MARKOVIAN WORKLOAD MODELING FOR ENTERPRISE APPLICATION SERVERS

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

UNIVERSITY OF REGINA

By

Nima Sharifimehr

Regina, Saskatchewan

2008

© Copyright 2008 : Nima Sharifimehr

Library and Archives
Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file *Votre référence*
ISBN: 978-0-494-55024-3
Our file *Notre référence*
ISBN: 978-0-494-55024-3

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

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

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

UNIVERSITY OF REGINA
FACULTY OF GRADUATE STUDIES AND RESEARCH
SUPERVISORY AND EXAMINING COMMITTEE

Nima Sharifimehr, candidate for the degree of Master of Science in Computer Science, has presented a thesis titled, ***Markovian Workload Modeling for Enterprise Application Servers***, in an oral examination held on May 6, 2008. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner: Dr. Allen Herman, Department of Mathematics and Statistics

Supervisor: Dr. Samira Sadaoui Mouhoub,
Department of Computer Science

Committee Member: Dr. Yiyu Yao, Department of Computer Science

Committee Member: Dr. Malek Mouhbou, Department of Computer Science

Chair of Defense: Dr. Iqbal Husain, Luther College

Abstract

The first part of this thesis introduces a novel Markovian approach to model the incoming workload toward an Enterprise Application Server (EAS). This Markovian approach is based on the Novel Dynamic Semi-Markov Model (*NDSMM*), proposed in this thesis, which is an intuitive combination of the Semi-Markov Model and the Dynamic Markov Model. *NDSMM* is designed with the intention in mind to be an efficient workload modeling tool for enterprise application servers. However, a formal definition of the *NDSMM* prepares a research ground for other academic researchers to investigate other possible applications of *NDSMM*. Subsequently, the detailed algorithm of building a *NDSMM* for a typical system along with adjustment techniques is illustrated. A comparison operator is introduced which allows us to measure the similarity of two given *NDSMMs*. The comparison operator is used to reduce the redundancy when *NDSMMs* are being used for workload modeling.

The second half of this thesis discusses the application of *NDSMMs* for workload modeling of enterprise application servers. It includes a description of how to design a Generic Modeling Component (GMC) and integrate it into enterprise application servers. *GMC* uses *NDSMMs* to model the workload of a given *EAS* and stores the extracted models in GraphXML format. Afterwards, different integration approaches, including general and advanced solutions, are investigated to show how to integrate

GM instances into an *EAS*.

To prove the efficiency and applicability of the proposed approach for real world applications, several experiments and evaluations are carried out. The results of these experiments are analyzed to show that NDSMM-based workload modeling for enterprise application servers is an efficient and accurate solution.

Acknowledgements

I would like to sincerely thank my supervisor Dr. Samira Sadaoui. She helped me to be accepted into the M.Sc. program in computer science and provided me with financial support. Her constant guidance is necessary for me to become a more mature researcher. Both her kindness and strictness will be appreciated forever.

I gratefully acknowledge the Faculty of Graduate Studies and Research and the Department of Computer Science for their financial support to conduct my research.

Dedication

I would like to dedicate my thesis to my beloved parents who have been the best source of motivation and inspiration for me since the beginning of my studies.

Contents

Abstract	i
Acknowledgements	iii
Dedication	iv
Table of Contents	v
List of Tables	viii
List of Figures	ix
List of Acronyms	xii
Chapter 1 Introduction	1
1.1 Problem and Motivation	1
1.2 Proposed Solution	3
1.3 Thesis Organization	6
Chapter 2 Background	8
2.1 Enterprise Application Servers	8
2.2 Workload Modeling	14

2.3	Markovian Models	16
2.4	Conclusion	21
Chapter 3	Novel Dynamic Semi-Markov Model	22
3.1	Novel Dynamic Semi-Markov Model	22
3.1.1	Dynamic Building Process	25
3.1.2	RMSE of a NDSMM	26
3.1.3	Algorithm Design	28
3.1.4	Algorithm Implementation	32
3.1.5	Adjustment of Parameters	34
3.2	Measuring the Similarity Between two NDSMMs	37
3.2.1	Sample Data Generation	39
3.2.2	Sample Data Tracing	42
3.2.3	Similarity Calculation	45
3.3	Conclusion	46
Chapter 4	NDSMM-based Workload Modeling	47
4.1	Generic Modeling Component (GMC)	47
4.1.1	GMC Functionalities	48
4.1.2	Using GraphXML to Store NDSMMs	50
4.2	Integration of GMC into Enterprise Application Servers	50
4.2.1	General Integration Approaches	50
4.2.2	Advanced Integration Approaches	53
4.2.3	Implementation Issues	55
4.3	Conclusion	56

Chapter 5	Experiments and Evaluations	57
5.1	Evaluation Results	58
5.2	Conclusion	61
Chapter 6	Conclusion and Prospects	62
6.1	Contributions	62
6.2	Future Work	64
Bibliography		65
Appendix A		75

List of Tables

5.1	Similarity of built NDSMMs and real workloads.	59
5.2	Required workload repetitions to achieve stabilized <i>NDSMMs</i>	60

List of Figures

1.1	Performance analysis of any software system must be done considering its operational environment and processing workload.	2
2.1	The overall architecture of a component-based EAS [35].	10
2.2	The architecture of JOnAS J2EE application server [2].	12
2.3	Sample workload generation based on workload models.	15
2.4	An example of Markov chain.	17
2.5	An example of 1-st order semi-Markov model, where E shows an exponential distribution, D a deterministic distribution, W a Weibull distribution, and U a uniform distribution [25].	19
2.6	An example of applying cloning operation on a partially built DMM [12].	20
3.1	An example of $NDSMM$ which only contains two states S_0 and S_1 and the elements of analyzing data belong to the finite set $\{0, 1\}$	24
3.2	An example of applying cloning operation on a $NDSMM$. Transitions in bold in each model show transitions which cause the cloning.	34
3.3	Comparing $NDSMM_X$ and $NDSMM_Y$ when they are capturing totally different behavioral features of the modeling system.	38

3.4	Comparing $NDSMM_X$ and $NDSMM_Y$ when both of them are capturing exactly the same behavioral features of the modeling system. .	39
3.5	The similarity calculator analyzes the received metadata from the sample data generator and the sample data tracer and returns the result of this analysis as the level of similarity between two given model $NDSMM_X$ and $NDSMM_Y$	40
3.6	An example of $NDSMM$ which only contains two states S_0 and S_1 and the elements of analyzing data belong to the finite set $\{0, 1\}$	44
4.1	A Generic Modeling Component (GMC) is composed of two major sub-components: (a) $NDSMM$ Builder Engine which builds $NDSMMs$; (b) $NDSMM$ Repository which stores built $NDSMMs$ for future applications.	48
4.2	The overall view of workload flows in an Enterprise Application Server (EAS). Different workload sources send their requests to the EAS , and the dispatcher inside the EAS distributes requests to appropriate applications. And afterwards dispatchers inside applications distribute requests to their components.	52
4.3	Modeling the incoming workload to EAS as a whole. In this approach the resulting $NDSMMs$ contain the general behavioral model of the workloads for the EAS	53
4.4	Modeling the incoming workload to each application inside the EAS separately. In this approach the resulting $NDSMMs$ contain the workload patterns of the workloads for each application.	54

4.5	Modeling the incoming workload to each component inside the <i>EAS</i> applications separately. In this approach the resulting <i>NDSMMs</i> contain the behavioral patterns of the workloads for applications' components.	55
5.1	Integration of an instance of <i>GMC</i> into JOnAS J2EE which hosts RUBiS application	58

List of Acronyms

The following acronyms are used frequently in this report with these meanings:

Term	Description
API	Application Programming Interface
DBMS	DataBase Management System
DMC	Dynamic Markov Compression
DMM	Dynamic Markov Model
EAS	Enterprise Application Server
EJB	Enterprise Java Bean
GMC	Generic Modeling Component
GUID	Global Unique Identifier
GVF	Graph Visualization Framework
J2EE	Java 2 Platform Enterprise Edition
JAAS	Java Authentication and Authorization Service
JCS	Java Cryptography Service
JDBC	Java DataBase Connectivity
JMX	Java Management eXtensions
JMS	Java Messaging Service
JOOnAS	Java Open Application Server
JTA	Java Transaction API
JTS	Java Transaction Service
NDSMM	Novel Dynamic Semi-Markov Model
PDF	Probability Distribution Function
RMSE	Root Mean Square Error
TPC-W	Transaction Processing Performance Council - (Web commerce)
VLMM	Variable Length Markov Model
XML	Extensible Markup Language

Chapter 1

Introduction

1.1 Problem and Motivation

The performance evaluation of e-business enterprises is an essential task which requires efficient approaches to be used [21]. The performance of software systems is significantly affected by their incoming workload and their operational environment (i.e. hardware, software configuration) (cf. Figure 1.1) [53]. Thus, performance analysis must be based on the processing workloads of target systems. In other words, the analysis of systems performance without considering their processing workloads is not only meaningless but also fruitless. Therefore, to improve the performance of such systems, we need to analyze the workloads they process. Through workload analysis, we can find the bottlenecks and issues which reduce the throughput of software systems. Also, one of the most important goals of performance analysis and evaluation is to compare different systems capabilities [20]. Usually this comparison is carried out to decide which system to buy [20]. However, to perform a fair comparison we need to measure the performance of all systems based on the same workloads.

Analyzing of the real workload is an extremely difficult task, but reproducing of

workload with specific features is also not easily feasible. Using modeling approaches to build a generic workload model for a given system is an appropriate solution to overcome the complexity of workload analysis. A workload model is a generalized behavioral model of a given system whose parameters can be changed easily to reflect possible workloads in different operational environments [9]. A good workload model contains both static characteristics and dynamic behavior of a real workload and it is compact, repeatable and accurate [9].

Figure 1.1: Performance analysis of any software system must be done considering its operational environment and processing workload.

Statistical workload models are well known alternatives to using real workloads [60]. Markovian models [25] as powerful statistical modeling tools have been widely used for the purpose of modeling the workload processed by software systems [15] [17] [56] [60]. Also some combinatorial approaches have been investigated, which take advantage of combining Markovian models with other techniques such as clustering [19], and neural networks [22]. However, proposed solutions only focus on extraction of sequential patterns of workload. They do not propose any method to capture

patterns of intervals between incoming requests, as building blocks of workload, and their required process times.

Any major organization needs to formulate an efficient strategy to build a responsive, flexible and scalable e-business infrastructure [41]. The core of this infrastructure for an e-business enterprise is an Enterprise Application Server (EAS) [41]. The low performance of an application server significantly affects the performance of all integrated applications [16][26]. Thus, the performance analysis of EASs is a very critical task which must be done in all e-business enterprises. Unfortunately, this performance analysis is an expensive process and the results are only usable for the specific analyzed system [59]. The literature survey in the workload modeling area reveals the lack of specific investigations on workload modeling solutions for EASs. Therefore, an effort is needed to find efficient workload modeling solutions specifically for EASs.

1.2 Proposed Solution

Enterprise application servers as server programs in distributed infrastructures [41] provide development and deployment facilities to integrate all of an organization's applications and back-end systems. Here, in this paper we are interested in workload modeling for enterprise application servers. We view workload modeling as the process of analyzing the sequence of incoming requests and finding patterns across method invocations on hosted applications and components.

