PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

Evaluation of Approximation for Response Time of Parallel Task Graph Model

by

Xianghong Jiang

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

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

April 10, 1996

**© copyright
1996, Xianghong Jiang**

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-612-13842-9

Name Xiang-hong Jiang

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronic & Electrical Engineering

SUBJECT TERM

0544

U·M·I

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture	0379
Art History	0377
Cinema	0300
Dance	0378
Fine Arts	0357
Information Science	0723
Jouralism	0371
Library Science	0399
Mass Communications	0708
Music	0413
Speech Communication	0459
Theater	0465

EDUCATION

General	0515
Administration	0514
Adult and Continuing	0516
Agricultural	0517
Art	0273
Bilingual and Multicultural	0282
Business	0688
Community College	0275
Curriculum and Instruction	0277
Early Childhood	0518
Elementary	0524
Finance	0277
Guidance and Counseling	0519
Health	0680
Higher	0745
History of	0520
Home Economics	0278
Industrial	0521
Language and Literature	0279
Mathematics	0280
Music	0522
Philosophy of	0598
Physical	0523

Psychology	0525
Reading	0535
Religion	0527
Science	0514
Secondary	0533
Social Sciences	0534
Sociology of	0340
Special	0529
Teacher Training	0530
Technology	0710
Tests and Measurements	0288
Vocational	0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language	
General	0679
Ancient	0289
Linguistics	0290
Modern	0291
Literature	
General	0401
Classical	0294
Comparative	0295
Medieval	0297
Modern	0298
African	0316
American	0591
Asian	0305
Canadian (English)	0352
Canadian (French)	0355
English	0593
Germanic	0311
Latin American	0312
Middle Eastern	0315
Romance	0313
Slavic and East European	0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy	0422
Religion	
General	0318
Biblical Studies	0321
Clergy	0319
History of	0320
Philosophy of	0322
Theology	0469

SOCIAL SCIENCES

American Studies	0323
Anthropology	
Archaeology	0324
Cultural	0326
Physical	0327
Business Administration	
General	0310
Accounting	0272
Banking	0770
Management	0454
Marketing	0338
Canadian Studies	0385
Economics	
General	0501
Agricultural	0503
Commerce-Business	0505
Finance	0508
History	0509
Labor	0510
Theory	0511
Folklore	0358
Geography	0366
Gerontology	0351
History	
General	0578

Ancient	0579
Medieval	0581
Modern	0582
Black	0328
African	0331
Asia, Australia and Oceania	0342
Canada	0334
European	0335
Latin American	0336
Middle Eastern	0333
United States	0337
History of Science	0585
Law	0398
Political Science	
General	0615
International Law and Relations	0616
Public Administration	0617
Recreation	0814
Social Work	0452
Sociology	
General	0626
Criminology and Penology	0627
Demography	0938
Ethnic and Racial Studies	0631
Individual and Family Studies	0628
Industrial and Labor Relations	0629
Public and Social Welfare	0630
Social Structure and Development	0700
Theory and Methods	0344
Transportation	0709
Urban and Regional Planning	0999
Women's Studies	0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture	
General	0473
Agronomy	0285
Animal Culture and Nutrition	0475
Animal Pathology	0476
Food Science and Technology	0359
Forestry and Wildlife	0478
Plant Culture	0479
Plant Pathology	0480
Plant Physiology	0817
Range Management	0777
Wood Technology	0746
Biology	
General	0306
Anatomy	0287
Biostatistics	0308
Botany	0309
Cell	0379
Ecology	0329
Entomology	0353
Genetics	0369
Limnology	0793
Microbiology	0410
Molecular	0307
Neuroscience	03
Oceanography	0416
Physiology	0433
Radiation	0821
Veterinary Science	0778
Zoology	0472
Biophysics	
General	0786
Medical	0760
EARTH SCIENCES	
Biogeochemistry	0425
Geochemistry	0996

Geodesy	0370
Geology	0372
Geophysics	0373
Hydrology	0388
Mineralogy	0411
Paleobotany	0345
Paleocology	0426
Paleontology	0418
Paleozoology	0985
Polynology	0427
Physical Geography	0368
Physical Oceanography	0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences	0768
Health Sciences	
General	0566
Audiology	0300
Chemotherapy	0992
Dentistry	0567
Education	0330
Hospital Management	0769
Human Development	0758
Immunology	0982
Medicine and Surgery	0564
Mental Health	0347
Nursing	0569
Nutrition	0570
Obstetrics and Gynecology	0380
Occupational Health and Therapy	0354
Ophthalmology	0381
Pathology	0571
Pharmacology	0419
Pharmacy	0572
Physical Therapy	0382
Public Health	0573
Radiology	0574
Recreation	0575

Speech Pathology	0460
Toxicology	0383
Home Economics	0386

PHYSICAL SCIENCES

Pure Sciences	
Chemistry	
General	0485
Agricultural	0749
Analytical	0486
Biochemistry	0487
Inorganic	0488
Nuclear	0738
Organic	0490
Pharmaceutical	0491
Physical	0494
Polymer	0495
Radiation	0754
Mathematics	0405
Physics	
General	0605
Acoustics	0986
Astronomy and Astrophysics	0606
Atmospheric Sciences	0608
Atomic	0748
Electronics and Electricity	0607
Elementary Particles and High Energy	0798
Fluid and Plasma	0759
Molecular	0609
Nuclear	0610
Optics	0752
Radiation	0756
Solid State	0611
Statistics	0463
Applied Sciences	
Applied Mechanics	0346
Computer Science	0984

Engineering	
General	0537
Aerospace	0538
Agricultural	0539
Automotive	0540
Biomedical	0541
Chemical	0542
Civil	0543
Electronics and Electrical	0544
Heat and Thermodynamics	0548
Hydraulic	0545
Industrial	0546
Marine	0547
Materials Science	0794
Mechanical	0548
Metallurgy	0743
Mining	0551
Nuclear	0552
Packaging	0549
Petroleum	0765
Sanitary and Municipal	0554
System Science	0790
Geotechnology	0428
Operations Research	0796
Plastics Technology	0795
Textile Technology	0994

PSYCHOLOGY

General	0421
Behavioral	0384
Clinical	0422
Developmental	0420
Experimental	0423
Industrial	0424
Personality	0425
Physiological	0426
Psychobiology	0429
Psychometrics	0432
Social	0431

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

**Evaluation of Approximation for Response
Time of Parallel Task Graph Model**

**Submitted by Xianghong Jiang, B. Eng.
in partial fulfillment of the requirements for
the degree of Master of Engineering**

Professor C. M. Woodside, Thesis Supervisor

Chair, Department of Systems and Computer Engineering

**Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University
April 10, 1996**

Abstract

Parallel processing has been widely used to implement highly complex computation or deal with real-time tasks when the requirements for system performance are critical. Response time is an important performance measure for systems with parallel processing. An approximate method to calculate the response time of delay of a parallel (Fork-join) segment of computation based on the task graph model is described.

In general, calculating the response time of Fork-join segment requires full information about the probability distribution of each path, which is difficult to obtain. It is much easier to obtain the mean and variance, from measurement. Therefore approximation must be used to calculating the response time for Fork-join segment.

The approximation method described in this thesis is fast but not always accurate. It only needs the mean and variance of each branch of a Fork-join segment, it approximates the distribution by three point values, and estimate the mean and variance of the total delay of the segment.

An automatic system is developed to generate many instances of task graphs for systematically evaluating the approximation. The size of the task graph generated by the system can be determined by user input arguments.

The approximation is fully evaluated by being used for calculating response time for different type of complex task graphs. Resource contention scenario are also studied. The results of the approximation compared closely to the results obtained from detailed simulation. It shows that the approximation is more accurate for Fork-join with small variance, with mean absolute error less than 1% over 100 cases with CV (coefficient of variation) ≤ 1 . For $CV > 1$, the mean absolute error over 100 cases is less than 4%. The approximation is good for solving performance measure for complex graphs with parallel path.

Acknowledgments

I would like to thank my supervisor, Professor Murray Woodside for his guidance and critique throughout the studies.

I wish to thank my husband Jeffrey for his support and understanding during the work.

I also would like to thank my sister Hua, for her encouragement during the year from Soochow.

Table of Contents

Abstract.....	iii
Acknowledgments	iv
Table of Contents	v
List of Figure	viii
List of Table.....	x
Chapter 1 Introduction.....	1
1.1 Perspective of the Research.....	1
1.2 Topic of the Thesis	5
1.3 Contributions.....	6
1.4 Thesis Organization.....	7
Chapter 2 Overview of Task Graph Notation	8
2.1 Task Graphs.....	8
2.1.1 Task Graph Models	8
2.1.2 Complete Theoretical Solution for Fork-join Subgraph	13
2.2 Tree Structure Notation of Task Graph	15
2.3 Previous Research on Task Graph Performance	17
Chapter 3 The Three-point Approximation	20

3.1	The Approximation Method.....	20
3.2	A Few Examples	23
3.2.1	Simple Fork-join Cases	23
3.2.2	Small Tractable Cases	39
3.3	Three-point Approximation Summary	43

Chapter 4 A Simulation Model 44

4.1	Objectives of Simulation.....	44
4.2	Detail of Setting Up a Simulation Model.....	44
4.2.1	The Overall Concept of the Simulation Model	44
4.2.2	The Algorithm & Parameters of the Simulation Model.....	46
4.3	Accuracy of the Simulation.....	47
4.4	Summary of Simulation	49

Chapter 5 A Framework for System Evaluation 50

5.1	Graph Parameters	50
5.2	Task Graph Creation.....	52
5.3	Output the Parameter File	55
5.4	Summary	61

Chapter 6 Evaluation of Approximation 62

6.1	Different Classes of Cases Under Testing.....	62
6.2	Results Analysis	64
6.2.1	Exponential Distribution	64
6.2.2	Uniform Distributions	66

6.2.3	Erlangian Distributions	67
6.2.4	Hyperexponential Distribution	68
6.2.5	Across the Four Groups.....	71
6.3	Mixed-Distribution Delay Cases.....	71
6.4	Source of Large Errors	73

Chapter 7 Cases With Contention..... 80

7.1	The Waiting Time Without Logical Resource Contention	80
7.2	Task Response Time Model With Logical Resource Contention.....	83
7.3	Simulation Model.....	87
7.4	Model Validation	89
7.4.1	The Example Case for Contention Study	89
7.4.2	Results Without Resource Contention	90
7.4.3	Results With Resource Contention.....	94
7.5	Summary	99

Chapter 8 Conclusion 100

8.1	Summary of the Results	100
8.2	Future Work.....	101

References..... 102

Appendix..... 106

List of Figure

Figure 2-1	Sequential Subgraph	10
Figure 2-2	Alternate Subgraph	11
Figure 2-3	Fork-join Subgraph	12
Figure 2-4	The Example of Task Graph Notation	16
Figure 2-5	The Example of Tree Structure Notation	16
Figure 3-1	Three-point Approximation: pdf of Three Point Values	21
Figure 3-2	Three-point Approximation: PDF of Three Point Approximation	22
Figure 3-3	The Histogram of Errors of 80 Fork-join Cases with $CV \leq 1$	27
Figure 3-4	The Errors of Mean with kernel Value	28
Figure 3-4	(continue.) The Errors of Mean with kernel Value	29
Figure 3-5	The Errors of Mean with Branch Value N	30
Figure 3-5	(continue.) The Errors of Mean with Branch Value N	31
Figure 3-6	The Errors of Mean with Range Value	32
Figure 3-6	(continue.) The Errors of Mean with Range Value	33
Figure 3-7	The Histogram of Errors of 80 fork-join Cases with $CV > 1$	37
Figure 3-8	Small Task Graph of Case5	41
Figure 6-1	Histogram of Errors of Mean for 100 Random Generated Task Graph with Exponential Distribution	64
Figure 6-2	Histogram of Errors of Mean for 100 Random Generated Task Graph with Uniform Distribution	66
Figure 6-3	Histogram of Errors of Mean for 100 Random Generated Task Graph with Erlangian Distribution	67
Figure 6-4	Histogram of Errors of Mean for 100 Hyperexponential Cases with $CV = 2$	68

Figure 6-5	Histogram of Errors of Mean for 100 Hyperexponential Cases with $CV = 3$.	69
Figure 6-6	Histogram of Errors of Mean for 100 Hyperexponential Cases with $CV = 4$.	70
Figure 6-7	Histogram of errors of 100 cases with mixed distribution of delay	73
Figure 7-1	Average Task Response Time for Task Assignment 1	90
Figure 7-2	Average Task Response Time for Task Assignment 2	92
Figure 7-3	Average Task Response Time for Task Assignment 1 With Resource Contention	95
Figure 7-4	Average Task Response Time for Task Assignment 2 With Resource Contention	97

List of Table

Table 3.1:	The parameters of the 80 cases in the first set ($CV \leq 1$)	25
Table 3.2:	The percentage error of 80 fork-join cases with small $CV \leq 1$	34
Table 3.3:	The parameters of the 80 cases in the second set ($CV > 1$).....	35
Table 3.4:	The parameters of 22 cases in the third set	38
Table 3.5:	Mean delay of branches of case1	39
Table 3.6:	Mean delay of branches of case2	40
Table 3.7:	Mean delay of tasks of case4	40
Table 3.8:	Mean delay of tasks of case5	40
Table 3.9:	The comparison results of 5 small tractable task graphs	42
Table 4.1:	The accuracy of the simulation of 16 cases with Fork-join task graphs of exponential distribution	48
Table 4.2:	Simulation error for five tractable cases	48
Table 5.1:	Definition of simulation parameters	50
Table 5.2:	The parameters of the distribution	51
Table 5.3:	Definition of approximation parameters for composite nodes.....	52
Table 5.4:	Definition of approximation parameters for atomic nodes	52
Table 5.5:	The statistics of the graph	60
Table 6.1:	The structure properties of the 100 cases.....	63
Table 6.2:	The structure properties of 100 cases with mixture distribution.....	72
Table 6.3:	Statistics of the structure of the 100 graphs	75
Table 7.1:	Task execution times and assignment for the contention example.....	89
Table 7.2:	Response time for task assignment 1	91
Table 7.3:	The comparison of waiting delay for task assignment 1.....	91
Table 7.4:	Response time for task assignment 2	93

Table 7.5:	The comparison of waiting delay for task assignment 2 (unbalanced).....	93
Table 7.6:	Response time for task assignment 1 with resource contention.....	95
Table 7.7:	The comparison of waiting delay for task assignment 1 with resource contention.....	96
Table 7.8:	Response time for task assignment 2 with resource contention.....	97
Table 7.9:	The comparison of waiting delay for task assignment 2 with resource contention.....	98
Table A.1:	Compare results of 80 Fork-join cases with small CV value ($CV \leq 1$) ...	102
Table A.2:	Compare results of 80 Fork-join cases with large CV value ($CV > 1$)....	105
Table A.3:	Compare results of 22 Fork-join cases with mixed distribution	108
Table A.4:	Compare results of 100 cases in exponential group.....	109
Table A.5:	Compare results of 100 cases in uniform group	113
Table A.6:	Compare results of 100 cases in Erlangian group.....	117
Table A.7:	Compare results of 100 cases in hyperexponential group with $CV = 2$...	121
Table A.8:	Compare results of 100 cases in hyperexponential group with $CV = 3$...	125
Table A.9:	Compare results of 100 cases in hyperexponential group with $CV = 4$...	129
Table A.10:	Compare results of 100 cases of mixed distrubution	133

Chapter 1 Introduction

1.1 Perspective of the Research

Parallel processing has been widely used to implement complex computation or deal with real-time tasks when the requirements for system performance are critical. The high performance of parallel processing is usually achieved by partitioning a complex software task into a set of small executable software sub-tasks which can be executed concurrently and cooperatively on a hardware system with parallel multiprocessor architecture [10].

The system performance gain or speed-up by parallel processing depends on the parallel algorithm design, software task partitioning, level of concurrency of software execution, and the underlying multiprocessor architecture. The overall system performance also depends on interprocessor communication, synchronization among the sub-tasks and utilization of the physical hardware resources. There has been much previous research work conducted in the area of parallel software algorithm design and hardware organization [7,8].

One of the most important aspects of parallel processing system design is performance modeling and evaluation. Performance is a key criterion in the design, procurement, and use of computer systems. Computer system users, administrators, and designers are all interested in getting the highest performance of their systems for a given cost. Ensuring a software system to meet its performance objectives presents a significant challenge. This challenge becomes even more difficult to resolve for concurrent and/or distributed software systems. In concurrent system research, prediction of parallel computations performance is also a useful tool. Software performance engineering prevents and solves performance problems by analyzing a software system and suggesting corrective measures when necessary.

In general, performance refers to system responsiveness: either the time required to respond to specific events, or the number of events processed in a given time interval. For traditional information systems, performance considerations are associated with usability issues such as response time for user commands. For “soft” real-time systems, performance considerations usually described by statistical values of response time. In “hard” real-time systems, performance concerns can become correctness issues. In these systems, failure to meet deadlines or throughput requirements is equivalent to producing incorrect results.

The performance measures of parallel processing include the following criteria:

- the expected execution time or response time of parallel computation.
- average utilization of the physical resources in the parallel system.
- average throughput of the parallel system.

Because of the concurrent nature of parallel processing, concurrent sub-tasks are assigned to different hardware processors, and there may exist complex relationships among the sub-tasks at the time of execution. Besides the actual service time spent on system resources, execution time of a parallel computation consists of two kinds of additional delay: queuing delay and synchronization delay. Queuing delay results when two or more sub-tasks compete for resources in the system. Synchronization delay results when a task has to idle and wait for others to finish before continuing. Because of the presence of queuing and synchronization delays, execution times of parallel computations are very difficult to predict. Therefore, it is difficult and challenging to accurately model and analyze the performance of parallel processing systems.

Currently available performance analysis methods for parallel processing systems fall into the following three general categories: *simulation*, *real measurement*, and *approximate analytical approach*.

Methods of the first category employ simulation tools to construct and run a

detailed model of a system. Usually general-purpose simulation packages are used, which have built-in facilities for gathering and analyzing performance statistics. Most of such simulation packages are very expensive to use, in terms of the consumption of computer resources and the required model development effort.

The second category of performance analysis technique is based on measurement. It is the most costly and difficult among the analysis techniques since it requires a real system, real software, real users and more time. Results produced can only be used to patch the system, and significant changes to an installed system are not cheap. It is also hard to use this approach when a new system is under design.

The third category of performance analysis methods is comprised of approximate analytic techniques based on modeling [13,14,18,25]. It is especially appropriate to be used for performance prediction. The performance model's structure is defined, the behavior is characterized, and parameters are assigned to the model structure. This technique is used to evaluate and provide performance estimates for the analysis. Compared to the measurement and simulation, analytic modeling requires the least time and the least cost, and it is less laborious and more flexible. These methods employ either standard queuing network (QN for short) models or graph models such as task graphs, which trace system states during program execution.

The queuing network model is widely used in performance analysis of computer systems because it has many advantages [13,14,15,31]. It provides both modeling power and efficient solutions. It is easy to build, highly abstract and requires a minimum of information for its parameters. However, QN modeling has only limited ability to represent synchronization directly and in modeling parallel software. The techniques based only on queuing networks, such as the ones proposed in [13], suffer from the inability to explicitly represent complex interdependencies of tasks in a program except very simple task structure.

With most of the techniques based purely on graph models, such as Stochastic Petri Nets, the model of program behavior is intimately linked with the model of the execution environment. Thus, a minor architectural change may require a new model to be constructed and solved. Another limitation of most graph-based methods is that their state space size usually grows combinatorially with respect to the size of the particular configuration being modeled. The techniques presented recently in [29] allow for efficient solution of acyclic graph models of program behavior. Although those techniques permit complex distributions of individual task execution times, they lack facilities for explicitly representing the architecture of the execution environment and the contention for system resources.

Therefore, some research work has been conducted to combine the QN model with the task graph model to solve the performance analysis problem [6,22]. The QN submodel is used for approximating sub-task response time. Resources in a concurrent system are modeled as service centers in the QN submodel. The QN submodel can be solved by decomposition technique, which will reduce the computation complexity. The task graph submodel is used to represent the parallel task system with precedence relationships. The response time of task system can be obtained by aggregating sub-task response times in accordance with the precedence relationships. This gives the advantage of an accurate and efficient method for predicting performance of parallel computations running on concurrent systems.

When solving a task graph model, distribution knowledge is needed for mathematically solving the Fork-join structure. When the Fork-join subgraph contains more than three parallel branches, the exact solution is difficult to determine accurately. In [6], an approximation was proposed to aggregate every two branches at one time.

The research work in this thesis attempts to give an even more efficient way in aggregating a Fork-join subgraph, resulting in an efficient way of calculating the response

time of task graph.

1.2 Topic of the Thesis

The topic of this thesis is system performance modeling, evaluation and analysis for parallel processing. Specifically, the thesis describes and evaluates an approximation method to calculate the mean and variance of delay of a parallel segment of computation based on the task graph model. The proposed approximation method suggested by Professor Woodside and Alex Hubbard, will be called a **three-point** approximation. The computational complexity and the memory requirement of this approach is much lower than the exact mathematical methods so that it is more practical and efficient to be used in the real world.

Compared to other approaches, this approximation method is simpler but not always accurate. Various aspects of the accuracy of the approximation method will be discussed throughout the thesis.

Other approaches need to make certain assumption for the distribution of delay of the individual sub-task. In most cases it is assumed to be an exponential distribution, which may seriously overestimate the waiting time [2]. However, the proposed **three-point** approximation method doesn't need such an assumption. Instead, it assumes a discrete distribution based on the mean and variance of the delay of each sub-task in the task graph, which can be obtained easily from measurement.

The accuracy of the **three-point** approximation depends on the relationship of the standard deviation to the mean of each branch, which is given by the Coefficient of Variation (CV):

$$CV = \frac{\sigma}{t}$$

In this thesis, a simulation model based on the PARASOL [23,24] system is established to evaluate the accuracy of the proposed approximation. The approximation results will be compared with simulation results of different distributions in order to evaluate the impact of variance value of the delay on the approximation.

The approximation accuracy is calculated for the delay to traverse two types of graph:

- simple Fork-join subgraphs with two to twenty branches,
- complex task graphs with nested constructs and dozens or hundreds of tasks.

The approximation will be used to calculate mean and variance of response time of the task graph in which the sub-tasks have different *CV* value of delays, in order to evaluate the impact of *CV* value on the approximation.

We also study different scenarios with resource contention, which introduces the extra delay into the task system. The results of the approximation will be compared with the values obtained from the simulation to validate the accuracy of the proposed approach.

1.3 Contributions

The main contribution of this thesis is to evaluate *an approximation method* to calculate the response time of a segment of computation in parallel task graph. Other contributions that enable the main contribution or make it easy to evaluate the accuracy of the main contribution when used for different scenarios are:

- *Development of an automatic graph-generating system* which can generate different instances of task graph according to specified parameters.
- *Development of a simulation model* which is used to evaluate the accuracy of the approximation method.

- *A model to calculate the waiting delay* when contention exists.

The significance of these contributions are:

- Validate the approximation for different scenarios.
- Make the approximation as the basis for other performance prediction.

1.4 Thesis Organization

This thesis is organized as follows.

Chapter 2, the definition and general description of the task graph model are presented. The previous research work based on this model is surveyed.

Chapter 3 describes the details of the three-point approximation method. We also provide a few examples to explain the general ideas behind this approach.

Chapter 4 explains the simulation models which are used to compare our analysis results.

Chapter 5 deals with the framework for system evaluation. The algorithm to generate various task graphs for different scenarios is described

Chapter 6 contains the analysis of the evaluation results for multi-level graphs with different distribution of delay time.

Chapter 7 discusses various resource contention scenarios, presenting a new model for calculating the resource contention delay. We also discuss the modification of the simulation model for these scenarios and the analysis of the results.

Chapter 8 concludes the research work and present the potential future research directions.

Chapter 2 Overview of Task Graph Notation

2.1 Task Graphs

Task graph models have been used extensively to model parallel computation with precedence relationships [6,7,21,22,31]. The nodes in the graph represent the tasks and the arcs represent the precedence relationships. The response time of the task graph can be obtained by aggregating the weighted graph model in accordance with the precedence and logical relationships among the tasks.

2.1.1 Task Graph Models

This section defines the notations for the task graph model and some delay equations used in the analysis.

A parallel computation is viewed as a collection of tasks with constraints on when a task can be executed relative to the other tasks. By definition, a task is a unit of computation that, once started, runs to completion without interruption and interacts with other tasks only at the beginning and the end. The tasks and their constraints are often represented in a task graph with nodes representing tasks and directed arcs representing precedence relationship. Tasks without predecessors are called *initial tasks* and tasks that are not the predecessors of any task are called *final tasks*. A task cannot start until all its predecessor tasks have completed execution. The execution time for the computation, also known as the response time of the task system, is the time from the execution start point to completion of all final tasks. The execution time consists of *task waiting times*, *task execution demands*, and *precedence waiting times* (synchronization delays). Task waiting time is the time from completion of all of a task's predecessors, that is it is enabled to execute, until it starts its execution on a processor. The waiting time is the time spent waiting for other task executions. Task execution time is the time for the task to be executed

on the processor. We refer to the sum of a task's waiting time and execution time as its response time, and the precedence waiting time as the inter-task synchronization delay due to the precedence relationships among tasks.

