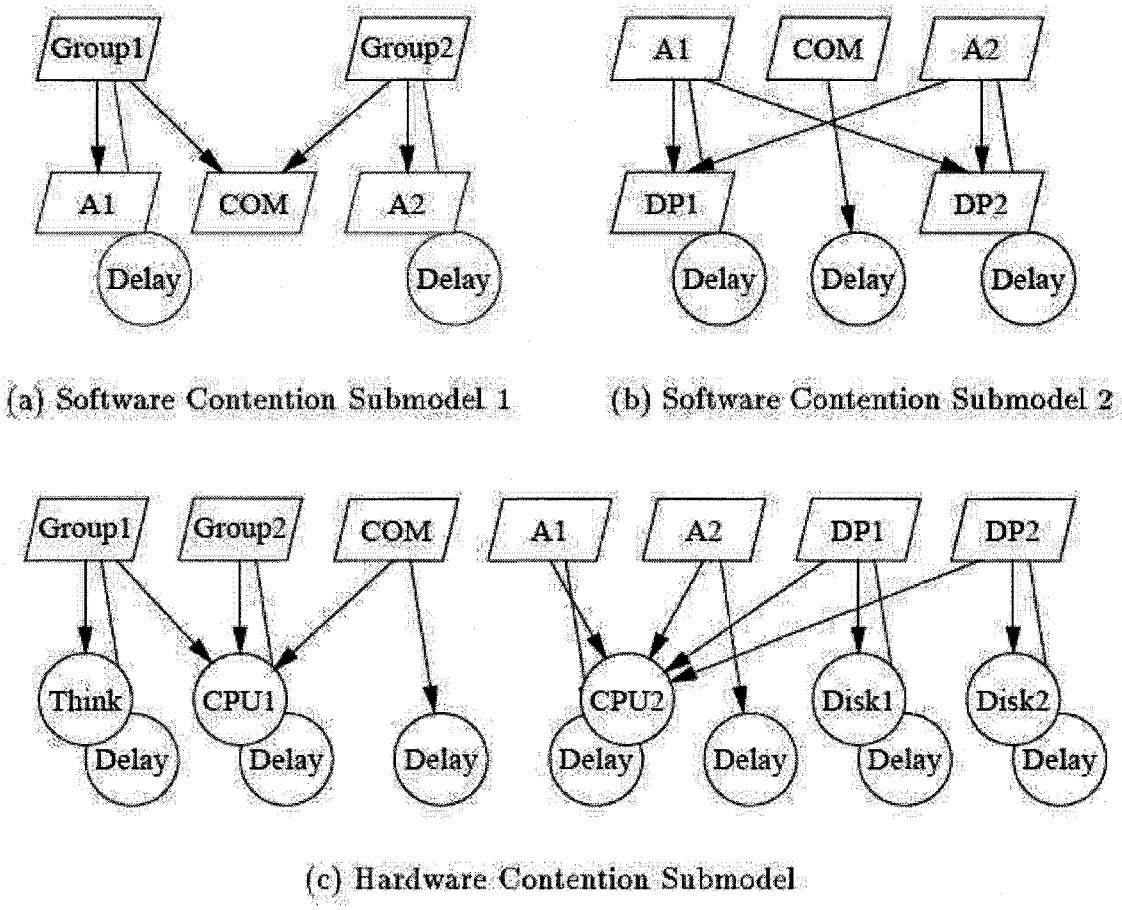


Figure 2.1: A multi-tier client-server system from [**135**]. Tasks are represented by parallelograms. The customers are represented by the tasks Group1 and Group2. Pure servers, such as devices and think times for customers, are represented by circles.

and the clients form the customers. Figure 2.1 shows a multi-tier client server model, and Figure 2.2 shows its submodels when the MOL is used.

The MVA submodels are constructed first by splitting the input model into two submodels, one for hardware contention and the other for software contention. The hardware contention submodel consists of all the tasks and devices from the input model; the software tasks act as clients and the devices act as servers. Next, the software contention submodel, using only the tasks from the input model, is sorted into layers. Software submodel n is



(a) Software Contention Submodel 1

(b) Software Contention Submodel 2

(c) Hardware Contention Submodel

Figure 2.2: Submodels generated by the Method of Layers for the model in Figure 2.1. The circles marked Delay are surrogate delays introduced during the solution of the model [135].

then constructed by using all the tasks in layer n as clients, and all the tasks from layer $n+1$ as servers.

The Method of Layers approximates the performance of the system under study by iterating among the various submodels. It begins by solving the software submodels, of N layers, from submodel 1 to submodel $N-1$. There is no software submodel N, because the pure servers at layer N make no requests. Once the software submodels have converged, the performance results are used to set the think times and service times for the tasks in

the hardware model. The performance approximations from the solution of the hardware model are then used to set the service times for the various software submodels. This sequence continues until the desired convergence in the service times is met.

2.6. Layered Queueing Network

The LQN formalism is an extension of the MOL, which is designed to model software systems with nested resource requests. The LQN does not differentiate between software and hardware models, and it allows internal concurrency in a task, and it solves mixed (open and closed) models.

An LQN model can be graphically presented as a directed graph with vertices and arcs. The vertices represent entries that provide different services. In the LQN formalism, a task represents a logical or a software resource that accepts requests and can make requests to other resource(s). In the LQN formalism, a processor represents a hardware resource that accepts requests only. A task has one or more entries that provide different services. Arcs represent visits from one entry to another entry [**55**]. Arcs, which indicate requests, are used to link tasks. Arcs also associate tasks with processors. There may be several layers of tasks and processors interacting simultaneously. When one task or processor is visited by more than one task, there is a contention and a queueing delay.

A task can have internal concurrency using threads that fork and join. In addition, a task can be multi-threaded by having multiple copies of itself. An entry consists of one ore more phases. Furthermore, an entry can be linked to an activity graph that represents an execution scenario.

2.6.1. Processors.

In the LQN formalism, processors are pure servers, as they only accept requests from tasks. They are used to consume *time* in the model. They are similar to service centers

in conventional queueing networks. Multiple identical copies of a processor are shown in an LQN model using a stacked icon. An infinite processor, one with no contention, is shown in an LQN with the label ∞. Each server processor has a single queue for requests. By default, requests are served in a first-in, first-out discipline. Processors also support processor-sharing and priority-preemptive-resume scheduling.

2.6.2. Tasks.

Three groups of tasks exist in LQN models: pure clients, active servers, and pure servers. Pure clients, also called reference tasks, only make requests. Reference tasks cycle indefinitely. They have the role of customers and can represent actual tasks running on a computer or some other source of requests such as actual users. A "stack" of tasks represents a multi-threaded task that has multiple customers of the same class or type. The number of copies is illustrated in an LQN model by braces. The number of copies represents the number of customers. Client tasks may be queued awaiting service from their processors.

A delay server task is represented using an infinite server, illustrated in an LQN model by the label ∞. A delay server can contend with other tasks in an LQN model, when the delay server and the other tasks make requests to the same processor. Active servers are tasks that both accept requests from higher layers, and make requests to lower layers. Pure servers are tasks that only accept requests from higher layers.

Tasks can represent multiple objects in the model, such as:

- Actual processes or threads in a system
- Non-processor devices such as disks
- Resources such as buffers.

The *host processor* is represented by a circular symbol attached to a task. Task co-allocation can be indicated by associating several tasks to the same processor. If no processor is

shown, then the task has its own processor. Requests for service to lower layer tasks can be one of three types: synchronous, asynchronous, or forwarded.

- **Synchronous or send-receive-reply requests**: tasks communicate between one and another using the send-receive-reply [**34**] messaging paradigm. The task making requests, the client, is blocked during the time between the send and the response. This style of interprocess communication is also known as a rendezvous [**2**], and it models the remote procedure call [**159**]. The queueing discipline for requests is first-in first-out. Tasks also support head-of-line priority queueing for requests. A synchronous request is illustrated in an LQN model as a solid arc with a filled arrow-head denoting the direction of the send. The mean number of requests is shown in an LQN model within parentheses. The mean number of requests can be either geometrically distributed with a stated mean, or deterministic. In either case, the RV associated with the number of requests takes a non-negative integer value.
- **Asynchronous or send-no-reply requests**: the client sends a request and continues execution. No response is generated by the serving task. The mean number of requests is shown in an LQN model within parentheses. The mean number of requests can be geometrically distributed with a given mean, or can be deterministic.
- **Forwarded requests**: the server passes the request to another lower layer server. The intermediate server is free to accept more requests. The client remains blocked. The probability of the request being sent to the next server is shown in parentheses. Otherwise, a response is sent to the originating client.

Finally, non-reference tasks can also accept open arrivals with given request rates.

2.6.3. Phases.

The time a server task spends processing a request, either awaiting responses from lower layer servers, or executing on a processor, is broken up into phases [**54**]. Phase one is the time spent between a receive operation and a response operation in a server. Any operation taken by the server after sending the response is executed in the second or subsequent phases. Second phases are a common performance optimization technique. For example, consider the case for transaction cleanup and logging, or delayed write operations. These phases are executed autonomously by the server task and run concurrently with the client task. The client that initiated their execution is no longer blocked.

Any phase can make synchronous or asynchronous requests. Forwarding requests always occur at the end of phase one. In an LQN input model, a phase type of an entry specifies whether an exact number of requests to a remote entry are made (deterministic phases), or a geometric number of requests are made (stochastic phases).

2.6.4. Entries.

A server uses entries to perform different actions. Entries may correspond to actual communication ports on software processes, or they may correspond to request types that invoke different actions at a server. Entries may have more than one phase. Each entry has its own phase execution demand and service time. Between tasks, an entry of one task visits or issues requests to the entry of another task using synchronous, asynchronous, or forwarding requests. A task always has at least one entry.

In an LQN model, entries are shown using small parallelograms within task icons. The mean total execution time of phase p in entry e in the host processor is called the execution demand of the phase. An execution demand is composed of slices. A slice is the time taken to execute an individual request. The execution demand for an entry is shown inside square brackets. By default, the service time of requests made by an entry to a processor are

exponentially distributed. In this case, the service time of a phase is the sum of a random number of exponentially distributed execution slices. However, a variance can be specified by setting the squared coefficient of variation of the execution demand of each phase:

$$CV^2 = \mathrm{Var}[S]/\mathrm{E}[S]^2, \tag{2.8}$$

where CV is the coefficient of variation,Var is the variance, and E is the expected or mean value of the RV S associated with the execution demand.

2.6.5. Activities.

Activities are the lowest granularity in an LQN model. Activities consume time on the processor on which their task runs. Activities can make requests to other tasks using synchronous or asynchronous requests. Requests are received by entries, and entries invoke activities that, in turn, pass control on to other activities. Activities are linked so as to structure a directed graph, called an activity graph, which represents one or more execution scenarios.

2.6.6. Threads.

In an LQN task with an activity graph, when a request is received by an entry of the task, the entry creates the main thread of control. The main thread of control starts by executing the top activity connected to the entry. Execution of the main thread of control may fork into concurrent threads of control. Forking occurs when a thread of control splits into two or more concurrent sub-threads. In the LQN formalism, there are two forms for forking: *AND-fork* and *OR-fork*. After an AND-fork, all successive threads can execute concurrently. It is assumed that extra threads are available or created after forking. Execution may also choose randomly between different paths using an OR-fork. After an OR-fork, only one of the successive threads is executed, with probability p. Sequential execution is a special

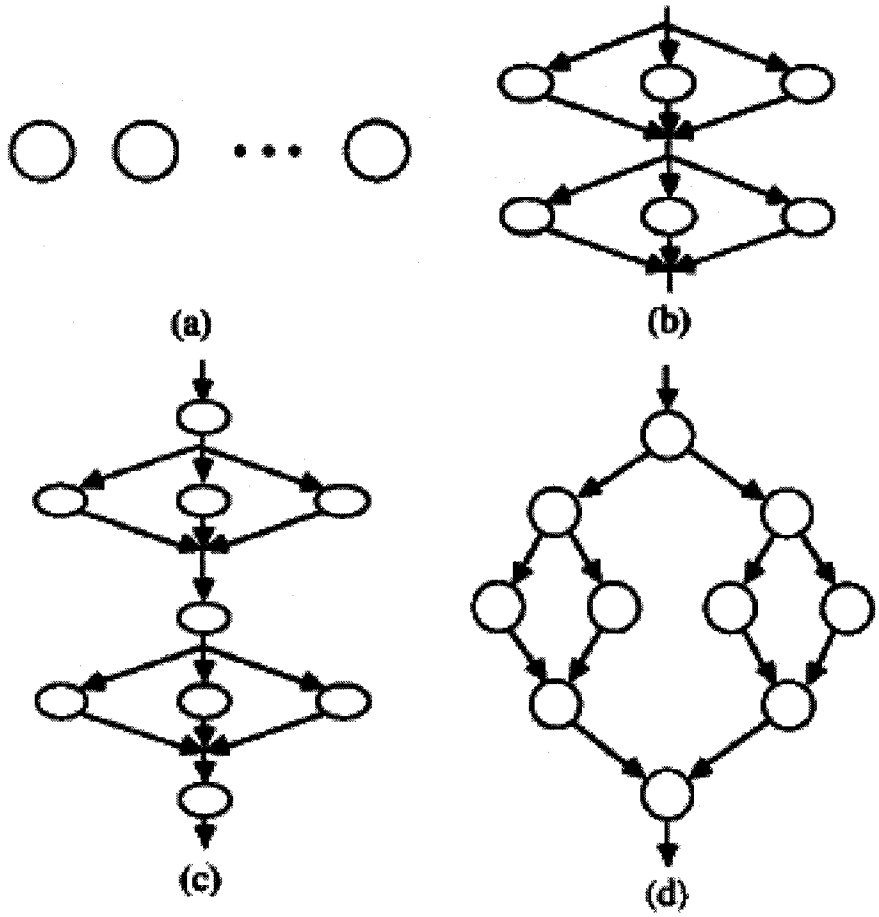


Figure 2.3: Four types of parallel computations (a) Asynchronous (b) Synchronous (c) Master-slave (d) Partitioned [**102**].

case of an OR-fork with only one thread. Tasks will only accept requests when all internally forked threads have been completed.

Four patterns for computations are presented in [**102**] and shown in Figure 2.3: asynchronous, synchronous, master-slave, and partitioning. The LQN formalism can model all these patterns.

In the LQN formalism, multi-threaded tasks, in which multiple copies of the same task are used, are called homogeneous threads. Conversely, threads, which are created as a

result of forking a main thread of control inside a task, are called heterogeneous threads. Internal parallelism is modeled by using heterogeneous threads.

2.6.7. Modeling Tasks with Internal Parallelism in LQN.

Parallelism can occur in Layered Queueing Networks through the use of internal parallelism within a task through the fork and join pattern. In the LQN formalism, each task, with the exception of the pure client tasks, consists of a single main thread of control that continually loops waiting for requests, then processes the corresponding requests when a request arrives. The behavior of a task with internal parallelism is more complicated than the behaviour of a task with no internal parallelism. In a task with internal parallelism, when the main thread of control for the task reaches the fork point, denoted by a circle labeled with an ampersand in an LQN model, independent threads of control are created, one for each branch after the fork point. These threads of control continue until they reach the join point, also denoted by a circle labeled with an ampersand, whereupon they are destroyed. Once all of these threads have been completed, the main thread of control continues. Note that this behavior itself may nest. The sub-threads may fork new threads that join later.

Two problems arise when trying to solve product form queueing models with fork-join behavior:

- creation of customers in the queueing network through the fork,
- computation of the synchronization delay.

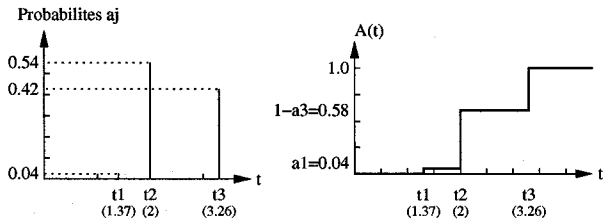
The creation of customers in the queueing network is a problem, because the creation violates the assumptions of the product form solution. Heidelberger et al. [**67**] develop an approximate approach to overcome this problem. They create separate routing chains for each of the threads in the model. The time needed for all of the sub-threads to complete execution, i.e. the *AND-join delay,* is inserted as a surrogate delay into the delay of the

parent thread. Mak and Lundstrom **[102]** improves upon Heidelberger's approach, by removing contention among the parent and child threads in the queueing model. Mark and Lundstrom's approach is further generalized to multi-servers in **[53]**.

The value of the AND-join synchronization delay is found by taking the product of the distributions of the execution times of all of the threads involved. In general, these distributions are not known *a-priori,* so some distribution is assumed, typically exponential. In an LQN model, the execution time distribution of an activity is typically non-exponential, because the activity can make blocking requests to lower layer server(s). These blocking requests cause the execution of the activity to be broken up into a possibly geometrically distributed number of *slices,* with the slices themselves having non-exponential service time distributions. Further, when solving the model analytically, the Mean Value Analysis does not compute higher-order moments so only the mean time to lower layer servers is known. An auxiliary model must be solved, though it only supplies variance **[166, 133, 172]**.

Notwithstanding these limitations, a simple *"three-point"* approximation was found to be effective for computing AND-join delays where a discrete distribution was fit to the mean and variance using only three points **[80]**. Figure 2.4 illustrates the point probabilities $a_j(t)$ for one thread and the approximation $A(t)$ for the probability distribution function for that thread. The locations t_j's, for $j = 1, 2, 3$, of the three points and the values a_j's of their probabilities are derived from the mean $\mathrm{E}[t]$, and variance $\mathrm{Var}[t]$ of their delay as follows:

Step 1: Find the locations of the three points:



(a) Probablity mass function. (b) Probability distribution function.

Figure 2.4: Three-point distribution with mean=2.0 and variance=0.4 [**53**].

$$
\begin{aligned}
t_1 &= \begin{cases} \mathrm{E}[t] - \sqrt{\mathrm{Var}[t]} & if\ E[t] - \sqrt{\mathrm{Var}[t]} > 0 \\ 0 & \text{otherwise} \end{cases}, \\
t_2 &= \mathrm{E}[t], \\
t_3 &= \mathrm{E}[t] + 2k\sqrt{\mathrm{Var}[t]},
\end{aligned}
\tag{2.9}
$$

where $k = max(1, \frac{\sqrt{\mathrm{Var}[t]}}{\mathrm{E}[t]})$.

Step 2: The discrete probabilities a_j's at the special points t_j's are calculated so as to give a discrete distribution with the same mean and variance, used in Step 1, as follows:

$$
\begin{aligned}
\sum_{j=1}^{3} t_j a_j &= \mathrm{E}[t], \\
\sum_{j=1}^{3} t_j^2 a_j &= \mathrm{E}[t]^2 + \mathrm{Var}[t], \\
\sum_{j=1}^{3} a_j &= 1.
\end{aligned}
\tag{2.10}
$$

The selection of the points t_j's is important for the accuracy of the approximation [**80**]. The likely reason for the success of this approach when used in an AND fork-join pattern is that the maximum of several distributions is weighted to the tails.

2.6.8. LQN Example.

Figure 2.5 shows the LQN notation with an example. The model represents clients that can read from remote replicas directly or through an intermediate replica called master replica. However, clients can only update the remote replicas by sending the updates to the master replica, which in turn sends the updates to the remote replicas. The software resources are concurrent processes shown by the parallelograms. The processor resources are shown as circles, attached to the tasks that use them.

In Figure 2.5, a task is shown as a parallelogram, containing parallelograms for its entries. The entries have directed arcs to other entries at lower layers to represent service requests. Requests may jump over layers as shown in request from activity aRc_1 (Activity Read of the Client, it has subscript number 1) to entry eR1 (Entry Read number 1). A synchronous request from one entry to another is indicated in Figure 2.5 by a solid arc with closed arrowhead. For instance, task tClient makes a request to task tMaster which then makes requests to tasks tReplica1 and tReplica2. While tasks tReplica1 and tReplica2 are servicing the requests, tasks tMaster and tClient are blocked. Alternatively, a request may be forwarded to another entry for a later response (indicated by a dashed arcs, as the forwarded requests from entry eRm to entries eR1 and eR2). Finally, an asynchronous request is indicated by an open arrowhead arc, as the request from entry eUc to entry eUm.

The parameters of an entry are the mean number of requests to lower entries, and the execution demand. For example, the mean number of requests from eUc to eUm is 1 request, and the execution demand for entry eR1 is 2 time units.

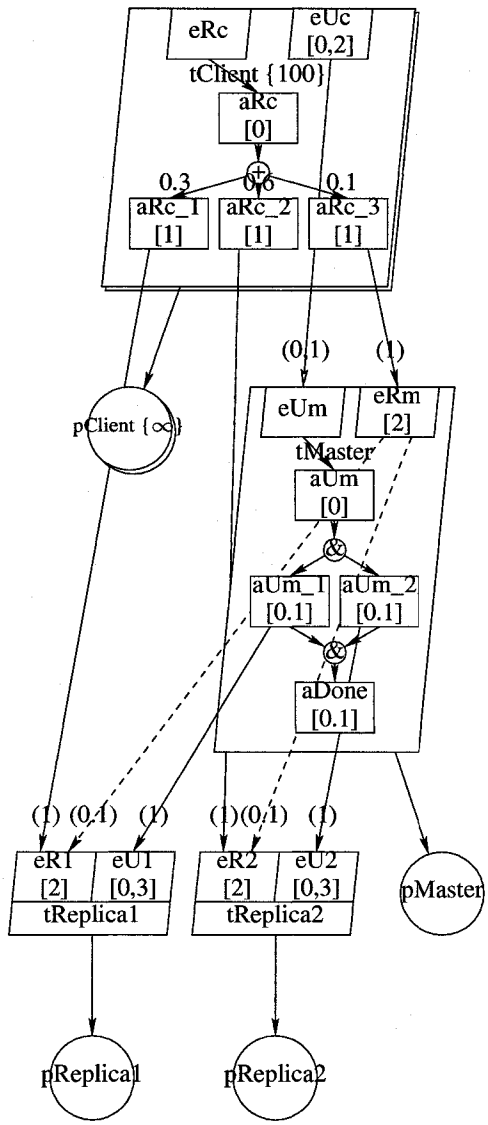


Figure 2.5: An LQN model for a four layer replicated system.

The multi-threaded task tClient, in Figure 2.5, has a multiplicity of 100. The multi-threaded task indicates a pool of clients each with a computer.

Parallel execution is shown in task tMaster where entry eUm is specified using activities. In Figure 2.5, a request that is received by entry eUm of task tMaster is processed using an activity called aUm that represents the main thread of control, after which the main thread of the task is AND-forked into two threads. One thread has the activity aUm_1, and the other thread has the activity aUm_2. These threads make requests to lower layer servers and the threads run logically in parallel, although they run on the same processor which serializes them and which may impose contention delays on both of them. When both responses from the lower layer servers are received, both threads join. Then, the main thread of control resumes execution and it executes the activity aDone. In this specific model, since entry eUm receives asynchronous requests from entry eUc, then eUm does not generate a response to eUc. If entry eUm were to generate a response, it would be indicated by a dashed line from entry eUm to entry eUc.

2.7. Layered Queueing Network Solver (LQNS)

Layered queueing considers that a software server can block itself when requesting a service from another server. In an LQN model, contention for software resources introduces layered queueing. Several tools exist today to solve layered queueing networks analytically [**164, 166, 133, 89, 55, 154, 84, 105**]. The common factor for most of these tools is that they use the method of surrogates and hierarchical decomposition to solve the model. The tools differ in the way that they partition the model and in some of their capabilities. They construct queueing submodels for clusters of servers at different layers and apply a fixed-point iteration to the submodels, to find a steady state solution for delays and resource utilizations.

The LQNS [**55**] merges the strengths of the Stochastic Rendezvous Network [**167, 164, 166**] and the Method of Layers solvers [**133, 134, 135**] to extend the modeling capability and to improve the accuracy of solutions to layered queueing networks.

The analytic approximations of the LQNS have been evaluated against simulation in [**55**], and in many applied studies such as [**146**]. The general experience is that absolute values of the percentage errors are less than 5% in throughputs and 10% in most delays, which makes the approximations useful in practice for searching a design space with many cases to be tested. Simulation is still useful for checking accuracy, and for cases where the approximations fail (for example for a system with priorities, as in [**171**]).

2.7.1. Submodel Generation.

The LQNS constructs queueing submodels for clusters of servers at different layers, and applies a fixed-point iteration between submodels to find a steady state solution for delays and utilizations of resources. The LQNS [**55**] incorporates four different layering strategies for partitioning the input model into submodels: Strict Layering, Loose Layering, Batched Layering, and Squashed Layering.

Figure 2.7 shows the submodels corresponding to the web server model in Figure 2.6. Tasks that only make requests, which normally represent users and load sources, are placed in the top layer (layer 1). Non-reference tasks, to which requests are made, are ordered by the maximum request depth to any of their entries. The request depth of a non-reference task is the distance from a task that makes requests to this non-reference task, measured in arcs. Note that a request may cross layers. The l^{th} layer submodel is created with two groups of queueing centers. There is a server center for each server task at depth l, and a source center for each client task which makes a request to any server task at depth l. A task appears as a server center in exactly one layer submodel. A task also appears as a source center in each submodel where a lower layer server, of the task, appears as a server

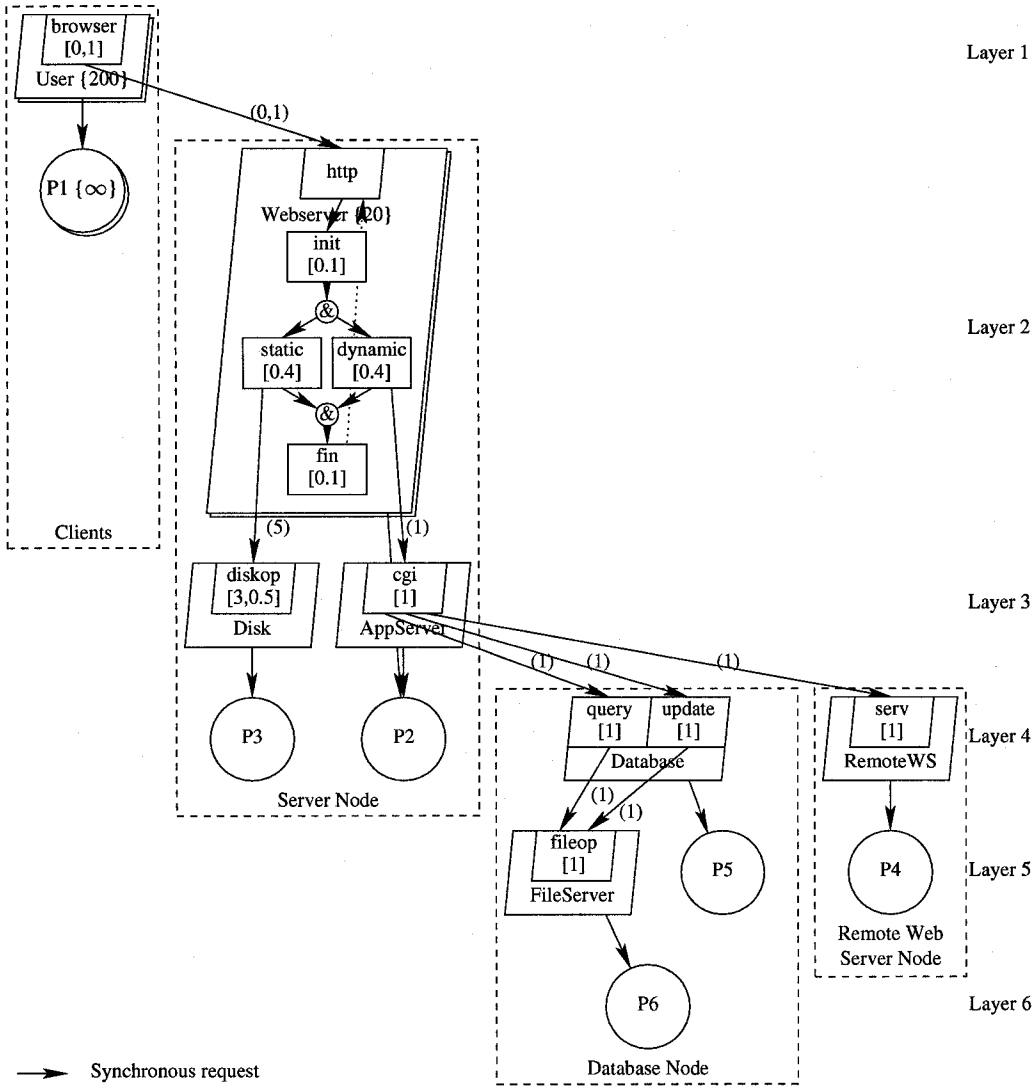


Figure 2.6: An LQN model for an application database.

center. Each source center represents the requests coming from one client task by a number of identical customers in a routing chain **[92]**. The number of customers is equal to the multiplicity (number of threads) of the client task, and their service time is the mean delay between the end of one request and beginning of the next.

Processors and other pure servers will only appear as servers in submodels. Reference tasks and other pure-clients will only appear as clients, while active servers will appear in some submodels as clients and in other submodels as servers.

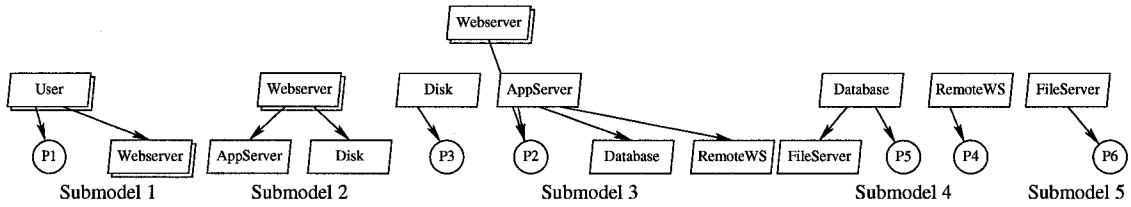


Figure 2.7: Submodels for the LQN model in Figure 2.6.

2.7.2. LQN Solution.

The LQNS solves the LQN model by using the Method of Layers [**135**] and the method of surrogate delays [**76**]. The LQNS solves an LQN model by representing each layer submodel as a conventional queueing network, as will be described in the next section, and evaluating the submodels analytically.

Within each layer submodel, the solver applies standard Mean Value Analysis approximations to solve the model, and special approximations to deal with special features. These features include non-exponential service with multi-class FIFO queues, servers with second phases and parallel threads within a service. The most complete description of these approximations is given in [**55**].

The LQNS uses the Gauss-Seidel method [**44**] or the Newton-Raphson [**44, 121**] method to iterate between solutions of submodels until a convergence, in the delay incurred at servers in other submodels, occurs. These two iterative methods are used to solve a system of nonlinear equations. The Newton-Raphson method can be used if the derivative of the function is easily calculated. If the solution converges, the method may be used to increase

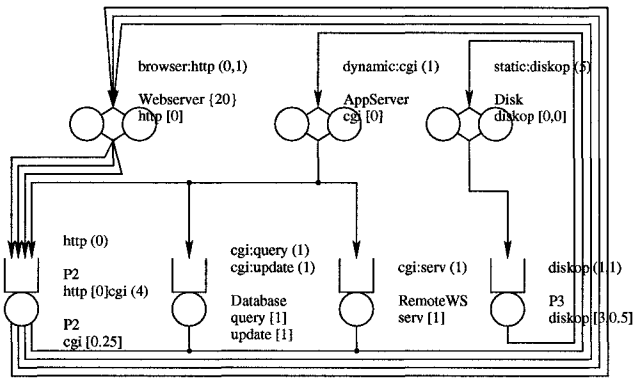


Figure 2.8: Chains constructed for submodel 3 in Figure 2.7.

the speed of convergence. Newton-Raphson sometimes converges where the Gauss-Seidel method fails.

2.7.3. Solution with Parallelism.

Parallelism within a task modifies the structure and parameters of the layer submodels. In a layer submodel, where a task with internal parallelism appears as a client, forking the main thread of control creates new threads. Forking new threads effectively adds customers to the queueing network, and joining the threads removes the customers. This fork-join behavior violates the assumptions for product form queueing networks. The LQNS uses the approach of Mak and Lundstrom **[102]** to model the additional concurrency with additional customer chains. An additional thread class is defined for each parallel subpath in the activity network within the task. In the layer submodel each thread class becomes a separate source node and customer chain, with multiplicity equal to the task multiplicity. Figure 2.8 shows the queueing network constructed for submodel number 3 in Figure 2.7.

Another example of the construction of a queueing network for a submodel is shown in Figures 2.9 and 2.10. In Figure 2.9, task tB runs activities b2 and b3 in parallel because of

the fork. Three routing chains are created: one for the main thread of control consisting of activities b1 and b4, and one each for activities b2 and b3. Each of these routing chains has one customer since there is only one replica of task tB. The corresponding queueing network for this model is shown in Figure 2.10. Routing Chain 2 corresponds to activities b1 and b4, Chain 3 to activity b2, and Chain 4 to activity b3. In Figures 2.9 and 2.10, Chain 2 cannot interfere with either Chain 3 or Chain 4 because Chain 2 is blocked waiting for the join while Chains 3 and 4 run, and Chains 3 and 4 are effectively suspended while Chain 2 runs.

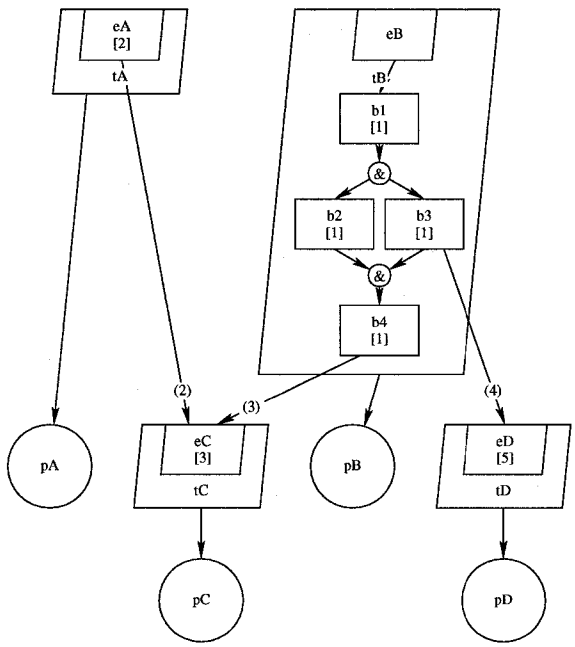


Figure 2.9: An LQN model with heterogeneous threads in task tB.

The service time at the source center is given by Eq. (2.11), but with delays and idle time calculated for each thread. The service time for chain c at the source center is the sum of delays to the client task in the LQN model, at servers (tasks or processors) which are

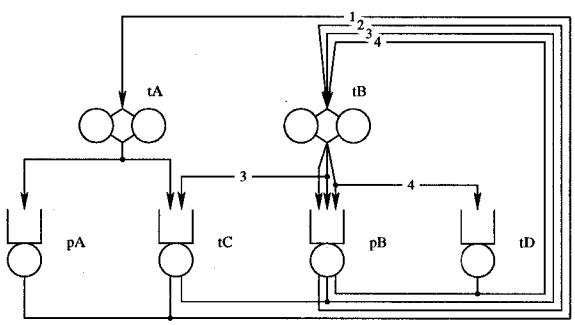


Figure 2.10: Queueing network for submodel 1 of the LQN model in Figure 2.9.

represented in other submodels. If client task t gives rise to a source center (and chain) c in layer l, the source center service time is S_c given by:

$$S_c = \sum_{l' \neq l} \sum_{e \in \mathcal{E}(l')} R_{ce} + (\text{idle time}), \tag{2.11}$$

where:

- R_{ce} = delay to task t at any entry e, per request to task m
- $\mathcal{E}(l')$ = the set of LQN entries of tasks at layer l'.

The treatment of processors as servers to the tasks allocated to them, and the "idle time" term (which accounts for delays when task t is idle) are described in [**55**].

2.8. Order Statistics

If N requests are sent to N replicas, then the time of the J^{th} response is called the quorum delay. The quorum delay X_{Quorum} is the J^{th} order statistic of the set of N thread delays $X_{Thread,1}$... $X_{Thread,N}$. The mean and variance of X_{Quorum} will later be computed from the distribution of the J^{th} order statistic of the thread delays. The quorum delay is the

subject of Chapters 5 and 6. The concept of order statistic will be described briefly in this section.

In general, let $X_1, X_2, \cdots, X_N$ be continuously distributed RVs, f_{X_i} be the probability density function (PDF) of X_i for $i = 1, \cdots, N$, and F_{X_i} be the CDF of X_i. The J^{th} order statistic $OS(J, \{X_i\}_{i=1}^N)$ of a statistical sample of size N is a RV, defined as the J^{th} smallest value out of N. The first order statistic is the minimum ($\min_i(X_i) = OS(1, \{X_i\}_{i=1}^N)$), and the N^{th} order statistic is the maximum ($\max_i(X_i) = OS(N, \{X_i\}_{i=1}^N)$).

If X_i's are independent and identically distributed (iid), then the CDF of $Y = OS(J, \{X_i\}_{i=1}^N)$ can be written as (Eq. (5) in **[137]**):

$$F_Y = \sum_{m=J}^{N} \binom{N}{m} F^m (1-F)^{N-m}, \tag{2.12}$$

where $F = F_{X_i} (1 \leq i \leq N)$. If the X_i's are independent but not identically distributed (Eq. (6) in **[137]**):

$$F_Y = \sum_{\substack{|T| \geq J \\ T \subseteq \{1,\ldots,N\}}} \left(\prod_{m \in T} F_{X_m} \right) \left(\prod_{m \notin T} (1 - F_{X_m}) \right), \tag{2.13}$$

where T is the set of indices ranging over all combinations of J or more indices chosen from $1, 2, \cdots, N$. That is, T ranges over all choices $m_1, m_2, \cdots, m_v$ such that $J \leq v \leq N$.

A recursive formula (Eq. (6a) in **[137]**) is used to compute F_Y. In this thesis, the J^{th} order statistic is found numerically, with efficiency enhanced by storing partial results and approximating the CDFs by piecewise linear functions.

CHAPTER 3

Replication Techniques in Distributed Systems

Replication maintains and allows access to copies of data and services on multiple servers. In distributed systems, replication is used to increase performance, fault-tolerance, availability, and reliability. Replication imposes a tradeoff between consistency and efficiency. The main challenge is to maintain the consistency of clients' operations on multiple replicas, while maintaining adequate response times and system throughput.

Performance, from the perspective of a client, is improved by introducing replicas. In addition, improvement is achieved by reducing the distance the data must travel to reach the client, and by having multiple sites to spread the load [**86**]. For example, replicating data items that are either read-only or read-mostly allows as many read operations to be executed in parallel as there are replicas. Further, each read operation can choose the replica from which to read, thus minimizing transmission delay. A large number of applications fall under the read-mostly category, for example, distributed name service for long-lived objects and file service for immutable objects. Furthermore, some replication protocols, such as general quorum consensus, permit a trade-off of read performance against write performance. Therefore, write-mostly replicated data can benefit from a similar performance improvement.

Replication improves performance also by exploiting the capacity of many servers. For example, Google uses replication for better request throughput [**57**]. The internal services are replicated across many servers to obtain sufficient capacity. Grid computing applications use replication to improve response times [**90, 83**]. In the USENET system, items are

posted to electronic bulletin boards across the Internet. The USENET maintains replicas of the items within or nearby the organizations that provide access to it. The domain naming service (DNS) maintains replicas of name-to-address mappings for computers and other resources. Access to services across the Internet depends on DNS. Other practical examples include web databases **[100]**.

Reliability may be improved by maintaining replicas and allowing applications to use any replica of the data. For example, replication is used in peer-to-peer content distribution [7] for improving the availability of content, and improving performance. Other examples, of using replication for reliability, include the Redundant Arrays of Inexpensive Disks (RAID) storage **[120]** and the safety critical systems like the air traffic control system **[41]**.

There are two fundamental approaches of creating replicas for reliability. The first is modular redundancy in which each component carries out the same function **[6, 101, 142]**. These replicas are called active or static replicas. The second approach involves primary/standby techniques where a primary component functions normally, periodically checking its state to the secondary replicas, which remain on standby and take over in case the primary fails **[19]**. Such standby replicas are called passive or dynamic replicas.

Replication is also used for fault-tolerance. Examples of fault-tolerant design include RAID **[120, 148]**, and the Simple Parallel Redundancy model. Another example of the usage of replication for fault-tolerance is the Triple Modular Redundancy (TMR), which is a special case of the generalization N-Modular Redundancy (NMR). NMR is a static redundancy technique in which the effect of the fault is masked instantly. In TMR, three processing modules are replicated and a majority voting is performed on the output of the modules. In addition, the Tandem Non-Stop Kernel, a popular commercial implementation

of the primary/standby technique, uses dynamic redundancy. Another example is the N-version programming model in which $N \geq 2$ versions of functionally equivalent programs are used. The N versions are developed by N independent groups. The fault in these N versions is detected using NMR with a majority voting technique.

In a mobile environment, users and their transactions move from one cell to another while accessing a database. Data is replicated to reduce the effect of forwarding the data from one cell to another. Forwarding wastes the limited communication bandwidth and slows down transaction processing because of forwarding delays. The data can be replicated on the location servers which are processors that reside in the fixed network, the home location servers of the servers, the location servers of the clients, or the mobile clients **[10]**.

Cache-based replication caches copies of content, as content is transferred through nodes in the network. Examples of cache-based replication include OceanStore, Mojonation, and Freenet **[39]**. In Freenet, when a file is located by a search request, copies of the file are cached on all intermediate nodes when the file is transferred, to the source of the request, through the network. The caching increases the availability of the file and the location characteristics for future requests. The Coda file system **[140]** is an example of a large-scale system which addresses issues of site scalability by caching complete data files at user sites for performance.

Software-based replication is typically quite expensive in terms of performance overhead **[132, 128, 119, 24]**. Multi-threading can help improve performance by exploiting concurrency in thread execution. However, multi-threaded replicas can exhibit non-deterministic behavior. This can happen in operating systems such as Unix since the thread and process scheduling are asynchronous with replica execution.

In most environments, static or user-driven replication services are used to replicate files. However, in a grid environment, the resource and network performance change dynamically. So, there is a need for a strategy that determines the replication level automatically. In **[90]**, a replication strategy is used to place subsets of the data replicas based on user data requests. Further, multiple replica servers are used to improve data availability. In **[129]**, a strategy is proposed to create replicas automatically in a decentralized Peer-to-Peer network. Replicas are created in a given site to guarantee a minimum availability requirement.

Dynamic replica management algorithms are designed to achieve both client response time and server workloads. They are used in the Scalable Content Access Network (SCAN) system **[32]** to dynamically place a minimum number of replicas that meets the quality of service requirements and server capacity constraints.

In the remaining of this chapter, an introduction to the transaction concurrency control protocols in replication systems is provided. In addition, replication techniques in parallel and distributed systems are discussed. Further, a literature review of prior work on modeling replication techniques using queueing networks is presented. Furthermore, relevant techniques to exploit symmetry in a replication system are discussed.

3.1. Transaction Concurrency Control

A logical unit of work, a software transaction **[173]**, is a sequence of one or more data manipulation statements that together form a unit of recovery. Transaction atomicity guarantees that a transaction must execute successfully on all-or-none of the sites, irrespective of the possibility of site or communication failures. A software transaction is said to be committed when all of its processing has been completed successfully.

Concurrency control protocols are used to limit concurrent transaction executions, in a centralized and distributed database system, to executions that are serializable. Serializability ensures that the execution of concurrent transactions is synchronized to make the execution equivalent (produces the same output and has the same impact) to a serial execution over the logical copy. The strongest correctness criterion for a replicated system is the one-copy-serializability. An object, with multiple copies, appears as one logical copy (also called one-copy-equivalence) and the execution of concurrent transactions is serializable over the logical copy. To procure the one-copy-serializability criterion, the physical copies of a logical data item must reflect the operations performed on it. In addition, the most current state of the replicas must always be presented despite failures.

Concurrency control protocols fall into one of two groups, pessimistic or optimistic. Pessimistic protocols, such as the two-phase locking protocol [**49**], prohibit potentially non-serializable executions. This prevents inconsistencies, because the effects of a committed transaction need not be revoked. The main principle in pessimistic designs is to prohibit access to data unless it is up-to-date by limiting availability. Each replica assumes that if an inconsistency can occur, it will occur. Optimistic replication protocols [**88, 155**] achieve high levels of availability and efficiency by letting users read or write any data at any time, but at the risk of inconsistent states. These protocols permit non-serializable executions to occur, with inconsistencies detected - and the relevant transaction aborted - during a validation phase [**88**] before the effects of the transactions are made visible. Replicas may find inconsistencies when trying to order the updates, thus, automatic conflict resolution protocols are applied to find an agreement. Finally, replicas commit the updates and the replica contents become stable. Optimistic strategies differ primarily in how they detect and solve inconsistencies and in the way in which they propagate updates.

The Tunable Availability and Consistency Tradeoff (TACT) replicated service **[66, 170]** attempts to give applications the means to bound inconsistency. TACT allows applications to dynamically choose their own consistency/availability tradeoffs. The system uses a set of metrics to depict the whole consistency spectrum. The metrics provide information about the current change of a given replica and can be used to control update propagation. Conversely, it is not clear how these architectures can affect the overall performance of systems in wide-area or peer to peer storage systems where scaling is an important factor.

3.2. Replication Techniques

Gray et al. **[62]** classify replica control protocols based on two parameters. The first parameter is when updates are propagated between the replicas, within transaction boundaries or after the transaction is committed. The second one is the choice of replicas that can be updated.

In an eager replication, a synchronous replication model, conflicts can be detected before the transaction commits. Early detection simplifies providing data consistency, but at the expense of increasing response times due to communication overhead. In a lazy replication **[75, 117, 157]**, an asynchronous replication model, the response times are minimized by postponing the propagation of modifications until the end of the transaction. However, inconsistencies of data among replicas might occur.

There are two possibilities for the case of which replicas to update. The first is a primary copy (centralized) approach, in which all updates on a certain data item are performed at the primary copy of the data item. After that, updates are propagated to the secondary copies. This primary copy approach simplifies concurrency control, because concurrent updates to different copies are prevented. However, this may cause the primary replica to be a bottleneck. The second possibility is an update everywhere (distributed) approach,

in which any copy of a data item can be updated. The updates have to be coordinated to all copies. In the eager approach, this coordination leads to communication overhead that increases the transaction execution time. In the lazy approach, inconsistency of data occurs when two transactions update two copies of a data item and both commit the updates locally before propagating the updates. The inconsistency must be detected and reconciled.

Replication techniques can be divided into the following categories: data replication, service replication, process replication, object replication, thread replication, and message replication. The replication techniques are classified in Chapter 4 based on both the server architecture **[160]** (which is where the transactions are executed), and the behaviour of the server.

From a performance perspective, different replication technique categories might be equivalent, as a performance model captures the message communication patterns irrespective of the granularity of the data. For example, both data replication and message replication can use a quorum protocol. The replication set (the set of data or services that are replicated) in a quorum data replication protocol can be the whole database, while the replication set in a quorum message replication protocol is just a message. However, both protocols follow the same sequence of interactions to achieve consistency, which is the quorum protocol. Should the designer be willing to capture the data granularity in a performance model, he is required to increase or decrease the execution demand of that replica of certain hardware or software resource. The concepts developed in this thesis can be applied to any hardware or software devices that can be replicated, and modeled as a task in the LQN formalism.

Data replication is commonly used in distributed systems to provide better performance and availability **[31, 161]**. In a fully replicated database system each site stores a copy of the database. A partial replication occurs when each site stores parts of the database.

Bestavros and Braoudakis [**16**] use process redundancy to improve performance in Real-Time Database Systems (RTDBS). RTDBS guarantee that in the rare event of a highly loaded system, critical transactions meet their deadlines. Usually, meeting the deadline objective requires significantly extra resources than resources required to keep average loads.

Creating replicas of processes can follow either a symmetrical technique, in which all replicas are active and perform identical functions, or a primary/standby technique where a single active replica performs the computation, and one or more standby replicas remain inactive as long as the primary replica does not fail.

In distributed object oriented systems, components of the composite object may be distributed across multiple servers. Creating replicas of a composite object improves the availability of objects and provides efficient reads, but this replication could be costly in update and storage. The high cost of update and storage is overcome by making the creation of replicas selective [**17**]. In distributed relational database systems, the relations are fragmented, then the horizontal and/or vertical fragments are replicated [**21, 50, 111**]. In object oriented database systems, selective creation of replicas is provided using the reference attributes of the object. Reference attributes can reference object replicas [**147**] or procedures [**79**].

The authors in [**112**] classify the replication models in distributed databases into the following models. The first is an All Objects to All Sites model as in [**94, 151**], the second is an All Objects to Some Sites model as in [**27**], and the third is a Some Objects to All Sites model as in [**152**]. In addition, the authors in [**112**] introduce a Some Objects to Some Sites model, and a Replication per Object [**107**] model.

Message replication is usually used in managing data or process replication. Message replicas are produced and exchanged to all or some of the replicas. The first form of message replication is to generate the necessary message replicas by implementing the

data or process replication techniques [**68**]. The second form of message replication is the reliable multicast group of message replication [**18, 33, 82**]. The third form of message replication is the quorum multicast suite of protocols introduced in [**60**] which pushes a significant part of replication management into an above-kernel, high-level communication subsystem. In these protocols, the one-copy semantic of data is maintained by the exchange of a necessary and minimum number of messages. Multicast protocols can be classified into FIFO [**28, 141**], casual order [**65**], and total order multicast protocols [**108, 83**].

Multi-threaded applications can benefit from replica creation. Solutions to replicate multi-threaded applications or objects may be based on a non-preemptive deterministic scheduler. The scheduler enforces the same thread interleaving, one physical thread is scheduled at a time, on all replicas to achieve determinism in replica state updates. Deterministic scheduling leads to poor scalability and performance of the replicated system but guarantees consistency among replicas. In the Eternal system [**109**], a component-based framework for transparent fault-tolerant CORBA, determinism is achieved by processing application threads sequentially, which is effectively a single-threaded solution [**109**]. Each thread serves a client request. In Transactional Drago [**81**], a programming language for building distributed transactional applications, the executions of multiple logical threads are interleaved if the running thread starts an I/O operation. The thread is suspended to wait for I/O to complete while another thread may be scheduled. The work in [**14**] enforces strong replica consistency without the need for the same thread interleaving on all replicas. Simultaneous multi-threading (SMT) [**144**] uses simultaneous and redundant multi-threads (SRT) [**158**] to detect transient faults. A program is replicated into communicating leading and trailing threads. Trailing threads repeat computations, and the results are compared. Only the leading thread accesses cache. Multi-threading replication can also be applied to web services as in [**169**].

Increasing parallelism can have a considerable impact on performance measures. Additional threads consume resources such as memory, disk I/O, network bandwidth, and database connections. Furthermore, additional threads may cause significant overhead from contention or from context switching. However, the following are cases where performing multiple tasks in parallel might be appropriate [**104**]:

- If there is no dependency among tasks. A task can run without waiting for results from other tasks.
- If work is I/O bound: a task making I/O operations can benefit from having its own thread, because while the thread is suspended other threads can execute.