Investigated approaches in the literature on workload modeling only focus on extraction of sequential patterns of incoming requests [15] [17] [56] [60]. They do not discuss solutions for capturing patterns of intervals between incoming requests and their process times. Therefore, in this paper we illustrate a Novel Dynamic

Semi-Markov Model (*NDSMM*) which can be used to efficiently model the incoming workload toward an enterprise application server. Our modeling approach enhances current Markov modeling solutions [31] [60] for accurate workload modeling. Our contributions on workload modeling are as follows:

- Presenting *NDSMM* which captures patterns of intervals between incoming requests, the required times to process them, and also their sequential model. In other words, it finds patterns across incoming requests and also statistical models of requests intervals and process times. *NDSMM* is designed with this goal in mind to be an accurate and efficient workload modeling tool for enterprise application servers. A *NDSMM* is a temporal structure [64] in which transitions and states are labelled with probability distributions. The probability distributions attached to a transition describe the speed of evolution from one state to another and also the probability of the occurrence of that transition in terms of discrete distribution functions. The probability distribution attached to a state is also a discrete distribution function but it describes the idle time spent when a transition evolves the system to a new state. Consequently, being a change in the system - a transition from the current state to another - depends on the attached time distribution functions to states and transitions.
- Defining algorithms to build and adjust *NDSMMs*. The algorithm used for building *NDSMMs* is a heavily modified version of the dynamic modeling approach used in Dynamic Markov Compression (DMC) [12] by Gordon Cormack and Nigel Horspool. Also the applicability and efficiency of this modeling approach has been proven before, in the literature on workload modeling with the purpose of developing a predictive automatic tuning service for object pools [57].

A Feedback-based adjustment algorithm is given which helps to tune parameters of the *NDSMM* building algorithm accurately and efficiently. This offered adjustment technique includes three fundamental elements: (a) prediction, (b) scoring and (c) accuracy calculation. Considering these three elements, more efficient and customized versions of this algorithm can be designed to model different systems effectively.

- Proposing an operator for comparing two *NDSMMs* and measuring their similarities which gives out a numerical quantity. Using this operator, we can effectively reduce the redundancy of a repository of *NDSMMs*. This comparison operator is a tool to find how close two *NDSMMs* are. This operator accepts two *NDSMMs* as its input and returns a value which shows the similarity of those two *NDSMMs*. The return value, where we are comparing two *NDSMMs*: $NDSMM_X$ and $NDSMM_Y$, is equal to R , a number between 0 and 1. $R = 0$ means $NDSMM_X$ has captured behavioral features from the modeling system which none of them has been captured by $NDSMM_Y$. $R = 1$ shows the fact that $NDSMM_X$ and $NDSMM_Y$ have extracted exactly the same features of the modeling system. And any return value between these two describes the amount of behavioral features of the modeling system which have been captured by both of $NDSMM_X$ and $NDSMM_Y$.
- Introducing a Generic Modeling Component (*GMC*) which uses *NDSMMs* to model the workload of a given *EAS*. A *GMC* is an element which is responsible for building and storing *NDSMMs* generated for a given workload of an *EAS*. This component is composed of two major sub-components:

1. *NDSMM* Builder Engine which builds *NDSMMs* using the designed dynamic *NDSMM* building algorithm.
2. *NDSMM* Repository which stores built *NDSMMs* for future applications. This repository also keeps the amount of redundancy in the storage as low as possible using the designed *NDSMMs* comparison operator.

The integration of *GMCs* into *EASs* is an important issue which significantly affects the result of workload modeling. Therefore, different integration approaches are discussed to show how to integrate *GMCs* into an *EASs*.

1.3 Thesis Organization

The rest of this thesis report is organized as follows:

Chapter 2 provides background about enterprise application servers and explains fundamental concepts of workload modeling.

Chapter 3 contains the formal definition of a novel Markovian model which is an intuitive combination of a Semi-Markov Model[39] and a Dynamic Markov Model [12]. This chapter also includes the design and implementation details of the proposed Markovian model.

Chapter 4 proposes a Generic Modeling Component (*GMC*) to facilitate the application of *NDSMMs* for workload modeling of enterprise application servers. It shows how the *GMC* uses *NDSMMs* to model the workload of a given *EAS*. Furthermore, it discusses different possibilities to integrate *GMC* instances into an *EAS*.

Chapter 5 illustrates experiments and evaluations carried out on *NDSMM*-based workload modeling. It investigates the efficiency and applicability of the proposed

workload modeling approach for real-world applications.

Finally, Chapter 6 describes the most important contributions of this report and reviews all its achievements. Subsequently, different possibilities to extend this work are presented.

Chapter 2

Background

2.1 Enterprise Application Servers

All major organizations need to formulate an efficient strategy to build a responsive, flexible and scalable e-business infrastructure [41]. This infrastructure allows facilities to integrate their software applications into the enterprise systems with minimum cost and effort. The core of this infrastructure for an e-business enterprise is enterprise application servers [41]. EASs provide secure access to system-level environmental features which significantly reduce the cost of development and integration of enterprise systems [70][16]. The development of these system-level features such as transaction processing, database interaction, concurrency control, and memory management is a monotonous and error prone task for developers [70][46]. Therefore, using an EAS as a middleware [28] is a necessity to integrate software applications existing in an e-business enterprise with minimum cost. The main responsibility of any middleware is to make communication and collaboration possible between software applications with minimum dependency and maximum level of transparency [11]. However, satisfying the minimum dependency and maximum transparency in

an EAS comes along with performance degradation. The low performance of an application server significantly affects the performance of all integrated applications [16][26]. Thus, the performance analysis of EASs is a very critical task which must be done in all e-business enterprises. However, unfortunately this performance analysis is an expensive process and the results are only usable for the specific analyzed system [59]. An EAS must support the following requirements [41]:

- Scalable infrastructure: An EAS must be scalable in following aspects:
 - Number of users interacting with EAS concurrently.
 - Adding new features and applications on-demand without having any side effect on other integrated applications.
 - Supporting different e-business technologies. (i.e. mobile devices, new network technologies, and new business models.)
- Dynamic configuration: An EAS must provide facilities for configuration and reconfiguration of features and applications with the minimum amount of demanding effort and required downtime which causes the interruption of services offering by EAS.
- Deep integration: An EAS must be able to integrate applications in different programming languages, operating systems, hardware platforms, network protocols and data formats.
- Reliability and robustness: A system failure in an EAS affects not only internal e-business activities but also those of external ones. Therefore, reliability and robustness besides satisfying information consistency and confidentiality is a necessity.

The most commonly used EASs implement a component-based architecture [50]. An overall architecture view of an EAS with component-based architecture is shown in Figure 2.1 [35].

Figure 2.1: The overall architecture of a component-based EAS [35].

This architecture provides an infrastructure for integrating applications made of several components [29] (i.e. composable but independently deployable modules) and a set of common services (i.e. transaction management, concurrency control, security management, ...)[35]. Applications implement client call interfaces to let clients communicate with them. To take advantage of common services available in an EAS, applications use a common service interface. This standard interface hides the complexity of common services and prepares an appropriate level of transparency.

In contrast, an EAS uses specific Application Programming Interfaces (*APIs*) to implement the offered common services. In this way, the complexity of dealing with different specific APIs is hidden inside EAS and applications use a standard interface. This architecture has the following advantages [35][29][8]:

- Providing an appropriate level of encapsulation for developers. Therefore, developers should only concentrate on the development of components as business logic units. Hiding the complexity of common services and building up applications based on component-based architecture, helps to manage the complexity of the whole e-business enterprise as it scales up. For example, when developers are implementing a component which is responsible for process of online transactions, they do not need to worry about the complexities of secure storage and retrieval of information. So they only focus on implementation of business logic and use exported common services interface to enable transaction processing on their component.
- Offering standard API for using common services results in achieving portability and interoperability. Therefore, any component built based on this API can be reused on any platform which supports this API. For example, switching between different transaction processing monitors [5] does not affect the applications using the transaction service through provided common interface API by EAS. Also, integration into legacy systems is possible through developing a wrapper which exports this standard API.
- Improving the scalability by separation of applications components and common services. This separation makes it possible to scale up the providers of common

services without applying any change on application components. Therefore, in the case of increase of clients and consequently the processing workload, we should only improve our back-end systems without making any change to our applications. For example, persistence service, which is a common service for information storage and retrieval, uses a DataBase Management System (DBMS) as a service provider. To improve the response time of the persistence service, we only need to apply load balancing strategies [52] provided through our DBMS.

Figure 2.2: The architecture of JOnAS J2EE application server [2].

An example of component-based EAS is Java Open Application Server (JOnAS) J2EE [2] application server which we will use in our experimentations later on. The

architecture of this EAS is shown in Figure 2.2 [2]. In this architecture, building blocks of an application are Enterprise Java Bean (EJB) components which are developed in Java programming language [36] and reside inside a EJB container [55]. EJB is a component technology which introduces a software component model to design, develop and deploy enterprise-level components [38]. EJB components implement business logic which is required for an enterprise e-business application [7]. EJB clients send requests to EJB components as if they were sending them directly to EJB components. However, in fact their requests are being sent to proxies which delegate the requests to EJB components. These proxies and supporting modules form the EJB container [55]. The EJB container is also responsible to make the common services available to EJB components [7]. The most important common services offered by JOnAS are as follows [2]:

- Java DataBase Connectivity (JDBC) which is a industry standard for database-independent connection between modules developed in Java and a wide range of DataBase Management Systems (DBMSs) [51].
- Java Messaging Service (JMS) which provides a standard way to asynchronously send requests to distributed applications. This service is a very useful tool to integrate a legacy application into an e-business enterprise [44].
- Java Transaction Service (JTS) which is a transaction management service which implements Java Transaction API (JTA) specification which guarantees atomicity, consistency, isolation and durability all over transactions [37].
- Java Cryptography Service (JCS) which provides encryption and decryption facilities according to Java Cryptography Architecture (JCA) introduced in the

Java platform [49].

- Java Authentication and Authorization Service (JAAS) which is a user access control and permission management system to assure confidentiality and privacy in an e-business enterprise [49].
- Java Mail which is a service to let developers build mail applications which are platform-independent and protocol-independent [3].
- Java Management eXtensions (JMX) which is a universal approach to manage and monitor enterprise applications in a distributed architecture [27].

2.2 Workload Modeling

Performance evaluation of e-business enterprises is an essential task which requires efficient approaches to be used [21]. One of the most important goals of performance evaluation is to compare different systems capabilities [20]. Usually this comparison is carried out to decide which system to buy [20]. However, to perform a fair comparison we need to measure the performance of all systems based on the same workloads. The workloads which are used for this purpose are called benchmarks [20]. Generally we can categorize workloads into two main groups:

- Natural workloads or workload traces which are the activities logs collected for a specific system. These logs contain the exact trace information about performed activities inside a system. Natural workloads are the most real workloads for a system [20].
- Synthetic workloads or workload models which are general models of activities

in a specific system [20]. Passing workload models to a synthetic workload generator, we can generate sample workloads for any target system [34] (cf. Figure 2.3).

Figure 2.3: Sample workload generation based on workload models.

We can view workload models as result of a heavy analytical process on natural workloads. Therefore, usually all information existing in workload models can be achieved from natural workloads. However, sometimes during the analytical process we can integrate information achieved from other sources of information into natural workloads. Advantages of using synthetic workloads over natural workloads are as follows [20] [20] [6] [60]:

- Using workload models, we can generate workloads which are statistically the same but not identical. In other words, generated workloads have the same statistical characteristics while they do not contain identical workload elements.

However, if we use natural workloads, then generated workloads will be always identical.