Any parallel computation can be expressed by a task graph. A task graph can be decomposed into four elementary types of subgraphs:

- sequential subgraph
- Fork-join subgraph (And-fork to And-join)
- alternate subgraph (Or-fork to Or-join)
- loop subgraph [6].

This thesis does not cover loop subgraphs. Because any computation containing a loop can be unrolled and different instances of the loop body can be represented by different sets of tasks which contain the previously mentioned subgraphs, so that cyclic graphs can be converted to acyclic graphs.

For each individual task, let x_i be its name, and its execution demand be t_{x_i} , the mean of execution demand $\bar{t}_{x_i}$ and the variance of execution demand $\sigma_{x_i}^2$.

For a subgraph or composite task, let y be its name. Its delay is represent by t_y , the mean of delay by $\bar{t}_y$, and the variance of delay by σ_y^2 .

Figure 2-1 Sequential Subgraph

- Sequential subgraph: This (see example in Figure 2-1) is a sequence of tasks connected in series in which each task (except the last one) has a single successor. Tasks execute in the sequence indicated by the arcs. The delay of each task is an independent random variable, and the total delay of the sequential subgraph is the sum of all the delay of tasks. So the mean of delay of the subgraph is the sum of the mean delay of each branch, the variance of delay of the subgraph is the sum of variance of delay of each branch. The mean and variance of the total delay of the subgraph are given by:

mean:
$$\bar{t}_y = \sum_{i=1}^n \bar{t}_{xi} \quad (2.1)$$

variance:
$$\sigma_y^2 = \sum_{i=1}^n \sigma_{xi}^2 \quad (2.2)$$

Figure 2-2 Alternate Subgraph

- **Alternate subgraph:** This subgraph consists of an or-fork to or-join as depicted in Figure 2-2. There are several branches to be executed based on selection criteria; the task enables one of its succeeding tasks and thus one of the several branches, with probabilities as shown. The delay of this subgraph is the sum of the weighted delay of all the tasks. The mean and variance of the total delay of the subgraph are given by:

mean:
$$\bar{t}_y = \sum_{i=1}^n p_i \bar{t}_{xi} \quad (2.3)$$

variance:
$$\sigma_y^2 = \sum_{i=1}^n p_i \sigma_{xi}^2 \quad (2.4)$$

Figure 2-3 Fork-join Subgraph

- Fork-Join subgraph: This subgraph begins from a task which simultaneously enables several succeeding tasks, and ends at a task which is enabled only when all of its preceding tasks have completed their execution as shown in Figure 2-3. This subgraph may correspond to the synchronous and partition parallel computation types, which require concurrent processing. The total delay of the Fork-join subgraph is the maximum delay of its branches:

$$t_y = \text{MAX}_i \{t_{xi}\} \quad i = 1, 2, \dots, n$$

$\bar{t}_y$ and σ_y^2 can be calculated approximately.

Computing the delay of this subgraph requires the knowledge of the PDFs for each branch, which is rather complicated. Next section will describe the theoretical solution to finding the distribution of the total delay of the Fork-join subgraph such that the first two moments can be obtained using the probability distribution function.

A complex task graph can be aggregated into layered simple subgraphs by using an iterative algorithm. Each of these subgraphs can be considered as a single "task". If the single task is actually a subgraph of a certain type, it is called a *composite task*, as enclosed by dotted line in the figures. The branches of the subgraph are called *child tasks* of the composite task, and the composite task is *parent task* of its children tasks inversely. If a single task is an individual task, it is called an *atomic task*. Every branch of a composite task can also be considered as a single task in the graph. Such successive graph reductions at last yield one top level composite task for the entire task graph and obtain the estimation of the response time of the entire task graph. This thesis uses the mean and variance for each task and each composite task to obtain the mean and variance of the top-level composite task, respectively, thus obtain the response time of the entire task graph.

2.1.2 Complete Theoretical Solution for Fork-join Subgraph

In this section, the complete theoretical solution to calculate the response time of Fork-join subgraph using distribution function knowledge is described.

For each branch i in a Fork-join subgraph in Figure 2-3,

- $f_{xi}(t)$ ---probability density function of its delay (pdf)
- $F_{xi}(t)$ ---probability distribution function of its delay (PDF)

For all branches combined,

- $f_y(t)$ ---pdf of total delay
- $F_y(t)$ ---PDF of total delay

The relationship between pdf and PDF is:

$$f(t) = \frac{dF(t)}{dt} \quad (2.5)$$

The mean and variance of a random variable with pdf $f(t)$ is:

$$\bar{t} = \int_0^{\infty} tf(t) dt \quad (2.6)$$

$$\sigma^2(t) = \bar{t}^2 - (\bar{t})^2 = \int_0^{\infty} t^2 f(t) dt - (\bar{t})^2 \quad (2.7)$$

The rate of $\sigma/\bar{t}$ is defined as the *Coefficient of Variation (CV)* of the distribution.

The complete analysis breaks down as follows:

a) For each branch, as pdf $f_{xi}(t)$ is known, find out the PDF of delay for each branch;

it is the probability of branch i to be completed in time t :

$$F_{xi}(t) = \int_0^t f_{xi}(\tau) d\tau \quad (2.8)$$

b) Because only when all branches complete does it mean the completion of the total subgraph,

Prob (completion of Fork-join in time t) = Prob (all branches complete in time t)

$$= \prod_{i=1}^n \text{Prob}(\text{branch } i \text{ has completed in time } t)$$

therefore

$$F_y(t) = \prod_{i=1}^n F_{xi}(t) \quad (2.9)$$

c) Calculate the pdf of the whole subgraph

$$f_y(t) = \frac{dF_y(t)}{dt} \quad (2.10)$$

d) Get the mean and variance for the whole subgraph

$$\bar{t}_y = \int_0^{\infty} t f_y(t) dt \quad (2.11)$$

$$\sigma_y^2 = \int_0^{\infty} t^2 f_y(t) dt - (\bar{t}_y)^2 \quad (2.12)$$

2.2 Tree Structure Notation of Task Graph

The task graph shows flow control. When dealing with aggregating a general task graph, we use a more convenient tree structure notation to represent the relationships between a composite task and its children tasks. In the tree notation, each composite task is represented by a composite node with a label describing how it is decomposed ("sequential", "alternative", "Fork-join"). Its child tasks are *child nodes* of this composite node. The composite node is conversely called the *parent node* of its children nodes. Children nodes of the same parent node are called *sibling nodes* to each other. If the composite task's child task is also a composite task itself, then it can have its own child nodes. If its child task is an atomic task (that is, it has its own distribution, mean and variance, and does not have any child task), it is a leaf node in the tree structure and has the label "atomic". A composite node in the tree structure will be represented by a rectangle, while an atomic node in the tree structure will be represented by an ellipse.

An example of task graph and its corresponding tree structure notation is given in Figure 2-4 and Figure 2-5.

Figure 2-4 The Example of Task Graph Notation

Figure 2-5 The Example of Tree Structure Notation

The tree structure is essential to organize the computation of delay. The delay values of the task graph can be attached to the layered tree structure; the mean and variance of each composite node's delay can be obtained from its children nodes' means and variances. Thus from the bottom level up, the response time of the whole task graph -- the root composite node -- is obtained.

Chapter 5 describes parameters of the tree structure notation and the task graph notation which are used for approximation and simulation purpose respectively.

2.3 Previous Research on Task Graph Performance

Several performance models have been proposed for analysis of parallel task systems. Queuing networks were commonly used to model concurrent operations and to study resource constraints without precedence relationships. Including precedence relationships among tasks, such as "forks" and "joins", in queuing network resulted in intractable models which must be approximated.

As early as in 1966, Richard Karp and Raymond Miller [18] established a graph-theoretical model for the description and analysis of parallel computation. The concept is to represent the sequencing of a parallel computation by a finite directed graph, to associate each step of the algorithm with a node, and to specify the transmission paths for data and control information by branches.

In [27], a task graph is used to describe a static decomposition algorithm for an asynchronous multiprocessor. With known probability distribution of each task in the task graph, the probability distribution of total execution time can be obtained. Even if the probability distribution of each task's execution time is unknown, it is still possible to bound mean total execution times when the means and variances of task execution times are known.

Heidelberger and Trivedi [12,13] used analytic queuing models to predict

performance for programs with internal concurrency. In [12], they developed an approximate method to model programs in which a parent task spawns off two or more statistically identical asynchronous child tasks. In another paper [13], they developed two approximate methods to model the case in which the parent task, after spawning off the children tasks, waits for their completion before continuing (just one level of fork-join). Both papers considered only very simple task structures. Their methods are not directly applicable to parallel computations with more general task structures.

In [30,31], Thomasian and Bay considered a more general task structure with precedence constraints expressed as a directed acyclic graph. Their method is based on the concept of hierarchical decomposition [5]. A two-level method is used to solve for the performance measures of the task system. At the higher level, the system behavior is specified by a Markov chain whose states correspond to the combination of tasks in execution. At the lower level, the transition rates among the states are computed using a queuing network solver, which determines the throughput of the computer system for each system state. Their method is a significant improvement over the exact method using a Markov chain alone. However, the size of the state space still grows rapidly with the number of tasks. For a system with N tasks, in the worst case when all N tasks are in parallel, the number of states in this system is $(2^N - 1)$. This exponential complexity limits the use of the method to small systems only.

In [32], QN models are categorized into resource views, and the task graphs are categorized into path views. As indicated, integration of the path view and resource view of concurrent software and system is suitable for displaying and analyzing performance information. However, no methods for doing this are given. The path view shows behavior because it describes the sequence of events, actions and delays in the typical behavior of the system. The task graph is a path view dominated approach. Task graph analysis carries complex paths with parallelism and synchronization, even to account for contention delays.

Chu et al [6,7,8] proposed an analytical model for estimating task response times in distributed systems. In [7], they proposed a model which consists of two submodels. The first submodel is a queuing network model used for approximating module response times; The second submodel is a weighted control-flow graph model from which task response time can be obtained by aggregating module response times in accordance with the precedence relationships. In [6], the first submodel is developed to an extended queuing network model in order to characterize module response time in the system with resource contention, such as common memory and data files. In the graph submodel, in order to aggregate the fork-join subgraph, it approximates the probability distribution function for the execution time of each branch by either an Erlangian or a hyperexponential distribution function, and solves the response time for fork-join subgraph by integrating the pdf of every two branches. Although it simplifies the problem or makes it mathematically solvable, the amount of computation is also very large when the number of branches of a fork-join subgraph is large.

This thesis has a viewpoint very close to Chu et al [6], but with the new **three-point** approximation in place of their Erlangian or hyperexponential distribution approximation. In terms of the complexity of calculation, this approximation method is much simpler than what proposed in [6]. In the mean time, it is also accurate.

Chapter 3 The Three-point Approximation

This chapter describes the three-point approximation to calculate the response time for a Fork-join subgraph. First the basic algorithm is described. Then, the approximation will be applied to simple Fork-join cases to study its accuracy for different coefficients of variation (CV) of the delay of each branch of the Fork-join subgraph. Some general task graphs whose exact response time can be calculated numerically will also be studied to evaluate the approximation using for the entire task graph.

Dr. Woodside and Alex Hubbard proposed the following three-point approximation method [35]. Response time of Fork-join subgraphs with three branches of different distribution combination were calculated using a three-point approximation to the distribution function, and a numerical computation. The three-point approximation method does not assume any particular distribution so that for different CV value cases it has the same complexity with known mean and variance value for each branch. Hong Zhang [34] has done a limited evaluation of the approximation.

3.1 The Approximation Method

In this section we will describe the procedure of using the three-point approximation to calculate the response time for the Fork-join subgraph as proposed by Professor Woodside and Alex Hubbard. The exact calculation for $\bar{r}_j$ was given in section 2.1.2. The three-point approximation to the Fork-join subgraph is used when the mean and variance of delay of each branch are known. It provides an easy and fast way of calculating the delay for the Fork-join subgraph. It does not assume exponential distribution to avoid the possibility of overestimating the response time. In fact, it assumes a discrete distribution. This makes the computation for delay of Fork-join subgraph as simple as for sequential and alternate subgraphs.

The process to find the mean and variance of the whole subgraph is as follows:

a) For each branch i , find a discrete distribution with three point-value which has the same mean and variance as $f_{xi}(t)$:

(1) Choose three points t_{ij} for each branch i , and $j = 1, 2, 3$. The choice of the point value is important for the accuracy of the approximation. The choice given in [34] is:

$$\begin{aligned} t_{i1} &= \bar{t}_{xi} - \sigma_{xi} & \text{when } \bar{t}_{xi} - \sigma_{xi} > 0, & \quad \text{otherwise } t_{i1} = 0 \\ t_{i2} &= \bar{t}_{xi} \\ t_{i3} &= \bar{t}_{xi} + 2k \cdot \sigma_{xi} & \text{where } k &= \sigma_{xi} / \bar{t}_{xi} \quad (\text{if } k < 1, \text{ let } k=1) \end{aligned} \quad (3.1)$$

(2) Calculate the discrete probabilities a_{ij} for each branch i ($j = 1, 2, 3$) at these special points, so as to give a discrete distribution with the same mean $\bar{t}_{xi}$ and variance σ_{xi}^2 :

$$\begin{aligned} \sum_{j=1}^3 t_{ij} \cdot a_{ij} &= \bar{t}_{xi} \\ \sum_{j=1}^3 t_{ij}^2 \cdot a_{ij} &= (\bar{t}_{xi})^2 + \sigma_{xi}^2 \\ \sum_{j=1}^3 a_{ij} &= 1 \end{aligned} \quad (3.2)$$

Figure 3-1 Three-point Approximation: pdf of Three Point Values

b) For each branch i compute the discrete distribution function $K_i(t)$ which approximates to $F_{xi}(t)$ ($i=1,2,\dots,n$), ($j=1,2,3$)

$$\begin{aligned}
 K_i(t) &= 0 & 0 \leq t < t_{i1} \\
 K_i(t) &= a_{i1} & t_{i1} \leq t < t_{i2} \\
 K_i(t) &= a_{i1} + a_{i2} = 1 - a_{i3} & t_{i2} \leq t < t_{i3} \\
 K_i(t) &= 1 & t_{i3} \leq t
 \end{aligned} \tag{3.3}$$

Figure 3-2 Three-point Approximation: PDF of Three Point Approximation

Figure 3-1 and Figure 3-2 are the approximate value for an Erlangian-10 distribution function with mean equal to 2.

c) Calculate the distribution function $K_0(t)$ for the whole graph, which approximates $F_Y(t)$:

$$K_0(t) = \prod_{i=1}^n K_i(t) \tag{3.4}$$

This is a function which jumps up at a set of discrete values $t_1, t_2, \dots, t_m$ ($m \geq n$), which includes all the points t_{ij} in the component distribution.

d) Calculate the discrete probabilities $L_0(t)$ for whole graph

$$\begin{aligned} L_0(t_1) &= K_0(t_1) \\ L_0(t_i) &= K_0(t_i) - K_0(t_{i-1}) \quad i = 2, 3, \dots, m \end{aligned} \quad (3.5)$$

e) Calculate the mean and variance for whole subgraph from the discrete point value t_i and the probabilities $L_0(t_i)$:

$$\bar{t}_y = \sum_{i=1}^m t_i \cdot L_0(t_i) \quad (3.6)$$

$$\sigma_y^2 = \sum_{i=1}^m t_i^2 \cdot L_0(t_i) - (\bar{t}_y)^2 \quad (3.7)$$

What is important in the three-point approximation is how to choose the three point values. Test results show that for small CV value, the choice of (3.1) described in this section is adequate. When it comes to the cases with large CV value, this choice of the three point value will have a different effect on the accuracy of the approximation. We'll study the impact of the choice later.

3.2 A Few Examples

In order to evaluate the approximation approach, we can compute the delay of some simple tractable cases using the approximation, and then compare the approximation result with the actual results which are obtained from numerical calculation.

3.2.1 Simple Fork-join Cases

The reasons why we choose simple Fork-join cases are as follows:

- Mean and variance calculation for sequential and alternation subgraph are exact. All errors will originate is from Fork-join subgraph, so they are key to the overall calculation.

- Because the three-point approximation is exactly used to calculate the response time of Fork-join subgraph, it is necessary to evaluate Fork-join cases first.
- It is not difficult to obtain the theoretical value of response time of a Fork-join subgraph in which each branch contain only one atomic task assumed with exponential or Erlangian distribution delay. Hence, it is easy for us to evaluate the accuracy of the three-point approximation.

Fork-join cases used in this thesis cover many scenarios. We consider different number of branches, different value of *CV* for each task, and different mean value for each task.

When choosing the cases for studying, the following factors are considered:

N: Number of branches of a Fork-join subgraph;

Range: this value is used to specify how different the means of each branch in a Fork-join subgraph could be. It is the ratio of the largest mean to the smallest mean;

$$Range = MAX_i \{ \overline{t_{xi}} \} / MIN_i \{ \overline{t_{xi}} \} \quad i = 1, 2, \dots, N$$

CV: the coefficient of variation of delay for each branch. When a branch has *CV* value larger than 1, we assume it has hyperexponential distribution.

$$CV = \sigma_{xi} / \overline{t_{xi}}$$

kernel: this is the kernel value of Erlangian distribution. When a branch has small *CV* value, we assume it has Erlangian distribution.

$$kernel = (\overline{t_{xi}})^2 / \sigma_{xi}^2 = 1 / (CV)^2$$

These known distribution are used for calculating the exact mean and variance of response time of the Fork-join subgraph using (2.11) and (2.12), which is used to compare with the approximate value.

The simple Fork-join cases studied can be grouped into three sets. The first set consists 80 cases with *CV* value smaller than or equal to 1. The second set consists 80 cases with *CV* value larger than 1. The third set contains 22 cases also studied in [34] with *CV* value larger than or equal to 1. These 22 cases are all three-branch Fork-join subgraphs with different combinations of distribution of delay.

The first set of cases we choose covers all the combination of parameters such as value of *N*, *Range*, *CV*. The *CV* value varies from 0.1 to 1 (*CV* value = 0.1 means *kernel* = 100); the *N* varies from 2 to 20; the *Range* varies from 1 to 10. One thing that should be noted is that for simplicity of choosing different values of mean delay for each branch, when the *Range* is not equal to 1 (that means the mean delays of different branches are not equal), the mean delays of all the branches are equally spaced from the minimum to the maximum value.

The second set contains 80 cases with combinations of *N*, *Range* and *CV*. The choice of *N* and *Range* are same as in the first set. The *CV* value chosen varies from 2 to 5.

The approximation results are compared with exact values obtained from Matlab in order to get a complete view of accuracy of the approximation. The results case by case are given in Appendix A.1 and A.2. They are summarized in the following sections.

3.2.1.1 Simple Fork-join Graph With Small CV

In the 80 cases of the first set we studied, the different combinations of *N*, *Range*, and *kernel* that occur are listed in Table 3.1. These all have Erlangian distributions.

Table 3.1: The parameters of the 80 cases in the first set ($CV \leq 1$)

case	<i>N</i>	<i>Range</i>	<i>CV</i>	<i>kernel</i>	case	<i>N</i>	<i>Range</i>	<i>CV</i>	<i>kernel</i>
1	2	1	1	1	41	10	1	1	1
2	2	1	0.5	4	42	10	1	0.5	4
3	2	1	0.25	16	43	10	1	0.25	16

Table 3.1: The parameters of the 80 cases in the first set ($CV \leq 1$)

case	<i>N</i>	Range	CV	kernel	case	<i>N</i>	Range	CV	kernel
4	2	1	0.167	36	44	10	1	0.167	36
5	2	1	0.1	100	45	10	1	0.1	100
6	2	2	1	1	46	10	2	1	1
7	2	2	0.5	4	47	10	2	0.5	4
8	2	2	0.25	16	48	10	2	0.25	16
9	2	2	0.167	36	49	10	2	0.167	36
10	2	2	0.1	100	50	10	2	0.1	100
11	2	5	1	1	51	10	5	1	1
12	2	5	0.5	4	52	10	5	0.5	4
13	2	5	0.25	16	53	10	5	0.25	16
14	2	5	0.167	36	54	10	5	0.167	36
15	2	5	0.1	100	55	10	5	0.1	100
16	2	10	1	1	56	10	10	1	1
17	2	10	0.5	4	57	10	10	0.5	4
18	2	10	0.25	16	58	10	10	0.25	16
19	2	10	0.167	36	59	10	10	0.167	36
20	2	10	0.1	100	60	10	10	0.1	100
21	5	1	1	1	61	20	1	1	1
22	5	1	0.5	4	62	20	1	0.5	4
23	5	1	0.25	16	63	20	1	0.25	16
24	5	1	0.167	36	64	20	1	0.167	36
25	5	1	0.1	100	65	20	1	0.1	100
26	5	2	1	1	66	20	2	1	1
27	5	2	0.5	4	67	20	2	0.5	4
28	5	2	0.25	16	68	20	2	0.25	16
29	5	2	0.167	36	69	20	2	0.167	36
30	5	2	0.1	100	70	20	2	0.1	100

Table 3.1: The parameters of the 80 cases in the first set ($CV \leq 1$)

case	N	Range	CV	kernel	case	N	Range	CV	kernel
31	5	5	1	1	71	20	5	1	1
32	5	5	0.5	4	72	20	5	0.5	4
33	5	5	0.25	16	73	20	5	0.25	16
34	5	5	0.167	36	74	20	5	0.167	36
35	5	5	0.1	100	75	20	5	0.1	100
36	5	10	1	1	76	20	10	1	1
37	5	10	0.5	4	77	20	10	0.5	4
38	5	10	0.25	16	78	20	10	0.25	16
39	5	10	0.167	36	79	20	10	0.167	36
40	5	10	0.1	100	80	20	10	0.1	100

Figure 3-3 shows the histogram of errors of mean delay of the 80 cases, we can see that most of the absolute errors are within 5%. There are four cases with larger errors. We'll analyze them later in this section.

Figure 3-3 The Histogram of Errors of 80 Fork-join Cases with $CV \leq 1$

Figure 3-4 The Errors of Mean with *kernel* Value

Figure 3-4 (continue.) The Errors of Mean with *kernel* Value

Figure 3-5 The Errors of Mean with Branch Value N

Figure 3-5 (continue.) The Errors of Mean with Branch Value N

Figure 3-6 The Errors of Mean with *Range* Value

Figure 3-6 (continue.) The Errors of Mean with Range Value

Figure 3-4 to Figure 3-6 shows the relationships between the error of mean delay and the parameters of the Fork-join subgraph, such as *kernel*, *N*, etc. From the results of the analysis of these 80 cases, we can observe some general trends. For one value of *N*, the larger the *kernel* and the larger the *Range*, then the smaller the percentage error of mean delay. For one value of *Range*, the smaller the *N* and the larger the *kernel*, then the smaller the percentage error.

Because the larger *kernel* value means smaller *CV* value, so it indicates that the approximation is more accurate for small variance of delay.

Large *Range* values also give more accurate approximation. A small *Range* will result in nearly equal mean delay of the branches, especially when *N* is large, that situation will be similar to equal mean branches. With equal mean branches, we have only three point values for the approximation, so it will reduce the accuracy of the approximation, no doubt. If the *Range* is larger, there will be more point values obtained; obviously, it will increase the accuracy of the approximation result.

Therefore, we can conclude from the above that the three-point approximation is more accurate for small *N*, large *Range* and small *CV*.

The following Table 3.2 is the analysis of the percentage error of the 80 cases. We will define MAE as Mean Absolute value of the percentage Error of approximation.

Table 3.2: The percentage error of 80 fork-join cases with small CV

MAE of mean delay	MAE of σ	error counts			
		0--1.0%	1.0--2.0%	2.0--5.0%	above 5.0%
1.56%	14.93%	44	17	15	4

We can observe from the above table that the MAE of mean delay of the approximation for 80 cases is quite small, and about half of these cases have an error less

than 1%. But that of standard deviation is fairly large.

In only four cases is the error of mean delay above 5%; these four cases are:

case41, -8.62%; case61, -18.06%; case62, -7.87%; case66, -7.46%.

Let's analyze why these large error appears:

All of these errors are for cases with equal, or nearly equal delay on the branches, and many branches with relatively large *CV* value (near to 1). Therefore they confirm our general conclusion on sources of inaccuracy.

Case41 has 10 branches with equal mean of delay, as indicated previously. There are altogether only three point values to do the approximation, and this decreases the accuracy of the approximation. Case61 and case66 are in the same situation as case41, both of them having 20 branches with equal delay. Case61 has *CV* value equal to 1, so its accuracy is the worst of all the cases. Case62 has 20 branches with *Range* equal to 2. Because of so many branches with similar mean delays, it is not surprising the accuracy may be decreased.

Further details of these results are found in Appendix A.1.