In addition to limits on device utilizations, there may also be constraints on the software components as well. These constraints may come from different sources [**96**]:

- Some software entities cannot be replicated, critical sections, for example. So, software queueing delays will occur. The existence of a critical section does not depend on the process or object distribution, but can be influenced by them.
- Replication level of software components is limited by design or implementation constraints. For example, a target system may only have the license for a positive integer of instances of a database server process subsystem. Alternatively, introducing additional instances of a database into a system may require significant modifications to its application software. In this case, a replication limit would be one replica.
- There can be operating system constraints on threading levels of processes, even though those processes can be multi-threaded. For example, constraints can limit the number of file descriptors used by a server process's connections to 64, hence the threading limit is 64.

3.3. Modeling Replication in Queueing Networks

Simulation and analytic performance studies of distributed databases commonly use queueing networks as the underlying models. A survey of queueing models for replication can be found in [**112**]. A fully replicated database of m local replicas has been modeled by an M/M/m/FIFO queueing model [**40, 110**]. In an M/M/m/FIFO queueing model, transactions have Poisson arrival times and exponentially distributed services times and are served by m servers according to a First Come First Serve scheduling discipline. The shared read operations are modeled by having the m servers process the read transactions in parallel, while exclusive write operations are modeled by allocating all m servers to write transactions during their service time [**112**].

In [**9**], an M/M/m queueing model with service interruptions is used to make write operations have preemptive priority over read operations. An M/M/1 is used to model the arrival and service of updates at which the preemption occurs. In [**110**] non-preemptive write operations are assumed. The authors in [**110**] compare the performance of updating the replicas in parallel or in sequence. The weaknesses of these queueing models are that they ignore communication delays among replicas, transactions from all replicas arrive at a single queue, and full replication is assumed [**112**]. To overcome these weaknesses, distributed databases can be modeled by queueing networks [**85**].

The studies in [**37, 38, 73, 103**] construct a queueing network with every local database modeled as an M/M/1 queueing model. In an M/M/1 queueing model, the execution demands of all transactions are exponentially distributed with same mean. Generally distributed execution demands are used to model the local databases using an M/G/1/FIFO queueing model in [**13**], and using an M/G/1/RR queueing model in [**138**]. Constant execution demands for the lock request are used in [**20**] to analyze the distributed database at the level of lock requests. Each replica is modeled as an M/D/1/FIFO queueing model.

Read-only transactions and updates are assigned different exponentially distributed execution demands using networks of M/H2/1 queueing models with two-phase hyperexponentially distributed execution demands [**56, 94**]. However, such queueing network models do not allow more than two types of transactions [**112**].

Communication models are based on different assumptions. In [**22, 131**], it is assumed that the local transaction processing times are negligible compared to communication delays. In [**35, 107**], it is assumed that communication delays are negligible compared to database execution demands. In [**153**], the performance of a distributed system is found by using a simplified conventional queueing model. Each replica is modeled as a server with a single FIFO queue and two transaction types are used. The communication delay is modeled using a delay center. However, it is assumed that messages among replicas are asynchronous. The authors in [**136**] model different replication techniques, such as, Read-Any-Write-All (RAWA), weighted voting, and demand replication, by modeling the network as a simple Markov chain.

The work in [**95**] finds the maximum level of useful replication or threading of a process. Creating replicas beyond that level does not improve the user response time, because the replicated process will not be the bottleneck anymore. The maximum level is found by solving the layered model analytically, based on the Method of Layers. The replication level is estimated with respect to given software and device utilization limits. When reaching the maximum replication level, more customers will be accepted, but those extra customers might make another system resource a bottleneck. The algorithm dynamically decides the number of replicas needed.

Figure 3.1 shows an example of using replication in LQN models. The Figure shows an LQN model, taken from [**124**], for the Handle Driven Object Request Broker (H-ORB) architecture in a CORBA middleware system used to access pools of replicated servers.

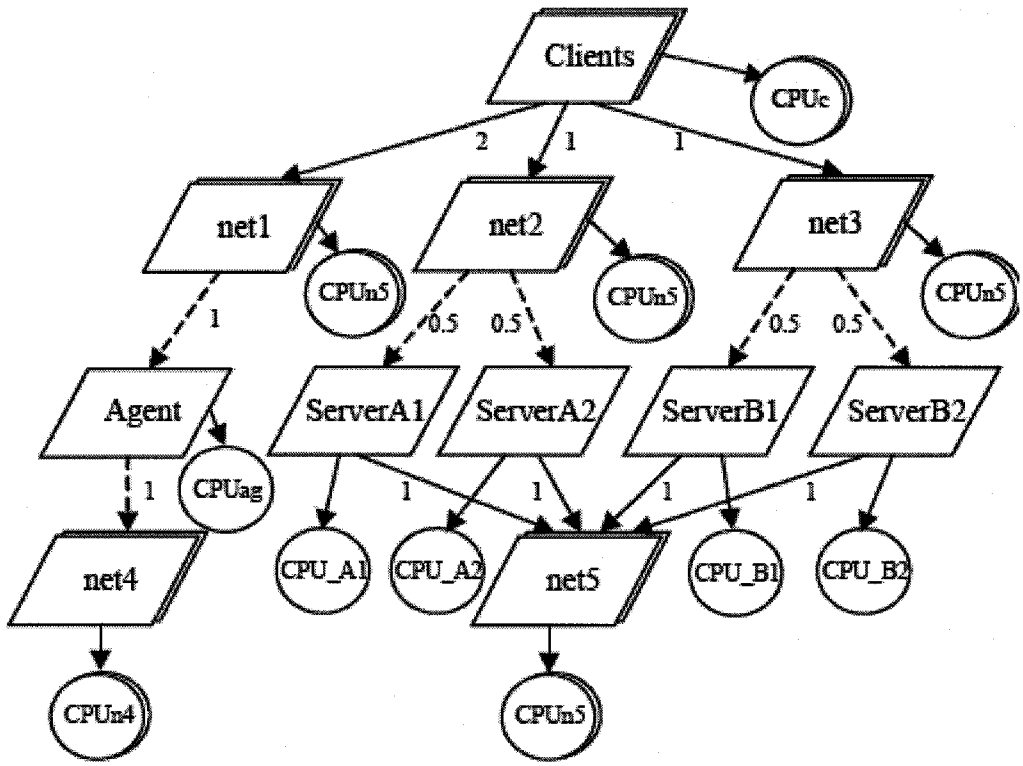


Figure 3.1: An LQN model for the H-ORB architecture [**124**].

The top node named Clients comprises the reference tasks driving the system. Each client loops infinitely and sends synchronous requests to the lower layer. This system shows two replicas each of ServerA and ServerB, whereas there might be many more of ServerA and ServerB. Each server has its own queue.

3.4. Symmetry Exploitation

Symmetry has been exploited in different ways, particularly in state-based models. Sanders and Meyer devised algorithms to identify symmetrical states in Markov models, and do exact state aggregation [**139**]. State lumping is an approach that reduces the size of a Continuous Time Markov Chain (CTMC) by considering the quotient of the CTMC with

respect to an equivalence relation that preserves the Markov property and many performance measures defined on the CTMC. Since the computation of that equivalence relation for a large CTMC is costly in space and time, most practical lumping approaches identify appropriate lumping by working on a higher-level formalism, rather than by constructing the unlumped CTMC and then operating on it.

Performance analysis methods for Stochastic Petri Net (SPN) models are usually based on constructing the whole state space, and deriving and analyzing a Markov chain. The crucial problem of these methods is the state space explosion. Stochastic Well-formed Colored Nets (SWNs) [**36**], an extension of SPN, overcome this problem by allowing the exploitation of the symmetry properties of the models [**48**]. From a SWN model an aggregated Reachability Graph can be built, called Symbolic Reachability Graph (SRG). From SRG, a lumped Markov chain can be directly derived. The reduction induced by the SRG may be poor in partially symmetrical models, because SRG does not exploit all possibilities for state space reduction. The Extended Symbolic Reachability Graph (ESRG) [**64**] was introduced to provide a more compact representation. The cost of ESRG is a loss of information that may affect the possibility of generating a lumped Markov chain directly from the ESRG.

In [**46**], the authors develop a representation of the CTMC of a hierarchical composed model that is built on shared state variables (SVs) among submodels. A hierarchical model is built using the replicate operator which imposes symmetries in a way that allows an associated CTMC to be lumped.

Petriu improves the accuracy of the estimate of the arrival instant probabilities in the SRVN formalism through a technique called "*Task Directed Aggregation*" (TDA) [**127**]. The TDA technique uses a Markov submodel to derive the in-service and in-queue probabilities. In [**51**], the concept of aggregating the states of symmetrical tasks into one state in

the Markov chain is presented. The symmetry of the tasks is used to reduce the complexity of the TDA technique by computing identical values only once.

The authors in [**123**] generalize the basic ideas in [**125, 122, 126**] by developing the Markov chain Aggregation by a Composition (MAC) method. MAC creates the submodel by composing the behavior of the components to be observed. The behavior of the indistinguishable components are "hidden" inside the aggregated states. MAC does not generate and aggregate the entire state space. Conversely, other composition-based methods hide internal component behavior, as in [**59, 70**]. MAC uses composition methods taken from other work in compositional modeling such as [**71**].

In [**23**], a technique that automatically computes classes of symmetric submodels in a multi-processor architecture is presented. Most multi-processor systems are partially symmetric. Symmetric permutations on the state level are derived based on the information about symmetric submodels. Exploration of these permutations allows the aggregation of the states of the CTMC. The exploration results in a smaller CTMC that can be used to compute exact performance measures for the complete symmetry with lower cost than the full model.

To study large client-server systems, greater than 100 tasks, a group of client tasks or a group of server tasks may have similar performance parameters and thus similar performance measures. In [**118**], a simplified notation for replication [**165**] is developed. In addition, a technique to expedite the solution time of a model that contains replicated tasks is implemented. The result of the work in [**118**] shows that the replication algorithm works best with the Schweitzer approximation with an absolute value of the percentage error of less than 5%, while the replication algorithm using the exact MVA and the Linearizer provide high errors of up to 12%.

Sheikh et al. in [**145**] describe a replication technique based on replicating *areas* in the model. An area can represent sets of processes and devices that describe a server subsystem, a type of node in a network, a network of nodes, and so on. Their work reduces the size of the LQN model description. The MVA residence time and throughput expressions are modified to exploit the symmetry in the LQN model, so fast performance estimates could be provided for even very large systems. Their solution is limited compared to the solution presented in this thesis, and in [**116**], in two ways. First, it assumes that a replicated task (area) can have at most one parent. Second, it does not consider the case when a client makes requests to more than one instance of a replicated task. Conversely, the work in [**116**], generalizes the work in [**145**] in two useful ways. First, it allows a replicated task (area) to have more than one parent. Second, it allows a client to spread its requests across several replicas, rather than using just one of the replicas.

CHAPTER 4

Performance Replication Patterns

One usage of replication in parallel and distributed systems is to enhance performance. In this chapter, various performance replication patterns in parallel and distributed systems are identified and classified. Unified Modeling Language (UML) sequence diagrams and LQN models are developed when possible. Furthermore, extensions to the LQN formalism, and to the LQNS to model these replication patterns are identified. These extensions improve the speed, scalability, accuracy, and expressiveness that the current LQN and the LQNS provide.

4.1. Performance Patterns

In this section, performance patterns are identified, UML sequence diagrams are developed to show the interaction among clients and replica servers, and LQN models for the patterns are developed when possible. Performance patterns capture the message communication required to access information and keep information consistent. Therefore, more than one replication technique may fall under one performance pattern.

In the LQN models constructed in this chapter, entry eRx represents a read entry of a task, while eUx represents an update entry of a task where the subscript x refers to the task that holds the entries. Similarly, aRx represents a read activity, and aUx represents an update activity in a task.

4.1.1. Master-Slave Performance Pattern.

In the master-slave pattern, a client can only update the master replica after which the master replica propagates the update to other replicas after replying to the client. A client can read from any replica including the master. Figure 4.1 shows the UML sequence diagram for the master-slave pattern. In the Figure, the update from the tMaster replica to other replicas occur in parallel. However, sequential updating is also possible in other implementations.

In the master-slave pattern, also called *primary copy* Read-One-Write-All (ROWA) in [**68**], at any one time every individual piece of data has only one primary source. All update activities for an individual piece of data always occur against only one replica at any particular time. Copies of data are sent asynchronously to updating the primary source [**25**]. This means there is some degree of latency before data consistency is achieved at any replica.

In addition, the master data is seen as having priority over the other replicas [**156**]. That is, if any modifications have been made to any of the slave replicas since the last replication, these can be overwritten by the next replication from the master to the slaves. However, there are cases in which slave modifications persist after a replication; for example, when replication only adds to the slave data, rather than updating or replacing it.

The LQN model for this pattern is shown in Figure 4.2. The master can update all replicas either sequentially or in parallel depending on the implementation. Task tClient1 has one read entry eRc1 and one update entry eUc1, while tClient2 has aRc2 read activity that forks into three other read activities aRc2_1, aRc2_2, and aRc2_3.

4.1.2. Master-Slave Snapshot Performance Pattern.

In the master-slave snapshot pattern, a client updates only a master replica. The master propagates the updates to other replicas periodically in a predefined time; this is known as

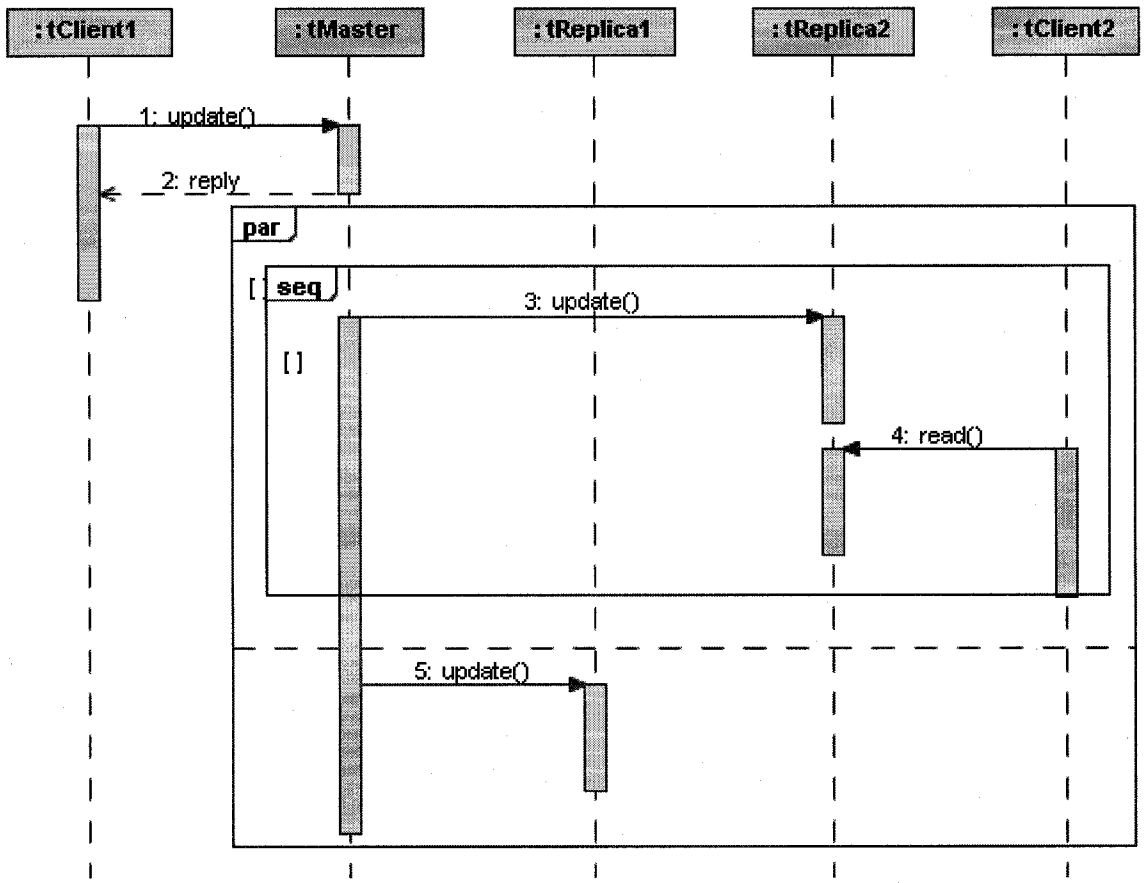


Figure 4.1: Sequence diagram for the master-slave performance pattern.

a snapshot. The periodic updates are performed to ensure that the replicas are consistent at a given time. A client can read from any replica including the master at any time. Figure 4.3 shows the UML sequence diagram for the master-slave snapshot pattern.

In this pattern, the replication set consists of entire rows, not just the modifications made since the last replication [**156**]. Target replicas can be updated by their local applications, but the modifications are overwritten in later replications. Hence, the snapshot replication is a master-slave replication.

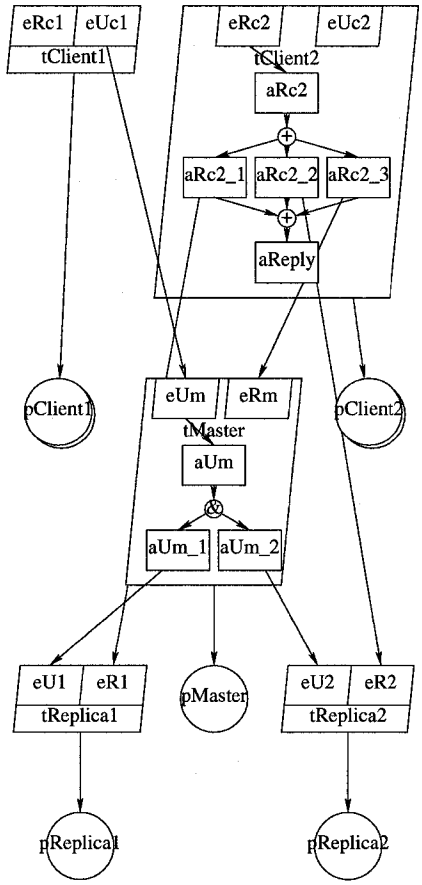


Figure 4.2: An LQN for the master-slave performance pattern, the master updates the replicas in parallel.

The LQN model for a master-slave snapshot performance pattern is shown in Figure 4.4. The periodic propagation of updates from the master replica to all other replicas is modeled as an open arrival with a predefined arrival rate. Figure 4.4 shows that the master replica updates the other replicas in parallel.

4.1.3. Master-Slave Cascading Performance Pattern.

In the master-slave cascading pattern [**156**], a client can update only a master replica. The master replica propagates its updates to an intermediate replica. Then, the intermediate replica propagates its updates asynchronously to all other replicas. A client can read from

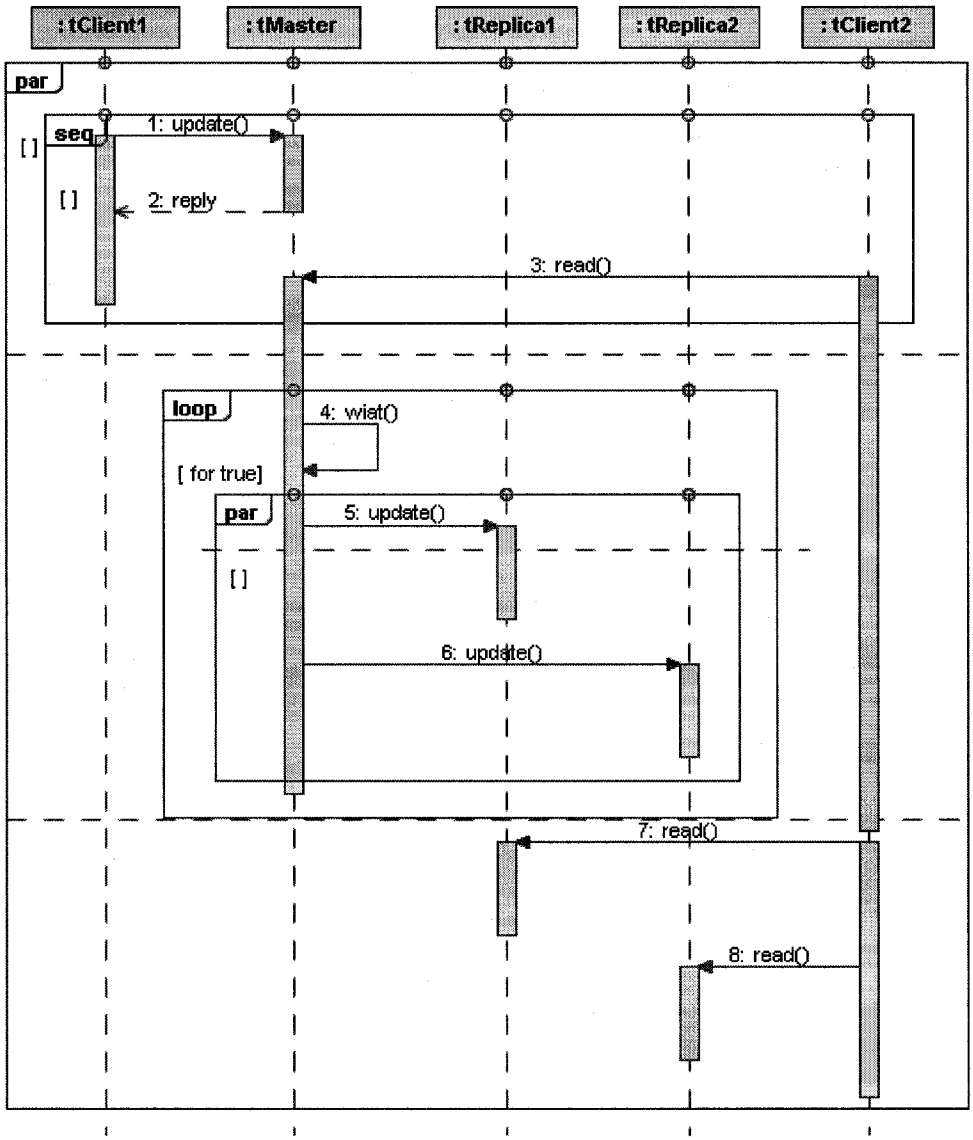


Figure 4.3: Sequence diagram for the master-slave snapshot performance pattern.

any replica, except the intermediate replica. Figure 4.5 shows the sequence diagram for the master-slave cascading pattern. The LQN model for the master-salve cascading pattern is shown in Figure 4.6.

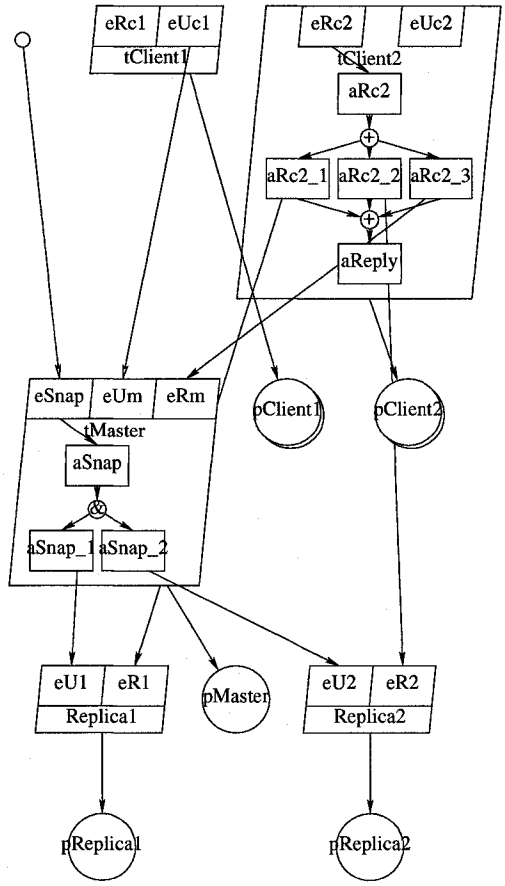


Figure 4.4: An LQN for the master-slave snapshot performance pattern.

A replication set is replicated in a one-way direction from one source to many targets. The replicated data in the targets is read-only, or it can be updated at the targets, but these updates are overwritten by later transmissions [**156**]. There is no need for conflict detection or conflict resolution because of target modifications.

Additional intermediary replicas can be added to increase the overall availability of the system, but at the expense of increasing the performance overhead. This arrangement adds the concept of a cascade intermediary target and source to the pattern.

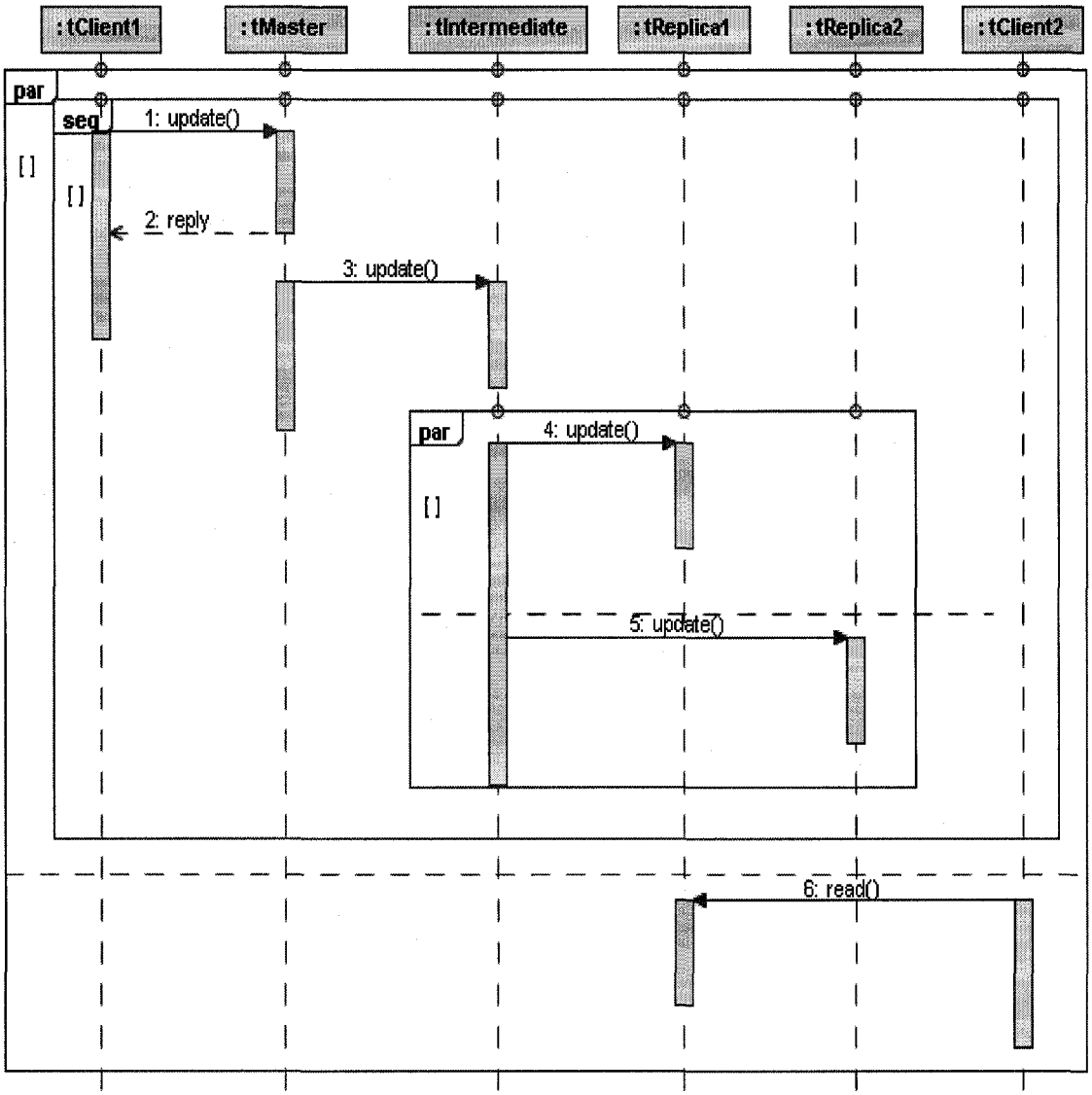


Figure 4.5: Sequence diagram for the master-slave cascading performance pattern.

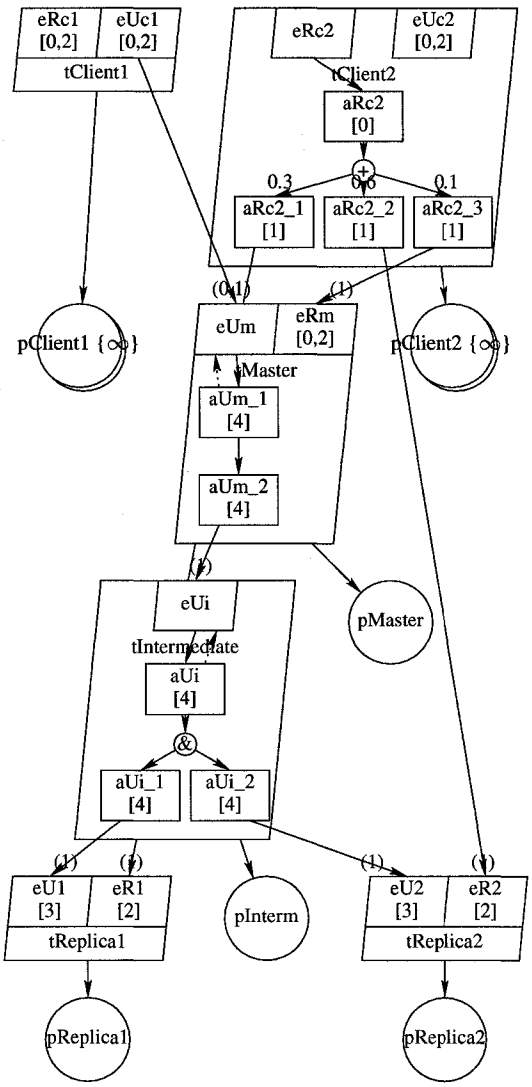


Figure 4.6: An LQN model for the master-slave cascading performance pattern.

4.1.4. Consolidation Performance Pattern.

In the consolidation pattern, normally replicas have primary fragments of data [**25**]. A client updates the master replica for that particular fragment. The client cannot update the consolidated replica. After a client updates a replica, the replica asynchronously propagates its updates to the consolidated replica. A client can read from any replica including the consolidated one. In that case, a client may be required to read multiple replicas in order to fetch a data set.

Figure 4.8 shows the LQN model for the consolidation performance pattern. In this LQN model, when a client updates a replica, the replica processes the update and responds to the client first, then it propagates its updates to the consolidation replica. Other equivalent LQN models can be used by replacing the activities in tReplica1 and tReplica2 by phases. The first phase processes the client's request and replies to it. The second phase propagates the updates to the consolidation copy. In addition, other possible implementations can lead to other LQN models by separating the update entry in the consolidated task into activities or phases.

4.1.5. Read-One-Write-All Performance Pattern.

In the read-one-write-all pattern, a client updates all replicas, but needs to read only one replica, as shown in the sequence diagram in Figure 4.9.

Figure 4.10 shows the LQN model for the read-one-write-all performance replication pattern. The update entries eU1 in tasks tReplica1 and eU2 in task tReplica2 can also be separated into activities, or phases to reply to the client earlier.

The Simple Read One Write All (ROWA) technique [**68**] executes a read operation only on a single replica. A write operation to a single replica is translated into writes to all replicas. The underlying concurrency controller at each location synchronizes access to

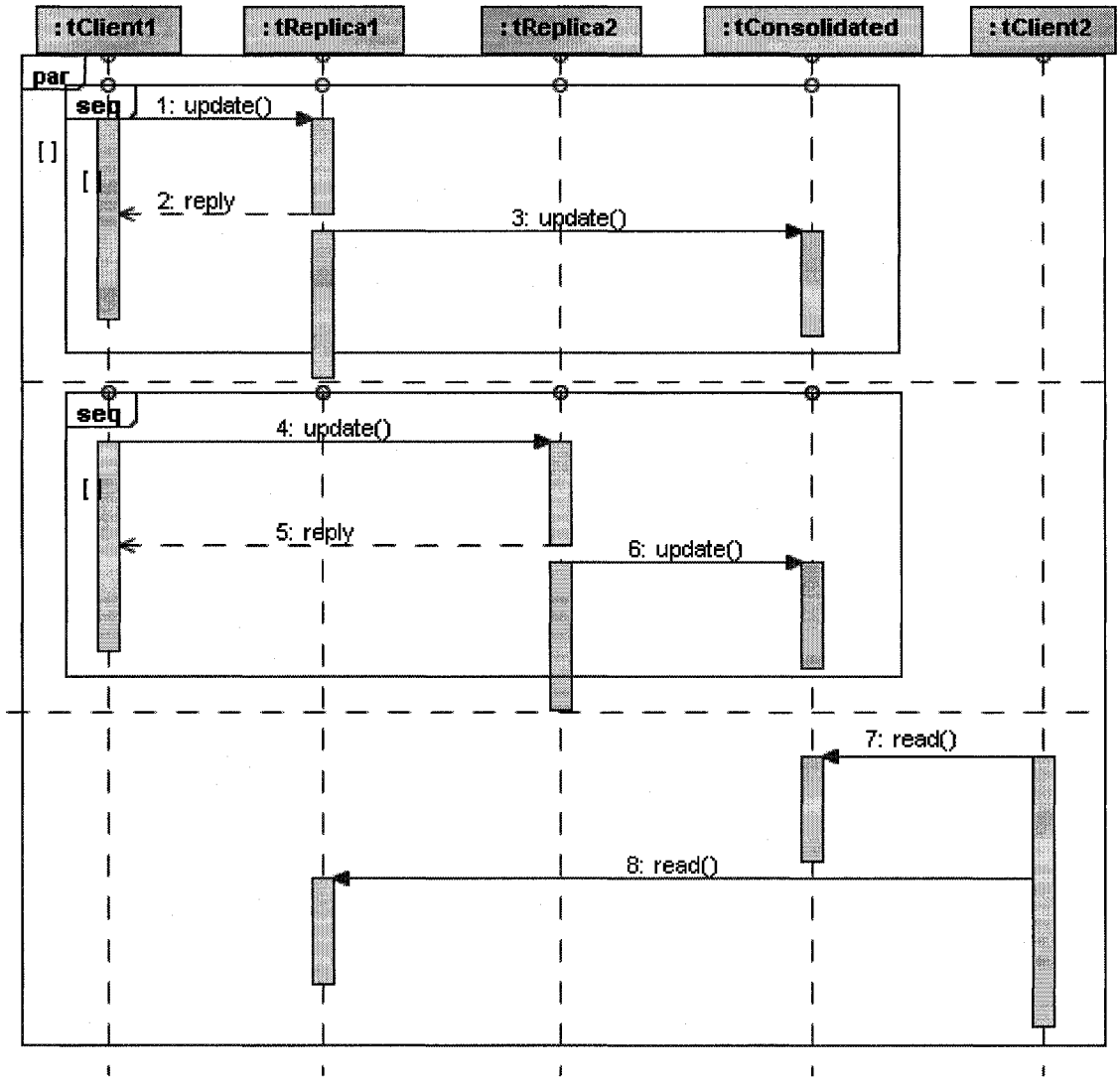


Figure 4.7: Sequence diagram for the consolidation performance pattern.

replicas. Hence, this execution is equivalent to a serial execution where each transaction that updates a data item will update either all copies or none.

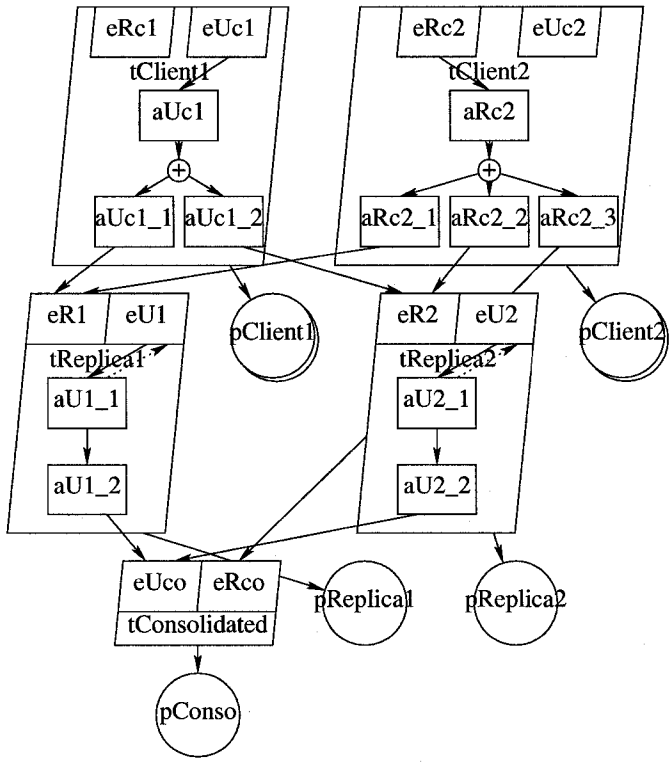


Figure 4.8: An LQN model for the consolidation performance pattern.

4.1.6. Master-Master Performance Pattern.

In master-master replication [**156**], also called update-anywhere model [**25**], a client can read any replica. After a client updates a replica, the updated replica propagates its updates to all other replicas. The source replica is called the master for the updated data. Updates can arrive at two replicas of a data item simultaneously. For example, consider laptops that work off-line and make data modifications. When the laptops are online, they need to synchronize the modifications with a shared server database that possibly has been updated by other applications [**156**].

Figure 4.11 shows the sequence diagram for the master-master replication pattern. The LQN model for this pattern is shown in Figure 4.12.

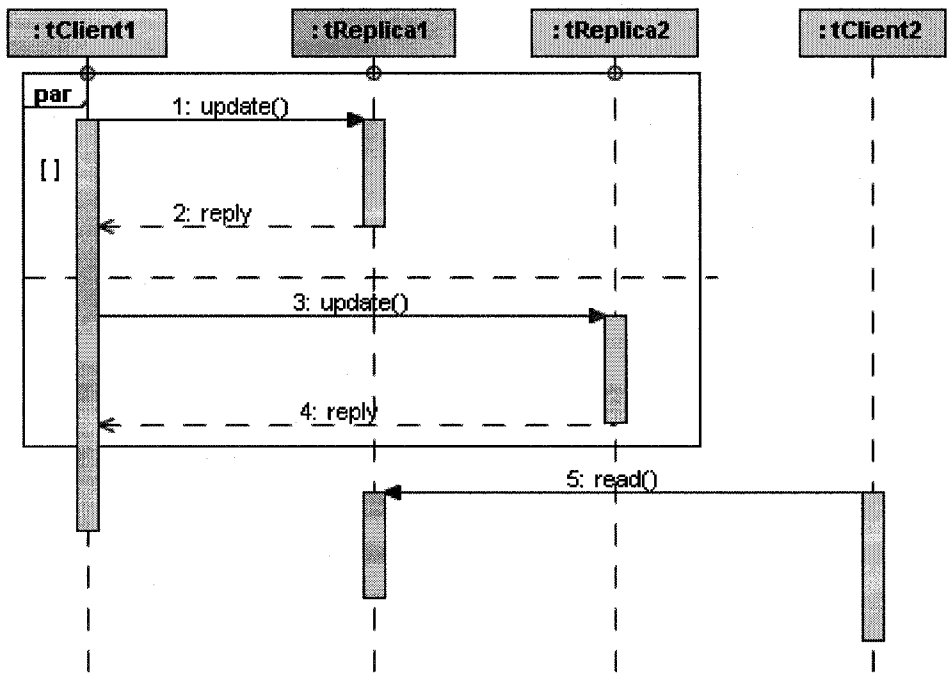


Figure 4.9: Sequence diagram for the read-one-write-all performance pattern.

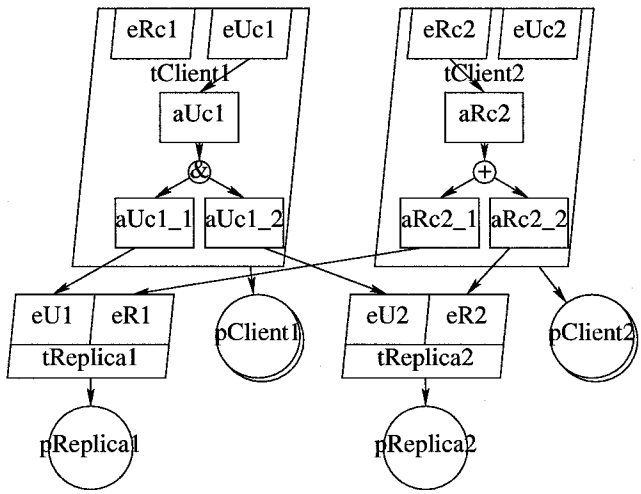


Figure 4.10: An LQN model for the read-one-write-all performance pattern.

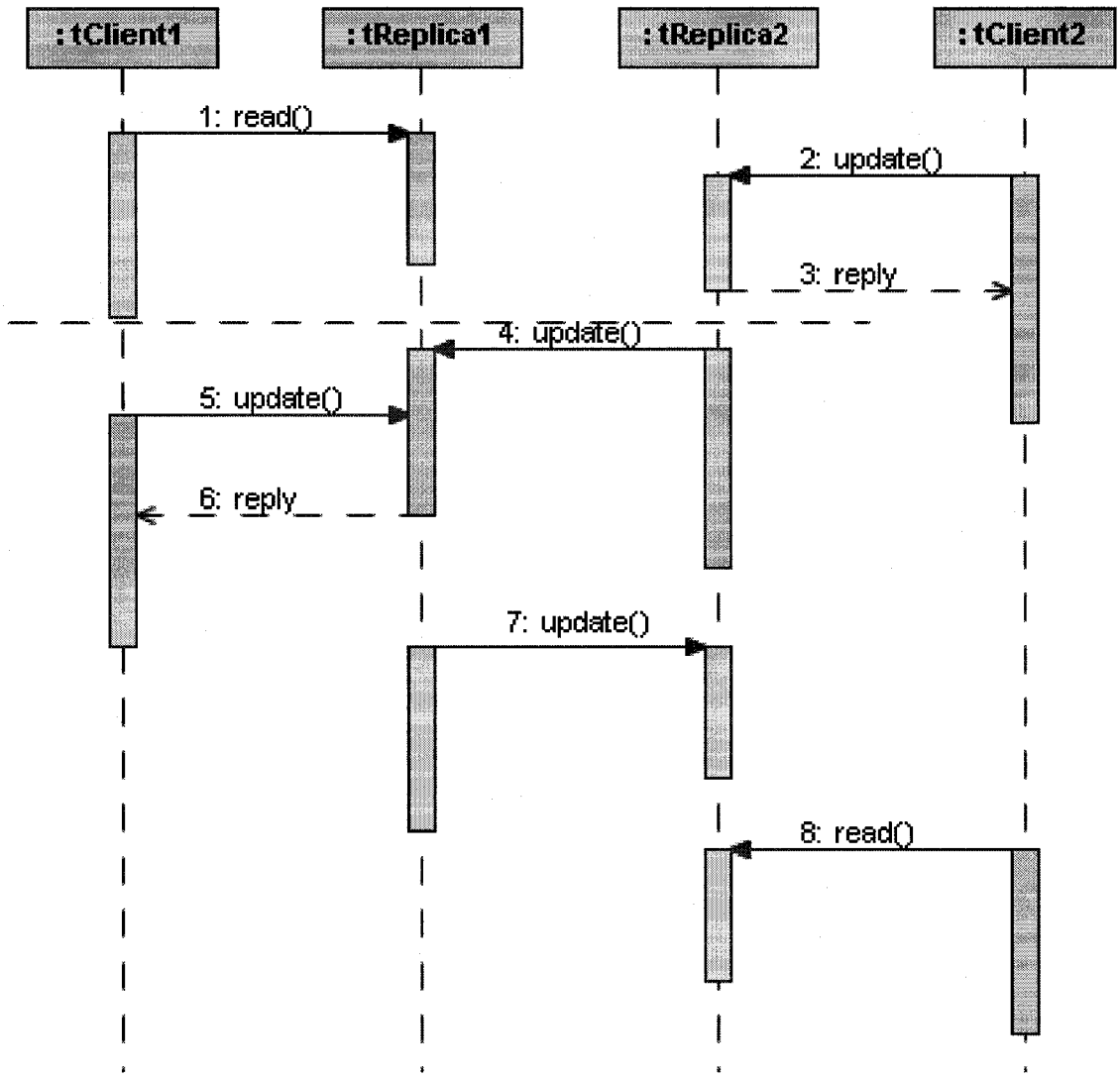


Figure 4.11: Sequence diagram for the master-master performance pattern.

In a master-master data movement [**156**], a source replica makes a redundant copy of the data in a target replica. If the target updates the copied data, it sends the modifications back to the source, to keep the source and target synchronized. Update conflicts that have occurred since the last replication are detected and resolved.

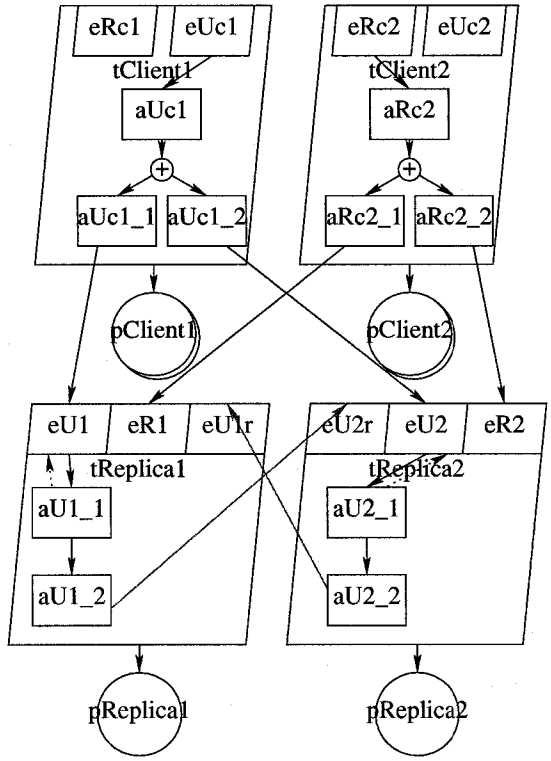


Figure 4.12: An LQN model for the master-master performance pattern.

A derived pattern from the master-master replication occurs when each data unit is assigned to a specific master to which all writes for that data unit must be performed. Then, that master replicates the updates to the other replicas. The result is that each data location has a complete replica, in which the data location updates only its particular primary data unit. This is called master-slave with distributed primary fragments in [**25**].

Other patterns based on master-master replication are introduced in [**156**] for the cases when potential conflicts in the modifications are to be resolved at the row level of a database table. This is called master-master row-level synchronization [**156**].

4.1.7. Quorum Performance Pattern.

In a quorum pattern, a client selects a quorum (a certain number of replicas) to update or read. An operation is allowed to proceed to completion when a subset of nodes responds to requests made by it. For example, in Figure 4.13, after the client sends three requests to three independent replicas, it waits only for the first two responses. The quorum is preset to two. After the client receives the two responses, it resumes execution. All other delayed responses that are received after the quorum is complete are ignored in this scenario of the sequence diagram. Different semantics for the delayed responses based on other scenarios are also possible and will be discussed in Section 6.1. In Figure 4.13, the response (message number 6) that occurs after the quorum action is ignored; the ignored response does not affect the performance. When the quorum is set to one, then this pattern handles the case in which only the first response is needed, as is the case in broadcasting. When the quorum is set to the number of replicas, then this is a voting case where all the responses are needed.

In quorum consensus (QC), or voting, protocols, an operation can be allowed to proceed to completion if it can get permission from a group of replicas. The weighted majority QC algorithm [**58**] is proposed to maintain the consistency in the replicated files. It generalizes the notion of uniform voting. Instead of assigning a single vote per replica, each copy of a file is assigned a certain number of votes, whose sum over all copies is u. In this thesis, it is assumed that each copy is assigned a single vote. A read transaction must collect a read quorum of replicas that have at least r votes, and a write transaction must collect a write quorum of replicas that have at least w votes. Consistency is maintained by restricting $r + w$ to be greater than the total number of votes assigned to all copies. This quorum intersection ensures that every read operation returns the most recently written value. In [**69**], Herlihy generalizes Gifford's file replication algorithm [**58**] by introducing a general quorum consensus for replicated arbitrary data types. The main shortcoming of

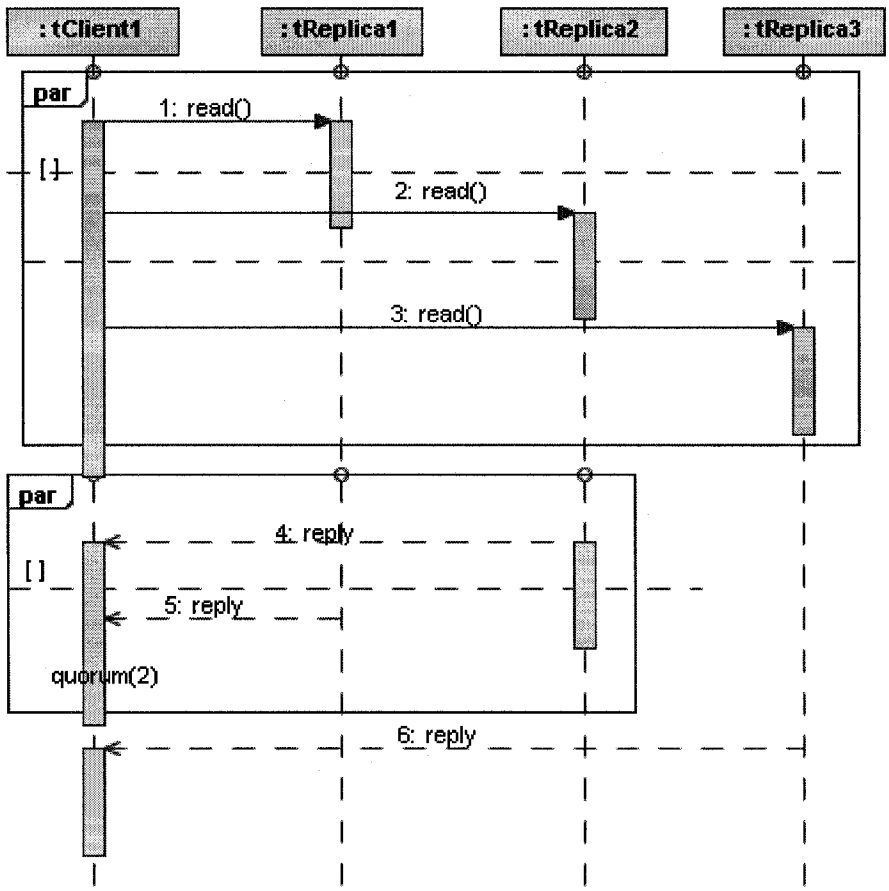


Figure 4.13: Sequence diagram for the quorum performance pattern.

quorum consensus protocols is the relatively high overhead incurred in the execution of read operations.

In [4], an analytic technique is developed to approximate the response time of arriving transactions that keep waiting until a quorum of operational sites is acquired. The waiting time is incurred because some of the sites are not available. It is assumed that a site's time-to-fail and time-to-repair are exponentially distributed. The work in [4] does not consider the waiting time incurred when the transaction makes requests to the operational remote

sites, and it does not address the performance overhead because of the contention of transactions. In [3], an analytic analysis for the availability of the system with quorum-based protocols for the distributed $(k+1)$ exclusion problem is conducted.

Prior to this thesis, the LQN could not model the quorum pattern in its general format. It could only model a special case in which the quorum is set to the maximum number of replicas (AND-join).

4.1.8. Select Performance Pattern.

In the select pattern, a client accesses only one replica from a set of replicas, at one time. Each replica is accessed with a predefined probability. This is also referred to as an OR pattern. In Figure 4.14, tClient1 generates a number randomly from a uniform distribution. If the generated number is less than a probability $p \leq 1$, tClient1 makes requests to tReplica1; otherwise it makes requests to tReplica2. In this model, tClient2 is another independent client who can read any replica at any time.