- Workload models can contain workload characteristics which can not be found in workload traces directly without any further analysis. For example, in workload traces we do not know how different sequences of workload elements depend on each other. However, workload models have this power to show the dependency between different elements of workloads.
- Workload models increase our understanding about the nature of workload. Therefore, we can easily manipulate and adjust the parameters of workload models to generate sample workloads which have specific characteristics. In this sense, workload models are more flexible because of their parametric nature.
- Workload models are more efficient compared to natural workloads. Using models we can run shorter simulations or experiments (i.e. workloads with less number of building elements) and still get the same result as using a long simulation based on a natural workload.

Different types of Markovian models as powerful statistical analysis tools have been widely used for workload modeling purposes [31] [15] [17] [60]. The next section provides an overview of basic knowledge about Markovian models.

2.3 Markovian Models

Modern probability theory investigates stochastic processes for which the occurrence of previous events affects predictions for the future events [25]. These stochastic

processes are called Markov chains (cf. Figure 2.4) and it is named after Andrey Andreyevich Markov whose research was on theory of stochastic processes [42].

Definition 1 (Markov Chain)

A Markov chain is composed of a set of states, $S = \{ S_1, S_2, \dots, S_n \}$. The process starts in $S_x, 1 \leq x \leq n$ with probability of P_x (i.e. initial state probability) and moves to $S_y, 1 \leq y \leq n$ with probability of $P_{x,y}$ (i.e. transition probability) and so on [25].

Figure 2.4: An example of Markov chain.

According to the given definition above, all Markov chains have the property that the next state in the process only depends on the current state and it is independent

of previous steps [25]. This Markov chain also is called a first order Markov model because of the dependency of each transition only on the current state [4]. Higher order Markov models are definable by extending the definition of a Markov chain. In higher order Markov models, the choice of next state depends on the past k states which have been seen in the process, not only the current state [20]. The advantage of higher order Markov models is in their power to capture high order temporal dependencies between state transitions [57].

Definition 2 (k^{th} order Markov model)

A k^{th} order Markov model contains a set of states, $S = \{ S_1, S_2, \dots, S_n \}$. The process starts in $S_x, 1 \leq x \leq n$ with probability of P_x (i.e. initial state probability) and moves to $S_y, 1 \leq y \leq n$ with probability of $P_{x,y} \mid L$ (i.e. conditional transition probability), where L is a sequence of states with length k which has been seen before reaching the state S_x , and so on [25].

An extension of a k^{th} order Markov model is to consider a probability distribution which describes the time between state transitions [39]. This type of Markov model is called k^{th} order semi-Markov model (cf. Figure 2.5) and allows us to model the intervals between state transitions using probability distributions.

Definition 3 (k^{th} order semi-Markov model)

A k^{th} semi-Markov model is composed of a set of states, $S = \{ S_1, S_2, \dots, S_n \}$. The process starts in $S_x, 1 \leq x \leq n$ with probability of P_x (i.e. initial state probability) and moves to $S_y, 1 \leq y \leq n$, with conditional transition probability of $P_{x,y} \mid L$ (where L is a sequence of states with length k which has been seen before reaching the

Figure 2.5: An example of 1-st order semi-Markov model, where E shows an exponential distribution, D a deterministic distribution, W a Weibull distribution, and U a uniform distribution [25].

state S_x) and transition time of $T_{x,y}$ where T is a distribution function, and so on [39].

To use a fixed order Markov model, we need to know the length of required memory for prediction [23]. However, if the system which we are going to analyze has a dynamic and variable nature, using fixed order Markov models is not applicable [57]. In contrast, a Variable Length Markov Model (VLMM) [62] not only captures high-order temporal dependencies but also locally optimizes the required length of memory [23]. Dynamic Markov Model (DMM) as a variant of VLMM is introduced in [12] for the purpose of modeling a data stream which is going to be compressed/decompressed.

DMM has been proven in practice to be a very powerful tool for finding patterns in a sequence of events. DMM has been successfully used in applications such as spam filtering [18], vector compaction [40], workload prediction [57], human behavior prediction [48], and agricultural firm size projection [54].

A DMM starts with only one initial state then it extends one state at a time based on a cloning operation [68]. Therefore, we do not need to specify the internal states of a system before modeling. The basic idea of cloning operation is shown in Figure 2.6 [12].

Figure 2.6: An example of applying cloning operation on a partially built DMM [12].

2.4 Conclusion

In this chapter, the architecture of Enterprise Application Servers (EASs) [41] as major building blocks of any e-business enterprise is explained briefly. The necessary architectural requirements of an EAS are shown and advantages of such an architecture are investigated. Also, Java Open Application Server (JOnAS) [2] J2EE application server as a component-based EAS is introduced. The concept of workload modeling and related necessities and requirements are illustrated. Finally, Markovian models as powerful statistical analysis tools which have been used widely for workload modeling purposes are studied. In Chapter 3, a novel Markovian model which is an intuitive combination of semi-Markov model [39] and dynamic Markov model [12] is presented.

Chapter 3

Novel Dynamic Semi-Markov Model

This chapter is divided into two main sections. Section 3.2 contains the formal definition of a novel Markovian model which is a an intuitive combination of semi-Markov model[39] and dynamic Markov model [12]. This section also includes the design and implementation details of the proposed Markovian model. In Section 3.3, a similarity measurement operator is designed and implemented to calculate the similarity between two given *NDSMMs*.

3.1 Novel Dynamic Semi-Markov Model

This section introduces a Novel Dynamic Semi-Markov Model (*NDSMM*) which dynamically adapts itself to analyzing data. *NDSMM* is designed with this goal in mind to be an accurate and efficient workload modeling tool for enterprise application servers. A *NDSMM* is a Kripkle or temporal structure¹[64] in which transitions and states are labeled with probability distributions. The probability distributions attached to a transition describe the speed of evolution from one state to another

¹Kripkle or temporal structures are labeled state-transition structures widely used to represent the behavioral model of systems.

and also the probability of the occurrence of that transition in terms of discrete distribution functions. The probability distribution attached to a state is also a discrete distribution function but it describes the idle time spent when a transition evolves the system to a new state. Consequently, being a change in the system - a transition from the current state to another - depends on the attached time distribution functions to states and transitions. This dependency which is in contradiction with Markov property[14] makes *NDSMM* a semi-Markov[39] rather than Markov. In the following, we give the formal definition of *NDSMM*.

Definition 4 (Novel Dynamic Semi-Markov Model)

Let AP be a finite set of tokens that can be seen in analyzing data. A *NDSMM* is a 5-tuple (S, T, P, PDF_S, PDF_T) where:

- S is a finite set of states.
- T is a finite set of discrete time units $[t_{min}, t_{max}]$.
- $P : S \times AP \rightarrow S \times [0, 1]$ is the transition function which states the probability of evolving the system from a state to another when a specific element is seen in the analyzing data.
- $PDF_S : S \times T \rightarrow [0, 1]$ is the idle time probability matrix, satisfying:

$$\sum_{t \in T} PDF_S(s, t) = 1, \forall s \in S.$$

- $PDF_T : S \times AP \times T \rightarrow [0, 1]$ is the evolution speed probability matrix when a specific element is seen in analyzing data, satisfying:

$$\sum_{t \in T} PDF_T(s, a, t) = 1, \forall s \in S, \forall a \in AP.$$

Example 1 (A NDSMM with only two states)

Figure 3.1 shows a simple *NDSMM* which only contains two states S_0 and S_1 and the tokens of the analyzing data belongs to the finite set $\{0, 1\}$.

Figure 3.1: An example of *NDSMM* which only contains two states S_0 and S_1 and the elements of analyzing data belong to the finite set $\{0, 1\}$.

As shown in Figure 3.1, when the system is in the state S_0 the probability of seeing '0' or '1' as the next token in the analyzing data equals to 0.5. The observation of '0' as the next token when the system is in the state S_0 evolves the system back to the current state and the observation of '1' evolves the system to the new state S_1 . In both cases the system leaves the current state and it takes a probabilistic amount of time to come back to the current state or move to a new state. This probabilistic amount of time which is attached to each transition describes the evolution speed through that specific transition. For example evolution speed for transitions from state S_0 which evolve the system to states S_1 is $PDF_T(S_0, 1, t)$, $t \in T$. This *PDF* states that how much is the probability of taking t units of time to system evolve

from S_0 to S_1 . After each evolution when the system reaches its new state there will be an idle time before the observation of the next token. This idle time is also a probability distribution function attached to each available state in the system. For example the idle times for state S_0 is $PDF_S(S_0, t)$, $t \in T$ which shows how much is the probability of being idle in S_0 for t units of time.

The notable fact about both the evolution speeds and idle times is that both are discrete distribution functions [33] and not continuous [63]. No continuous distribution function is able to capture the behavioral patterns of idle times and evolution speeds accurately. Because idle times and evolution speeds do not have a specific shape in all edges and states of *NDSMMs*. Even if a continuous function has this capability, we still need to find its parameters separately for each idle time and evolution speed function attached to each state and edge. Finding these parameters is a very time consuming process [65] and in our application it should happen frequently. To avoid these time consuming process of finding parameters which do not even give us the accuracy required, we take advantage of discrete distribution functions.

3.1.1 Dynamic Building Process

The formal definition of *NDSMM* mentioned above describes the static structure of a *NDSMM* and the example shown afterward illustrates how a typical *NDSMM* looks like at a time snapshot of its lifecycle. However, one significant aspect of a *NDSMM* is its dynamic nature. A *NDSMM* changes dynamically during its lifecycle to find the best fit model for a specific system. The dynamic process of building a *NDSMM* brings along two significant advantages:

- When all or some internal states of a system are unknown, the dynamic building

process starts building the *NDSMM* with only one state and then extracts the others through the statistical analysis of the processing data. This speeds up the modeling process and makes it possible to easily model systems with unperceivable internal states.

- A *NDSMM* is useful even before being close enough to the best fit model. In other words, a *NDSMM* can be used during the dynamic building process when it is not as accurate as the final built model. This feature is so critical for real-time systems [58] where late results of a process are considered incorrect.

Before giving a formal definition for the best fit model, an accuracy measurement tool is required. For this sake, it is shown how to measure Root Mean Square Error (RMSE) [69] of a *NDSMM* based on a stream of sample data taken from the modeling system.

3.1.2 RMSE of a NDSMM

Let us assume a stream of analyzing data from a specific system is described as a finite sequence of 3-tuples $(I_0, A_0, E_0) \rightarrow \dots \rightarrow (I_i, A_i, E_i) \rightarrow \dots \rightarrow (I_n, A_n, E_n)$ where I_i shows the idle time before the observation of the next token in the stream of analyzing data, A_i is the next token which system emits after spending the idle time, and E_i is the evolution speed which describes the time that system spends to emit A_i . Processing the tuples of this sequence through the steps mentioned below generates a prediction sequence which will be used directly to compute the *RMSE* of a model:

Algorithm 1 (Prediction Sequence Generation)

1. Define S as a state

2. $S \leftarrow$ Starting state of *NDSMM*
3. $j \leftarrow 0$
4. For each $i \in [0, n]$
 - (a) Define $P_t^j, P_t^{j+1}, P_t^{j+2}$ as probabilities
 - (b) $P_t^j \leftarrow PDF_S(S, I_i)$
 - (c) $P_t^{j+1} \leftarrow P(S, A_i)$
 - (d) $P_t^{j+2} \leftarrow PDF_T(S, A_i, E_i)$
 - (e) Add P_t^j, P_t^{j+1} , and P_t^{j+2} to the prediction sequence
 - (f) $S \leftarrow P(S, A_i)$
 - (g) $j \leftarrow j + 3$

Subsequently using the resulted prediction sequence generated through Algorithm 1, *RMSE* of a model can be computed through the following formula [69]:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{n*3} (P_t^i - 1)^2}{n * 3}}$$

Let us assume we have a sequence of 3-tuples as follows: $(t_0, 0, t_2) \rightarrow (t_1, 1, t_3) \rightarrow (t_2, 1, t_0)$. We generate the prediction sequence using Algorithm 1 according to *NDSMM* shown in Example 1: $(0.2, 0.5, 0.1) \rightarrow (0.2, 0.5, 0.5) \rightarrow (0.1, 0.25, 0.2)$. Then we calculate *RMSE* of the *NDSMM* using the given formula:

$$RMSE = \sqrt{\frac{.64 + .25 + .81 + .64 + .25 + .25 + .81 + .56 + .64}{3 * 3}} = 0.73$$

Next comes the formal definition of the best fit model which defines the ideal goal of building a *NDSMM*.

