

**LOAD BALANCING:
AN AUTOMATED LEARNING APPROACH**

BY

PANKAJ MEHRA

Recom Technologies/NASA Ames Research Center
Moffett Field, CA 94035, USA

BENJAMIN W. WAH

University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA

World Scientific Publishing Co. Pte. Ltd., 1995

PREFACE

With the advent of cluster computing, there is growing need for tools and techniques that will transparently and intelligently harness distributed resources. Commercial software packages such as LSF and a plethora of public-domain software from Condor to NQS have made remote execution convenient, but the policies governing remote execution are buried deep inside the software manuals and involve parameters too numerous to set manually. In this world of heterogeneous systems and rapidly evolving configurations, factory setting of parameters is not possible. Therefore, the dream of intelligent resource management can only be realized through the development of self-tuning software.

This book examines automated learning of dynamic load-balancing strategies, addressing issues ranging from measurement and generation of computer workloads to automated refinement of load indices and load-balancing policies via machine learning. It describes SMALL (Systematic Method for Automated Learning of Load-balancing strategies), an integrated learning system that can be used for efficiently and automatically tuning commercial-grade load-balancing software.

Dynamic workload generation. We present DWG (Dynamic Workload Generator), a program for generating realistic and reproducible background workloads with high accuracy and high resolution. DWG's most innovative feature is its ability to mimic the behavior of a background workload in the presence of a foreground job. The critical functions for measurement and generation of resource-utilization levels are implemented in each site's operating-system kernel, ensuring low overhead. Actual resource-usage patterns of a distributed system can be captured and replayed with high fidelity, test jobs introduced at precise instants, and their completion time measured accurately. Such experiments can be repeated, running the job at a different site each time but under the same background workload. Since a policy's performance is completely determined by where it schedules an incoming job, alternative policies can be compared *under identical loading conditions*. Thus, DWG allows us to perform reproducible load-balancing experiments, a facility hitherto unavailable to experimenters in this area.

Learning consistent and comparable performance indices. DWG tracks the utilization levels of each site's key resources: computational, primary memory, secondary storage, and communication. It provides a precise account of the loading conditions prevalent just before a job begins execution. DWG also measures the completion time of that job when it finishes execution. Such "before" and "after" data suffice for learning to compare alternative destinations for incoming tasks. The problem is one of *learning to compare functions of multivariate time series*.

We have adopted the neural-network architecture used for learning evaluation functions in Tesauro's renowned backgammon-playing program [160, 161], and added various smoothing and extrapolation capabilities to his method of learning to compare multivariate functions. We also present an innovative learning algorithm that obviates the 'linking of weights' [93] required by Tesauro's original architecture. This modification allows us to use off-the-shelf neural-network-simulation packages [52]. Extensive statistical tests on the load-index functions learned using our *comparator neural network*

reveal high positive correlations (at 99% level of significance) between the true ranking of sites and the one induced by the new index functions. Thus, if load indices were computed right before each decision point, and if they could be communicated instantly across sites, then we could (with high confidence) select the destination having the the least completion time for each incoming job.

The comparator neural network transforms the multi-dimensional and highly dynamic measurements provided by DWG into smooth one-dimensional load indices that can be efficiently communicated over the network, and compared across sites in a meaningful fashion, unlike the traditional load average.

Automated tuning of policy parameters Wah, *et al.* have developed a domain-independent population-based learning system, called TEACHER (TEchniques for Automated Creation of HEuRistics), which also accommodates point-based learning [165]. TEACHER rationally schedules limited learning time between *generating* new parameter sets and *testing* the promising ones in the current population. We have integrated into TEACHER a point-based learning procedure for adjusting certain parameters of load-balancing policies.

We find that the optimal setting of parameters is sensitive to the overheads involved in migrating tasks and communicating load indices from site to site. Further, we report experiments with different intervals of computation for the load-index function; these affect the average age of load-index values. Since the quality of a load index degrades with age, the setting of policy parameters is also sensitive to the interval of load-index computation. In all these cases, the integrated (population- and point-based) learning system described above is able to quickly determine the appropriate parameter set with high confidence, given data about the completion times of test jobs and information about various overheads and delays.

Thus, what used to be an unsystematic, manual, and tedious process of discovering good parameter settings by trial and error has been replaced by a systematic, automated, and efficient process of population-based learning. Likewise, what used to be the ad hoc process of adjusting the thresholds of policies based on human experience has now been replaced by an automated performance-driven process of point-based learning.

By carrying out an in-depth investigation into the use of artificial intelligence for developing self-tuning load-balancing software, we have conclusively demonstrated not only that we can develop good load-balancing policies but also that we can do so systematically, automatically and efficiently. We believe that it is only a matter of time before machine learning techniques will be used for improving other similar computer-system functions, such as branch prediction in RISC microprocessors, data prefetching in NUMA multiprocessors and file placement in distributed information servers.

Acknowledgements

We are grateful for the support of National Aeronautics and Space Administration Grants NCC 2-481 and NAG 1-613, National Science Foundation grant MIP 92-18715, and a gift from Sumitomo Electric Industries in the preparation of this book.