The select pattern can be modeled in the LQN formalism by having a client entry fork its main thread of control into a number of threads equal to the number of replicas to be accessed. The pattern of the forking of the threads is the OR-fork pattern that is defined in the existing LQN formalism. For example, in task tClient1 in Figure 4.12 the main thread that executes activity aUc1 is forked to send an update to either aUc1_1 or aUc1_2.

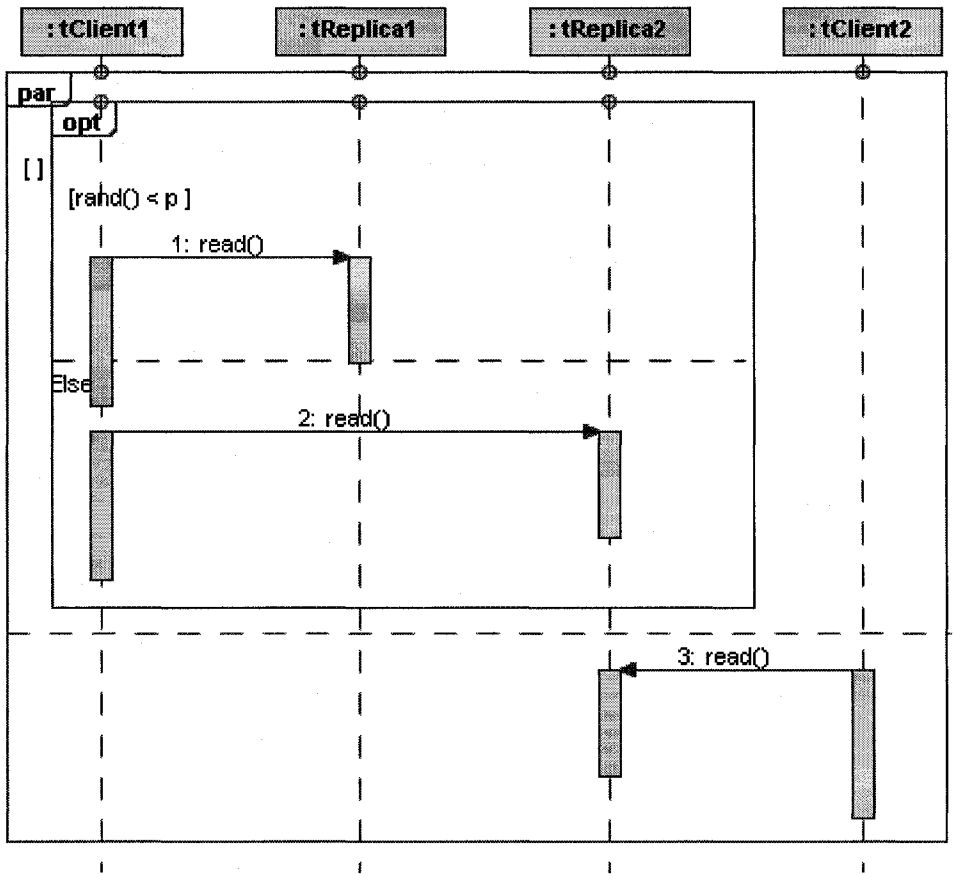


Figure 4.14: Sequence diagram for the select performance pattern.

4.2. Summary

There are different potential LQN models for a performance pattern, and the choice of an LQN model depends on the implementation, as the pattern usually does not focus on specific details. Most performance replication patterns developed in this chapter can be implemented using the current LQN only if they are modeled as standalone models, and the expanded notation is used. The main pattern that cannot be modeled by the current LQN, even when using the expanded notation, is the quorum performance pattern which will be covered in Chapters 5 and 6.

The existing replication notation in the LQN formalism has the advantage of being expressive by making it easier to specify and capture different replication patterns. In addition, the solution of a model with replication notation is rapid, and the solution time is scalable. However, two limitations arise when trying to model the performance replication patterns using the existing LQN replication notation. The first limitation is that the LQN formalism does not handle a replicated task with heterogeneous threads or internal parallelism. The solution to replicated tasks with internal parallelism will be covered in Chapter 7. The second limitation is that the LQN formalism does not handle a replicated task when the task receives asynchronous requests, or open arrivals, but the solution for this limitation will be a future work.

The solutions that will be developed for replicated systems in the following chapters can enable the designer to develop a library of performance replication patterns based on the performance measurements obtained after modeling various replication techniques in distributed systems. The results can be used by designers to rapidly compare the performance of alternative replication designs for their domains of applications.

CHAPTER 5

Deriving the Distribution Function of a Thread Delay

In some replication techniques, a J out of N quorum pattern (also called voting) is used. The main thread of control that executes the quorum waits for J responses before it resumes execution. Integrating a quorum pattern (see Subsection 4.1.7) into the LQN performance modeling formalism necessitates the computation of the quorum delay as the J^{th} order statistic. To do so, the exact or an accurate approximation of the delay distribution of individual responses is needed. This distribution was approximated in previous work, but only for the special case of ($J = N$). It gives large errors for $J \ll N$ in the models tested in this chapter. This chapter presents a new analytic approach for the derivation of the distributions. Under a number of assumptions, closed-form expressions for the cumulative distribution functions of the responses are derived. This chapter is concerned only with the derivation of the CDF of a thread delay in an LQN task, and the evaluation of the accuracy of the approximation. The CDFs will be used in Chapter 6 to calculate the service time of an entry in an LQN task. The application of this approach on a number of LQN models shows that, even for models that violate the assumptions, it is far more accurate than previous approaches. This approach gives an absolute value for the percentage error in the quorum delay of less than 10% for most example models. The results were verified by comparing them to the results of simulations and the results of solving Markov chains generated from PetriNet models with quorum. Much of the content of this chapter has been presented in **[113]**.

5.1. Quorum Performance Modeling

In an LQN task, a quorum pattern is modeled using an AND-fork followed by a quorum-join. Figure 5.1 shows an LQN task called App. The main thread of control executes activity aPre, then it is forked into N concurrent threads thread$i$'s for $i = 1 \cdots N$. The delay of a thread$i$ is the time required to execute activity aThread$i$, and to make requests to lower layer server(s) in the External Service Network in Figure 5.1. The notation $q(J)$ inside a circle denotes that the main thread of control is suspended until any J out-of N forked threads respond, with $J \leq N$. The time from the fork point until J threads join is called the quorum delay. This chapter is concerned with the derivation of the CDF of a thread delay. The quorum delay is used in this chapter to evaluate the accuracy of the thread delay approximation. The semantics of the disposition of the $(N - J)$ delayed threads will be covered in Chapter 6, and it does not affect the results obtained in this chapter. The CDF of each thread will be used in Chapter 6 to calculate the service time of task App.

In the LQN formalism, the quorum pattern presents an interesting problem for creating analytic performance models, because the quorum-join amounts to finding the distribution of the delay or completion time of the J^{th} response of the N replica threads. This delay is called the quorum delay which is very sensitive to the shape of the distribution functions of the N threads' delays. Previous methods for finding a thread delay distribution include the "three-point" approximation fitting [**80**], and the gamma distribution fitting [**172**]. In [**53**], the three-point method was used with good results for the case when $J = N$. A gamma distribution was used for fitting the thread delay distribution for estimating soft deadlines. In both of these cases, the tails of the cumulative distribution functions were more important in computing the result than any other part of the distribution. If a client makes a deterministic number of requests to the server, then the gamma distribution works well. However, if

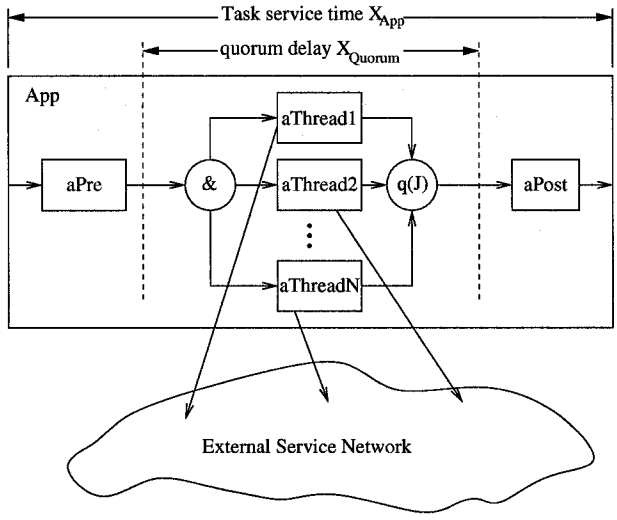


Figure 5.1: An LQN task with a quorum pattern.

the client makes a random number of requests with a geometric distribution, the errors increase significantly (if the gamma distribution is used) as shown from the results in Section 5.3 . Therefore, a more formal approach for the derivation of the distribution of the thread delay needs to be developed.

The solution of a model with a quorum-join is a more general case of solving a model with internal parallelism using an AND fork-join. First, submodels are solved by applying the Mean Value Analysis algorithm to find the service time at each queueing center. Second, an auxiliary model [**172**] is solved to find the variance for each activity. Next, sequences of activities which do not involve forks and joins are aggregated into a single activity using the technique in [**149**]. Finally, the overall delay for J out of N paths in the quorum-join is found by computing the J^{th} order statistic [**137**] (see Section 2.8) for the delays of all of the threads. In an AND fork-join, the number of joining threads is equal to the number of the forked threads, i.e. $J = N$.

5.2. Solution of LQN with Quorum

According to LQN semantics [**164**], upon accepting a new customer, the client enters into a cycle during which it alternates service periods of its own with requests made to other servers for nested requests. By default, the client's own service periods are random with an exponential distribution with a mean that is fixed across the different synchronous requests and it is state-independent. An execution demand with a coefficient of variation different than one and greater than zero can be specified in the LQN input model, but this is not handled in this thesis. Note that the distribution of the delay of each request, to a lower layer server, is not necessarily exponential because the server itself may make requests to other servers.

In the numerical solution of an LQN model using MVA, it is important that the distributions of the delays of all threads are computed as accurately as possible. Those distributions are needed to compute the distribution, and therefore the mean and variance, of the quorum delay using the J^{th} order statistic formula in [**137**] for independent RVs. In order to compute the J^{th} order statistic, the CDF for each RV has to be determined. The RV here is the delay of the thread that makes requests to server(s).

In order to make the derivation tractable, the analysis is temporarily restricted to models for which the following assumptions hold:

(1) The client makes requests to only one server.

(2) The distribution of the request's service time is exponential. This holds only if the server does not make requests to any lower layer servers.

(3) After the MVA solution of a submodel, which accounts for contention for shared resources among threads, the RVs associated with delays of the threads are assumed to be independent. Independence is important so that the order statistic formula can be used.

5.2.1. Mathematical Notation and Formal Problem Description.

The problem is to determine the distribution of the thread delay, i.e., the time it takes for a client to complete all its host processing and all its requests to the server(s).

Let X be an arbitrary RV. f_X, F_X, and $\mathrm{E}[X]$ are used to denote the PDF, CDF, and expected value of X respectively. Let X_c be the exponential RV with rate λ_c associated with the service time of each local processing of the client c, and X_s be the RV associated with the service time of each request to the server s. Based on the above assumptions, X_s is exponentially distributed with a given rate λ_s. Therefore,

$$f_{X_c}(t) = \lambda_c e^{-\lambda_c t} \quad F_{X_c}(t) = 1 - e^{-\lambda_c t} \quad \mathrm{E}[X_c] = \theta_c = 1/\lambda_c, \tag{5.1}$$

$$f_{X_s}(t) = \lambda_s e^{-\lambda_s t} \quad F_{X_s}(t) = 1 - e^{-\lambda_s t} \quad \mathrm{E}[X_s] = \theta_s = 1/\lambda_s. \tag{5.2}$$

The RV $Z = X_c + \sum_{i=1}^{K}(X_c + X_s)$ gives the thread delay, where K is a non-negative integer RV associated with the number of requests. The goal is to compute F_Z.

As previously mentioned, the LQN allows the distribution of K to be either geometric or deterministic. So, two cases will be considered: when K is geometrically distributed, and when K is deterministic.

5.2.2. Geometrically Distributed Requests.

When K's distribution is geometric, exact closed-form expressions for f_Z and F_Z will be derived. In order to do so, the functions are transformed into the frequency domain using Laplace transform. The transformation is important, since without it the derivation would involve modified Bessel functions [**61**], and therefore be more difficult to manipulate. $L\{f_X\}(s) = \int_0^{+\infty} e^{-st} f_X(t)dt$ is used to denote the Laplace transform of $f_X(t)$. Note that the steps of the derivations in the rest of this section, including the Laplace and inverse

Laplace transforms, are all performed symbolically (as opposed to numerically). Only the final result (F_Z) will be evaluated numerically during the numerical solution of the LQN.

Let the probability mass function of K be $\Pr\{K = k\} = p(1-p)^k$ where $k \in \{0, 1, \ldots\}$ and $0 < p \leq 1$. Note that $\mathrm{E}[K] = \frac{1}{p} - 1$, and hence, p can be calculated given the mean number of requests that the modeler provides in input the LQN model. Moreover, when K is geometric $\mathrm{E}[K]$ does not need to be an integer.

$\mathrm{E}[X_s]$ cannot be zero, because it would be meaningless to make requests to a server with zero service time. So, $\mathrm{E}[X_s] \neq 0$ always holds. Therefore, two cases are considered: $\mathrm{E}[X_c] = 1/\lambda_c \neq 0$ and $\mathrm{E}[X_c] = 0$. Case $\mathrm{E}[X_c] = 0$ arises if the client spends zero (or a negligible amount of) time at its own processor. The reason it is considered as a different case is that λ_c is not a real number and cannot be inserted into the final result.

5.2.2.1. *Case* $\mathrm{E}[X_c] \neq 0$.

Let the RVs $W = X_c + X_s$, $W_k = \sum_{i=1}^{k} W$ $(k \geq 0)$, and $W_K = \sum_{i=1}^{K} W$. Since $f_{X_c}(t) = \lambda_c e^{-\lambda_c t}$ and $f_{X_s}(t) = \lambda_s e^{-\lambda_s t}$, therefore

$$L\{f_W\} = L\{f_{X_c}\} \cdot L\{f_{X_s}\} = \frac{\lambda_c}{s+\lambda_c} \cdot \frac{\lambda_s}{s+\lambda_s}.$$

Moreover,

$$\begin{aligned} L\{f_{W_k}\} &= (L\{f_W\})^k = \left(\frac{\lambda_c \lambda_s}{(s+\lambda_c)(s+\lambda_s)}\right)^k, \\ f_{W_K}(t) &= \sum_{k=0}^{\infty} f_{W_k}(t) \Pr\{K = k\}. \end{aligned} \tag{5.3}$$

Taking the Laplace transform of both sides of Eq. (5.3), results in:

$$
\begin{aligned}
L\{f_{W_K}(t)\} &= \sum_{k=0}^{\infty} L\{f_{W_K}(t)|K=k\}\Pr\{K=k\} \\
&= \sum_{k=0}^{\infty} L\{f_{W_k}(t)\}\Pr\{K=k\} \\
&= \sum_{k=0}^{\infty} \left(\frac{\lambda_c\lambda_s}{(s+\lambda_c)(s+\lambda_s)}\right)^k (1-p)^k p \\
&= p\sum_{k=0}^{\infty} \left(\frac{\lambda_c\lambda_s(1-p)}{(s+\lambda_c)(s+\lambda_s)}\right)^k \\
&= p\frac{1}{1-\frac{\lambda_c\lambda_s(1-p)}{(s+\lambda_c)(s+\lambda_s)}} \\
&= \frac{p(s+\lambda_c)(s+\lambda_s)}{(s+\lambda_c)(s+\lambda_s)-\lambda_c\lambda_s(1-p)}.
\end{aligned} \tag{5.4}
$$

Moreover, using $Z = X_c + W_K$ and Eq. (5.4), then:

$$
\begin{aligned}
L\{f_Z\} &= L\{f_{X_c}\}\cdot L\{f_{W_K}\} \\
&= \frac{\lambda_c}{s+\lambda_c}\cdot\frac{p(s+\lambda_c)(s+\lambda_s)}{(s+\lambda_c)(s+\lambda_s)-\lambda_c\lambda_s(1-p)} \\
&= \frac{p\lambda_c(s+\lambda_s)}{(s+\lambda_c)(s+\lambda_s)-\lambda_c\lambda_s(1-p)}.
\end{aligned} \tag{5.5}
$$

To derive f_Z, the partial fraction decomposition method is used, that is, to find A_1 and A_2 such that $L\{f_z\} = \frac{A_1}{s+\gamma_1} + \frac{A_2}{s+\gamma_2}$ where $-\gamma_1$ and $-\gamma_2$ are the roots of the denominator of

$L\{f_Z\}$. Performing the calculations gives:

$$\begin{aligned}
\gamma_1 &= \frac{\lambda_c + \lambda_s + \sqrt{(\lambda_c + \lambda_s)^2 - 4p\lambda_c\lambda_s}}{2}, \\
\gamma_2 &= \frac{\lambda_c + \lambda_s - \sqrt{(\lambda_c + \lambda_s)^2 - 4p\lambda_c\lambda_s}}{2}, \\
A_1 &= \frac{p\lambda_c(\lambda_s - \gamma_1)}{(\gamma_2 - \gamma_1)}, \\
A_2 &= \frac{p\lambda_c(\gamma_2 - \lambda_s)}{(\gamma_2 - \gamma_1)}.
\end{aligned} \tag{5.6}$$

Using the inverse Laplace transform, $f_Z(t) = A_1 e^{-\gamma_1 t} + A_2 e^{-\gamma_2 t}$, and by integration

$$F_Z(t) = 1 - (A_1/\gamma_1)e^{-\gamma_1 t} - (A_2/\gamma_2)e^{-\gamma_2 t}. \tag{5.7}$$

5.2.2.2. *Case* $\mathrm{E}[X_c] = 0$.

The derivation above does not hold for this case, because λ_c is not a real number and F_Z cannot be derived from Eq. (5.7) without resorting to limit theorems. Therefore, it is considered separately.

Since $X_c = 0$, then $W = X_s$ and $Z = W_K = \sum_{i=1}^{K} X_s$. Therefore,

$$\begin{aligned} L\{f_Z(t)\} &= L\{f_{W_K}(t)\} \\ &= \sum_{k=0}^{\infty} L\{f_{W_k}(t)\}\, Pr\{K = k\} \\ &= \sum_{k=0}^{\infty} (\frac{\lambda_s}{s+\lambda_s})^k (1-p)^k p \\ &= p \sum_{k=0}^{\infty} (\frac{\lambda_s(1-p)}{s+\lambda_s})^k \\ &= p \frac{1}{1 - \frac{\lambda_s(1-p)}{(s+\lambda_s)}} \\ &= \frac{p(s+\lambda_s)}{s+\lambda_s p} = p + \frac{\lambda_s p(1-p)}{s+\lambda_s p}. \end{aligned} \tag{5.8}$$

Taking the inverse Laplace transform of Eq. (5.8), then $f_Z(t) = p\delta(t) + \lambda_s p(1-p)e^{-\lambda_s pt}$ for $t \geq 0$. Hence,

$$F_Z(t) = \begin{cases} 0 & t = 0 \\ p + (1-p)(1 - e^{-\lambda_s pt}) & t > 0 \end{cases}. \tag{5.9}$$

5.2.3. Deterministic Requests.

When K is deterministic, it is assumed that $\mathrm{E}[K] = k \in \mathbb{N}^1$. Before f_Z, and $F_Z(t) = \int f_Z(t)dt$ are derived, two distributions: gamma and McKay are introduced.

DEFINITION 5.2.1. A continuous RV X follows the gamma distribution with shape $i > 0$ and scale $\theta > 0$ when its probability density function can be expressed as follows:

$$f_X(x) = \begin{cases} 0 & x < 0 \\ \frac{x^{i-1}e^{-x/\theta}}{\theta^i \Gamma(i)} & x \geq 0 \end{cases}, \tag{5.10}$$

where $\Gamma(i)$ is the gamma function. ■

$\Gamma(i) = (i-1)!$ holds when $i \in \mathbb{N}$ and $\mathbb{N}$ is the set of natural numbers. It is known that mean and variance of X are $i\theta$ and $i\theta^2$ respectively. If $i \in \mathbb{N}$, the CDF of the gamma distribution can be written as

$$F_X(x) = 1 - e^{-\frac{x}{\theta}} \sum_{j=0}^{i-1} \frac{x^j}{\theta^j j!}. \tag{5.11}$$

In addition, it is known that the sum of $i \in \mathbb{N}$ exponentially distributed RVs with mean θ is a gamma distribution with parameters i and θ.

DEFINITION 5.2.2. A continuous RV X follows the McKay distribution [72] with parameters $a > -(1/2)$, $b > 0$, $|c| < 1$ when its probability density function can be expressed as follows:

$$f_X(x) = \begin{cases} 0 & x < 0 \\ \frac{\sqrt{\pi}(c^2-1)^{a+1/2}}{2^a b^{a+1}\Gamma(a+1/2)} x^a e^{-\frac{c}{b}x} I_a(\frac{x}{b}) & x \geq 0 \end{cases}, \tag{5.12}$$

where I_a is the modified Bessel function of the first kind and of order a [1]. ■

To compute f_Z and F_Z, three cases: $\theta_c = \theta_s$, $\theta_c > \theta_s$, and $\theta_c < \theta_s$ are considered.

[1]Because it is meaningless to make requests to a server a non-integer or negative number of times.

5.2.3.1. *Case* $\theta_c = \theta_s$.

Let $\theta_c = \theta_s = \theta$, then $X_c = X_s$ and $Z = X_c + \sum_{i=1}^{k}(X_c + X_s) = \sum_{i=1}^{2k+1} X_c$. Therefore, Z has a gamma distribution with shape $2k+1$ and scale θ. Using Eq. (5.11), its CDF can be written as:

$$F_Z(t) = \begin{cases} 0 & t \leq 0 \\ 1 - e^{-t/\theta} \sum_{i=0}^{2k} \frac{t^i}{\theta^i i!} & t > 0 \end{cases}. \tag{5.13}$$

5.2.3.2. *Case* $\theta_c \neq \theta_s$.

Consider $Z = X_c + \sum_{i=1}^{k}(X_c + X_s) = X_c + \sum_{i=1}^{k} X_c + \sum_{i=1}^{k} X_s$, where $k = \mathrm{E}[K]$ and $\theta_c \neq \theta_s$. $Y_c = \sum_{i=1}^{k} X_c$ is the sum of k exponential RVs with mean θ_c. Therefore, Y_c has a gamma distribution with parameters k and θ_c. Similarly, $Y_s = \sum_{i=1}^{k} X_s$ has a gamma distribution with parameters k and $\theta_s = 1/\lambda_s$. It can be shown (E.g., see [72]) that the RV $Y = Y_c + Y_s$ has a McKay distribution with parameters:

$$a = k - \frac{1}{2} \quad b = \frac{2\theta_c\theta_s}{|\theta_c - \theta_s|} \quad c = \frac{\theta_c + \theta_s}{|\theta_c - \theta_s|}. \tag{5.14}$$

Since $Z = Y + X_c$ and the PDF of the sum of two RVs is the convolution of the PDFs of the RVs, then:

$$\begin{aligned} f_Z(t) &= \int_0^t f_Y(x) f_{X_c}(t-x)\, dx \\ &= \underbrace{\frac{\sqrt{\pi}(c^2-1)^{a+1/2}}{2^a b^{a+1}\Gamma(a+1/2)\theta_c}}_{D} \int_0^t x^a e^{-\frac{c}{b}x} I_a(\frac{x}{b}) e^{-\frac{t-x}{\theta_c}}\, dx \\ &= De^{-\frac{t}{\theta_c}} \int_0^t x^a e^{-(\frac{c}{b}-\frac{1}{\theta_c})x} I_a(\frac{x}{b})\, dx. \end{aligned} \tag{5.15}$$

There are two subcases: $\theta_c > \theta_s$ and $\theta_c < \theta_s$. For the case when $\theta_c > \theta_s$,

$$\frac{c}{b} - \frac{1}{\theta_c} = \frac{\theta_c - \theta_s}{2\theta_c\theta_s} = \frac{1}{b}. \tag{5.16}$$

Using formula 11.3.12 in [**1**], variable change of $z = x/b$, and a number of simplification steps, the following equality is derived:

$$\int_0^t x^a e^{-\frac{x}{b}} I_a(\frac{x}{b})\, dx = \frac{t^{a+1}e^{-\frac{t}{b}}(I_a(\frac{t}{b}) + I_{a+1}(\frac{t}{b}))}{2a+1}. \tag{5.17}$$

Continuing Eq. (5.15) gives

$$\begin{aligned} f_Z(t) &= De^{-t(\frac{1}{\theta_c}+\frac{1}{b})}\frac{t^{a+1}(I_a(\frac{t}{b}) + I_{a+1}(\frac{t}{b}))}{2a+1} \\ &= \frac{De^{-t\frac{c}{b}}t^{a+1}(I_a(\frac{t}{b}) + I_{a+1}(\frac{t}{b}))}{2a+1}, \end{aligned} \tag{5.18}$$

since $\frac{1}{b} + \frac{1}{\theta_c} = \frac{c}{b}$.

For the case when $\theta_c < \theta_s$,

$$\frac{c}{b} - \frac{1}{\theta_c} = \frac{\theta_c - \theta_s}{2\theta_c\theta_s} = -\frac{1}{b}. \tag{5.19}$$

Similar to Eq. (5.17), the following equality is derived:

$$\int_0^t x^a e^{\frac{x}{b}} I_a(\frac{x}{b})\, dx = \frac{t^{a+1}e^{\frac{t}{b}}(I_a(\frac{t}{b}) - I_{a+1}(\frac{t}{b}))}{2a+1}. \tag{5.20}$$

Continuing Eq. (5.15) results in

$$\begin{aligned} f_Z(t) &= De^{-t(\frac{1}{\theta_c}-\frac{1}{b})}\frac{t^{a+1}(I_a(\frac{t}{b}) - I_{a+1}(\frac{t}{b}))}{2a+1} \\ &= \frac{De^{-t\frac{c}{b}}t^{a+1}(I_a(\frac{t}{b}) - I_{a+1}(\frac{t}{b}))}{2a+1}, \end{aligned} \tag{5.21}$$

since $\frac{1}{\theta_c} - \frac{1}{b} = \frac{c}{b}$.

This concludes that

$$f_Z(t) = \begin{cases} \frac{De^{-t\frac{c}{b}}t^{a+1}(I_a(\frac{t}{b})+I_{a+1}(\frac{t}{b}))}{2a+1} & \theta_c > \theta_s \\ \frac{De^{-t\frac{c}{b}}t^{a+1}(I_a(\frac{t}{b})-I_{a+1}(\frac{t}{b}))}{2a+1} & \theta_c < \theta_s \end{cases}. \tag{5.22}$$

In order to derive the indefinite integral of Eq. (5.22) and derive $F_Z(t)$ in a closed-form, assume $z = t/b$, and recall that in Eq. (5.22) $a = k - 1/2$. Then, using Eq. (8.467) in **[61]** results in:

$$I_{k-1/2}(z) = \frac{1}{\sqrt{2\pi z}}\left[e^z \sum_{i=0}^{k-1} \frac{(-1)^i(k-1+i)!}{i!(k-1-i)!(2z)^i} + (-1)^k e^{-z} \sum_{i=0}^{k-1} \frac{(k-1+i)!}{i!(k-1-i)!(2z)^i}\right], \tag{5.23}$$

$$I_{k+1/2}(z) = \frac{1}{\sqrt{2\pi z}}\left[e^z \sum_{i=0}^{k} \frac{(-1)^i(k+i)!}{i!(k-i)!(2z)^i} + (-1)^{k+1} e^{-z} \sum_{i=0}^{k} \frac{(k+i)!}{i!(k-i)!(2z)^i}\right]. \tag{5.24}$$

Then:

$$e^{-cz}z^{k+1/2}I_{k-1/2}(z) = \frac{1}{\sqrt{2\pi}}\left[e^{z(1-c)} \sum_{i=0}^{k-1} \frac{(-1)^i(k-1+i)!}{i!(k-1-i)!2^i} z^{k-i} + (-1)^k e^{-z(1+c)} \sum_{i=0}^{k-1} \frac{(k-1+i)!}{i!(k-1-i)!2^i} z^{k-i}\right], \tag{5.25}$$

$$e^{-cz}z^{k+1/2}I_{k+1/2}(z) =$$

$$\frac{1}{\sqrt{2\pi}}\left[e^{z(1-c)}\sum_{i=0}^{k}\frac{(-1)^i(k+i)!}{i!(k-i)!2^i}z^{k-i}+(-1)^{k+1}e^{-z(1+c)}\sum_{i=0}^{k}\frac{(k+i)!}{i!(k-i)!2^i}z^{k-i}\right]. \quad (5.26)$$

Furthermore, using [**61**]:

$$\int z^n e^{az}dz \;=\; e^{az}\sum_{j=0}^{n}\frac{(-1)^j n!}{a^{j+1}(n-j)!}z^{n-j}. \quad (5.27)$$

Then, let:

$$\begin{aligned}
A_1 &= \sqrt{2\pi}\int e^{-cz}z^{k+1/2}I_{k-1/2}(z)dz \\
&= e^{z(1-c)}\sum_{i=0}^{k-1}\frac{(-1)^i(k-1+i)!}{i!(k-1-i)!2^i}\sum_{j=0}^{k-i}\frac{(-1)^j(k-i)!}{(1-c)^{j+1}(k-i-j)!}z^{k-i-j} \\
&+ (-1)^k e^{-z(1+c)}\sum_{i=0}^{k-1}\frac{(k-1+i)!}{i!(k-1-i)!2^i}\sum_{j=0}^{k-i}\frac{(-1)^j(k-i)!}{(-(1+c))^{j+1}(k-i-j)!}z^{k-i-j} \\
&= e^{z(1-c)}\sum_{i=0}^{k-1}\frac{(-1)^i(k-1+i)!(k-i)}{i!2^i}\sum_{j=0}^{k-i}\frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!}z^{k-i-j} \\
&+ (-1)^{k+1}e^{-z(1+c)}\sum_{i=0}^{k-1}\frac{(k-1+i)!(k-i)}{i!2^i}\sum_{j=0}^{k-i}\frac{1}{(1+c)^{j+1}(k-i-j)!}z^{k-i-j} \quad (5.28)
\end{aligned}$$

and let:

$$
\begin{aligned}
A_2 &= \sqrt{2\pi} \int e^{-cz} z^{k+1/2} I_{k+1/2}(z) dz \\
&= e^{z(1-c)} \sum_{i=0}^{k} \frac{(-1)^i (k+i)!}{i!(k-i)!2^i} \sum_{j=0}^{k-i} \frac{(-1)^j (k-i)!}{(1-c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&+ (-1)^{k+1} e^{-z(1+c)} \sum_{i=0}^{k} \frac{(k+i)!}{i!(k-i)!2^i} \sum_{j=0}^{k-i} \frac{(-1)^j (k-i)!}{(-(1+c))^{j+1}(k-i-j)!} z^{k-i-j} \\
&= e^{z(1-c)} \sum_{i=0}^{k} \frac{(-1)^i (k+i)!}{i!2^i} \sum_{j=0}^{k-i} \frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&+ (-1)^k e^{-z(1+c)} \sum_{i=0}^{k} \frac{(k+i)!}{i!2^i} \sum_{j=0}^{k-i} \frac{1}{(1+c)^{j+1}(k-i-j)!} z^{k-i-j}. \qquad (5.29)
\end{aligned}
$$

Case $\theta_c > \theta_s$

$$\begin{aligned} F_Z(t) = \int_0^t f_Z(x)dx &= \int_0^t \frac{De^{-x\frac{c}{b}}x^{a+1}(I_a(\frac{x}{b}) + I_{a+1}(\frac{x}{b}))}{2a+1}dx \\ &= \frac{Db^{a+2}}{2a+1}\int_0^{\frac{t}{b}} e^{-cz}z^{a+1}(I_a(z) + I_{a+1}(z))dz \\ &= \frac{Db^{a+2}(A_1 + A_2 + C)}{\sqrt{2\pi}(2a+1)} \\ &= \frac{(c^2-1)^k b(A_1 + A_2 + C)}{k!2^{k+1}\theta_c}. \end{aligned} \tag{5.30}$$

C should be determined such that $F_Z(0) = 0$, or equivalently $C = -(A_1 + A_2)|_{z=0}$.

Using the fact that $(k+i)! = (k+i-1)! \times (k+i) = (k-1+i)! \times (k+i)$, and by substituting for $i = k$ in the first term of $A2$, then

$$\begin{aligned} A_1 + A_2 = {} & e^{z(1-c)}\sum_{i=0}^{k-1}\frac{(-1)^i(k-1+i)!}{i!2^i} \\ & [(k-i)+(k+i)]\sum_{j=0}^{k-i}\frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!}z^{k-i-j} \\ & + e^{z(1-c)}\frac{(-1)^k(2k)!}{k!2^k(1-c)} \\ & + (-1)^k e^{-z(1+c)}\sum_{i=0}^{k-1}\frac{(k-1+i)!}{i!2^i}[-(k-i)+(k+i)] \\ & + \sum_{j=0}^{k-i}\frac{1}{(1+c)^{j+1}(k-i-j)!}z^{k-i-j} + (-1)^k e^{-z(1+c)}\frac{(2k)!}{k!2^k(1+c)} \end{aligned}$$

$$
\begin{aligned}
&= 2ke^{z(1-c)} \sum_{i=0}^{k-1} \frac{(-1)^i (k-1+i)!}{i!2^i} \sum_{j=0}^{k-i} \frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&+ e^{z(1-c)} \frac{(-1)^k (2k)!}{k!2^k(1-c)} \\
&+ (-1)^k 2e^{-z(1+c)} \sum_{i=1}^{k-1} \frac{(k-1+i)!}{(i-1)!2^i} \sum_{j=0}^{k-i} \frac{1}{(1+c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&+ (-1)^k e^{-z(1+c)} \frac{(2k)!}{k!2^k(1+c)} \\
&= 2ke^{z(1-c)} \sum_{i=0}^{k} \frac{(-1)^i (k-1+i)!}{i!2^i} \sum_{j=0}^{k-i} \frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&+ (-1)^k 2e^{-z(1+c)} \sum_{i=1}^{k} \frac{(k-1+i)!}{(i-1)!2^i} \sum_{j=0}^{k-i} \frac{1}{(1+c)^{j+1}(k-i-j)!} z^{k-i-j} \qquad (5.31)
\end{aligned}
$$

and,

$$
\begin{aligned}
C &= -(A_1 + A_2)|_{z=0} \\
&= -2k \sum_{i=0}^{k} \frac{(-1)^i (k+i-1)!(-1)^{k-i}}{i!2^i(1-c)^{k-i+1}} - (-1)^k 2 \sum_{i=1}^{k} \frac{(k+i-1)!}{(i-1)!2^i(1+c)^{k-i+1}} \\
&= (-1)^{k+1} 2k \sum_{i=0}^{k} \frac{(k+i-1)!}{i!2^i(1-c)^{k-i+1}} + (-1)^{k+1} 2 \sum_{i=1}^{k} \frac{(k+i-1)!}{(i-1)!2^i(1+c)^{k-i+1}} \qquad (5.32)
\end{aligned}
$$

Case $\theta_c < \theta_s$

Start with

$$\begin{aligned} F_Z(t) = \int_0^t f_Z(x)dx &= \int_0^t \frac{De^{-x\frac{c}{b}}x^{a+1}(I_a(\frac{x}{b}) - I_{a+1}(\frac{x}{b}))}{2a+1}dx \\ &= \frac{Db^{a+2}}{2a+1}\int_0^{\frac{t}{b}} e^{-cz}z^{a+1}(I_a(z) - I_{a+1}(z))dz \\ &= \frac{Db^{a+2}(A_1 - A_2 + C)}{\sqrt{2\pi}(2a+1)} \\ &= \frac{(c^2-1)^k b(A_1 - A_2 + C)}{2^{k+1}k!\theta_c}. \end{aligned} \tag{5.33}$$

C should be determined such that $F_Z(0) = 0$, or equivalently, $C = -(A_1 - A_2)|_{z=0}$. Using the fact that $(k+i)! = (k+i-1)! \times (k+i) = (k-1+i)! \times (k+i)$, then

$$\begin{aligned} A_1 - A_2 = {} & e^{z(1-c)} \sum_{i=0}^{k-1} \frac{(-1)^i(k-1+i)!}{i!2^i} \\ & [(k-i) - (k+i)] \sum_{j=0}^{k-i} \frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!} z^{k-i-j} \\ & -e^{z(1-c)} \frac{(-1)^k(2k)!}{k!2^k(1-c)} \\ & +(-1)^k e^{-z(1+c)} \sum_{i=0}^{k-1} \frac{(k-1+i)!}{i!2^i} \\ & [-(k-i) - (k+i)] \sum_{j=0}^{k-i} \frac{1}{(1+c)^{j+1}(k-i-j)!} z^{k-i-j} \\ & -(-1)^k e^{-z(1+c)} \frac{(2k)!}{k!2^k(1+c)} \end{aligned}$$

$$
\begin{aligned}
&= -2e^{z(1-c)} \sum_{i=1}^{k-1} \frac{(-1)^i (k-1+i)!}{(i-1)! 2^i} \sum_{j=0}^{k-i} \frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&\quad -e^{z(1-c)} \frac{(-1)^k (2k)!}{k! 2^k (1-c)} \\
&- (-1)^k 2k e^{-z(1+c)} \sum_{i=0}^{k-1} \frac{(k-1+i)!}{i! 2^i} \sum_{j=0}^{k-i} \frac{1}{(1+c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&- (-1)^k e^{-z(1+c)} \frac{(2k)!}{k! 2^k (1+c)} \\
&= -2e^{z(1-c)} \sum_{i=1}^{k} \frac{(-1)^i (k-1+i)!}{(i-1)! 2^i} \sum_{j=0}^{k-i} \frac{(-1)^j}{(1-c)^{j+1}(k-i-j)!} z^{k-i-j} \\
&- (-1)^k 2k e^{-z(1+c)} \sum_{i=0}^{k} \frac{(k-1+i)!}{i! 2^i} \sum_{j=0}^{k-i} \frac{1}{(1+c)^{j+1}(k-i-j)!} z^{k-i-j}, \qquad (5.34)
\end{aligned}
$$

and,

$$
\begin{aligned}
C &= -(A_1 - A_2)|_{z=0} \\
&= 2 \sum_{i=1}^{k} \frac{(-1)^i (k+i-1)! (-1)^{k-i}}{(i-1)! 2^i (1-c)^{k-i+1}} + (-1)^k 2k \sum_{i=0}^{k} \frac{(k+i-1)!}{i! 2^i (1+c)^{k-i+1}} \\
&= (-1)^k 2 \sum_{i=1}^{k} \frac{(k+i-1)!}{(i-1)! 2^i (1-c)^{k-i+1}} + (-1)^k 2k \sum_{i=0}^{k} \frac{(k+i-1)!}{i! 2^i (1+c)^{k-i+1}}. \qquad (5.35)
\end{aligned}
$$

5.2.4. Violation of the Assumptions.

The closed-form formulas developed above were derived using a number of assumptions mentioned in the beginning of the section. When the first two assumptions are violated, the approximation approaches to use those formulas will be explained. In Section 5.3, the effect of the approximations will be evaluated.

5.2.4.1. *Requests to More than One Server.*

The first assumption is that a client thread makes requests to only one server. Otherwise, a thread makes requests to a set of servers $s_1, \ldots, s_j$ ($j > 1$). According to the LQN definition, the distribution of the RV associated with the number of requests made by a thread to its servers must be the same for all servers (i.e., either all geometric or all deterministic). In this case, an approximation method is used and the set of servers are replaced by a single server.

Let θ_{s_i} be the mean service time of server i, k_i be the mean number of requests from the client to server i ($1 \leq i \leq j$), θ_s be the mean service time of the approximated server, and k be the mean number of requests from the client to the approximated server. The approximation computes k and θ_s such that it conserves the mean number of requests and also the mean of the service time. More formally,

$$k = \sum_{i=1}^{j} k_i \qquad \theta_s = \frac{1}{k} \sum_{i=1}^{j} \theta_{s_i} k_i. \tag{5.36}$$

5.2.4.2. *Non-exponential Service Times.*

The second assumption is that the server has an exponentially distributed service time. This is a valid assumption if the server does not make lower layer requests. However, if the server makes lower layer requests, the service time of the server will not be exponentially distributed. Based on this study, there will be no closed-form formula for the distribution of the thread delay.

In this case, the non-exponential distribution of the delay is simply approximated with an exponential distribution that has the same mean as that of the non-exponential distribution.

ALGORITHM 5.2.3. *Sampling of a distribution function when using the closed-form formulas.*

- INPUTS: avgNumRequestsToLowerLayerTasks, layer1Mean, layer2Mean;
- INITIALIZE: previousCdfValue = 0; chebyshevProb = 0.999;
- calcVariance = calculateVar(); %calculate the variance based on the CDF used.
- maxSamplingTime = sqrt(calcVariance)/(1- chebyshevProb);
- calcMean = calcMean(); %calculate the mean based on the CDF used.
- threshold = calcMean$\times$(1 - chebyshevProb);
- FOR (time = 0.0, index =1; time $\leq$ maxSamplingTime; time = time+stepSize)
  - ✧ cdfValue = closedFormFormula(time, avgNumRequestsToLowerLayerTasks , layer1Mean, layer2Mean);
  - ✧ IF (((cdfValue - previousCdfValue) $\times$ time) > threshold)
    - ❑ cdfTimeVector.insert(index, time);
    - ❑ cdfValueVector.insert(index, cdfValue);
    - ❑ previousCdfValue = cdfValue;
    - ❑ index++;
  - ✧ END IF
- END FOR
- RETURN cdfTimeVector, cdfValueVector %vectors with results for CDF sampled points.

5.3. Results and Analysis

In this section, the accuracy of fitting the distribution of a thread delay to the new derived closed-form formulas, a gamma distribution or a three-point distribution is evaluated. Three LQN models are considered, each with several configurations. For each configuration, the quorum delay is computed using three methods when possible: (1) LQNS using different distributions, (2) simulation, and (3) exact solution of the Markov chain of the model. The Markov chain is generated from a PetriNet model of an LQN model with a quorum and is solved using GreatSPN. The solution obtained using GreatSPN does not provide a variance. The results from the three methods are then compared.

As mentioned in Section 5.2, the distribution of the thread delay was derived under the assumption that a number of assumptions hold. Different LQN models are presented to show the accuracy of the distributions both when those assumptions are satisfied and also when they are violated. In particular, in sections 5.3.2 and 5.3.3, two LQN models are studied for which the first and the second assumptions do not hold, respectively. For each model, the results for both cases when the mean number of requests to lower layer servers is deterministic, and geometrically distributed are examined. Moreover, various combinations of parameter values are chosen to see whether the errors, in the studied approaches, are sensitive to the different parameters required. Particularly, the ratio of the client's execution demand to the server's execution demand is varied from a low to a high value.

One assumption that must hold to apply the order statistic formula is that the RVs associated with the delays of the forked threads are independent. The independence hold only if the local executions of the forked threads are on an infinite server host processor, and the threads make requests to independent remote servers. Otherwise, the usage of the order statistic formula will be an approximation. This section discusses the accuracy of the

approximation when using a limited number of copies of the host processor, that runs the forked threads, with processor sharing or FIFO scheduling disciplines.

The CDF of a thread, in the LQNS, is expressed numerically at a sequence of values, using the expressions derived in this chapter. Then, the J^{th} order statistic is found numerically using the equations in Subsection 2.8. Algorithm 5.2.3 shows the steps to numerically sample a cumulative distribution function fitted by the mean and variance of a thread delay.

In this thesis, the confidence intervals obtained from simulations are used to plot the error bars in some Figures. The error bars represent the absolute value of the highest and lowest percentage errors in the LQNS results compared to the simulation results. Assume that the simulation gives a result of $y \pm \delta$ and the LQNS gives a result of x, for the mean value of a given RV. Also, let $U = 100 \times (x - (y + \delta))/(y + \delta)$, and $L = 100 \times (x - (y - \delta))/(y - \delta)$, then the absolute value of the highest percentage error is equal to

$$max(|U|, |L|), \tag{5.37}$$

and the absolute value of the lowest percentage error is calculated as follows:

$$\begin{cases} 0 & if\ U \times L < 0 \\ min(|U|, |L|) & Otherwise \end{cases} . \tag{5.38}$$

.

5.3.1. Effect on One-Layer Models.

Figure 5.2 shows an LQN model with task tB1 having six threads. When a request arrives to entry eB1, the main or first thread of the task is started. The main thread executes activity aa. When activity aa completes execution, the main thread forks into five threads: threads $1, 2, 3, 4, 5$ that execute activities a1,...,a5 respectively. The execution of each of

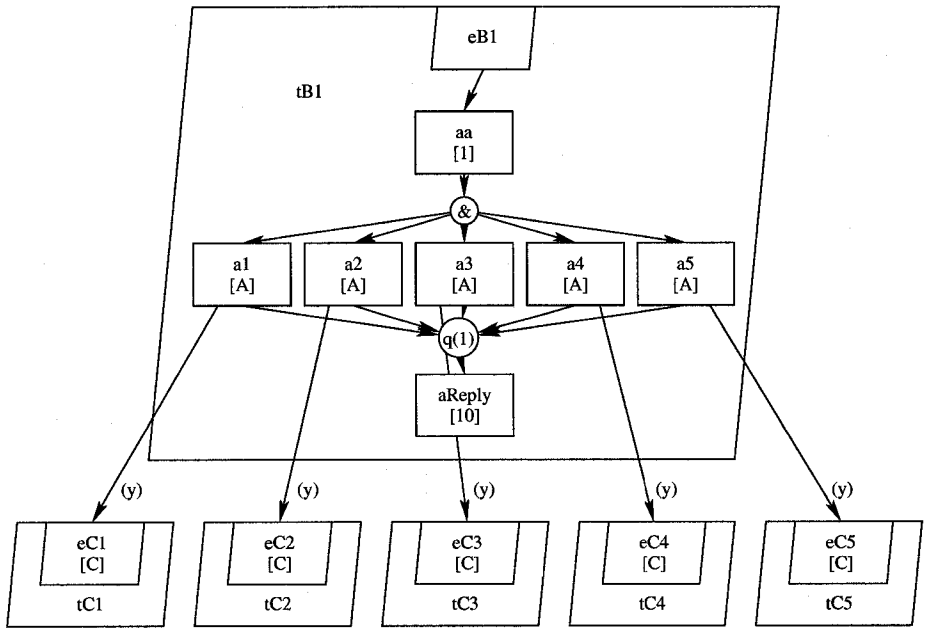


Figure 5.2: One-layer LQN model with a quorum notation.

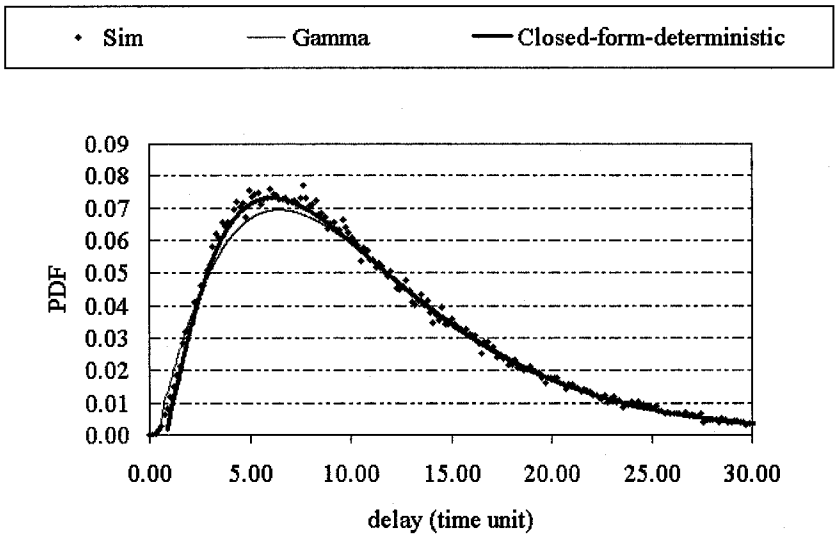
these five activities requires requests to the lower layer servers tC1,...,tC5 respectively. Activities ai, $\forall i = 1, \ldots, 5$, each has an execution demand of A time units, and entries eCi each has an execution demand of C time units. After one thread responds (a quorum of 1 in Figure 5.2), the main thread resumes execution, and it executes activity aReply. The objective here is to analytically compute the time required since forking the main thread until one thread, out of the 5 threads, responds: this time is the quorum delay.

Figure 5.3 shows the probability distribution function for the delay of thread 1, that executes activity a1 in Figure 5.2, with $y = 2, A = 1, C = 5$. The PDFs of the thread delay are obtained from the simulation runs, fitting the the delay to a gamma distribution, and the closed-form formula derived in this chapter for deterministic requests. In Figure 5.3(a), processor pB1 that executes task tB1 is an infinite server, while in Figure 5.3(b), the pB1 is a single server with a processor sharing scheduling discipline. When pB1 is an

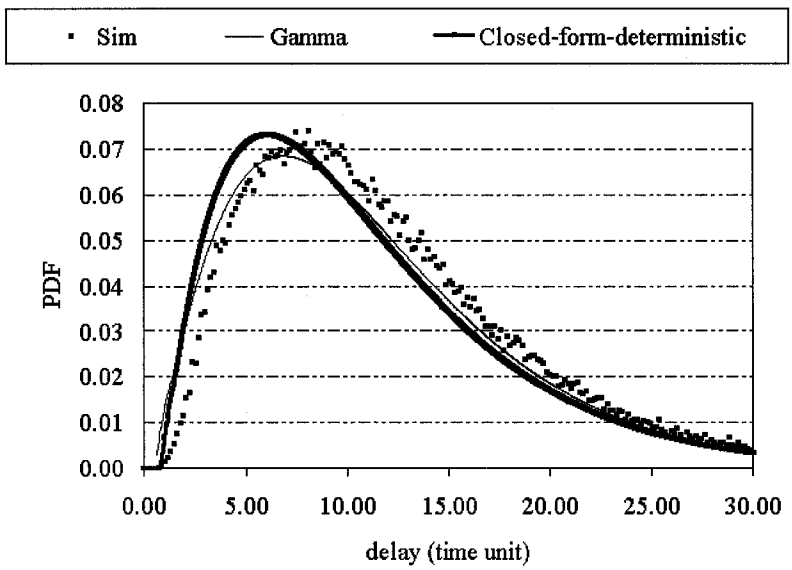
infinite server, then the assumptions for the closed-form formulas for deterministic requests are satisfied. When pB1 is a single server with a processor sharing discipline, then the forked threads will not be independent, as the threads are contending for pB1, hence the assumptions are violated. However, the results show that both the gamma and the closed-form formulas for determinsitic requests follow the simulation very well.