Definition 5 (The Best Fit Model)

A *NDSMM* is the best fit model of a specific system when its *RMSE* for any sample data from that system equals to zero.

However, building the best fit models in the real world is almost impossible in most cases. Therefore, the ideal goal of modeling a system can be stated as building the closest model to the best fit model. In other words, the modeling process aims to reduce the *RMSE* of a model as much as possible.

3.1.3 Algorithm Design

Having the formal definition of the ideal goal for modeling a system, it is time to dig into the dynamic process of building a model in details. The algorithm used for this purpose is based on the dynamic modeling approach used in Dynamic Markov Compression (DMC)[12] by Gordon Cormack and Nigel Horspool, though it is heavily modified to be applicable for the semi-Markov model proposed above. Also the applicability and efficiency of this modeling approach has been proven before, in the literature on workload modeling with the purpose of developing a predictive automatic tuning service for object pools[57]. Basically the algorithm starts with a *NDSMM* which only contains one state S_0 and this starting *NDSMM* can be illustrated as follows: (Assume $AP = \{a, b\}$)

- $P_t^1 = PDF_S(S_0, t)$ where $PDF_S(S_0, t)$ is a uniform discrete distribution. So we can say for all values of $t \in T$ we have $PDF_S(S_0, t) = 1/(t_{max} - t_{min})$ which also satisfies $\sum_{t=t_{min}}^{t_{max}} PDF_S(S_0, t) = 1$.

- $P(S_0, a) = P(S_0, b) = (S_0, 0.5)$ which states observation of all allowed tokens (i.e. a and b) in the analyzing data evolves the system back to S_0 and the probability of observation of each is equal.
- $P_t^2 = PDF_T(S_0, a, t)$ and $P_t^3 = PDF_T(S_0, b, t)$ where both of them are again uniform discrete distributions, so we have $PDF_T(S_0, a, t) = PDF_T(S_0, b, t) = 1/(t_{max} - t_{min})$ and also $\sum_{t=t_{min}}^{t_{max}} PDF_T(S_0, a, t) = \sum_{t=t_{min}}^{t_{max}} PDF_T(S_0, b, t) = 1$.

Subsequently the *NDSMM* mentioned above sets its only state as the current state, namely S_C , and then goes through the following steps for each 3-tuple (i.e. (I_i, A_i, E_i)) in the analyzing data. These steps dynamically evolve the *NDSMM* into a model closer to the best fit model.

Algorithm 2 (Building a NDSMM Dynamically)

1. Update the idle time distribution of the current state based on the idle time I_i . To store idle time distributions, each *NDSMM* uses a two dimensional array: $IDLE[S_0..S_{MAX}][t_{min}..t_{max}]$. Consequently, to figure out the probability of spending t_i time units in state S_C , the following formula can be used:

$$PDF_S(S_C, t_i) = IDLE[S_C][t_i] / \sum_{t=t_{min}}^{t_{max}} IDLE[S_C][t]$$

So, to update the idle time distribution of state S_C when the idle time I_i is in the current processing 3-tuple of the analyzing data, we need to only perform the following operation:

$$IDLE[S_C][t_i] \leftarrow IDLE[S_C][t_i] + 1$$

2. Update the transition time distribution of the transition which evolves the system to its next new state based on the observation of A_i which takes E_i time duration. A three dimensional array $SPEED[S_0..S_{MAX}][A_0..A_{|AP|}][t_{min}..t_{max}]$ is used to store transition time distributions for each *NDSMM*. Therefore, to compute the probability of evolving from state S_C after the observation of A_i within E_i time, we have:

$$PDF_T(S_C, A_i, E_i) = SPEED[S_C][A_i][E_i] / \sum_{t=t_{min}}^{t_{max}} SPEED[S_C][A_i][t]$$

So, to update the transition time distribution of evolution from the current state S_C with observation of A_i when the transition time E_i is in the current processing 3-tuple of the analyzing data, we need to only perform the following operation:

$$SPEED[S_C][A_i][E_i] \leftarrow SPEED[S_C][A_i][E_i] + 1$$

3. Update the transition probability of evolving from the current state as a result of the observation of A_i . For this sake, there is a couple of two dimensional arrays: $MOVE[S_0..S_{MAX}][A_0..A_{|AP|}]$ and $COUNT[S_0..S_{MAX}][A_0..A_{|AP|}]$ attached to each *NDSMM*. The array *MOVE* shows which state the system evolves to from each state after the observation of each token and the array *COUNT* keeps track of the number of times each token has been observed in each state of the model. These two arrays are the building elements of the transition function P :

$$P(S_C, A_i) = (MOVE[S_C][A_i], COUNT[S_C][A_i] / \sum_{a \in AP} COUNT[S_C][a])$$

Then updating the transition probability of evolving from the current state after the observation of A_i is done by:

$$COUNT[S_C][A_i] \leftarrow COUNT[S_C][A_i] + 1$$

4. If it is suitable add a new state(s) to the model to reduce its *RMSE* and afterwards adjust involved parameters of the model. Adding a new state to the model which we refer to as cloning is an effective way to improve its fitness. The idea is when transition to the state $S_N = MOVE[S_C][A_i]$ from the current state S_C is more probable than any other state, we clone the state S_N to capture more details about that specific evolution in the system. But how can we determine when to clone a state? We can use the cloning conditions proposed in DMC[12] and adapt it to the definition of *NDSMM*. So when the following conditions are satisfied cloning happens:

$$(a) \ COUNT[S_C][A_i] \geq Threshold_1$$

$$(b) \ \sum_{a \in AP} COUNT[S_N][a] - \sum_{a \in AP} COUNT[S_C][a] \geq Threshold_2$$

First condition is to make sure the transition from S_C to S_N has taken enough times (i.e. more than $Threshold_1$ times). And the second condition is to assure that transitions to S_N from S_C is more probable than any other state and $Threshold_2$ is used to describe this condition. But how do we clone a state? the cloning process will be shown in lines 11 to 16 of the pseudo-code coming after this overall algorithm description.

5. Change the current state of the model according to the observed token A_i . Having the array *MOVE* as explained above based on the observation A_i , this change of state is carried out as follows:

$$S_C \leftarrow MOVE[S_C][A_i]$$

3.1.4 Algorithm Implementation

Knowing the overall design of the algorithm for building a *NDSMM* for a specific system, we are ready to explore the detailed implementation of this algorithm which is offered here in pseudo-code:

1. Define S_C and S_N as states
2. Define MAX_S as constant maximum number of states
3. $NUM_S \leftarrow 1$
4. $S_C \leftarrow S_0$
5. For each $(I_C, A_C, E_C) \leftarrow$ in the analyzing data of the system
 - (a) $S_N \leftarrow MOVE[S_C][A_C]$
 - (b) $IDLE[S_C][I_C] \leftarrow IDLE[S_C][I_C] + 1$
 - (c) $SPEED[S_C][A_C][E_C] \leftarrow SPEED[S_C][A_C][E_C] + 1$
 - (d) $COUNT[S_C][A_C] \leftarrow COUNT[S_C][A_C] + 1$
 - (e) $C1 = (NUM_S > MAX_S)$
 - (f) $C2 = (COUNT[S_C][A_C] < Threshold_1)$
 - (g) $C3 = (\sum_{a \in AP} COUNT[S_N][a] - \sum_{a \in AP} COUNT[S_C][a] < Threshold_2)$
 - (h) if $C1 == C2 == C3 == TRUE$
 - i. $NUM_S \leftarrow NUM_S + 1$

- ii. Define S_{NEW} as a new state
- iii. $ratio \leftarrow COUNT[S_C][A_C] / \sum_{a \in AP} COUNT[S_N][a]$
- iv. $MOVE[S_C][A_C] \leftarrow S_{NEW}$
- v. For each $a \in AP$
 - A. $MOVE[S_{NEW}][a] \leftarrow MOVE[S_N][a]$
 - B. $COUNT[S_{NEW}][a] \leftarrow ratio \times COUNT[S_N][a]$
 - C. $SPEED[S_{NEW}][a][t_{min}..t_{max}] \leftarrow SPEED[S_N][a][t_{min}..t_{max}]$
 - D. $COUNT[S_N][a] \leftarrow COUNT[S_N][a] - COUNT[S_{NEW}][a]$
- vi. $IDLE[S_{NEW}][t_{min}..t_{max}] \leftarrow IDLE[S_N][t_{min}..t_{max}]$
- (i) $S_C \leftarrow MOVE[S_C][A_C]$

Example 2 (Dynamic process of building a NDSMM)

Figure 3.2: An example of applying cloning operation on a NDSMM. Transitions in bold in each model show transitions which cause the cloning.

3.1.5 Adjustment of Parameters

To model real world systems accurately and efficiently using *NDSMMs*, we need to take the following points into account when we adjust parameters of the Algorithm 2 proposed above:

- Choosing small values for $Threshold_1$ and $Threshold_2$ will increase the growth speed of the model. In contrast, assigning large values to these parameters decrease the speed of adding new states to the model. Adding more states to the model helps to capture more details about the behavior of the system. However, it does not mean necessarily that the model will get closer to the best fit model faster. But how can we figure out optimal values for these parameters? Unfortunately there are no absolute values for these parameters which can be used to model all systems. An applicable way is to find their values specifically for each system through a feed-back based approach. A typical feed-back based solution measures the accuracy of the model actively and whenever the accuracy is not showing any change, the values of $Threshold_1$ and $Threshold_2$ will reduce. Different methods can be used to measure the accuracy of the model during the building process. A simple approach can be described by the pseudo-code shown below:

Algorithm 3 (*Feedback-based adjustment of NDSMM building parameters*)

1. Before reading the next 3-tuple, (I_C, A_C, E_C) , from the analyzing data stream, predict it as the 3-tuple of (I_X, A_X, E_X) where:

- (a) A_X is the most probable token that can be seen next in the analyzing data and it satisfies:

$$\forall a \in AP, a \neq A_X \Rightarrow COUNT[S_C][a] \leq COUNT[S_C][A_X]$$

- (b) I_X is the most probable idle time in the current state of the model which satisfies:

$$\forall t \in T, t \neq I_X \Rightarrow IDLE[S_C][t] \leq IDLE[S_C][I_X]$$

- (c) E_X is the most probable evolution time for the predicted next token A_X which satisfies:

$$\forall t \in T, t \neq E_x \Rightarrow SPEED[S_C][A_X][t] \leq SPEED[S_C][A_C][E_X]$$

2. Read the next 3-tuple, (I_C, A_C, E_C) , from the analyzing data stream, and then update the score of the predicted 3-tuple as follows:
 - if $A_C == A_X$ then $score \leftarrow score + 1$
 - if $I_C == I_X$ then $score \leftarrow score + 1$
 - if $A_C == A_X$ and $E_C == E_X$ then $score \leftarrow score + 1$
3. Then calculate the accuracy of the model during the building process using this formula where n is the number of predictions carried out:

$$accuracy = score / (3 * n)$$

This algorithm was just a simple version of feed-back based approach to show the fundamental elements which matter most in the design of similar algorithms. These fundamental elements are (a) prediction, (b) scoring and (c) accuracy calculation. Considering these three elements, more efficient and customized versions of this algorithm can be designed to adjust values of $Threshold_1$ and $Threshold_2$ in modeling different systems.