3.2.1.2 Simple Fork-join graph with large *CV*

The parameters of the second set of 80 cases studied are described in Table 3.3 below. These all have hyperexponential distribution.

Table 3.3: The parameters of the 80 cases in the second set ($CV > 1$)

case	<i>N</i>	<i>Range</i>	<i>CV</i>	case	<i>N</i>	<i>Range</i>	<i>CV</i>
1	2	1	2	41	10	1	2
2	2	1	3	42	10	1	3
3	2	1	3.5	43	10	1	3.5
4	2	1	4	44	10	1	4

Table 3.3: The parameters of the 80 cases in the second set ($CV > 1$)

<i>case</i>	<i>N</i>	<i>Range</i>	<i>CV</i>	<i>case</i>	<i>N</i>	<i>Range</i>	<i>CV</i>
5	2	1	5	45	10	1	5
6	2	2	2	46	10	2	2
7	2	2	3	47	10	2	3
8	2	2	3.5	48	10	2	3.5
9	2	2	4	49	10	2	4
10	2	2	5	50	10	2	5
11	2	5	2	51	10	5	2
12	2	5	3	52	10	5	3
13	2	5	3.5	53	10	5	3.5
14	2	5	4	54	10	5	4
15	2	5	5	55	10	5	5
16	2	10	2	56	10	10	2
17	2	10	3	57	10	10	3
18	2	10	3.5	58	10	10	3.5
19	2	10	4	59	10	10	4
20	2	10	5	60	10	10	5
21	5	1	2	61	20	1	2
22	5	1	3	62	20	1	3
23	5	1	3.5	63	20	1	3.5
24	5	1	4	64	20	1	4
25	5	1	5	65	20	1	5
26	5	2	2	66	20	2	2
27	5	2	3	67	20	2	3
28	5	2	3.5	68	20	2	3.5
29	5	2	4	69	20	2	4
30	5	2	5	70	20	2	5
31	5	5	2	71	20	5	2

Table 3.3: The parameters of the 80 cases in the second set ($CV > 1$)

case	N	Range	CV	case	N	Range	CV
32	5	5	3	72	20	5	3
33	5	5	3.5	73	20	5	3.5
34	5	5	4	74	20	5	4
35	5	5	5	75	20	5	5
36	5	10	2	76	20	10	2
37	5	10	3	77	20	10	3
38	5	10	3.5	78	20	10	3.5
39	5	10	4	79	20	10	4
40	5	10	5	80	20	10	5

Figure 3-7 shows the histogram of errors of mean of the 80 cases with large CV value. The data of these errors are in Table A.2 of the Appendix. We can see that the errors are worse than for small CV cases. And all of them distributed from -6% to 6%. The MAE of mean delay is 2.73%, MAE of σ is 3.28%. Further details of these results are found in Appendix A.2.

Figure 3-7 The Histogram of Errors of 80 fork-join Cases with $CV > 1$

3.2.1.3 Simple Fork-join graph with mixed distribution

The third set of tests consists of 22 cases chosen from [34]. These are Fork-join subgraphs with three branches. The delay of the three branches includes a combination of exponential distribution and hyperexponential distribution. They are as shown in Table 3.4:

Table 3.4: The parameters of 22 cases in the third set

case	$\bar{t}_1$	CV_1^2	$\bar{t}_2$	CV_2^2	$\bar{t}_3$	CV_3^2
case1	1	1	1.5	1	2.25	2
case2	1	1	1.5	1	2.25	4
case3	1	1	2	1	4	4
case4	1	1	2	2	4	1
case5	1	1	1.2	4	1.44	1
case6	1	1	1.5	4	2.25	1
case7	1	1	2	4	4	1
case8	1	2	2	1	4	1
case9	1	4	2	1	4	1
case10	1	1	1.5	2	1.5	2
case11	1	1	2	2	2	4
case12	1	1	1.2	4	1.44	4
case13	1	1	1.5	4	2.25	4
case14	1	1	2	4	4	4
case15	1	2	2	1	4	2
case16	1	4	1.2	1	1.44	4
case17	1	4	1.5	1	2.25	4
case18	1	4	2	1	4	4
case19	1	2	2	2	4	1
case20	1	4	1.2	4	1.44	1
case21	1	4	1.5	4	2.25	1
case22	1	4	2	4	4	1

When using the approximation, the errors of comparing the approximation results with simulation results are listed in Table A.3 in Appendix.

We can see that the approximation for the simple Fork-join subgraph is accurate. But the accuracy for hyperexponential distribution cases is worse than for exponential distribution cases, and both of them are less accurate than Erlangian distribution cases. So it is concluded that the approximation is more suitable for small CV value cases.

Since the Fork-join subgraph is only a part in a task graph, from the above analysis of the accuracy of the single Fork-join subgraph, we can say that using the approximation in a whole task graph results in a tolerable error of response time. It should be better than in the single Fork-join subgraph. So we'll evaluate the approximation used for general cases in next section.

3.2.2 Small Tractable Cases

We also try to use the approximation to compute the delay of a task system including all of the three types of subgraphs. We choose the cases in which, in each Fork-join subgraph, each branch is an atomic task, so that the exact value of delay of the Fork-join subgraph can be calculated. We concentrate on the mean of total response time since it is the most important measure in performance prediction.

case1, single Fork-Join graph: 20 tasks of different mean response time, all of which are exponentially distributed. The mean delay of each task is shown in Table 3.5:

Table 3.5: mean delay of branches of case1

$\bar{t}_1$	$\bar{t}_2$	$\bar{t}_3$	$\bar{t}_4$	$\bar{t}_5$	$\bar{t}_6$	$\bar{t}_7$	$\bar{t}_8$	$\bar{t}_9$	$\bar{t}_{10}$
1	2	2.5	1.5	1	3	2	2	1.5	2.5
$\bar{t}_{11}$	$\bar{t}_{12}$	$\bar{t}_{13}$	$\bar{t}_{14}$	$\bar{t}_{15}$	$\bar{t}_{16}$	$\bar{t}_{17}$	$\bar{t}_{18}$	$\bar{t}_{19}$	$\bar{t}_{20}$
2	1	1.5	1.5	2	1.5	2	2	2.5	1

case2, single Fork-Join graph: 10 tasks of different mean response time, all of which

are exponentially distributed. The mean delay of each task is described in Table 3.6:

Table 3.6: Mean delay of branches of case2

$\bar{t}_1$	$\bar{t}_2$	$\bar{t}_3$	$\bar{t}_4$	$\bar{t}_5$	$\bar{t}_6$	$\bar{t}_7$	$\bar{t}_8$	$\bar{t}_9$	$\bar{t}_{10}$
1	2	2.5	1.5	1	3	2	2	1.5	2.5

case3, single Fork-Join graph: 10 tasks of equal mean response time, all of which are exponentially distributed with mean delay of 1.

case4, a small task graph contains Fork-Join, alternate, sequential subgraphs as shown in Chapter 2 (Figure 2-4). This is the same task graph as in Chu's paper [6].

The delay of each tasks is exponentially distributed, the mean delay of them are as listed in Table 3.7 below:

Table 3.7: Mean delay of tasks of case4

$\bar{t}_1$	$\bar{t}_2$	$\bar{t}_3$	$\bar{t}_4$	$\bar{t}_5$	$\bar{t}_6$	$\bar{t}_7$	$\bar{t}_8$
5	1	1	5	5	1	5	1

At first we use numerical method calculate the total delay of the Fork-join subgraph (that is task3 and task4), then use the equations discussed in section 3 to calculate the response time of the whole task graph, as the exact value.

case5, a comparatively larger task graph contains Fork-join, alternate, and sequential subgraphs shown in Figure 3-9. The weight of the Fork-join subgraph is a little more than in case4. This graph is such that it is easy to compute the exact solution. The delay of each task is exponentially distributed, and the mean of them are as listed in Table 3.8:

Table 3.8: Mean delay of tasks of case5

$\bar{t}_1$	$\bar{t}_2$	$\bar{t}_3$	$\bar{t}_4$	$\bar{t}_5$	$\bar{t}_6$	$\bar{t}_7$	$\bar{t}_8$	$\bar{t}_9$	$\bar{t}_{10}$	$\bar{t}_{11}$
1	2.5	2	2	2	2.5	1.4	1	3	2	1
$\bar{t}_{12}$	$\bar{t}_{13}$	$\bar{t}_{14}$	$\bar{t}_{15}$	$\bar{t}_{16}$	$\bar{t}_{17}$	$\bar{t}_{18}$	$\bar{t}_{19}$	$\bar{t}_{20}$	$\bar{t}_{21}$	$\bar{t}_{22}$
1	1.5	2.5	1.5	1.4	2	1	1	1.5	2	1

Figure 3-8 Small Task Graph of Case5

Table 3.9: The comparison results of 5 small tractable task graphs

case	exact mean	approximation mean	approximation error
case 1	7.088	6.738	-4.94%
case 2	6.025	5.993	-0.53%
case 3	2.929	2.677	-8.62%
case 4	14.067	14.133	0.47%
case 5	16.594	16.739	0.88%

We can see from Table 3.9 that the approximation mean error is -4.94% for case1, and -0.534% for case2. These two cases both have unequal means. Case1 has 20 branches, case2 has 10 branches; therefore it is clear that for more branches, the approximation is less accurate.

Case2 and case3 both have 10 branches, but the approximation error differed greatly, since in case3 the 10 branches have equal mean. In our three-point approximation, there's only three point values that are obtained for the whole subgraph, which decreases the accuracy of the approximation. This is the same conclusion we got from the previous section.

In case4 and case5, the weight of the Fork-join subgraph in the whole graph is different, the error of the approximation are all below 1%, and it seems that the approximation is good. Yet we need more experiments to validate this. We will study large amount of general cases in Chapter 6.

This result also validates our results obtained in the previous section: the approximation is more accurate for cases with a small number of branches and different mean delay for each branch. And since the sequential and alternate subgraph does not introduce error, it is sure that when the approximation is used in general cases, the results

of response time will be better. To confirm this, we need do some comparison over general cases, which is the object of next chapter, to create a task graph system.

3.3 Three-point Approximation Summary

In this chapter, the basic method of the three-point approximation is described in detail. Simple Fork-join cases with different parameters are studied. Small tractable task graph contains combination of sequential, Fork-join, and alternate subgraphs are also studied in this chapter.

The experiment results shows that the approximation method is more accurate for cases with following characteristics:

- Cases with small CV value of delay;
- Fork-join subgraph has small number of branches;
- The mean delay of branches of Fork-join subgraph differed significantly.

Chapter 4 A Simulation Model

4.1 Objectives of Simulation

In order to evaluate the accuracy of the three-point approximation, the approximation results need to be compared with the exact values obtained from the mathematical calculation.

Since it is not easy to calculate the response time mathematically for a complex task graph, we need to use simulation to imitate the behavior of the system over time. The data obtained from simulation can be used as the exact value to verify the three-point approximation.

4.2 Detail of Setting Up a Simulation Model

4.2.1 The Overall Concept of the Simulation Model

To evaluate the response time of three-point approximation method when it applies to an entire task graph, the approximation results should be compared with some exact value. For simple Fork-join subgraphs, Alex Hubbard and Hong Zhang [34] used a numerical method to calculate the exact value. Their research work shows that the approximation results have different accuracy for different combinations of distribution branches. This approach works only for a small graph. However it is difficult to calculate the numerical value for a complicated task graph. So that the simulation result is used.

We use PARASOL as the simulation tool [23,24]. PARASOL is a simulator/emulator software package for distributed and/or parallel computer systems. The simulation consists of two parts: an application routine, and a special task called **genesis** within which the simulated hardware is defined.

The simulation can be arranged to determine delays with no processor contention.

The simulation model has a separate node (processor) and a **simulated task** which runs on it for each task in the task graph. The delay of each task is its execution demand, defined with a suitable distribution. Since each **simulated task** has its own node (processor) to run, there is no extra queuing delay introduced in the simulation. Each simulated task starts to execute when it has received the messages from all of its predecessor(s). When completed, it activates its successor(s) by passing the message to them.

The **genesis task** creates **simulated tasks** for all the tasks in a graph from a single task class, initiates their parameters and starts the first task by sending a message containing the time stamp to it. If there exist several tasks that need to begin simultaneously, a dummy node will be added before them which will consume no service time, but is used as a template to which the **genesis task** could send the message. So this simplifies the **genesis task**. It always sends one message to start the whole task graph, and it does not need to deal with starting several tasks.

When the last **simulated task** completes its execution, it records the time, and sends a fresh message to the first **simulated task**. The difference between this value and the value of the time stamp, which is called completion time of the simulation model, is the delay of the whole task graph. A while loop is used in the **simulated task** so that it can keep running until the total simulation time is due. The simulation ran in a cycle, giving a large number of complete executions of the graph.

To initiate a PARASOL simulation, a random seed and a simulation duration need to be provided by the user. For the random seed, a large odd integer is preferred; For simulation duration, the accuracy of the result is in proportion to the duration value. The goal is to get results as accurate as possible but keep simulation time relatively short. In general, simulation duration has about 20,000 times the response time, giving about 20,000 response.

In our simulation model, library call *ps_block_stat* is used to generate a number of

independent replications of data, and a long batch time is used to make the results of simulation accurate. In fact, we try to keep the 95% confidence interval within 1% of the exact mean value so that we can treat the simulation value as the exact value of response time for the task graph. For simple Fork-join subgraph and some mathematically tractable simple graphs, the simulation results are compared with the exact value obtained from mathematical formula [13]. It shows the simulation results are accurate enough to be considered as the "exact value", within $\pm 1\%$

4.2.2 The Algorithm & Parameters of the Simulation Model

The following is the pseudo-code of the algorithm for the simulation model:

```

genesis task
begin
    while (not the end of parameter file) do
        begin
            input parameters of one task;
            set up a node for the task;
            resume simulated task on the node;
        end
    end while
    initiate the simulation by sending message to the first simulated task;
    suspend itself;
end

simulated task
begin
    while (TRUE) do
        wait for messages from predecessors;
        consume cpu time;
        send message(s) to its successors;
    end

```

The **genesis** task reads the parameters for each task and sets up a node (processor) for it with a unique numerical ID number in an increasing manner. We make use of this ID

number for coordination. Each task in the graph is assigned an unique task ID number starting from 1, and the last task has the largest task ID number. The parameters of tasks listed in the parameter file in their ID sequence. When **genesis** task reads in each task's parameter, a **simulated task** is generated with assigned ID number matching the task ID in the task graph.

The parameters of a task include:

- number of its predecessors,
- number of its successors, and
- probability of its successor(s),
- ID of its successor(s),
- its mean delay,
- its distribution of delay,
- its flag (whether it is the last one in the task graph).

The ID number of its successor is very important for determining the message flow control.

4.3 Accuracy of the Simulation

To validate the simulation model, simulation results are compared with exact results for some tractable cases. We examined cases containing the Fork-join subgraph and cases with tractable response time which are used in the previous chapter.

The following Table 4.1 contains the comparison results of the exact mean value of the Fork-join subgraphs with the simulation results. The 16 cases are chosen from the first set of 80 cases studied in Chapter 4, which the CV value of mean delay for each task is 1, that is, the mean delay have exponential distribution, so that it is easy to calculate the exact

mean response time of the 16 Fork-join subgraph cases.

Table 4.1: The accuracy of the simulation of 16 cases with Fork-join task graphs of exponential distribution

case	exact mean	simulation mean	simulation error
1	1.500	1.500	-0.03%
2	2.333	2.331	-0.09%
3	5.167	5.165	-0.04%
4	10.091	10.032	-0.59%
5	2.283	2.283	-0.01%
6	3.534	3.528	-0.15%
7	7.689	7.683	-0.08%
8	14.806	14.787	-0.13%
9	2.929	2.929	0.02%
10	4.551	4.557	0.13%
11	9.962	9.970	0.08%
12	19.203	19.201	-0.01%
13	3.598	3.597	-0.01%
14	5.625	5.628	0.04%
15	12.444	12.458	0.11%
16	24.068	24.116	0.20%

Table 4.2 contains the compare results for 5 tractable cases studied in Chapter 3:

Table 4.2: Simulation error for five tractable cases

case	exact mean	simulation mean	simulation error
case 1	7.088	7.073	-0.21%
case 2	6.025	6.020	-0.08%
case 3	2.929	2.929	0.02%

Table 4.2: Simulation error for five tractable cases

case	exact mean	simulation mean	simulation error
case 4	14.067	14.016	-0.36%
case 5	16.594	16.589	-0.03%

The simulation results with 95% confidence interval of 1% accuracy are very close to the exact results, the simulation error being below 0.6%. This gives us confidence in using the simulation to evaluate our approximation in further work.

4.4 Summary of Simulation

In this chapter a simulation model is established. The algorithm and parameters for the simulation model are described.

Simulation results are validated by comparing with exact results from mathematical calculation. The simulation results of 95% confidence interval is less than 0.6% of the mean. This gives us confidence in considering simulation results as exact value to evaluate our three-point approximation.

Chapter 5 A Framework for System Evaluation

In order to evaluate the approximation accuracy for a large number of different models, it is essential to study wide variety of task graphs. It is a very time-consuming and arduous task to manually create all these task graphs.

To avoid the tedious manual work, we have developed a software system named **autogene**. This program can automatically generate task graphs with randomly chosen structures based on the input parameters.

In this chapter, we first discuss the structure of a task graph, and then describe the algorithm of **autogene** for generating such structures.

5.1 Graph Parameters

We described the following two types of graph notation in Chapter 2: task graph notation and tree structure notation. Task graph notation parameters are used for simulation and tree structure notation parameters are used for approximation purposes.

From the task graph notation (Figure 2-4), the execution flow of the whole task is clear. In the simulation model, we set up the structure of task graph using the **genesis** task. The parameters for each node are:

Table 5.1: Definition of simulation parameters

parameter	definition
<i>distribution</i>	Distribution function of the node. It can be "uniform", "exponential", "Erlangian", "hyperexponential";
<i>parameter1</i>	The parameter of the distribution; (see Table 5.2 for detail)
<i>parameter2</i>	Same as above;

Table 5.1: Definition of simulation parameters

parameter	definition
<i>No_receive_msg</i>	Number of triggered messages the node will receive in order to be able to execute;
<i>No_send_msg</i>	Number of triggered messages the node will send to its successor(s) after its completion of execution;
<i>destination_of_msg</i>	It is the identity of each successor node, to which a triggered message must be sent;
<i>prob_of_destination</i>	Probability of sending each message. This is less than 1 only if the node is a "alternate" one. The <i>destination_of_msg</i> and the <i>prob_of_msg</i> come in set, the number of sets is equal to <i>No_send_msg</i> ;
<i>last_node_flag</i>	flag to indicate the last node. (0 means it is the last node)

Table 5.2: The parameters of the distribution

distribution	uniform	exponential	Erlangian	hyperexponential
<i>parameter1</i>	<i>a</i>	<i>mean</i>	<i>mean</i>	<i>mean</i>
<i>parameter2</i>	<i>b</i>	<i>CV(=1)</i>	<i>kernel</i>	<i>CV</i>

The parameters of nodes are listed in the `simu_parameter` file in sequence of each node's ID number.

For approximation purposes, we use the tree structure notation (Figure 2-5). The parameters for approximation include every node in the tree structure (both composite nodes and atomic nodes), and they need to be listed in sequence from the root of the tree, then the first level, the second level, and so on, until it comes to the last level which only has atomic nodes.

For a composite node, the input parameters are:

Table 5.3: Definition of approximation parameters for composite nodes

parameter	definition
<i>node_name</i>	The name of the node, usually starts with <i>comp</i> ;
<i>node_topology</i>	Including sequential(1), Fork-join(2), alternate(3);
<i>node_prob</i>	Generally it is equal to 1. In the case of alternate subgraph, the probability of each branch counts as the <i>node_prob</i> of each child node;
<i>name_of_children</i>	A list which includes all its child nodes' <i>node_name</i> . An integer is added at the end of the list to indicate the end of the list of child node names.

Table 5.4: Definition of approximation parameters for atomic nodes

parameter	definition
<i>node_name</i>	The name of the node, usually start with <i>atom</i> ;
<i>node_topology</i>	For atomic node, <i>node_topology</i> is equal to 0;
<i>node_prob</i>	Probability of the node be executed;
<i>node_mean</i>	Mean delay of the node;
<i>node_variance</i>	Variance of delay of the node. What is needed for the three-point approximation is mean and variance of delay, so that the last two parameters are important.

It is straightforward to generate a graph using tree structure notation. To decide how large the graph would be, we need two arguments: (1) *max_level*, the maximum number of levels, and (2) *max_children*, the maximum number of children nodes a composite node can have.

5.2 Task Graph Creation

There are two main steps when execute the *autogene* system: first, set up the

structure of the task graph; second, output the parameters needed by simulation and approximation. A conversion from the structure to certain representation is needed in order to output the desired parameter files for simulation and approximation. In this section, we discuss the mechanism in choosing the appropriate values for creating and setting up the structure according to the known value of *max_level* and *max_children*.

Each composite node created is assigned a name *comp_i*, where *i* is an integer starting from 0. Each atomic node created is assigned a name *atom_i*, where *i* is an integer starting from 1. This guarantees that each node has a unique name.

First, the root node of the tree is created and is assigned the name of *comp_0*. It is always a sequential composite node containing three child nodes. In order for the simulation to start from and end at one atomic node, the first and last child nodes must be atomic node (First one has the name of *atom_1*). The ID number of the last atomic node is unknown at this point, since it has to be the last one and has the largest ID number. Whenever an atomic node is created, its ID number is increased from the previous one, and is recorded. When the whole graph is generated, the ID for the last node can be derived. The second child node of *comp_0* is a composite node, and its name is *comp_1*. The whole graph is set up based on this node. The topology of *comp_1* is chosen randomly from the Fork-join, and the alternate with equal probabilities (can change). For *comp_1*, as well as other composite nodes, a random number generator is used to generate a number *n* such that $2 \leq n \leq \text{max_children}$ is the number of children nodes of this composite node, and another random number selects the topology of each of its children nodes: from the four topologies (atomic, sequential, Fork-join, alternate) with certain probabilities, but the topology of child node must be different from its parent node. After that step, go through its child nodes in the same way.

For each atomic node, random number are generated to decide its distribution and its parameters.

For a composite node, the number of its children nodes is selected randomly with equal probability between 2 and *max_childrzn*. This range is to guarantee this composite node to have children nodes -- hence it is a composite node.

Since we want to maintain the desired level and size of the structure, at the high level the probability that a composite node's child node's topology is atomic should be small, so that it can be extended into a deeper level. At the deeper level, the probability that its child node's topology is atomic increases accordingly. Therefore, we select this probability depending on the number of levels specified and the number of the level in which the composite node reside. The residing level of the composite node is found as *existing_level* (*existing_level* < *max_level*), and an intermediate argument is defined as $l = \text{max_level} - \text{existing_level}$. The larger l used, the smaller probability that its child node's topology would be atomic. The probability of the topology to be atomic is equal to $1 / (3l)$ ($l > 1$), when $l = 1$, the probability is equal to 1. (The choice of function was made arbitrarily).

It is reasonable to force a child node to have a different topology from its parent node (If not, it could be merged with the parent). That is, for any sequential composite node, its child nodes' topology have to be atomic, Fork-join or alternate ($\text{Prob}(\text{Forkjoin}) = \text{Prob}(\text{alternate}) = 0.5 \cdot (1 - 1 / (3l))$). For any Fork-join or alternate composite node, its children nodes' topology have to be atomic or sequential ($\text{Prob}(\text{sequential}) = 1 - 1 / (3l)$). To determine the distribution of trigger messages to successor tasks is awkward if the Fork-join is a child of an alternate, or vice versa. This is why we do not let the Fork-join's child be a alternate node, and vice versa.

A sequential composite node's first and last child nodes must be atomic. And if two consecutive child nodes of a sequential composite node are both Fork-join or alternate, it is difficult in deciding the parameters used for simulation, such as *No_receive_msg*, *No_sending_msg*, etc. So that a dummy node is inserted between them. A dummy node has

0 execution time, and so it will have no impact on the response time of the whole graph structure. This can be done after setting up the whole structure, by going through the tree structure to check if adjacent siblings of a sequential composite node are both of Fork-Join or alternate topology. All of these considerations are to guarantee the simplicity of obtaining parameters of simulation.

For an atomic node, two ways were used to choose the distribution of its execution time. First one is to construct a graph with a combination of distributions with a random choice among exponential, uniform, Erlangian, hyperexponential (with probabilities of 10%, 40%, 40%, and 10%). The second one is to define a specific distribution function for all atomic nodes, and choose a mean value for them, then generate other parameters as needed by their distribution. For example, for exponential distribution, we do not choose any parameter other than the mean. since the mean is adequate enough to decide the variance of exponential distribution, $CV = 1$; for uniform distribution, we can choose the second random number between 0 and the mean, get the upper bound and lower bound of uniform distribution by adding or subtracting the random number to or from the mean; for Erlangian distribution, we can choose $kernel = [1,100]$ (integer), so that $CV = \sqrt{1/(kernel)}$. For hyperexponential distribution, we choose the second value between [2,5] as CV . By doing so, we can generate task graph with same structure and mean delay for each atomic node, while the distribution and variance of each node may vary. So we can calculate how the approximation method behaves under different CV value, to evaluate the approximation more completely.