Tables 5.1 and 5.2 show the quorum delay, for the LQN model in Figure 5.2, when all requests shown as the parameter (y) from the forked threads to the servers are deterministic. The 3-tuple in the first column of the tables is the number of deterministic requests (y) by a thread to a server, the execution demand (A) of activities ai, and the execution demand (C) for entries eCi for $i = 1, \ldots, 5$. The second column is the exact solution for the quorum delay calculated from a Markov chain generated from a PetriNet model of the LQN model. This solution does not compute the variance. The third column shows the mean and variance results from the simulation of the LQN model. The simulation confidence intervals with 95% confidence coefficient for all values of the mean of the quorum delay are less than 5% of the mean, and for the all values of the variance of the quorum delay are less than 10% of the variance. The remaining columns show the percentage errors of the LQNS analytic solution results when using the three-point approximation, and the gamma distribution fitting in Table 5.1, and using the closed-form formulas for deterministic requests fitting in Table 5.2. This error is found by dividing the difference between the analytic and simulation results by the simulation result, and multiplying by 100. The results show that the gamma distribution fit has an absolute value of the percentage error of less than 2% for the mean quorum delay, while the closed-forms formula for deterministic requests has an absolute value of the percentage error of less than 3.5%.

Figure 5.4 shows the PDF for the delay of thread 1, that executes activity a1 in Figure 5.2, with $y = 2, A = 1, C = 5$. The PDFs are obtained from the simulation run, and



(a) Infinite Server



(b) Processor Sharing

Figure 5.3: PDF for the delay of thread 1 in the one-layer LQN model in Figure 5.2 with deterministic requests. The processor for tB1 is an infinite server in (a), or a single server with a processor sharing discipline in (b). $y = 2, A = 1, C = 5$.

Table 5.1: Quorum delay of the one-layer model with deterministic requests in Figure 5.2. The processor for tB1 is an infinite server.

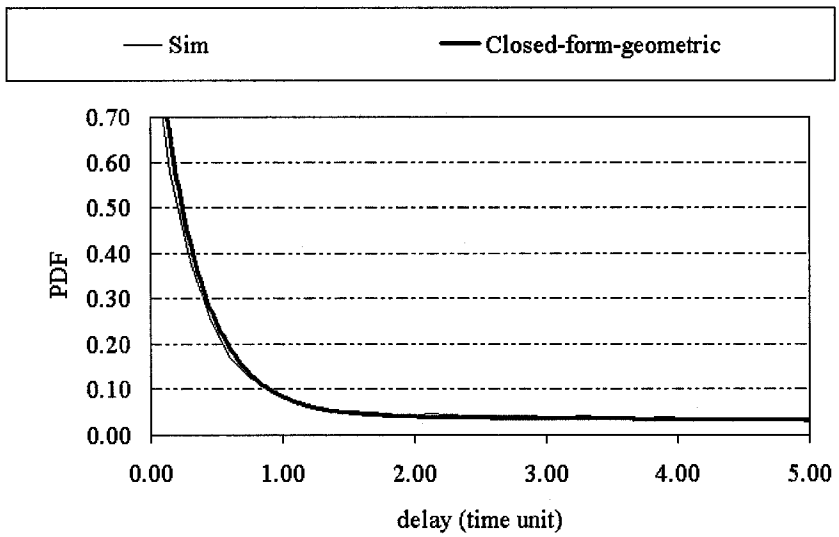
Parameters (y,A,C)	Markov Chain	Sim		LQNS % Error			
		95% conf.	interval	Three-point		Gamma	
	mean	mean	variance	mean	variance	mean	variance
(2D,1,5)	4.46	4.45±0.028918	4.91±0.17959	8.70	18.15	-1.29	10.12
(2D,5,5)	7.60	7.60±0.065367	8.21±0.23492	10.12	-18.02	-1.29	8.22
(2D,5,1)	3.76	3.76±0.0087922	1.78±0.032556	12.07	-33.14	0.83	-0.48

Table 5.2: Quorum delay of the one-layer model with closed-form formulas for deterministic requests in Figure 5.2. The processor for tB1 is an infinite server.

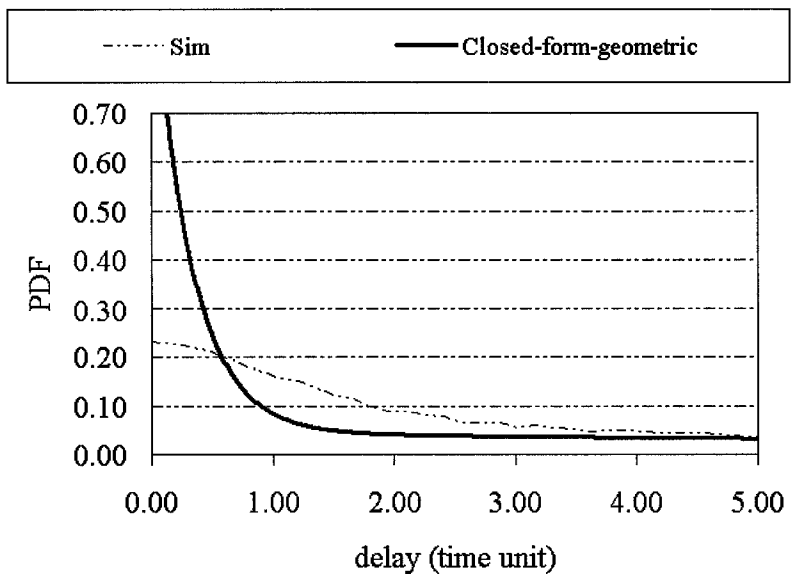
Parameters (y,A,C)	Markov Chain	Sim 95% conf.	interval	LQNS % Error Closed-form-deterministic	
	mean	mean	variance	mean	variance
(2D,1,5)	4.46	4.45±0.028918	4.91±0.17959	1.76	-2.63
(2D,5,5)	7.60	7.60±0.065367	8.21±0.23492	1.27	-3.17
(2D,5,1)	3.76	3.76±0.0087922	1.78±0.032556	1.29	-2.57

the from fitting the thread delay to the closed-form formula derived in this chapter for geometric requests. In Figure 5.4(a), processor pB1, that executes task tB1, is an infinite server, while in Figure 5.4(b), the pB1 is a single server with a processor sharing scheduling discipline. The results show that the closed-form formula for geometric requests is very close to the simulation when pB1 is an infinite server. But, when pB1 has a PS scheduling discipline, then in the first part of the PDF of the delay differs between the simulation and the new closed-form formula. The difference is due to the violation of the independence assumption for the closed-form formula for geometric requests. The shapes of the PDFs (not shown in Figure 5.4) of the thread delay obtained from fitting the thread delay to a gamma or a three-point distributions are significantly different from the PDF obtained from the simulation run.

Table 5.3 shows the quorum delay, for the LQN model in Figure 5.2, when the threads make two geometrically distributed requests with mean of 2 (parameter "y" in the table) to the lower layer servers tCi. This model is solved using a Markov chain for an exact solution, simulation of the LQN model, and the LQNS. For the LQNS solutions, three distribution fittings are used: the three-point approximation, the gamma distribution, and the closed-form formula for geometric requests. As shown in Table 5.3, the absolute value of the percentage error in the mean for the quorum delay can go up to about 34% for the case of the three-point approximation, and up to 70% for the gamma fitting, while when using the closed-form formula the errors are less than 10%. The shapes of the PDFs of the thread delay when it is fitted to a gamma and a three-point distributions do not follow the shape of the PDF obtained from the simulation run. Even though this LQN model satisfies all the closed-form formula derivation assumptions, the errors in the quorum delay when using the closed-form formula for geometric requests is due to other approximations used in the LQNS solution.



(a) Infinite Server



(b) Processor Sharing

Figure 5.4: PDF for the delay of thread 1 in the one-layer LQN model in Figure 5.2 with geometric requests. The processor for tB1 is an infinite server in (a), or a single server with a processor sharing discipline in (b). $y = 2, A = 1, C = 5$.

Table 5.3: Quorum delay of the one-layer model with geometric requests in Figure 5.2. The processor for **tB1** is an infinite server.

Parameters (y,A,C)	Markov Chain	Sim 95% conf. interval		LQNS % Error					
						Gamma		Closed-form-geometric	
	mean	mean	variance	mean	variance	mean	variance	mean	variance
(2,1, 5)	0.67976	0.67802±0.025777	2.7914±0.20266	34.28	230.14	70.23	1.49	9.10	0.62
(2,5, 5)	1.7245	1.7238±0.061144	6.5239±0.69985	-14.53	206.85	21.77	3.59	5.68	0.03
(2,5,1)	1.2496	1.2438±0.034939	1.8738±0.047702	-30.43	185.07	8.52	-2.61	3.69	0.39

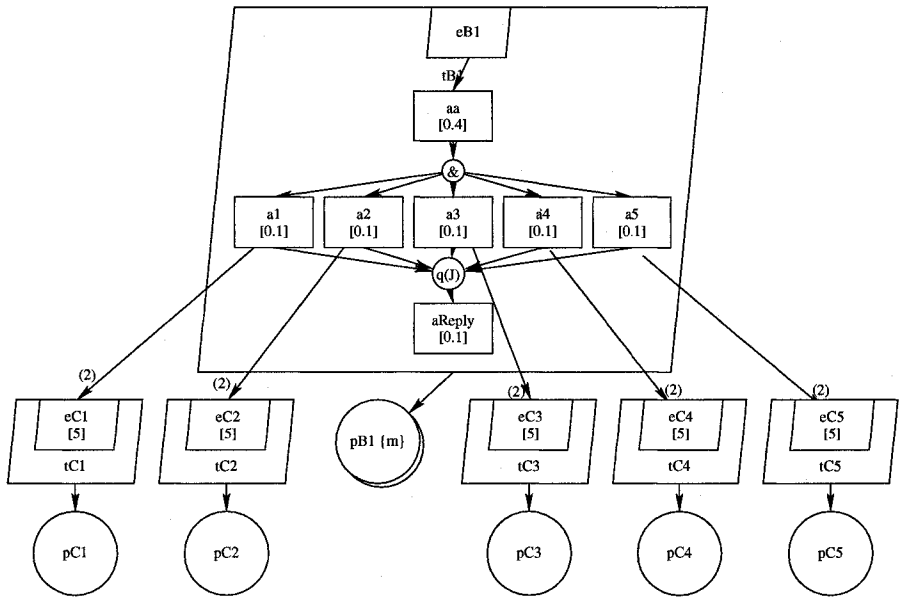


Figure 5.5: One-layer LQN model with multiplicity m for pB1, and quorum of J.

Figure 5.5 shows a more realistic LQN model, because the execution demands of the forked threads on their host processor are more than an order of magnitude less than the execution demands of the remote servers. So, the forked threads run in parallel most of the time. The local execution demand of each forked thread is 0.1 time units and the execution demand of activity aReply is 0.1 time units. Processor pB1 has a multiplicity of m which will be varied to show the effect of the multiplicity of pB1 on the accuracy of the service time when using the quorum approximations derived in this chapter.

Figure 5.6 shows the absolute value of the percentage error in the quorum delay of J out of the five forked threads in task tB1 in Figure 5.5 with $m = 1$, and when geometric requests are used for the forked threads. The scheduling discipline for processor pB1 that is running task tB1 is processor sharing (PS). Figure 5.6 shows that the maximum absolute value of the percentage error is less than 7%. In this model, the independence of the RVs associated with the delays of the forked threads assumption is violated, because

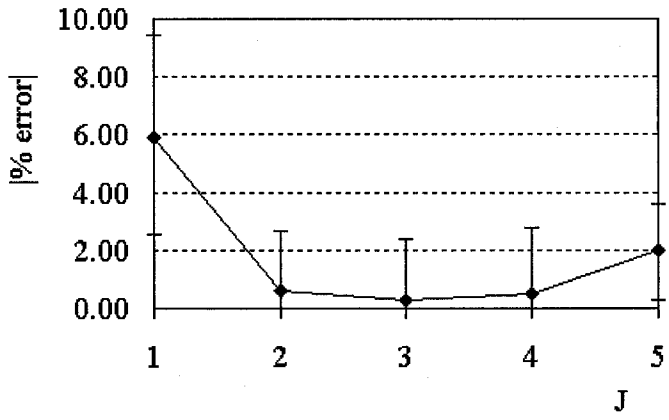


Figure 5.6: Error in the quorum delay in tB1 in Figure 5.5 with geometric requests and $m = 1$. The scheduling discipline for pB1 is PS.

the scheduling discipline for processor pB1 is processor sharing, so the threads contend among each other for pB1. The maximum error occurs when J is minimum. However, the absolute value of the percentage error is not always decreasing as J is increasing, because the error in approximating the distributions for the forked threads does not necessarily decrease monotonically as J increases.

The effect of the multiplicity of the host processor of the forked threads on the accuracy of the quorum delay computations for the LQN model in Figure 5.5 is shown in Figure 5.7. The scheduling discipline for processor pB1 is processor sharing (PS). The effect of the multiplicity (m) for processor pB1 is shown for the cases when $J = 3$ and $J = 1$. The absolute value of the percentage error reaches about 20% when $J = 1$ and less than 2% when $J = 3$. As expected, when $J = 1$ the error is greater than the error when $J = 3$. When the multiplicity increases, the absolute value of the percentage error in the quorum delay increases. The increase in error might be due to the increase in the dependency among the RVs associated with the delays of the forked threads when m increases, because

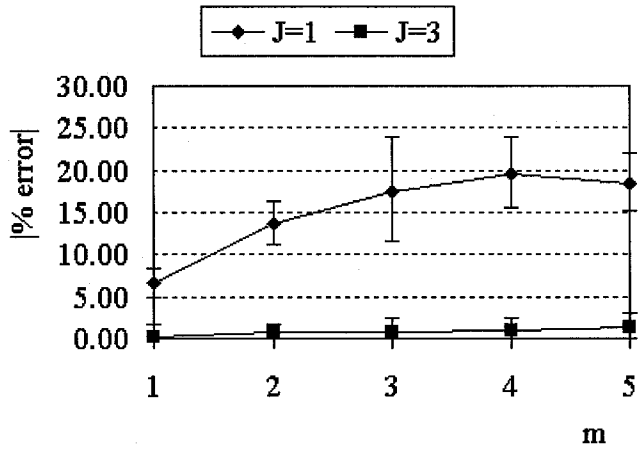


Figure 5.7: Effect of the multiplicity m of pB1 on the error in the quorum delay in tB1 in Figure 5.5 with geometric requests. The scheduling discipline for pB1 is PS.

the threads share the same queue at processor pB1. However, the effect of the multiplicity tends to stabilize when a multiplicity level, at which the processor pB1 is not the bottleneck anymore, is reached.

Figure 5.8 shows an LQN model for a Triple Modular Redundancy (TMR) model. In this LQN model the local execution demands for the forked threads are 0.01 time units. There are three forked threads and each makes a number of requests with mean $y = 3$ to its remote server tCi for $i = 1, \cdots, 3$.

The absolute value of the percentage error in the quorum delay of J responses out of three forked threads in task tB1 in Figure 5.8 with geometric requests is shown in Figure 5.9 for different values for the coefficient of variations (CV) of entries eCi's for $i = 1 \cdots, 3$. When $CV > 1$, the LQN model violates the assumptions for the closed-form formula, because the service times for the remote servers are non-exponentially distributed. In addition, there is dependence among the RVs associated with delays of the forked threads, because the scheduling discipline for processor pB1 is processor sharing. The results show

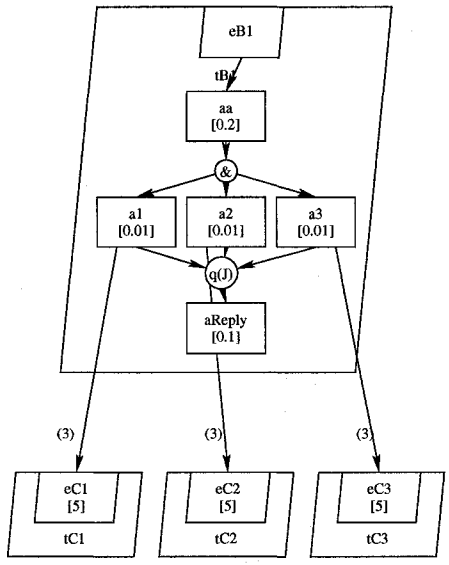


Figure 5.8: An LQN model for a Triple Modular Redundancy configuration.

that increasing the coefficient of variation increases the absolute value of the percentage error in the quorum delay of the J out of the three forked threads in task tB1. For the case when $CV = 1.5$, the absolute value of the percentage error increases with almost the same rate for different values of J. The similarity in the increase rates is because the closed-form formula for geometric requests assumes that the remote servers have exponentially distributed service time (even in the case when their coefficients of variations are set to be greater than one in the input model).

To have more insight on the effect of the coefficient of variation of the execution demands of the remote servers on the accuracy of the service time of the client entry eB1, Figure 5.10 shows the trend on the accuracy of the results of the quorum delay of $J = 2$ threads out of the three forked threads in task tB1 in Figure 5.8. In this model, geometric requests are used , and processor pB1 has a processor sharing scheduling discipline. The results show that the absolute value of the percentage error increases almost linearly as the coefficient of variation increases.

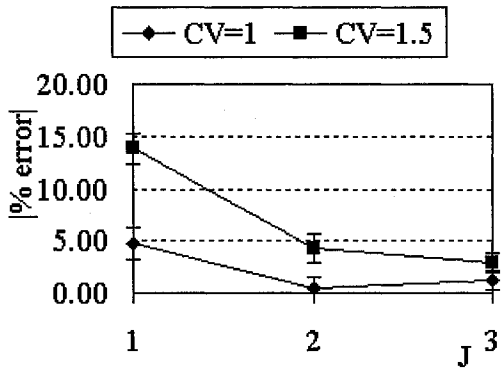


Figure 5.9: Error in the quorum delay in **tB1** in Figure 5.8 with geometric requests for different values for the coefficient of variation (CV) of entries **eCi**'s $i = 1 \cdots, 3$. **pB1** is scheduled using PS.

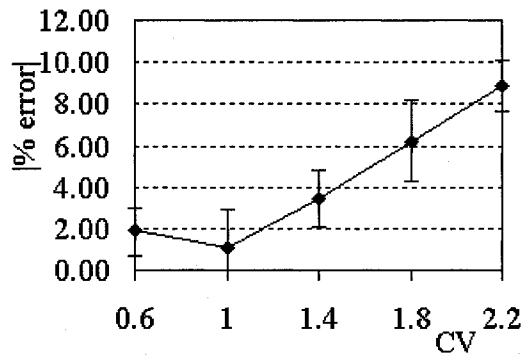


Figure 5.10: Effect of coefficient of variation (CV) of entries **eCi**'s for $i = 1, \cdots, 3$ on the error in the quorum delay in **tB1** in Figure 6.10 when $J = 2$ and geometric requests are used. The scheduling discipline for **pB1** is PS.

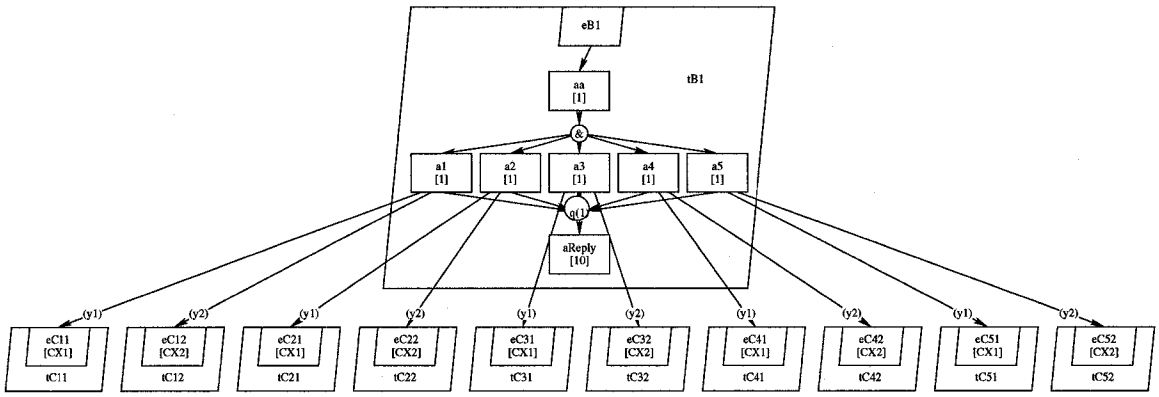


Figure 5.11: An LQN model with two-request branches per thread.

5.3.2. Effect on One-Layer with Multi-Request Models.

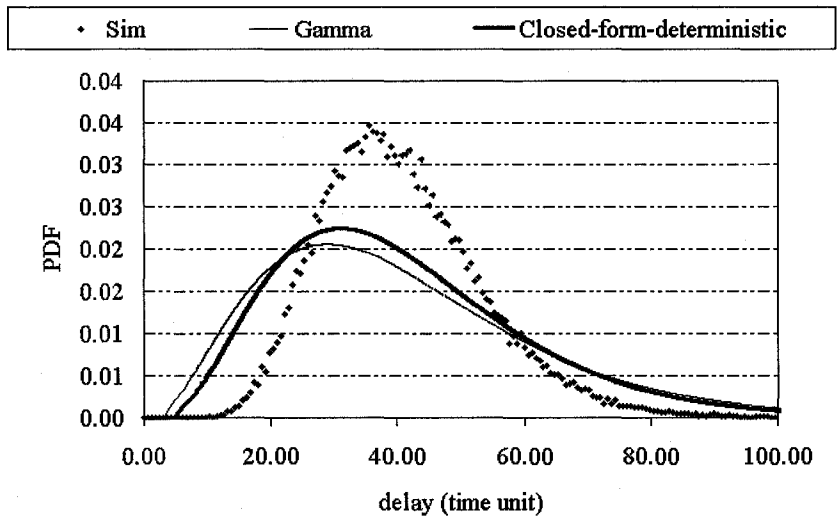
In this section, the accuracy of different distribution fittings is evaluated when a thread makes requests to more than one server. This case relaxes the assumptions made for the closed-form formula for geometric requests and for deterministic requests, because the service times at the server most likely will be different. Figure 5.11 shows the model used, with each forked thread i making requests to two lower layer servers tCi1 and tCi2 for $i = 1, \ldots, 5$. The number of requests to entries tCi1 and tCi2 are represented by the variables $y1$ and $y2$ respectively. The variables $Cx1$ and $Cx2$ represent the execution demands at the servers. The accuracy of the various distribution approximations for both deterministic and geometric requests are shown in Tables 5.4, 5.5, and 5.6. The results will be discussed in the following paragraphs.

Figure 5.12 shows the PDF for the delay of thread 1, that executes activity a1 in Figure 5.11, with $y1 = 2, Cx1 = 15, y2 = 2, Cx2 = 5$. The PDFs are obtained from the simulation run, and from fitting the thread delay to both the gamma distribution, and the closed-form formula derived in this chapter for deterministic requests. In Figure 5.12(a), processor pB1, that executes task tB1, is an infinite server, while in Figure 5.12(b), the

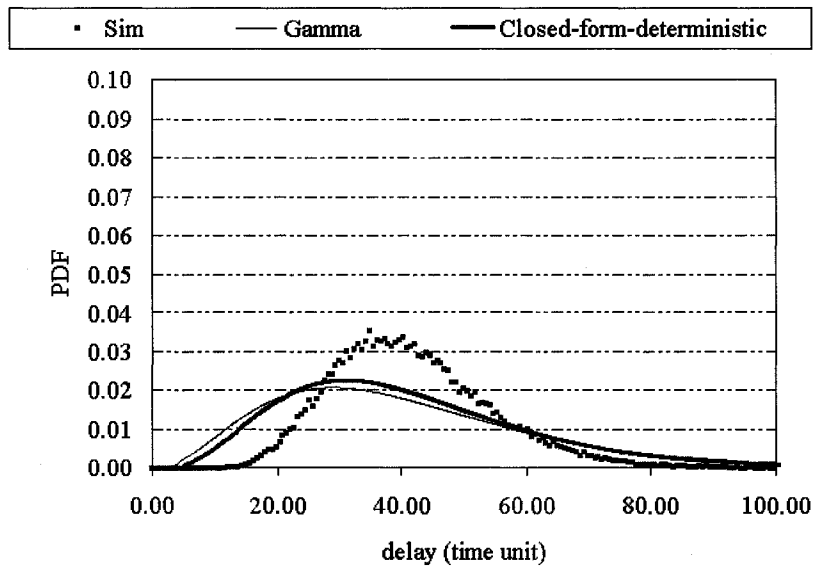
processor is a single server with a processor sharing scheduling discipline. When pB1 is an infinite server, then the assumptions for the closed-form formulas for deterministic requests are satisfied. When pB1 is a single server with a processor sharing discipline, then the forked threads are not independent, hence the independence assumption is violated. The gamma distribution and the closed-form formulas for deterministic requests are very close to each other, but they differ from the simulation. The difference between the closed-form formula and the simulation is due to the error introduced when approximating the distribution of the two branches by combining them into one branch while solving the model using the LQNS.

Table 5.4 shows the result of fitting the thread delays to a three-point and a gamma distribution when the requests $y1$ and $y2$ in Figure 5.11 are deterministic and when 1 out of 5 forked threads responds. Table 5.5 shows the result of fitting the thread delays to the closed-form formulas for determinsitic requests. The LQN model in Figure 5.11 could not be solved using GreatSPN, because the state space is too large to handle. All simulation results have a 95% confidence interval of less than 5% for the mean and less than 10% for the variance. When the LQNS results are compared to the simulation, the results show that the three-point approximation gives an error in the mean of up to about 12%, the gamma distribution fitting gives an absolute value of the percentage error of less than 5%, and the closed-form formulas for determinsitic requests give an absolute value of the percentage error of less than 5%. The results indicate that even in a multi-request model with deterministic requests, the gamma distribution and the new closed-form formulas give the highest accuracy.

Figure 5.13 shows a PDF for the delay of thread 1, that executes activity a1 in Figure 5.11, with $y1 = 2, Cx1 = 15, y2 = 2, Cx2 = 5$, and with geoemtric requests. The PDFs are obtained from the simulation run, and from fitting the delay to the closed-form



(a) Infinite Server



(b) Processor Sharing

Figure 5.12: PDF for the delay of thread 1 in the one-layer two-branch LQN model in Figure 5.11 with deterministic requests. The processor for tB1 is an infinite server in (a), or a single server with a processor sharing discipline in (b). $y1 = 2, Cx1 = 15, y2 = 2, Cx2 = 5$.

Table 5.4: Quorum delay for the two-request branches per thread LQN model in Figure 5.11 with deterministic requests.

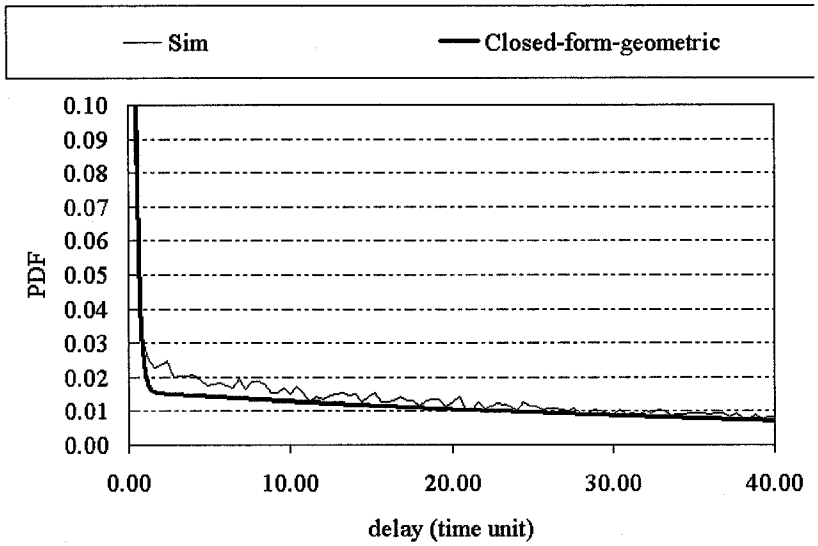
Parameters (y1,Cx1,y2, Cx2)	Sim 95% conf. interval		LQNS % Error			
			Three-point		Gamma	
	mean	variance	mean	variance	mean	variance
(2D,5,2D,5)	10.99±0.064451	16.24±0.27292	11.98	-28.84	0.59	1.52
(2D,10,1D,10)	14.19±0.10727	39.97±1.079	12.46	-13.38	0.86	0.79
(1D,15,2D,5)	11.05±0.077535	25.87±0.6566	4.99	22.68	-4.79	16.43

Table 5.5: Quorum delay using closed-form formulas for the deterministic two-request branches per thread LQN model in Figure 5.11.

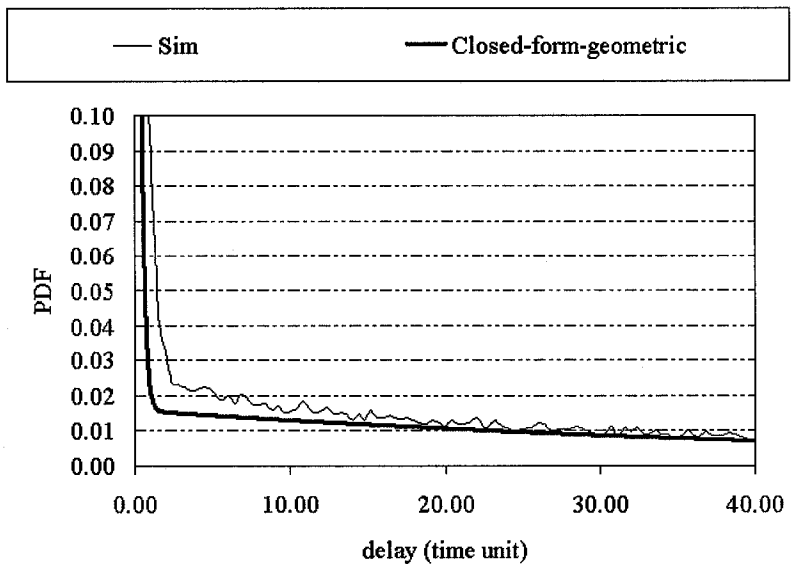
Parameters (y1,Cx1,y2, Cx2)	Sim 95% conf. interval		LQNS % Error Closed-form-deterministic	
	mean	variance	mean	variance
(2D,5,2D,5)	10.99±0.064451	16.24±0.27292	1.26	-2.92
(2D,10,1D,10)	14.19±0.10727	39.97±1.079	1.59	-2.96
(1D,15,2D,5)	11.05±0.077535	25.87±0.6566	10.19	4.54

formula derived in this chapter for geometric requests. In Figure 5.13(a), processor pB1, that executes task tB1, is an infinite server, while in Figure 5.13(b), the processor is a single server with a processor sharing scheduling discipline. When pB1 is an infinite server, then the assumptions for the closed-form formulas for geometric requests are satisfied. When pB1 is a single server with a processor sharing discipline, then there will be a dependency among the RVs associated with the delays of the forked threads, hence the assumptions are violated. The closed-form formula for geometric requests is very close to the simulation in both cases. The difference between the closed-form formula and the simulation is due to the error introduced when approximating the distribution of the two branches by combining them into one branch while solving the model using the LQNS.

Table 5.6 lists the results for the case when the requests $y1$ and $y2$ in Figure 5.11 are geometrically distributed. One of the assumptions of the derivation of the closed-form formula for geometric requests is that each thread makes requests to only one lower layer server. Even though this assumption is violated, the closed-form geometric solution has the best result with an error in the mean less than 23%. Even though this is a relatively high error, it is significantly lower compared to the case when using a gamma or a three-point distribution fitting. Further, the error in the variance for the closed-form formula is more than an order of magnitude smaller than the error when using either a three-point or a



(a) Infinite Server



(b) Processor Sharing

Figure 5.13: PDF for the delay of thread 1 in the one-layer two-branch LQN model in Figure 5.11 with geometric requests. The processor for tB1 is an infinite server in (a), or a single server with a processor sharing discipline in (b). $y1 = 2, Cx1 = 15, y2 = 2, Cx2 = 5$.

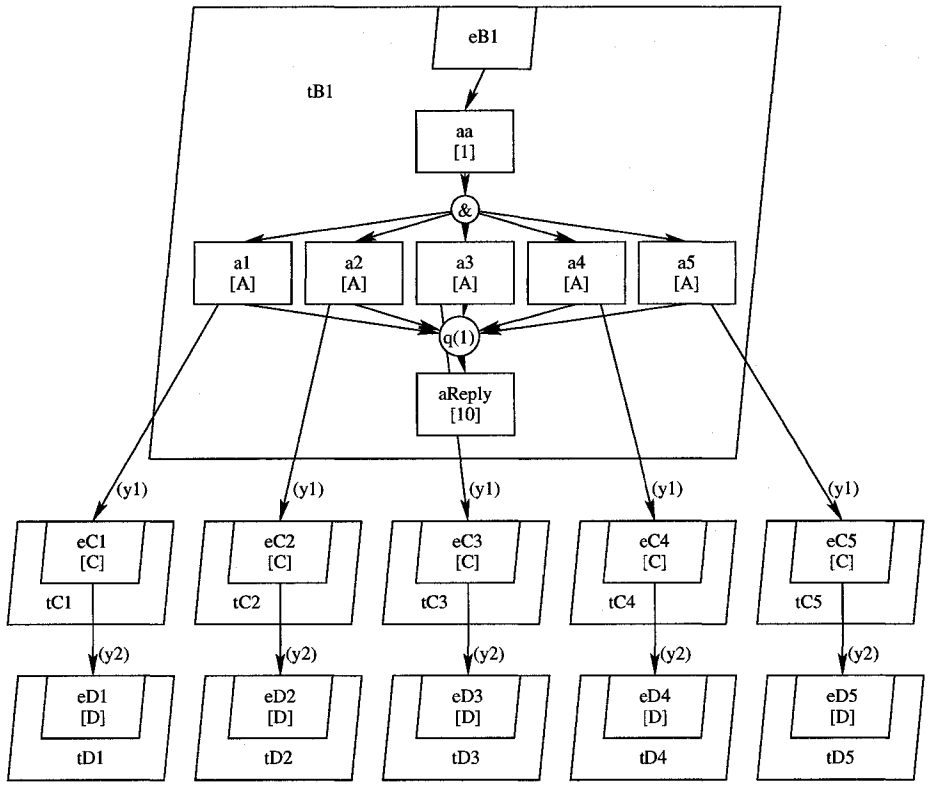


Figure 5.14: An LQN model with two layers.

gamma distribution. It is clear that the gamma distribution and three-point distribution are not suitable for fitting a thread delay with geometric requests.

5.3.3. Effect on Multi-Layer Models.

Figure 5.14 shows an LQN extension to the model in Figure 5.2, where there are two lower layers of servers in the new LQN model. This change is done to show the effect of multiple lower layers of servers on the accuracy of different distribution fittings. Activities ai have execution demands A time units each, entries eCi have execution demands C time units each, and entries eDi have execution demands of D time units each.

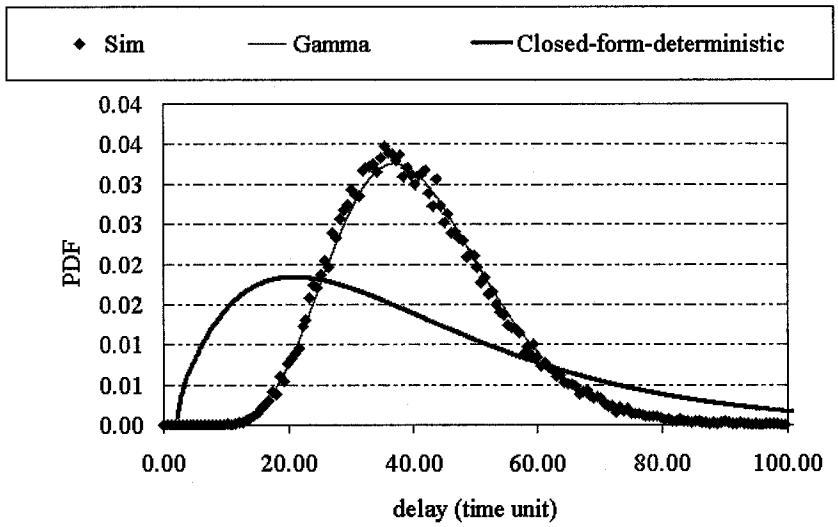
Figure 5.15 shows the PDF for the delay of thread 1, that executes activity a1 in Figure 5.14 with $y1 = 2, y2 = 3, A = 1, C = 5, D = 5$. The PDFs shown in the figure are

Table 5.6: Quorum delay for the two-request branches per thread LQN model in Figure 5.11 with geometric requests.

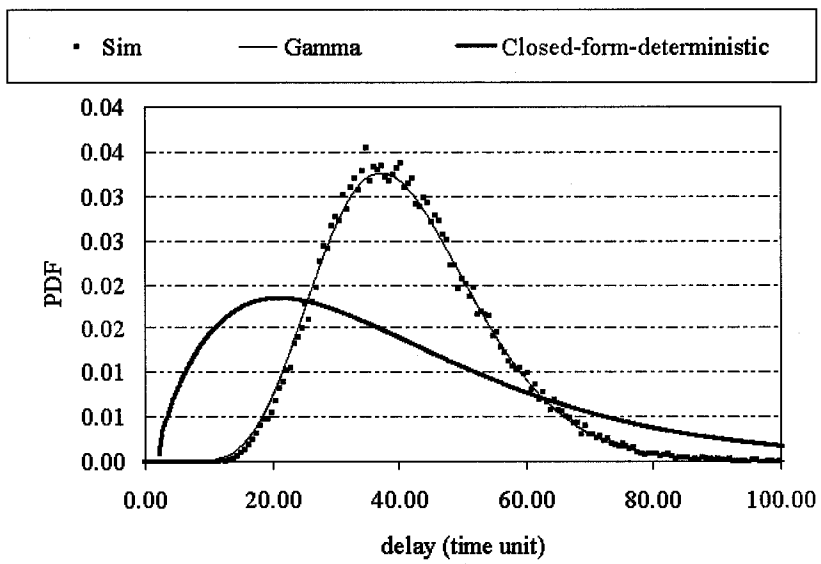
Parameters (y1,Cx1,y2, Cx2)	Sim 95% conf. interval		LQNS % Error					
			Three-point		Gamma		Closed-form-geometric	
	mean	variance	mean	variance	mean	variance	mean	variance
(2,5,2,5)	1.934±0.04409	15.35±0.54517	60.87	219.01	135.16	13.42	5.79	-1.52
(2,10,1,10)	2.1932±0.07262	28.89±2.1433	61.97	239.42	147.86	16.23	8.57	-0.92
(1,15,2,5)	1.6545±0.055799	16.94±1.3255	63.57	273.99	142.63	26.29	22.64	19.14

obtained form the simulation run, and from fitting the thread delay to both the gamma distribution and the closed-form formula derived in this chapter for deterministic requests. In Figure 5.15(a), processor pB1, that executes task tB1, is an infinite server, while in Figure 5.15(b), the processor is a single server with a processor sharing scheduling discipline. In this LQN model, the service time of entry eC1 is non-exponentially distributed, because eC1 makes requests to entry eD1. Therefore, the exponentially distributed service time(s) of the server(s) assumption for the closed-form formulas, derived in this chapter, is violated. That is why the closed-form formula for determinsitic requests does not follow the simulation. However, the gamma distribution follows the simulation, because it takes the mean and the variance of the service time as its parameters.

Tables 5.7 and 5.8 show the result for the quorum delay when requests $y1$, and $y2$ in the LQN model in Figure 5.14 are deterministic. This model could not be solved using a Markov chain, because of the state space explosion problem. The 95% confidence interval for the simulation results for all values of the mean are less than 5%, and for all values of the variance are less than 10%. The results in the tables show that the gamma distribution fitting gives a high accuracy when the requests are deterministic, with an error less than 1% in the mean of the quorum delay. The three-point approximation is a less suitable approximation as it gives an error of up to about 9%. The closed-form formulas derived in this thesis for deterministic requests give the highest error, and it is clear that the formulas cannot be used for performance analysis. The high error is because the closed-form formulas assume the servers' service times are exponentially distributed, which is not the case in this model, while the gamma distribution takes the variance into account. For example, for the case (2D,3D,5,5,1), when using the gamma, the variance of each thread is found to be 26.83 (calculated from the MVA auxiliary model), while using the closed-form formulas for determinsitic requests, the variance is estimated (after fitting the closed-form formula)



(a) Infinite Server



(b) Processor Sharing

Figure 5.15: PDF for the delay of thread 1 in the two-layer LQN model in Figure 5.14 with deterministic requests. The processor for tB1 is an infinite server in (a), or a single server with a processor sharing discipline in (b). $y1 = 2, y2 = 3, A = 1, C = 5, D = 5$.

to be 139.92. The variance 139.92 is overestimated, and it makes the mean of the quorum delay significantly lower. The quorum delay (J^{th} order statistic) is very sensitive for the variance of each individual thread.

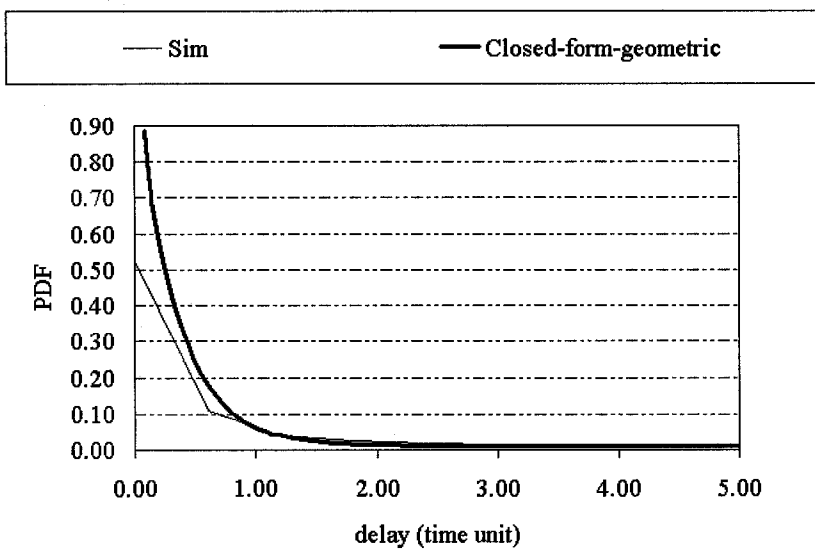
Figure 5.16 shows the PDF for the delay of thread 1, that executes activity a1 in Figure 5.14 with $y1 = 2, y2 = 3, A = 1, C = 5, D = 5$. The PDFs are obtained from the simulation run, and from fitting the thread delay to the closed-form formula derived in this chapter for geometric requests. In Figure 5.16(a), processor pB1, that executes task tB1, is an infinite server, while in Figure 5.16(b), the processor is a single server with a processor sharing scheduling discipline. In this LQN model, the service time of entry eC1 is non-exponentially distributed, because entry eC1 makes requests to entry eD1. Therefore, the exponentially distributed service time(s) of the server(s) assumption for the closed-form formulas derived in this chapter is violated. The results show that in both cases, the closed-form formula for geometric requests follows the simulation. However, when pB1 has a processor sharing scheduling discipline, the PDF differs from the simulation more than when pB1 is an infinite server. This is because in the PS case, one more assumption is violated. That is the independence of the forked threads assumption.

Table 5.9 shows the results for the quorum delay when the requests $y1$ and $y2$ are geometrically distributed. The closed-form formula assumes that the distribution of the RV associated with the service time of the underlying layer is exponentially distributed, a assumption which is violated in this test case. With the exception of one test-case where the error in the mean of the three-point approximation is slightly smaller than the closed-form formula, the closed-form formula is superior.

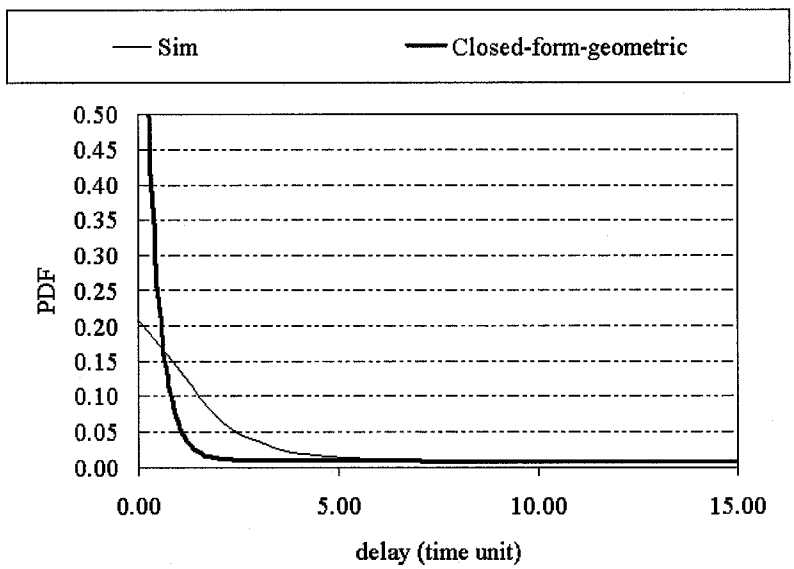
The show the effect of heterogeneous threads on the approximation accuracy, Figure 5.18 shows the accuracy of quorum delay approximation when using geometric requests in the LQN model in Figure 5.17. The scheduling discipline for processor pB1 is processor

Table 5.7: Quorum delay for the two-layer model in Figure 5.14 with deterministic requests.

Parameters (y1,y2,A,C,D)	Sim 95% conf. interval		LQNS % Error			
			Three-point		Gamma	
	mean	variance	mean	variance	mean	variance
(2D,3D,1, 5,5)	27.62±0.016965	35.903±2.0374	8.34	-47.70	0.03	9.13
(2D,3D,5, 5,5)	31.248±0.18335	39.401±2.0251	7.70	-50.01	-0.08	9.18
(2D,3D,5,5,1)	15.373±0.038585	7.521±0.1799	7.35	-58.86	0.68	-1.45
(2D,3D,1,1,5)	23.833±0.1654	33.957±0.68698	9.34	-46.07	0.29	5.67



(a) Infinite Server



(b) Processor Sharing

Figure 5.16: PDF for the delay of thread 1 in the two-layer LQN model in Figure 5.14 with geometric requests. The processor for tB1 is an infinite server in (a), or a single server with a processor sharing discipline in (b). $y1 = 2, y2 = 3, A = 1, C = 5, D = 5$.

Table 5.8: Quorum delay using closed-form formulas for the deterministic requests in the two-layer model in Figure 5.14.

Parameters (y1,y2,A,C,D)	Sim 95% conf. interval		LQNS % Error Closed-form-deterministic	
	mean	variance	mean	variance
(2D,3D,1, 5,5)	27.62±0.016965	35.903±2.0374	-44.51	104.6
(2D,3D,5, 5,5)	31.248±0.18335	39.401±2.0251	-36.30	99.64
(2D,3D,5,5,1)	15.373±0.038585	7.521±0.1799	-34.27	110.74
(2D,3D,1,1,5)	23.833±0.1654	33.957±0.68698	-47.69	38.17

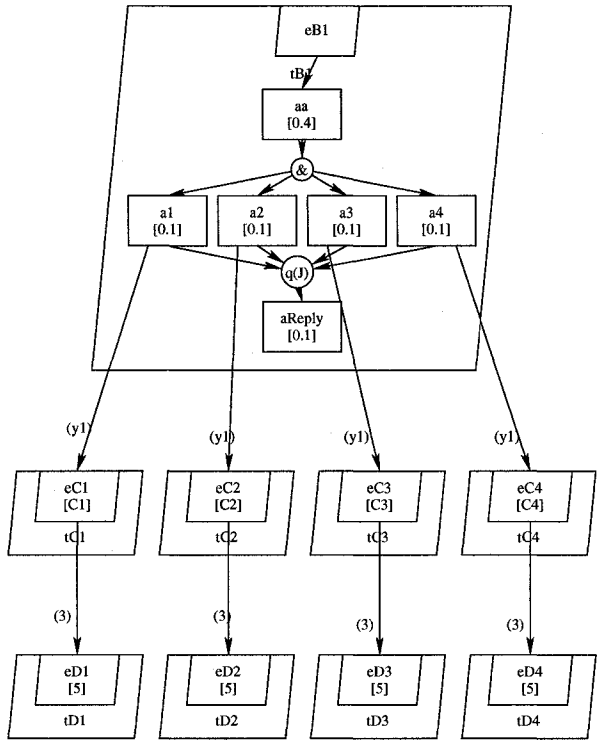


Figure 5.17: Two-layer LQN model with four threads.

sharing. The execution demands for servers **tCi** for $i = 1, \cdots, 4$ are calculated using the formula $Ci = 5 + (i - 1) \times \alpha$, where α is a real number that is a measure of the heterogeneity level. When $\alpha = 0$ all threads are homogeneous. The mean number of

Table 5.9: Quorum delay for the two-layer model in Figure 5.14 with geometric requests.

Parameters (y1,y2,A,C,D)	Sim 95% conf. interval		LQNS % Error					
			Three-point		Gamma		Closed-form-geometric	
	mean	variance	mean	variance	mean	variance	mean	variance
(2,3,1,5,5)	1.5602±0.10925	32.16±4.3527	96.13	261.91	126.17	0.43	37.51	16.14
(2,3,5,5,5)	2.673±0.16107	41.272±6.1296	33.77	259.93	63.77	6.76	21.12	15.75
(2,3,5,5,1)	1.9735±0.046351	11.626±0.97708	-3.52	213.95	31.16	2.42	6.76	0.92
(2,3,1,1,5)	1.1565±0.063273	18.76±2.1451	107.66	292.63	133.24	5.59	52.74	29.46

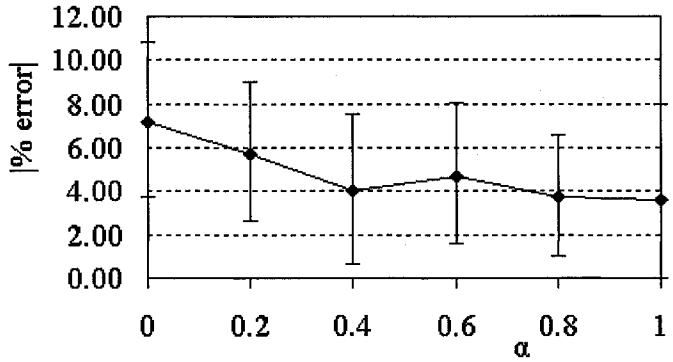


Figure 5.18: Error in the quorum delay in tB1 in the two-layer model in Figure 5.17 with geometric requests. The scheduling discipline for pB1 is PS, $Ci = 5 + (i-1) \times \alpha, y1 = 3 \times (1+\alpha), J = 2$.

requests $y1$ to servers tCi is calculated as using the formula $y1 = 3 \times (1+\alpha)$. The results show that increasing the heterogeneity level decreases the absolute value of the percentage error. The maximum error of about 7% occurs when all threads are homogeneous.