- Another parameter which needs to be adjusted in the building algorithm of a *NDSMM* is MAX_S which shows the maximum number of states that can exist in each *NDSMM*. The limitation on the maximum number of states is forced by available memory accessible to the modeling process. Therefore, the value of this parameter should be assigned according to the operational environment (i.e. available memory space) of the modeling process. However, one issue

which should be addressed is how to react when the number of states in a model reaches this maximum. One way is to just keep updating statistical information of the model without adding any other new state like what is shown in mentioned algorithm before. Other ways can matter considering the application of *NDSMM*. In Dynamic Markov Compression (DMC)[12] Gordon Cormack and Nigel Horspool suggest to discard all the model and start the modeling process from scratch again. Though this approach is useful when modeling is happening online (i.e. modeling a system which is emitting a stream of data which is being compressed/decompressed) and the modelled behavior is not going to be used in the future. Another way can be saving the built model in a secondary storage other than main memory and start building a new model. When the second model reaches the maximum allowed number of states, then we can compare the similarity of these two models. If both models have captured similar features of the system we can discard one of them and keep the other one. Otherwise again we add the second model to the secondary storage and start building another model and so on. Thus before saving a model in the secondary storage we assure that other equivalent models do not exist there. However, to implement this solution we need a comparison operator to compare the similarity of two *NDSMMs*. Therefore the next section will illustrate how to efficiently and effectively design and implement this operator.

3.2 Measuring the Similarity Between two *NDSMMs*

The comparison operator is a tool to find how close two *NDSMMs* are. This operator should accept two *NDSMMs* as its inputs and return a value which shows

the similarity of those two *NDSMMs*. The return value where we are comparing two *NDSMMs*: $NDSMM_X$ and $NDSMM_Y$ equals to R which is a number between 0 and 1. $R = 0$ means $NDSMM_X$ has captured behavioral features from the modeling system for which none of them has been captured by $NDSMM_Y$ (cf. Figure 3.3). $R = 1$ shows the fact that $NDSMM_X$ and $NDSMM_Y$ have extracted exactly the same features of the modeling system (cf. Figure 3.4). And any return value between these two describes the amount of behavioral features of the modeling system which have been captured by both of $NDSMM_X$ and $NDSMM_Y$.

Figure 3.3: Comparing $NDSMM_X$ and $NDSMM_Y$ when they are capturing totally different behavioral features of the modeling system.

To find the similarity of two *NDSMMs*: $NDSMM_X$ and $NDSMM_Y$, the comparison operator uses $NDSMM_X$ for sample data generation and then traces the generated sample data on $NDSMM_Y$ to calculate the level of similarity between them. The overall architecture of this operator is shown in Figure 3.5. A sample data generator is used to create sample data based on the given model $NDSMM_X$. This element also creates some metadata about the generated sample data. The generated metadata will be passed to the similarity calculator and the sample data will

Figure 3.4: Comparing $NDSMM_X$ and $NDSMM_Y$ when both of them are capturing exactly the same behavioral features of the modeling system.

be sent to the sample data tracer. The sample data tracer receives the generated sample data and traces it on the given model $NDSMM_Y$. The output of this tracing phase is also a metadata which will be passed to the similarity calculator. When the similarity calculator has the metadata received from the sample data generator and the sample data tracer analyzes them and returns the result of this analysis as the level of similarity between two given model $NDSMM_X$ and $NDSMM_Y$.

3.2.1 Sample Data Generation

Sample data generation based on a given $NDSMM$ is similar to random walking on a graph [45]. A random walk on a $NDSMM$ with the purpose of sample data generation can be done by following the steps shown below: (Assuming n is the number of demanded 3-tuples in the sample data)

Algorithm 4 (Random walk on a NDSMM)

1. Set the current state to the starting state of the given $NDSMM$: $S_C \leftarrow S_0$

Figure 3.5: The similarity calculator analyzes the received metadata from the sample data generator and the sample data tracer and returns the result of this analysis as the level of similarity between two given model $NDSMM_X$ and $NDSMM_Y$.

2. Define L_S as a list which stores the generated sample data.
3. Define L_M^G as a list which stores the metadata of the generated sample data.
4. Repeat following steps n times:
 - (a) Generate a random idle time I_R from the domain $T = [t_{min}..t_{max}]$ using the idle time discrete distribution function $PDF_S(S_C, t), t \in T$ of the current state:

$$I_R \leftarrow RandomGen(T, PDF_S(S_C, t) \text{ where } t \in T)$$

- (b) Find the occurrence probability of the randomly generated idle time I_R where the current state is S_C and store it in M_{I_R} which is a metadata for I_R :

$$M_{I_R} \leftarrow PDF_S(S_C, I_R)$$

- (c) Generate a random token $A_R \in AP$ considering the probability distribution of tokens $P(S_C, a), a \in AP$ in the current state:

$$A_R \leftarrow \text{RandomGen}(AP, P(S_C, a) \text{ where } a \in AP)$$

- (d) Find the occurrence probability of the randomly generated token A_R where the current state is S_C and store it in M_{A_R} which is a metadata for A_R :

$$M_{A_R} \leftarrow P(S_C, A_R)$$

- (e) Generate a random evolution time $E_R \in T$ based on the evolution speed discrete distribution function $PDF_T(S_C, A_R, t), t \in T$ of the current state and A_R is observed as the next token:

$$E_R \leftarrow \text{RandomGen}(T, PDF_T(S_C, A_R, t) \text{ where } t \in T)$$

- (f) Find the occurrence probability of the randomly generated evolution time E_R where the current state is S_C and A_R is observed as the next token. Then store it in M_{E_R} which is a metadata for E_R :

$$M_{E_R} \leftarrow PDF_T(S_C, A_R, E_R)$$

- (g) Build a 3-tuple (I_R, A_R, E_R) and add it to L_S which keeps all randomly generated 3-tuples orderly:

$$\text{AddtoList}(L_S, (I_R, A_R, E_R))$$

- (h) Build a 3-tuple $(M_{I_R}, M_{A_R}, M_{E_R})$ and add it to L_M which keeps the metadata for all randomly generated 3-tuples orderly:

$$\text{AddtoList}(L_M^G, (M_{I_R}, M_{A_R}, M_{E_R}))$$

- (i) Move to the next state using the randomly generated token A_R :

$$S_C \leftarrow MOVE[S_C][A_R]$$

5. Send L_S to the sample data tracer and pass L_M^G to the similarity calculator.

To implement the function *RandomGen* which is used in the algorithm proposed above, we can use techniques that generate random numbers with a given distribution. For instance, the alias method is proposed by Walker [67] with initialization time complexity of $O(N * \log(N))$ - where N is the length of the domain which generated random numbers belong to - and generation time complexity of $O(1)$. However, there is also a variation of Walker's original model by Vose [66] whose initialization time complexity is linear $O(N)$ and generation time is the same $O(1)$. Consequently, using this latter to generate a random sample data which contains T of 3-tuples requires $O(T * N + T)$ time.

Assume in Example 1 that every PDF_S and PDF_T is a uniform distribution [25] and all time units are in the range 1 to 5 (i.e. $t \in [t_{min} = 1, t_{max} = 5]$). Then a random walk based on Algorithm 4 where $n = 5$ will produce the following outputs:

$$L_S = [1, 0, 2], [2, 1, 3], [4, 1, 5], [3, 0, 4], [2, 1, 5]$$

$$L_M^G = [0.2, 0.5, 0.2], [0.2, 0.5, 0.2], [0.2, 0.25, 0.2], [0.2, 0.75, 0.2], [0.2, 0.5, 0.5]$$

3.2.2 Sample Data Tracing

To trace sample data on a given *NDSMM*, we simply start from the starting state of the given *NDSMM* and move to new states based on the tokens in the sample data. The following steps show how to trace a sample data on a given *NDSMM* and gather the related metadata. (Let us assume L_S is the given sample data and its length is n .)

Algorithm 5 (Tracing sample data on a NDSMM)

1. Set the current state to the starting state of the given *NDSMM*: $S_C \leftarrow S_0$
2. Define L_M^T as a list which stores the metadata of the tracing process.
3. Repeat following steps n times:

- (a) Extract the next 3-tuple from the given sample data L_S to trace:

$$(I_C, A_C, E_C) \leftarrow GetNextTuple(L_S)$$

- (b) Find the occurrence probability of the idle time I_C where the current state is S_C and store it in M_{I_C} which is a metadata for I_C :

$$M_{I_C} \leftarrow PDF_S(S_C, I_C)$$

- (c) Find the occurrence probability of the token A_C where the current state is S_C and store it in M_{A_C} which is a metadata for A_C :

$$M_{A_C} \leftarrow P(S_C, A_C)$$

- (d) Find the occurrence probability of the evolution time E_C where the current state is S_C and A_C is observed as the next token. Then store it in M_{E_C} which is a metadata for E_C :

$$M_{E_C} \leftarrow PDF_T(S_C, A_C, E_C)$$

- (e) Build a 3-tuple $(M_{I_C}, M_{A_C}, M_{E_C})$ and add it to L_M^T which keeps the metadata for all traced 3-tuples orderly:

$$AddtoList(L_M^T, (M_{I_C}, M_{A_C}, M_{E_C}))$$

- (f) Move to the next state using the observed token A_C in the given sample data:

$$S_C \leftarrow \text{MOVE}[S_C][A_C]$$

4. Send L_M^T to the similarity calculator.

Figure 3.6: An example of *NDSMM* which only contains two states S_0 and S_1 and the elements of analyzing data belong to the finite set $\{0, 1\}$.