TABLE OF CONTENTS

Chapter	
Preface	ix
Acknowledgements	x
1. INTRODUCTION	1
1.1. Load Balancing in Distributed Computer Systems	1
1.2. Our Load-Balancing Policy	2
1.3. Automated Learning of Load-Balancing Strategies	3
1.4. SMALL: A System for Learning Load-Balancing Strategies	6
1.5. Book Outline	17
2. SYNTHETIC WORKLOAD GENERATION	18
2.1. Introduction	18
2.2. Previous Work	20
2.3. Architecture of DWG: A Dynamic Workload Generator	22
2.4. Operation of DWG	26
2.5. Evaluation, Parameterization and Tuning of Generation Rules	30
2.6. Summary	40
3. AUTOMATED LEARNING OF LOAD BALANCING STRATEGIES	41
3.1. Previous Work	42
3.2. Experimentation Environment and Workload Collection	46
3.3. Automated Learning of Workload Indices	47
3.4. Automated Tuning of Load-Balancing Policies	60
3.5. Data Preparation	65
3.6. Empirical Results	67
3.7. Summary	70
4. CONCLUSIONS	71
APPENDIX A. A SURVEY OF STRATEGY LEARNING	74
A.1. Introduction	74
A.2. The Anatomy of Strategy-Learning Problems	77
A.3. Methods of Strategy Learning	100
A.4. Architectures for Strategy Learning	117
A.5. Summary	123
REFERENCES	126
INDEX	137

LIST OF TABLES

Table	
2.1	Benchmark programs used in this book 32
2.2	Parameter sets for doctoring rules 33
2.3	Signed and unsigned errors of different parameter sets 34
3.1	Typical performance-indicating variables available in UNIX-like operating systems 43
3.2	Previous Research on Intelligent Load Balancing 45
3.3	Significance test for coefficient of rank-correlation (I) 54
3.4	Significance test for coefficient of rank-correlation (II) 56
3.5	Performance of policies learned under different assumptions 68
4.1	Times required by different phases of SMALL's operation 72
A.1	Components of strategy-learning problems 76
A.2	Components of a performance task 80
A.3	Examples of performance tasks in strategy learning 88
A.4	Examples of problem solvers in strategy-learning tasks 93
A.5	Examples of environments for strategy learning 99
A.6	Issues and approaches of strategy learning 101
A.7	Examples of issues in strategy learning 102
A.8	Architectural models characterized by their approach 122
A.9	Examples of strategy-learning problems and architectures 124

LIST OF FIGURES

Figure	
1.1 Parametrized load-balancing policy studied in this book	2
1.2 Comparison between experimentation environments	6
1.3 Overview of SMALL	8
1.4 The five phases of SMALL 's operation	9
1.5 Operation of SMALL : Collection of background-load patterns	11
1.6 Operation of SMALL : Measurement of completion times	11
1.7 Operation of SMALL : Load-index learning	13
1.8 Operation of SMALL : Policy learning	14
1.9 Operation of SMALL : Application phase	16
2.1 Four ways of workload generation	21
2.2 Architecture of DWG	23
2.3 Format of DWG log files	25
2.4 The need for <i>dynamic doctoring</i> of generated load	28
2.5 Measuring the accuracy of generated loads	31
2.6 Contour plots of cumulative probability of error with nine different parameter sets	36
2.7 Contour plot of cumulative probability of error for the selected parameter set (7)	37
2.8 Comparison of true and generated resource-utilization patterns (Expt. #15)	38
2.9 Comparison of true and generated resource-utilization patterns (Expt. #17)	39
3.1 An episode in training a comparator network	50
3.2 Trend extraction via curve fitting	51
3.3 Average speedup of all test-jobs under the worst-behaved load (#23)	57
3.4 Performance of all test-jobs under load #23; policies: <i>opt</i> , <i>min</i> , <i>rnd</i> , and <i>max</i>	59
3.5 Performance of all test-jobs under load #11; policies: <i>opt</i> , <i>min</i> , <i>rnd</i> , and <i>max</i>	60
3.6 Contour plot of cumulative probability of speed-up	60
3.7 Raw resource-utilization patterns and their load index	61
3.8 Representation of one set of load-balancing policies for N sites	61
3.9 An example of mutation applied to an HM	64
3.10 An illustration of the crossover-1 operation	64
3.11 Rules for point-based learning	66
3.12 Overheads and delays in load balancing	66
3.13 Contour plot of cumulative probability of speed-up	69
A.1 Classification of strategy-learning problems	76
A.2 Typical information flow in a strategy-learning system	77
A.3 Attributes for classifying performance tasks	80

A.4	Components of problem solvers and their properties	88
A.5	Components of a learning environment	93
A.6	Knowledge-based model of strategy-learning systems	118
A.7	Reinforcement-learning model of learning systems	118
A.8	The hybrid point-based model of learning systems	118
A.9	Population-based model of learning systems	119