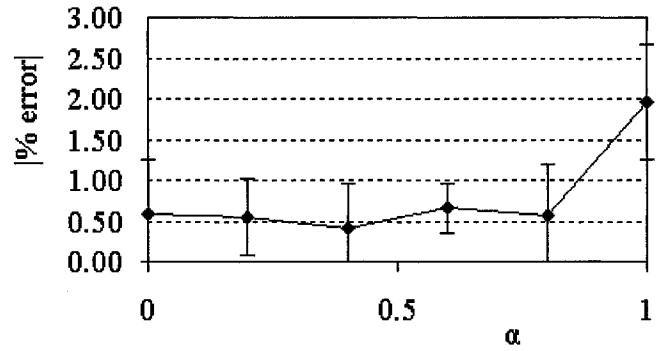
The effect on the accuracy of the quorum delay of the forked threads when deterministic requests are made to servers tCi for $i = 1, \cdots, 4$ (see Figure 5.17) is shown in Figure 5.19. Figure 5.19(a) shows the percentage of the absolute value of the percentage error when the gamma distribution fitting is used to approximate the distribution of a thread delay. Figure 5.19(b) shows the results when the closed-form formula for deterministic requests fitting is used to approximate the distribution of the thread delay. The results are evaluated for the case when $J = 3$. The number of requests to remote servers tCi is calculated using the formula $y1 = \lceil 3 \times (1+\alpha)) \rceil$, where α is a measure of the heterogeneity of the forked threads. The results show that when the gamma distribution fitting is used, the error in the quorum delay is less than 2.5%, while it is less than 5% when the closed-form formula for deterministic requests fitting is used. Note that the closed-form formula for determinsitic requests assumes that the service times of the remote servers are exponentially distributed,

which is not the case in this model, while the gamma distribution takes the variance of the remote servers into account. The results show whether the gamma distribution or the closed-form formula for determinsitic requests is used, the absolute value of the percentage error tends to decrease as the heterogeneity increases among the forked threads. In Figure 5.19, for the case when $\alpha = 1$, the error has increased; this might be because of the approximation when fitting the thread delay distributions. Note that the increase in the absolute value of the percentage error is within the acceptable range of errors which is less than 5% in this case.

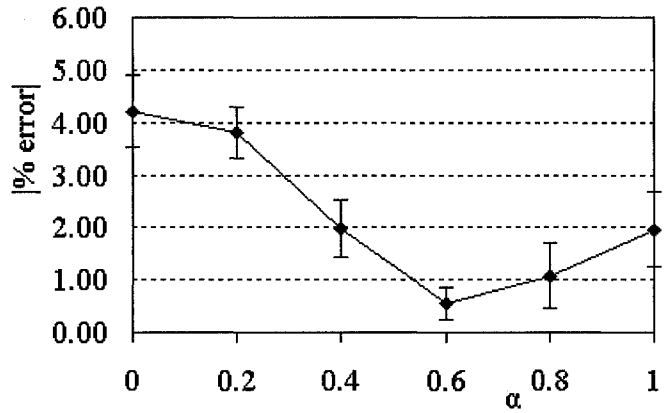
5.4. Summary

Generally, computing the distribution of the quorum delay, and therefore its mean and variance, by means of a J^{th} order statistic is very sensitive to the first part of the CDF of the delay of each thread involved in the quorum-join. The computation of the quorum delay is most sensitive when the minimum number of threads' responses is required, i.e. $J = 1$, and is least sensitive when the maximum of the threads' responses is required, i.e. $J = N$. The latter case is simply the AND-join of all of the threads and is most dependent on the tails of the distributions.

The application of the new approach on a variety of test cases shows that the approach achieves errors less than 10% (except the closed-form formulas when used in a multi-layer model) in most cases when compared to simulation results. In particular, two groups of models are examined: models in which the number of requests to lower layer servers is geometrically distributed and models in which the number of requests is deterministically distributed. For the first case, the accuracy in the mean and variance of the quorum delay when using the closed-form formula for the CDF of a thread delay is clearly superior to the



(a) Gamma distribution fitting



(b) Closed-form-deterministic fitting

Figure 5.19: Error in the quorum delay in tB1 in the two-layer model in Figure 5.17 with deterministic requests. The scheduling discipline for pB1 is PS. $Ci = 5 + 10 \times (i-1) \times \alpha, y1 = \lceil 3 \times (1+\alpha) \rceil, J = 3$.

accuracy when using both three-point and gamma approximations of the CDF. For the second case, the newly proposed gamma approximation is better than the three-point by virtue of the smaller errors in the variance; the errors in the mean are close in both cases. The new closed-form formulas for deterministic requests is comparable to the gamma distribution fitting, but they give high errors when the service times of the servers are non-exponentially

distributed. Therefore, in the following chapters, the gamma distribution will be used for the fitting of the thread delay for deterministic requests.

For the closed-form formula for geometric requests, even if the assumptions required for the closed-form derivation are violated (i.e., the number of servers a client makes requests to is more than one and the servers' service times are non-exponential) the errors are mostly acceptable and are again far smaller than when three-point or gamma distributions are used.

CHAPTER 6

Performance Modeling of a Quorum Pattern in Layered Service Systems

This chapter approximates the performance of a system with a quorum consensus section that executes N requests in parallel, but terminates after J responses are received. In Chapter 5, the cumulative distribution function for the delay of each forked thread was derived, and the accuracy of the approximations was evaluated by computing the time to meet the quorum. In this chapter, the derived CDFs will be used in a model to approximate the service time of an entry in an LQN task that has forked threads. The service time is affected by both the delay to meet the quorum, and the contention and delay caused by the delayed threads. The approximation error in service times of entries, in the analytic solutions compared to simulations, in some dozens of LQN model examples is mostly less than 10% and always less than 25%.

6.1. Introduction

Quorum consensus (QC) or voting is used in systems with replicated components to enhance reliability and performance. The QC considered here are those in which a request is translated to a set of requests to N replicas. The translated request is satisfied when J out of N positive responses are received. Special cases arise with $J = 1$ in which the first response is taken, with $J = \lfloor N/2 \rfloor + 1$ where a majority is required, and $J = N$ which ensures that all replicas have completed the operation.

The performance of a system with behavior illustrated in Figure 6.1(a) is studied in this chapter. An application App serving a request includes a QC section shown by parallel threads terminating in a node labeled *q(J)*. The QC section is preceded by a set of operations represented by an activity labeled as aPre, and followed by operations represented by aPost. The QC section spawns N threads labeled aThreadi ($N = 5$ in Figure 6.1(a)) and requires $J = 3$ responses, indicated by the notation q(3) in the node terminating the section. The 5 threads make requests into an external service subsystem indicated by a cloud, which may include corresponding replicated servers, possibly in a complex layered structure.

The thread completion delays $X_{Thread,i}$, $i = 1, ..., 5$ are illustrated in Figure 6.1(b), showing the time at which the quorum of three terminations is satisfied. The threads left out of the quorum are expressed to *overhang*. Two different possibilities exist for the overhanging threads:

(1) the overhanging threads can all be aborted, although this action may be difficult to accomplish in practice, as the overhanging threads may be blocked on lower layer servers, which in turn will have to be aborted. Aborting the execution of requests in a complex service structure is itself complicated, and may weaken the correctness of the state of the remaining processes. In a replication system, voting termination needs an additional round of messages to synchronize the replicas [**160**]. This round can be as complicated as the atomic commitment protocol, E.g., the two-phase commit protocol [**130**]. Conversely, the round of messages can be a single confirmation message sent by a replica.

(2) the overhanging threads may continue to execute, with the parent thread ignoring their results. If the parent thread finishes and accepts new requests while the overhanging threads are still executing, there is a possibility of exhausting resources

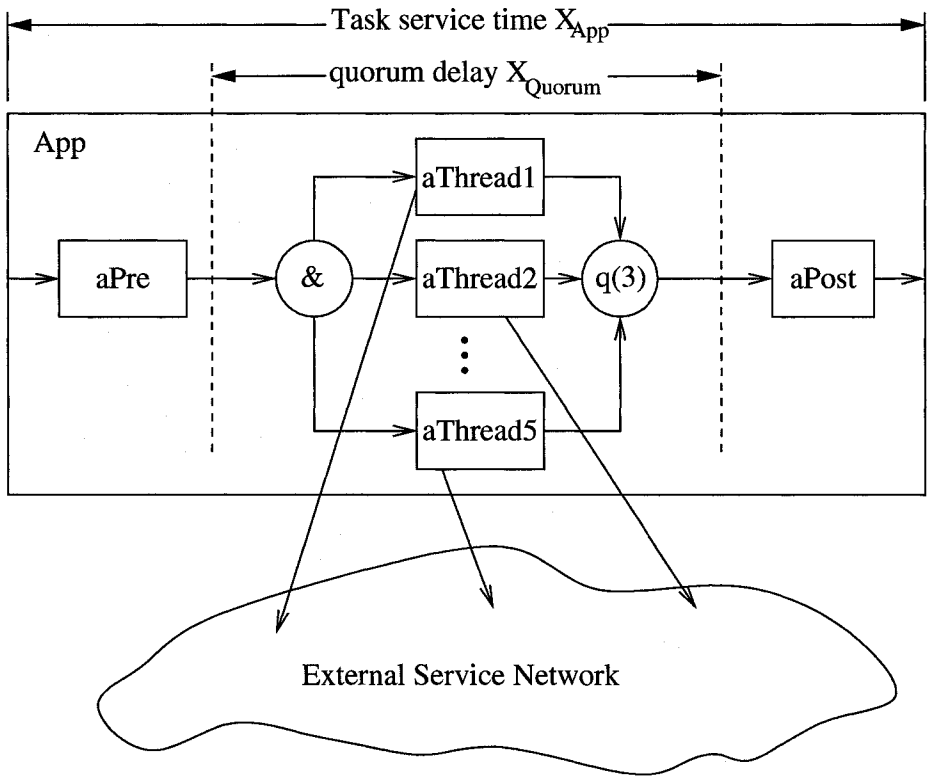
as overhanging threads may continue to build up in the system. Alternatively, the parent thread may wait until all of its children or overhanging threads complete, i.e., an implicit join at the end of the parent thread. The rationale for requiring that all threads run to completion is that it may not be possible to cancel requests to external servers. They may not have the capability to accept cancellations, or a cancellation might leave a server in an inconsistent state. Web services, for instance, may well have these limitations.

In this thesis, it is assumed that the overhanging threads are allowed to run to completion (an implicit join at the end of the parent thread). The remaining time until all N threads are finished is termed here the *overhang delay* denoted $X_{Overhang}$; it blocks the Application and delays the moment when the Application can accept another request.

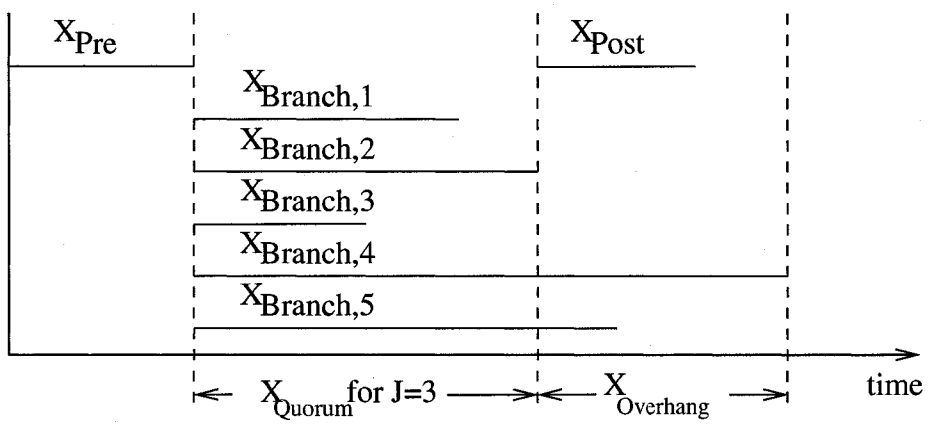
The derivation of the distribution function of a thread delay that can be used to compute the quorum delay X_{Quorum} was described in Chapter 5. The overhang delay can have a severe effect on performance, and must be accounted for. Each combination of overhanging threads creates a distinct contention situation in the service subsystem. A full analysis of all these contention situations may lead to exploding complexity, especially as there may be more than one QC section in a system. This work seeks an appropriate approximation to the overhang delay $X_{Overhang}$ and to its effect on the application service time. Then, it computes App service time X_{App} as:

$$X_{App} = X_{Pre} + X_{Quorum} + max(X_{Post}, X_{Overhang}), \tag{6.1}$$

where X_{Quorum} is the delay for the QC section, X_{Pre} and X_{Post} are the delays of the defined activities before and after it. The approximation developed here is implemented in the LQNS.



(a)



(b)

Figure 6.1: Behavior of a program with a quorum consensus section.

6.2. The Model for Quorum Delays

The notation in Figure 6.1(a) is called "activity graphs" in [55] and represents well-known task graph semantics, with boxes for activities (representing sequential computations), sequence indicated by arcs, and forks indicated by circles with a plus sign for OR (alternative path) forks and joins, and an ampersand for AND (parallel) forks and joins. A quorum-join with a quorum of J completions is illustrated by a circled q(J), illustrated by q(3) in Figure 6.1(a).

An activity graph models a thread of execution within a process (called here a *task*), and a parallel activity represents a forked thread. True physical parallelism requires that the forked thread make a request to some concurrent, physically parallel task (which is the case in real parallel software), and these requests are represented in Figure 6.1(a) by arrows to the "concurrent subsystem" indicated as a cloud.

Since the activity graph models execution in response to a request to a task, at some point a response must be sent to the requester. The default is to send the response when the last activity terminates; however it is possible for some activity in the middle of the graph, called a *reply-activity*, to explicitly send the response. For performance computations, the important delays are from initiation of the first activity until the response, and until termination of all activities in the graph. This thesis is concerned with the latter delay, which is the service time (denoted X_{App} in Figure 6.1(a) and in Eq. (6.1)).

6.2.1. Quorum Consensus Sections, and Overhanging Thread Semantics.

In the quorum behavior being considered, a QC with a preset quorum J forks N threads with activities that optionally may read or update N external concurrent tasks. Once J activities have terminated, the main thread of control resumes execution, and completes its remaining work. This may be an arbitrary subgraph of activities, represented in Figure 6.1(a) by the activity aPost.

The remaining $N - J$ *overhanging threads* that do not participate in the quorum are allowed to run to completion. This means that they contend with aPost, and the task is occupied with the response until the last activity terminates. The service time of the task, which is the delay from accepting a request until App is ready to accept another, is denoted by X_{App}.

The default timing of a service reply is affected by a QC section. When $J < N$, it comes after the termination of the last explicit activity in the graph (E.g., after aPost in Figure 6.1(a)). There may be a period after the reply when the task is still busy, caused by the overhang, and this extra delay is accommodated in layered queueing with the concept of a *"second phase"*.

6.3. Solution Strategy

An analytical solution is sought for system performance in the hope that it will be faster than simulation. One approach, not used here, would be to construct a Markov chain model for all the states of the system. This approach suffers from state explosion due to large numbers of queue states, even for moderate levels of concurrency, and due to different states of contention for the different numbers of active parallel threads.

Queueing models and Mean Value Analysis provide more scalable solutions. However, the number of customers in a queueing model must be constant while the quorum construct

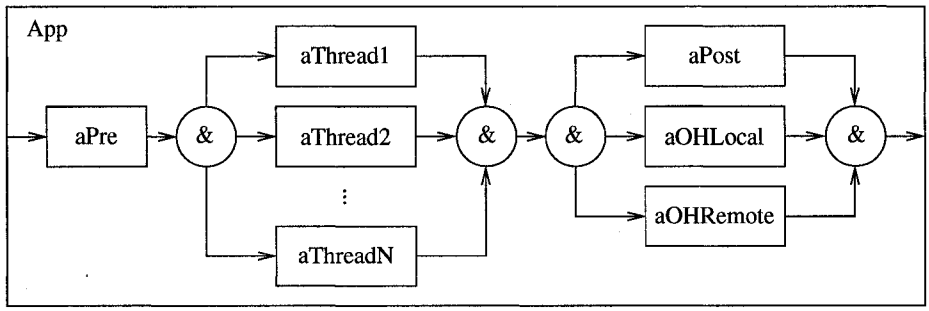


Figure 6.2: Model M': the transformed activity graph for the model in Figure 6.1(a). M' is constructed in this way to account for contention of threads for resources, but the moments of App are calculated using Eqs. (6.10), and (6.11).

dynamically creates customers for the forked threads. The present approach uses layered queueing (to capture the contention and the layered aspects of the system structure), and converts a model M with the quorum construct to an approximate model M' with only AND-joins. Existing solution techniques are then applied. One method, not used here, to compute the performance measure of a model with a quorum construct is to *decompose* each quorum construct of M around all possible combinations of active threads and combine the results. The method is costly because it solves a combinatorial number of models M'. Note that the number of models grows exponentially with the number of quorum sections used in the original model.

In this thesis, the activity graph of a task with a QC section, such as task App in Figure 6.1(a), is replaced in M' with another activity graph as shown in Figure 6.2. The behavior of the QC section is changed to a full parallel section (i.e., the quorum becomes $q(N)$, denoted "&" in Figure 6.2), followed by the behavior during the overhanging period described by a second parallel section with surrogate activities for the overhanging threads. The surrogate activities aOHLocal and aOHRemote are constructed to account for the delay contribution of the overhanging threads. The model M' is constructed in this way to account for contention of threads for resources using the existing LQN constructs. This

double-counts the effect of the overhanging operations. The model M' gives the correct results, except for the service time of task App, which includes the overhang twice. The double-counting is corrected by a special computation for X_{App} described in Section 6.4, using values of the means and variances of the RVs X_{Pre}, $X_{Thread,i}$, and X_{Post} provided by the contention analysis in the LQNS solver. For notation, the RV is X_{Pre}, its mean value is $\mathrm{E}[X_{Pre}]$, and its variance is $\mathrm{Var}[X_{Pre}]$. The same conventions are applied to other variables related to measures of activities and tasks.

In general, the activities aPre and aPost may be replaced by arbitrary activity subgraphs, which does not change the strategy described above. A task may only have one quorum section with $J < N$, but a system may have multiple quorum sections.

6.3.1. Reply-activity Semantics in LQN.

The *reply-activity* can be indicated explicitly by a dashed arc from the activity back to the invoking entry. The response is sent after termination of the reply-activity. However, in Figure 6.5 the reply-activity is implicit (no dashed arc is needed). Having an implicit response indicates that a response is sent to entry eB1 when all threads complete execution. A task with no defined activity graph has an assumed graph with one or two activities for each entry, with the first activity being the reply-activity for that entry. The second activity, if present, is called a *second phase* of service for the entry. From the point of view of the task as a server, the second phase keeps the server busy after a customer has departed (after the response). In queueing theory, such as server is known as a "walking server", and the LQNS includes approximate mean value algorithms for servers with a second phase. Second phases may be explicit or arise from unfinished parallel threads such as the overhanging threads in the current work.

6.4. Solving Quorum Models Analytically

Parallel sections (including QC sections) give rise to a routing chain for each thread, with a correction applied to their contention because of constraints that arise in parallel operations [**102**]. The thread delays $X_{Thread,i}$'s are the sum of the delays for all of the requests made by activities in the thread to servers (including its host processor and external software servers). These service delays are computed in other submodels, and give the mean and variance of the thread delay. The variance calculation is an extension to the MVA [**172, 133, 166**].

The overall delay for a parallel section is then found by computing the mean of the maximum thread delay. First the distribution of each thread delay is approximated from its mean and variance. One method of approximation is the three-point approximation [**53**] which was found in Chapter 5 to be inadequate for a J^{th} order statistic computation when $J \ll N$. Another approximation to the distribution function of a thread delay was derived in Chapter 5, and is used here.

In this section, a solution technique to compute the performance measures of an LQN task entry, that has a QC section, is developed.

6.4.1. Mathematical Formulation.

The goal of this section is to approximate $\mathrm{E}[X_{App}]$, and $\mathrm{Var}[X_{App}]$. X_{Pre} and X_{Post} are the RVs for the delays for activities before and after the QC section. $X_{Thread,i}$ is the delay of the i^{th} thread of the QC section, and X_{Quorum} is the quorum delay from the fork to the termination of the J^{th} thread, $X_{Quorum} = OS(J, \{X_{Thread,i}\}_{i=1}^{N})$.

The analysis of X_{App} will use the details of thread execution, illustrated in Figure 6.3. In particular, the host processor service time for thread i, denoted $R_{Local,i}$, and the blocking delay, denoted $R_{Remote,i}$, are needed. Figure 6.3 shows that the thread delay is made up

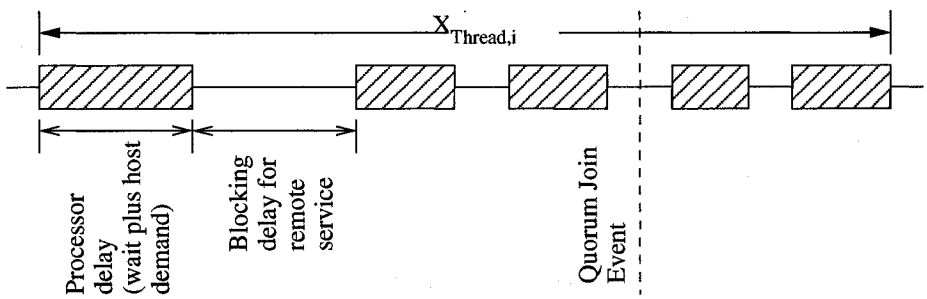


Figure 6.3: Host processor delays and remote blocking delays of a thread.

of host execution intervals (the shaded rectangles) with blocking delays between them. $R_{Local,i}$ is the total of the shaded rectangles, and $R_{Remote,i}$ is the total of the spaces.

- $S_{OHLocal}$ is the total host demand for overhanging execution.
- $X_{OHLocal}$ is the local host processor service time for the activities executed by the overhanging threads after the quorum-join event, and totaled for all overhanging executions.
- $X_{OHRemote}$ is the total blocking time for remote requests, for all overhanging executions.

The transformed model represents the overhang by two surrogate activities aOHLocal and aOHRemote, whose parameters relate the thread local and remote delays. aOHLocal has a host demand $S_{OHLocal}$ equal to the total overhanging host demand, which is found as follows. In Figure 6.4, the host demands of the threads are shown, and they are ordered in magnitude. If all N demands were presented to the processor at the beginning, the longest service time would be the time to complete the total demands (N^{th} order statistic). This time is used to give an approximate value for $S_{OHLocal}$. Similarly, if the order of host demands is the same as the order of total thread delays (host demand plus blocking delay on remote server(s)), the total host demand used up to the quorum-join event is given by the J^{th} order statistic. Then, the total host demand after the quorum point is the difference:

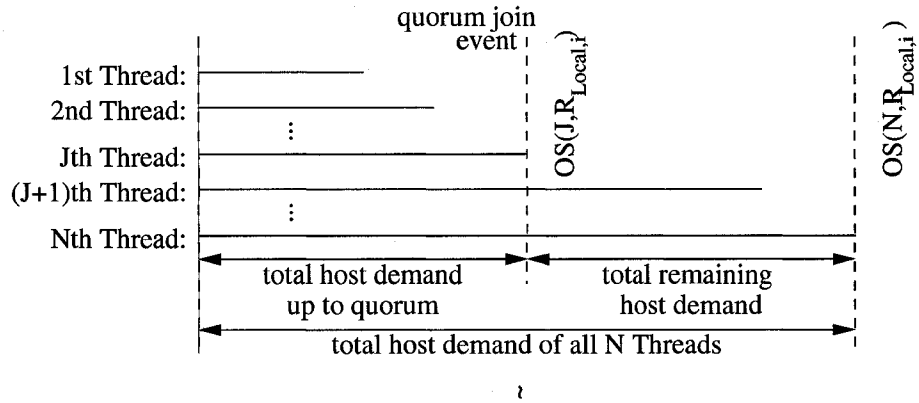


Figure 6.4: Determination of the overhanging host demand.

$$S_{OHLocal} = OS(N, \{R_{Local,i}\}_{i=1}^{N}) - OS(J, \{R_{Local,i}\}_{i=1}^{N}). \tag{6.2}$$

The mean is given by:

$$\mathrm{E}[S_{OHLocal}] = \mathrm{E}[OS(N, \{R_{Local,i}\}_{i=1}^{N})] - \mathrm{E}[OS(J, \{R_{Local,i}\}_{i=1}^{N})]. \tag{6.3}$$

Assuming that the two terms in Eq. (6.2) are independent,

$$\mathrm{Var}[S_{OHLocal}] = \mathrm{Var}[OS(N, \{R_{Local,i}\}_{i=1}^{N})] + \mathrm{Var}[OS(J, \{R_{Local,i}\}_{i=1}^{N})]. \tag{6.4}$$

When the model is solved using approximate MVA, aOHLocal has the local-effect delay $X_{OHLocal}$.

The remote-effect delay $X_{OHRemote}$ is computed by the MVA solver, using its execution demand $S_{OHRemote}$. The delay $S_{OHRemote}$ is found directly from the thread delays, by assuming that they are entirely made up of blocking intervals. In such a case, $S_{OHRemote}$

would just be the difference between the N^{th} order statistic and the J^{th} order statistic for the remote delays, giving:

$$S_{OHRemote} = OS(N, \{R_{Remote,i}\}_{i=1}^{N}) - OS(J, \{R_{Remote,i}\}_{i=1}^{N}). \tag{6.5}$$

The mean is given by:

$$\mathrm{E}[S_{OHRemote}] = \mathrm{E}[OS(N, \{R_{Remote,i}\}_{i=1}^{N})] - \mathrm{E}[OS(J, \{R_{Remote,i}\}_{i=1}^{N})]. \tag{6.6}$$

Assuming that the two terms in Eq. (6.5) are independent,

$$\mathrm{Var}[S_{OHRemote}] = \mathrm{Var}[OS(N, \{R_{Remote,i}\}_{i=1}^{N})] + \mathrm{Var}[OS(J, \{R_{Remote,i}\}_{i=1}^{N})]. \tag{6.7}$$

Finally, the activity (or activity subgraph) aPre has a total delay X_{Pre}, and aPost has a total delay X_{Post}, both found by the MVA.

The local delay is exact for the extreme case where there are no remote requests, and the remote delay is exact for the extreme case where the local execution demands are zero. All other cases lie between these extremes, and are approximated by combining the two partly in parallel and partly sequentially. A fraction of $X_{OHLocal}$ is transferred to $X_{OHRemote}$ on the grounds that it is sequential with the remote delays. This fraction should be zero when there are no remote requests, and one when all threads make remote requests, and also depends on the amount of the local delay. The fraction transferred is defined heuristically as:

$$P_{seq} = \frac{\sum_{i \in A} R_{Local,i}}{\sum_{i \in \{1,\ldots,N\}} R_{Local,i}}, \tag{6.8}$$

where A is the set of indices of threads that make requests to external services. This definition improved the approximation.

Finally, the application delay calculation corresponding to Eq. (6.1) can be written as:

$$\begin{aligned} X_{App} &= X_{Pre} + OS(J, \{X_{Thread,i}\}_{i=1}^{N}) \\ &\quad + OS(3, \{X_{post}, X_{OHLocal} \cdot (1 - P_{seq}), \\ &\quad X_{OHLocal} \cdot P_{seq} + X_{OHRemote}\}). \end{aligned} \tag{6.9}$$

The solver uses the mean and variance of X_{App}. Assuming the terms in the equation above are independent,

$$\begin{aligned} \mathrm{E}[X_{App}] &= \mathrm{E}[X_{Pre}] + \mathrm{E}[OS(J, \{X_{Branch,i}\}_{i=1}^{N})] \\ &\quad + \mathrm{E}[OS(3, \{X_{Post}, X_{OHLocal} \cdot (1 - P_{seq}), \\ &\quad X_{OHLocal} \cdot P_{seq} + X_{OHRemote}\})], \end{aligned} \tag{6.10}$$

$$\begin{aligned} \mathrm{Var}[X_{App}] &= \mathrm{Var}[X_{Pre}] + \mathrm{Var}[OS(J, \{X_{Thread,i}\}_{i=1}^{N})] \\ &\quad + \mathrm{Var}[OS(3, \{X_{Post}, X_{OHLocal} \cdot (1 - P_{seq}), \\ &\quad X_{OHLocal} \cdot P_{seq} + X_{OHRemote}\})]. \end{aligned} \tag{6.11}$$

The moments of the order statistics require distribution information, which is considered next. These calculations are summarized in Algorithm 6.4.1.

ALGORITHM 6.4.1. *Solution of an LQN model with quorum consensus sections.*

- FOR each task with a quorum, construct the transformed model in Figure 6.2. Initialize aOHRemote and aOHLocal to zeros.
- WHILE {convergence in delay, incurred at servers in other submodels, is not met}
  - ✧ GO to next layer submodel and DO the following:
    (1) Solve the submodel as a queueing network.
    (2) IF a task has a quorum section THEN
      (a) obtain the moments of X_{Pre}, X_{Post}, $X_{Thread,i}$, $X_{OHLocal}$, $X_{OHRemote}$ for all threads from the current iteration of the solver.
      (b) calculate the moments for $R_{Local,i}$'s, and $R_{Remote,i}$'s, as in Subsection 6.4.2.
      (c) fit distributions to these moments, as in Chapter 5.
      (d) calculate P_{seq}.
      (e) find the moments of $S_{OHLocal}$, $S_{OHRemote}$ and set them as parameters of aOHLocal and aOHRemote.
    (3) Use Eqs. (6.10), and (6.11) to compute $\mathrm{E}[X_{App}]$, and $\mathrm{Var}[X_{App}]$. Then, set them for the entry of task App.
- LOOP

6.4.2. Delay Distributions.

In order to use Eqs. (6.10) and (6.11), the distribution function for each RV must be found or approximated. The distributions are needed to numerically compute the order statistic terms.

In this thesis, it is assumed that X_{Pre} and X_{Post} have exponential distributions with means determined by the queueing analysis. The distribution of $X_{Thread,i}$, for $i = 1, \cdots, N$,

is determined as in Chapter 5 from the mean and variance found by the queueing computations, is either as a gamma distribution or a rational distribution based on the properties of the local and remote intervals in Figure 6.3.

The same technique, presented in Chapter 5, will be applied for the distributions of $R_{Remote,i}$, and $R_{Local,i}$. However variances of these RVs are not provided by the queueing computation and need to be determined. This is done in the following sub-sections based on whether the number of requests (to the local host processor or to external server(s)) is deterministic or geometrically distributed.

Since there is not enough information about the distributions of $X_{OHLocal}$ and $X_{OHRemote}$, the distributions are always fitted to a gamma distribution, irrespective of the types of requests of the threads. Given the mean and variance of a RV, the shape and scale of a gamma distribution can be obtained as follows (see Definition 5.2.1):

shape $i = \mathrm{E}[X]^2/\mathrm{Var}[X]$, and scale $\theta = \mathrm{Var}[X]/\mathrm{E}[X]$.

6.4.2.1. *Deterministic Requests to Lower Services.*

In Chapter 5, it was concluded that a gamma distribution is often a good fit for the delay of a thread that makes a deterministic number k of requests to external servers. As it is seen from Figure 6.3, the delay of a thread includes $k+1$ processor delays, each of mean $\mathrm{E}[R_{Local,i}]/(k+1)$. Each processor delay is assumed to be exponentially distributed. The sum has a gamma distribution with mean $\mathrm{E}[R_{Local,i}]$, and variance

$$\mathrm{Var}[R_{Local,i}] = \mathrm{E}[R_{Local,i}]^2/(k+1). \tag{6.12}$$

Further, it includes external services with mean $\mathrm{E}[R_{Remote,i}]$, and an unknown variance which is approximated as:

$$\mathrm{Var}[R_{Remote,i}] = max(\mathrm{Var}[X_{Thread,i}] - \mathrm{Var}[R_{Local,i}],\ 0). \tag{6.13}$$

Its distribution is taken as a gamma with these mean and variance. These distributions of all thread delays now enter the order statistics computations for the quorum delay.

6.4.2.2. *Geometric Requests to Lower Services.*

In Chapter 5, a closed-form formula for the distribution of the delay of a thread that makes a geometric number of requests to external servers was derived. $\mathrm{E}[R_{Local,i}]$, $\mathrm{E}[R_{Remote,i}]$, $\mathrm{E}[X_{Thread,i}]$, and $\mathrm{Var}[X_{Thread,i}]$ are calculated by the queueing computations. It is known that the RV for the sum of a geometric number of independent and identical (iid) exponentially distributed RVs is exponentially distributed. Therefore, the local part of a thread makes a geometric number of requests to its host processor with the service time of each request being exponentially distributed with mean $\mathrm{E}[R_{Local,i}]$.

The distribution function of the remote delay is approximated by fitting the mean of the delay and the mean number of requests to the closed-form formula for geometric requests derived in Chapter 5.

6.5. Results and Analysis

In this section, the analytic results from solving different models using the modified LQNS are compared to simulations to demonstrate the accuracy of the new approximations. The models shown here were chosen to stress the analytic approximations in different ways. The number of server layers in an LQN model are varied. Further, as the computation of the delay distribution depends on the type of requests between a client and its servers, most test cases are run with both geometrically and deterministically distributed requests. Finally, the scheduling discipline of the processor running the client task that has the quorum section

is varied for some of the test cases. For each test case, simulations were run with results having 95% confidence intervals of no more than ±4%. The errors reported are calculated using $\left|\frac{\text{analytic}-\text{simulation}}{\text{simulation}}\right| \times 100\%$. Based on my experience with the solution technique, the highest errors in almost all cases occur when the first order statistic is computed (when $J = 1$).

Two assumptions may affect the accuracy of the approximate distributions if they are violated (see Chapter 5).

(1) Remote Service: the individual external service delays must be exponentially distributed, and independent.

(2) Local Service: the queueing delay on the processor that runs the quorum-join is insignificant. This is satisfied by making the processor an infinite processor. It will in general be satisfied if the load on the host processor that is running the forked threads is low.

6.5.1. One-Layer Models.

Figure 6.5 shows an LQN model that consists of only one layer of servers below a client with a quorum-join. Each server task accepts requests from only one forked thread. The processor for task tB1 is an infinite server, so there is no blocking in the model. Both the Remote and Local Service assumptions above are satisfied.

Figure 6.6 shows the absolute value of the percentage error in the service time for entry eB1 for different values of y, A, C, and J in Figure 6.5. When the requests y to the lower layer server are geometrically distributed, the error for the parameters shown are less than 2.5%. When the requests are deterministic, the error is less than 14%. The gamma distribution is used to fit the distribution of the deterministic requests, while the closed-form formula for geometric requests (derived in Chapter 5) is used to fit the distribution of the delay for geometric requests.

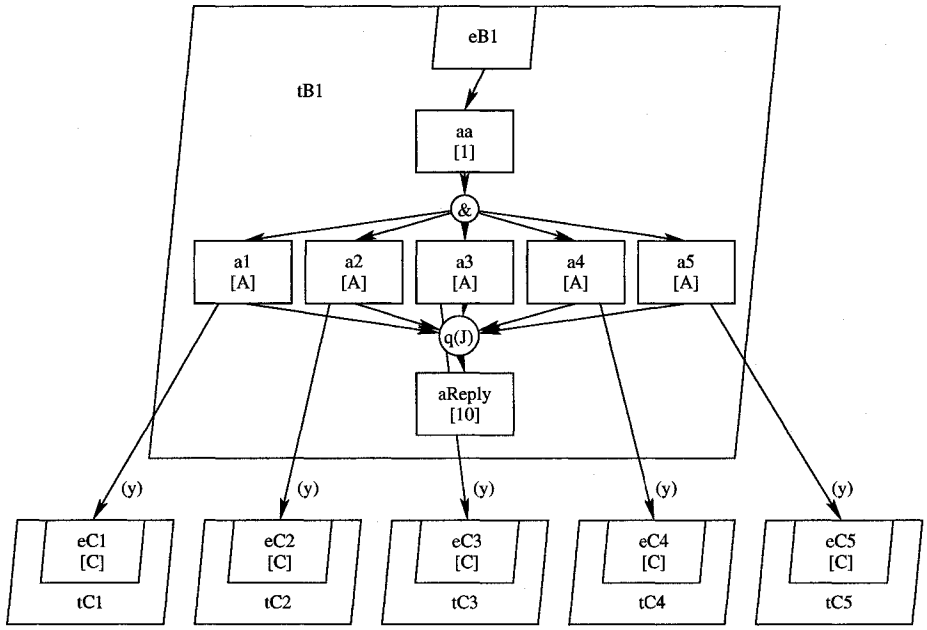
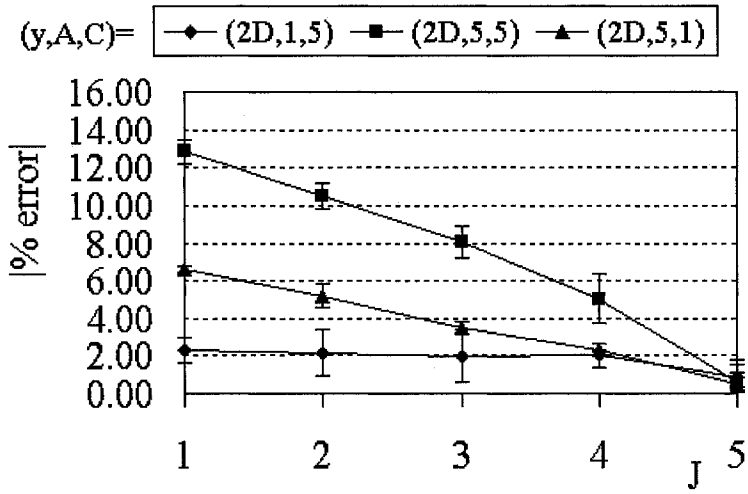


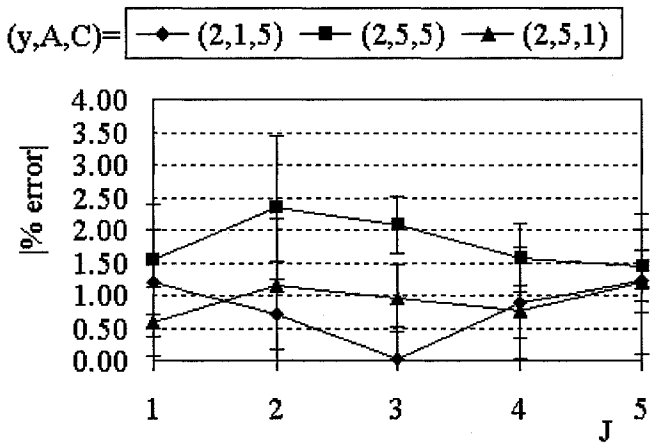
Figure 6.5: One-layer LQN model with five parallel threads. There is a processor for each task, which is not shown.

Figure 6.7 shows an LQN model in which the local execution demand of each forked thread is 0.1 time units and the execution demand of activity aReply is 0.1 time units. Processor pB1 has a multiplicity of m which will be varied to show the effect of multiplicity of pB1 on the accuracy of the service time when using the quorum approximations derived in this chapter.

Figure 6.8 shows the absolute value of the percentage error in the service time of eB1 shown in Figure 6.7 when geometric requests are used for the forked threads, and when the scheduling discipline for processor pB1 that is running task tB1 is processor sharing. The results show that the maximum absolute value of the percentage error is less than 3%. In this model, the Local Service assumption is violated because the scheduling discipline for processor pB1 is processor sharing rather than being an infinite server. However, the



(a) Deterministic requests



(b) Geometric requests

Figure 6.6: Error in the service time of eB1 in the one-layer LQN model in Figure 6.5. The processor for tB1 is an infinite server.

queueing delay on processor pB1 is insignificant since the execution demands of the activities a1,$\cdots$,a5 are relatively small compared to the execution demands on the processors of the remote or external services.

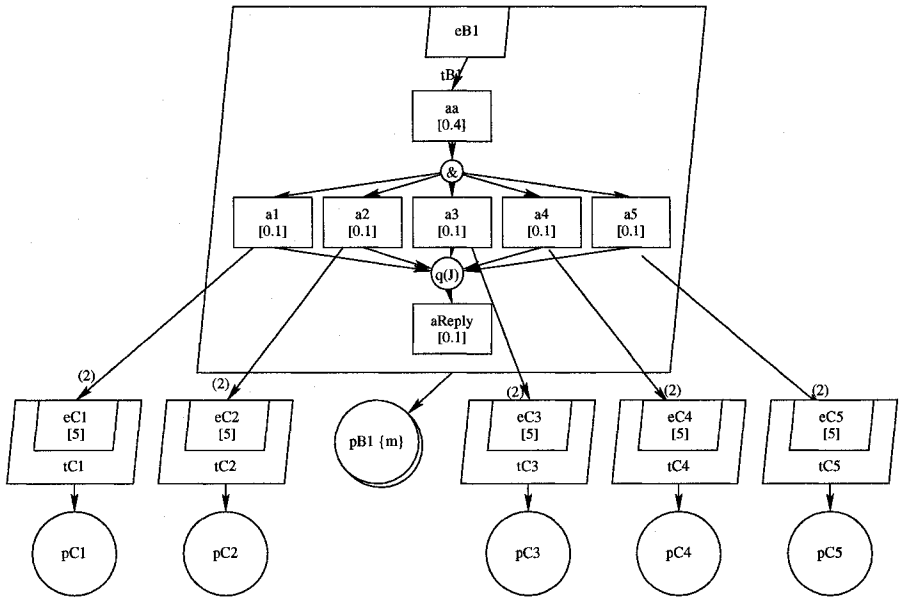


Figure 6.7: One-layer model with multiplicity m for pB1, and quorum of J.

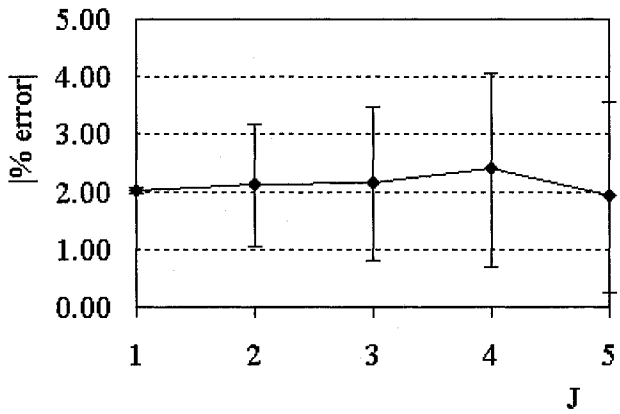


Figure 6.8: Error in the service time of eB1 in Figure 6.7 with geometric requests. The scheduling discipline for pB1 is PS.

Figure 6.9 shows the quorum approximation accuracy when the multiple processors of pB1 that share one queue are used. This is done by varying the multiplicity m of processor

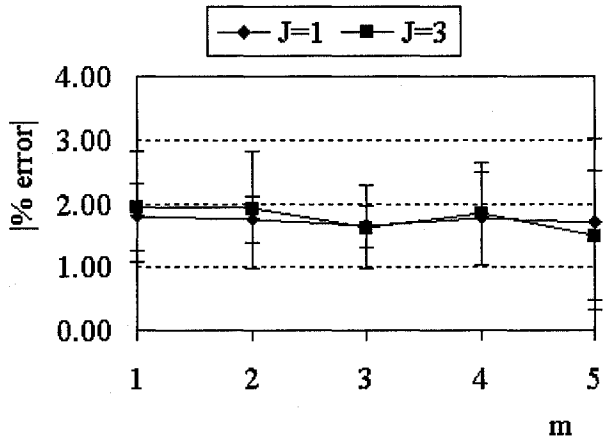


Figure 6.9: Effect of the multiplicity m of pB1 on the error in the service time of eB1 in Figure 6.7 with geometric requests. The scheduling discipline for pB1 is PS.

pB1 in Figure 6.7. The scheduling discipline for processor pB1 is processor sharing. Figure 6.9 shows that the absolute value of the percentage error in the service time is less than 2.5% for both cases when $J = 1$ and when $J = 3$.

Figure 6.10 shows an LQN model for a Triple Modular Redundancy (TMR) system. In this LQN model, the local execution demands for the forked threads are 0.01 time units. There are three forked threads, and each thread makes a number of requests with mean $y = 3$ to its remote server tCi for $i = 1, \cdots, 3$.

The absolute value of the percentage error in the service time of entry eB1 in Figure 6.10 is shown in Figure 6.11. All requests are associated with a RVs that are geometrically distributed. Values of the coefficient of variation (CV) of entries eCi's $i = 1, \cdots, 3$ are varied. When $CV > 1$, the LQN model violates the Remote Service assumption, because the service times for the remote servers are non-exponentially distributed. In addition, the Local Service is violated since the scheduling discipline for processor pB1 is processor sharing. The results show that increasing the coefficient of variation increases the absolute

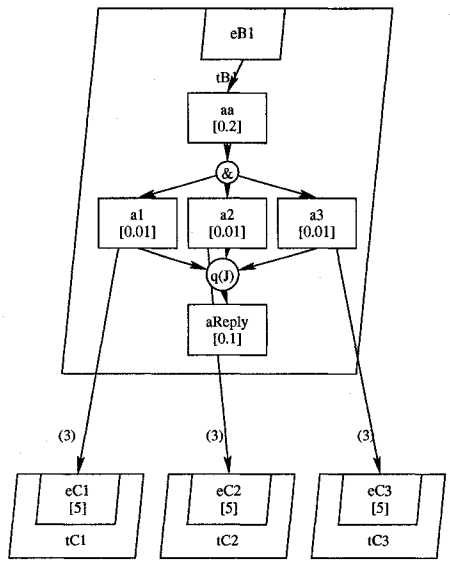


Figure 6.10: An LQN model for a Triple Modular Redundancy system.

value of the percentage error in the service time of entry **eB1**. For the case when $CV = 1.5$, the absolute value of the percentage error increases with almost the same percentage for different values of J, compared to the case when $CV = 1$. This is because the closed-form formula for geometric requests assumes that the remote servers have exponentially distributed service times (even in the case when their coefficients of variations are set to be greater than one in the input model).

To have more insight on the effect of the coefficient of variation of the remote servers on the accuracy of the service time of the client entry **eB1**, Figure 6.12 shows the results of solving the model in Figure 6.10 when $J = 2$, and when geometric requests are used. The scheduling discipline of processor **pB1** is processor sharing. The absolute value of the percentage error increases monotonically as the coefficient of variation increases.

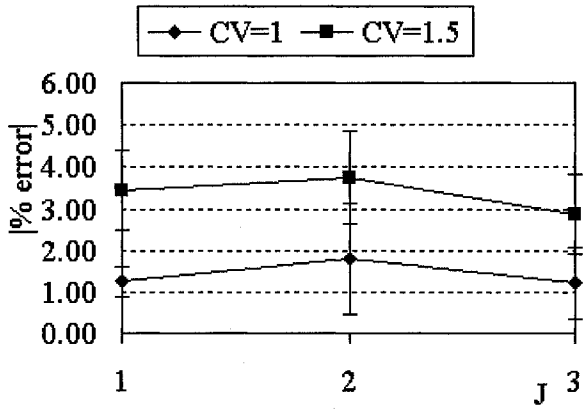


Figure 6.11: Error in the service time of eB1 in Figure 6.10 with geometric requests for different values for the coefficient of variations (CV) of entries eCi's $i = 1, \cdots, 3$. The scheduling discipline for pB1 is PS.

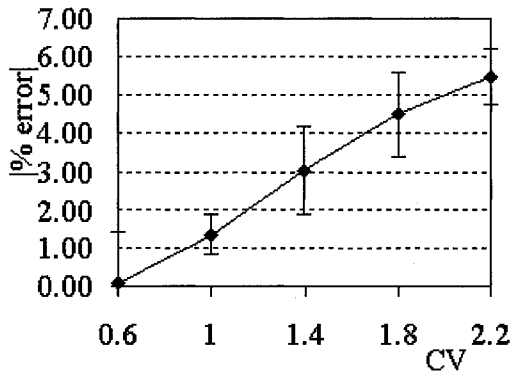


Figure 6.12: Effect of coefficient of variation (CV) of entries eCi's $i = 1, \cdots, 3$ on the error in the service time of eB1 in Figure 6.10 when $J = 2$ and geometric requests are used. pB1 is scheduled using PS.

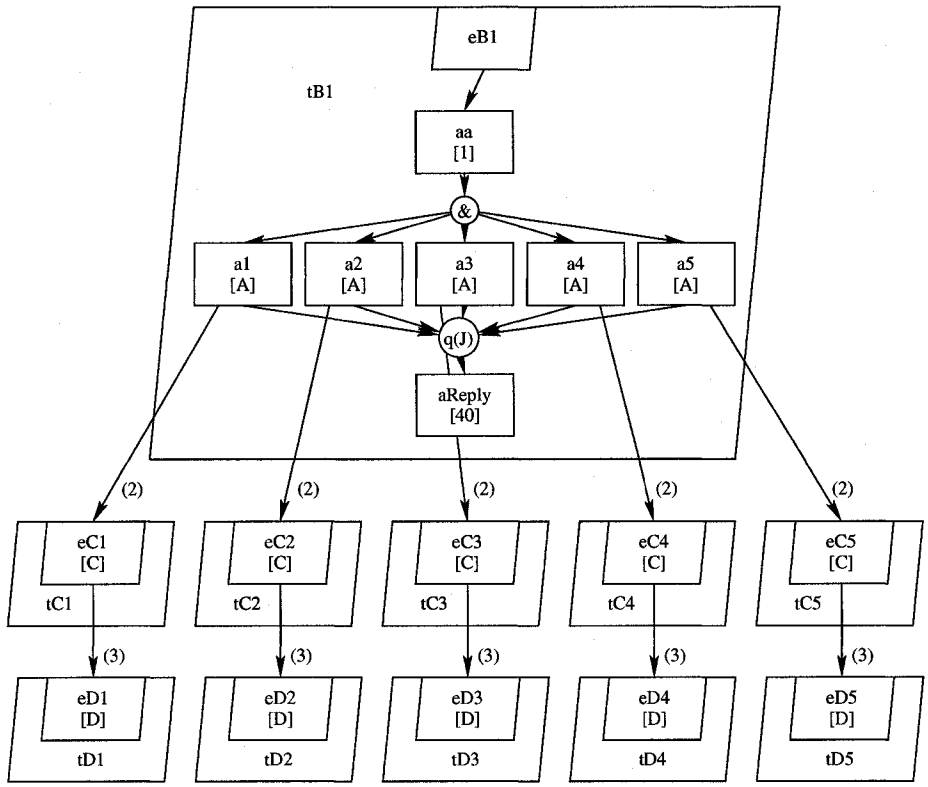


Figure 6.13: Two-layer LQN model with five parallel thread.

6.5.2. Two-Layer Models.

In this section, the constraints of the closed-form formula for geometrically distributed requests derived in Chapter 5 are violated. This is done by using an LQN model with two-layers of servers and using different scheduling disciplines for the processor running the forked threads. The model shown in Figure 6.13 relaxes the two assumptions:

(1) Remote Service: the requests to the second layer of servers, labeled as tD1 through tD5, change the service time distribution of the entries called by the client task.

(2) Local Service: using the scheduling disciplines first-in, first-out (FIFO) and processor sharing (PS) for the processor of the client task tB1 causes queueing at the processor pB1.

Task tB1 is running on an infinite server.

Figure 6.14 shows the absolute value of the percentage error in the LQNS results compared to the simulation results. The percentage error is for the service time of entry eB1 for different values of A, C, D, and J shown in Figure 6.13. Despite the relaxation of the exponential service time constraint, the error for the model with deterministically distributed requests is less than 3.5%, and for geometrically distributed requests is less than 6%.

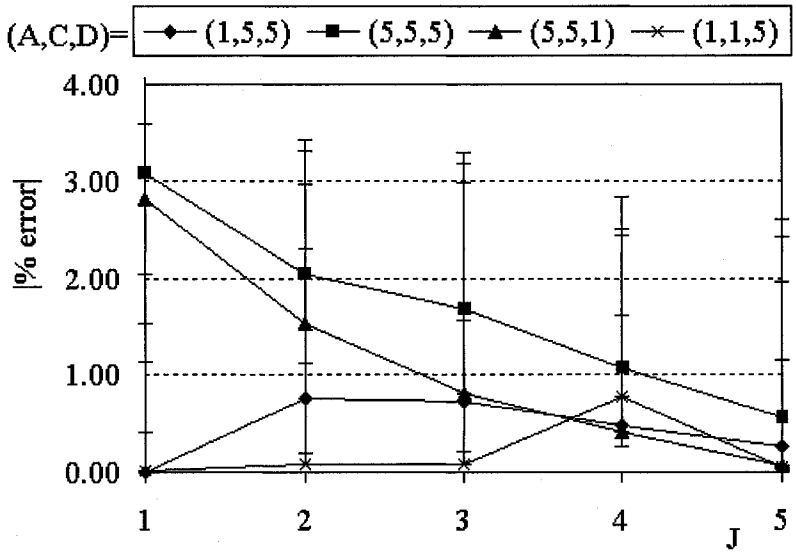
Figure 6.15 shows the effect of the value of the execution demand of activity aReply in 6.13 on the accuracy of the service time of entry eB1. The processor pB1 is an infinite server. As expected, the increase in the execution demand of activity aReply decreases the absolute value of the percentage error, because an increase in the value of the execution demand of aReply will dominate the computation of the absolute value of the percentage error.