Let us assume in the *NDSMM* shown in Figure 3.6 that every PDF_S and PDF_T is a uniform distribution [25] and all time units are in the range of 1 to 5 (i.e. $t \in [t_{min} = 1, t_{max} = 5]$). If the given sample data is as follows:

$$L_S = [1, 0, 2], [2, 1, 3], [4, 1, 5], [3, 0, 4], [2, 1, 5]$$

Then a tracing L_S on the *NDSMM* shown in Figure 3.6 based on Algorithm 5 will result the following output:

$$L_M^T = [0.2, 0.75, 0.2], [0.2, 0.75, 0.2], [0.2, 0.25, 0.2], [0.2, 0.75, 0.2], [0.2, 0.75, 0.2]$$

3.2.3 Similarity Calculation

Similarity calculation analyzes the metadata received from the sample data generator and the sample data tracer (i.e. L_M^G and L_M^T respectively) with the purpose of finding the level of similarity between two given *NDSMMs*. The proposed approach in this thesis report is as follows:

Algorithm 6 (Similarity calculation between two NDSMMs)

1. $diff \leftarrow 0$
2. Repeat the following steps n times:
 - (a) Extract the next 3-tuple from the metadata L_M^G received from the sample data generator:

$$(M_{IG}, M_{AG}, M_{EG}) \leftarrow GetNextTuple(L_M^G)$$

- (b) Extract the next 3-tuple from the metadata L_M^T received from the sample data tracer:

$$(M_{IT}, M_{AT}, M_{ET}) \leftarrow GetNextTuple(L_M^T)$$

- (c) Calculate the average difference between extracted 3-tuples of L_M^G and L_M^T and add it up to the calculated average difference between past extracted 3-tuples:

$$diff \leftarrow diff + \frac{|M_{IG}-M_{IT}|+|M_{AG}-M_{AT}|+|M_{EG}-M_{ET}|}{3}$$

3. The similarity between L_M^G and L_M^T is equal to the complement of the average difference between all of their 3-tuples, so we have:

$$similarity \leftarrow 1 - \frac{diff}{n}$$

Let us assume that we have the following metadata received from the sample data generator and the sample data tracer which are respectively produced from *NDSMMs* shown in Figure 3.1 and Figure 3.6:

$$L_M^G = [0.2, 0.5, 0.2], [0.2, 0.5, 0.2], [0.2, 0.25, 0.2], [0.2, 0.75, 0.2], [0.2, 0.5, 0.5]$$

$$L_M^T = [0.2, 0.75, 0.2], [0.2, 0.75, 0.2], [0.2, 0.25, 0.2], [0.2, 0.75, 0.2], [0.2, 0.75, 0.2]$$

Then based on Algorithm 6, the average difference and similarity level between *NDSMMs* shown in Figure 3.1 and Figure 3.6 will be:

$$Average\ difference = (0.25/3 + 0.25/3 + 0 + 0 + 0.25/3)/5 = 0.05$$

$$Similarity = 1 - 0.05 = 0.95$$

3.3 Conclusion

In this chapter, the formal definition of the Novel Dynamic Semi-Markov Model (*NDSMM*) is given following by a simple example. Afterwards, the dynamic process of building *NDSMMs* is proposed in detail including design, implementation and adjustment approaches. Subsequently, a comparison operator is defined on *NDSMMs* which measures the level of similarity between two given *NDSMMs*. In Chapter 4, the application of *NDSMMs* to workload modeling of enterprise application servers is discussed. It includes how to design a Generic Modeling Component (GMC) and integrate it into enterprise application servers.

Chapter 4

NDSMM-based Workload Modeling

As shown in Chapter 3, *NDSMM* is a novel modeling tool which not only captures the probabilistic transition model of a system but also extracts the probabilistic idle time and evolution speed of the system in each of its possible states. These features allow *NDSMM* to be used as an efficient and strong key element in modeling complex systems such as Enterprise Application Servers (EAS). In this chapter we will see how to design a Generic Modeling Component (GMC) which uses *NDSMMs* to model the workload of a given *EAS*. Afterwards, integration approaches will be discussed to show how to integrate *GMC* into an *EAS*.

4.1 Generic Modeling Component (GMC)

A Generic Modeling Component (GMC) is an element which is responsible for building and storing *NDSMMs* generated for given workload of an *EAS*. As shown in Figure 4.1, this component is composed of two major sub-components:

- *NDSMM* Builder Engine which builds *NDSMMs* using the designed dynamic *NDSMM* building algorithm.

- *NDSMM* Repository which stores built *NDSMMs* for future applications. This repository also keeps the amount of redundancy in the storage as low as possible using the designed *NDSMMs* comparison operator.

Figure 4.1: A Generic Modeling Component (GMC) is composed of two major sub-components: (a) *NDSMM* Builder Engine which builds *NDSMMs*; (b) *NDSMM* Repository which stores built *NDSMMs* for future applications.

4.1.1 GMC Functionalities

This component implements the following functionalities which let other applications take advantage of features it offers for modeling purposes:

- *Creating a new model.* Through this functionality we can create new *NDSMMs* in *GMC*. When *GMC* creates a new model, it returns a Global Unique Identifier

(GUID) which uniquely identifies each *NDSMM* in the *GMC*. This *GUID* is used for future references to each *NDSMM*.

- *Updating an existing model.* *GMC* lets us update created *NDSMMs* by passing analyzing data tokens in the format of 3-tuples as discussed in Chapter 3. To specify which *NDSMM* is to be updated, we should use the associated *GUID* by the desired *NDSMM*.
- *Storing a model into the repository.* To be able to use created *NDSMMs* for future applications, *GMC* provides a repository which stores them in a persistent storage. This repository uses the *NDSMMs* comparison operator to prevent any redundancy. Every time that a *NDSMM* is asked to be saved in the repository, *GMC* compares the passed *NDSMM* with *NDSMMs* already existing in the repository. If the similarity between the passed *NDSMM*, say *NDSMM_X*, and none of *NDSMMs* in the repository is more than a given threshold, then *GMC* inserts a copy of *NDSMM_X* in the repository. Otherwise, *GMC* only returns the *GUID* of an *NDSMM* in the repository which is the most similar *NDSMM* to *NDSMM_X*.
- *Retrieving a model from the repository.* *GMC* offers this functionality to bring *NDSMMs* back to the *NDSMM* Builder Engine from the repository. *GMC* receives *GUIDs* of *NDSMMs* existing in the repository and loads the *NDSMMs* with given *GUIDs* into the *NDSMM* Builder Engine.
- *Destroying an existing model.* Sometimes we need to remove a *NDSMM* from the *NDSMM* Builder Engine or *NDSMMs* Repository. Therefore, *GMC* offers this feature to permanently destroy a *NDSMM* with a specific *GUID*. For a

given *GUID*, *GMC* searches both the *NDSMM* Builder Engine and *NDSMMs* Repository, and permanently destroys *NDSMMs* with the given *GUIDs*.

4.1.2 Using GraphXML to Store NDSMMs

It is very important to save the created *NDSMMs* in the *NDSMMs* Repository in a format which can be easily interpreted by other applications. For this purpose, *GMC* uses GraphXML [30], a graph description language based on XML [1], to store *NDSMMs* in the repository (see Appendix A). The major reason which makes the GraphXML an appropriate format for storing *NDSMMs* in the repository is that it is based on XML [1] which is used by many existing applications. Therefore, it has a high chance to be accepted by the community of software developers easily[30].

A significant advantage of using GraphXML lies in its potential capability for information visualization[61]. When we describe *NDSMMs* in GraphXML format, we can use utilities such as the Graph Visualization Framework (GVF)[43] to visualize them and also manipulate their structure manually.

4.2 Integration of GMC into Enterprise Application Servers

The integration of *GMC* into Enterprise Application Servers is an important issue which significantly affects the result of workload modeling. Therefore in this section we briefly investigate the possible integration approaches.

4.2.1 General Integration Approaches

To integrate *GMC* instances into an Enterprise Application Server (EAS), we should decide which instance is going to analyze what part of the incoming workload

into the *EAS*. Therefore, the main idea is to classify the incoming workload and then assign an instance of *GMC* to each class. One way to classify the incoming workload is to parse the network traffic coming into the *EAS*. However, in the real world it is easier to use the facilities offered by *EAS* to achieve this goal. Usually *EASs* introduce workload monitoring points in their architecture which can be used as integration points for instances of *GMC*. The overall view of workload flows in an *EAS* has shown in Figure 4.2. Different workload sources send their requests to the *EAS*, and the dispatcher inside the *EAS* distributes requests to appropriate applications. And afterwards dispatchers inside applications distribute requests to their components.

According to the workload flows shown in Figure 4.2, one way to classify the incoming workload is to classify all the incoming workload as a single class. In other words, we create only one instance of *GMC* and send all the incoming workload to it (cf. Figure 4.3). In this approach the resulting *NDSMMs* contain the general workload model of the workloads for the *EAS*.

However, if the goal of modeling is to extract the workload patterns of each application separately, then we need to use the approach shown in Figure 4.4. This integration approach lets us plug instances of *GMC* into the workload flows which are dispatched to each application through the *EAS* workload dispatcher. Therefore, there is a one-to-one association between *GMC* instances and the workload dispatching to each separate application inside the *EAS*. Also, we can use a variant of this integration approach which builds one-to-many association between *GMC* instances and applications. In this way, an instance of *GMC* will capture the incoming workload to one or more applications. Consequently, each *GMC* instance will extract

Figure 4.2: The overall view of workload flows in an Enterprise Application Server (EAS). Different workload sources send their requests to the *EAS*, and the dispatcher inside the EAS distributes requests to appropriate applications. And afterwards dispatchers inside applications distribute requests to their components.

the workload patterns of all its associated applications.

To increase the granularity of workload patterns extraction, we can use the component level of workload modeling. In this approach, which is shown in Figure 4.5, there is a one-to-one association between *GMC* instances and workloads dispatching to each component inside applications. Therefore, the resulting *NDSMMs* for each instance of *GMC* contain the workload patterns of the workloads for the associated component. Also we can use the variant of this approach which lets us connect an instance of *GMC* to more than one component. Using this one-to-many variant, we are able to extract the workload patterns of a group of components together. Also

Figure 4.3: Modeling the incoming workload to EAS as a whole. In this approach the resulting *NDSMMs* contain the general behavioral model of the workloads for the *EAS*.

we can attach *GMC* instances to components from different applications inside the *EAS*.

4.2.2 Advanced Integration Approaches

To extract more specific workload patterns, we need to use lower levels of workload classification. This thesis report proposes context-oriented workload modeling as an effective solution to capture more specific patterns of workloads.

Along with each request sent to an *EAS*, some implicit information is transferred which is called the request's context. A request's context can be used by the target

Figure 4.4: Modeling the incoming workload to each application inside the *EAS* separately. In this approach the resulting *NDSMMs* contain the workload patterns of the workloads for each application.

component or it can be easily ignored. The main idea of context-oriented workload modeling is to classify the workload according to requests' contexts. For instance, the requests' contexts may contain the requests senders' source addresses (i.e. IP addresses). Then we can classify the incoming workload according to the source addresses and use separate *GMC* instances to model workloads coming from specific addresses. Another example is the requests' contexts which contain security related information such as user names or user roles. So, the workload classification in this case can be carried out according to security related information available in the contexts of requests.

Figure 4.5: Modeling the incoming workload to each component inside the *EAS* applications separately. In this approach the resulting *NDSMMs* contain the behavioral patterns of the workloads for applications' components.

As a matter of fact, the context-oriented approach can be used in combination with general integration solutions proposed in the last section. The combinatorial approach is useful for capturing very specific patterns of workloads coming into an *EAS*.

4.2.3 Implementation Issues

The *GMC* uses the dynamic building process, mentioned in Section 3.2.1 of this report, to build *NDSMMs*. So, it is only capable of processing a sequence of 3-tuples $(I_0, A_0, E_0) \rightarrow (I_1, A_1, E_1) \rightarrow \dots \rightarrow (I_n, A_n, E_n)$ where I_i , $i \in [0, n]$ shows the idle

time before processing the next request from a workload source, A_i , $i \in [0, n]$ is the next request which a workload source generates after spending the idle time, and E_i , $i \in [0, n]$ is the time spent to process the request A_i , $i \in [0, n]$. Therefore, we need to implement a converter module which is responsible for generating this type of sequence from a given workload stream. However, the implementation of this converter absolutely depends on the target *EAS*. In other words, we need to implement this converter for each different *EAS* which we are going to integrate the *GMC* into. This issue should be addressed for different *EASs* separately, because different *EASs* offer different workload monitoring interfaces.

4.3 Conclusion

In this chapter, a Generic Modeling Component (GMC) is proposed to facilitate the application of *NDSMMs* for workload modeling of Enterprise Application Servers (EAS). It is shown how the *GMC* uses *NDSMMs* to model the workload of a given *EAS* and stores the extracted models in a well-formatted repository. Afterwards, different integration approaches are investigated to show how to integrate *GMC* instances into an *EAS*. Chapter 5 illustrates experiments and evaluations carried out on *NDSMM*-based workload modeling. It investigates the efficiency and applicability of the proposed workload modeling approach for real-world applications.