5.3 Output the Parameter File

After determining the structure of a task graph, the second step is to output the parameter files for simulation and approximation.

During the procedure of creating a node, each atomic node is assigned a unique

identifier number i which is part of its name $atom_i$. We set up a link list of these atomic nodes in their ID sequence. For each atomic node, we observe its parent's topology and its siblings' topologies, and find the value of the following parameters used in the simulation: (1) *No_send_msg*; (2) *No_receive_msg* of this atomic node; (3) Task ID number of *destination_of_msg*, and (4) *prob_of_destination*. The algorithm of finding these parameters is as follows:

begin

 check the atomic node's parent node's topology

if its parent node is Fork-join or alternate node

 set *No_receive_msg* to 1;

 set *No_send_msg* to 1;

 set *destination_of_msg* to the task ID of the Fork-join or alternate node's next sibling node; (it must be atomic);

 set *prob_of_destination* to 1;

else if its parent node is sequential node

 check the position of this atomic node in the sequential composite node

if it is the first child node

 set *No_receive_msg* to 1;

if its next sibling node is atomic

 set *No_send_msg* to 1;

 set *destination_of_msg* to the task ID of that atomic node;

 set *prob_of_destination* to 1;

else if its next sibling node is Fork-join or alternate

 set *No_send_msg* to n (n is the number of branches of the sibling node);

if the sibling node's child node is atomic node

 set *destination_of_msg* to the task IDs of the sibling node's child nodes;

 set *prob_of_destination* to the probabilities of the branches of the sibling node (for Fork-join sibling node, the probability of each branch equals to 1);

else if the sibling node's child node is composite node

 set *destination_of_msg* to the task IDs of the first child node of the sibling node's child nodes (if the

child node is composite node, it must be sequential composite node, and the sequential composite node's first child node must be atomic node);
 set *prob_of_destination* to the probabilities of the branches of the sibling node (for Fork-join sibling node, the probability of each branch equals to 1);
else if it is the middle child node
 if its previous sibling node is atomic node or alternate node
 set *No_receive_msg* to 1;
 else its previous sibling node is Fork-join node
 set *No_receive_msg* to *n*, (*n* is the number of branches of the sibling node);
 if its next sibling node is atomic
 set *No_send_msg* to 1;
 set *destination_of_msg* to the task ID of the atomic node;
 set *prob_of_destination* to 1;
 else if its next sibling node is Fork-join or alternate
 set *No_send_msg* to *n* (*n* is the number of branches of the sibling node);
 if the sibling node's child node is atomic node
 set *destination_of_msg* to the task IDs of the sibling node's child nodes;
 set *prob_of_destination* to the probabilities of the branches of the sibling node (for Fork-join sibling node, the probability of each branch equals to 1);
 else if the sibling node's child node is composite node
 set *destination_of_msg* to the task IDs of the first child node of the sibling node's child nodes (if the child node is composite node, it must be sequential composite node, and the sequential composite node's first child node must be atomic node);
 set *prob_of_destination* to the probabilities of the branches of the sibling node (for Fork-join sibling node, the probability of each branch equals to 1);
else it is the last child node
 if its previous sibling node is atomic node or alternate node
 set *No_receive_msg* to 1;
 else its previous sibling node is Fork-join node
 set *No_receive_msg* to *n*, (*n* is the number of branches of

```

        the sibling node);
    set No_send_msg to 1;
    set destination_of_msg to the task ID of the sequential node's
    parent node's next sibling node (it must be an atomic node);
    set prob_of_destination to 1;
end

```

All these parameters are stored in that node's record. We print the parameters for each node in the atomic node linked list to a file called **simu_parameter**.

To transfer the tree structure notation to the task graph notation, we need the above procedure to obtain the parameters for the simulation parameter file. If we have a task graph of certain instance, then the simulation parameters can be obtained by examining the graph directly. We just observe each node's input and output path, and can obtain these parameters.

For example, the **simu_parameter** file for Figure 2-4:

```

atom1:
    2      m1    1      1      2      1
    (distribution, exp) (mean) (CV) (No_receive_msg) (No_send_msg) (not last node)
    2      P1
    (destination_msg[0]) (destination_prob[0])
    3      P2
    (destination_msg[1]) (destination_prob[1])
atom2:
    2 m2 1 1 1 1
    4 1
atom3:
    2 m3 1 1 2 1
    5 1
    6 1
atom4:
    2 m4 1 1 1 1
    8 1

```


atom5:

2 m_5 1 1 1 1
7 1

atom6:

2 m_6 1 2 1 1
7 1

atom7:

2 m_7 1 2 1 1
8 1

atom8:

2 m_8 1 1 1 0 (0 means it is the last node)
1 1

For the data file for the approximation solution, we go through the tree structure, make up another linked list including all the composite nodes and atomic nodes in the tree structure notation, layer by layer, from the root node to the leaf nodes at the bottom level. So when we create the parameter file for approximation use, we just need to print out the parameters of each node in this linked list in sequence to a file called **appr_parameter**.

The **appr_parameter** file for Figure 2-5 is:

level 0:

comp0 1 1 atom1 comp1 atom8 10
(node_topology) (node_prob) (children list, the last 10 indicate the end)

level 1:

atom1 0 1 m_1 m_1^2
(node_topology) (node_prob) (node_mean) (node_variance)
comp1 3 1 comp2 comp3 10
(node_topology, 3 means alternate) (node_prob) (node list)
atom8 0 1 m_8 m_8^2

level 2:

comp2 1 p_1 atom3 comp4 atom7 10
(node_topology) (node_prob) (children list)
comp3 1 p_2 atom2 atom4 10
(node_topology) (node_prob) (children list)

level 3:

atom3 0 1 m_3 m_3^2

```

comp4  2  1  atom5  atom6  10
atom7  0  1  m7  m72
atom2  0  1  m2  m22
atom4  0  1  m4  m42
level 4:
atom5  0  1  m5  m52
atom6  0  1  m6  m62

```

We also output some statistics of the graph that has been created. These statistics are listed in Table 5.5:

Table 5.5: The statistics of the graph

statistics	definition
<i>No_atom1</i>	Total number of atomic nodes in the graph (including dummy nodes);
<i>No_atom2</i>	Total number of atomic nodes in the graph (excluding dummy nodes);
<i>No_comp</i>	Total number of composite nodes in the graph (of all topologies);
<i>No_FJ</i>	Number of composite nodes with Fork-join topology in the graph;
<i>weight_FJ</i>	Weight of Fork-join composite node ($weight_FJ = No_FJ / No_comp$);
<i>L</i>	Average number of Fork-join composite nodes an atomic node belongs to.

The last statistic *L* is defined as follows:

For each atomic node *i* (excluding dummy node), go up from it's position in the tree structure through until to the root node, *L_i* is the number of the Fork-Join composite nodes it passed. The average number of the whole graph is $L = (\sum L_i) / No_atom$ (*i* from 1 to *No_atom*). The larger the value of *L*, means more atomic node are contained in Fork-join subgraph, and Fork-join subgraph contains Fork-join subgraph.

These statistics are for later use when analyzing the accuracy of the approximation.

Execution of **autogene** will generate one **simu_parameter** and one **appr_parameter** once, that is to say one execution will create one instance. Our purpose is to generate several different instances of the same structure size. Therefore a shell script

was written for this purpose. It can run n times (n is specified by user), each time with different integer number inputted as a random number seed to create different instances. Every time it ran the **autogene**, the random number seed is increased by 1000000, and it will copy **sim_parameter** and **appr_parameter** into other new files. So after the shell script complete, n different parameter files are created for simulation and approximation respectively. We can start evaluation at this time.

5.4 Summary

In this chapter, a software system called **autogene** is described. It can be used to create task graph automatically for systematic evaluation of the approximation.

As the prerequisite, the task graph notation and tree structure notation and their parameters are introduced.

Autogene creates a task graph with randomly chosen structures based on the input parameters, convert it into tree-structure notation and task graph notation, obtain the parameters for approximation and simulation using, and output these parameters into **appr_parameter** and **simu_parameter** accordingly.

By using **autogene**, a large number of task graph instances can be generated according to the specified parameters. The generated graphs can be used to evaluate the approximation in a systematic way. **Autogene** eliminates large amount of manual work.

Chapter 6 Evaluation of Approximation

In this chapter, we evaluate the accuracy of the three-point approximation when used for large task graphs. First, different sets of cases are described; for each case, the results of the approximation are compared with the simulation results; then the accuracy of different classes is analyzed. We also discuss the root cause of certain errors which are observed when processing large graphs.

6.1 Different Classes of Cases Under Testing

As described previously, a task graph can be generated to represent certain structure, in which the mean value of delay of each atomic node is specified with different distribution function of delay. To evaluate the impact of the distribution shape (as expressed for example in the *CV* value) on the accuracy of three-point approximation, all the cases are categorized into the following four groups, which are then evaluated and compared individually. The cases of the four groups have the same structure and mean delay value, except that the distribution function of each group differs.

- the first group are cases with exponential distribution delay time for all tasks;
- the second group are cases with uniform distribution delay time for all tasks;
- the third group are cases with Erlangian distribution delay time for all tasks;
- the fourth and last group are cases with hyperexponential distribution delay time for all tasks, that is they have large variance value.

Since the structures of the task graphs in each group of 100 cases are same, the structure will be described for one group only.

Each group has 100 cases with different task graph sizes, with 10 cases for each size

as shown in Table 6.1:

Table 6.1: The structure properties of the 100 cases with single distribution

case number	<i>max_level</i>	<i>max_children</i>	average # of atomic nodes	largest # of atomic nodes	smallest # of atomic nodes
case1--10	3	2	5.7	6	5
case11--20	3	5	23.6	38	12
case21--30	3	10	133.6	243	44
case31--40	4	2	5.8	6	5
case41--50	4	5	46.6	114	5
case51--60	4	10	495.3	835	140
case61--70	5	2	5.8	6	5
case71--80	5	5	79.9	147	9
case81--90	6	2	5.6	6	5
case91--100	6	5	211.2	535	47

The response time of these cases ranged from 8 to 480 and the total number of atomic nodes range from 6 to 856 (or, excluding the dummy nodes, from 5 to 835).

As stated before, when atomic node appear at outer level, the graph can not be extended into inner level as expected, so that the actual graph may not have the maximum size. Although the parameters *max_level* and *max_children* are used, the graph generated does not always have the specified level, especially for small *max_children* value and *max_level* value. Among the 100 cases, when *max_level* was defined as being 3, 4, 5, 6 respectively, when the *max_children* is defined as 2, all the cases actually have structure with *max_level* = 3, *max_children* = 2. Though this does not affect the evaluation much. For the topology of a node in an outer level, the probability of it being made atomic is larger than when the level is small. This is specified by the construction algorithm described in Chapter 5, when the topology of a node is decided, we choose atomic or composite according to its *existing_level* with certain probability. In most instances, the graph

generated have the number of levels specified by *max_level*.

Remember that **autogene** can also generate task graph with mixed distributions of delay. In section 6.3, we will study 100 mixed-distribution cases.

6.2 Results Analysis

The approximation values for response time of a task graph is compared with the simulation value. The accuracy of simulation is such that a 95% confidence interval is less than 1% of the average. The percentage error for each group is shown in a table and a histogram.

6.2.1 Exponential Distribution

Figure 6-1 Histogram of Errors of Mean for 100 Random Generated Task Graph with Exponential Distribution

In each of the 100 cases, the average approximation error was found from about 20,000 simulated executions of the task graph, to find its overall delay. The results of the errors of mean of the 100 cases in exponential group are in Table A.4 in the Appendix and Figure 6-1 above. We can observe from the histogram (Figure 6-1) that the errors are distributed evenly near zero. Two extreme cases with large errors are negative. The mean absolute error (MAE) of mean delay over the 100 cases is 0.65%. Most of the absolute percentage error of mean delay in exponential group are within 2% (79 cases within 1%, 17 cases between 1% and 2%). Only four cases have absolute error between 2-5%. The smallest error is 0.00% (with two digit accuracy) which can be ignored. The largest error is -4.41% appeared in case 52, which has 420 nodes (6 dummy nodes) in 4 levels, the number of child nodes that a composite node have can be as high as 10.

While the error of mean delay is not very large, the error of standard deviation (σ) of the overall delay is relatively larger, with the largest one being -73.56% (for case 57). This case has 856 (835 excluding dummy tasks) tasks in 4 levels. The *max_children* is 10. The MAE of σ is 11.49%.

6.2.2 Uniform Distributions

Figure 6-2 Histogram of Errors of Mean for 100 Random Generated Task Graph with Uniform Distribution

Accuracy of the approximation for uniform distribution is better than that for exponential distribution, they are shown in Figure 6-2 above and Table A.5 in Appendix. From the histogram (Figure 6-2) we can see that the errors are more close to zero compared with exponential group. The overall performance of uniform is better than exponential cases with MAE equal to 0.52%. There are 95 cases having percentage error of mean delay within 2% (87 cases within 1%, 8 cases between 1-2%). Only one case has error larger than 5%, that is case 52 with error of -7.09%. The largest error is bigger than in the exponential cases.

The error of standard deviation (σ) of delay is once again larger than error of mean delay. The largest one is -90.27% for case 57. Remember that this case has the largest σ error in exponential group too. It has 856 nodes (835 nodes excluding dummy nodes) in 4 levels. The *max_children* is equal to 10. The MAE of σ is 25.16%.

6.2.3 Erlangian Distributions

Figure 6-3 Histogram of Errors of Mean for 100 Random Generated Task Graph with Erlangian Distribution

The comparison results of 100 Erlangian cases are shown in Table A.6 in Appendix and in Figure 6-3 as histogram. Accuracy of the approximation for Erlangian distribution cases are even better than for uniform distribution cases. The MAE is 0.44%, it is the best one among the four groups. The same as in uniform cases and exponential cases, the largest

error appeared for case 52, and is negative too, -8.65%. The smallest error is 0.00% for several cases (with two digit accuracy).

The error of standard deviation (σ) of delay is also larger compared with error of mean of delay. The largest one is -93.93%, also for case 57 as in exponential group and in uniform group. The MAE of σ is 29.96%.

6.2.4 Hyperexponential Distribution

For hyperexponential distribution group, we study the cases with CV equal to 2, 3, 4 separately. All the graph structure are the same, except that the delay of each task is hyperexponentially distributed with different CV value. The comparison results are shown in Table A.7, A.8, A.9 respectively. They are also shown in Figure 6-4 to 6-6 as histogram for directly observation.

Figure 6-4 Histogram of Errors of Mean for 100 Hyperexponential Cases with $CV = 2$

Figure 6-5 Histogram of Errors of Mean for 100 Hyperexponential Cases with $CV = 3$

Figure 6-6 Histogram of Errors of Mean for 100 Hyperexponential Cases with $CV = 4$

The cases with hyperexponential distribution have larger errors than the previous cases. We can observe that when the CV value increases, the accuracy of the approximation decreases. The MAE of mean delay over the 100 cases are 2.26%, 3.22%, 3.99% respectively. Also, the larger the CV , the larger the number of errors bigger than 5%. This results is in accordance with the results in Chapter 3, which showed that simple Fork-join cases with hyperexponential distribution of delay have larger error than cases with lower variance.

It is also found that the error of σ is smaller as the variance increases. The MAE of σ over the 100 cases are 11.81%, 10.32%, 8.49% respectively, we can see the trend towards smaller σ error when the CV increases.

From the histogram of the distribution of errors, we can see that almost all the errors are positive, except a few cases with negative error close to zero. This shows that the approximation overestimates the response time for hyperexponential distributions.

6.2.5 Across the Four Groups

From the previous sections we can see that when the approximation is used for cases with small *CV* value, it is pretty accurate. Most of the errors are near zero. Since the simulation have accuracy of 1% with 95% confidence interval, so the errors smaller than 1% can be considered as effectively zero. For most of cases, errors are below 1%.

The uniform group & Erlangian group are nearly same. They are more close with each other than with exponential group. Although errors of exponential group are not as close to zero as the other three groups, it is also true that most of the errors are below 1%.

But the approximation is not very accurate for the fourth group -- cases with hyperexponential distribution delay. The errors are much larger than those in the first three groups, and most of them are positive.

6.3 Mixed-Distribution Delay Cases

We will study the accuracy of the approximation when it is used for cases containing a mixture of different distributions. This is more close to reality. The distribution chosen for creating the task graphs are:

- hyperexponential distribution: 10%
- exponential distribution: 10%
- uniform distribution: 40%
- Erlangian distribution: 40%

The size of the 100 cases studied belong to 10 sets, each set contain 10 cases as in

Table 6.2:

Table 6.2: The structure properties of 100 cases with mixture distribution

case number	level	max_childre n	average # of atomic nodes	largest # of atomic nodes	smallest # of atomic nodes
case1--10	3	2	5.8	6	5
case11--20	3	5	21.6	45	12
case21--30	3	10	103.5	246	17
case31--40	4	2	5.7	6	5
case41--50	4	5	50	87	6
case51--60	4	10	567.1	1390	27
case61--70	5	2	6	6	6
case71--80	5	5	92.5	165	50
case81--90	6	2	6	6	6
case91--100	6	5	257.5	614	35

The compare results of the approximation and simulation are show as histogram in Figure 6-7 below and Table A.10 in the appendix:

Figure 6-7 Histogram of errors of 100 cases with mixed distribution of delay

The results of the mixed-distribution cases are not as good as for exponential cases, but is much better than for hyperexponential cases, with MAE of mean delay equal to 2.22%, MAE of σ equal to 21.96%.

6.4 Source of Large Errors

From the analysis of MAE and percentage error for each group, it is found that for the first three groups (exponential distribution, uniform distribution, and Erlangian distribution), the cases with large errors are same. Case43, case48, and case52 have relatively large error. Remember that each case has the same graph structure and mean

delays among the groups, so it can be concluded that the error may be related to the structure of a graph (including the mean delay values) as well as to its distribution of delay of each node. (The hyperexponential distribution is excluded).

The approximation needs the mean and variance of each branch of a Fork-join subgraph to estimate the mean and variance of this Fork-join subgraph, and this value may be used in turn to determine other Fork-join subgraph at the outer level which contains this Fork-join subgraph. So the error may transfer across the entire graph if it is a large one. Since the error in estimating the mean is relatively smaller than the error in σ , when a graph is large, the larger error in σ may affect the approximation more. In a large graph, the error of σ value will decrease the accuracy of the approximate calculation of response time of the graph. Since the error in σ observed is of the whole graph, instead of individual Fork-join subgraph, so there is no clear relationship between the error in σ and the error in the mean of delay. It is observed that all combinations of error of mean and error of σ existed. e.g. small error of mean with small error of σ ; small error of mean with large error of σ ; large error of mean with small error of σ ; large error of mean with large error of σ .

In previous experiments it was found that, in a simple Fork-join subgraph, when the number of branches increased, or when *CV* value of delay of each task increased, or when the means of each branches are close to each other, it will reduce the accuracy of the approximation. It is not known if the large error in large size structure graph is caused by these factor. So in the approximation calculation, when a Fork-join subgraph with branches' means close to each other (the largest value was within 20% of the smallest mean of the branches), the node was recorded. But it is found that both cases with large error of mean and cases with small error of mean have nodes been recorded. And some cases with large error do not have any such node recorded. So the close of mean value of Fork-join branches does not have significant role in decreasing the accuracy of the approximation for multiple level graphs.

Now that it should be noticed that although some cases are categorized into set with large error, it is only relatively large compare with other values. Actually error under 2% (even under 5%) is acceptable. Several cases do have higher error, they appeared for both distribution cases. Such as case52, the errors are -4.411%, -7.09%, -8.65% for exponential, uniform and Erlangian distribution respectively.

Therefore, the three-point approximation method is accurate enough for estimate the response time for most task graph with *CV* value smaller than or equal to one.

But for cases with large *CV* value, such as hyperexponential distribution cases, the three-point approximation method has less accuracy.

It is worth noting that the autogene program outputs certain statistics on the task graph it created. The two statistics interested are:

- *weight_FJ*: Fraction of Fork-join composite node among all the composite nodes;
- *L*: Average number of Fork-join composite node an atomic node belongs to.

The first statistics is not related closely to the error. This is because although Fork-join composite node may occupant large parts of Composite nodes, if the number of children nodes of Fork-join node is small, then it will not affect the entire graph much.

On the other side, if *L* is large, that means the atomic node may stay in the level where it will be aggregated by Fork-join calculation for several times to get the response time of the entire graph. The more times to aggregated, the more error will be carried into the calculation of the response time of entire graph.

Table 6.3: Statistics of the structure of the 100 graphs with single distribution

case	<i>No_atom1</i>	<i>No_atom2</i>	<i>No_comp</i>	<i>No_FJ</i>	<i>weight_FJ</i>	<i>L</i>
1	6	6	4	1	0.25	0.67
2	6	6	4	0	0.00	0.00
3	6	6	4	1	0.25	0.67

Table 6.3: Statistics of the structure of the 100 graphs with single distribution

case	No_atom1	No_atom2	No_comp	No_FJ	weight_FJ	L
4	6	6	4	1	0.25	0.67
5	5	5	3	1	0.33	0.60
6	5	5	3	1	0.33	0.60
7	6	6	4	0	0.00	0.00
8	5	5	3	1	0.33	0.60
9	6	6	4	0	0.00	0.00
10	6	6	4	1	0.25	0.67
11	16	15	7	3	0.43	1.20
12	16	16	8	3	0.38	1.25
13	43	38	13	5	0.38	1.32
14	13	12	6	0	0.00	0.00
15	21	20	8	3	0.38	1.15
16	29	27	9	2	0.22	0.33
17	28	27	9	4	0.44	1.33
18	24	22	9	2	0.22	0.41
19	30	27	9	4	0.44	1.41
20	33	32	10	2	0.20	1.09
21	110	103	23	11	0.48	1.54
22	189	171	34	19	0.56	1.60
23	84	76	19	8	0.42	0.42
24	48	44	10	3	0.30	1.20
25	152	138	29	13	0.45	0.56
26	252	232	43	24	0.56	1.62
27	153	139	29	14	0.48	1.60
28	114	104	21	12	0.57	1.67
29	94	86	16	9	0.56	1.62
30	267	243	49	29	0.59	0.69

Table 6.3: Statistics of the structure of the 100 graphs with single distribution

case	No_atom1	No_atom2	No_comp	No_FJ	weight_FJ	L
31	6	6	4	1	0.25	0.67
32	6	6	4	0	0.00	0.00
33	6	6	4	1	0.25	0.67
34	5	5	3	0	0.00	0.00
35	5	5	3	1	0.33	0.60
36	6	6	4	1	0.25	0.67
37	6	6	4	1	0.25	0.67
38	6	6	4	1	0.25	0.67
39	6	6	4	0	0.00	0.00
40	6	6	4	1	0.25	0.67
41	82	79	31	6	0.19	0.54
42	121	114	42	10	0.24	1.77
43	49	48	19	4	0.21	1.46
44	11	11	6	2	0.33	1.27
45	29	28	11	2	0.18	1.18
46	41	41	15	1	0.07	0.17
47	67	65	23	3	0.13	1.40
48	61	59	22	2	0.09	0.39
49	16	16	7	2	0.29	1.44
50	5	5	3	0	0.00	0.00
51	716	702	138	16	0.12	1.61
52	420	414	87	7	0.08	1.33
53	606	590	118	13	0.11	1.57
54	141	140	26	3	0.12	1.48
55	182	178	40	5	0.12	0.74
56	834	808	171	21	0.12	1.54
57	856	835	164	20	0.12	0.74

Table 6.3: Statistics of the structure of the 100 graphs with single distribution

case	No_atom1	No_atom2	No_comp	No_FJ	weight_FJ	L
58	813	794	163	17	0.10	1.52
59	246	240	46	4	0.09	1.38
60	258	252	51	9	0.18	1.76
61	6	6	4	1	0.25	0.67
62	6	6	4	0	0.00	0.00
63	6	6	4	1	0.25	0.67
64	5	5	3	0	0.00	0.00
65	6	6	4	1	0.25	0.67
66	5	5	3	0	0.00	0.00
67	6	6	4	1	0.25	0.67
68	6	6	4	1	0.25	0.67
69	6	6	4	1	0.25	0.67
70	6	6	4	1	0.25	0.67
71	159	147	63	22	0.35	1.10
72	101	89	40	14	0.35	1.66
73	81	75	30	9	0.30	0.72
74	77	72	31	7	0.23	1.49
75	9	9	6	2	0.33	1.11
76	106	101	42	14	0.33	2.10
77	113	105	42	17	0.40	2.09
78	55	51	21	8	0.38	0.84
79	113	103	38	17	0.45	1.79
80	53	47	20	7	0.35	0.96
81	6	6	4	1	0.25	0.67
82	5	5	3	0	0.00	0.00
83	4	4	2	1	0.50	0.50
84	6	6	4	1	0.25	0.67

Table 6.3: Statistics of the structure of the 100 graphs with single distribution

case	No_atom1	No_atom2	No_comp	No_FJ	weight_FJ	L
85	6	6	4	1	0.25	0.67
86	6	6	4	1	0.25	0.67
87	5	5	3	0	0.00	0.00
88	6	6	4	1	0.25	0.67
89	6	6	4	0	0.00	0.00
90	6	6	4	1	0.25	0.67
91	562	535	210	39	0.19	2.17
92	48	47	20	4	0.20	1.87
93	224	214	88	18	0.20	2.10
94	319	302	120	23	0.19	1.40
95	166	159	60	12	0.20	2.50
96	153	148	67	9	0.13	0.61
97	203	193	80	16	0.20	2.25
98	228	222	93	12	0.13	0.74
99	165	159	58	9	0.16	2.13
100	139	133	50	10	0.20	2.38

It is found that the rate of Fork-join composite node over composite node is not related to the large error. It is true that the atomic node which has relation with Fork-join node play the role in the accuracy of the approximation. It is found that both case43($L=1.46$), case52 ($L=1.33$), case58 ($L=1.52$), case94 ($L=1.4$) which have relatively large error have L of about 1.4-1.5.