Task tB1 is running on a FIFO server.

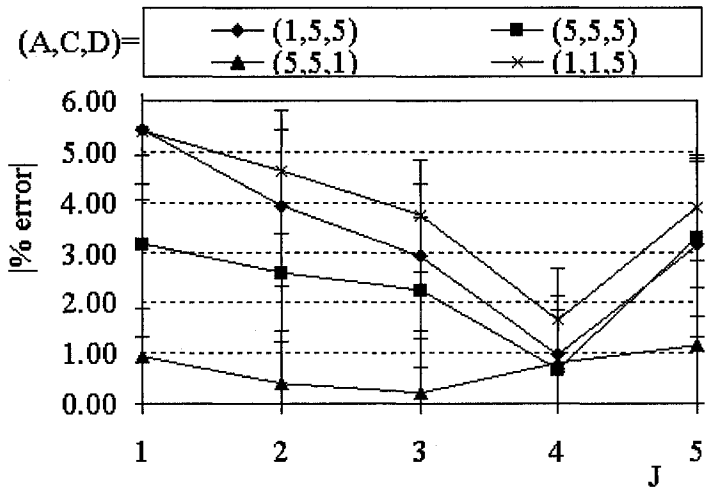
Figure 6.16 shows the results when using a single server with FIFO scheduling for processor pB1. The maximum error for the model with deterministic requests is about 19%, and for the model with requests distributed geometrically is 24%, both occurring with $J = 1$.

Task tB1 is running on a PS server.

Figure 6.17 shows the results when the client's processor is scheduled using the processor sharing discipline. Both request distribution types have a maximum error of about 12%, with the largest errors occurring with $J = 1$.



(a) Deterministic requests



(b) Geometric requests

Figure 6.14: Error in the service time of eB1 for the two-layer model shown in Figure 6.13. The processor for tB1 is an infinite server.

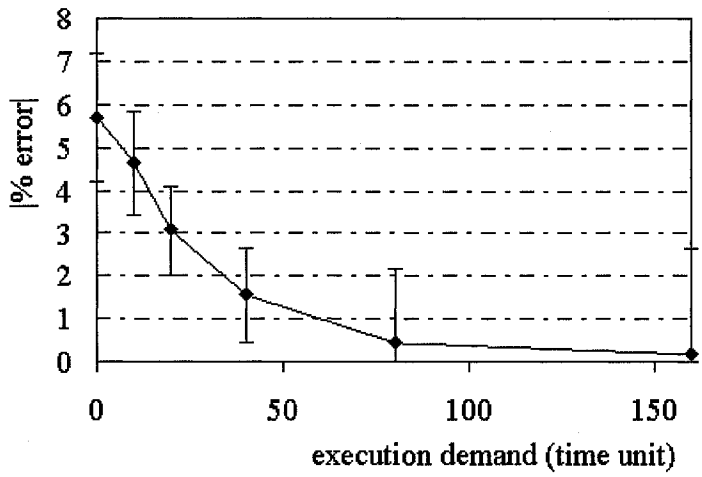
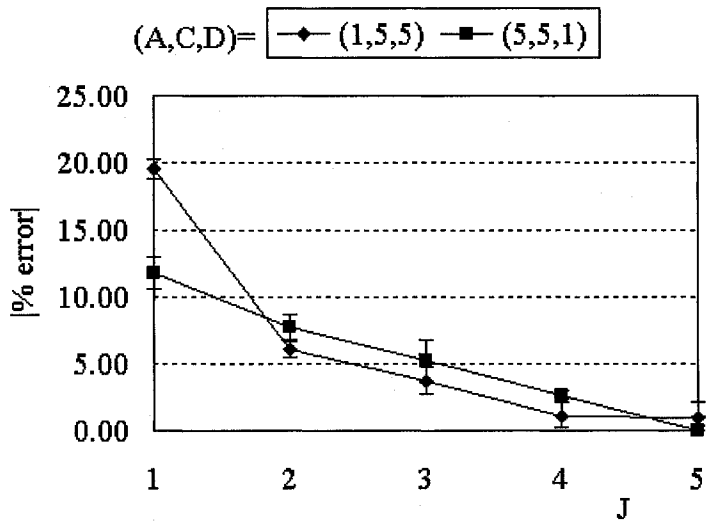
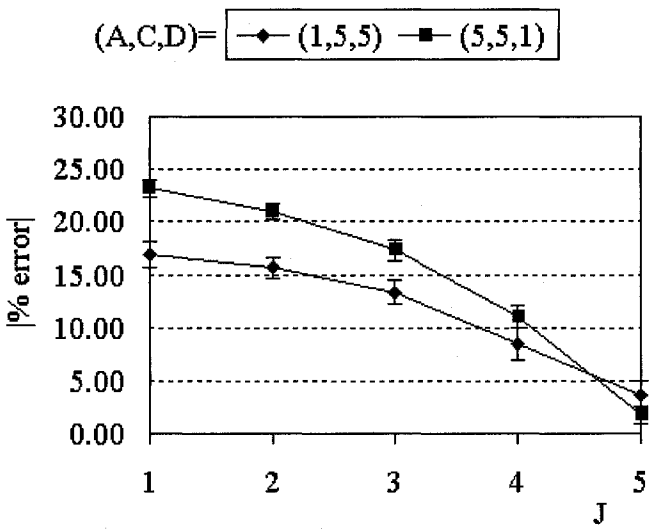


Figure 6.15: Effect of the execution demand of aReply in Figure 6.13 on the accuracy of the service time of eB1. pB1 is an infinite server. $J = 4$.

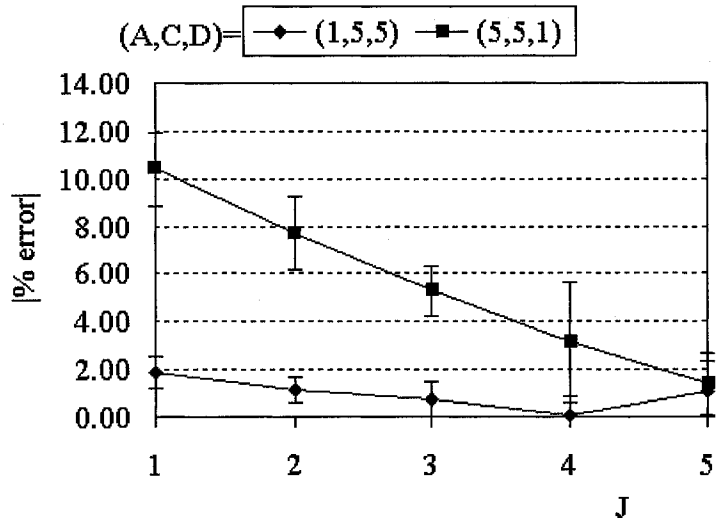


(a) Deterministic requests

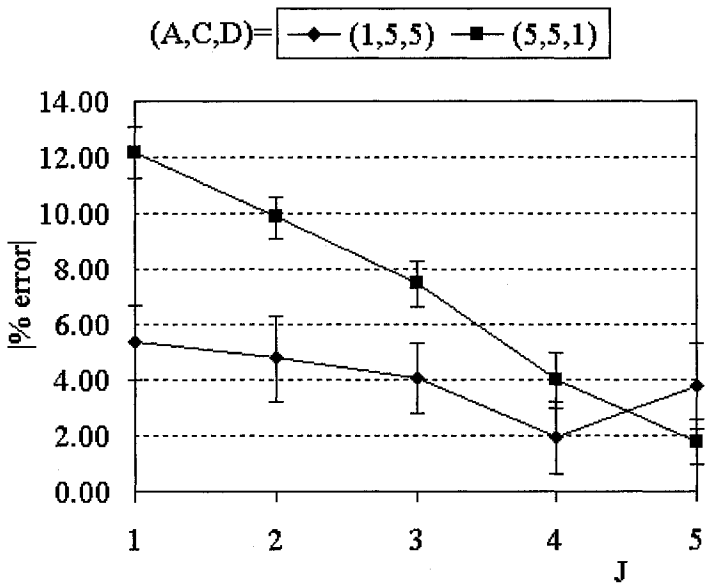


(b) Geometric requests

Figure 6.16: Error in the service time of eB1 for the two-layer model in Figure 6.13. The processor for tB1 is scheduled using FIFO.



(a) Deterministic requests



(b) Geometric requests

Figure 6.17: Error in the service time of eB1 in the two-layer model in Figure 6.13. The processor for tB1 is scheduled using PS.

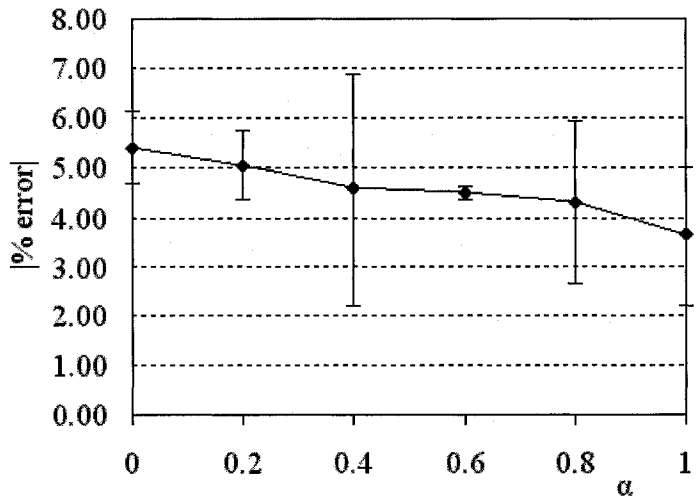


Figure 6.18: Effect of heterogeneity on the error in the service time of **eB1** in the two-layer model in Figure 6.13 with geometric requests. The processor for **tB1** is an infinite server, $y2 = 3, A = 1, D = 5, Ci = 1 + (i-1) \times \alpha, y1 = 2 \times (1+\alpha), J = 1$.

Heterogeneous threads (and infinite processor for tB1).

Figure 6.18 shows the effect of having heterogeneous threads on the service time of entry eB1. The threads in Figure 6.13 are labeled with i with values from $1, \cdots, 5$ (from left to right), and the mean number of requests from the client to the first layer of servers is given by the parameter $y1$. The parameters C_i and $y1$ are functions of α, where α is a real number that is a measure of the heterogeneity level, as follows: $Ci = 1 + \alpha(i-1), y1 = 2(1+\alpha)$. When $\alpha = 0$, all threads are homogeneous. The result shows that the highest error occurs when the forked threads are homogeneous, then it decreases slightly as the heterogeneity increases.

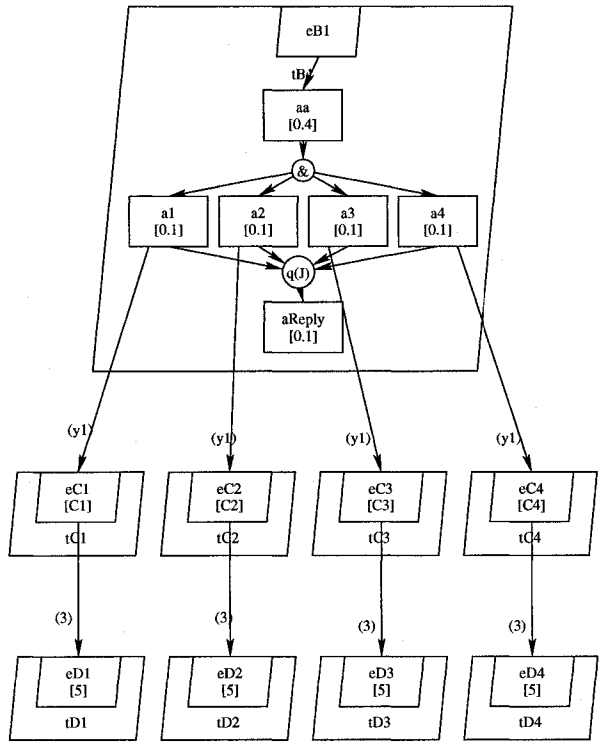


Figure 6.19: Two-layer LQN model with four parallel threads.

Heterogeneous threads (and processor sharing for pB1).

Figure 6.19 shows an LQN model with two layers of servers. The model has four forked threads, each with an execution demand of 0.1 time units on processor pB1 of task tB1. Figure 6.20 shows the accuracy in the service time approximation when using geometric requests in the LQN model in Figure 6.19. The scheduling discipline for processor pB1 is processor sharing. The execution demand for servers tCi for $i = 1, \cdots, 4$ is calculated using the formula $Ci = 5 + (i - 1) \times \alpha$. The mean number of requests $y1$ to servers tCi is calculated using $y1 = 3 \times (1 + \alpha)$.

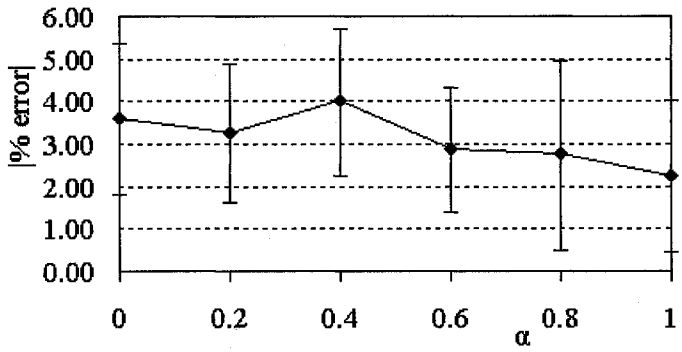
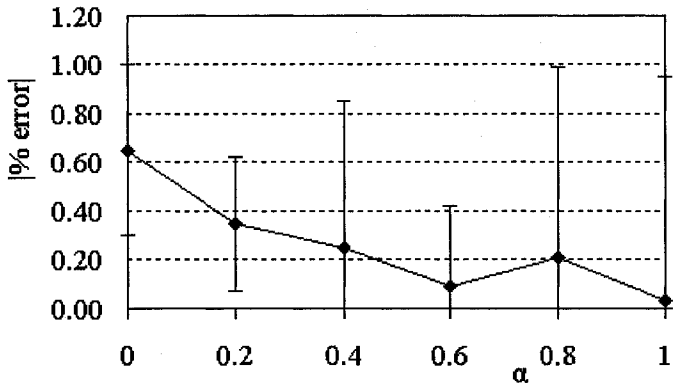


Figure 6.20: Error in the service time of **eB1** in the two-layer model in Figure 6.19 with geometric requests. The scheduling discipline for **pB1** is PS, $Ci = 5 + (i-1) \times \alpha, y1 = 3 \times (1+\alpha)$, $J = 2$.

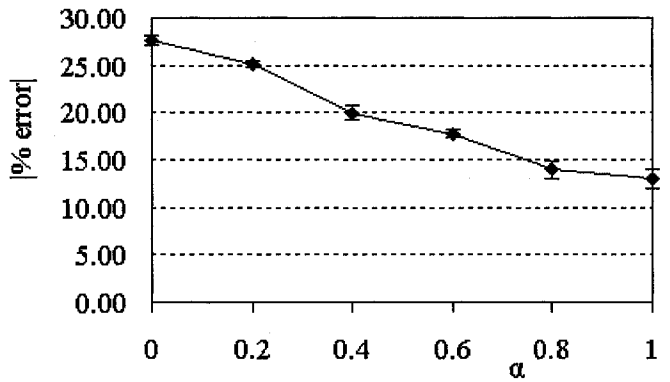
In this model, the main thread of control resumes execution when $J = 2$ out of four forked threads respond. The results show that increasing the heterogeneity level decreases the absolute value of the percentage error.

When deterministic requests are used for the requests to servers **tCi** for $i = 1, \cdots, 4$, the results are shown in Figure 6.21. Figure 6.21(a) shows the absolute value of the percentage error when the gamma distribution fitting is used to approximate the distribution of a thread delay. Figure 6.21(b) shows the results when the closed-form formula for deterministic requests fitting is used to approximate the distribution of the thread delay. The results are evaluated for the case when $J = 3$ responses are required for the main thread of control to resume execution. The number of requests $y1$ to remote servers **tCi** is calculated as follows: $y1 = \lceil 3 \times (1+\alpha) \rceil$, where α is a measure of the heterogeneity of the forked threads.

The results show that when the gamma distribution fitting is used, the error in the service time is less than 1%, while it can be up to about 28% when the closed-form formula for deterministic requests fitting is used. Note that the closed-form formula for determinsitic requests assumes that the service time of the remote servers are exponentially distributed,



(a) Gamma fitting



(b) Closed-form-deterministic fitting

Figure 6.21: Error in the service time of eB1 in the two-layer model in Figure 6.19 with deterministic requests. The scheduling discipline for pB1 is PS, $Ci = 5 + 10 \times (i - 1) \times \alpha, y1 = \lceil 3 \times (1 + \alpha) \rceil, J = 3$.

which is not the case in this model, while the gamma distribution takes the variance of the remote servers into account. The results show that whether the gamma distribution or the closed-form formula for determinsitic requests is used, the absolute value of the percentage error decreases as the heterogeneity level increases among the forked threads.

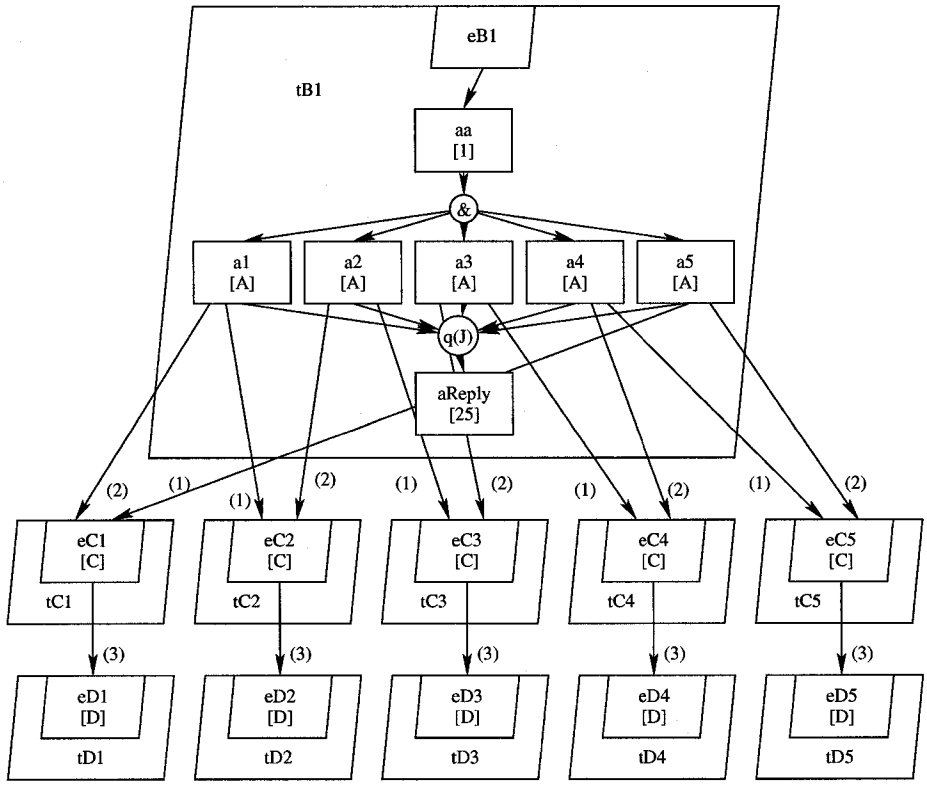
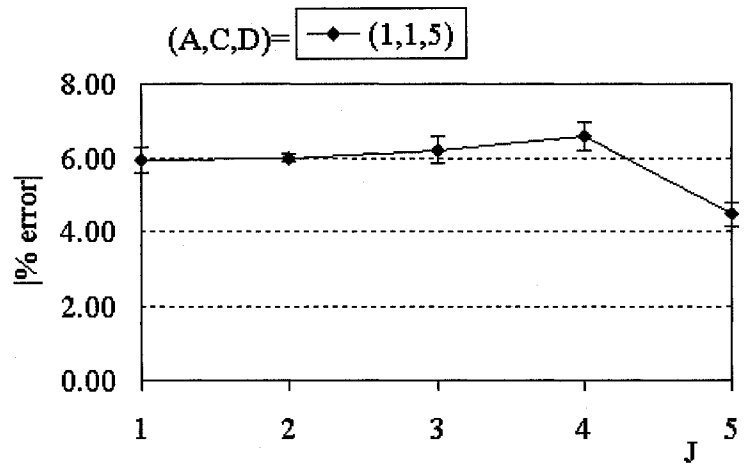


Figure 6.22: Two-layer two-request per thread LQN model.

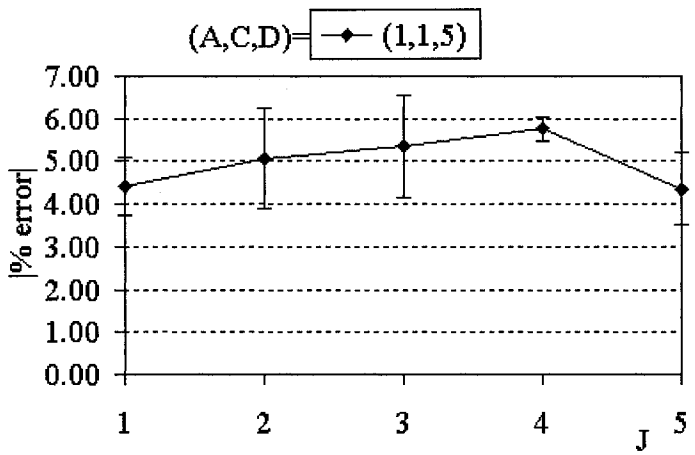
6.5.3. Two-Layer Models with Multiple Requests.

In this section, the exponential distribution part of the Remote Service assumption is violated further by adding another layer of service with three requests, and by having each forked thread make two different requests to two remote servers. This model, shown in Figure 6.22, has contention delay at the tasks labeled aC1 through aC5.

Figure 6.23 shows the results when comparing the LQNS solution to simulation. The processor for task tB1 is an infinite server. For both cases of geometrically and deterministically distributed requests, the error in the solution is less than 7%. Note that the error when $J = 5$ is decreased in this model. When $J = 5$, there is no overhanging threads, which changes the amount of approximation compared to the cases when $J = 1, \cdots, 4$.



(a) Deterministic requests



(b) Geometric requests

Figure 6.23: Error in the service time of eB1 in the two-layer two-request per thread model with geometric requests in Figure 6.22. pB1 is an infinite server.

The approximation of the delays of the overhanging threads makes the trend in the error different for the case when $J = 5$.

If the execution of activity aReply is 100 time units, then the results for the case when geometric requests are used are shown in Figure 6.24. As expected, the absolute value

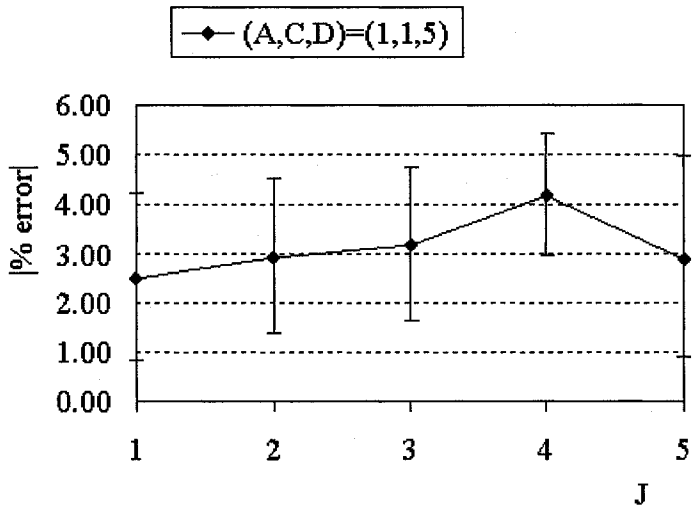


Figure 6.24: Error in the service time of eB1 for the two-layer two-request per thread with geometric requests model shown in Figure 6.22 with execution demand for aReply=100. The processor for tB1 is an infinite server.

of the percentage error is lower when using a higher value for the execution demand of activity aReply. When the requests in the LQN model in Figure 6.22 are deterministic, the results for the absolute value of the percentage error in the LQNS for the service time of entry eB1 are shown in Figure 6.25. The execution demand for aReply is 100 time units, and processor pB1 is an infinite server. The results show that the maximum absolute value of the percentage error is less than 3.5%.

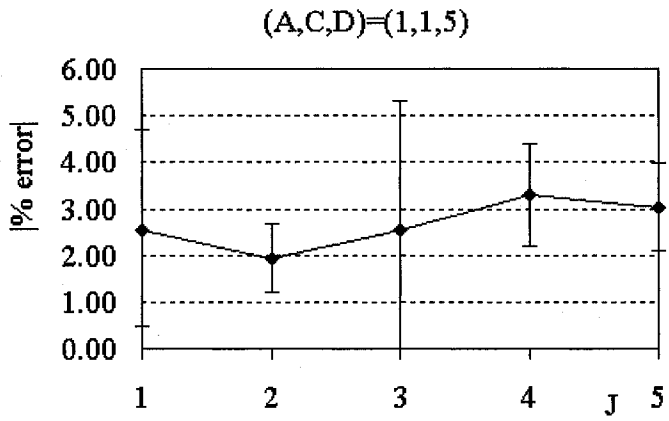


Figure 6.25: Error in the service time of eB1 for the two-layer two-request per thread with deterministic requests model shown in Figure 6.22 with execution demand for aReply=100. The processor for tB1 is an infinite server.

Table 6.1: Results for the mean value of the service time of eB1 in Figure 6.5 when ignoring the overhanging threads. The scheduling discipline for pB1 is PS. $A = 1, C = 5, y = 2$.

J	Simulation	LQNS with	LQNS without
	95% conf. interval	overhang	overhang
1	34.24 ± 0.40	32.56	11.77
2	34.30 ± 0.65	32.93	13.87
3	35.09 ± 0.96	33.67	18.17
4	36.16 ± 0.93	35.70	25.75
5	42.02 ± 0.98	40.91	40.91

6.5.4. Impact of Ignoring the overhanging Threads.

Ignoring the overhanging threads has a significant impact on the performance measures. Table 6.1 shows the results when the overhanging threads in the LQN model of Figure 6.5 are ignored. The maximum percentage error is 66% .

6.5.5. Scalability of Analytic Solution Time.

The scalability of analytic solutions versus simulation is illustrated in Figure 6.27 which shows run-times (in seconds) for solving the replicated database model shown in Figure 6.26. The application system with its two databases was replicated r times, with proportionately more customers, and customer requests split equally among the applications. The simulation run-time was adjusted to be sufficient to give a 95% confidence interval of $\pm 5\%$. The simulation is one to two orders of magnitude slower. At larger numbers of replicas, LQNS runs out of memory due to inefficient coding for large numbers of chains (one per thread). The LQNS algorithm has a space complexity of chains times entries. The present algorithm should be linear in replicas.

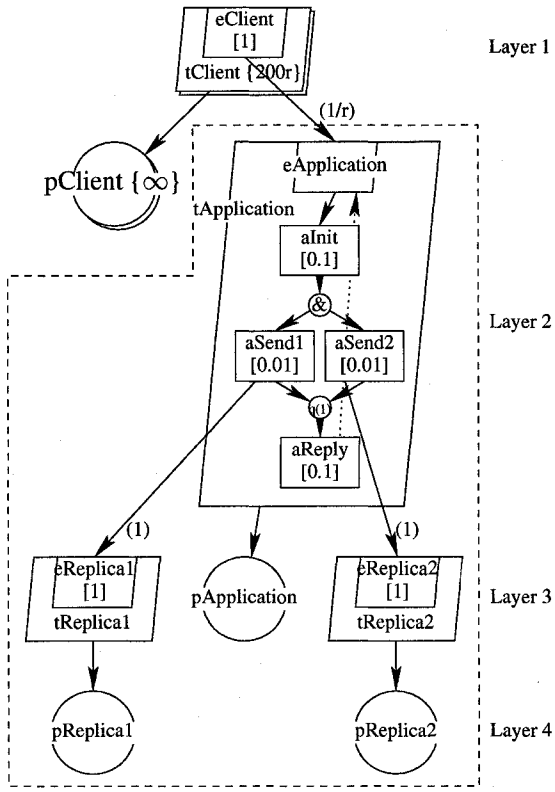


Figure 6.26: An LQN model for a typical database application. r is the number of replicas.

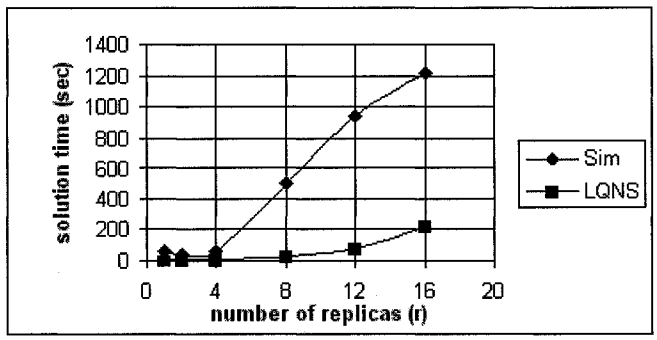


Figure 6.27: Run time for simulation and LQNS analytic solutions.

6.6. Summary

This chapter has described the first analytic model for quorum consensus that includes queueing analysis of the layered contention that arises in such systems. The novel part of the computation is that it accounts for the effect of threads that continue after the quorum is reached, and the integration of the computation into the LQNS.

The approximation was evaluated by many tests covering parameter variations with overall adequate accuracy in the vast majority of tests. Further, the test parameters were chosen to stress the algorithm. For instance rather heavy loads were put on the main processor that executes the quorum section, and this tends to reduce accuracy.

The approximation is best for geometric requests, and when all the assumptions of the approximation of the distributions are met. Deterministic requests (especially when using the closed-form formula for deterministic requests with non-exponential service time for the servers) give poor accuracy, because of a general weakness in the approximation for deterministic requests, as it overestimates the variance of the thread delay which will often result in a the moments of the quorum delay.

Mostly the accuracy is better for larger J and worst for $J = 1$ (the exceptions are for very small errors, which do not matter much). When only the Remote Service assumption is violated the accuracy is still quite good. However when the Local Service assumption is violated the accuracy is degraded (especially when the local activity execution demands are fairly high compared to the execution demand of the remote servers). In general, increasing the heterogeneity of the forked threads decreases the absolute value of the percentage error in the service time of the client's entry.

Overall the accuracy demonstrated here is adequate for most purposes involving early modeling of systems.

CHAPTER 7

Replicated Servers and Parallel Behavior

Capacity planning for large computer systems may require very large performance models, which are difficult or slow to solve. Layered queueing models solved by Mean Value Analysis can be scaled to dozens of servers and hundreds of service classes, with large class populations, but this may not be enough. A common feature of planning models for large systems is structural repetition expressed through replicated subsystems, which can provide both scalability and reliability, and this replication can be exploited to scale the solution technique. A model has been described **[116]** for symmetrically replicated layered servers, and their integration into the system, with a mean-value solution approximation. However, parallelism is often combined with replication; high availability systems use parallel data-update operations on redundant replicas, to enhance reliability, and grid systems use parallel computations for scalability. This work extends the replicated layered server model to systems with parallel execution paths. Different servers may be replicated to different degrees, with different relationships among them. The solution time is insensitive to the number of replicas of each replicated server, so systems with thousands or even millions of servers can be modelled efficiently. Much of the content in this chapter has been published in **[115]**.

7.1. Introduction

Large computer systems often use server replication to provide capacity, reliability or a combination of the two **[92, 149]**. To plan and manage these, it is useful to predict

properties such as capacity and delay with performance models, such as layered queueing models. Even the most efficient computational techniques eventually run into limitations on the number of different servers and service classes they can model, and in general there is a need for ways to extend the scope of analytic modeling techniques. One well-known approach to simplify a model state space or solution complexity is to discover and exploit symmetry [**139, 162, 26, 145**]. So, this work considers application-level symmetry in server systems. Large models often have a structure based on replication of servers and subsystems, either because replication is a vital feature of the system, or as a simplifying assumption in making the model.

An approach, for systems with patterns of interaction among the replicated servers, is described in [**116**]. Any replicated server can interact with any other server, and with different numbers of different replica groups. As in [**116**], this chapter considers layered server systems with groups of replicated elements or subsystems which partition or share a workload. It extends [**116**] with servers that may fork internal parallel threads that interact with other servers. The central idea is that each group of replicated entities is represented only once, and its properties are computed once. The results are then used for all the members of the group. This is an approximation, because jobs processed by some systems do distinguish between instances of replicated components [**153**]. However, the approximation is justified by the greater simplicity in determining and expressing the model, as well as expediting the solution time. The approach shows that the time and space complexity of the solution is insensitive to the number of replicas of any server, and to the number of groups of replicas.

7.2. Layered Queueing with Replicated Servers

Replication is used to add resources to a system, which are modelled by tasks and processors in the LQN formalism. When an entity is replicated, it is replaced by a replication group which is a set of entities with the same properties. In symmetrical replication, the interactions of all the replicas in the group are also similar; that is, they have the same number of clients with the same properties and servers, which are all similar. In effect "similar" entities have numeric parameters with the same value, and interactions with corresponding entities which are themselves "similar".

7.2.1. Notation.

The notation for defining replication in LQN models is illustrated by the example in Figure 7.1, with no activity detail or parallel execution. Beginning from Figure 7.1(a), task Client is replicated twice, and task Server is replicated three times, and each Client makes two requests to each Server. Each replica will run on its own processor. The notation for this is shown in Figure 7.1(b), with three new elements:

- a replication count K for each replicated task and processor, in angle brackets, as $< K >$
- a fanout count O for each arc, showing how many separate target tasks there are for each source replica
- a fanin count I for each arc, showing how many separate source tasks there are for each target replica.

All of these elements have default values of one and are not shown if they equal one. The meaning of the replication notation is shown in the expanded model in Figure 7.1(c), where each replica is shown separately, with a replica number appended to the name, as in Client_1. It is seen that each Client makes requests to 3 Server replicas, expressing the

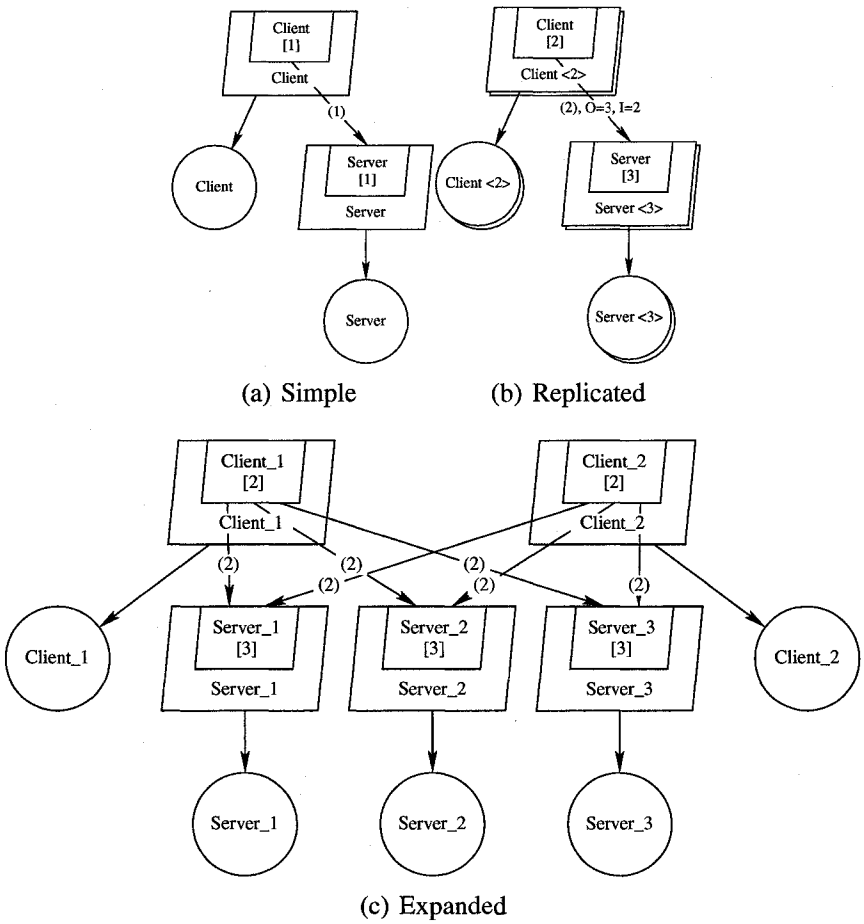


Figure 7.1: Replication of a simple client-server model.

fanout of 3, and each Server receives requests from 2 Client replicas, expressing the fanin of 2. Because the number of processor replicas matches the number of task replicas, each replica task has its own processor. Note that for an original request from a task with K_C replicas to a task with K_S replicas, the total request arcs = $K_C \times O = K_S \times I$, as well as $O \leq K_S$. In the present work all requests between activities and entries of the same pair of original tasks must have the same fanout and fanin.

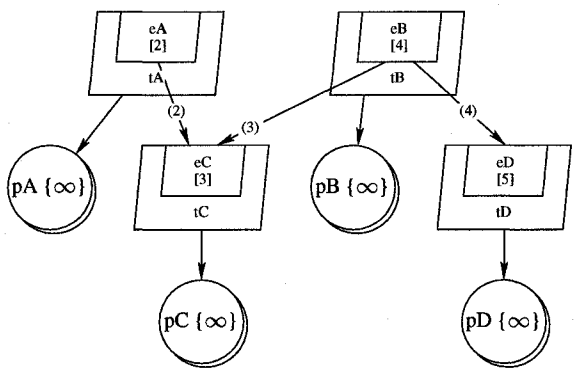
A more elaborate case is illustrated in Figure 7.2, with two groups of clients and two groups of servers. tA is replicated twice, tB twice, tC three times and tD twice, with each

client distributing its requests across all replicas of each server. Figure 7.2(c) shows how the replicas interact. The same architecture, with parallel execution within task tB, is shown in Figure 7.3. In Figure 7.3, requests from activity b3 go from one replica of tB to one replica of tD, since the fanout and fanin are both the default value of one.

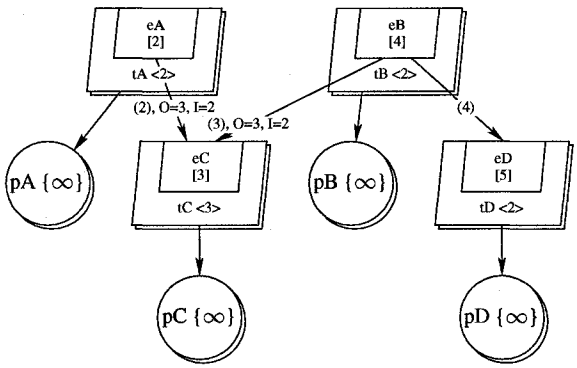
7.2.2. Semantics of Replication.

A model with replicated servers implies an expanded model of the actual system, in which every replica is a separate model entity. To expand a model M_R with replicated servers, into the corresponding expanded model M_E, one does:

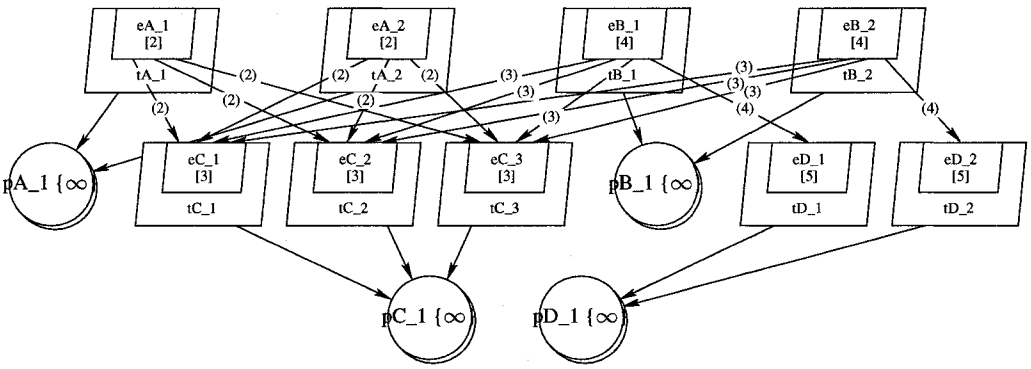
- for each original replicated server task t in M_R, with K_S replicas, create K_S replica tasks with the same entry and activity structure. The tasks, entries and activities are distinct (with separate names) but have the same parameters, and symmetric service requests, as follows.
- for each original request arc from an entry/activity of M_R to an entry e of M_R, with fanout O, create O request arcs of the same type (and the same visit rate parameter) from each corresponding entry/activity of M_E. Each arc has as target one of the entries in M_E that corresponds to entry e, as follows.
- the target entries for the request arcs are chosen as follows: the O target tasks for the arcs from the first source replica are chosen in sequence from the K_T target replicas; for the next source replica the sequence is continued, modulo K_T.
- for each replicated processor in M_R with K_P replicas, create the K_P replica processors. For the replicated tasks allocated to this processor in M_R, allocate the first replica task in M_E to the first replica processor in M_E, and then allocate each succeeding replica task to the next replica processor, modulo K_P. Commonly there are equal numbers of replicas of tasks and processors, and each task is allocated to a separate processor.



(a) Simple

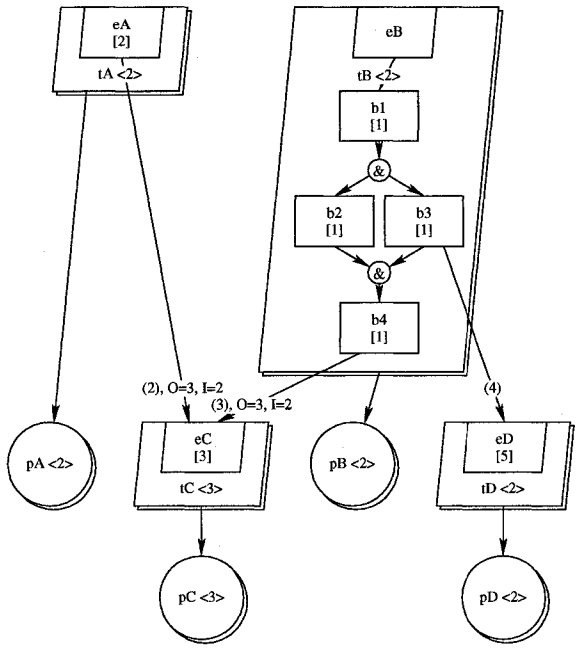


(b) Replicated

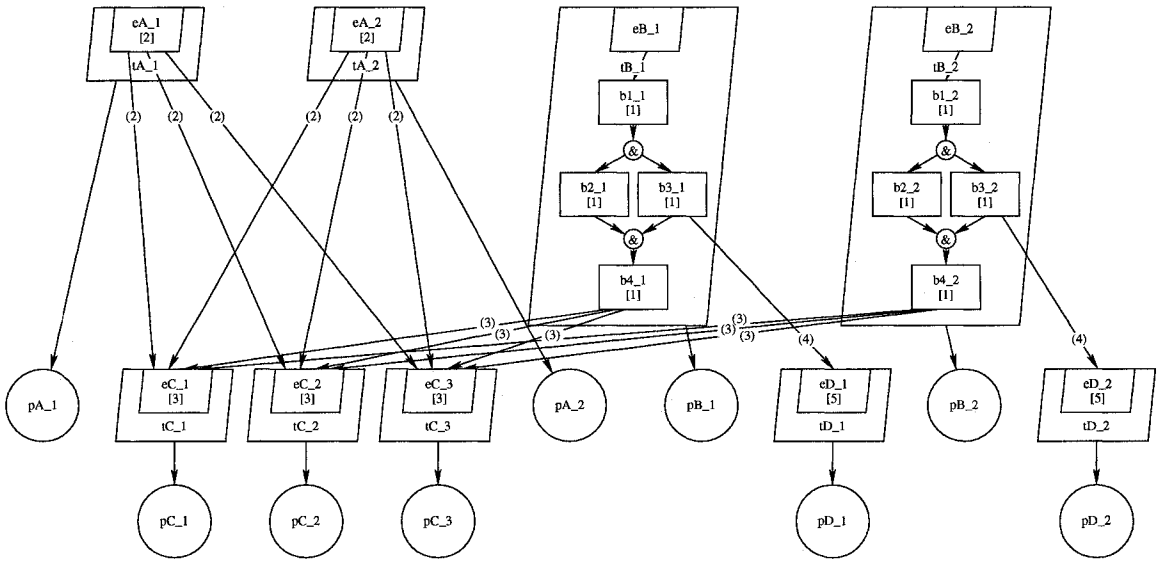


(c) Expanded

Figure 7.2: Replication in a larger simple model.



(a) Replicated



(b) Expanded

Figure 7.3: Replication in a model with parallelism.

This distributes the requests evenly across the target replicas. It follows that for any two request arcs between the entries of the same requester and target tasks in M_R, the arcs created in M_E from one task replica go to entries of the same target task in M_E. Thus, the interaction relationships are per task. When three or more replicated tasks interact, all with the same replication number and fanout/fanin values, then K subsystems are automatically created with the same interaction pattern within each subsystem.

The expressive power of this replication model is high. It supports any number of replicas of any task, with any layered interactions, subject only to the constraints expressed by $K_C \times O = K_S \times I$ and $O \leq K_S$. In the present work O and I are constrained to be integers; fractional fanout or fanin could however be interpreted as distributing a single request stream among a number of targets.

7.2.3. Some Examples.

The expressive power of this replication model will be explored with a few examples, that will also explain the notation further.

Enterprise Systems: Figure 7.4 shows a financial application with a very large user population served by 10 regional application servers. The application integrates information from two different databases, covering different aspects of the business. The users are initially modelled as belonging to 100 subregions, with a fanin from 10 subregions to one RegionalServer, and a further fanin from 10 RegionalServers to each database.

The solution effort of this model will not be affected by changes in the number of RegionalServer replicas, say from 10 to 100, or in the number of users.

Figure 7.5 modifies this model in two ways. First, database DB1 is replicated for reliability, in such a way that each write interaction is sent by the RegionalServer to update both replicas, but each read interaction goes to only one replica. The reduced number of

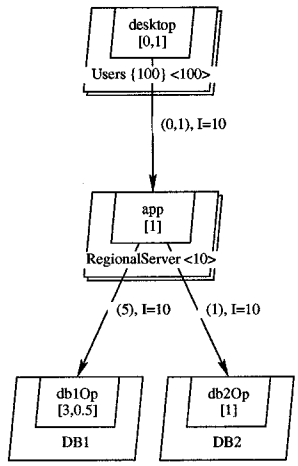


Figure 7.4: An LQN model for a financial application.

reads is represented by showing a mean number of requests of 5 to represent 10 Query interactions, sent to half of the replicated servers (this will give the correct mean access rate at the server). The details of read and write locking are not represented here.

Second, detail is added to the modeling of DB1, by describing the data storage subsystem explicitly with two disks for each node. A disk is described by a combination of a task (for the logical operations) and a processor (for the disk device). Since there are two replicas of DB1, there are four replicas of db1Disk. Similar detail is not added to DB2 in this case. Note that the allocation of requests to replicas puts the requests from DB1_1 (the first replica) on the first two disk replicas, and from DB1_2 on the third and fourth, creating two distinct subsystems for the two database replicas.

Middleware-based Replication: Figure 7.6(a) is a model of a CORBA middleware system from [**124**], without showing the details of entries. Clients access a set of Servers through the ORB, which forwards requests according to the service required, and the currently available Servers. The net tasks represent network delays in the model. The dashed

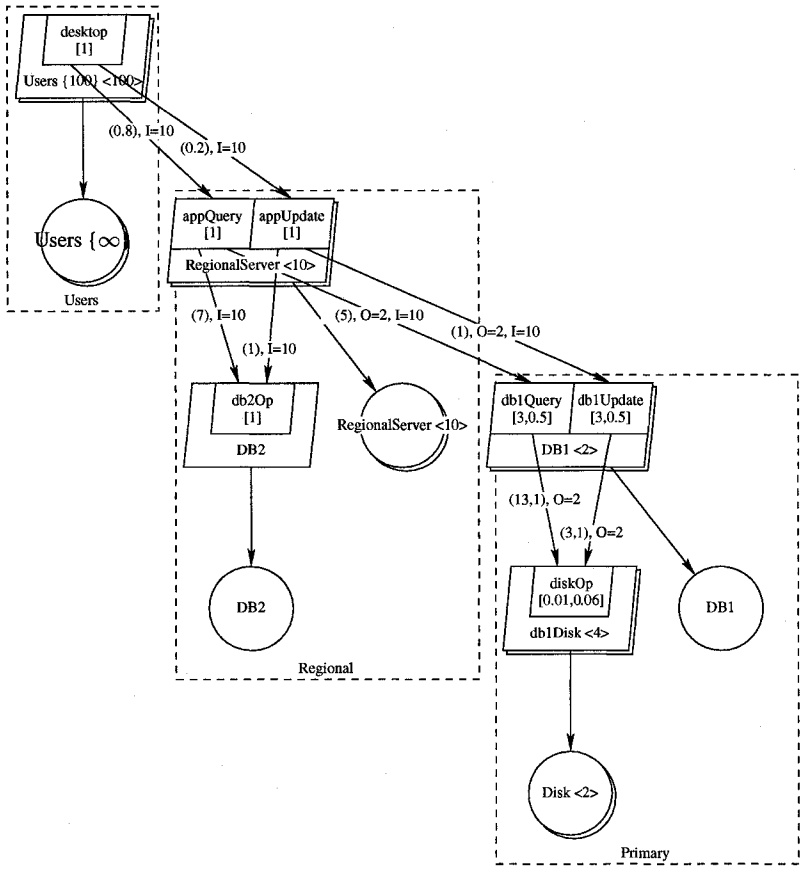
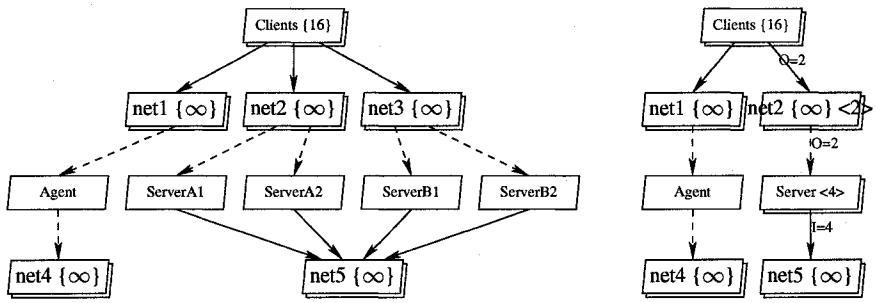


Figure 7.5: A modified LQN model for the financial application in Figure 7.4.

arcs represent forwarding of requests, so that the eventual response is sent to the originator, and the forwarder is unblocked. Figure 7.6(b) shows the same system, but it exploits or assumes symmetry among the operations beginning with net2, and with net3. The representation is naturally more compact and solution effort will be reduced, and will be unaffected in going from 2 paths to, say, 1000 paths.