Chapter 5

Experiments and Evaluations

We conduct our experimentations using synthetic workload models defined by RUBiS [10] to evaluate the accuracy of the proposed approach for workload modeling using *NDSMMs*. These workload models are designed according to standards introduced in TPC-W [24] which model an online bookstore (see clauses 5.3.1.1. and 6.2.1.2 of the TPC-W v1.6 specification [13]). For this purpose, we use JOnAS J2EE [2] as our enterprise application server and integrate an incoming request monitor into it as is shown in Figure 5.1. We use a machine equipped with PIV 2.80 GHz CPU, 1 GB of RAM. Also we use MSSQL Server 2000 as the backend DBMS running on MS Windows Server 2003 Service Pack 1. In this architecture, we use RUBiS client module to simulate our workload generators. The RUBiS client module generates workloads with high randomness which simulates the most chaotic situation. Therefore, the results of our experimentations show the accuracy of *NDSMMs* for the worst case scenarios.

As Algorithm 2 is only capable of processing a sequence of 3-tuples $(I_0, A_0, E_0) \rightarrow \dots \rightarrow (I_i, A_i, E_i) \rightarrow \dots \rightarrow (I_n, A_n, E_n)$ (where I_i shows the idle time before processing the next request, A_i is the next request which a workload source generates after

Figure 5.1: Integration of an instance of *GMC* into JOnAS J2EE which hosts RUBiS application

spending the idle time, and E_i is the time spent to process the request A_i). Therefore, we implement a converter module which is responsible for generating this sequence from a monitored workload coming toward the JOnAS application server.

5.1 Evaluation Results

To evaluate the accuracy of our approach which is based on *NDSMM*, we use the sample data tracing approach explained in our *NDSMMs* comparison operator. Therefore, at first we build a *NDSMM* for workload generated by RUBiS clients according to a specific workload model. Afterward, we generate another sequence of workload based on the same workload model and trace it on the built *NDSMM*. The tracing process will generate a sequence of 3-tuples $(P_I^0, P_A^0, P_E^0) \rightarrow \dots \rightarrow (P_I^i, P_A^i, P_E^i) \rightarrow \dots \rightarrow (P_I^n, P_A^n, P_E^n)$. Then the average difference between the

Table 5.1: Similarity of built NDSMMs and real workloads.

Workload	MAX_S = 10	MAX_S = 50	MAX_S = 100	MAX_S = 500
W0	0.36	0.45	0.45	0.49
W1	0.37	0.41	0.45	0.49
W2	0.37	0.44	0.45	0.49
W3	0.34	0.43	0.46	0.50
W4	0.36	0.44	0.46	0.49
W5	0.34	0.42	0.47	0.48
W6	0.36	0.41	0.45	0.49
W7	0.37	0.43	0.46	0.49
W8	0.37	0.42	0.42	0.49
W9	0.35	0.45	0.45	0.49

NDSMM and real workload model can be measured as follows:

$$difference = \frac{\sum_{i=0}^n \frac{(1-P_I^i) + (1-P_A^i) + (1-P_E^i)}{3}}{n}$$

And as the similarity between the *NDSMM* and the real workload equals to the complement of the average difference, we have:

$$similarity \leftarrow 1 - difference$$

Table.1 shows the resulted similarity of built *NDSMM*s for each generated workload i.e. W_i . It also illustrates the relationship between size of *NDSMM*s and their measured similarity for different workloads. Interestingly, increasing the size of a *NDSMM* increases the similarity only for a while. In other words, the accuracy of a *NDSMM* becomes stable after a specific size and does not increase anymore. These results show that *NDSMM*-based workload modeling can reach an accuracy of approximately 50% in the most chaotic situation for which the incoming workload has high randomness.

Table 5.2: Required workload repetitions to achieve stabilized *NDSMMs*

Repetition times	MAX_S = 10	MAX_S = 50	MAX_S = 100	MAX_S = 500
W0:05	0.28	0.27	0.27	0.35
W0:10	0.35	0.39	0.43	0.45
W0:15	0.35	0.40	0.45	0.49
W0:20	0.34	0.45	0.44	0.48
W0:25	0.36	0.42	0.42	0.49
W0:30	0.34	0.45	0.45	0.49
W1:05	0.25	0.27	0.31	0.37
W1:10	0.36	0.40	0.42	0.47
W1:15	0.35	0.39	0.45	0.49
W1:20	0.35	0.41	0.44	0.48
W1:25	0.36	0.42	0.44	0.48
W1:30	0.34	0.41	0.43	0.49

To evaluate the required learning time for different workload models, we generate workloads based on same workload models for repeated times and measure the similarity of built *NDSMMs* with real workload after each repetition. The result of this experiment for two workload models is shown in Table. 2. With respect to acquired results, it can be seen that the accuracy of *NDSMMs* in most of test cases gets stabilized after ten repetitions.

5.2 Conclusion

In this chapter, we illustrated experiments and carried out evaluations on NDSMM-based workload modeling. Our investigations proved the applicability and efficiency of the proposed workload modeling solution for real-world applications. Chapter 6 concludes the most important contributions of this thesis report and goes through all its achievements. Also it discusses possibilities to extend this work.

Chapter 6

Conclusion and Prospects

This chapter describes the most important contributions of this thesis and reviews all its achievements. Subsequently, different possibilities to extend this work are presented.

6.1 Contributions

In this thesis a novel Markovian approach is introduced to model the incoming workload toward an Enterprise Application Server (EAS). This Markovian approach is based on a Novel Dynamic Semi-Markov Model (*NDSMM*), proposed in this thesis, which is an intuitive combination of a Semi-Markov Model[39] and a Dynamic Markov Model [12]. *NDSMM* is designed with this intention to be an efficient workload modeling tool for enterprise application servers. However, a formal definition of the *NDSMM* independent of its proposed application in this thesis (i.e. workload modeling for enterprise application servers) is given followed by a general example. This formal definition allows academic researchers to investigate other possible applications of *NDSMM*.

After studying the formal definition of a *NDSMM*, a step by step procedure of building a *NDSMM* for a typical system is illustrated. The brief explanation of the design and implementation of this procedure ensures that sufficient information is provided for real-world applications. Also to make the *NDSMM* useful for a wide range of applications, adjustment techniques are proposed. Subsequently, a comparison operator is introduced along with its detailed design and implementation to determine how close two *NDSMMs* are. This thesis uses this operator to keep the redundancy as low as possible when *NDSMMs* are being used for workload modeling. However, as this operator is defined according to the formal definition of a *NDSMM*, it can be used in other applications which use *NDSMMs* for other purposes.

Subsequently, the application of *NDSMMs* for workload modeling of enterprise application servers is discussed. It includes how to design a Generic Modeling Component (GMC) and integrate it into enterprise application servers. *GMC* is proposed to facilitate the application of *NDSMMs* for modeling the workload of an *EAS*. It is shown how the *GMC* uses *NDSMMs* to model the workload of a given *EAS* and stores the extracted models in a well-formatted repository. GraphXML [30] is suggested for the format of storing *NDSMMs* in the repository and it is shown how to use it to describe a *NDSMM*. Afterwards, different integration approaches, including general and advanced solutions, are investigated to show how to integrate *GMC* instances into an *EAS*.

To prove the efficiency and applicability of the proposed approach for real-world applications, several experiments and evaluations are carried out. The results of these experiments show that *NDSMM*-based workload modeling for enterprise application servers is an efficient and accurate solution. In conclusion, the goals of this thesis are

achieved.

6.2 Future Work

The focus of future work on *NDSMM*'s will be to improve the accuracy of the process which extracts workload models. In other words, the intention will be to capture finer characteristics of processing workloads and to reduce the RMSE [69] of each workload model as much as possible. Combining *NDSMM*'s with other machine learning [47] methods such as Bayesian networks [32] will be an effective way to achieve this goal. Also considering the integration of the automatic modeling process with a workload information repository can accelerate the process of workload modeling significantly. This repository can contain information about the processing workloads in a descriptive language. Using this information repository, workload modeling processes will be able to start with more accurate starting models.

Also, the general definition of *NDSMMs* let us investigate other possible applications for it. For example, *NDSMMs* can be used to capture the behavioral model of moving objects in different environments. This is useful for the creation of more realistic animations or to predict the behavior of players in video games. Another application of *NDSMMs* can be capturing the surfing behaviors of website users. A simple usage of this application is to figure out the appropriate placement of advertisements in different pages of a website.

Bibliography

- [1] *Extensible Markup Language (XML) 1.0 Specifications: from the W3C recommendations*. iUniverse, Incorporated, 2000.
- [2] Takoua Abdellatif. Enhancing the management of a J2EE application server using a component-based architecture. In *EUROMICRO '05: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 70–79, Washington, DC, USA, 2005. IEEE Computer Society.
- [3] Subrahmanyam Allamaraju. *Professional Java Server Programming J2EE Edition*. Peer Information Inc., 2000.
- [4] R. Begleiter, R. El-Yaniv, and G. Yona. On prediction using variable order Markov models. *Journal of Artificial Intelligence Research*, 22:385–421, 2004.
- [5] Philip A. Bernstein. Transaction processing monitors. *Commun. ACM*, 33(11):75–86, 1990.
- [6] Robert Bodnarchuk and Richard Bunt. A synthetic workload model for a distributed system file server. In *SIGMETRICS '91: Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 50–59, New York, NY, USA, 1991. ACM.

- [7] Kevin Boone. *Applied Enterprise JavaBeans Technology*. Prentice Hall PTR, 2003.
- [8] Alan W. Brown. *Large-Scale, Component Based Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2000.
- [9] Maria Calzarossa and Giuseppe Serazzi. Workload characterization: A survey. *Proc. IEEE*, 81(8):1136–1150, 1993.
- [10] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. Performance and scalability of EJB applications. In *OOPSLA '02: Proceedings of the 17th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pages 246–261, New York, NY, USA, 2002. ACM Press.
- [11] Alan Colman, Linh Duy Pham, Jun Han, and Jean-Guy Schneider. Adaptive application-specific middleware. In *MW4SOC '06: Proceedings of the 1st Workshop on Middleware for Service Oriented Computing (MW4SOC 2006)*, pages 6–11, New York, NY, USA, 2006. ACM.
- [12] G. V. Cormack and R. N. S. Horspool. Data compression using dynamic Markov modelling. *The Computer Journal*, 30(6):541–550, 1987.
- [13] Transaction Processing Performance Council. *TPC Benchmark W, Standard Specification*. 2001.
- [14] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.
- [15] Devanshu Dhyani, Sourav S. Bhowmick, and Wee-Keong Ng. Modelling and predicting web page accesses using Markov processes. In *DEXA '03: Proceedings of*

the 14th International Workshop on Database and Expert Systems Applications, page 332, Washington, DC, USA, 2003. IEEE Computer Society.

- [16] Darpan Dinker and Herb Schwetman. Using Java method traces to automatically characterize and model J2EE server applications. In *WSC '06: Proceedings of the 38th Conference on Winter Simulation*, pages 1187–1194. Winter Simulation Conference, 2006.
- [17] Magdalini Eirinaki, Michalis Vazirgiannis, and Dimitris Kapogiannis. Web path recommendations based on page ranking and Markov models. In *WIDM '05: Proceedings of the 7th Annual ACM International Workshop on Web Information and Data Management*, pages 2–9, New York, NY, USA, 2005. ACM.
- [18] Andrej Bratko et al. Spam filtering using statistical data compression models. *J. Mach. Learn. Res.*, 7:2673–2698, 2006.
- [19] J. Li F. Khalil and H. Wang. Integrating Markov model with clustering for predicting web page accesses. In *AusWeb'07: Proceedings of the 13th Australian World Wide Web conference*, Australia, 2007.
- [20] Dror G. Feitelson. Workload modeling for performance evaluation. In *Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002, Tutorial Lectures*, pages 114–141, London, UK, 2002. Springer-Verlag.
- [21] Domenico Ferrari. On the foundations of artificial workload design. In *SIGMETRICS '84: Proceedings of the 1984 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 8–14, New York, NY, USA, 1984. ACM.