Another interesting point is that when both the three groups have large σ errors, the hyperexponential group is a little better in terms of error of σ . The errors of σ are increased corresponding to the sizes of the graphs. This is because in a large graph, the error of mean and error of σ of the approximation will transfer to the outer levels to affect the accuracy for the whole graph.

Chapter 7 Cases With Contention

In a parallel processing environment, we assume that each processor executes a certain set of tasks. When more than one task is invoked, the tasks assigned to one processor will compete for the processor resource. Then each processor can be modeled as a queueing system with several tasks, each with a specified service time. From the logical structures among tasks and the task invocation rate, the invocation rate of each individual task on the processor can be determined. Contention has the effect of increasing the time to execute a task, and modifying its distribution.

In this chapter, an approximation method for systems with such resource contention among tasks are discussed. A simulation model is described. The task waiting time due to the resource contention is added into each task's execution time. The response time for the task graph is calculated by using the three-point approximation. The results of approximation and simulation for task graph response time are compared. Different task invocation rates and task allocation scenarios are also studied. The method is based on that of Chu et al. [6], and so is the example used to test it.

7.1 The Waiting Time Without Logical Resource Contention

For tractability of the waiting time calculation, we make the following assumption also made by Chu et al [6]:

- (i) task invocations have Poisson arrivals at the processors,
- (ii) the scheduling discipline of the processor is first-come-first-serve (FCFS)

As we described previously, the overall task response time consists of task waiting times, task execution demands, and synchronization delays. Task waiting time is the

interval between the time it received all the messages from its predecessor(s) and its execution start time on a processor.

There is a set of processors, and task graphs are initiated by the arrival of jobs to the system with overall rate λ_T jobs/sec. Consider a processor (say processor m) that is assigned with a set I_m of distinct tasks. Let λ_i and t_i denote the invocation rate and execution time for task i , π_i is the probability of executing task i , where $i \in I_m$. Based on assumptions (i) and (ii), this queueing system is an extension of the regular FCFS M/G/1 queue with total arrival rate

$$\lambda_m = \sum_{i \in I_m} \lambda_i$$

Since each task's invocation rate is a fraction π_i of the arrival rate of an entire task graph λ_T :

$$\lambda_i = \lambda_T \pi_i$$

so that

$$\lambda_m = \sum_{i \in I_m} \lambda_i = \lambda_T \sum_{i \in I_m} \pi_i \quad (7.1)$$

For the M/G/1 queue, the first two moments of the task waiting time from the invocation arrival to its first task execution start point are quote in [17] as:

$$\overline{W}_m = \frac{\lambda_m \cdot \overline{t_m^2}}{2(1 - \rho_m)} = \frac{\lambda_m \cdot \overline{t_m^2}}{2(1 - \lambda_m \overline{t_m})} \quad (7.2)$$

and

$$\overline{W}_m^2 = 2(\overline{W}_m)^2 + \frac{\lambda_m \overline{t_m^3}}{3(1 - \rho_m)} \quad (7.3)$$

From (7.2) and (7.3) we obtain the variance of task i waiting time at processor m as:

$$Var(W_m) = \sigma_m^2 = \overline{W_m^2} - (\overline{W_m})^2 = (\overline{W_m})^2 + \frac{\lambda_m \overline{t_m^3}}{3(1-\rho_m)} \quad (7.4)$$

Here $\overline{t_m}$ and $\overline{t_m^2}$ are the first two moments of the weighted average service time at processor m , the k th moments of service time can be obtained from (7.5):

$$\overline{t_m^k} = \frac{\sum_{i \in I_m} \overline{t_i^k} \pi_i}{\sum_{i \in I_m} \pi_i} \quad k = 1, 2, 3 \quad (7.5)$$

If the distribution of t_m is exponential we would have $\overline{t_m^2}$ from $\overline{t_m}$:

$$\overline{t_m^2} = (\overline{t_m})^2 + \sigma_m^2 = (\overline{t_m})^2 + (\overline{t_m})^2 = 2(\overline{t_m})^2 \quad (7.6)$$

The mean cube value in (7.5) is not available, so it is approximated. Based on the assumption of exponential execution time, the mean cube of execution time of task i $\overline{t_i^3}$ can be obtained from $\overline{t_i}$:

$$\overline{t_i^3} = 3\overline{t_i^2}\overline{t_i} = 6(\overline{t_i})^3 \quad (7.7)$$

therefore the weighted average of cube service time at processor m can be obtained by substituting (7.7) into (7.5).

$$\overline{t_m^3} = \frac{\sum_{i \in I_m} \overline{t_i^3} \pi_i}{\sum_{i \in I_m} \pi_i} = \frac{6 \sum_{i \in I_m} (\overline{t_i})^3 \pi_i}{\sum_{i \in I_m} \pi_i} \quad (7.8)$$

Substitute $\overline{t_m^2}$ and $\overline{t_m^3}$ into (7.2) and (7.4), we can obtain the mean and variance of the

waiting delay at processor m .

The average response time for task i , can be written as:

$$\bar{T}_i = \bar{W}_m + \bar{t}_i \quad (7.9)$$

Since W_m and t_i are independent random variables, the variance of the response time for task i is the sum of variances of each component in (7.10):

$$\sigma^2 [T_i] = \sigma_m^2 + \sigma_i^2 \quad (7.10)$$

where σ_m^2 is given in (7.4) and σ_i^2 is the variance of execution demand for task i . When there is processor contention, the quantities $\bar{T}_i$ and $\sigma^2 [T_i]$ can be used in place of $\bar{t}_i$ and σ_i^2 .

7.2 Task Response Time Model With Logical Resource Contention

Logical resource contention occurs frequently in a distributed system. In tightly coupled systems, resources, such as memory, are shared and accessed by different processors. In loosely coupled systems, processors may access shared data files as common resources. Let resource contention delay be the time from the request of a resource until it is granted. This delay is affected by the quantity of resource requested, frequency and duration of possession.

We assume that there is just one logical resource. Task invocations which require resource first enter the resource queue to acquire the resource. Resource is granted to the task on a FCFS basis. If there is resource available, it is allocated to the requesting task invocation, which leaves the resource queue without delay. However, if there is no resource available, the requesting task has to wait at the resource queue until one is available. These tasks enter their assigned processor queues after acquiring the resource. These task

invocations possess the resource until their execution completes. At those points in time, the resource is returned, and become available for other tasks. For task invocations which do not require resource, they join their assigned processor queue directly.

Consider the component of response time for task i processed at processor m which needs a logical resource:

$$\bar{T}_i = \bar{W}_m + \bar{W}_r + \bar{t}_i \quad (7.11)$$

$$\sigma^2 [T_i] = \sigma_m^2 + \sigma_r^2 + \sigma_i^2 \quad (7.12)$$

We consider different waiting model for those tasks do need the resource and those do not need the resource.

Since there is only one resource, only one task holding the resource token can be queued at any processor queue at any time. Assume a set of tasks I_{res} need the resource, λ_i is the invocation rate at the processor for task $i \in I_{res}$, we can obtain the overall arrival rate at the resource queue:

$$\lambda_r = \sum_{i \in I_{res}} \lambda_i \quad (7.13)$$

Similarly to (7.2) and (7.4), the mean and variance of waiting delay $\bar{W}_r$ and σ_r^2 at the resource is:

$$\bar{W}_r = \frac{\lambda_r \cdot \bar{t}_r^2}{2(1 - \rho_r)} = \frac{\lambda_r \cdot \bar{t}_r^2}{2(1 - \lambda_r \bar{t}_r)} \quad (7.14)$$

$$\sigma_r^2 = \overline{W_r^2} - (\bar{W}_r)^2 = (\bar{W}_r)^2 + \frac{\lambda_r \bar{t}_r^3}{3(1 - \rho_r)} \quad (7.15)$$

2 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

t_r in (7.14) and (7.15) is the service time of the resource, which is a weighted combination of the processor waiting time plus execution time for the tasks which use the resource. For task $i \in I_{res}$, this is $\overline{W_{m(i)}} + \bar{t}_i$, where $m(i)$ is the processor for task i .

The one task which holds the resource behaves like a special class of customer to the processor queues, with only one member. We think of it as a closed class of job, visiting queues which are also used by an open class of job. The open class contains all the tasks that do not use the resource.

The closed class is treated as if the processors were slowed down by a factor $1/(1 - \rho_m')$ accounting for their utilization by the open class. The average value will be approximated by

$$(\overline{W_{m(i)}} + \bar{t}_i) = \frac{\bar{t}_i}{1 - \rho_{m(i)}'} \quad (7.16)$$

This approximation comes from product-form queueing networks with open classes, where ρ_m' is the utilization at processor m by the open class. It is simpler than the approximation used by Chu [6], and seems to be roughly as good.

$$\rho_{m(i)}' = \sum_{j \in I_{m(i)}, j \in I_{res}} \lambda_j \bar{t}_j$$

The weighted average service time of the resource (the average time a task will hold the resource) is then obtained from (7.17), in which $\overline{W_{m(i)}}$ and $\rho_{m(i)}'$ are functions of i , depends on which processor the task will be running on.

$$\bar{t}_r = \frac{\sum_{i \in I_{res}} \lambda_i (\overline{W_{m(i)}} + \bar{t}_i)}{\sum_{i \in I_{res}} \lambda_i} = \frac{\sum_{i \in I_{res}} \lambda_i \frac{\bar{t}_i}{(1 - \rho_{m(i)}')}}{\sum_{i \in I_{res}} \lambda_i} \quad (7.17)$$

The higher moments of t_r are quite complicated; if it were exponentially

distributed, they would be:

$$\bar{t}_r^2 = 2 (\bar{t}_r)^2$$

$$\bar{t}_r^3 = 3 \bar{t}_r \cdot \bar{t}_r^2 = 6 (\bar{t}_r)^3$$

and this will be assumed. These functions are used in (7.14) and (7.15) to give the mean and variance of resource delay.

From theorem of PASTA (Poisson Arrivals See Time Averages), $\bar{n}_{im}$ is probability that task i is in queue of processor m :

$$\bar{n}_{im} = \bar{W}_{m(i)} \lambda_i \quad \text{for } i \text{ is a resource holder}$$

$$\bar{n}_{im} = \bar{W}_m \lambda_i \quad \text{for a non resource hold task } i$$

and the mean time to finish the task in service equal to

$$(\text{prob of task } i \text{ in service}) \cdot \left(\frac{\bar{t}_i^2}{2\bar{t}_i} \right) = \lambda_i \cdot \bar{t}_i \cdot \left(\frac{\bar{t}_i^2}{2\bar{t}_i} \right) = \frac{\lambda_i \bar{t}_i^2}{2}$$

So that for the tasks that do not need the resource, the waiting time can be expressed as follows:

$$\bar{W}_m = \sum_i \bar{n}_{im} \bar{t}_i + \sum_i \frac{\lambda_i \bar{t}_i^2}{2} = \sum_{i \in I_{res}} (\bar{W}_{m(i)} \lambda_i) \bar{t}_i + \sum_{i \in I_{non}} (\bar{W}_m \lambda_i) \bar{t}_i + \sum_i \frac{\lambda_i \bar{t}_i^2}{2}$$

The first two parts are waiting time for the resource holding tasks and non resource holding tasks in the queue to be finished. The last part is waiting time for the task in service to be finished, this task can be either resource holding or non resource task. From (7.16)

$$\bar{W}_{m(i)} = \frac{\rho'_{m(i)} \bar{t}_i}{1 - \rho'_{m(i)}}$$

so:

$$\overline{W}_m = \sum_{i \in I_m} \frac{\rho'_{m(i)} (\bar{r}_i)^2 \lambda_i}{1 - \rho'_{m(i)}} + \overline{W}_m \sum_{i \in I_m} \lambda_i \bar{r}_i + \sum_i \frac{\lambda_i \bar{r}_i^2}{2} \quad (7.18)$$

Solve equation (7.18), we got:

$$\overline{W}_m = \frac{\sum_{i \in I_m} \frac{\rho'_{m(i)} (\bar{r}_i)^2 \lambda_i}{1 - \rho'_{m(i)}} + \sum_i \frac{\lambda_i \bar{r}_i^2}{2}}{1 - \sum_{i \in I_m} \lambda_i \bar{r}_i} \quad (7.19)$$

The variance of the waiting time at processor m was obtained using the following equation:

$$\sigma_m^2 = \left(\frac{2}{\rho_m} - 1 \right) (\overline{W}_m)^2 \quad (7.20)$$

which is the variance of the waiting time at a M/M/1 queue with traffic intensity ρ_m .

7.3 Simulation Model

In the original simulation model, each spawned task has its own dedicated processor, and hence does not experience any queuing delays.

When more than one task graph instances is invoked, there will be contention for processor and logical resource between tasks that belong to different task instances, therefore the simulation model described in Chapter 4 needs to be modified.

We set up an open QN simulation model to deal with contention cases.

In the **genesis** task, several processor node are built to represent processors. The **simulated task** is similar to what is described in chapter 4. Each task will be prepared to

execute once it receive all the message(s) it needs, and will send message(s) to its successor(s) after finishing the execution.

The difference is that it is an open QN model here, so a particular **generator** task is created to generate task invocation according to specified arrival time interval. When the last task in the task graph finishes execution, it will not send message to first task as in the original simulation model described in Chapter 4. Instead, it will record the time, and quit. So the replication of execution of task graph is based on the **generator** task to create the task invocation.

Each **simulated task** will not consume CPU time itself. Since there will be contention for processor, so once a task is prepared to execute, it was send to the processor node. So the task will be queued at the processor node. The **processor** task is designed so that it receives message, then consume certain duration according to the task's execution demand. After that, it will send a message to the task indicate its completion, then receive next message in the queue. When the **simulated task** receive the message from the **processor** task indicating the completion, it will send message(s) to its successor(s).

When resource contention is introduced, when a task receives all the message(s) it needs, if it does not need resource for execution, it will go straight to the **processor** node, and queue there waiting for execution. If it does need resource, it will send message to the **resource** task first, until it acquires the resource will it be able to queue at the **processor** node. And after its execution, it will return the resource by sending message to **resource** task, and at the same time, sends message(s) to its successor(s) to indicate its completion.

The **resource** task is designed to receive requests, and if it is available, send an acknowledge message to the requested task. If not, the request message(s) will be queued. When a resource is returned by receiving message from the releasing task, it will check the request queue, and acknowledge the first task in the queue.

7.4 Model Validation

7.4.1 The Example Case for Contention Study

To validate the new task response time model with contention, we apply it to a distributed system with two processors. The task graph for the application task is exactly the one used as example in Chapter 2 (Figure 2-4), and is also the example case studied in Chu's paper [6]. The processors process task invocations in a first-come-first-serve (FCFS) order. Table 7.1 shows the average task execution times, and two different task assignments. To examine the effect of resource contentions on task response time, we first apply the model to the system by ignoring the resource contention, there are only processor contention alone. Then we study the same system with resource contentions in section 7.4.3.

Table 7.1: Task execution times and assignment for the contention example

task	execution time	task assignment 1	task assignment 2
1	5.0	CPU1	CPU1
2	1.0	CPU2	CPU2
3	1.0	CPU2	CPU1
4	5.0	CPU2	CPU1
5	5.0	CPU2	CPU2
6	1.0	CPU1	CPU1
7	5.0	CPU1	CPU2
8	1.0	CPU2	CPU2

In Figure 2-4, $p_1 = 0.6$, $p_2 = 0.4$.

Task execution times in Table 7.1 are exponentially distributed.

7.4.2 Results Without Resource Contention

This section considers a system with processor contention but no use of the logical resource.

For the case of processor contention, based on the task invocation rates and task execution times, the mean and variance of new task delay is obtained from (7.9) and (7.10), which are used to calculate the response time of the task graph. The task response time is obtained by aggregating the task graph by the method of Chapter 2.

For task assignment 1, the average task response times with processor contention alone are shown in Figure 7-1 as a function of task invocation rates λ_T .

Figure 7-1 Average Task Response Time for Task Assignment 1

Value of response time and waiting delay at processors for approximation and simulation are shown in Table 7.2 and Table 7.3 below:

Table 7.2: Response time for task assignment 1

invocation-rate	simulation mean	approximation mean	error of mean
65	27.14	26.13	-3.74%
75	31.35	30.16	-3.80%
95	44.05	43.92	-0.03%
105	54.90	57.22	4.24%
110	63.13	67.48	6.90%

Table 7.3: The comparison of waiting delay for task assignment 1

invo-rate	w_{1s}	w_{1a}	σ_{1s}^2	σ_{1a}^2	w_{2s}	w_{2a}	σ_{2s}^2	σ_{2a}^2
65	4.14	3.81	54.36	45.84	2.19	1.95	21.97	12.89
75	5.65	5.13	84.63	68.45	2.88	2.58	30.95	18.69
95	10.11	9.73	184.65	174.7	4.98	4.63	63.7	43.07
105	14.10	14.32	316.26	322.97	6.67	6.47	97.43	72.08
110	17.19	17.99	430.90	471.63	7.90	7.81	124.04	97.48

w_{ms} : simulation value of waiting delay at processor m

w_{ma} : approximation value of waiting delay at processor m

σ_{ms}^2 : simulation value of variance of waiting delay at processor m

σ_{ma}^2 : approximation value of variance of waiting delay at processor m

We can see from Table 7.3 that the approximate mean waiting time at processor m is close to the value of simulation. But the approximate variance of waiting time is less accurate, especially for processor 2.

It is found that the approximation of both the mean and variance of the waiting time increased faster than the simulation value. This is why the approximation of response time at high invocation rate will over-estimate the response time.

Task assignment 1 yields about the same loading on both processors, while assignment 2 allocates almost twice as much processing load to processor 1 as to processor 2. Task response time for task assignment 2 are shown in Figure 7-2 as a function of task invocation rates.

Figure 7-2 Average Task Response Time for Task Assignment 2

The value of response time and waiting time at processors for assignment 2 are

shown in Table 7.4 and Table 7.5 below:

Table 7.4: Response time for task assignment 2

invocation-rate	simulation mean	approximation mean	error of mean
80	44.51	38.47	-13.58%
85	51.05	44.07	-13.68%
90	59.61	51.90	-12.94%
95	70.09	63.70	-9.12%

Table 7.5: The comparison of waiting delay for task assignment 2 (unbalanced)

invo-rate	w_{1s}	w_{1a}	σ_{1s}^2	σ_{1a}^2	w_{2s}	w_{2a}	σ_{2s}^2	σ_{2a}^2
80	9.30	8.72	179.74	140.00	4.61	1.89	64.80	12.42
85	11.69	10.88	254.63	198.27	5.21	2.12	78.99	14.38
90	14.98	13.96	390.07	297.31	5.83	2.37	92.70	16.69
95	19.15	18.69	562.24	486.42	6.54	2.65	108.90	19.41

From Table 7.5 we see that the estimated mean waiting time is close to the simulation value for processor 1, while estimated mean waiting time for processor 2 is much less than the simulation value. The same for the variance value. We should note that processor 1 has twice as much load as processor 2. From the simulation value, it shows this does not make the wait time at processor 2 much better. Instead, our approximation underestimates the waiting time because the processor workload plays a important role in the calculation. This underestimation of waiting time make the calculation of the response time of the entire task graph be less than the simulation value for assignment 2.

We can see from Figure 7-1 and Figure 7-2 that when the task assignment for processors is balanced, the approximation result is more accurate than for unbalanced task assignment. Under both situation, the approximation slightly underestimates the response

time. But when task invocation rate increased, the approximation response time is increasing faster than the simulation value. So at high invocation rate for task assignment 1, the approximation response time is higher than the simulation results. We also note that for a given task invocation rate, the task assignment which balances the load among processors yields better response time.

7.4.3 Results With Resource Contention

This section consider a case with processor contention and logical resource contention.

For the case with resource contention, we assume that there is only one resource and it is requested by tasks $T1$, $T2$, $T4$, $T8$. The mean and variance for each task were obtained from equations (7.11) and (7.12).

For task assignment 1, the average task response times with resource contention are shown in Figure 7-3 as a function of task invocation rates as follows:

Figure 7-3 Average Task Response Time for Task Assignment 1
With Resource Contention

The value of response time of the task graph and waiting time at processors and resource for assignment 1 are shown in Table 7.6 and Table 7.7 below:

Table 7.6: Response time for task assignment 1 with resource contention

invocation-rate	simulation mean	approximation mean	error of mean
57	50.6468	46.0511	-9.07%
61	61.0209	56.0738	-8.11%
65	75.1777	72.1471	-4.03%
68	93.1535	92.7252	-0.45%

Table 7.7: The comparison of waiting delay for task assignment 1 with resource contention

invo-rate	w_{1s}	w_{1a}	σ_{1s}^2	σ_{1a}^2	w_{2s}	w_{2a}
57	1.41	5.87	12.99	59.07	0.73	5.07
61	1.49	6.92	13.46	75.64	0.78	5.94
65	1.57	8.16	14.06	97.99	0.83	6.96
68	1.62	9.26	14.56	120.03	0.86	7.85
invo-rate	σ_{2s}^2	σ_{2a}^2	w_{rs}	w_{ra}	σ_{rs}^2	σ_{ra}^2
57	5.75	32.90	10.57	7.67	248.69	115.56
61	6.17	43.35	13.92	10.39	384.89	186.21
65	6.58	57.51	18.53	14.93	589.77	337.16
68	6.69	71.45	24.43	20.93	951.96	600.55

w_{rs} : simulation value of waiting delay at resource

w_{ra} : approximation value of waiting delay at resource

σ_{rs}^2 : simulation value of variance of waiting delay at resource

σ_{ra}^2 : approximation value of variance of waiting delay at resource

Table 7.7 shows that the approximation waiting time at processors and the value obtained from simulation differed very much. This is because in the simulation, the processor waiting delay is the average value of all the tasks, so it is small. While the approximation values for w_{1a} and w_{2a} reported in the table does not include the processor waiting time for those tasks using the resource. For those tasks do need resource, only resource waiting delay is calculated considering the factor of those non resource tasks. If we average the waiting delay at processor among all the tasks, the value will obviously be smaller.

This is also true for the value in Table 7.9 which contain the approximation and simulation waiting delay for task assignment 2.

When for task assignment 2 with resource contention, the comparison of response time between simulation value and the approximate value are shown in Figure 7-4 and Table 7.8 as follows:

Figure 7-4 Average Task Response Time for Task Assignment 2 With Resource Contention

Table 7.8: Response time for task assignment 2 with resource contention

invocation-rate	simulation mean	approximation mean	error of mean
50	35.4434	34.7060	-2.08%
57	44.5817	45.0792	1.12%
60	50.9632	51.9171	1.87%
67	71.8064	82.1259	14.37%

The estimated waiting time at processors and resource queue are compared with the simulation value in Table 7.9:

Table 7.9: The comparison of waiting delay for task assignment 2 with resource contention

invo-rate	w_{1s}	w_{1a}	σ_{1s}^2	σ_{1a}^2	w_{2s}	w_{2a}
50	0.52	5.70	3.74	53.60	1.36	2.97
57	0.59	8.03	4.24	91.49	1.52	3.78
60	0.62	9.33	4.51	117.18	1.59	4.17
67	0.69	13.48	4.86	220.36	1.73	5.24
invo-rate	σ_{2s}^2	σ_{2a}^2	w_{rs}	w_{ra}	σ_{rs}^2	σ_{ra}^2
50	13.75	13.07	5.74	5.02	96.90	61.56
57	15.48	19.36	8.60	8.00	175.87	123.87
60	16.11	22.92	10.62	10.08	244.80	178.06
67	17.36	34.03	17.36	20.15	531.73	563.65

From Figure 7-3 and Figure 7-4, we can see that when resource contention is introduced, the approximate response time for task assignment 1 is a little less than the simulation value. While the approximate response time for task assignment 2 is very close to simulation value, especially when task invocation rate is not very high.

When task invocation rate is high, the approximation response time increase faster than the simulation value, so at high invocation rate, the approximation value is very close to the simulation value for assignment 1. But for assignment 2, the approximation lost its accuracy when the task invocation rate is very high.