Distributed High-Assurance System: Figure 7.7 shows a model studied in **[41]**, that describes an en-route air traffic control system with replication for availability and reliability. The analysis used the replication model and algorithms described in this thesis, and added additional logic for reconfiguring the system based on the availability of replicas.



(a) Non-replicated (b) Replicated

Figure 7.6: An LQN model for the H-ORB architecture [**124**].

The AND and OR notations in Figure 7.7 refer to relationships used to compute availability and to reconfigure the system, and will not be explained here. The open-headed request arcs between entries denote asynchronous messages generated periodically from the radar and its front-end processing.

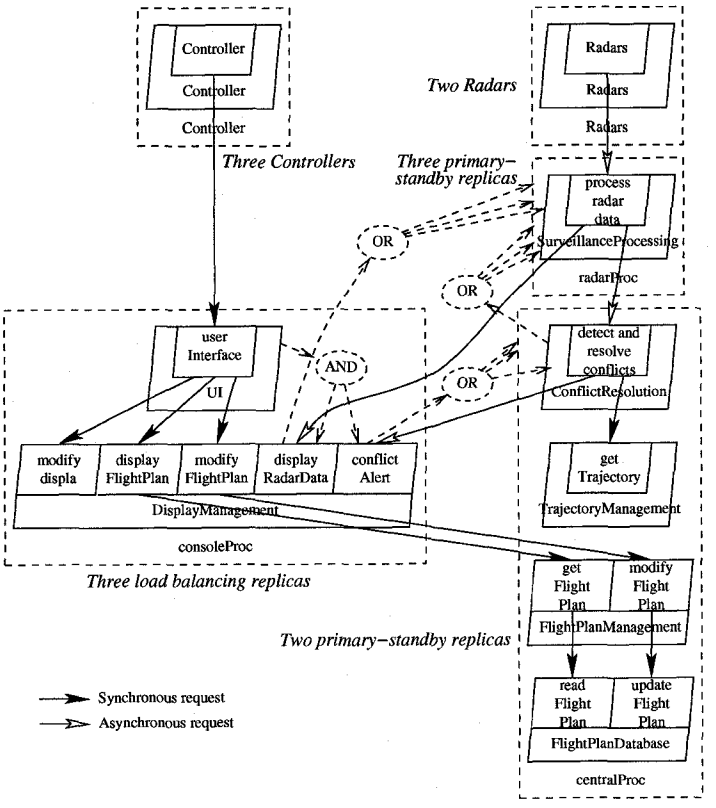


Figure 7.7: Air traffic control system [**41**].

7.3. Layered Solution Strategy

In the LQNS, a submodel is transformed to a queueing network and then the queueing network is solved using MVA. In the LQNS, a chain is generated in the queueing network for each client task of the LQN submodel. The number of customers in each chain corresponds to the number of copies of a client task. A single-threaded higher-layer task results in a customer chain with one customer; a multi-threaded task gives a chain with a customer for each thread. In a non-replicated client task, a chain traverses all the servers visited by the client. Thus, only one chain may traverse each client (or delay server), but a server (queueing center) may be traversed by several chains. For example, in Figure 7.2(a) the top layer submodel has one chain for task tA and one chain for task tB, with the following parameters:

$$\text{Chain 1}: \quad N_1 = 1, \quad V_{1A} = 1, V_{1C} = 2, V_{1,pA} = 1,$$

$$\text{Chain 2}: \quad N_2 = 1, \quad V_{2B} = 1, V_{2C} = 3, V_{2D} = 4, V_{2,pB} = 1,$$

where tasks tA, tB,$\cdots$ are denoted by A, B,$\cdots$, processors are pA, pB and:

- N_c= the number of customers in chain c
- V_{cy}= the number of visits of chain c to center y in the queueing submodel.

7.3.1. Solution with Replication.

With replication (but no parallelism) the solution algorithm represents each set of replicated servers by just one server, and the LQNS replication algorithm described in **[118, 116]** solves for the contention delays and the service times of that replica. The algorithm constructs a reduced version of any layer submodel that includes servers that represent replicated tasks. It includes only one server for each group of replicas, and a set of source centers and chains for the clients of each server. For each server, one chain is constructed

for each group of replicated client tasks whose instances visit it, with a population defined by the number of potential customers to the server. This is equal to the product of the client task multiplicity and the fanin parameter to the server task (representing replication of the client tasks). Each chain visits just one source center and one server center, but a server may be traversed by many chains. The chains in the submodel for the top layer of the example in Figure 7.2(b) have the parameters:

$$\begin{aligned}
&\text{Chain 1 (for tA visiting tC)}: \quad N'_1 = I_{AC} = 2, \quad V'_{1A} = 1, V'_{1C} = 2,\\
&\text{Chain 2 (for tA visiting pA)}: \quad N'_2 = I_{ApA} = 1, \quad V'_{2A} = 1, V'_{2pA} = 1,\\
&\text{Chain 3 (for tB visiting tC)}: \quad N'_3 = I_{BC} = 2, \quad V'_{3B} = 1, V'_{3C} = 3,\\
&\text{Chain 4 (for tB visiting pB)}: \quad N'_4 = I_{BpB} = 1, \quad V'_{4B} = 1, V'_{4pB} = 1,\\
&\text{Chain 5 (for tB visiting tD)}: \quad N'_5 = I_{BD} = 1, \quad V'_{5B} = 1, V'_{5D} = 4,
\end{aligned}$$

where N'_c is the number of customers in the chain, A, B... denote servers for the LQN tasks, pA and pB denote the processors for tA and tB, and as before V'_{cy} is the number of visits by a client of chain c to server y. Note that the requests modelled by Chain 2 without replication are now modelled by Chain 3, Chain 4, and Chain 5, which have different numbers of customers. Splitting the customer chains, according to the server they visit, is necessary if different fanout values can be applied to different server tasks in the LQN model, since there is one server center in the layer submodel for each server task in the LQN model.

Source Service Time Computation with Replication

The source service time computation of Eq. (2.11) is modified to add the delay due to visits to replicas which are not represented in the submodel. As an example, the modified

source service time S'_{1A} of Chain 1 in Figure 7.2(b) is given as:

$$S'_{1A} = S_{1A} + (O_{AC} - 1)R_{1C} + O_{A,pA}R_{2pA}, \tag{7.1}$$

where:

- S_{1A} = Source service time for Chain 1, calculated according to Eq. (2.11)
- O_{AC} = Fanout from task A to task C
- R_{1C} = Delay to task A (which creates Chain 1) at task $C = Y_{AC}W_{AC}$
- Y_{AC} = Number of visits from task A to task C
- W_{AC} = Delay for one visit from task A at task C (service time plus queueing time)
- $O_{A,pA}$ = Fanout from task A to processor pA
- R_{2pA} = Delay to task A (which creates Chain 2) at processor pA.

In general, the modified source service time S'_c for a chain c created by a client task t for its requests to server task or processor m becomes, in the model with replication:

$$S'_c = S_c + (O_{tm} - 1) \sum_{e \in \mathcal{E}(m)} R_{te} + \sum_{m' \neq m} \sum_{e \in \mathcal{E}(m')} O_{te}R_{te},$$

where:

- S_c = Service time for chain c, calculated by Eq. (2.11)
- O_{tm} = Fanout of client task t to server m in the LQN model
- O_{te} = Fanout of client task t to a particular entry e in the LQN model
- R_{te} = Delay to task t at any entry e, per request to server
- $\mathcal{E}(m)$ = The set of all entries of task m.

This equation makes use of the fact that $R_{te} = 0$ if task t does not make any requests to entry e. The first sum includes entries of task m visited by the chain; the second sum includes

terms for other entries in the same layer submodel and other layers. Since the chain does not visit them in the submodel, their delays are added to the source center service time.

7.3.2. Solving Models with Replication and Parallelism.

The LQNS replication algorithm, described in **[116, 118]**, assumes that the task does not have parallelism or heterogeneous threads. In this thesis, the algorithm has been modified in order to remove these restrictions. A source center and customer chain are constructed for each server and for each thread class that visits it, with the number of customers equal to the product of the client task's multiplicity and the fanin parameter to the server task. Each chain visits just one source center and one server center, but a server may be traversed by many chains.

Consider the model shown in Figure 7.3(a). Its upper-most submodel is shown in Figure 7.8(a), and the underlying queueing model for this submodel is shown in Figure 7.8(b). The chains for the queueing model are constructed as follows:

$$\text{Chain 1 (for thread 1 of tA)}: \quad N'_1 = I_{tA,pA} = 1, \quad V'_{1,tA} = 1, V'_{1,pA} = 1,$$

$$\text{Chain 2 (for thread 1 of tA)}: \quad N'_2 = I_{tA,tC} = 2, \quad V'_{1,tA} = 1, V'_{1,tC} = 2,$$

$$\text{Chain 3 (for thread 1 of tB)}: \quad N'_3 = I_{tB,tC} = 2, \quad V'_{3,tB} = 1, V'_{3,tC} = 3,$$

$$\text{Chain 4 (for thread 1 of tB)}: \quad N'_4 = I_{tB,pB} = 1, \quad V'_{4,tB} = 1, V'_{4,pB} = 1,$$

$$\text{Chain 5 (for thread 2 of tB)}: \quad N'_5 = I_{tB,pB} = 1, \quad V'_{5,tB} = 1, V'_{5,pB} = 1,$$

$$\text{Chain 6 (for thread 3 of tB)}: \quad N'_6 = I_{tB,pB} = 1, \quad V'_{6,tB} = 1, V'_{6,pB} = 1,$$

$$\text{Chain 7 (for thread 3 of tB)}: \quad N'_7 = I_{tB,tD} = 1, \quad V'_{7,tB} = 1, V'_{7,tD} = 4,$$

where $V'_{c,y}$ is the number of visits by a client of chain c to a server y and N'_c is the number of customers in the chain. Task tB has three threads which create five chains. The first thread

executes activities b1 and b4, and creates Chains 3 and 4, the second thread executes activity b2 on processor pB using Chain 5, and the third thread executes activity b3 on processor pB using Chain 6 and makes requests to task tD using Chain 7. Servers tC, pB, and tD each have only one chain that visit the client tB. Server tD has no chain that visit task tA because task tA does not make any request to task tD. This means that a client task can have more than one chain corresponding to each of its threads, but a server can have one and only one chain for each thread that makes requests to it.

7.3.2.1. *Source Center Service Time Computation.*

The source service time computation of Eq. (2.11) is adjusted to consider the behavior of just one thread class per source station, and to add the delay due to visits to replicas that are not represented in the submodel. The delays that would be observed at these centers are added to the source center service time. The modified source center service time S'_2 for Chain 2 (which is created by thread 1 of tA, visiting tC) is given as:

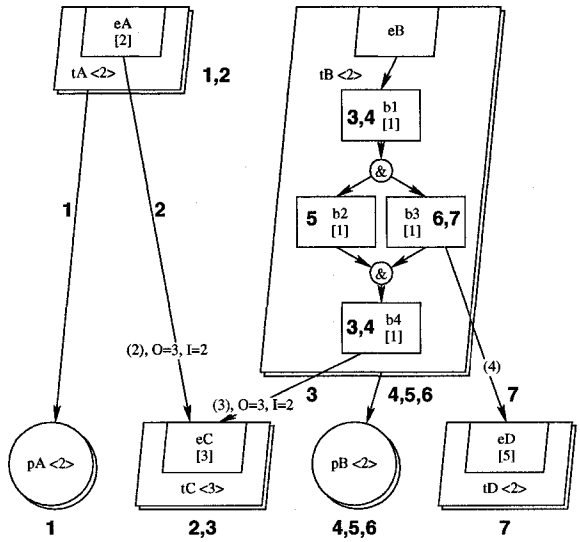
$$S'_2 = S_2 + (O_{\mathrm{tA,tC}} - 1)R_{2,\mathrm{tC}} + R_{1,\mathrm{pA}}, \tag{7.2}$$

where:

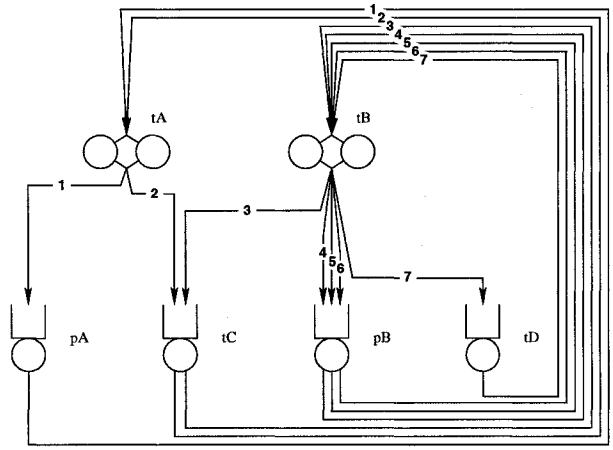
- S_2 = Source service time for Chain 2, calculated by Eq. (2.11)
- $R_{1,\mathrm{pA}}$= Delay imposed on thread 1 of tA by processor pA
- $R_{2,\mathrm{tC}}$= Delay imposed on thread 1 of tA by task tC = $Y_{2,\mathrm{tC}}W_{2,\mathrm{tC}}$
- $Y_{2,\mathrm{tC}}$= Number of visits to task tC by Chain 2
- $W_{2,\mathrm{tC}}$= Delay for a single visit from thread 1 of tA to task tC.

Similarly,

$$S'_3 = S_3 + (O_{\mathrm{tB,tC}} - 1)R_{3,\mathrm{tC}} + O_{\mathrm{tB,pB}}R_{4,\mathrm{pB}}. \tag{7.3}$$



(a) Layer submodel



(b) Queueing network model

Figure 7.8: The top-most submodel and queueing network for the replicated model in Figure 7.3. Objects in (a) are annotated with the chains used in the queueing network in (b).

In addition to the delay for the other replica tC servers, the modified service time of Chain 3 of client tB includes delays to the thread in the LQN model, from other service requests. In this case the requests are to the processor, and they are incurred by Chain 4.

In general, with parallelism the source center service times for a customer chain c created by thread thr of task t making requests to server task m is calculated by:

$$S'_c = S_c + (O_{tm} - 1) \sum_{e \in \mathcal{E}(m)} R_{thr,e} + \sum_{m' \neq m} \sum_{e \in \mathcal{E}(m')} O_{te} R_{thr,e}, \tag{7.4}$$

where:

- S_c = Source center service time for chain c
- O_{tm}= Fanout of client task t to server m in the LQN model
- $R_{thr,e}$= Total delay to activities in the thread which creates chain c, from requests to entry e, per request made to entries of task m
- $\mathcal{E}(m)$= The set of entries of task m.

7.3.3. Implementation.

The algorithm that solves replicated tasks with fork-join thread patterns, shown in Algorithm 7.3.1, has been implemented in the LQNS. After the model is loaded the solver checks if it has replication in the initialization. If it does, then it creates the chains for the replicated clients and servers from the servers' point of view as explained in previous sections. The algorithm sets the number of customers of a chain equal to the fanin of that chain multiplied by a population of an instance of the replicated client. For each LQN submodel, the solver solves the underlying queueing network of the submodel by MVA. If the submodel has any of its clients or servers replicated then it adjusts the clients' service times as explained before. This is because the customer of a specific client's thread might visit more than one chain, so the service time of a chain needs to be modified to account for the delay incurred when other customers visit all other chains visited by that thread.

ALGORITHM 7.3.1. *Pseudo-code for "inner" iteration.*

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 %Layer has no replicated tasks.
      - ▷ Set validity flag to true;
      - ▷ Set iteration count to limit;
      - ▷ Set convergence to false; %Execute loop 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 the LQNS model;
- WHILE (iteration limit not reached)
- Cleanup;
- RETURN validity flag;

END

The solution of a specific submodel that has replication is iterative. The clients' service times are modified, and then the submodel is solved using MVA. Then, the new service times are used to update the clients' service times, and then the model is solved again. This process is repeated until the submodel solution converges. This is called "inner" fixed-point iteration using the multi-variate Newton-Raphson method. When the replicated submodel converges, the solver moves to the next submodel and uses the results of the current submodel as input to the next submodel if needed. This process of iterating between

the submodels is called "outer" fixed-point iteration. This iteration process is repeated until the outer iteration converges or reaches an iteration limit.

7.4. Replicated Servers with Quorum Pattern

The replication technique in the LQN formalism is used to expedite the solution time of the LQNS. This is done by solving the underlying queueing networks (QNs) of an LQN model, only for one replica, then distributing the results to the other replicas accordingly. This section extends the work in Subsection 7.3.2 by extending the modeling power of the LQN formalism to allow tasks that are replicated to have threads with a quorum pattern.

The solution technique of an LQN model with replicated tasks, that have internal parallelism of quorum fork-join, combines the technique of solving replicated tasks with AND fork-join the technique developed in [**113**] and in Chapter 6. In order to apply the two techniques, a thread delay in a replicated client task needs to be divided into two parts: the local part that runs on the host processor, and the remote part that runs on the lower layer (external) server(s). The mean delay, and number of requests to lower layer server(s) are approximated for each part as outlined in Algorithm 7.4.1. Then, the means and number of requests to lower layer servers are used to approximate the variance of both the local and remote parts, based on whether the mean number of requests is deterministic or geometrically distributed, as described in Subsection 6.4.2. The means and variances of the delays of the two parts are used to approximate the distribution functions of the local and remote parts as explained in Subsection 6.4.2. Then, the service time for the entry of the task is calculated as in Eqs. (6.10), and (6.11).

ALGORITHM 7.4.1. *Approximation of Local and Remote parts of a thread in a replicated task with quorum.*

- meanLocalWait=meanRemoteWait=totalNumberRequest=requestWait=0;
- FOR each activity in a thread
  - ✧ FOR each request made by the activity
    - ❑ IF request's destination is host processor
      - ▷ meanLocalWait = meanLocalWait + requestWait;
      - ▷ totalNumLocalRequests= totalNumLocalRequests+ meanNumberRequest
    - ❑ ELSE %External server.
      - ▷ meanRemoteWait =meanRemoteWait + requestWait;
      - ▷ totalNumRemoteRequests = totalNumRemoteRequests+ meanNumberRequest $\times$ fanout;
    - ❑ END IF
  - ✧ END FOR
- END FOR
- RETURN meanLocalWait, meanRemoteWait, totalNumLocalRequests, totalNumRemoteRequests;

Figure 7.9 shows an LQN model with task tB having a replication factor of two and forked threads that join using a quorum of one. The model is solved using the LQNS. The service time of entry eA is 225.63 time units and the variance is 70733.4. The same model is expanded, then solved using the LQNS. The result, in the expanded model, for the service time of entry eA is 224.60 time units and the variance is 44487.3.

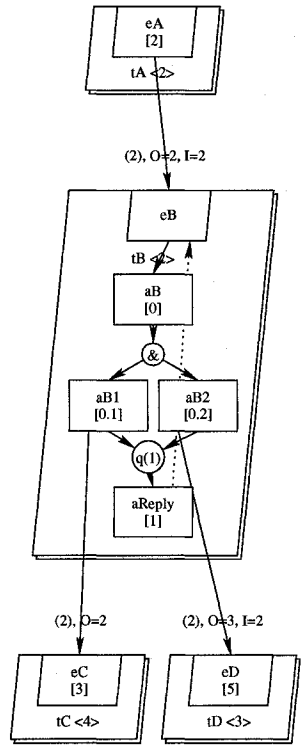


Figure 7.9: An LQN model using replicated task with a quorum.

7.5. From Expanded to Replicated Models

In this section, the steps required to detect replication in an expanded LQN model are outlined, and the conversion of the expanded LQN model to an LQN model with replication notation is described.

Two LQN entries or activities are "symmetric" if they have the same input parameters and they make the same requests to symmetric lower layer servers. In addition, two LQN tasks in a submodel are "symmetric" if:

(1) they have the same input parameters

(2) they have the same activity graph if applicable (servers in a submodel do not have activity graph)

(3) their entries or activities are symmetric

ALGORITHM 7.5.1. *Conversion of an expanded LQN model to a replicated one.*

(1) M_e: expanded model, M_r: replicated model, $K_{c,i}$: replication level for client i; $K_{s,j}$: replication factor for server j.

(2) Initialize $K_{c,i}$, $K_{s,j}$ to zeros.

(3) WHILE (next expanded submodel has not been visited)

  (a) Make a replicated client $R_{c,i}$ for each set i of symmetric clients.

  (b) $K_{c,i}$= number of symmetric clients in set i.

  (c) Make a replicated server $R_{s,j}$ for each set j of symmetric servers of the replicated clients in set i.

  (d) $K_{s,j}$= number of symmetric servers in set j.

  (e) FOR each symmetric entry or activity in $R_{c,i}$

    (i) fanout = number of distinct requests made to servers in set j in M_e.

    (ii) fanin = $K_{c,i}\times$ fanout / $K_{s,j}$.

  (f) END FOR

(4) END WHILE

7.6. Near Symmetry in Expanded Models

The balanced symmetry requirements for the conversion of an expanded LQN model to a replicated one, as outlined in Algorithm 7.5.1, is restrictive. In some expanded models, tasks do not satisfy such requirements, but it is still required to approximately convert the expanded model to take advantage of the replication notation. The concept of near symmetry is used which is based on symmetry but when the participating parties are not exactly the same. In this section, the concepts of behavioral and structural near symmetry are introduced. The concepts can be used by modelers to perform an approximate conversion.

7.6.1. Behavioral Near Symmetry.

Behavioral Near Symmetry (BNS) occurs when the tasks, that have the potential of being replicated, differ in the means or variances of their host demands. In this case, a modeler can approximate the behavior of these tasks by using a replicated task that represents the average behavior with:

$mean_{replicated}$ =average of means of the expanded tasks,

$variance_{replicated}$= average of variances of the expanded tasks.

7.6.2. Structural Near Symmetry.

Structural Near Symmetry (SNS) occurs when clients do not all make the same number of distinct requests to their servers. In this case, the modeler can do one of two things:

(1) Set the mean number of requests to lower layer servers in the replicated task as:
y_r = sum of all mean number of requests made by a near symmetric source (entry or activity in the expanded model) divided by number of clients in the expanded model.

(2) Compute a positive real number for the mean of the fanout or fanin parameters.
fanout = total number of requests made by the expanded tasks divided by number of tasks.
However, the current LQNS allows the fanout and fanin to be positive integers only.

7.7. Results and Analysis

To demonstrate the replicated solver, several models are shown below. The first subsection demonstrates the scalability of the solution technique. The second subsection consists of several examples both in their replicated and expanded forms, and are used to demonstrate the accuracy of the solution.

7.7.1. Scalability.

The replicated model in Figure 7.10 is a hypothetical implementation of a typical search engine. It consists of one million customers requesting services from $100K$ index servers, which access, in turn, $50K$ and $10K$ Document and Ranking servers. The parameter K was varied to change the replication level of the components from 1 to $10,000$. Table 7.1 shows the number of times the core one-step MVA computation is executed to solve the various configurations and is an indication of the complexity of the computation. The fourth column shows that the number of steps is approximately 240 on average regardless of the scaling with the index server saturated. The algorithm is much more efficient when the model is not bottlenecked, because the iterations are sensitive to small changes in throughputs when the corresponding utilizations are high. The non-replicated model would not be solvable when $K = 10,000$ as there would be 1.6 million centers. It took less than 20 milliseconds to solve the LQN model for any value of K on a Pentium 4 2.8GHz machine running Windows XP using the LQNS version 3.8.

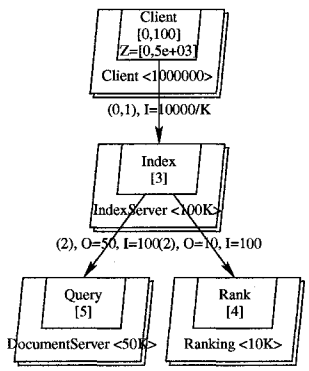


Figure 7.10: An LQN model for a typical search engine.

Table 7.1: Results of solving the LQN model in Figure 7.10.

K	Client Response Time (ms)	IndexServer Utilization	Number of Computational Steps
1	15341500	1.0	188
100	153276	1.0	214
500	30533.9	1.0	240
1000	15249.7	1.0	240
2000	8216.24	0.844	331
2500	7221.13	0.685	272
5000	6062.56	0.281	100
10000	5798.32	0.120	70

7.7.2. Accuracy.

The example system shown in Figure 7.3 is used to consider the accuracy of the approximations made in the replication computation. Three different MVA queueing network solvers were used to solve the layer submodels for the replicated model: the Schweitzer approximation, Linearizer, and exact MVA. The expanded system was solved using simulation; results are shown with 95% confidence levels.

Tables 7.2, 7.3 and 7.4 list the service times, throughputs and task utilizations respectively and the percentage error $((\text{approx} - \text{sim})/\text{sim}) \times 100\%$ of these values when compared to the simulation runs. These results show that the Schweitzer approximation gives the closest results to the full system on average with errors of less than 3% in magnitude for all of the cases, and in some instances the results lie within the bounds of the confidence levels of the simulation. Linearizer and Exact MVA fare less well, though the errors from these algorithms are still quite acceptable at less than 5% in magnitude.

The higher errors for the exact MVA and Linearizer may be explained by inspecting the MVA algorithm. Modifying the service time of the client delay server with the delay at the 'missing' centers is essentially estimating the residence times of the chains at the

Table 7.2: Service time results for the replication example in Figure 7.3.

Task	Expanded Model (Simulation)	Replicated Model					
		(Schweitzer)		(Linearizer)		(Exact MVA)	
			% Error		% Error		% Error
tA	33.10 ± 0.40	33.94	2.5	31.54	-4.7	31.64	-4.7
tB	73.19 ± 0.79	73.58	0.5	70.79	-3.3	70.92	-3.1

Table 7.3: Throughput results for the replication example in Figure 7.3.

Task	Expanded Model (Simulation)	Replicated Model					
		(Schweitzer)		(Linearizer)		(Exact MVA)	
			% Error		% Error		% Error
tA	0.0302 ± 0.0003	0.0295	-2.3	0.0317	5.0	0.0316	4.6
tB	0.0137 ± 0.0001	0.0136	-0.7	0.0141	2.9	0.0141	2.9
tC	0.2025 ± 0.0009	0.2005	-1.0	0.2128	5.1	0.2122	4.8
tD	0.0546 ± 0.0014	0.0544	-0.4	0.0565	3.5	0.0564	3.3

Table 7.4: Utilization results for the replication example in Figure 7.3.

Task	Expanded Model (Simulation)	Replicated Model					
		(Schweitzer)		(Linearizer)		(Exact MVA)	
			% Error		% Error		% Error
tC	0.6077 ± 0.0045	0.6016	-1.0	0.6385	5.1	0.6368	4.8
tD	0.2733 ± 0.0081	0.2718	-0.5	0.2826	3.4	0.2820	3.2

centers, R, in the MVA algorithm. However, in the case of exact MVA and Linearizer, the R values for different populations are required in the MVA iteration. For the exact MVA, the R values for the populations from $\mathbf{0}$ to $\mathbf{N}$ are required. For Linearizer, the R values for population $\mathbf{N}$ and $\mathbf{N} - 1_c$ are required. By modifying the service time of the client, an approximation of R for a population $\mathbf{N}$ is used, which is fixed throughout the iteration. That is, it is used even though for the exact MVA and R value for the range of populations

from 0 to N is needed. The approximation for R is incorrect and therefore produces larger errors in the exact MVA and Linearizer.

The error for the Schweitzer approximation is low on average since, in this algorithm, only the delays for population 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 expanded models are very close to their corresponding replicated model results which uses the Schweitzer MVA algorithm. In addition, the error for the service time result is increased, since the service 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 service time result by the number of visits. The Schweitzer algorithm is computationally more efficient than the Linearizer or thee exact MVA algorithms. Therefore, the Schweitzer algorithm is recommended for solving larger replicated models.

The Schweitzer MVA approximation gives the best results on average, therefore, its space and time complexity is discussed. The space requirements for Schweitzer is proportional to the product of the number of chains, C, and the number of centers, 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 centers needed, thereby reducing the space and time requirements 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 is $O(LN^2)$ or $O(L(\sum_{m=1}^{M} K_m)^2)$, where L is the number of layers [**135**]. This is derived from the time

Table 7.5: Time complexity of the Schweitzer, Linearizer, and Exact MVA algorithms.

Algorithm	Operations
Schweitzer	$1.08 \quad 10^4$
Linearizer	$1.46 \quad 10^6$
Exact	$1.21 \quad 10^{11}$

complexity of one iteration of Schweitzer which is $O(CN)$. Since the replication algorithm reduces the number of centers by representing a set of replicas by one center, the time complexity of one LQNS iteration is reduced to (LM^2) or by a factor of $(N/M)^2$[**118**]. Table 7.5 shows the difference in the complexity of the three MVA algorithms when solving this model. The column labelled "operations" is the number of times the residence time is computed by the MVA solver. The results show that the Schweitzer approximation is two orders of magnitude more efficient than the Linearizer and seven orders of magnitude more efficient than exact MVA.

The advantages of the replication approximation are obvious when comparing the computation time, the ease of reading the results, and the simplification of the model between the expanded model and the replicated model. Further, changing the level of replication with the replicated model amounts to a simple parameter change which can simplify parametric analysis.

Figure 7.11 shows a quite complex LQN model as a request jumps over a layer, since task C1 is called by both task A1 and task B1, and since task B1 is a server for task A1. Jumping of a request modifies the contention at task C1. Entry A1 can only make requests to entry C1 when entry A1 is not blocked at task B1. Entry B1 can only make requests to entry C1 when entry A1 is blocked, this is called interlocking [**55**]. All requests in the model are geometrically distributed. The threads' delays are fitted to the closed-form

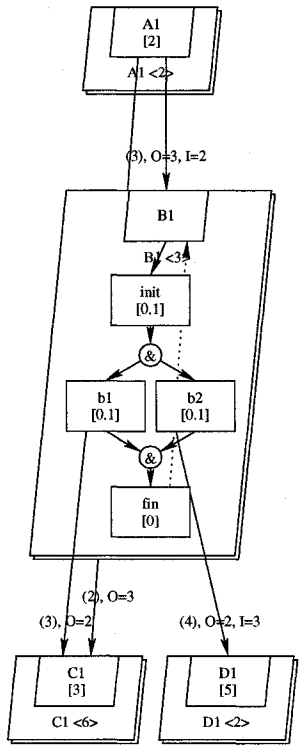


Figure 7.11: Replicated LQN model with interlocking.

formula derived in Chapter 5. The execution demands for the activities init, b1, and b2 are relatively low compared to the execution demands of the remote servers C1, and D1.

Table 7.6 shows the percentage error in the service time of task A1 (when solving the replicated LQN model using the solution technique developed in this chapter) is 8.45%, in the utilization of processor A1 is 7.71%, and in the throughput of task A1 is 7.83%.

Figure 7.12 shows an LQN model with two layers of servers, but without interlocking. When geometrically distributed requests are used, Table 7.7 shows that the maximum absolute value of the percentage error in the performance measures for task A1 and processor A1 is less than 1%. When all requests in the LQN model in Figure 7.12 are deterministic, the results are shown in Table 7.8. The results show that the maximum difference, between

Table 7.6: Results for the replicated LQN model in Figure 7.11.

	Expanded Model (Simulation)	LQNS (Linearizer)			
		Expanded Model		Replicated Model	
			% Error		% Error
A1 Service time	667.05± 22.945	660.68	0.95	723.678	8.45
proc. A1 Utilization	0.0029946±1.29e-5	0.00302718	1.09	0.00276366	-7.71
A1 Throughput	0.0014993±5.10e-5	0.00151359	1.35	0.00138183	-7.83

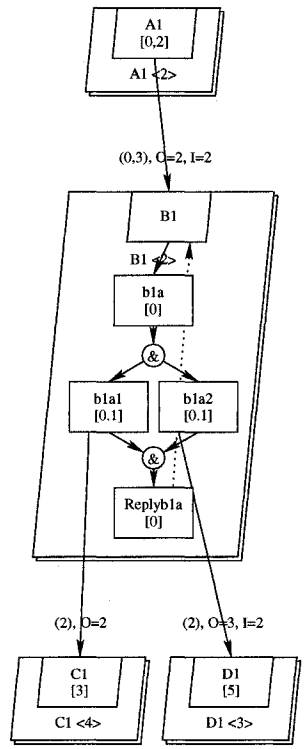


Figure 7.12: Replicated LQN model without interlocking.

the LQNS solution for the replicated model and the LQNS solution for the expanded model, in the absolute value of the percentage error is less than 5.1%.

Table 7.7: Results for the replicated LQN model in Figure 7.12 with geometric requests.

	Expanded Model (Simulation)	LQNS (Linearizer)			
		Expanded Model		Replicated Model	
			% Error		% Error
A1 Service time	338.48 ± 8.33	332.847	-1.66	336.1	-0.70
proc. A1 Utilization	0.0059318±5.15e-5	0.00600877	1.30	0.00595064	0.32
A1 Throughput	0.0029544±7.29e-5	0.00300438	1.69	0.00297532	0.71

Table 7.8: Results for the replicated LQN model in Figure 7.12 with deterministic requests.

	Expanded Model (Simulation)	LQNS (Linearizer)			
		Expanded Model		Replicated Model	
			% Error		% Error
A1 Service time	308.12 ±4.08	272.974	-11.40	285.883	-7.22
proc. A1 Utilization	0.0065379±1.04e-4	0.00732668	12.06	0.00699583	7.00
A1 Throughput	0.0032455±4.28e-5	0.00366334	12.87	0.00349791	7.78

7.8. Summary

The approach to describing and solving models with parallel threads in replicated components and subsystems, expands the ability to use analytic models for planning. It makes it easier to describe large parallel systems since each group of replicas is described only once. It is scalable in time and space, since the solution is insensitive to replication levels. Thus it is much faster than the solution of the expanded model.

This approach gives an advantage over previous replication-based solvers using state-based models (providing better scalability due to use of Mean Value Analysis), over replication analysis in queueing networks (because it handles extended queueing models with simultaneous resource possession) and over other work in replicated layered queues (in that it handles replicas with fanin).

A subtle advantage of this approach is that it makes replication a parameter of the model, so it can be rapidly studied as a parameter change, rather than requiring restructuring for each level of replication. The approximations necessary to use the present approach introduce some error. Using the Schweitzer MVA algorithm for the layer submodel solutions, errors under 3% were introduced. MVA algorithms which are more accurate for product form queueing networks (Linearizer and exact MVA) gave larger errors.

CHAPTER 8

Case Studies: Air Traffic Control and Information Management Systems

In this chapter, the techniques developed in this thesis are applied to two complex systems used in the industry. These are an Air Traffic Control (ATC) system and an Information Management System (IMS). The models developed in this chapter are not based on a specific real system. However, they represent my understanding of the architecture of those systems based on the literature. The purpose of these two case studies is to show the benefit of the results of the work presented in this thesis. The same modeling procedure developed here can be applied if real data is available.

8.1. Case Study I: Air Traffic Control System

An Air Traffic Control system is a large scale complex distributed system which requires high performance and high availability. For example, the US National Air Space (NAS) can have more than 5000 aircraft present in its airspace at the same time, a single ACT can communicate with more than 15 aircrafts at a given time, and there are about 17000 ATCs in the US NAS infrastructure each controlling a zone with a rough diameter from 20 to 200 miles [**15**]. The en-route airspace is the region away from the airports in which the aircraft normally fly at a high altitude. Air traffic is delivered to the en-route airspace from the Terminal Radar Approach Control (TRACON) facilities. The US contains more than 150 TRACON facilities and TRACONS may serve more than one airport [**15**].

In this thesis, the performance of a single ATC of an en-route system is evaluated. The performance model is based on the model developed in [**41**] which is a layered architecture shown in Figure 8.1. The *'OR'* and *'AND'* notations, enclosed in a dashed circle in Figure 8.1, refer to relationships used to compute availability and to reconfigure the system. 'OR' denotes that only one replica should be available at any given time, and 'AND' denotes that all replicas should be available at any given time. The open-headed arcs between entries denote asynchronous messages generated periodically from the radar and its front-end processing. In this thesis, the model in Figure 8.1 is changed to incorporate parallel threads in replicated LQN tasks. When a quorum is used in some tasks of the en-route ATC system, the system performance will be considered.

Each ATC system collects aircraft surveillance and weather data from radars. It communicates with other external subsystems such as other en-route facilities. In each ATC, air traffic services are provided by four subsystems [**41**]:

(1) A Surveillance Processing Service which collects radar data, from the Radar subsystem, and correlates it with individual aircraft routes.

(2) A Flight Plan Processing and Conflict Alert Service is provided by the Central subsystem.

(3) A Display Service is provided by the Console subsystem. The service displays aircraft position information collected from the radars. It allows air traffic controllers to modify flight plan data or change display format.

(4) A Monitoring Service is provided by the Monitor and Control subsystem. The service allows the monitoring and control of other ATC services to guarantee their availability policies.

Fault-tolerance is achieved by having redundant software server groups. There are up to three Display Management (DM) load balanced redundant servers per sector. In addition,

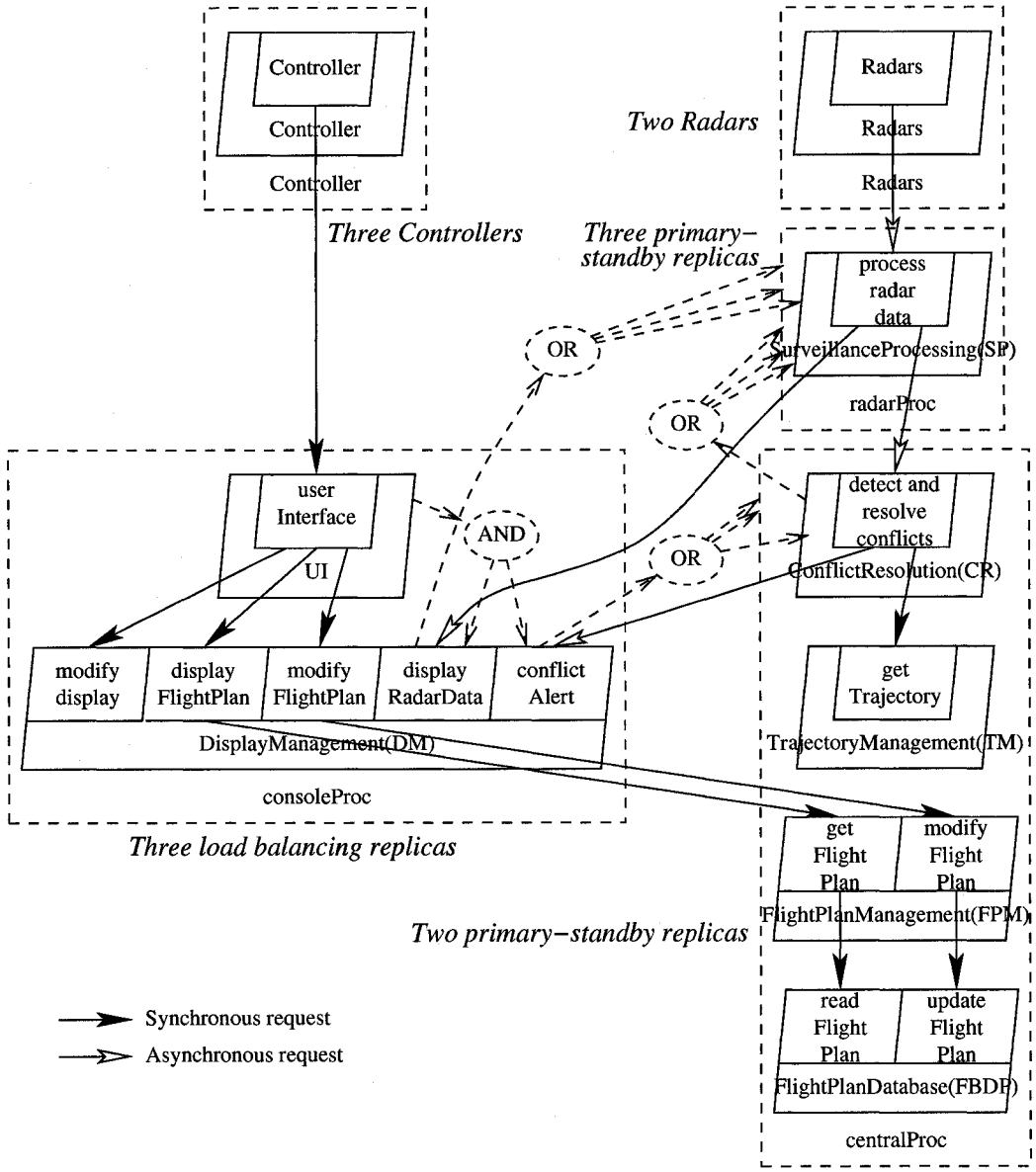


Figure 8.1: A Dependable-LQN model for an Air Traffic Control en-route system [**41**].

there are three primary/standby active redundant Surveillance Processing (SP) servers, and two primary/standby Flight Plan Management (FPM) servers. All three redundant SP

servers collect the raw radar data from the radars in parallel. However, only the primary server sends the processed radar data to the DM servers.

8.1.1. ATC with Quorum Pattern.

Figure 8.2 shows the LQN model for the system under study where the architecture in Figure 8.1 is modified to include internal parallelism in the DM task. It is assumed that there are two replicas for Flight Plan Management (FPM), and Flight Plan Database (FPDB) tasks. Each set of replicas runs on its own processor pFPlan. When entry DMdisplayFP (Display Management Display Flight Plan) is called, it makes requests to activity prep1 to prepare the request message, then it forks two threads. The first thread executes activity read1, and the second executes activity read2. Each thread makes requests to a separate replica of task FPM. For the purpose of performance analysis, it is assumed that the two replicas of task FPM run simultaneously. In addition, entry DMmodifyFP forks into two threads to modify the flight plan in the databases, activity confirm executes only after the two forked threads return, this is to guarantee data integrity in the databases.

Two different configurations of Figure 8.2 are considered for the performance study, both involving the forked threads of entry DMdisplayFP in Display Management subsystem. The first configuration uses a quorum of one ($J = 1$) for the read operation; it allows the Display Service (activity display in Figure 8.2) to run as soon as a response is received from either Flight Plan Management replicas, while the second waits for both responses (it uses a quorum of two). In either case, modifications to the flight plan (activity confirm in Figure 8.2) always wait for both replicas to complete their operations. Most parameters (shown in the LQN model in Figure 8.2) are chosen similar to those in [41], although the internal details of entries DMdisplayFP and DMmodifyFP in Figure 8.2 (that correspond respectively to displayFlightPlan and modifyFlightPlan in Figure 8.1) are changed to use quorum internal parallelism. The metric of interest is the mean service time for commands

issued by the controller at the console (service time of entry eController). The LQN model is solved analytically, using the LQNS, and by simulation. When the quorum is set to one at entry DMdisplayFP, the mean of the service time obtained from the LQNS solution is 2.4 seconds (the simulation result is 2.376 seconds with 95% confidence interval of ± 0.0140). When the quorum is set to two, the mean of the service time increases slightly to 2.5 seconds (the simulation result is 2.452 seconds with 95% confidence interval of ± 0.0606). This shows that there is only a very small performance gain when using a quorum of one in this model.

The industrial example of air traffic control demonstrates the scalability of the technique applied to large systems. The total analytic solution time, with about 300 points in each numerical distribution fitting is about 30 seconds. The computation of distributions is the major part of this time. The queueing computations are quite rapid.

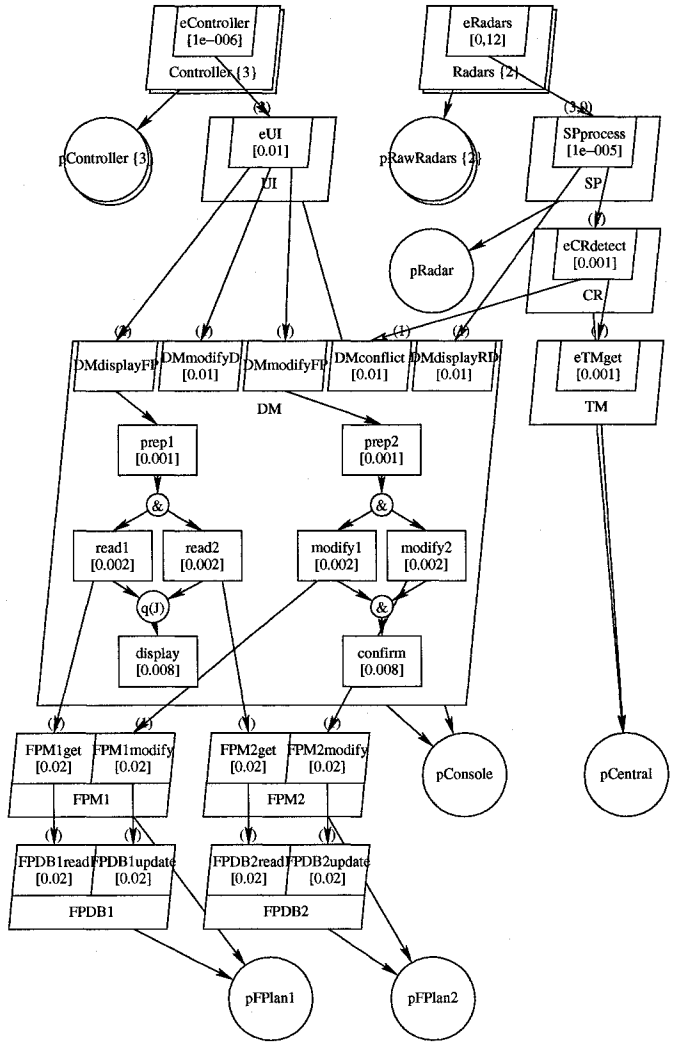


Figure 8.2: An LQN model for the Air Traffic Control system in Figure 8.1 with a quorum J.

8.1.2. ATC with Replicated Servers and Quorum Pattern.

In this section, an ATC system that has replicated tasks that use a quorum pattern is modeled. The LQN model for replicated tasks with quorum is shown in Figure 8.3. Note that the asynchronous requests in Figure 8.2 are changed to synchronous requests with second phases. This is to be able to use replication notation, as the replicated tasks should only be called using synchronous requests. Allowing a replicated task to receive asynchronous requests is a future work.

Table 8.1 shows the results for the service time of entry eController when using quorum of $J = 2$ out of $N = 2$. The first row shows the solution of the LQN when the replicated model is solved using the LQNS. The second row shows the solution when the replicated model is transformed to an expanded version then solved using the LQNS. Finally, the last row shows the simulation results for the expanded model. It can be seen from the results that the absolute value of the percentage error in the mean of the replicated model is 5.28% compared to the LQNS solution of the expanded model, and is 19.1% in the mean of the expanded model compared to the simulation of the expanded model. The 19.1% error is partly due to the LQNS overlap approximation (using Mak and Lundstrom technique [**102**]). The overlap occurs between the main thread of control and the forked threads in task DM. This problem is out of the scope of this thesis. The focus of this thesis is on the accuracy of the LQNS solution of the replicated model compared to the solution of the expanded model. However, another possible source of errors is because the assumptions of the closed-form formula for geometric requests, derived in Section 5.2, are violated in task DM. The service times of the servers of the threads in task DM are non-exponentially distributed, because tasks FPM request services from lower layer servers. Furthermore, there is a dependency among the RVs associated with the delays of the forked threads in

Table 8.1: Results for the service time of eController in the LQN model in Figure 8.3 when $J = 2$.

	Mean	Variance
LQNS (replicated model)	1.690	4.85
LQNS (expanded model)	1.784	5.37
Sim (expanded model)	2.206 (95% ± 0.081543)	9.08 (95% ±1.0714)

Table 8.2: Results for the service time of eController in the LQN model in Figure 8.3 when $J = 1$.

	Mean	Variance
LQNS (replicated model)	1.659	4.71
LQNS (expanded model)	1.742	5.16
Sim (expanded model)	2.171 (95% ±0.021647)	8.84 (95% ±0.2815)

task DM when the forked threads compete for their host processor, which is a single server with a processor sharing scheduling discipline.

Table 8.2 shows the results for the service time of entry eController when using quorum of $J = 1$ out of $N = 2$. The first row shows the solution of the LQN model when the replicated model is solved using the LQNS. The second row shows the solution when the replicated model is transformed to an expanded version then solved using the LQNS. The last row shows the simulation results for the expanded model. The absolute value of the percentage error in the mean of the service time of eController in the solution of the replicated model compared to the LQNS solution of the expanded model is 4.76%.

The case study shows that models for complex real systems can be solved rapidly and with an adequate accuracy when compared to the expanded analytic model solution. However, the accuracy of the final solution might suffer from other approximations used in the LQNS.

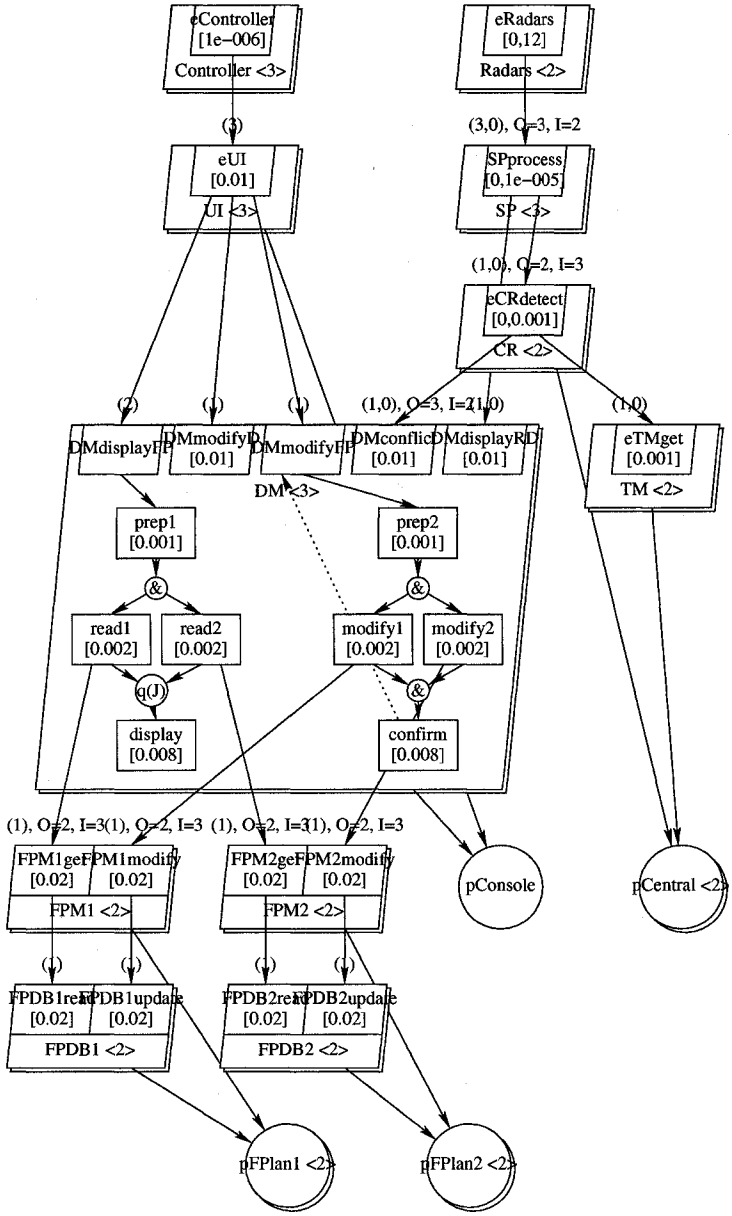


Figure 8.3: An LQN model for the ATC system in Figure 8.1 with replicated servers having internal quorum parallelism.