- [22] Laura Firoiu and Paul R. Cohen. Segmenting time series with a hybrid neural networks - hidden Markov model. In *Eighteenth National Conference on Artificial Intelligence*, pages 247–252, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.
- [23] A. Galata, N. Johnson, and D. Hogg. Learning behaviour models of human activities. In *Proc. British Machine Vision Conference (BMVC'99)*, pages 12–22, 1999.
- [24] Daniel F. Garcia and Javier Garcia. TPC-W e-commerce benchmark evaluation. *Computer*, 36(2):42–48, 2003.
- [25] Charles M. Grinstead and J. Laurie Snell. *Introduction to Probability*. American Mathematical Society, 1997.
- [26] John Grundy, Steve Newby, Thomas Whitmore, and Peter Grundeman. Extending a persistent object framework to enhance enterprise application server performance. *Aust. Comput. Sci. Commun.*, 24(2):57–64, 2002.
- [27] J. Jeffrey Hanson. *Pro JMX Java Management Extensions*. Apress, 2003.
- [28] Franz J. Hauck, Rüdiger Kapitza, Hans P. Reiser, and Andreas I. Schmied. A flexible and extensible object middleware: CORBA and beyond. In *SEM '05: Proceedings of the 5th International Workshop on Software Engineering and Middleware*, pages 69–75, New York, NY, USA, 2005. ACM.
- [29] George T. Heineman and William T. Councill, editors. *Component-based Software Engineering: Putting the Pieces Together*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

- [30] Ivan Herman and M. Scott Marshall. GraphXML - an XML-based graph description format. In *GD '00: Proceedings of the 8th International Symposium on Graph Drawing*, pages 52–62, London, UK, 2001. Springer-Verlag.
- [31] R.J. Honicky, Swami Ramany, and Darren Sawyer. Workload modeling of stateful protocols using HMMs. In *proceedings of the 31st Annual International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems*, Orlando, Florida, 2005.
- [32] Finn V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
- [33] Michael Kearns, Yishay Mansour, Dana Ron, Ronitt Rubinfeld, Robert E. Schapire, and Linda Sellie. On the learnability of discrete distributions. In *STOC '94: Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, pages 273–282, New York, NY, USA, 1994. ACM.
- [34] Daniel L. Kiskis and Kang G. Shin. SWSL: A synthetic workload specification language for real-time systems. *Software Engineering*, 20(10):798–811, 1994.
- [35] Sacha Krakowiak. *Middleware Architecture with Patterns and Frameworks*. <http://sardes.inrialpes.fr/~krakowia/MW-Book/main.pdf>, 2007.
- [36] Y. Daniel Liang. *Introduction to Java Programming-Comprehensive Version*. Prentice Hall, 2006.
- [37] Mark Little, Jon Maron, and Greg Pavlik. *Java Transaction Processing: Design and Implementation*. Prentice Hall PTR, 2004.

- [38] Yi Liu. Tutorial on the enterprise javabeans (ejb) component model. *J. Comput. Small Coll.*, 18(6):110–111, 2003.
- [39] Gabriel G. Infante López, Holger Hermanns, and Joost-Pieter Katoen. Beyond memoryless distributions: Model checking semi-Markov chains. In *PAPM-PROBMIV '01: Proceedings of the Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification*, pages 57–70, London, UK, 2001. Springer-Verlag.
- [40] R. Marculescu, D. Marculescu, and M. Pedram. Vector compaction using dynamic Markov models, 1996.
- [41] Marcello Mariucci. Enterprise application server development environments. Technical report, Institute of Parallel and Distributed High Performance Systems, 2000.
- [42] A. Markov. Extension of the limit theorems of probability theory to a sum of variables connected in a chain. *Dynamic Probabilistic Systems, Vol. I: Markov Models*, 1(1), 1971.
- [43] M. S Marshall, Ivan Herman, and Guy Melancon. An object-oriented design for graph visualization. Technical report, Centre for Mathematics and Computer Science, Amsterdam, The Netherlands, 2000.
- [44] Richard Monson-Haefel and David Chappell. *Java Message Service*. O'Reilly Media, Inc., 2000.

- [45] Mohamed Mosbah and Nasser Saheb. A syntactic approach to random walks on graphs. In *WG '97: Proceedings of the 23rd International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 258–272, London, UK, 1997. Springer-Verlag.
- [46] Martin Nally. Application servers: virtualizing location, resources, memory, users and threads for business applications and web applications. In *VEE '05: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, pages 110–110, New York, NY, USA, 2005. ACM.
- [47] G. F. Page. Introduction to Machine Learning. *Robotica*, 24(1):143–143, 2006.
- [48] Alex Pentland and Andrew Liu. Modeling and prediction of human behavior. *Neural Comput.*, 11(1):229–242, 1999.
- [49] Marco Pistoia and Nataraj Nagaratnam. *Enterprise Java(TM) Security: Building Secure J2EE(TM) Applications*. Addison-Wesley Professional, 2004.
- [50] Marek Prochazka, Romain Rouvoy, and Thierry Coupaye. On enhancing component-based middleware with transactions. In Robert Meersman and Zahir Tari, editors, *OTM Workshops*, volume 2889 of *Lecture Notes in Computer Science*, pages 1–2. Springer, 2003.
- [51] George Reese. *Database Programming with JDBC and Java*. O'Reilly Media, Inc., 2000.
- [52] Andreas Reuter. Load control and load balancing in a shared database management system. In *Proceedings of the Second International Conference on Data*

- Engineering*, pages 188–197, Washington, DC, USA, 1986. IEEE Computer Society.
- [53] J. Rolia, L. Cherkasova, and R. Friedrich. Performance engineering for enterprise software systems in next generation data centres. Technical report, HP Laboratories, 2006.
 - [54] Michael S. Salkin, Richard E. Just, and O. A. Cleveland Jr. Estimation of nonstationary transition probabilities for agricultural firm size projection. *The Annals of Regional Science*, 10(1):71–82, 1976.
 - [55] P G Sarang, Kyle Gabhart, Dave Young, Andre Tost, Tim McAllister, Rahim Adatia, Keyur Shah, and John Griffin. *Professional Ejb*. Wrox Press Ltd., Birmingham, UK, UK, 2001.
 - [56] Ramesh R. Sarukkai. Link prediction and path analysis using Markov chains. *Comput. Networks*, 33(1-6):377–386, 2000.
 - [57] Nima Sharifimehr and Samira Sadaoui. A predictive automatic tuning service for object pooling based on dynamic Markov modeling. In *Proc. of 2nd International Conference on Software and Data Technologies*, Barcelona, Spain, 2007.
 - [58] Alan C. Shaw. *Real-Time Systems and Software*. John Wiley & Sons, Inc., New York, NY, USA, 2000.
 - [59] Sylvain Sicard, Noel De Palma, and Daniel Hagimont. J2EE server scalability through EJB replication. In *SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 778–785, New York, NY, USA, 2006. ACM.

- [60] B. Song, C. Ernemann, and R. Yahyapour. Parallel computer workload modeling with Markov chains. In *Job Scheduling Strategies for Parallel Processing*, pages 47–62, London, UK, 2004. Springer Verlag LNCS.
- [61] Robert Spence. *Information Visualization: Design for Interaction (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2007.
- [62] Nikolay Stefanov, Aphrodite Galata, and Roger Hubbard. Real-time hand tracking with variable-length Markov models of behaviour. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, page 73, Washington, DC, USA, 2005. IEEE Computer Society.
- [63] Marcelo Tenório, Silvia Nassar, Paulo Freitas, and Carlos Magno. Recognition of continuous probability models. In *WSC '05: Proceedings of the 37th Conference on Winter Simulation*, pages 2524–2531. Winter Simulation Conference, 2005.
- [64] Wiebe van der Hoek, Wojciech Jamroga, and Michael Wooldridge. A logic for strategic reasoning. In *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 157–164, New York, NY, USA, 2005. ACM Press.
- [65] Farrokh Vatan. Distribution functions of probabilistic automata. In *STOC '01: Proceedings of the Thirty-third Annual ACM Symposium on Theory of Computing*, pages 684–693, New York, NY, USA, 2001. ACM.
- [66] Michael D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Softw. Eng.*, 17(9):972–975, 1991.

- [67] Harold B. Westlund and Gary W. Meyer. A BRDF database employing the Beard-Maxwell reflection model. In *Graphics Interface*, pages 189–201, 2002.
- [68] Matthew Young-Lai. Adding state merging to the DMC data compression algorithm. *Inf. Process. Lett.*, 70(5):223–228, 1999.
- [69] Arnold Zellner. *Statistics, Econometrics and Forecasting*. Cambridge University Press, 2004.
- [70] Jia Zhang, Jen-Yao Chung, and Carl K. Chang. Towards increasing web application productivity. In *SAC '04: Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 1677–1681, New York, NY, USA, 2004. ACM.

Appendix A

The following list shows how we use GraphXML to describe the sample *NDSMM* shown in Figure 3.1:

```
<?xml version="1.0"?>
<!DOCTYPE GraphXML SYSTEM "file:GraphXML.dtd">
<GraphXML>
  <graph>
    <node name="S0">
      <label>

$$P_t^5 = PDF_S(S_0, t)$$

      </label>
      <data name=IDLE>

$$\$IDLE[S_0][t_{min}], \dots, \$IDLE[S_0][t_{max}]$$

      </data>
      <data name=COUNT>

$$\$COUNT[S_0][0], \$COUNT[S_0][1]$$

      </data>
```

```

</node>
<node name="S1">
  <label>
     $P_t^6 = PDF_S(S_1, t)$ 
  </label>
  <data name=IDLE>
    $IDLE[S1][tmin],..., $IDLE[S1][tmax]
  </data>
  <data name=COUNT>
    $COUNT[S1][0], $COUNT[S1][1]
  </data>
</node>
<edge source="S0" target="S0">
  <label>
     $P(S_0, 0) = (S_0, 0.5), P_t^2 = PDF_T(S_0, 0, t)$ 
  </label>
  <data name=SPEED>
    $SPEED[S0][0][tmin],..., $SPEED[S0][0][tmax]
  </data>
</edge>
<edge source="S0" target="S1">
  <label>
     $P(S_0, 1) = (S_0, 0.5), P_t^1 = PDF_T(S_0, 1, t)$ 
  </label>

```

```

    <data name=SPEED>
        $SPEED[$S_0][1][$t_{min}],..., $SPEED[$S_0][1][$t_{max}]
    </data>
</edge>
<edge source="$S_1" target="$S_1">
    <label>
         $P(S_1, 1) = (S_1, 0.25), P_t^3 = PDF_T(S_1, 1, t)$ 
    </label>
    <data name=SPEED>
        $SPEED[$S_1][1][$t_{min}],..., $SPEED[$S_0][1][$t_{max}]
    </data>
</edge>
<edge source="$S_1" target="$S_0">
    <label>
         $P(S_1, 0) = (S_0, 0.75), P_t^4 = PDF_T(S_1, 0, t)$ 
    </label>
    <data name=SPEED>
        $SPEED[$S_1][0][$t_{min}],..., $SPEED[$S_1][0][$t_{max}]
    </data>
</edge>
</graph>
</GraphXML>

```