This is the same conclusion we obtained from the study of the situation when there is no resource contention exist: at higher task invocation rate, the approximation will overestimate the response time.

7.5 Summary

In this chapter we studied the cases with processor contention and logical resource contention. A model for estimate the contention delay has been presented. It considers queueing effects and resource contention. The response time of task graph can be obtained by aggregating the task graph with new task execution time -- add the components of waiting delay.

The simulation model has also been modified to accustom the contention situation. The results of the simulation model are used to evaluate the analytical results for task graph with contention.

Our estimated response times for the example task graph are within 7% and 14% of those from simulation for task assignments 1 & 2 without logical resource contention, respectively. This is very good agreement. This accuracy are also close to that of Chu's, 7% and 10% for task assignments 1 & 2 respectively, which are estimated from the graph in [6].

The estimate response times for the example task graph with logical resource contention are within 10% and 15% of those value obtained from simulation for task assignments 1 & 2, respectively. The estimation overestimates the response time when the task invocation rate is higher too. The accuracy of approximation with resource contention are a little worse than that of Chu's 6% and 7%, which are also estimated form the graph in [6].

Chapter 8 Conclusion

8.1 Summary of the Results

An approximation method to calculate the mean and variance of delay of a parallel segment of computation based on task graph -- **three-point approximation** -- is described and evaluated in this thesis. The approximation is simple, and the accuracy is as good as that in [6].

A software system **autogene** is generated to create different types of task graph. This software system makes it possible to evaluate the accuracy of three-point approximation in a systematic way.

A detailed simulation model is established to obtain the value considered as exact value to evaluate the results of approximation.

Simple Fork-join subgraphs (Chapter 3) and task graphs containing combination of Fork-join subgraph, sequential subgraph and alternate subgraph (Chapter 5) are studied separately. Same task graph structure with different distribution of delay are studied to see the impact of different shape distribution on the approximation method (Chapter 6).

For simple Fork-join subgraphs, it shows that 1) when the number of branches in the Fork-join subgraph is small, 2) when mean delays of each branch are not very close, 3) when CV value of delay is small, the approximation is accurate for the mean of response time of whole Fork-join subgraph. However, the variance accuracy is not as good as that of mean.

When the approximation is used to calculate the response time of an entire task graph containing combination of subgraphs, if the distribution of delay for each task is uniform or Erlangian, the approximation has better accuracy (0.52% and 0.44% MAE). While for exponential distribution, it also have high accuracy, the MAE over 100 cases is

0.65%. When for hyperexponential distribution, the accuracy decreased according to the increasing of CV value.

The evaluation shows that the approximation is good for small CV cases. While used for large CV cases, as though as the CV value does not exceed 5, the accuracy is still acceptable. But the error of σ is worse than that of mean.

When resource contention exists, a model to estimate waiting time is described in Chapter 7. There are two types of contention. One is processor contention, the other is logical resource contention. The model suggested in this thesis is much simpler than that in Chu et al's [6]. Different task assignments are studied, the approximation is more accurate when the system has balanced load over processors. The results also shows that the system with balanced load will yield better response time.

8.2 Future Work

The approximation is fast, but not entirely accurate for certain cases, especially for those cases with long tail -- large CV value of distribution. However, it is still good for solving performance measure of processes with parallel path. Some modification on the choice of three point values is needed to improve the accuracy.

For the resource contention case, the accuracy is not ideal. Further work might be able to improve the waiting time model for better results.

References

- [1] Gail Anderson, Paul Anderson. *"The Unix C Shell Field Guide."* Prentice-Hall, Englewood Cliffs, N.J. 07632, 1986
- [2] Vikvam Adve, Mary Vernon. *"The Influence of Random Delays on Parallel Execution Times."* Proceedings of 1993 ACM SIGMETRICS, Performance Evaluation Review, Vol. 21, No 1, pp61-77.
- [3] Francois Baccelli. *"Acyclic Fork-Join Queuing Networks."* Journal of the Association for Computing Machinery, Vol. 36 pp615-642, July, 1989.
- [4] Jerry Banks, John S. Carson. *"Discrete-event System Simulation."* Prentice-Hall, 1984
- [5] P. Courtois. *"Decomposability: Queueing and Computer System Applications."* New York: Academic, 1977.
- [6] Wesley Chu, Chi-Man Sit, Kin K. Leung. *"Task Response Time for Real-time Distributed Systems with Resource contention."* IEEE Transactions on Software Eng. Vol.17 pp1076-1092, October, 1991.
- [7] Wesley Chu, Kin Leung. *"Module Replication and Assignment for Real-time Distributed Processing Systems."* Proceedings of the IEEE. pp547-562, May, 1987.
- [8] Wesley Chu, Lance Lan. *"Task Allocation and Precedence Relations for Distributed Real-time Systems."* IEEE Transactions on Computers, Vol. c-36 pp667-679, June, 1987.
- [9] G. Estrin. *"A Methodology for Design of Digital System."* Proceeding of National Computer Conference, pp313-324, 1978
- [10] Erol Gelenbe *"Multiprocessor Performance"*, WILEY Series in Parallel Computing

- [11] Curtis E. Hrischuk. *"The Automatic Generation of Software Performance Models from a Prototype."* M. Eng thesis, Department of Systems and Computer Engineering, Carleton University, 1995
- [12] P. Heidelberger, K.S.Trivedi. *"Queueing Network Models for Parallel Processing with Asynchronous Tasks."* IEEE Transactions on Computers, Vol. c-31, pp1099-1109, November, 1982.
- [13] Philip Heidelberger, Kishor Trivedi. *"Analytic Queueing Models for Programs with Internal Concurrency."* IEEE Transactions on Computers, Vol. c-32 pp.73-82, January 1983.
- [14] A. Kapelnikov, R. Muntz, M,D, Ercegovac. *"A Modeling Methodology for the Analysis of Concurrent Systems and Computations."* Journal of Parallel and Distributed Computing, pp568-597, 1989.
- [15] A. Kapelnikov, R. Muntz, M,D, Ercegovac. *"A Methodology for Performance Analysis of Parallel Computations with Looping Constructs."* Journal of Parallel & Distributed Computing, pp105-120, 1992.
- [16] Brian W. Kernighan, Dennis M. Richie. *"The C Programming Language."* 2nd edition, Prentice-Hall, Englewood Cliffs, New Jersey 07632, 1988.
- [17] L. Kleinrock. *"Queueing Systems, Vol. I: Theory."* New York: Wiley, 1975.
- [18] Richard Karp, Raymond Miller. *"Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing."* SIAM J. APPL. MATH. Vol.14 pp1390-1411, November, 1966.
- [19] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *"Quantitative System Performance."* Prentice-Hall, Englewood Cliffs, N.J. 07632, 1984.
- [20] Jie Ma. *"Measurement and Performance Analysis of World Wide Web Applications."*

- M. Eng thesis, Department of Systems and Computer Engineering, Carleton University, January, 1996
- [21] Shrdhar Madala, James B. Sinclair. "*Performance of Synchronous Parallel Algorithms with Regular Structures.*" IEEE Transactions on Parallel and Distributed Systems, Vol. 2 pp105-116, January, 1991.
- [22] Victor Mak, Stephen Lunstrom "*Predicting Performance of Parallel Computations.*" IEEE Transactions on Parallel & Distributed System, Vol.1 pp. 257-270, July 1990.
- [23] J. Neilson. "*Parasol User's Manual.*" School of Computer Science, Carleton University.
- [24] J. Neilson. "*Parasol Tutorial.*" School of Computer Science, Carleton University.
- [25] Peterson "*Petri net Theory and the Modeling of Systems*", Prentice Hall Inc. 1981.
- [26] Dorina C. Petriu. "*Approximate Mean Value Analysis of Client-server Systems with Multi-class REquests.*" Report SCE-93-30, Department of Systems and Computer Engineering, Carleton University, 1993.
- [27] John T. Robison. "*Some Analysis Technique for Asynchronous Multiprocessor Algorithms.*" IEEE Transactions on Software Eng. Vol. SE-5 pp24-31, January, 1979.
- [28] Connie U. Smith. "*Performance Engineering of Software Systems*", Addison-Wesley Publishing Company, 1990.
- [29] R. A. Sahner, K.S.Trivedi. "*Performance and Reliability Analysis using Directed Acyclic Graphs.*" IEEE Transactions on Software Eng. Vol. SE-13 pp1105-1114, October, 1987.
- [30] Alexander Thomasian, Paul F. Bay. "*QN Models for Parallel Processing of Task Systems.*" Proceeding of 1983 International Conference on Parallel Processing, pp421-428, 1983

- [31] Alexander Thomasian, Paul F. Bay. "*Analytic Queueing Network Models for Parallel Processing of Task Systems.*" IEEE Transactions on Computers, Vol. c-35 pp1045-1054, December, 1986.
- [32] C.M.Woodside. "*A Three-View Model for Performance Engineering of Concurrent Software.*" IEEE Transactions on Software Eng. July, 1995
- [33] C.M. Woodside, S. Majumdar, J.E.Neilson, D.C. Petriu, J.A. Rolia, A. Hubbard, R.B. Franks. "*A Guide to Performance Modeling of Distributed Client-server Software Systems with Layered Queueing Networks.*" Report SCE-95-23, Department of Systems and Computer Engineering, Carleton University, November, 1995.
- [34] Hong Zhang. "*Approximation to Software Activity Graph.*" Project Report, Department of Systems and Computer Engineering, Carleton University, April, 1994.
- [35] C.M. Woodside, A. Hubbard, Private Communication, 1993.

Appendix

In this appendix, contain the table of compare results between approxiamtion and simulation for all the group of cases we studied.

Table A.1: Compare results of 80 Fork-join cases with small CV value ($CV \leq 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	1.500	1.500	0.03%	1.119	1.041	-7.00%
2	1.273	1.250	-1.84%	0.494	0.520	5.32%
3	1.140	1.125	-1.34%	0.227	0.260	14.45%
4	1.094	1.083	-0.95%	0.147	0.174	18.02%
5	1.056	1.050	-0.59%	0.086	0.104	21.09%
6	2.331	2.417	3.66%	1.914	1.801	-5.91%
7	2.083	2.056	-1.32%	0.942	0.970	2.97%
8	2.006	2.000	-0.28%	0.494	0.500	1.30%
9	2.000	2.000	0.01%	0.333	0.333	0.11%
10	2.000	2.000	-0.00%	0.200	0.200	0.08%
11	5.165	5.333	3.27%	4.903	4.714	-3.85%
12	4.999	5.000	0.01%	2.488	2.500	0.49%
13	5.000	5.000	-0.00%	1.251	1.250	-0.04%
14	4.998	5.000	0.04%	0.833	0.833	0.03%
15	4.999	5.000	0.01%	0.500	0.500	-0.06%
16	10.032	10.333	3.01%	9.867	9.690	-1.80%
17	9.999	10.000	0.01%	5.011	5.000	-0.22%
18	9.990	10.000	0.10%	2.497	2.500	0.11%
19	9.994	10.000	0.06%	1.667	1.667	0.04%
20	10.000	10.000	0.00%	1.001	1.000	-0.08%
21	2.283	2.192	-3.98%	1.211	0.988	-18.46%

Table A.1: Compare results of 80 Fork-join cases with small CV value ($CV \leq 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
22	1.629	1.596	-2.05%	0.474	0.494	4.19%
23	1.305	1.298	-0.57%	0.202	0.247	21.94%
24	1.200	1.199	-0.14%	0.127	0.165	29.60%
25	1.119	1.119	0.04%	0.073	0.099	36.02%
26	3.528	3.587	1.67%	2.003	1.662	-17.05%
27	2.608	2.640	1.23%	0.855	0.862	0.89%
28	2.198	2.210	0.57%	0.414	0.460	10.93%
29	2.089	2.092	0.13%	0.283	0.316	11.82%
30	2.025	2.025	-0.01%	0.180	0.196	8.78%
31	7.683	7.888	2.67%	4.855	4.212	-13.25%
32	5.989	6.042	0.89%	2.192	2.232	1.81%
33	5.270	5.278	0.14%	1.095	1.183	7.97%
34	5.100	5.093	-0.15%	0.759	0.814	7.25%
35	5.015	5.017	0.02%	0.482	0.488	1.15%
36	14.787	15.211	2.87%	9.729	8.495	-12.68%
37	11.745	11.817	0.62%	4.413	4.504	2.06%
38	10.445	10.434	-0.11%	2.230	2.402	7.72%
39	10.147	10.139	-0.08%	1.542	1.619	4.97%
40	10.021	10.017	-0.04%	0.978	0.986	0.77%
41	2.929	2.677	-8.62%	1.244	0.736	-40.84%
42	1.891	1.838	-2.76%	0.459	0.368	-19.85%
43	1.418	1.419	0.09%	0.188	0.184	-2.15%
44	1.271	1.280	0.64%	0.116	0.123	5.82%
45	1.159	1.168	0.72%	0.065	0.074	12.75%
46	4.557	4.483	-1.63%	2.071	1.384	-33.15%
47	3.035	3.092	1.89%	0.826	0.745	-9.78%
48	2.381	2.426	1.89%	0.380	0.415	9.19%

Table A.1: Compare results of 80 Fork-join cases with small CV value ($CV \leq 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
49	2.201	2.229	1.25%	0.254	0.292	15.04%
50	2.082	2.092	0.48%	0.159	0.186	16.80%
51	9.970	9.973	0.03%	5.002	3.855	-22.92%
52	6.965	7.067	1.47%	2.102	2.058	-2.06%
53	5.678	5.742	1.13%	0.997	1.107	11.06%
54	5.333	5.364	0.59%	0.676	0.769	13.75%
55	5.115	5.120	0.10%	0.428	0.483	12.92%
56	19.201	19.292	0.47%	9.933	7.888	-20.59%
57	13.637	13.804	1.22%	4.236	4.192	-1.04%
58	11.219	11.329	0.98%	2.018	2.243	11.14%
59	10.584	10.640	0.54%	1.375	1.548	12.61%
60	10.190	10.194	0.04%	0.874	0.978	11.91%
61	3.597	2.948	-18.06%	1.261	0.319	-74.73%
62	2.142	1.974	-7.87%	0.444	0.159	-64.14%
63	1.522	1.487	-2.29%	0.176	0.080	-54.73%
64	1.336	1.325	-0.84%	0.107	0.053	-50.46%
65	1.196	1.195	-0.08%	0.060	0.032	-46.59%
66	5.628	5.208	-7.46%	2.118	0.862	-59.29%
67	3.457	3.489	0.93%	0.803	0.517	-35.60%
68	2.568	2.642	2.89%	0.353	0.328	-7.15%
69	2.320	2.373	2.31%	0.232	0.247	6.11%
70	2.148	2.176	1.28%	0.142	0.166	16.72%
71	12.458	12.012	-3.58%	5.179	2.897	-44.05%
72	7.966	8.124	1.97%	2.029	1.682	-17.11%
73	6.121	6.268	2.41%	0.920	0.974	5.83%
74	5.611	5.703	1.63%	0.614	0.697	13.56%
75	5.264	5.303	0.76%	0.381	0.450	17.94%

Table A.1: Compare results of 80 Fork-join cases with small CV value ($CV \leq 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
76	24.116	23.421	-2.88%	10.331	6.182	-40.16%
77	15.612	15.919	1.96%	4.086	3.526	-13.71%
78	12.096	12.364	2.22%	1.868	2.005	7.35%
79	11.136	11.293	1.40%	1.249	1.424	13.96%
80	10.473	10.545	0.69%	0.777	0.911	17.35%

Table A.2: Compare results of 80 Fork-join cases with large CV value ($CV > 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	1.650	1.667	1.01%	2.635	2.582	-2.00%
2	1.701	1.711	0.56%	4.107	4.068	-0.94%
3	1.712	1.721	0.47%	4.830	4.798	-0.67%
4	1.721	1.727	0.38%	5.548	5.523	-0.45%
5	1.732	1.735	0.19%	6.987	6.963	-0.35%
6	2.522	2.639	4.62%	4.234	4.171	-1.49%
7	2.591	2.697	4.10%	6.519	6.493	-0.40%
8	2.607	2.711	3.97%	7.649	7.639	-0.13%
9	2.620	2.720	3.80%	8.778	8.779	0.00%
10	2.636	2.730	3.57%	11.029	11.046	0.154%
11	5.351	5.556	3.83%	9.989	9.895	-0.94%
12	5.444	5.658	3.92%	15.100	15.071	-0.19%
13	5.465	5.681	3.96%	17.622	17.648	0.15%
14	5.484	5.697	3.88%	20.151	20.220	0.34%
15	5.503	5.716	3.86%	25.094	25.352	1.03%
16	10.240	10.444	1.99%	19.934	19.828	-0.53%

Table A.2: Compare results of 80 Fork-join cases with large CV value ($CV > 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
17	10.348	10.592	2.36%	29.983	29.933	-0.17%
18	10.366	10.632	2.57%	34.887	34.982	0.27%
19	10.397	10.659	2.52%	39.877	40.026	0.38%
20	10.388	10.691	2.92%	49.312	50.107	1.61%
21	3.013	2.971	-1.37%	3.642	3.470	-4.73%
22	3.306	3.223	-2.50%	6.051	5.960	-1.49%
23	3.383	3.284	-2.92%	7.238	7.169	-0.96%
24	3.438	3.325	-3.30%	8.409	8.359	-0.59%
25	3.513	3.375	-3.92%	10.742	10.703	-0.37%
26	4.580	4.746	3.63%	5.707	5.429	-4.88%
27	5.015	5.120	2.10%	9.392	9.243	-1.58%
28	5.127	5.210	1.63%	11.202	11.098	-0.93%
29	5.209	5.271	1.20%	12.996	12.927	-0.54%
30	5.315	5.346	0.58%	16.562	16.530	-0.19%
31	9.541	10.071	5.56%	12.719	12.159	-4.40%
32	10.335	10.811	4.61%	20.499	20.229	-1.32%
33	10.558	10.990	4.09%	24.392	24.177	-0.88%
34	10.712	11.110	3.72%	28.185	28.079	-0.38%
35	10.921	11.257	3.08%	35.812	35.784	-0.08%
36	18.020	18.971	5.28%	24.891	23.770	-4.50%
37	19.445	20.296	4.38%	39.899	39.104	-1.99%
38	19.822	20.622	4.04%	47.271	46.622	-1.37%
39	20.108	20.842	3.65%	54.701	54.063	-1.17%
40	20.484	21.109	3.05%	69.389	68.774	-0.89%
41	4.586	4.483	-2.26%	4.455	3.96	-10.96%
42	5.409	5.213	-3.63%	7.894	7.622	-3.44%
43	5.647	5.401	-4.37%	9.627	9.406	-2.30%

Table A.2: Compare results of 80 Fork-join cases with large CV value ($CV > 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
44	5.818	5.530	-4.94%	11.334	11.156	-1.57%
45	6.045	5.690	-5.86%	14.731	14.580	-1.03%
46	6.967	7.155	2.70%	6.968	6.273	-9.98%
47	8.184	8.231	0.57%	12.213	11.831	-3.13%
48	8.545	8.507	-0.44%	14.861	14.550	-2.09%
49	8.806	8.698	-1.23%	17.479	17.221	-1.48%
50	9.143	8.934	-2.28%	22.636	22.450	-0.82%
51	14.495	15.172	4.67%	15.537	14.196	-8.63%
52	16.847	17.284	2.60%	26.611	25.801	-3.05%
53	17.540	17.827	1.64%	32.141	31.504	-1.98%
54	18.016	18.201	1.02%	37.572	37.121	-1.20%
55	18.711	18.664	-0.25%	48.466	48.145	-0.66%
56	27.319	28.551	4.51%	30.262	27.851	-7.97%
57	31.472	32.373	2.86%	51.306	49.754	-3.03%
58	32.688	33.360	2.06%	61.791	60.533	-2.04%
59	33.502	34.039	1.60%	72.088	71.161	-1.29%
60	34.703	34.881	0.51%	92.559	92.049	-0.55%
61	6.722	6.450	-4.05%	5.124	3.728	-27.24%
62	8.652	8.441	-2.44%	9.742	8.864	-9.01%
63	9.281	9.012	-2.89%	12.198	11.494	-5.77%
64	9.762	9.420	-3.50%	14.659	14.091	-3.88%
65	10.412	9.942	-4.51%	19.575	19.161	-2.11%
66	9.946	10.420	4.77%	7.742	6.192	-20.02%
67	13.110	13.291	1.36%	15.141	14.005	-7.50%
68	14.060	14.109	0.35%	18.912	17.994	-4.86%
69	14.773	14.693	-0.54%	22.669	21.933	-3.24%
70	15.733	15.439	-1.87%	30.154	29.637	-1.72%

Table A.2: Compare results of 80 Fork-join cases with large CV value ($CV > 1$)

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
71	21.368	22.328	4.49%	18.278	14.827	-18.88%
72	26.986	27.838	3.16%	33.277	31.147	-6.40%
73	28.803	29.399	2.07%	41.111	39.452	-4.03%
74	30.189	30.511	1.07%	48.965	47.667	-2.65%
75	32.084	31.928	-0.49%	64.537	63.756	-1.21%
76	40.245	42.188	4.83%	35.708	29.708	-16.80%
77	50.374	52.084	3.39%	64.374	60.540	-5.96%
78	53.743	54.883	2.12%	79.320	76.187	-3.95%
79	56.250	56.874	1.11%	94.317	91.663	-2.81%
80	59.821	59.411	-0.68%	124.000	122.006	-1.61%

Table A.3: Compare results of 22 Fork-join cases with mixed distribution

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	3.133	3.273	4.47%	3.061	2.882	-5.84%
2	3.217	3.361	4.47%	4.406	4.284	-2.77%
3	4.998	5.240	4.87%	7.852	7.631	-2.81%
4	4.896	5.128	4.72%	4.130	3.783	-8.41%
5	2.400	2.481	3.36%	2.446	2.352	-3.82%
6	3.225	3.374	4.62%	3.230	3.078	-4.70%
7	5.041	5.287	4.89%	4.911	4.623	-5.88%
8	4.834	5.036	4.18%	3.813	3.468	-9.03%
9	4.917	5.148	4.69%	3.967	3.599	-9.29%
10	3.219	3.379	4.96%	3.332	3.142	-5.71%
11	5.003	5.293	5.80%	5.700	5.428	-4.77%

Table A.3: Compare results of 22 Fork-join cases with mixed distribution

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
12	2.530	2.620	3.56%	3.419	3.332	-2.54%
13	3.376	3.541	4.88%	4.989	4.863	-2.52%
14	5.225	5.512	5.50%	8.459	8.224	-2.78%
15	4.953	5.202	5.02%	5.474	5.196	-5.07%
16	2.528	2.620	3.65%	3.235	3.136	-3.04%
17	3.356	3.540	5.50%	4.611	4.477	-2.90%
18	5.147	5.432	5.54%	7.952	7.701	-3.16%
19	4.954	5.195	4.86%	4.162	3.790	-8.92%
20	2.529	2.620	3.57%	2.912	2.805	-3.67%
21	3.368	3.539	5.10%	3.549	3.363	-5.25%
22	5.176	5.455	5.39%	5.046	4.721	-6.44%

Table A.4: Compare results of 100 cases in exponential group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	32.922	33.264	1.04%	15.504	15.349	-1.00%
2	22.065	22.150	0.38%	12.093	12.237	1.19%
3	34.085	34.109	0.07%	16.296	16.219	-0.47%
4	37.714	37.899	0.49%	17.052	17.052	0.00%
5	21.376	21.829	2.12%	11.908	11.249	-5.54%
6	31.321	31.345	0.07%	15.785	15.782	-0.02%
7	28.526	28.550	0.08%	14.524	14.444	-0.55%
8	22.484	22.601	0.52%	12.512	12.025	-3.90%
9	18.878	18.950	0.38%	10.077	10.132	0.55%
10	30.207	30.424	0.72%	14.271	14.213	-0.41%

Table A.4: Compare results of 100 cases in exponential group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
11	52.266	52.376	0.21%	18.946	18.206	-3.91%
12	43.802	43.779	-0.05%	14.655	13.833	-5.61%
13	70.379	70.132	-0.35%	20.343	20.035	-1.51%
14	33.436	33.413	-0.07%	16.800	16.034	-4.56%
15	44.584	44.663	0.18%	16.548	16.006	-3.28%
16	34.584	34.849	0.77%	20.874	15.205	-27.16%
17	58.979	59.446	0.79%	20.188	19.700	-2.42%
18	30.684	30.713	0.09%	17.750	13.785	-22.34%
19	53.489	52.862	-1.17%	15.796	14.711	-6.87%
20	53.147	52.923	-0.42%	17.926	16.973	-5.31%
21	103.758	103.040	-0.69%	22.255	22.502	1.11%
22	135.699	135.583	-0.09%	24.533	24.594	0.25%
23	55.675	55.998	0.58%	28.435	19.077	-32.91%
24	96.126	97.035	0.95%	26.423	25.373	-3.97%
25	41.106	42.003	2.18%	39.475	14.404	-63.51%
26	152.906	152.305	-0.39%	26.798	24.940	-6.93%
27	144.039	143.401	-0.44%	28.814	28.710	-0.36%
28	105.287	104.300	-0.94%	21.101	20.191	-4.31%
29	114.457	113.811	-0.56%	25.509	22.336	-12.44%
30	78.996	80.136	1.44%	33.058	21.798	-34.06%
31	25.368	25.305	-0.25%	11.926	11.864	-0.52%
32	24.956	25.000	0.18%	14.033	13.217	-5.81%
33	31.627	31.609	-0.05%	15.829	15.834	0.03%
34	8.853	8.850	-0.04%	6.491	5.710	-12.03%
35	17.427	17.511	0.48%	9.349	9.215	-1.44%
36	25.429	25.635	0.81%	12.189	12.079	-0.90%
37	30.036	30.248	0.70%	14.465	14.415	-0.35%