8.2. Case Study II: Information Management System

This case study has two configurations shown in Figures 8.4 and 8.5. It describes a large Information Management System, which accesses two back end databases (called RF and BC) through local servers (shown as the "LAN servers") which do routing and some processing. Some of the LANs are connected to the backbone network and others are connected to a wide area network (WAN). The configurations differ in that the second configuration in Figure 8.5 has "regional servers" (shown as "RS_DB") to off-load work from one of the two databases.

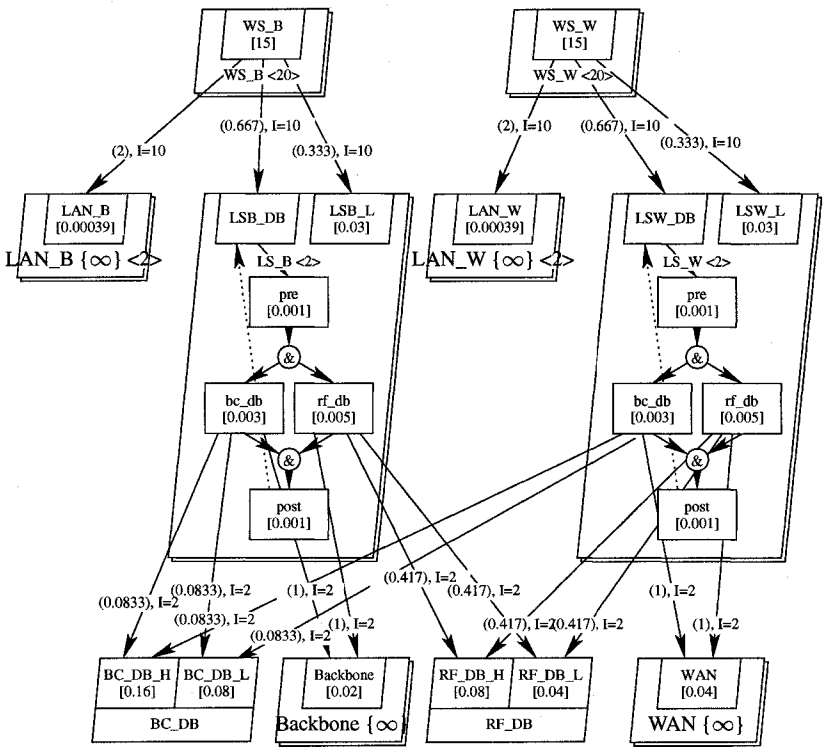


Figure 8.4: An LQN model for the base case of the database system.

The model uses simplifying assumptions to obtain symmetry in the entities. In the model, the workstations are identical sources of workload, and each LAN has the same

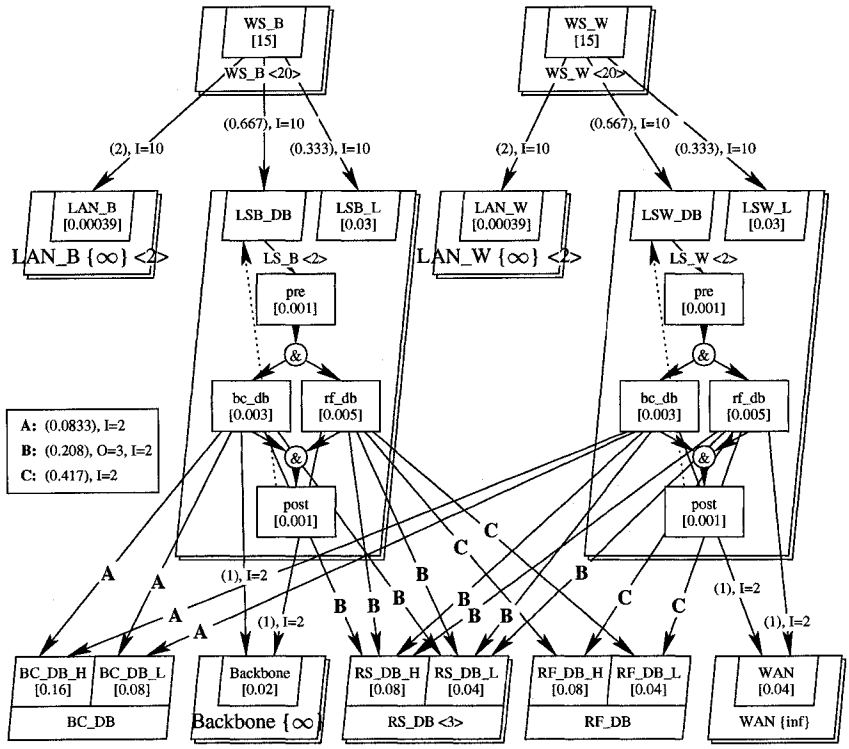


Figure 8.5: An LQN model for the regional servers' case.

number of workstations. The number of LANs attached to the WAN equals that attached to the backbone. However, the two databases are different. In the model without regional server, shown in Figure 8.4, the users and their workstations are the two sets of client tasks at the top, one set for the backbone and one for the WAN connection. The LAN servers are in the middle, they fork their requests to the database servers and WAN backbone. The two database servers are modelled as server tasks in the bottom layer. The LANs are modelled as delay servers attached to the Client workstations which use them and the WAN and backbone are similar delay servers attached to the LAN servers. The database server subsystems have been greatly simplified, and the effects of the communications front end and the storage are incorporated into the parameters of the database server tasks.

The LQN model in Figure 8.4 is used to study the performance of the system without regional servers, under varying configurations and parameters. The service time at the workstations is determined when the number of workstations in the system is increased, and the results are shown in Figure 8.6, labeled as the "base case". There are ten workstations attached to each LAN (fanin $I = 10$). Note that there are two LAN_W attached to twenty WS_W workstations, and there are two LAN_B attached to twenty WS_B workstations. The number of WS_W and WS_B workstations is increased by attaching new LANs to the network. As expected, the service time increases with the number of workstations. The service time increases dramatically at more than 600 clients since the RF database becomes saturated between 500 and 600 clients. The RF database is the first component to saturate and is the bottleneck of the system. After saturation, the service time continues to increase linearly.

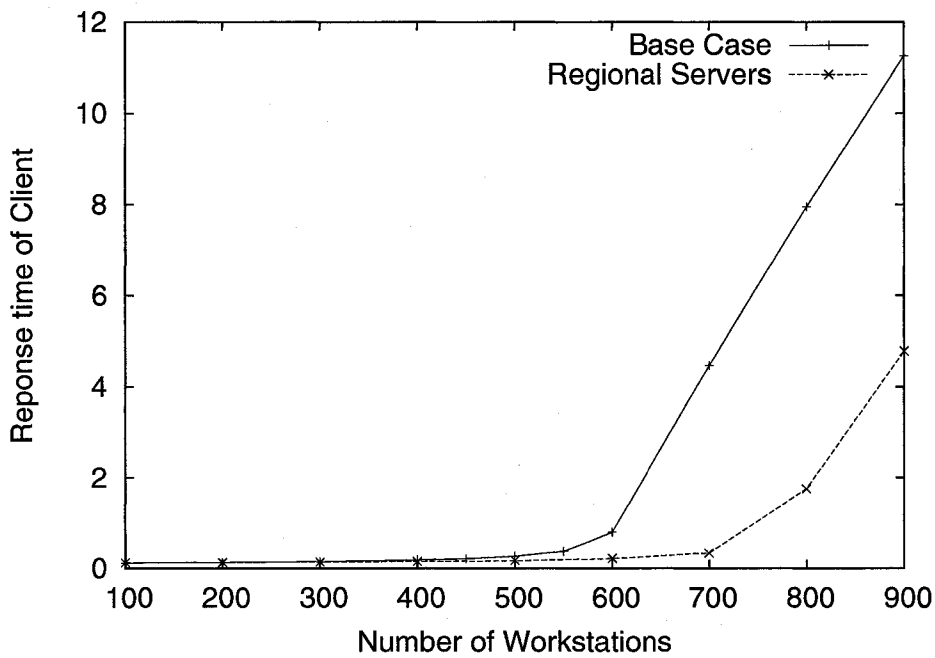


Figure 8.6: Performance of the database system in Figure 8.4.

The second case using regional servers is shown in Figure 8.5. Three regional servers, RS_DB, are introduced into the design to off-load the work at the RF database, by handling transactions that are local to the region of the originating workstation. 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. Figure 8.7 shows the effect of off-loading to regional servers on the service time at clients. Figure 8.6 (regional servers' case) shows the service time observed at the client with 20% of the RF database traffic routed to the regional servers. Clearly, the service time at the client is improved and the RF database now saturates only between 700 and 800 clients. This example shows a typical use of the replicated solver in a planning context, and is based on an industrial system. Further, it shows the use of replication level (of the LAN servers) as a parameter, rather than a structural change in the model.

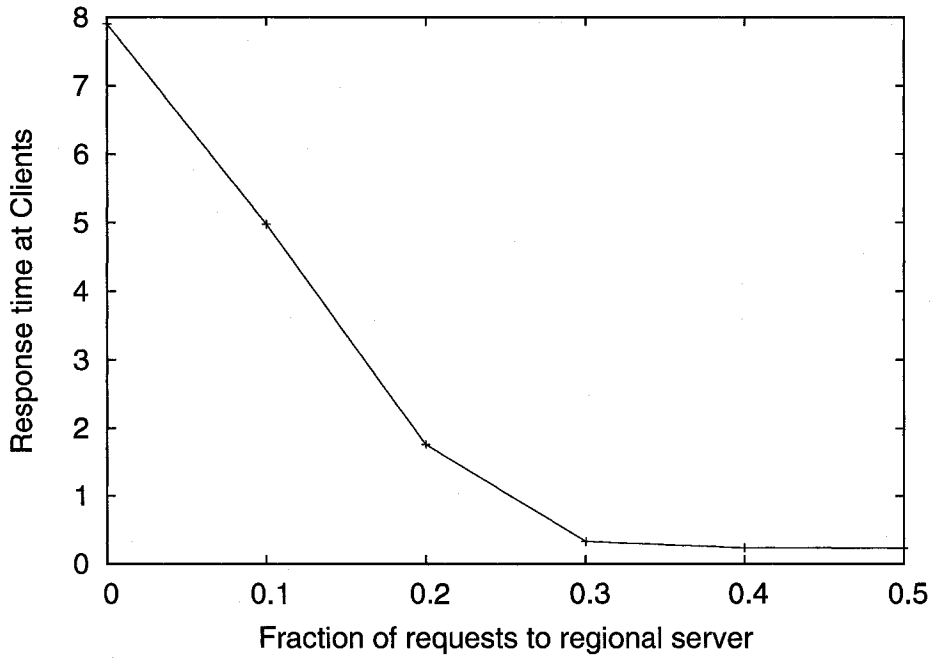


Figure 8.7: Effect on performance of off-loading to regional servers.

CHAPTER 9

Conclusions

This chapter identifies the main contributions of this thesis, and draws conclusions. Finally, future directions to extend this work are identified.

9.1. Contributions

The contribution of this thesis is to extend the Layered Queueing Network formalism and its solver LQNS to meet the challenges of large distributed systems, especially those systems with replication components. These challenges include the need to consistently compare the performance patterns of different replication techniques in a replication system. This requires a performance modeling tool that is rapid, accurate, and rich in modeling constructs. To do so, different replication patterns were identified and modeled from a performance perspective. Then, new approximations were created. Known solution techniques, that had not been used in combination before, were combined. Finally, the accuracy of these approximations was evaluated using many examples. The contributions were used in case studies drawn from real systems. The result of this research was an extended LQNS that a designer of a distributed software system could use to compute relevant performance measures at an early stage of the system design.

The contributions are summarized under the headings of performance replication patterns, modeling power enhancement, accuracy improvement, and case studies.

9.1.1. Performance Replication Patterns.

Performance replication patterns were identified and developed in the LQN formalism. These are the master-slave, master-slave snapshot, master-slave cascading, consolidation, read-one-write-all, quorum, master-master, and select patterns. The patterns represent different architectures and behaviors that affect the performance of a system. They enable a developer to compare different replication techniques consistently. Designers could choose from these patterns for their specific application and needs.

9.1.2. Modeling Power Enhancement.

Internal parallelism, in an LQN task, is a technique to enhance performance of a system when the task requires the responses of remote servers that run in parallel. A new quorum pattern for the joining threads was defined and implemented in the LQNS. In this pattern, a main thread of control (that is forked into N threads) is allowed to proceed to completion when J out of the N forked threads respond, where $J \leq N$. The pattern is useful to model quorum consensus protocols, or broadcast and multicast protocols for example.

The effect of the overhanging (delayed) threads on the contention of the main thread of control for resources, after the quorum-join, was incorporated in the solution technique developed for the quorum pattern. The effect of the overhanging threads was approximated by converting an LQN task with a quorum pattern to a new LQN task without one, so that existing solution techniques could be used.

The LQN formalism was extended and the LQNS was modified to model and solve replication patterns in the level of tasks with internal parallelism for closed models by exploiting symmetry in replicated LQN tasks. Symmetry is exploited by solving the underlying queueing networks of an LQN model only for one replica, then distributing the results to the other replicas accordingly. This is especially beneficial when the performance replication patterns are part of a large system model and it is required to use the compact

replication notation to take advantage of the speed, scalability, and expressiveness that the replication notation provides.

When a replicated task uses a quorum pattern, an approximation for the replicated overhanging threads was developed. The solution technique combined the technique of solving replicated tasks with an AND fork-join pattern with the technique developed to approximate the quorum delay.

9.1.3. Accuracy Improvement.

To compute the quorum delay of an AND-fork and a quorum-join in the LQNS, the cumulative distribution function of each forked thread had to be approximated. A previous approximation method, called three-point, was proved to be inadequate for the cases when $J < N$. In this thesis, closed-form formulas for the thread delay distribution, for the cases when the thread makes deterministic or geometrically distributed requests to lower layer servers, were derived. The accuracy of the LQNS solution when $J < N$ was significantly improved. In addition, the accuracy of the existing AND fork-join , $J = N$, was also improved when using the new CDFs.

Generally, the computation of the quorum delay and the service time of an entry was most sensitive to the early part of the CDF when only the response of one forked thread was required. The computation was least sensitive when the responses of all forked threads were required. In general, when the forked threads were heterogeneous the accuracy of the approximation was better than the case when the threads were homogeneous.

In the derivation of the CDFs for both the geometrically distributed and deterministic requests, it was assumed that the service times of the lower layer servers were exponentially distributed. This assumption was violated when the remote servers make requests to other lower layer servers. The accuracy suffered in that case, but still, that gave better accuracy in the performance measures than the three-point approximation.

When the number of requests was geometrically distributed, the accuracy of the mean and variance of a task service time when using the derived CDF was superior to both the three-point and the gamma approximations of the CDF. Even if the assumptions required for the closed-form derivation were violated (i.e., the number of servers a client makes requests to is more than one and the servers' service times are non-exponentially distributed) the errors are mostly acceptable.

When the number of request was deterministic, the gamma approximation was better than the three-point approximation by virtue of the smaller errors in the variance; the errors in the mean were close. The accuracy of the new closed-form formulas for deterministic requests was comparable to the gamma distribution fitting, but they gave high errors when the servers' service times are non-exponentially distributed. Therefore, the gamma distribution should be used for the fitting of the thread delay for deterministic requests, as it is computationally more efficient than the closed-form formula derived for deterministic requests.

In an LQN model, a thread can only make one type of request, either geometrically distributed or determinsitic, but the model can have a mixture of both deterministic and geometrically distributed requests. In this case, each thread delay is approximated using the CDF derived for the type of request the thread makes.

9.1.4. Case Studies.

Analytic models for an Air Traffic Control (ATC) system and an industrial Information Management System (IMS) were constructed and solved using the modified LQNS.

Two models for the ATC were created. The first model used the new quorum pattern. That model demonstrated the scalability of the technique applied to large systems. The total analytic solution time, with about 300 points in each numerical distribution fitting, was about 30 seconds. The computation of distributions was the major part of this time;

the queueing computations are quite rapid. The second ATC model used the new notation for replication with the quorum as the method of internal parallelism. The model indicated that models for complex real systems can be solved rapidly and with an adequate accuracy using the solution techniques developed in this thesis. However, the accuracy of the final solution might suffer from other approximation used in the LQNS.

The second case study was a model for an IMS. The model used the replication with AND-forks and AND-joins for internal parallelism. This model showed a typical use of the replicated solver in a planning context in an industrial system. Furthermore, the model showed the use of the replication level of a server as a parameter, rather than a structural change in the model.

9.2. Future Work

The closed-form formulas derived for the cumulative distribution function of a thread delay have the limitation that the service times of the lower layer servers are exponentially distributed. The accuracy may suffer when the servers request service from other lower layer servers. Further research is needed to include the variances of the servers' service times in the closed-form formulas to improve the accuracy further.

The semantic for the quorum pattern, defined in this thesis, is that the task running the threads does not accept a new customer until after all threads, including the delayed ones, complete execution. However, some systems may use different semantics. One possible semantic is to accept a new customer in the task right after the main thread of control completes execution. Another possibility is to terminate all delayed threads. Termination can be done at the local host, remote hosts, or both. Further research is needed to model these semantics and compute their performance effect analytically.

Finally, the replication semantic in the LQN formalism is strict. It assumes a fully balanced symmetry among the replicated tasks. The structural symmetry and behavioral symmetry were introduced in this thesis. Further research is needed in this area to relax those symmetry constraints and provide a solution with a tolerated accuracy.

Bibliography

[1] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables*. Dover Publications, 1965.

[2] Ada. The programming language ada reference manual. *Lecture Notes in Computer Science, Springer-Verlag, Berlin*, 155, 1983.

[3] D. Agrawal, O. Egecioglu, and A. Abbadi. Analysis of quorum-based protocols for distributed (k+1)-exclusion. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):533 – 7, May 1997.

[4] M. Ahamad and M.H. Ammar. Performance characterization of quorum-consensus algorithms for replicated data. *IEEE Transactions on Software Engineering*, 15(4):492–496, April 1989.

[5] Y. Amir and A. Wool. Evaluating quorum systems over the internet. *Proceedings of the Twenty-Sixth International Symposium on Fault-Tolerant Computing*, pages 26–35, 1996.

[6] T. Anderson and P. A. Lee. *Fault tolerance. Principles and practice*. Prentice-Hall, 1981.

[7] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[8] M. Arai, T. Suzuki, M. Ohara, S. Fukumoto, K. Iwasaki, and Hee Yong Youn. Analysis of read and write availability for generalized hybrid data replication protocol. *Proceedings. 10th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 143–50, 2004.

[9] François Baccelli and Edward G. Coffman. A data base replication analysis using an M/M/m queue with service interruptions. *SIGMETRICS Performance Evaluation Review*, 11(4):102–107, 1982.

[10] B. R. Badrinath and T. Imielinski. Replication and mobility. In *Second Workshop on the Management of Replicated Data (Cat. No.92TH0489-5)*, pages 9–12, Monterey, CA, USA, 1992. IEEE Comput. Soc. Press.

[11] S. Balsamo. Product form queueing networks. *Perfomance Evaluation: Origins and Directions*, 1769:377–401, 2000.

[12] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[13] Sujata Banerjee, Victor O. K. Li, and Chihping Wang. Performance analysis of the send-on-demand: A distributed database concurrency control protocol for high-speed networks. *Computer Communications*, 17(3):189–204, 1994.

[14] C. Basile, Z. Kalbarczyk, and R.K. Iyer. Active replication of multithreaded applications. *IEEE Transactions on Parallel and Distributed Systems*, 17(5):448–65, May 2006.

[15] Alexandre M. Bayen. *Computational Control of Networks of Dynamical Systems: Application to the National Airspace System*. PhD thesis, Stanford University, 2003.

[16] Azer Bestavros and Spyridon Braoudakis. Timeliness via speculation for real-time databases. In *Real-Time Systems Symposium*, pages 36–45, 1994.

[17] B. Bhargava, K. Friesen, A. Helal, S. Jagannathan, and J. Riedl. *Design and implementation of the RAID-V2 distributed database system.* Technical report cse-tr-962, Purdue University, 1990.

[18] Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):47–76, 1987.

[19] A. Borg, J. Baumbach, and S. Glazer. A message system supporting fault tolerance. *Operating Systems Review*, 17(5, special issue):90–99, 1983.

[20] E. Born. Analytical performance modelling of lock management in distributed systems. *Distributed Systems Engineering*, 3(1):68–76, 1996.

[21] A. J. Borr. Transaction monitoring in encompass [tm]: reliable distributed transaction processing. In *Seventh International Conference on Very Large Data Bases*, pages 155–165, Cannes, France, 1981. IEEE.

[22] C. J. Bouras and P. G. Spirakis. Performance modeling of distributed timestamp ordering: perfect and imperfect clocks. *Performance Evaluation*, 25(2):105–130, 1996.

[23] P. Buchholz. Efficient analysis techniques for symmetric multiprocessor architectures. In *Fifth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 125–130, Haifa, Israel, 1997. IEEE.

[24] R. Buerraoui, P. Felber, B. Barbinato, and K. Mazouni. System support for object groups. In *OOPSLA'98: Conference on Object-Oriented Programming, Systems, Languages and Applications*, pages 244–58, Vancouver, BC, Canada, 1998. ACM.

[25] Marie Buretta. *Data Replication: Tools and Techniques for Managing Distributed Information*. John Wiley & Sons, New York, 1997.

[26] Lorenzo Capra, Claude Dutheillet, and Jean Michel Iliè Giuliana Franceschinis. Towards performance analysis with partially symmetrical swn. In *Seventh International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '99)*, pages 148–155, College Park, MD, USA, 1999. IEEE.

[27] M. J. Carey and M. Livny. Conflict detection tradeoffs for replicated data. *ACM Transactions on Database Systems*, 16(4):703–746, 1991.

[28] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[29] K. Mani Chandy and Doug Neuse. Linearizer: a heuristic algorithm for queueing network models of computing systems. *Communications of the ACM*, 25(2):126–134, 1982.

[30] K. Mani Chandy and C. H. Sauer. Approximate methods for analyzing queueing network models of computing systems. *Computing Surveys*, 10(3):281–317, 1978.

[31] Shu-Wie Chen and Calton Pu. *A Structural Classification of Integrated Replica Control Mechanisms*. Technical report cucs-006-92, Columbia University, 1992.

[32] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. Scan: a dynamic, scalable, and efficient content distribution network. In *Pervasive Computing. First International Conference, Pervasive 2002.*, pages 282–296, Zurich, Switzerland, 2002. Springer-Verlag.

[33] D. R. Cheriton. The v distributed system. *Communications of the ACM*, 31(3):314–333, 1988.

[34] David R. Cheriton, Michael A. Malcolm, Lawrence S. Melen, and Gary R. Sager. Thoth, a portable real-time operating system. *Communications of the ACM*, 22(2):105–115, 1979.

[35] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: a high performance scheme for maintaining replicated data. *IEEE Transactions on Knowledge and Data Engineering*, 4(6):582–592, 1992.

[36] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, 1993.

[37] Bruno Ciciani, Daniel M. Dias, and Philip S. Yu. Analysis of replication in distributed database systems. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):247–261, 1990.

[38] Bruno Ciciani, Daniel M. Dias, and Philip S. Yu. Analysis of concurrency-coherency control protocols for distributed transaction processing systems with regional locality. *IEEE Transactions on Software Engineering*, 18(10):899–914, 1992.

[39] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath (Ed.), editor, *International Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, USA, 2000. Springer-Verlag GmbH.

[40] Edward G. Coffman, Erol Gelenbe, and Brigitte Plateau. Optimization of the number of copies in a distributed data base. *IEEE Transactions on Software Engineering*, 7(1):78–84, 1981.

[41] Olivia Das and Murray Woodside. Dependability modeling of self-healing client-server applications. In R. De Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems II*, volume 3069 of *Lecture Notes in Computer Science*, pages 266–285. Springer-Verlag, December 2004.

[42] Edmundo de Souza e Silva and Richard R. Muntz. Approximate solutions for a class of non-product form queueing network models. *Performance Evaluation*, 7(3):221–242, 1987.

[43] Edmundo de Souza e Silva and Richard R. Muntz. Queueing networks: Solutions and applications. In Hideaki Takagi, editor, *Stochastic Analysis of Computer and Communication Systems*, pages 319–399. North Holland, Amsterdam, 1990.

[44] Graham de Vahl Davis. *Numerical methods in engineering & science*. Allen & Unwin (Publishers) Ltd, London; Boston, 1986.

[45] A. Derhab and N. Badache. A pull-based service replication protocol in mobile ad hoc networks. *European Transactions on Telecommunications*, 18(1):1–11, January 2007.

[46] S. Derisavi, P. Kemper, and W. H. Sanders. Symbolic state-space exploration and numerical analysis of state-sharing composed models. *Linear Algebra and Its Applications*, 386:137–166, 2004.

[47] K. Diks, E. Kranakis, D. Krizanc, B. Mans, and A. Pelc. Optimal coteries and voting schemes. *Information Processing Letters*, 51(1):1–6, July 12 1994.

[48] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1-2):105–131, 1996.

[49] K.P. Eswaran, J. N. Gray, R. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19:624–633, 1976.

[50] Arlette Ferrier and Christine Stangret. Heterogeneity in the distributed database management system sirius-delta. In *Very Large Data Bases, International Conference on Very Large Data Bases*, pages 45–53, 1982.

[51] Greg Franks, Alex Hubbard, Shikharesh Majumdar, John Neilson, Dorina Petriu, Jerome Rolia, and Murray Woodside. A toolset for performance engineering and software design of client-server systems. *Performance Evaluation*, 24(1-2):117–136, 1995.

[52] Greg Franks, Peter Maly, Murray Woodside, Dorina C. Petriu, and Alex Hubbard. *Layered Queueing Network Solver and Simulator User Manual*. Real-time and Distributed Systems Lab, Carleton University, Ottawa. http://www.sce.carleton.ca/rads/lqn/lqn-documentation/.

[53] Greg Franks and Murray Woodside. Performance of multi-level client-server systems with parallel service operations. In *Proceedings of the First International Workshop on Software and Performance (WOSP '98)*, pages 120–130, Santa Fe, NM, October 12–16 1998. ACM Sigmetrics, Association for Computing Machinery.

[54] Greg Franks and Murray Woodside. Effectiveness of early replies in client-server systems. *Performance Evaluation*, 36-37:165–183, 1999.

[55] Roy Gregory Franks. *Performance Analysis of Distributed Server Systems*. PhD thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, December 1999.

[56] R. Gallersdorfer and M. Nicola. Improving performance in replicated databases through relaxed coherency. In *21st International Conference on Very Large Data Bases (VLDB '95)*, pages 445–456, Zurich, Switzerland, 1995. Morgan Kaufmann.

[57] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google file system. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, pages 29 – 43, Bolton Landing, NY, USA, October 19 – 23 2003. Association for Computing Machinery.

[58] D. K. Gifford. Weighted voting for replicated data. In *Seventh Symposium on Operating Systems Principles*, pages 150–162, Pacific Grove, CA, USA, 1979. ACM.

[59] S. Gilmore and J. Hillston. The pepa workbench: a tool to support a process algebra-based approach to performance modelling. In *7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 353–368, Vienna, Austria, 1994. Springer-Verlag.

[60] Richard A. Golding and Darrell D. E. Long. Quorum-oriented multicast protocols for data replication. In *International Conference on Data Engineering*, pages 490–497, Tempe, AZ, USA, 1992. IEEE.

[61] I. S. Gradshteyn and I. M. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, San Diego, CA, 6th edition, 2000.

[62] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *ACM SIGMOD International Conference on Management of Data*, volume 25 of *SIGMOD Rec. (USA)*, pages 173–182, Montreal, Que., Canada, 1996.

[63] Donald Gross and Carl M. Harris. *Fundamentals of Queueing Theory*. Wiley-Interscience, 3rd edition, 1998.

[64] S. Haddad, J. M. Ilie, M. Taghelit, and B. Zouari. Symbolic reachability graph and partial symmetries. In *16th International Conference on Applications and Theory of Petri Nets*, pages 238–257, Torino, Italy, 1995. Springer-Verlag.

[65] Vassos Hadzilacos and Sam Toueg. Fault-tolerant broadcasts and related problems. In *Distributed systems (2nd Ed.)*, pages 97–145. ACM Press/Addison-Wesley Publishing Co., 1993.

[66] Yu Haifeng and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Transactions on Computer Systems*, 20(3):239–282, 2002.

[67] Philip Heidelberger and Kishor S. Trivedi. Analytic queueing models for programs with internal concurrency. *IEEE Transactions on Computers*, C-32(1):73–82, 1983.

[68] Abdelsalam A. Helal, Abdelsalam A. Heddaya, and Bharat B. Bhargava. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, 1996.

[69] M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53, February 1986.

[70] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the tipptool. *Performance Evaluation*, 39(1-4):5–35, 2000.

[71] Jane Hillston. *A compositional approach to performance modelling*. Cambridge University Press, Cambridge; New York, 1996.

[72] H. Holm and M. S. Alouini. Sum and difference of two squared correlated nakagami variates in connection with the McKay distribution. *IEEE Transactions on Communications*, 52(8):1367–1376, August 2004.

[73] S. L. Hung and K. Y. Lam. Performance study of 2 phase locking in distributed database system with mixed transaction classes. In *24th Annual Computer Simulation Conference*, pages 289–293, Reno, NV, USA, 1992. SCS.

[74] Oliver C. Ibe, Hoon Choi, and Kishor S. Trivedi. Performance evaluation of client-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(11):1217–1229, 1993.

[75] IBM. *Replication Guide and Reference, DB2: Number SC26-9642-00.* PhD thesis, IBM, 1999.

[76] Patricia A. Jacobson and Edward D. Lazowska. Analyzing queueing networks with simultaneous resource possession. *Communications of the ACM*, 25(2):142–151, 1982.

[77] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, New York, 1991.

[78] Bao-Chyuan Jenq, Walter H. Kohler, and Don Towsley. Queueing network model for a distributed database testbed system. *IEEE Transactions on Software Engineering*, 14(7):908–921, 1988.

[79] A. Jhingran and M. Stonebraker. Alternatives in complex object representation: a performance perspective. In *Sixth International Conference on Data Engineering (Cat. No.90CH2840-7)*, pages 94–102, Los Angeles, CA, USA, 1990. IEEE Comput. Soc.

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

[81] Ricardo Jimenez-Peris, Marta Patino-Martinez, and Sergio Arevalo. Deterministic scheduling for transactional multithreaded replicas. In *IEEE Symposium on Reliable Distributed Systems*, pages 164–173, 2000.

[82] T. A. Joseph and K. P. Birman. Low cost management of replicated data in fault-tolerant distributed systems. *ACM Transactions on Computer Systems*, 4(1):54–70, 1986.

[83] Bettina Kemme. *Database Replication for Clusters of Workstations*. Phd thesis, Swiss Federal Institute of Technology Zurich, 2000.

[84] Pekka Khkipuro. *Performance Modeling Framework for CORBA Based Distributed systems*. Phd thesis, University of Helsinki, 2000.

[85] Leonard Kleinrock. *Theory, Volume 1, Queueing Systems*. John Wiley & Sons, 1975.

[86] Diwakar Krishnamurthy and Jerome Rolia. Prediting the QoS of an electronic commerce server: Those mean percentiles. *Performance Evaluation Review*, 26(3):16–22, December 1998.

[87] A. Kumar. Performance analysis of a hierarchical quorum consensus algorithm for replicated objects. *The 10th International Conference on Distributed Computing Systems*, pages 378–85, 1990.

[88] H. T. Kung and J.T. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database systems*, 6(2):213–226, 1981.

[89] Toshiyasu Kurasugi and Issei Kino. Approximation methods for two-layer queueing models. *Performance Evaluation*, 36-37:55–70, 1999.

[90] H. Lamehamedi, Shentu Zujun, B. Szymanski, and E. Deelman. Simulation of dynamic data replication strategies in data grids. In *International Parallel and Distributed Processing Symposium*, page 10 pp., Nice, France, 2003. IEEE Comput. Soc.

[91] Edward D. Lazowska, John Zahorjan, Graham G. Scott, and Kenneth C. Sevcik. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc., NJ, USA, 1984.

[92] Edward D. Lazowska, John Zhorjan, Scott G. Graham, and Kenneth C. Sevcik. *Quantitative System Performance; Computer System Analysis Using Queueing Network Models*. Prentice Hall, Englewood Cliffs, NJ, 1984.

[93] Louis-Marie Le Ny and C. Murray Woodside. Performance modelling of queues with rendezvous service. Technical Report 941, Institut National de Recherche en Informatique et en Automatique (INRIA), Domaine de Voluceau, Rocquencourt, B.P.105, 78153 Le Chesnay Cedex, France, December 1988.

[94] Kin K. Leung. Update algorithm for replicated signaling databases in wireless and advanced intelligent networks. *IEEE Transactions on Computers*, 46(3):362–367, 1997.

[95] M. Litoiu, J. Rolia, and G. Serazzi. Designing process replication and activation: a quantitative approach. *IEEE Transactions on Software Engineering*, 26(12):1168–1178, 2000.

[96] Marin Litoiu. *Designing High Performance Software Systems - A Quantitative Approach*. Phd thesis, Carleton University, 1999.

[97] J. D. C. Little. A proof for the queueing formula: $L = \lambda W$. *Operations Research*, 9:383–387, 1961.

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

[99] T.J. Liu. A simulation study of database application in a distributed system: data replication. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2003)*, 2:616–20, 2003.

[100] T. Loukopoulos, I. Ahmad, and D. Papadias. An overview of data replication on the internet. In *International Symposium on Parallel Architectures, Algorithms and Networks. I-SPAN'02*, pages 31–36, Makati City, Metro Manila, Philippines, 2002. IEEE Comput. Soc.

[101] R.E. Lyons and W. Vanderkulk. The use of triple modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6(2):200–209, 1962.

[102] Victor W. Mak and Stephen F. Lundstrom. Predicting performance of parallel computations. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):257–270, 1990.

[103] John McDermott and Ravi Mukkamala. Analytic performance comparison of transaction processing algorithms for the sintra replicated-architecture database system. *Journal of Computer Security*, 4(2-3):189–228, 1996.

[104] J.D. Meier, Srinath Vasireddy, Ashish Babbar, and Alex Mackman. *Improving .NET Application Performance and Scalability*. Microsoft Corporation, 2004.

[105] D. A. Menasce. Two-level iterative queuing modeling of software contention. In *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS 2002)*, pages 267–276, Fort Worth, TX, USA, 2002. IEEE.

[106] Daniel A. Menasce, Virgilio A. F. Almeida, and Larry W. Dowdy. *Capacity planning and performance modeling: from mainframes to client-server systems*. Prentice Hall, Englewood Clis, New Jersey, 1994.

[107] Y. Miyanishi, K. Nakamura, F. Sato, T. Watanabe, and T. Mizuno. An analysis of data updating performance in distributed systems and a proposal of a data updating algorithm. In *10th International Conference on Information Networking, ICOIN-10*, pages 53–59, Kyung-ju, South Korea, 1996. POSTECH Inf. Res. Lab.

[108] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia, and C. A. Lingley-Papadopoulos. Totem: a fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63, 1996.

[109] L. E. Moser, P. M. Melliar-Smith, and P. Narasimhan. Consistent object replication in the eternal system. *Theory and Practice of Object Systems*, 4(2):81–92, 1998.

[110] Randolph D. Nelson and Balakrishna R. Iyer. Analysis of a replicated data base. *Performance Evaluation*, 5(3):133–148, 1985.

[111] Erich J. Neuhold and Bernd Walter. Overview of the architecture of the distributed data base system 'porel'. In *Distributed Data Bases: 2nd International Symposium*, pages 247–290, 1982.

[112] M. Nicola and M. Jarke. Performance modeling of distributed and replicated databases. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):645–672, 2000.

[113] Tariq Omari, Salem Derisavi, and Greg Franks. Deriving distribution of thread service time in layered queueing networks. In *WOSP'07*, Buenos Aires, Argentina, February 5-7 2007. ACM Press.

[114] Tariq Omari and Greg Franks. Performance analysis of replication in a mobile environment using layered queueing networks. In *6th Jordanian International Electrical and Electronics Engineering Conference*, Amman, Jordan, March 14-16 2006.

[115] Tariq Omari, Greg Franks, Murray Woodside, and Amy Pan. Efficient performance models for layered server systems with replicated servers and parallel behaviour. *(Elsevier) Journal of Systems and Software*, 80(4):510–527, April 2007.

[116] Tariq Omari, Greg Franks, Murray Woodside, and Amy M. Pan. Solving layered queueing networks of large client-server systems with symmetric replication. In *ACM Fifth International Workshop on Software and Performance (WOSP '05)*, pages 159–166, Palma de Mallorca, Spain, July 11-14 2005.

[117] Oracle. *Oracle8itm Advanced Replication*. Oracle technical white paper, Oracle Corporation, 1998.

[118] Amy M. Pan. Solving stochastic rendezvous networks of large client-server systems with symmetric replication. Master's thesis, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, September 1996.

[119] G. D. Parrington, S. K. Shrivastava, S. M. Wheater, and M. C. Little. The design and implementation of arjuna. *Computing Systems*, 8(3):255–308, 1995.

[120] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (raid). *SIGMOD Record*, 17(3):109–16, 1988.

[121] Carl E Pearson. *Numerical Methods in Engineering and Science*. Van Nostrand Reinhold Co., New York, 1986.

[122] D. C. Petriu and C. M. Woodside. Approximate mva from markov model of software client/server systems. In *Third IEEE Symposium on Parallel and Distributed Processing*, pages 322–329, Dallas, TX, USA, 1991. IEEE.

[123] D. C. Petriu and C. M. Woodside. Approximate mean value analysis based on markov chain aggregation by composition. *Linear Algebra and Its Applications*, 386:335–358, 2004.

[124] Dorina Petriu, Hoda Amer, Shikharesh Majumdar, and Istabrak Abdull-Fatah. Using analytic models predicting middleware performance. In *WOSP'00*, pages 189–194, Ottawa,Canada, 2000. Association for Computing Machinery.

[125] Dorina C. Petriu. *Approximate Solution for Stochastic Rendezvous Networks by Markov Chain Task-Directed Aggregation*. PhD thesis, Carleton University, 1991.

[126] Dorina C. Petriu. Approximate mean value analysis of client-server systems with multi-class requests. In *ACM Sigmetrics on Measurement and Modeling of Computer Systems*, volume 22, pages 77–86, Nashville, TN, USA, 1994.

[127] Dorina C. Petriu and Songtao Chen. Approximate mva for client-server systems with nonpreemptive priority. In *IEEE International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 155–162, Durham, NC, USA, 1995. IEEE.

[128] S. Pleisch and A. Schiper. Fatomas-a fault-tolerant mobile agent system based on the agent-dependent approach. In *International Conference on Dependable Systems and Networks*, pages 215–224, Goteborg, Sweden, 2001. IEEE Comput. Soc.

[129] K. Ranganathan, A. Iamnitchi, and I. Foster. Improving data availability through dynamic model-driven replication in large peer-to-peer communities. In *CCGRID 2002. 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 376–81, Berlin, Germany, 2002. IEEE Comput. Soc.

[130] Y. Raz. The principle of commitment ordering, or guaranteeing serializability in a heterogeneous environment of multiple autonomous resource managers using atomic commitment. In *the 18th International Conference on Very Large Data Bases*, pages 292–312, 1992.

[131] Jing Fei Ren, Yutaka Takahashi, and Toshiharu Hasegawa. Analysis of impact of network delay on multiversion conservative timestamp algorithms in ddbs. *Performance Evaluation*, 26(1):21–50, 1996.

[132] Yansong Ren, David E. Bakken, Tod Courtney, Michel Cukier, David A. Karr, Paul Rubel, Chetan Sabnis, William H. Sanders, Richard E. Schantz, and Mouna Seri. Aqua: An adaptive architecture that provides dependable distributed objects. *IEEE Transactions on Computers*, 52(1):31–50, 2003.

[133] J.A. Rolia and K.C. Sevcik. The method of layers. *IEEE Transactions on Software Engineering*, 21(8):689–700, 1995.

[134] Jerome Alexander Rolia. Performance estimates for systems with software servers: the lazy boss method. In *VIII SCCC International Conference on Computer Science*, pages 25–43, Santiago, Chile, 1988.

[135] Jerome Alexander Rolia. *Predicting the performance of software systems*. Phd thesis, University of Toronto, Toronto, Ontario, Canada. M5S 1A1, January 1992.

[136] Zuwang Ruan and Walter F. Tichy. Performance analysis of file replication schemes in distributed systems. In *1987 ACM SIGMETRICS Conf on Meas and Model of Comput Syst*, volume 15, pages 205–215, Banff, Alberta, Canada, 1987.

[137] R.A. Sahner and K.S. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, SE-13(10):1105–1114, October 1987.

[138] Hwang San-Yih, K. K. S. Lee, and Y. H. Chin. Data replication in a distributed system: a performance study. In *7th International Conference on Database and Expert Systems Applications (DEXA '96)*, pages 708–717, Zurich, Switzerland, 1996. Springer-Verlag.

[139] W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.

[140] Mahadev Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.

[141] Andre Schiper and Michel Raynal. From group communication to transactions in distributed systems. *Communications of the ACM*, 39(4):84–87, 1996.

[142] R. D. Schlichting and F. B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, 1983.

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

[144] M. J. Serrano. Performance estimation in a simultaneous multithreading processor. In *Fourth International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '96)*, pages 97–101, San Jose, CA, USA, 1996. IEEE.

[145] Fahim Sheikh, Jerome Rolia, Pankaj Garg, Svend Frolund, and Allan Shepherd. Performance evaluation of a large scale distributed application design. In *1st World Congress on Systems Simulation, Quality of Service Modelling*, pages 247–254, Singapore, 1997.

[146] Fahim Sheikh and Murray Woodside. Layered analytic performance modelling of a distributed database system. In *17th International Conference on Distributed Computing Systems*, pages 482–490, Baltimore, MD, USA, 1997.

[147] E. J. Shekita and M. J. Carey. Performance enhancement through replication in an object-orientated dbms. *SIGMOD Record*, 18(2):325–336, 1989.

[148] Chen Shenze and D. Towsley. A performance evaluation of raid architectures. *IEEE Transactions on Computers*, 45(10):1116–1130, 1996.

[149] Connie U Smith. *Performance Engineering of Software Systems*. The SEI Series in Software Engineering. Addison-Wesley Pub. Co., New York, 1990.

[150] C.U. Smith. *Software Performance Engineering, Encyclopedia of Software Engineering*. Wiley, 2002.

[151] S. H. Son and Zhang Fengjie. Real-time replication control for distributed database systems: algorithms and their performance. In *Fourth International Conference on Database Systems for Advanced Applications*, pages 214–221, Singapore, 1995. World Scientific.

[152] A. Sousa, F. Pedone, R. Oliveira, and F. Moura. Partial replication in the database state machine. In *IEEE International Symposium on Network Computing and Applications (NCA 2001)*, pages 298–309, Cambridge, MA, USA, 2001. IEEE Comput. Soc.

[153] B. Spitznagel and D. Garlan. Architecture-based performance analysis. In *Tenth International Conference on Software Engineering and Knowledge Engineering (SEKE '98)*, pages 146–151, San Francisco, CA, USA, June 1998. Knowledge System Institute.

[154] Ramesh Sridhar and H. G. Perros. A multilayer client-server queueing network model with synchronous and asynchronous messages. *IEEE Transactions on Software Engineering*, 26(11):1086–1100, 2000.

[155] Y. C. Tay. The availability of (k, n)-resilient distributed systems. In *3rd IEEE Symposium on Reliable Distributed Systems*, pages 119–122, Silver Spring, Maryland, 1984.

[156] Philip Teale, Christopher Etz, Michael Kiel, and Carsten Zeitz. *Data Patterns*. Microsoft Corporation., 1.0 edition, 2003.

[157] Douglas B. Terry, Karin Petersen, Mike J. Spreizer, and Marvin. M. Theimer. The case for non-transparent replication: Example from bayou. *IEEE Data Engineering*, 4(21):12–20, 1998.

[158] T. N. Vijaykumar, I. Pomeranz, and K. Cheng. Transient-fault recovery using simultaneous multithreading. In *29th Annual International Symposium on Computer Architecture*, pages 87–98, Anchorage, AK, USA, 2002. IEEE.

[159] James E. White. A high-level framework for network-based resource sharing. In *National Computer Conference*, New York, 1976. AFIPS Press.

[160] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Database replication techniques: a three parameter classification. In *19th IEEE Symposium on Reliable Distributed Systems SRDS-2000*, pages 206–215, Nurnberg, Germany, 2000. IEEE Comput. Soc.

[161] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *20th IEEE International Conference on Distributed Computing Systems*, pages 464–474, Taipei, Taiwan, 2000. IEEE.

[162] C. M. Woodside. Performance potential of communications interface processors. In *Proceedings of the Eighth Data Communications Symposium*, pages 245–253, North Falmouth, MA, USA, October 3–6 1983. Association for Computing Machinery.

[163] C. M. Woodside. An active-server model for the performance of parallel programs written using rendezvous. In *Proc. IFIP Workshop on Performance Evaluation of Parallel Systems*, Grenoble, December 13–15 1984.

[164] C. Murray Woodside. Throughput calculation for basic stochastic rendezvous networks. *Performance Evaluation*, 9:143–160, 1989.

[165] C.M. Woodside. *Replicated components in LQNS (Layered Queueing Network Solver)*. Technical report, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, 1995.

[166] C.M. Woodside, J.E. Neilson, D.C. Petriu, and S. Majumdar. The stochastic rendezvous network model for performance of synchronous client-server-like distributed software. *IEEE Transactions on Computers*, 44(1):20–34, 1995.

[167] C.M. Woodside, E. Neron, E. D. S. Ho, and B. Mondoux. *An 'active-server' model for the performance of parallel programs written using rendezvous*, volume 6. 1986.

[168] Murray Woodside, Greg Franks, and Dorina C. Petriu. The future of software performance engineering. In Lionel C. Briand and Alexander L. Wolf, editors, *29th Int. Conference on Software Engineering (ICSE 2007)*, pages 20–26, Minneapolis, MN, USA, May 2007. IEEE Computer Society Press.

[169] Xinfeng Ye and Y. Shen. Replicating multithreaded web services. *Parallel and Distributed Processing and Applications. Third International Symposium, ISPA 2005. Proceedings (Lecture Notes in Computer Science Vol.3758)*, pages 162–167, 2005.

[170] H. Yu and A. Vahdat. Building replicated internet services using tact: a toolkit for tunable availability and consistency tradeoffs. In *Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000)*, pages 75–84, Milpitas, CA, USA, 2000. IEEE Comput. Soc.

[171] Tao Zheng and C. Murray Woodside. Heuristic optimization of scheduling and allocation for distributed systems with soft deadlines. *Computer Performance Evaluation / TOOLS 2003*, pages 169–181, 2003.

[172] Tao Zheng and Murray Woodside. Fast estimation of probabilities of soft deadline misses in layered software performance models. In *5th Int. Workshop on Software and Performance (WOSP 2005)*, pages 1181–1186, July 2005.

[173] Wanlei Zhou, Li Wang, and Weijia Jia. An analysis of update ordering in distributed replication systems. *Future Generation Computer Systems*, 20(4):565–590, 2004.

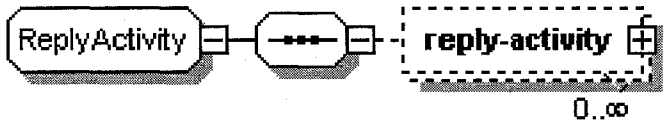
APPENDIX A

New LQN Input File XML Schema

This appendix gives the formal description of the LQN input file grammar using XML. The modifications were applied to the LQN core XML schema definition file (lqn-core.xsd). In this appendix, only the relevant modifications are shown. The LQN user manual **[52]** has detailed information about the LQN XML schema.

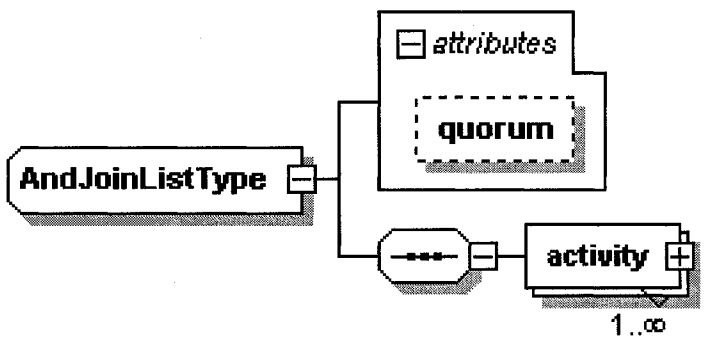
```
<lqn-model>
<solver-params>
        <pragma/>
</solver-params>
<processor>
        <task>
                <entry>
                        <entry-phase-activities>
                                <activity>
                                        <synch-call/>
                                        <asynch-call/>
                                </activity>
                                <activity>  ... </activity>
                        </entry-phase-activities>
                </entry>
                <entry>     ... </entry>
                <task-activities>
                        <activity/>
                        <precedence/>
                </task-activities>
        </task>
        <task>      ... </task>
</processor>
<processor> ... </processor>
</lqn-model>
```

Figure A.1: XML input file layout.



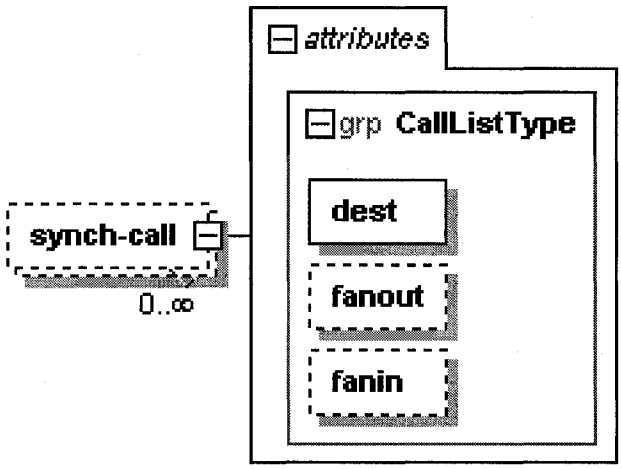

```
<xsd:group name="ReplyActivity">
 <xsd:sequence>
  <xsd:element name="reply-activity" minOccurs="0" maxOccurs="unbounded">
            <xsd:complexType>
            <xsd:attributename="name" type="xsd:string" use="required"/>
                    </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:group>
```

Figure A.2: XML schema for Reply Activity.




```
<xsd:complexType name="AndJoinListType">
 <xsd:sequence>
  <xsd:element name="activity" type="ActivityType" maxOccurs="unbounded"/>
 </xsd:sequence>
 <xsd:attribute name="quorum" type="xsd:nonNegativeInteger" default="0"/>
</xsd:complexType>
```

Figure A.3: XML schema for the quorum pattern.




```
<xsd:attributeGroup name="CallListType">
        <xsd:attribute name="dest" type="xsd:string" use="required"/>
        <xsd:attribute name="fanout" type="xsd:int"/>
        <xsd:attribute name="fanin" type="xsd:int"/>
</xsd:attributeGroup>
<xsd:element name="synch-call" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                <xsd:attributeGroup ref="CallListType"/>
        </xsd:complexType>
</xsd:element>
```

Figure A.4: XML schema for the compact notation for replication with internal parallelism.