Table A.4: Compare results of 100 cases in exponential group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
38	26.780	26.873	0.35%	13.988	13.835	-1.09%
39	16.738	16.800	0.37%	9.404	9.450	0.49%
40	26.882	26.974	0.34%	13.758	13.763	0.04%
41	52.819	53.301	0.91%	33.748	17.732	-47.46%
42	141.571	141.410	-0.11%	25.117	26.338	4.86%
43	98.140	96.237	-1.94%	24.452	24.178	-1.12%
44	36.576	36.950	1.02%	14.835	14.974	0.94%
45	71.144	70.858	-0.40%	23.226	21.606	-6.98%
46	44.681	44.587	-0.21%	27.803	18.728	-32.64%
47	123.426	123.665	0.19%	30.991	28.301	-8.68%
48	61.812	61.227	-0.95%	38.170	20.756	-45.62%
49	60.436	60.155	-0.46%	20.274	20.217	-0.28%
50	29.714	29.800	0.29%	16.916	16.450	-2.76%
51	321.266	315.944	-1.66%	48.396	44.439	-8.17%
52	209.691	200.441	-4.41%	36.591	34.485	-5.76%
53	475.731	476.396	0.14%	50.100	52.895	5.58%
54	127.181	126.040	-0.90%	30.599	28.121	-8.10%
55	224.727	225.568	0.37%	81.317	39.102	-51.91%
56	416.979	413.304	-0.88%	46.503	51.538	10.83%
57	214.454	205.863	-4.01%	138.947	36.733	-73.56%
58	396.190	390.204	-1.51%	46.995	48.281	2.74%
59	342.566	340.943	-0.47%	61.323	46.835	-23.63%
60	288.108	286.651	-0.51%	38.932	41.254	5.97%
61	21.921	21.897	-0.11%	10.097	10.043	-0.53%
62	22.846	22.900	0.23%	12.800	12.138	-5.17%
63	32.967	33.328	1.09%	15.955	15.695	-1.63%
64	16.440	16.450	0.06%	10.709	9.736	-9.09%

Table A.4: Compare results of 100 cases in exponential group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
65	35.935	35.969	0.10%	17.781	17.877	0.54%
66	25.785	25.800	0.06%	14.912	14.948	0.24%
67	25.359	25.413	0.21%	13.002	13.034	0.25%
68	36.844	37.345	1.36%	17.137	16.813	-1.89%
69	35.992	35.994	0.01%	17.880	17.809	-0.40%
70	19.630	20.019	1.98%	10.769	10.260	-4.73%
71	75.300	75.321	0.03%	53.987	20.231	-62.53%
72	118.808	119.034	0.19%	33.573	26.518	-21.02%
73	42.612	42.816	0.48%	26.266	15.314	-41.70%
74	108.303	108.169	-0.12%	30.620	26.350	-13.94%
75	38.843	38.931	0.23%	15.538	15.175	-2.33%
76	151.431	151.431	-0.00%	29.947	29.349	-2.00%
77	121.740	120.100	-1.35%	26.514	26.540	0.10%
78	43.341	43.233	-0.25%	26.402	17.182	-34.92%
79	143.725	142.813	-0.63%	34.002	28.098	-17.36%
80	52.376	51.884	-0.94%	47.526	17.296	-63.61%
81	33.259	33.315	0.17%	14.381	14.309	-0.50%
82	26.752	26.750	-0.01%	15.746	15.535	-1.34%
83	18.512	18.750	1.28%	10.381	9.796	-5.63%
84	29.620	29.917	1.00%	13.719	13.549	-1.24%
85	28.403	28.732	1.16%	14.651	14.365	-1.95%
86	29.698	30.092	1.33%	14.548	14.266	-1.93%
87	18.988	19.000	0.06%	11.386	11.381	-0.04%
88	24.129	24.302	0.71%	12.678	12.510	-1.32%
89	14.959	14.950	-0.06%	8.565	8.533	-0.37%
90	25.511	25.657	0.57%	13.542	13.491	-0.37%
91	272.168	267.023	-1.89%	36.863	36.975	0.30%

Table A.4: Compare results of 100 cases in exponential group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
92	107.067	106.591	-0.44%	29.904	28.423	-4.95%
93	180.068	179.401	-0.37%	30.811	31.482	2.18%
94	167.178	164.445	-1.63%	101.391	31.166	-69.26%
95	203.194	202.772	-0.21%	35.513	35.461	-0.15%
96	60.402	60.233	-0.28%	35.188	20.714	-41.13%
97	238.523	239.101	0.24%	37.962	36.765	-3.15%
98	73.150	73.265	0.16%	58.043	21.523	-62.92%
99	156.017	155.696	-0.21%	45.646	28.083	-38.48%
100	189.802	189.049	-0.40%	35.475	35.192	-0.80%

Table A.5: Compare results of 100 cases in uniform group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	31.013	31.000	-0.04%	4.223	4.225	0.05%
2	22.157	22.150	-0.03%	3.318	3.316	-0.07%
3	32.900	32.900	0.00%	4.229	4.225	-0.10%
4	34.760	34.5 ⁵ 1	-0.60%	6.176	6.442	4.31%
5	18.608	18.600	-0.04%	2.178	2.176	-0.13%
6	30.917	30.900	-0.06%	6.563	6.564	0.02%
7	28.542	28.550	0.03%	3.816	3.407	-10.74%
8	20.751	20.703	-0.23%	3.772	4.346	15.23%
9	18.954	18.950	-0.02%	1.874	1.822	-2.78%
10	29.123	29.000	-0.42%	6.757	6.974	3.20%
11	45.988	46.134	0.32%	3.989	4.181	4.81%
12	34.230	34.330	0.29%	5.833	6.793	16.47%

Table A.5: Compare results of 100 cases in uniform group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
13	56.954	57.045	0.16%	4.252	4.802	12.93%
14	33.333	33.413	0.24%	8.449	6.725	-20.40%
15	38.717	38.760	0.11%	3.917	3.104	-20.76%
16	32.798	32.800	0.01%	13.238	4.825	-63.55%
17	52.684	52.366	-0.60%	7.158	7.257	1.38%
18	28.638	28.414	-0.78%	10.175	5.007	-50.79%
19	39.820	39.928	0.27%	4.210	5.016	19.14%
20	43.898	42.992	-2.06%	5.090	4.858	-4.55%
21	81.140	81.406	0.33%	6.243	6.675	6.91%
22	102.068	101.961	-0.11%	8.569	8.334	-2.74%
23	51.399	51.128	-0.53%	19.643	6.665	-66.07%
24	85.821	86.125	0.36%	7.665	5.652	-26.26%
25	34.689	35.107	1.20%	28.032	4.797	-82.89%
26	108.686	109.834	1.06%	7.256	8.825	21.62%
27	113.636	113.777	0.12%	9.698	10.209	5.27%
28	77.782	77.148	-0.82%	7.633	9.063	18.74%
29	78.768	78.998	0.29%	4.498	4.941	9.85%
30	66.473	66.475	0.00%	18.657	6.557	-64.86%
31	21.820	21.723	-0.45%	2.925	3.630	24.09%
32	24.972	25.000	0.11%	6.865	5.001	-27.15%
33	31.311	31.300	-0.04%	5.416	5.406	-0.18%
34	8.865	8.850	-0.17%	3.337	1.356	-59.36%
35	16.466	16.435	-0.19%	2.033	2.213	8.84%
36	24.195	24.200	0.02%	4.098	4.106	0.21%
37	28.197	28.088	-0.39%	4.132	4.390	6.24%
38	25.974	26.000	0.10%	3.860	3.861	0.02%
39	16.814	16.800	-0.08%	2.148	2.147	-0.06%

Table A.5: Compare results of 100 cases in uniform group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
40	26.278	26.300	0.08%	4.755	4.769	0.29%
41	50.017	50.172	0.31%	27.371	6.130	-77.60%
42	117.944	117.644	-0.25%	8.556	10.124	18.32%
43	86.565	83.048	-4.06%	9.234	8.601	-6.86%
44	35.556	35.461	-0.27%	3.741	4.057	8.46%
45	70.323	70.200	-0.18%	10.682	6.183	-42.12%
46	44.371	44.484	0.25%	21.036	5.399	-74.33%
47	110.013	109.609	-0.37%	15.474	9.362	-39.50%
48	60.654	59.810	-1.39%	31.012	6.136	-80.21%
49	56.362	56.300	-0.11%	8.830	8.934	1.18%
50	29.834	29.800	-0.11%	6.330	4.470	-29.38%
51	295.477	294.412	-0.36%	33.788	16.304	-51.75%
52	186.508	173.291	-7.09%	22.803	11.408	-49.97%
53	424.162	423.409	-0.18%	20.386	18.655	-8.49%
54	114.561	114.290	-0.24%	19.076	7.654	-59.87%
55	215.047	215.483	0.20%	65.530	13.024	-80.13%
56	381.814	377.702	-1.08%	22.496	15.914	-29.26%
57	198.864	190.684	-4.11%	122.812	11.944	-90.27%
58	361.195	351.147	-2.78%	30.646	17.486	-42.94%
59	317.155	316.836	-0.10%	44.807	15.274	-65.91%
60	257.010	257.077	0.03%	12.229	13.170	7.69%
61	19.493	19.233	-1.33%	3.001	3.608	20.24%
62	22.904	22.900	-0.02%	6.399	4.876	-23.80%
63	30.781	30.449	-1.08%	5.084	5.576	9.67%
64	16.460	16.450	-0.06%	5.559	3.333	-40.05%
65	35.703	35.700	-0.01%	4.385	4.399	0.33%
66	25.835	25.800	-0.14%	6.466	6.377	-1.39%

Table A.5: Compare results of 100 cases in uniform group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
67	24.780	24.800	0.08%	3.103	3.097	-0.18%
68	34.438	34.400	-0.11%	5.618	5.662	0.78%
69	35.413	35.400	-0.04%	3.982	3.996	0.36%
70	17.111	16.958	-0.89%	3.201	3.580	11.82%
71	65.108	64.849	-0.40%	40.540	7.203	-82.23%
72	104.389	103.621	-0.74%	20.215	10.917	-45.99%
73	38.935	38.981	0.12%	17.526	4.750	-72.90%
74	95.587	94.932	-0.69%	12.392	8.364	-32.50%
75	35.528	35.581	0.15%	5.991	6.154	2.73%
76	123.215	123.006	-0.17%	7.454	8.379	12.40%
77	98.216	97.366	-0.87%	9.110	9.314	2.24%
78	40.234	40.288	0.13%	16.825	5.882	-65.04%
79	120.501	118.632	-1.55%	17.338	11.632	-32.91%
80	45.159	44.786	-0.83%	37.228	6.514	-82.50%
81	29.272	29.319	0.16%	3.042	3.196	5.07%
82	26.762	26.750	-0.05%	4.318	3.485	-19.29%
83	15.320	15.422	0.66%	1.203	1.291	7.26%
84	27.836	27.750	-0.31%	5.483	5.736	4.62%
85	27.204	27.200	-0.01%	4.367	4.375	0.17%
86	27.359	27.217	-0.52%	5.762	6.137	6.51%
87	19.002	19.000	-0.01%	2.036	1.975	-3.01%
88	23.419	23.400	-0.08%	6.118	6.135	0.27%
89	14.947	14.950	0.02%	3.015	2.963	-1.72%
90	25.204	25.200	-0.01%	5.229	5.248	0.37%
91	243.077	239.641	-1.41%	12.691	12.063	-4.95%
92	103.286	103.258	-0.03%	14.498	9.848	-32.07%
93	155.709	155.877	0.11%	9.960	10.541	5.83%

Table A.5: Compare results of 100 cases in uniform group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
94	145.709	143.157	-1.75%	82.893	11.307	-86.36%
95	180.495	180.666	0.09%	13.552	12.355	-8.83%
96	58.472	58.291	-0.31%	27.328	8.270	-69.74%
97	208.253	207.835	-0.20%	16.025	13.943	-12.99%
98	68.248	68.176	-0.11%	48.525	7.129	-85.31%
99	133.244	132.998	-0.18%	30.324	9.838	-67.56%
100	174.883	174.633	-0.14%	11.625	12.358	6.31%

Table A.6: Compare results of 100 cases in Erlangian group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	30.992	31.000	0.03%	5.376	5.392	0.30%
2	22.157	22.150	-0.03%	2.298	2.295	-0.13%
3	32.883	32.900	0.05%	2.563	2.560	-0.10%
4	34.536	34.641	0.30%	2.729	2.803	2.72%
5	19.270	19.384	0.59%	2.943	2.869	-2.50%
6	30.898	30.900	0.01%	1.901	1.899	-0.11%
7	28.531	28.550	0.07%	3.358	2.866	-14.66%
8	19.364	19.370	0.03%	1.566	1.589	1.50%
9	18.955	18.950	-0.02%	2.236	2.186	-2.27%
10	28.996	29.000	0.02%	1.644	1.643	-0.07%
11	45.369	45.277	-0.20%	3.673	3.708	0.97%
12	32.330	32.316	-0.04%	1.832	1.841	0.51%
13	55.209	54.703	-0.92%	4.368	4.478	2.51%
14	33.435	33.413	-0.07%	5.590	2.227	-60.15%

Table A.6: Compare results of 100 cases in Erlangian group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
15	38.264	38.200	-0.17%	4.667	3.827	-17.99%
16	32.245	32.327	0.25%	12.295	2.413	-80.38%
17	52.013	52.007	-0.01%	4.109	3.512	-14.53%
18	27.575	27.377	-0.72%	8.599	1.957	-77.24%
19	38.007	37.706	-0.79%	3.345	2.909	-13.05%
20	44.021	43.143	-1.99%	5.601	4.971	-11.25%
21	78.768	78.962	0.25%	5.921	5.947	0.44%
22	100.659	100.592	-0.07%	7.679	6.424	-16.35%
23	49.198	49.283	0.17%	18.574	4.022	-78.34%
24	85.691	85.483	-0.24%	8.349	6.395	-23.40%
25	33.672	33.791	0.35%	26.591	2.660	-90.00%
26	104.601	104.483	-0.11%	5.994	5.850	-2.41%
27	111.785	111.570	-0.19%	8.283	7.686	-7.21%
28	71.751	71.451	-0.42%	4.044	3.278	-18.95%
29	80.884	81.419	0.66%	6.623	6.641	0.27%
30	65.065	65.299	0.36%	18.039	6.346	-64.82%
31	21.119	21.092	-0.13%	2.590	2.807	8.39%
32	24.942	25.000	0.23%	5.101	2.031	-60.18%
33	31.301	31.300	-0.00%	3.509	3.495	-0.38%
34	8.854	8.850	-0.04%	3.182	0.910	-71.41%
35	16.005	15.990	-0.09%	1.499	1.525	1.74%
36	24.198	24.200	0.01%	2.570	2.576	0.25%
37	27.597	27.602	0.02%	2.276	2.293	0.74%
38	25.975	26.000	0.10%	4.019	4.044	0.62%
39	16.796	16.800	0.02%	1.711	1.709	-0.11%
40	26.300	26.300	-0.00%	2.276	2.264	-0.49%
41	49.699	49.550	-0.30%	26.692	3.659	-86.29%

Table A.6: Compare results of 100 cases in Erlangian group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
42	114.531	113.827	-0.61%	8.041	8.554	6.39%
43	85.751	81.125	-5.40%	7.333	4.451	-39.31%
44	34.985	35.000	0.04%	2.596	2.603	0.27%
45	70.191	70.200	0.01%	9.696	4.245	-56.22%
46	44.500	44.484	-0.04%	20.827	4.145	-80.10%
47	108.004	107.743	-0.24%	13.698	5.099	-62.78%
48	60.209	59.810	-0.66%	30.531	3.493	-88.56%
49	56.304	56.300	-0.01%	2.737	2.736	-0.04%
50	29.822	29.800	-0.07%	5.286	2.815	-46.74%
51	295.684	294.542	-0.39%	31.788	8.438	-73.45%
52	184.302	168.359	-8.65%	21.191	6.046	-71.47%
53	418.748	417.761	-0.24%	16.737	11.131	-33.49%
54	114.252	113.632	-0.54%	19.026	6.539	-65.63%
55	216.525	215.128	-0.65%	64.208	8.877	-86.17%
56	380.452	375.044	-1.42%	21.570	13.398	-37.89%
57	194.981	189.142	-2.99%	122.736	7.445	-93.93%
58	357.683	346.214	-3.21%	28.476	10.939	-61.58%
59	316.087	315.131	-0.30%	43.635	11.541	-73.55%
60	255.062	255.199	0.05%	7.203	7.257	0.76%
61	18.358	18.360	0.01%	1.354	1.517	12.07%
62	22.838	22.900	0.27%	5.438	3.602	-33.78%
63	29.735	29.739	0.01%	2.194	2.193	-0.05%
64	16.424	16.450	0.16%	4.690	1.498	-68.06%
65	35.692	35.700	0.02%	2.760	2.768	0.27%
66	25.811	25.800	-0.04%	2.309	2.078	-10.00%
67	24.785	24.800	0.06%	2.868	2.893	0.86%
68	34.432	34.400	-0.09%	4.845	4.838	-0.14%

Table A.6: Compare results of 100 cases in Erlangian group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
69	35.370	35.400	0.08%	3.775	3.784	0.25%
70	16.439	16.447	0.05%	1.511	1.517	0.40%
71	63.879	63.914	0.05%	39.170	4.944	-87.38%
72	102.720	102.765	0.04%	17.709	5.109	-71.15%
73	38.392	38.415	0.06%	16.570	3.212	-80.62%
74	94.491	94.363	-0.13%	13.479	9.480	-29.67%
75	34.993	34.968	-0.07%	2.017	2.061	2.20%
76	121.695	121.164	-0.44%	7.445	7.468	0.31%
77	98.236	97.651	-0.60%	8.316	8.491	2.10%
78	40.335	40.365	0.07%	16.247	3.344	-79.42%
79	116.077	114.904	-1.01%	13.843	5.040	-63.59%
80	45.213	45.114	-0.22%	36.833	3.256	-91.16%
81	29.445	29.394	-0.17%	3.303	3.486	5.53%
82	26.748	26.750	0.01%	3.797	2.819	-25.76%
83	15.560	15.511	-0.31%	2.184	2.331	6.75%
84	27.504	27.500	-0.02%	2.209	2.207	-0.06%
85	27.200	27.200	0.00%	2.544	2.547	0.14%
86	26.702	26.700	-0.01%	1.953	1.961	0.40%
87	19.002	19.000	-0.01%	2.169	2.109	-2.74%
88	23.401	23.400	-0.00%	1.459	1.459	0.04%
89	14.950	14.950	-0.00%	1.342	1.226	-8.68%
90	25.198	25.200	0.01%	1.735	1.740	0.27%
91	240.247	236.985	-1.36%	9.449	7.624	-19.32%
92	103.176	103.220	0.04%	12.228	6.074	-50.33%
93	154.058	153.936	-0.08%	7.618	7.791	2.27%
94	144.268	141.263	-2.08%	81.162	6.381	-92.14%
95	178.857	178.340	-0.29%	9.147	6.792	-25.75%

Table A.6: Compare results of 100 cases in Erlangian group

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
96	58.165	58.042	-0.21%	26.029	3.499	-86.56%
97	205.849	205.620	-0.11%	11.135	6.034	-45.81%
98	67.821	67.728	-0.14%	47.613	3.902	-91.81%
99	129.941	130.044	0.08%	29.021	5.278	-81.82%
100	171.535	171.531	-0.00%	6.659	6.792	1.99%

Table A.7: Compare results of 100 cases in hyperexponential group with CV = 2

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	34.912	36.377	4.20%	33.555	32.347	-3.60%
2	22.159	22.150	-0.04%	24.515	24.473	-0.17%
3	35.678	37.110	4.01%	34.156	33.153	-2.94%
4	40.096	41.076	2.44%	37.043	35.929	-3.01%
5	22.738	23.527	3.47%	26.739	26.153	-2.19%
6	32.322	33.188	2.68%	32.365	31.820	-1.68%
7	28.585	28.550	-0.12%	29.065	28.888	-0.61%
8	23.691	24.195	2.13%	27.022	26.564	-1.70%
9	18.923	18.950	0.14%	20.266	20.265	-0.01%
10	31.877	33.268	4.37%	30.533	29.444	-3.57%
11	59.783	63.602	6.39%	43.441	39.148	-9.88%
12	58.023	61.271	5.60%	43.415	34.021	-21.64%
13	87.255	89.813	2.93%	52.288	43.698	-16.43%
14	33.364	33.413	0.15%	32.431	32.068	-1.12%
15	54.193	58.015	7.05%	41.012	34.059	-16.95%
16	36.209	36.675	1.29%	36.178	32.432	-10.35%

Table A.7: Compare results of 100 cases in hyperexponential group with $CV = 2$

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
17	68.432	72.168	5.46%	46.377	40.134	-13.46%
18	32.972	33.171	0.60%	34.007	30.996	-8.85%
19	69.374	71.593	3.20%	44.941	39.081	-13.04%
20	65.628	69.441	5.81%	45.350	36.365	-19.81%
21	149.010	148.410	-0.40%	62.282	50.123	-19.52%
22	193.365	193.716	0.18%	68.732	61.371	-10.71%
23	59.685	60.652	1.62%	48.359	42.464	-12.19%
24	106.782	109.065	2.14%	57.884	55.950	-3.34%
25	47.749	48.923	2.46%	58.417	36.188	-38.05%
26	227.006	227.550	0.24%	73.665	62.764	-14.80%
27	194.300	197.808	1.81%	73.419	65.685	-10.53%
28	150.420	152.078	1.10%	61.102	52.951	-13.34%
29	154.305	156.597	1.49%	68.018	62.316	-8.38%
30	91.388	94.095	2.96%	62.145	52.906	-14.87%
31	27.986	28.975	3.53%	28.727	27.111	-5.63%
32	25.012	25.000	-0.05%	26.921	26.434	-1.81%
33	32.235	33.121	2.75%	32.077	31.544	-1.66%
34	8.842	8.850	0.10%	11.781	11.420	-3.06%
35	18.020	18.286	1.48%	19.259	19.217	-0.22%
36	26.702	27.782	4.04%	25.784	25.111	-2.61%
37	31.609	32.481	2.76%	30.695	30.006	-2.25%
38	27.701	28.722	3.69%	28.965	28.375	-2.04%
39	16.780	16.800	0.12%	18.902	18.901	-0.00%
40	27.886	28.848	3.45%	28.418	27.786	-2.23%
41	57.074	58.711	2.87%	50.599	36.867	-27.14%
42	189.821	190.265	0.23%	68.292	53.508	-21.65%
43	117.142	118.474	1.14%	57.738	48.744	-15.58%

Table A.7: Compare results of 100 cases in hyperexponential group with CV = 2

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
44	38.450	39.419	2.52%	31.160	30.462	-2.24%
45	74.607	76.714	2.82%	45.871	42.900	-6.48%
46	45.184	45.378	0.43%	43.717	37.535	-14.14%
47	144.942	149.605	3.22%	67.506	55.941	-17.13%
48	64.919	66.003	1.67%	56.680	41.105	-27.48%
49	69.622	73.597	5.71%	48.040	42.318	-11.91%
50	29.873	29.800	-0.24%	33.406	32.899	-1.52%
51	401.390	388.298	-3.26%	94.409	81.133	-14.06%
52	266.675	254.522	-4.56%	79.428	63.284	-20.33%
53	582.557	583.762	0.21%	119.366	105.979	-11.22%
54	157.854	154.466	-2.15%	64.148	53.496	-16.61%
55	256.281	259.707	1.34%	127.440	75.469	-40.78%
56	498.667	491.622	-1.41%	105.888	98.408	-7.06%
57	251.817	245.151	-2.65%	184.951	70.167	-62.06%
58	491.131	479.081	-2.45%	101.269	92.933	-8.23%
59	388.295	392.756	1.15%	113.075	91.736	-18.87%
60	362.948	362.042	-0.25%	94.100	79.907	-15.61%
61	23.942	24.714	3.22%	23.691	22.587	-4.66%
62	22.904	22.900	-0.02%	24.681	24.277	-1.64%
63	34.792	35.806	2.91%	34.667	33.734	-2.69%
64	16.482	16.450	-0.20%	20.125	19.473	-3.24%
65	36.738	37.530	2.16%	36.265	35.473	-2.18%
66	25.853	25.800	-0.21%	30.106	29.896	-0.70%
67	26.419	27.529	4.20%	27.127	26.255	-3.22%
68	39.206	40.783	4.02%	38.130	36.625	-3.95%
69	36.894	37.989	2.97%	36.427	35.778	-1.78%
70	21.300	22.244	4.43%	24.884	23.995	-3.57%

Table A.7: Compare results of 100 cases in hyperexponential group with CV = 2

case	simulation mean	approxim. mean	mean error	simulation σ	approxim. σ	σ error
71	87.257	91.473	4.83%	79.877	44.453	-44.35%
72	139.391	140.633	0.89%	66.628	56.406	-15.34%
73	47.848	48.927	2.26%	45.096	33.361	-26.02%
74	124.417	127.207	2.24%	65.320	57.935	-11.31%
75	41.866	43.934	4.94%	34.458	32.505	-5.67%
76	192.384	194.976	1.35%	75.856	63.316	-16.53%
77	157.819	159.121	0.83%	66.628	55.921	-16.07%
78	46.365	46.978	1.32%	44.353	35.454	-20.06%
79	170.323	172.462	1.26%	73.485	66.734	-9.19%
80	61.737	63.648	3.09%	68.632	37.922	-44.75%
81	35.721	36.788	2.99%	32.401	31.041	-4.20%
82	26.822	26.750	-0.27%	31.385	31.069	-1.01%
83	19.735	20.083	1.76%	23.770	23.287	-2.03%
84	31.665	33.177	4.78%	30.244	28.931	-4.34%
85	29.450	30.482	3.51%	30.515	29.964	-1.81%
86	31.904	33.251	4.22%	32.349	31.035	-4.06%
87	18.998	19.000	0.01%	22.741	22.762	0.10%
88	25.036	26.006	3.87%	26.137	25.657	-1.84%
89	14.933	14.950	0.12%	17.079	17.066	-0.08%
90	26.147	26.893	2.85%	27.634	27.105	-1.91%
91	350.167	342.527	-2.18%	89.867	75.726	-15.74%
92	120.701	121.581	0.73%	61.766	53.421	-13.51%
93	229.380	232.906	1.54%	78.062	64.444	-17.45%
94	202.501	205.398	1.43%	144.122	64.059	-55.55%
95	256.405	258.164	0.69%	84.009	69.882	-16.82%
96	63.663	64.877	1.91%	54.609	41.994	-23.10%
97	293.331	300.164	2.33%	91.529	77.247	-15.60%

Table A.7: Compare results of 100 cases in hyperexponential group with CV = 2

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
98	82.247	84.701	2.98%	80.903	44.063	-45.54%
99	196.687	196.913	0.11%	83.420	58.153	-30.29%
100	223.280	226.012	1.22%	79.985	71.456	-10.66%

Table A.8: Compare results of 100 cases in hyperexponential group with CV = 3

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	35.694	37.256	4.38%	51.520	50.656	-1.68%
2	22.089	22.150	0.28%	36.494	36.710	0.59%
3	36.429	37.968	4.22%	52.252	51.476	-1.49%
4	40.929	42.027	2.68%	56.702	55.906	-1.40%
5	23.263	23.970	3.04%	41.406	40.962	-1.07%
6	32.748	33.631	2.70%	49.309	48.668	-1.30%
7	28.454	28.550	0.34%	43.321	43.332	0.02%
8	24.102	24.614	2.13%	41.317	41.083	-0.56%
9	18.893	18.950	0.30%	30.326	30.397	0.24%
10	32.599	34.063	4.49%	46.581	46.001	-1.25%
11	64.608	70.257	8.74%	70.201	65.481	-6.72%
12	65.745	70.514	7.25%	74.189	67.516	-8.99%
13	98.403	104.401	6.10%	88.316	76.678	-13.18%
14	33.401	33.413	0.03%	48.409	48.101	-0.63%
15	60.087	66.105	10.02%	68.212	61.168	-10.33%
16	36.864	37.274	1.11%	52.533	49.866	-5.08%
17	74.929	80.912	7.98%	74.561	68.076	-8.70%
18	33.831	33.996	0.49%	50.615	48.429	-4.32%

Table A.8: Compare results of 100 cases in hyperexponential group with $CV = 3$

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
19	78.287	82.299	5.13%	76.866	69.979	-8.96%
20	73.941	80.463	8.82%	76.036	66.829	-12.11%
21	189.695	192.398	1.43%	114.383	82.283	-28.06%
22	240.893	241.066	0.07%	123.691	100.363	-18.86%
23	61.521	62.319	1.30%	70.433	66.260	-5.92%
24	113.217	116.115	2.56%	91.826	89.118	-2.95%
25	50.594	51.506	1.80%	77.495	59.034	-23.82%
26	294.980	294.807	-0.06%	132.067	99.212	-24.88%
27	237.360	244.276	2.91%	128.370	104.156	-18.86%
28	188.014	193.607	2.97%	111.333	86.836	-22.00%
29	183.635	191.863	4.48%	117.507	100.983	-14.06%
30	96.939	99.297	2.43%	92.874	85.566	-7.87%
31	28.967	30.056	3.76%	45.061	44.081	-2.18%
32	24.944	25.000	0.23%	39.923	39.651	-0.68%
33	32.669	33.643	2.98%	48.468	48.041	-0.88%
34	8.829	8.850	0.23%	17.281	17.130	-0.87%
35	18.233	18.488	1.40%	29.192	29.223	0.11%
36	27.289	28.394	4.05%	39.383	38.978	-1.03%
37	32.223	33.120	2.79%	46.714	46.229	-1.04%
38	28.172	29.210	3.68%	43.952	43.575	-0.86%
39	16.752	16.800	0.29%	28.352	28.351	-0.00%
40	28.395	29.413	3.59%	42.911	42.589	-0.75%
41	59.024	61.096	3.51%	69.484	58.983	-15.11%
42	233.442	241.575	3.48%	122.535	92.563	-24.46%
43	131.334	140.730	7.15%	96.433	78.758	-18.33%
44	39.299	41.002	4.33%	47.267	46.652	-1.30%
45	77.360	80.761	4.40%	71.620	67.606	-5.60%

Table A.8: Compare results of 100 cases in hyperexponential group with $CV = 3$

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
46	45.443	45.834	0.86%	61.690	56.872	-7.81%
47	162.955	170.934	4.90%	111.763	96.690	-13.49%
48	67.005	68.931	2.87%	77.793	64.797	-16.71%
49	75.210	82.660	9.91%	78.100	71.218	-8.81%
50	29.765	29.800	0.12%	49.505	49.349	-0.31%
51	505.354	481.834	-4.65%	160.933	124.179	-22.84%
52	331.636	321.351	-3.10%	139.505	98.739	-29.22%
53	687.495	704.599	2.49%	200.411	169.293	-15.53%
54	190.914	188.416	-1.31%	111.927	83.723	-25.20%
55	282.060	297.643	5.52%	181.574	121.691	-32.98%
56	597.273	592.602	-0.78%	176.994	149.255	-15.67%
57	284.171	288.888	1.66%	239.209	114.573	-52.10%
58	598.928	592.876	-1.01%	176.120	146.469	-16.84%
59	426.436	442.383	3.74%	176.540	147.609	-16.39%
60	433.492	448.051	3.36%	164.264	131.017	-20.24%
61	24.698	25.538	3.40%	36.958	36.352	-1.64%
62	22.831	22.900	0.30%	36.635	36.415	-0.60%
63	35.452	36.481	2.90%	52.953	52.334	-1.17%
64	16.484	16.450	-0.21%	29.840	29.209	-2.12%
65	37.178	38.191	2.72%	54.420	53.958	-0.85%
66	25.780	25.800	0.08%	44.838	44.844	0.01%
67	26.997	28.158	4.30%	41.263	40.691	-1.38%
68	40.020	41.693	4.18%	58.410	57.710	-1.20%
69	37.390	38.578	3.18%	55.142	54.587	-1.01%
70	21.918	22.828	4.15%	38.527	38.115	-1.07%
71	94.755	102.031	7.68%	108.017	75.355	-30.24%
72	157.041	161.071	2.57%	108.429	92.089	-15.07%

Table A.8: Compare results of 100 cases in hyperexponential group with $CV = 3$

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
73	51.112	53.111	3.91%	65.907	54.402	-17.46%
74	136.435	140.235	2.79%	104.235	95.208	-8.66%
75	43.336	45.511	5.02%	53.743	52.086	-3.08%
76	226.163	233.348	3.18%	128.749	108.860	-15.45%
77	188.061	195.821	4.13%	114.472	91.966	-19.66%
78	48.378	49.532	2.38%	64.037	55.226	-13.76%
79	189.939	198.587	4.55%	119.964	103.511	-13.72%
80	67.844	71.824	5.87%	93.405	64.501	-30.94%
81	36.634	37.832	3.27%	50.082	49.251	-1.66%
82	26.762	26.750	-0.04%	46.870	46.604	-0.57%
83	20.098	20.434	1.67%	36.781	36.399	-1.04%
84	32.519	34.127	4.94%	46.581	45.896	-1.47%
85	29.984	30.953	3.23%	46.372	45.956	-0.90%
86	32.752	34.147	4.26%	49.888	49.012	-1.76%
87	18.971	19.000	0.15%	33.989	34.143	0.45%
88	25.504	26.508	3.94%	39.637	39.416	-0.56%
89	14.901	14.950	0.33%	25.483	25.600	0.46%
90	26.521	27.363	3.18%	41.445	41.308	-0.33%
91	433.070	429.526	-0.82%	156.800	122.755	-21.71%
92	135.189	143.008	5.78%	102.477	88.434	-13.70%
93	274.429	285.279	3.95%	136.501	106.687	-21.84%
94	230.419	240.056	4.18%	191.914	108.976	-43.22%
95	307.940	316.311	2.72%	145.894	118.475	-18.79%
96	65.618	67.474	2.83%	76.649	65.995	-13.90%
97	343.564	360.467	4.92%	155.756	130.410	-16.27%
98	87.958	92.849	5.56%	106.229	72.822	-31.45%
99	231.030	238.993	3.45%	132.109	101.117	-23.46%

Table A.8: Compare results of 100 cases in hyperexponential group with CV = 3

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
100	258.264	266.682	3.26%	135.197	113.947	-15.72%

Table A.9: Compare results of 100 cases in hyperexponential group with CV = 4

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	36.086	37.615	4.24%	69.475	68.740	-1.06%
2	22.056	22.150	0.43%	48.828	48.947	0.24%
3	36.753	38.321	4.27%	69.984	69.640	-0.49%
4	41.363	42.418	2.55%	76.484	75.674	-1.06%
5	23.351	24.141	3.38%	55.417	55.513	0.17%
6	32.848	33.807	2.92%	65.880	65.420	-0.70%
7	28.412	28.550	0.48%	57.768	57.775	0.01%
8	24.247	24.777	2.19%	55.565	55.426	-0.25%
9	18.841	18.950	0.58%	40.275	40.529	0.63%
10	32.954	34.388	4.35%	62.702	62.375	-0.52%
11	67.143	73.080	8.84%	96.789	93.424	-3.48%
12	69.972	74.811	6.92%	105.872	101.123	-4.49%
13	104.494	111.988	7.17%	124.819	116.319	-6.81%
14	33.367	33.413	0.14%	64.396	64.135	-0.41%
15	63.337	69.722	10.08%	95.613	90.396	-5.46%
16	37.067	37.514	1.21%	68.811	67.189	-2.36%
17	78.973	85.348	8.07%	104.363	98.432	-5.68%
18	34.197	34.326	0.38%	67.103	65.668	-2.14%
19	82.936	86.909	4.79%	109.721	104.408	-4.84%
20	78.738	85.942	9.15%	107.906	100.224	-7.12%

Table A.9: Compare results of 100 cases in hyperexponential group with CV = 4

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
21	222.679	229.398	3.02%	175.630	143.304	-18.41%
22	280.791	283.267	0.88%	187.148	142.780	-23.71%
23	62.419	63.004	0.94%	93.007	89.861	-3.38%
24	116.507	121.596	4.37%	125.529	121.929	-2.87%
25	51.775	52.591	1.58%	96.206	81.559	-15.22%
26	358.964	363.122	1.16%	204.577	146.766	-28.26%
27	272.959	285.417	4.56%	191.409	150.022	-21.62%
28	215.994	227.259	5.22%	168.057	133.271	-20.70%
29	204.544	219.789	7.45%	171.047	149.461	-12.62%
30	99.537	101.471	1.94%	123.258	117.824	-4.41%
31	29.413	30.497	3.69%	61.260	60.630	-1.03%
32	24.859	25.000	0.57%	53.015	52.868	-0.28%
33	32.859	33.854	3.03%	64.785	64.489	-0.46%
34	8.811	8.850	0.44%	22.801	22.840	0.17%
35	18.301	18.567	1.45%	39.063	39.175	0.29%
36	27.552	28.645	3.96%	53.035	52.710	-0.61%
37	32.510	33.381	2.68%	62.911	62.322	-0.94%
38	28.322	29.404	3.82%	58.593	58.668	0.13%
39	16.696	16.800	0.62%	37.561	37.802	0.64%
40	28.675	29.645	3.38%	57.746	57.317	-0.74%
41	60.012	62.173	3.60%	89.330	81.099	-9.21%
42	270.077	286.915	6.23%	188.575	135.248	-28.28%
43	141.158	156.816	11.09%	139.043	117.044	-15.82%
44	39.844	41.549	4.28%	63.694	63.228	-0.73%
45	78.488	82.994	5.74%	97.187	93.451	-3.84%
46	45.587	46.007	0.92%	80.537	76.322	-5.23%
47	174.711	191.167	9.42%	157.916	140.107	-11.28%

Table A.9: Compare results of 100 cases in hyperexponential group with CV = 4

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
48	67.986	70.315	3.43%	99.150	89.050	-10.19%
49	78.089	86.202	10.39%	108.195	103.232	-4.59%
50	29.861	29.800	-0.20%	66.175	65.798	-0.57%
51	614.100	594.897	-3.13%	240.693	186.337	-22.58%
52	392.686	391.574	-0.28%	210.071	147.218	-29.92%
53	777.348	808.977	4.07%	294.377	259.682	-11.79%
54	217.481	226.244	4.03%	168.151	127.661	-24.08%
55	301.158	321.316	6.69%	240.549	183.771	-23.60%
56	694.367	695.831	0.21%	260.855	214.155	-17.90%
57	306.300	318.138	3.86%	294.485	178.598	-39.35%
58	701.548	702.738	0.17%	264.884	225.225	-14.97%
59	457.111	478.837	4.75%	247.076	217.039	-12.16%
60	495.717	522.010	5.30%	243.472	205.488	-15.60%
61	25.092	25.874	3.11%	50.274	49.817	-0.91%
62	22.769	22.900	0.58%	48.517	48.553	0.07%
63	35.763	36.750	2.76%	71.239	70.717	-0.73%
64	16.495	16.450	-0.27%	39.744	38.945	-2.01%
65	37.413	38.462	2.81%	72.555	72.415	-0.19%
66	25.814	25.800	-0.06%	59.787	59.792	0.01%
67	27.279	28.414	4.16%	55.281	55.012	-0.49%
68	40.477	42.059	3.91%	78.853	78.461	-0.50%
69	37.649	38.816	3.10%	73.807	73.318	-0.66%
70	22.207	23.057	3.83%	52.236	51.933	-0.58%
71	98.531	106.887	8.48%	136.331	109.384	-19.77%
72	168.711	182.679	8.28%	152.639	130.828	-14.29%
73	53.008	55.035	3.83%	87.480	77.612	-11.28%
74	142.216	152.755	7.41%	144.061	131.653	-8.61%

Table A.9: Compare results of 100 cases in hyperexponential group with $CV = 4$

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
75	43.963	46.170	5.02%	72.577	71.409	-1.61%
76	254.561	270.820	6.39%	188.100	161.536	-14.12%
77	211.143	230.050	8.95%	170.046	131.849	-22.46%
78	49.337	50.672	2.71%	83.460	76.438	-8.41%
79	202.420	218.308	7.85%	168.184	148.242	-11.86%
80	71.392	75.506	5.76%	119.348	95.584	-19.91%
81	37.087	38.262	3.17%	67.742	67.164	-0.85%
82	26.817	26.750	-0.25%	62.595	62.139	-0.73%
83	20.204	20.568	1.80%	49.571	49.281	-0.58%
84	32.962	34.517	4.72%	63.043	62.594	-0.71%
85	30.227	31.140	3.02%	62.186	61.827	-0.58%
86	33.098	34.510	4.27%	67.043	66.700	-0.51%
87	18.929	19.000	0.38%	45.043	45.524	1.07%
88	25.733	26.712	3.80%	53.302	53.084	-0.41%
89	14.874	14.950	0.51%	34.008	34.133	0.37%
90	26.708	27.554	3.17%	55.421	55.463	0.08%
91	511.284	515.437	0.81%	240.039	179.395	-25.26%
92	145.564	160.102	9.99%	145.956	128.069	-12.25%
93	310.238	331.393	6.82%	199.916	159.130	-20.40%
94	251.866	267.933	6.38%	245.117	159.815	-34.80%
95	349.910	368.589	5.34%	220.295	175.329	-20.41%
96	66.518	68.649	3.20%	98.760	90.424	-8.44%
97	384.802	408.861	6.25%	224.861	190.458	-15.30%
98	91.418	98.466	7.71%	133.295	102.597	-23.03%
99	258.409	276.420	6.97%	190.186	150.016	-21.12%
100	287.685	305.546	6.21%	195.968	161.523	-17.58%

Table A.10: Compare results of 100 cases of mixed distribution

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
1	19.939	19.844	-0.48%	2.599	2.744	5.56%
2	18.855	19.255	2.12%	6.147	4.854	-21.03%
3	20.679	20.712	0.16%	7.848	7.858	0.12%
4	29.912	29.900	-0.04%	4.995	4.995	0.01%
5	27.855	27.850	-0.02%	4.051	3.707	-8.50%
6	30.601	30.600	-0.00%	3.174	2.735	-13.83%
7	29.935	31.466	5.11%	20.837	18.886	-9.36%
8	28.744	28.510	-0.82%	9.462	9.671	2.21%
9	21.775	21.873	0.45%	6.461	6.830	5.71%
10	21.955	21.950	-0.02%	4.999	4.753	-4.93%
11	26.292	26.321	0.11%	8.056	6.200	-23.04%
12	53.083	53.175	0.17%	13.481	7.858	-41.71%
13	47.381	48.070	1.46%	12.621	10.126	-19.77%
14	42.898	46.488	8.37%	22.454	18.965	-15.54%
15	42.084	42.071	-0.03%	17.747	17.507	-1.35%
16	25.605	25.610	0.02%	13.408	12.610	-5.96%
17	37.610	37.695	0.23%	2.779	3.166	13.92%
18	68.914	73.509	6.67%	41.009	27.982	-31.77%
19	47.791	53.747	12.46%	28.257	23.280	-17.61%
20	22.015	21.998	-0.08%	7.222	4.877	-32.47%
21	61.380	62.289	1.48%	35.438	23.666	-33.22%
22	95.253	96.831	1.66%	23.959	18.769	-21.66%
23	73.822	76.828	4.07%	37.250	32.616	-12.44%
24	31.794	31.534	-0.82%	11.725	11.263	-3.95%
25	33.116	32.968	-0.45%	19.740	15.430	-21.83%

Table A.10: Compare results of 100 cases of mixed distribution

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
26	59.067	59.180	0.19%	41.688	33.146	-20.49%
27	128.610	128.100	-0.40%	41.646	32.730	-21.41%
28	84.203	87.318	3.70%	37.622	19.832	-47.29%
29	91.900	92.928	1.12%	17.055	14.710	-13.75%
30	54.192	55.214	1.89%	40.425	19.481	-51.81%
31	21.602	21.600	-0.01%	8.030	7.911	-1.48%
32	36.046	36.116	0.19%	10.630	10.616	-0.14%
33	26.194	26.232	0.15%	7.346	7.322	-0.32%
34	29.114	29.000	-0.39%	7.514	7.626	1.50%
35	21.932	21.900	-0.15%	7.092	7.067	-0.36%
36	23.893	23.900	0.03%	9.514	8.705	-8.50%
37	20.072	20.049	-0.11%	11.547	11.550	0.02%
38	29.050	29.343	1.01%	1.669	2.000	19.79%
39	17.424	19.423	11.47%	20.310	19.312	-4.91%
40	29.676	29.700	0.08%	11.266	11.299	0.30%
41	38.109	39.689	4.15%	43.506	21.495	-50.59%
42	112.880	121.071	7.26%	58.211	42.796	-26.48%
43	83.180	88.077	5.89%	38.008	30.627	-19.42%
44	22.193	22.801	2.74%	9.341	8.446	-9.58%
45	97.464	104.319	7.03%	46.206	39.078	-15.43%
46	60.434	61.450	1.68%	42.006	19.682	-53.14%
47	115.860	118.383	2.18%	43.633	27.108	-37.87%
48	97.791	105.501	7.88%	44.552	29.662	-33.42%
49	91.663	97.553	6.43%	47.332	39.445	-16.66%
50	21.830	21.877	0.22%	17.570	17.395	-1.00%
51	175.666	181.226	3.16%	152.317	37.535	-75.36%
52	207.733	210.557	1.36%	60.042	47.903	-20.22%

Table A.10: Compare results of 100 cases of mixed distribution

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
53	350.101	366.178	4.59%	64.151	44.440	-30.73%
54	374.227	382.388	2.18%	87.631	50.181	-42.74%
55	401.964	387.573	-3.58%	99.493	42.509	-57.27%
56	144.279	147.750	2.41%	129.038	30.129	-76.65%
57	477.868	482.598	0.99%	87.047	51.901	-40.38%
58	361.969	378.768	4.64%	89.702	59.144	-34.07%
59	281.847	294.796	4.59%	87.359	57.307	-34.40%
60	383.695	386.892	0.83%	58.899	42.894	-27.17%
61	27.017	26.900	-0.43%	42.837	42.043	-1.85%
62	31.605	31.517	-0.28%	36.845	36.953	0.29%
63	24.110	24.284	0.72%	32.300	32.626	1.01%
64	35.267	39.244	11.28%	32.572	30.377	-6.74%
65	28.697	28.987	1.01%	5.123	4.150	-18.99%
66	26.035	26.100	0.25%	33.943	33.978	0.10%
67	18.961	18.950	-0.06%	6.413	2.616	-59.22%
68	42.013	42.106	0.22%	10.892	10.280	-5.63%
69	28.967	29.247	0.97%	9.670	9.378	-3.02%
70	23.884	23.834	-0.21%	2.729	2.850	4.45%
71	63.080	64.638	2.47%	57.542	25.833	-55.11%
72	48.126	48.365	0.50%	20.811	16.395	-21.22%
73	78.877	84.189	6.74%	36.342	31.262	-13.98%
74	101.102	105.458	4.31%	32.689	27.357	-16.31%
75	107.986	105.279	-2.51%	29.620	14.481	-51.11%
76	65.045	67.891	4.37%	59.039	31.446	-46.74%
77	47.024	47.113	0.19%	22.822	12.320	-46.02%
78	73.408	77.305	5.31%	32.091	25.788	-19.64%
79	106.049	103.777	-2.14%	54.157	51.001	-5.83%

Table A.10: Compare results of 100 cases of mixed distribution

case	simulation mean	approx. mean	mean error	simulation σ	approx. σ	σ error
80	140.162	139.919	-0.17%	38.751	32.672	-15.69%
81	23.697	23.700	0.01%	5.952	5.924	-0.48%
82	16.698	16.700	0.01%	7.135	7.102	-0.47%
83	30.929	30.910	-0.06%	3.284	3.555	8.25%
84	23.540	23.616	0.32%	3.614	3.566	-1.32%
85	29.202	30.896	5.80%	24.426	23.477	-3.88%
86	22.942	22.950	0.04%	10.047	8.310	-17.29%
87	25.844	25.850	0.02%	6.135	4.325	-29.51%
88	25.552	25.364	-0.73%	6.883	7.169	4.17%
89	17.531	17.422	-0.62%	4.483	4.573	2.01%
90	38.856	38.825	-0.08%	21.277	21.273	-0.02%
91	76.577	77.822	1.63%	48.430	19.972	-58.76%
92	38.428	38.518	0.23%	34.379	28.110	-18.23%
93	197.055	208.885	6.00%	46.362	34.924	-24.67%
94	189.797	181.107	-4.58%	66.411	22.206	-66.56%
95	247.816	247.991	0.07%	74.378	49.863	-32.96%
96	127.963	129.198	0.97%	95.894	32.459	-66.15%
97	129.646	136.324	5.15%	120.709	34.919	-71.07%
98	262.844	273.725	4.14%	74.715	42.235	-43.47%
99	196.314	192.321	-2.03%	32.510	21.383	-34.23%
100	228.315	231.837	1.54%	54.189	32.097	-40.77%

END

2 1 - 1 1 1 9 6

FIN